CS 229, Autumn 2016
Problem Set #0: Linear Algebra and Multivariable Calculus

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) This specific homework is *not graded*, but we encourage you to solve each of the problems to brush up on your linear algebra. Some of them may even be useful for subsequent problem sets. It also serves as your introduction to using Gradescope for submissions.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[0 points] Gradients and Hessians**

   Recall that a matrix $A \in \mathbb{R}^{n \times n}$ is *symmetric* if $A^T = A$, that is, $A_{ij} = A_{ji}$ for all i, j. Also recall the gradient $\nabla f(x)$ of a function $f : \mathbb{R}^n \to \mathbb{R}$, which is the n-vector of partial derivatives

   $$\nabla f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} \quad \text{where} \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

   The hessian $\nabla^2 f(x)$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the $n \times n$ symmetric matrix of twice partial derivatives,

   $$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(x) & \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \frac{\partial^2}{\partial x_2 \partial x_1} f(x) & \frac{\partial^2}{\partial x_2^2} f(x) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} f(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \frac{\partial^2}{\partial x_n \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_n^2} f(x) \end{bmatrix}.$$

   (a) Let $f(x) = \frac{1}{2} x^T A x + b^T x$, where A is a symmetric matrix and $b \in \mathbb{R}^n$ is a vector. What is $\nabla f(x)$?

   (b) Let $f(x) = g(h(x))$, where $g : \mathbb{R} \to \mathbb{R}$ is differentiable and $h : \mathbb{R}^n \to \mathbb{R}$ is differentiable. What is $\nabla f(x)$?

   (c) Let $f(x) = \frac{1}{2} x^T A x + b^T x$, where A is symmetric and $b \in \mathbb{R}^n$ is a vector. What is $\nabla^2 f(x)$?

   (d) Let $f(x) = g(a^T x)$, where $g : \mathbb{R} \to \mathbb{R}$ is continuously differentiable and $a \in \mathbb{R}^n$ is a vector. What are $\nabla f(x)$ and $\nabla^2 f(x)$? (*Hint:* your expression for $\nabla^2 f(x)$ may have as few as 11 symbols, including $'$ and parentheses.)

2. **[0 points] Positive definite matrices**

   A matrix $A \in \mathbb{R}^{n \times n}$ is *positive semi-definite* (PSD), denoted $A \succeq 0$, if $A = A^T$ and $x^T A x \geq 0$ for all $x \in \mathbb{R}^n$. A matrix A is *positive definite*, denoted $A \succ 0$, if $A = A^T$ and $x^T A x > 0$ for

all $x \neq 0$, that is, all non-zero vectors x. The simplest example of a positive definite matrix is the identity I (the diagonal matrix with 1s on the diagonal and 0s elsewhere), which satisfies $x^T I x = \|x\|_2^2 = \sum_{i=1}^n x_i^2$.

(a) Let $z \in \mathbb{R}^n$ be an n-vector. Show that $A = zz^T$ is positive semidefinite.

(b) Let $z \in \mathbb{R}^n$ be a *non-zero* n-vector. Let $A = zz^T$. What is the null-space of A? What is the rank of A?

(c) Let $A \in \mathbb{R}^{n \times n}$ be positive semidefinite and $B \in \mathbb{R}^{m \times n}$ be arbitrary, where $m, n \in \mathbb{N}$. Is BAB^T PSD? If so, prove it. If not, give a counterexample with explicit A, B.

3. **[0 points] Eigenvectors, eigenvalues, and the spectral theorem**

The eigenvalues of an $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$ are the roots of the characteristic polynomial $p_A(\lambda) = \det(\lambda I - A)$, which may (in general) be complex. They are also defined as the the values $\lambda \in \mathbb{C}$ for which there exists a vector $x \in \mathbb{C}^n$ such that $Ax = \lambda x$. We call such a pair (x, λ) an *eigenvector, eigenvalue* pair. In this question, we use the notation $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ to denote the diagonal matrix with diagonal entries $\lambda_1, \ldots, \lambda_n$, that is,

$$\mathrm{diag}(\lambda_1, \ldots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_n \end{bmatrix}.$$

(a) Suppose that the matrix $A \in \mathbb{R}^{n \times n}$ is diagonalizable, that is, $A = T\Lambda T^{-1}$ for an invertible matrix $T \in \mathbb{R}^{n \times n}$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ is diagonal. Use the notation $t^{(i)}$ for the columns of T, so that $T = [t^{(1)} \ \cdots \ t^{(n)}]$, where $t^{(i)} \in \mathbb{R}^n$. Show that $At^{(i)} = \lambda_i t^{(i)}$, so that the eigenvalues/eigenvector pairs of A are $(t^{(i)}, \lambda_i)$.

A matrix $U \in \mathbb{R}^{n \times n}$ is orthogonal if $U^T U = I$. The spectral theorem, perhaps one of the most important theorems in linear algebra, states that if $A \in \mathbb{R}^{n \times n}$ is symetric, that is, $A = A^T$, then A is *diagonalizable by a real orthogonal matrix*. That is, there are a diagonal matrix $\Lambda \in \mathbb{R}^{n \times n}$ and orthogonal matrix $U \in \mathbb{R}^{n \times n}$ such that $U^T A U = \Lambda$, or, equivalently,

$$A = U \Lambda U^T.$$

Let $\lambda_i = \lambda_i(A)$ denote the ith eigenvalue of A.

(b) Let A be symmetric. Show that if $U = [u^{(1)} \ \cdots \ u^{(n)}]$ is orthogonal, where $u^{(i)} \in \mathbb{R}^n$ and $A = U\Lambda U^T$, then $u^{(i)}$ is an eigenvector of A and $Au^{(i)} = \lambda_i u^{(i)}$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

(c) Show that if A is PSD, then $\lambda_i(A) \geq 0$ for each i.

CS 229, Autumn 2016
Problem Set #0 Solutions: Linear Algebra and Multivariable Calculus

---

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) This specific homework is *not graded*, but we encourage you to solve each of the problems to brush up on your linear algebra. Some of them may even be useful for subsequent problem sets. It also serves as your introduction to using Gradescope for submissions.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[0 points] Gradients and Hessians**

   Recall that a matrix $A \in \mathbb{R}^{n \times n}$ is *symmetric* if $A^T = A$, that is, $A_{ij} = A_{ji}$ for all i, j. Also recall the gradient $\nabla f(x)$ of a function $f : \mathbb{R}^n \to \mathbb{R}$, which is the n-vector of partial derivatives

   $$\nabla f(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{bmatrix} \quad \text{where} \quad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}.$$

   The hessian $\nabla^2 f(x)$ of a function $f : \mathbb{R}^n \to \mathbb{R}$ is the $n \times n$ symmetric matrix of twice partial derivatives,

   $$\nabla^2 f(x) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} f(x) & \frac{\partial^2}{\partial x_1 \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n} f(x) \\ \frac{\partial^2}{\partial x_2 \partial x_1} f(x) & \frac{\partial^2}{\partial x_2^2} f(x) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n} f(x) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} f(x) & \frac{\partial^2}{\partial x_n \partial x_2} f(x) & \cdots & \frac{\partial^2}{\partial x_n^2} f(x) \end{bmatrix}.$$

   (a) Let $f(x) = \frac{1}{2} x^T A x + b^T x$, where A is a symmetric matrix and $b \in \mathbb{R}^n$ is a vector. What is $\nabla f(x)$?

   Answer: In short, we know that $\nabla(\frac{1}{2} x^T A x) = Ax$ for a symmetric matrix A, while $\nabla(b^T x) = b$. Then $\nabla f(x) = Ax + b$ when A is symmetric. In more detail, we have

   $$\frac{1}{2} x^T A x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j,$$

so for each $k = 1, \ldots, n$, we have

$$\begin{aligned}\frac{\partial}{\partial x_k}\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}A_{ij}x_i x_j &\overset{(i)}{=} \frac{\partial}{\partial x_k}\frac{1}{2}\sum_{i=1,i\neq k}^{n}A_{ik}x_i x_k + \frac{\partial}{\partial x_k}\frac{1}{2}\sum_{j=1,j\neq k}^{n}A_{kj}x_k x_j + \frac{\partial}{\partial x_k}\frac{1}{2}A_{kk}x_k^2 \\ &\overset{(ii)}{=} \frac{1}{2}\sum_{i=1,i\neq k}^{n}A_{ik}x_i + \frac{1}{2}\sum_{j=1,j\neq k}^{n}A_{kj}x_j + A_{kk}x_k \\ &= \sum_{i=1}^{n}A_{ki}x_i\end{aligned}$$

where step (i) follows because $\frac{\partial}{\partial x_k}A_{ij}x_i x_j = 0$ if $i \neq k$ and $j \neq k$, step (ii) by the definition of a partial derivative, and the final equality because $A_{ij} = A_{ji}$ for all pairs i, j. Thus $\nabla(\frac{1}{2}x^T Ax) = Ax$. To see that $\nabla b^T x = b$, note that

$$\frac{\partial}{\partial x_k}b^T x = \frac{\partial}{\partial x_k}\sum_{i=1}^{n}b_i x_i = \frac{\partial}{\partial x_k}b_k x_k = b_k.$$

(b) Let $f(x) = g(h(x))$, where $g : \mathbb{R} \to \mathbb{R}$ is differentiable and $h : \mathbb{R}^n \to \mathbb{R}$ is differentiable. What is $\nabla f(x)$?

Answer: In short, if g' is the derivative of g, then the chain rule gives

$$\nabla f(x) = g'(h(x))\nabla h(x).$$

Expanding this by components, we have for each $i = 1, \ldots, n$ that

$$\frac{\partial}{\partial x_i}f(x) = \frac{\partial}{\partial x_i}g(h(x)) = g'(h(x))\frac{\partial}{\partial x_i}h(x)$$

by the chain rule. Stacking each of these in a column vector, we obtain

$$\nabla f(x) = \begin{bmatrix} g'(h(x))\frac{\partial}{\partial x_1}h(x) \\ \vdots \\ g'(h(x))\frac{\partial}{\partial x_n}h(x) \end{bmatrix} = g'(h(x))\nabla h(x).$$

(c) Let $f(x) = \frac{1}{2}x^T Ax + b^T x$, where A is symmetric and $b \in \mathbb{R}^n$ is a vector. What is $\nabla^2 f(x)$?

Answer: We have $\nabla^2 f(x) = A$. To see this more formally, note that $\nabla^2(b^T x) = 0$, because the second derivatives of $b_i x_i$ are all zero. Let $A = [a^{(1)} \ \cdots \ a^{(n)}]$, where $a_i \in \mathbb{R}^n$ is an n-vector (because A is symmetric, we also have $A = [a^{(1)} \ a^{(2)} \ \cdots \ a^{(n)}]^T$). Then we use part (1a) to obtain

$$\frac{\partial}{\partial x_k}(\frac{1}{2}x^T Ax) = {a^{(k)}}^T x = \sum_{i=1}^{n}A_{ik}x_i,$$

and thus

$$\frac{\partial^2}{\partial x_k x_i}(\frac{1}{2}x^T Ax) = \frac{\partial}{\partial x_i}{a^{(k)}}^T x = A_{ik}.$$

(d) Let $f(x) = g(a^T x)$, where $g : \mathbb{R} \to \mathbb{R}$ is continuously differentiable and $a \in \mathbb{R}^n$ is a vector. What are $\nabla f(x)$ and $\nabla^2 f(x)$? (*Hint:* your expression for $\nabla^2 f(x)$ may have as few as 11 symbols, including $'$ and parentheses.)

Answer: We use the chain rule (part (1b)) to see that $\nabla f(x) = g'(a^T x)a$, because $\nabla(a^T x) = a$. Taking second derivatives, we have

$$\frac{\partial}{\partial x_i}\frac{\partial}{\partial x_j} = \frac{\partial}{\partial x_i} g'(a^T x)a_j = g''(a^T x)a_i a_j.$$

Expanding this in matrix form, we have

$$\nabla^2 f(x) = g''(a^T x)\begin{bmatrix} a_1^2 & a_1a_2 & \cdots & a_1a_n \\ a_2a_1 & a_2^2 & \cdots & a_2a_n \\ \vdots & \vdots & \ddots & \vdots \\ a_na_1 & a_na_2 & \cdots & a_n^2 \end{bmatrix} = g''(a^T x)aa^T.$$

2. **[0 points] Positive definite matrices**

A matrix $A \in \mathbb{R}^{n\times n}$ is *positive semi-definite* (PSD), denoted $A \succeq 0$, if $A = A^T$ and $x^T Ax \geq 0$ for all $x \in \mathbb{R}^n$. A matrix A is *positive definite*, denoted $A \succ 0$, if $A = A^T$ and $x^T Ax > 0$ for all $x \neq 0$, that is, all non-zero vectors x. The simplest example of a positive definite matrix is the identity I (the diagonal matrix with 1s on the diagonal and 0s elsewhere), which satisfies $x^T Ix = \|x\|_2^2 = \sum_{i=1}^n x_i^2$.

(a) Let $z \in \mathbb{R}^n$ be an n-vector. Show that $A = zz^T$ is positive semidefinite.

Answer: Take any $x \in \mathbb{R}^n$. Then $x^T Ax = x^T zz^T x = (x^T z)^2 \geq 0$.

(b) Let $z \in \mathbb{R}^n$ be a *non-zero* n-vector. Let $A = zz^T$. What is the null-space of A? What is the rank of A?

Answer: If $n = 1$, the null space of A is empty. The rank of A is always 1, as the null-space of A is the set of vectors orthogonal to z. That is, if $z^T x = 0$, then $x \in \mathsf{Null}(A)$, because $Ax = zz^T x = 0$. Thus, the null-space of A has dimension $n - 1$ and the rank of A is 1.

(c) Let $A \in \mathbb{R}^{n\times n}$ be positive semidefinite and $B \in \mathbb{R}^{m\times n}$ be arbitrary, where $m, n \in \mathbb{N}$. Is BAB^T PSD? If so, prove it. If not, give a counterexample with explicit A, B.

Answer: Yes, BAB^T is positive semidefinite. For any $x \in \mathbb{R}^m$, we may define $v = B^T x \in \mathbb{R}^n$. Then

$$x^T BAB^T x = (B^T x)^T A(B^T x) = v^T Av \geq 0,$$

where the inequality follows because $v^T Av \geq 0$ for any vector v.

3. **[0 points] Eigenvectors, eigenvalues, and the spectral theorem**

The eigenvalues of an $n \times n$ matrix $A \in \mathbb{R}^{n\times n}$ are the roots of the characteristic polynomial $p_A(\lambda) = \det(\lambda I - A)$, which may (in general) be complex. They are also defined as the the values $\lambda \in \mathbb{C}$ for which there exists a vector $x \in \mathbb{C}^n$ such that $Ax = \lambda x$. We call such a pair (x, λ) an *eigenvector, eigenvalue* pair. In this question, we use the notation $\mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ to denote the diagonal matrix with diagonal entries $\lambda_1, \ldots, \lambda_n$, that is,

$$\mathrm{diag}(\lambda_1, \ldots, \lambda_n) = \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \lambda_n \end{bmatrix}.$$

(a) Suppose that the matrix $A \in \mathbb{R}^{n\times n}$ is diagonalizable, that is, $A = T\Lambda T^{-1}$ for an invertible matrix $T \in \mathbb{R}^{n\times n}$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ is diagonal. Use the notation $t^{(i)}$ for the columns of T, so that $T = [t^{(1)} \ \cdots \ t^{(n)}]$, where $t^{(i)} \in \mathbb{R}^n$. Show that $At^{(i)} = \lambda_i t^{(i)}$, so that the eigenvalues/eigenvector pairs of A are $(t^{(i)}, \lambda_i)$.

Answer: The matrix T is invertible, so if we let $t^{(i)}$ be the ith column of T, we have

$$I_{n\times n} = T^{-1}T = T^{-1}\left[t^{(1)} \ t^{(2)} \ \cdots \ t^{(n)}\right] = \left[T^{-1}t^{(1)} \ \ T^{-1}t^{(2)} \ \ \cdots \ \ T^{-1}t^{(n)}\right]$$

so that

$$T^{-1}t^{(i)} = \big[\underbrace{0 \ \cdots \ 0}_{i-1 \text{ times}} \ 1 \ \underbrace{0 \ \cdots \ 0}_{n-i \text{ times}}\big]^T \in \{0,1\}^n,$$

the ith standard basis vector, which we denote by $e^{(i)}$ (that is, the vector of all-zeros except for a 1 in its ith position. Thus

$$\Lambda T^{-1}t^{(i)} = \Lambda e^{(i)} = \lambda_i e^{(i)}, \ \text{ and } \ T\Lambda T^{-1}t^{(i)} = \lambda_i T e^{(i)} = \lambda_i t^{(i)}.$$

A matrix $U \in \mathbb{R}^{n\times n}$ is orthogonal if $U^TU = I$. The spectral theorem, perhaps one of the most important theorems in linear algebra, states that if $A \in \mathbb{R}^{n\times n}$ is symetric, that is, $A = A^T$, then A is *diagonalizable by a real orthogonal matrix*. That is, there are a diagonal matrix $\Lambda \in \mathbb{R}^{n\times n}$ and orthogonal matrix $U \in \mathbb{R}^{n\times n}$ such that $U^TAU = \Lambda$, or, equivalently,

$$A = U\Lambda U^T.$$

Let $\lambda_i = \lambda_i(A)$ denote the ith eigenvalue of A.

(b) Let A be symmetric. Show that if $U = [u^{(1)} \ \cdots \ u^{(n)}]$ is orthogonal, where $u^{(i)} \in \mathbb{R}^n$ and $A = U\Lambda U^T$, then $u^{(i)}$ is an eigenvector of A and $Au^{(i)} = \lambda_i u^{(i)}$, where $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$.

Answer: Once we see that $U^{-1} = U^T$ because $U^TU = I$, this is simply a repeated application of part (3a).

(c) Show that if A is PSD, then $\lambda_i(A) \geq 0$ for each i.

Answer: Let $x \in \mathbb{R}^n$ be any vector. We know that $A = A^T$, so that $A = U\Lambda U^T$ for an orthogonal matrix $U \in \mathbb{R}^{n\times n}$ by the spectral theorem. Take the ith eigenvector $u^{(i)}$. Then we have

$$U^Tu^{(i)} = e^{(i)},$$

the ith standard basis vector. Using this, we have

$$0 \leq {u^{(i)}}^T Au^{(i)} = (U^Tu^{(i)})^T\Lambda U^Tu^{(i)} = {e^{(i)}}^T\Lambda e^{(i)} = \lambda_i(A).$$

CS 229, Autumn 2016
Problem Set #1: Supervised Learning

Due Wednesday, October 19 at 11:00 am on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `http://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a copy of your code (with comments) and any figures that you are asked to plot. If typing your solutions, include your code as text in your PDF. Do not submit extra files. (5) To account for late days, the due date listed on Gradescope is October 22 at 11 am. If you submit after October 19, you will begin consuming your late days. If you wish to submit on time, submit before October 19 at 11 am.

All students must submit an electronic PDF version. We highly recommend typesetting your solutions via latex. If you are scanning your document by cell phone, please check the Piazza forum for recommended scanning apps and best practices.

1. **[25 points] Logistic regression**

(a) [10 points] Consider the average empirical loss (the risk) for logistic regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \log(1 + e^{-y^{(i)}\theta^T x^{(i)}}) = -\frac{1}{m}\sum_{i=1}^{m} \log(h_\theta(y^{(i)}x^{(i)}))$$

where $h_\theta(x) = g(\theta^T x)$ and $g(z) = 1/(1 + e^{-z})$. Find the Hessian H of this function, and show that for any vector z, it holds true that

$$z^T H z \geq 0.$$

Hint: You might want to start by showing the fact that $\sum_i \sum_j z_i x_i x_j z_j = (x^T z)^2 \geq 0$.

Remark: This is one of the standard ways of showing that the matrix H is positive semi-definite, written "$H \succeq 0$." This implies that J is convex, and has no local minima other than the global one.[1] If you have some other way of showing $H \succeq 0$, you're also welcome to use your method instead of the one above.

(b) [10 points] We have provided two data files:

- `http://cs229.stanford.edu/ps/ps1/logistic_x.txt`
- `http://cs229.stanford.edu/ps/ps1/logistic_y.txt`

These files contain the inputs ($x^{(i)} \in \mathbb{R}^2$) and outputs ($y^{(i)} \in \{-1, 1\}$), respectively for a binary classification problem, with one training example per row. Implement[2] Newton's method for optimizing $J(\theta)$, and apply it to fit a logistic regression model to the data. Initialize Newton's method with $\theta = \vec{0}$ (the vector of all zeros). What are the coefficients θ resulting from your fit? (Remember to include the intercept term.)

[1] If you haven't seen this result before, please feel encouraged to ask us about it during office hours.
[2] Write your own version, and do not call a built-in library function.

(c) [5 points] Plot the training data (your axes should be x_1 and x_2, corresponding to the two coordinates of the inputs, and you should use a different symbol for each point plotted to indicate whether that example had label 1 or -1). Also plot on the same figure the decision boundary fit by logistic regression. (This should be a straight line showing the boundary separating the region where $h_\theta(x) > 0.5$ from where $h_\theta(x) \leq 0.5$.)

2. **[15 points] Poisson regression and the exponential family**

(a) [5 points] Consider the Poisson distribution parameterized by λ:

$$p(y; \lambda) = \frac{e^{-\lambda}\lambda^y}{y!}.$$

Show that the Poisson distribution is in the exponential family, and clearly state what are $b(y)$, η, $T(y)$, and $a(\eta)$.

(b) [3 points] Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter λ has mean λ.)

(c) [7 points] For a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)}; \theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent rule for learning using a GLM model with Poisson responses y and the canonical response function.

(d) **[3 extra credit points]** Consider using GLM with a response variable from any member of the exponential family in which $T(y) = y$, and the canonical response function $h(x)$ for the family. Show that stochastic gradient ascent on the log-likelihood $\log p(\vec{y}|X; \theta)$ results in the update rule $\theta_i := \theta_i - \alpha(h(x) - y)x_i$.

3. **[15 points] Gaussian discriminant analysis**

Suppose we are given a dataset $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ consisting of m independent examples, where $x^{(i)} \in \mathbb{R}^n$ are n-dimensional vectors, and $y^{(i)} \in \{-1, 1\}$. We will model the joint distribution of (x, y) according to:

$$\begin{aligned}
p(y) &= \begin{cases} \phi & \text{if } y = 1 \\ 1 - \phi & \text{if } y = -1 \end{cases} \\
p(x|y = -1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_{-1})^T\Sigma^{-1}(x - \mu_{-1})\right) \\
p(x|y = 1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_1)^T\Sigma^{-1}(x - \mu_1)\right)
\end{aligned}$$

Here, the parameters of our model are ϕ, Σ, μ_{-1} and μ_1. (Note that while there're two different mean vectors μ_{-1} and μ_1, there's only one covariance matrix Σ.)

(a) [5 points] Suppose we have already fit ϕ, Σ, μ_{-1} and μ_1, and now want to make a prediction at some new query point x. Show that the posterior distribution of the label at x takes the form of a logistic function, and can be written

$$p(y \mid x; \phi, \Sigma, \mu_{-1}, \mu_1) = \frac{1}{1 + \exp(-y(\theta^T x + \theta_0))},$$

where $\theta \in \mathbb{R}^n$ and the bias term $\theta_0 \in \mathbb{R}$ are some appropriate functions of $\phi, \Sigma, \mu_{-1}, \mu_1$. (Note: the term θ_0 corresponds to introducing an extra coordinate $x_0^{(i)} = 1$, as we did in class.)

(b) [10 points] For this part of the problem only, you may assume n (the dimension of x) is 1, so that $\Sigma = [\sigma^2]$ is just a real number, and likewise the determinant of Σ is given by $|\Sigma| = \sigma^2$. Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^{m} 1\{y^{(i)} = 1\} \\
\mu_{-1} &= \frac{\sum_{i=1}^{m} 1\{y^{(i)} = -1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = -1\}} \\
\mu_1 &= \frac{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} x^{(i)}}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\}} \\
\Sigma &= \frac{1}{m}\sum_{i=1}^{m} (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T
\end{aligned}$$

The log-likelihood of the data is

$$\begin{aligned}
\ell(\phi, \mu_{-1}, \mu_1, \Sigma) &= \log \prod_{i=1}^{m} p(x^{(i)}, y^{(i)}; \phi, \mu_{-1}, \mu_1, \Sigma) \\
&= \log \prod_{i=1}^{m} p(x^{(i)} | y^{(i)}; \mu_{-1}, \mu_1, \Sigma) p(y^{(i)}; \phi).
\end{aligned}$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, μ_{-1}, μ_1, and Σ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_{-1} and μ_1 above are non-zero.)

(c) **[3 extra credit points]** Without assuming that $n = 1$, show that the maximum likelihood estimates of ϕ, μ_{-1}, μ_1, and Σ are as given in the formulas in part (b). [Note: If you're fairly sure that you have the answer to this part right, you don't have to do part (b), since that's just a special case.]

4. **[10 points] Linear invariance of optimization algorithms**

Consider using an iterative optimization algorithm (such as Newton's method, or gradient descent) to minimize some continuously differentiable function $f(x)$. Suppose we initialize the algorithm at $x^{(0)} = \vec{0}$. When the algorithm is run, it will produce a value of $x \in \mathbb{R}^n$ for each iteration: $x^{(1)}, x^{(2)}, \ldots$.

Now, let some non-singular square matrix $A \in \mathbb{R}^{n \times n}$ be given, and define a new function $g(z) = f(Az)$. Consider using the same iterative optimization algorithm to optimize g (with initialization $z^{(0)} = \vec{0}$). If the values $z^{(1)}, z^{(2)}, \ldots$ produced by this method necessarily satisfy $z^{(i)} = A^{-1}x^{(i)}$ for all i, we say this optimization algorithm is **invariant to linear reparameterizations**.

(a) [7 points] Show that Newton's method (applied to find the minimum of a function) is invariant to linear reparameterizations. Note that since $z^{(0)} = \vec{0} = A^{-1}x^{(0)}$, it is sufficient

to show that if Newton's method applied to $f(x)$ updates $x^{(i)}$ to $x^{(i+1)}$, then Newton's method applied to $g(z)$ will update $z^{(i)} = A^{-1}x^{(i)}$ to $z^{(i+1)} = A^{-1}x^{(i+1)}$.[3]

(b) [3 points] Is gradient descent invariant to linear reparameterizations? Justify your answer.

5. **[35 points] Regression for denoising quasar spectra**[4]

Introduction. In this problem, we will apply a supervised learning technique to estimate the light spectrum of *quasars*. Quasars are luminous distant galactic nuclei that are so bright, their light overwhelms that of stars in their galaxies. Understanding properties of the spectrum of light emitted by a quasar is useful for a number of tasks: first, a number of quasar properties can be estimated from the spectra, and second, properties of the regions of the universe through which the light passes can also be evaluated (for example, we can estimate the density of neutral and ionized particles in the universe, which helps cosmologists understand the evolution and fundamental laws governing its structure). The *light spectrum* is a curve that relates the light's intensity (formally, lumens per square meter), or *luminous flux*, to its wavelength. Figure 1 shows an example of a quasar light spectrum, where the wavelengths are measured in Angstroms (Å), where 1Å$= 10^{-10}$ meters.

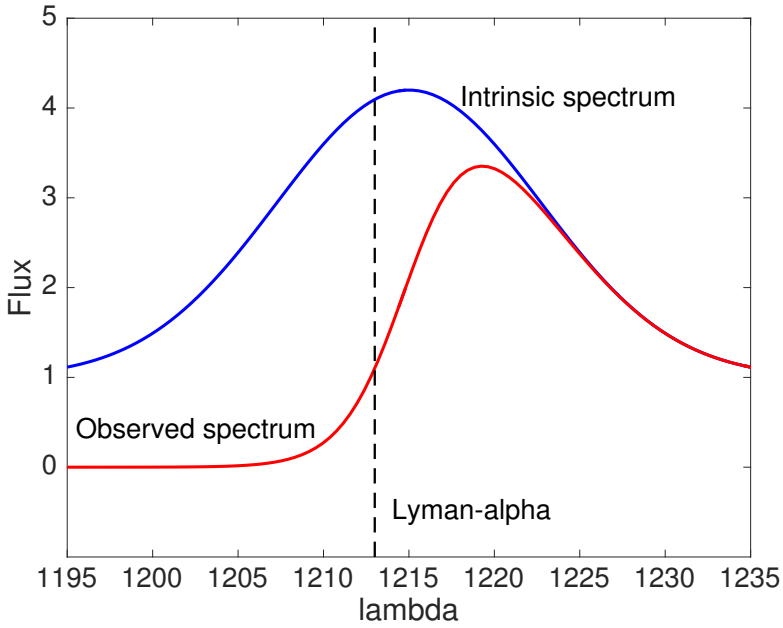


Figure 1: Light spectrum of a quasar. The blue line shows the intrinsic (i.e. original) flux spectrum emitted by the quasar. The red line denotes the observed spectrum here on Earth. To the left of the Lyman-α line, the observed flux is damped and the intrinsic (unabsorbed) flux continuum is not clearly recognizable (red line). To the right of the Lyman-α line, the observed flux approximates the intrinsic spectrum.

The Lyman-α wavelength is a wavelength beyond which intervening particles at most negligibly interfere with light emitted from the quasar. (Interference generally occurs when a photon is

[3] Note that for this problem, you must explicitly prove any matrix calculus identities that you wish to use that are not given in the lecture notes.

[4] Ciollaro, Mattia, et al. "Functional regression for quasar spectra." arXiv:1404.3168 (2014).

absorbed by a neutral hydrogen atom, which only occurs for certain wavelengths of light.) For wavelengths greater than this Lyman-α wavelength, the observed light spectrum f_{obs} can be modeled as a smooth spectrum f plus noise:

$$f_{\text{obs}}(\lambda) = f(\lambda) + \text{noise}(\lambda)$$

For wavelengths below the Lyman-α wavelength, a region of the spectrum known as the Lyman-α forest, intervening matter causes attenuation of the observed signal. As light emitted by the quasar travels through regions of the universe richer in neutral hydrogen, some of it is absorbed, which we model as

$$f_{\text{obs}}(\lambda) = \text{absorption}(\lambda) \cdot f(\lambda) + \text{noise}(\lambda)$$

Astrophysicists and cosmologists wish to understand the absorption function, which gives information about the Lyman-α forest, and hence the distribution of neutral hydrogen in otherwise unreachable regions of the universe. This gives clues toward the formation and evolution of the universe. Thus, it is our goal to estimate the spectrum f of an observed quasar.

Getting the data. We will be using data generated from the Hubble Space Telescope Faint Object Spectrograph (HST-FOS), Spectra of Active Galactic Nuclei and Quasars.[5] We have provided two comma-separated data files located at:

- Training set: `http://cs229.stanford.edu/ps/ps1/quasar_train.csv`
- Test set: `http://cs229.stanford.edu/ps/ps1/quasar_test.csv`

Each file contains a single header row containing 450 numbers corresponding integral wavelengths in the interval $[1150, 1600]$ Å. The remaining lines contain relative flux measurements for each wavelength. Specifically, `quasar_train.csv` contains 200 examples and `quasar_test.csv` contains 50 examples. You may use the helper file `load_quasar_data.m` to load the data in Matlab: `http://cs229.stanford.edu/ps/ps1/load_quasar_data.m`

(a) [10 points] Locally weighted linear regression

Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

In class, we worked out what happens for the case where all the weights (the $w^{(i)}$'s) are the same. In this problem, we will generalize some of those ideas to the weighted setting.

i. [2 points] Show that $J(\theta)$ can also be written

$$J(\theta) = (X\theta - \vec{y})^T W (X\theta - \vec{y})$$

for an appropriate diagonal matrix W, and where X and $\vec{y}$ are as defined in class. State clearly what W is.

ii. [4 points] If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is

$$X^T X\theta = X^T \vec{y},$$

[5] `https://hea-www.harvard.edu/FOSAGN/`

and that the value of θ that minimizes $J(\theta)$ is given by $(X^TX)^{-1}X^T\vec{y}$. By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and $\vec{y}$.

iii. [4 points] Suppose we have a training set $\{(x^{(i)}, y^{(i)}); i = 1\ldots, m\}$ of m independent examples, but in which the $y^{(i)}$'s were observed with differing variances. Specifically, suppose that

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

I.e., $y^{(i)}$ has mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.

(b) [6 points] Visualizing the data

i. [2 points] Use the normal equations to implement (unweighted) linear regression ($y = \theta^T x$) on the *first* training example (i.e. first non-header row). On one figure, plot both the raw data and the straight line resulting from your fit. State the optimal θ resulting from the linear regression.

ii. [2 points] Implement locally weighted linear regression on the *first* training example. Use the normal equations you derived in part (a)(ii). On a different figure, plot both the raw data and the smooth curve resulting from your fit. When evaluating $h(\cdot)$ at a query point x, use weights

$$w^{(i)} = \exp\left(-\frac{(x - x^{(i)})^2}{2\tau^2}\right),$$

with bandwidth parameter $\tau = 5$.

iii. [2 points] Repeat (b)(ii) four more times with $\tau = 1, 10, 100$ and 1000. Plot the resulting curves. You can submit one plot with all four τ values or submit four separate plots. If you submit one plot, make sure all curves are visible. Additionally, in **2-3 sentences**, comment on what happens to the locally weighted linear regression line as τ varies.

(c) [19 points] Predicting quasar spectra with functional regression

We now go a step beyond what we have covered explicitly in class, and we wish to predict an entire part of a spectrum—a curve—from noisy observed data. We begin by supposing that we observe a random sample of m absorption-free spectra, which is possible for quasars very close (in a sense relative to the size of the universe!) to Earth. For a given spectrum f, define f_{right} to be the spectrum to the right of the Lyman-α line. Let f_{left} be the spectrum within the Lyman-α forest region, that is, for lower wavelengths. To make the results cleaner, we define:

$$f(\lambda) = \begin{cases} f_{\text{left}}(\lambda) & \text{if } \lambda < 1200 \\ f_{\text{right}}(\lambda) & \text{if } \lambda \geq 1300 \end{cases}$$

We will learn a function r (for regression) that maps an observed f_{right} to an unobserved target f_{left}. This is useful in practice because we observe f_{right} with *only* random noise: there is no systematic absorption, which we cannot observe directly, because hydrogen does not absorb photons with higher wavelengths. By predicting f_{left} from a noisy version of

f_{right}, we can estimate the unobservable spectrum of a quasar as well as the absorption function. Imaging systems collect data of the form

$$f_{\text{obs}}(\lambda) = \text{absorption}(\lambda) \cdot f(\lambda) + \text{noise}(\lambda)$$

for $\lambda \in \{\lambda_1, \ldots, \lambda_n\}$, a *finite* number of points λ, because they must quantize the information. That is, even in the quasars-close-to-Earth training data, our observations of f_{left} and f_{right} consist of noisy evaluations of the true spectrum f at multiple wavelengths. In our case, we have $n = 450$ and $\lambda_1 = 1150, \ldots, \lambda_n = 1599$.

We formulate the functional regression task as the goal of learning the function r mapping f_{left} to f_{right}:

$$r(f_{\text{right}})(\lambda) = \mathbb{E}(f_{\text{left}} \mid f_{\text{right}})(\lambda)$$

for λ in the Lyman-α forest.

i. [1 points] First, we must smooth the data in the training dataset to make it more useful for prediction. For each $i = 1, \ldots, m$, define $f^{(i)}(\lambda)$ to be the weighted linear regression estimate the i^{th} spectrum. Use your code from part (b)(ii) above to smooth all spectra in the training set using $\tau = 5$. Do the same for the test set. We will now operate on these smoothed spectra.

ii. [14 points] Using your estimated regression functions $f^{(i)}$ for $i = 1, \ldots, m$, we now wish to estimate the unobserved spectrum f_{left} of a quasar from its (noisy) observed spectrum f_{right}. To do so, we perform a weighted regression of the *locally weighted regressions*. In particular, given a new noisy spectrum observation:

$$f_{\text{obs}}(\lambda) = f(\lambda) + \text{noise}(\lambda) \quad \text{for } \lambda \in \{1300, \ldots, 1599\}.$$

We define a metric d which takes as input, two spectra f_1 and f_2, and outputs a scalar:

$$d(f_1, f_2) = \sum_i \Big(f_1(\lambda_i) - f_2(\lambda_i)\Big)^2.$$

The metric d computes squared distance between the new datapoint and previous datapoints. If f_1 and f_2 are right spectra, then we take the preceding sum only over $\lambda \in \{1300, \ldots, 1599\}$, rather than the entire spectrum.

Based on this distance function, we may define the nonparametric *functional* regression estimator, which is a locally weighted sum of *functions* f_{left} from the training data (this is like locally weighted linear regression, except that instead of predicting $y \in \mathbb{R}$ we predict a function f_{left}). Specifically, let f_{right} denote the right side of a spectrum, which we have smoothed using locally weighted linear regression (as you were told to do in the previous part of the problem). We wish to estimate the associated *left* spectrum f_{left}. Define the function $\ker(t) = \max\{1 - t, 0\}$ and let $\mathsf{neighb}_k(f_{\text{right}})$ denote the k indices $i \in \{1, 2, \ldots, m\}$ that are closest to f_{right}, that is

$$d(f^{(i)}_{\text{right}}, f_{\text{right}}) < d(f^{(j)}_{\text{right}}, f_{\text{right}}) \quad \text{for all } i \in \mathsf{neighb}_k(f_{\text{right}}),\ j \notin \mathsf{neighb}_k(f_{\text{right}})$$

and $\mathsf{neighb}_k(f_{\text{right}})$ contains exactly k indices. In addition, let

$$h := \max_{i \in \{1, \ldots, m\}} d(f^{(i)}_{\text{right}}, f_{\text{right}}).$$

Then define the estimated function $\widehat{f_{\text{left}}} : \mathbb{R} \to \mathbb{R}$ by

$$\widehat{f_{\text{left}}}(\lambda) = \frac{\sum_{i \in \mathsf{neighb}_k(f_{\text{right}})} \ker(d(f^{(i)}_{\text{right}}, f_{\text{right}})/h) f^{(i)}_{\text{left}}(\lambda)}{\sum_{i \in \mathsf{neighb}_k(f_{\text{right}})} \ker(d(f^{(i)}_{\text{right}}, f_{\text{right}})/h)}. \tag{1}$$

Recall that $f^{(i)}_{\text{right}}$ is the *smoothed* (weighted linear regression) estimate of the ith training spectrum.

Construct the functional regression estimate (1) for each spectrum in the entire training set using $k = 3$ nearest neighbors: for each $j = 1, \ldots, m$, construct the estimator $\widehat{f_{\text{left}}}$ from (1) using $f_{\text{right}} = f^{(j)}_{\text{right}}$. Then compute the error $d(f^{(j)}_{\text{left}}, \widehat{f_{\text{left}}})$ between the true spectrum $f^{(j)}_{\text{left}}$ and your estimated spectrum $\widehat{f_{\text{left}}}$ for each j, and return the average over the training data. What is your average training error?

iii. [4 points] Perform functional regression on the test set using the same procedure as in the previous subquestion. What is your average test error? For test examples 1 and 6, include a plot with both the entire smooth spectrum and the fitted curve $\widehat{f_{\text{left}}}$ curve on the same graph. You should submit two plots: one for test example 1 and one for test example 6.

Reminder: Please include in your submission a printout of your code and figures for the programming questions.

CS 229, Autumn 2016
Problem Set #1 Solutions: Supervised Learning

Due Wednesday, October 19 at 11:00 am on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `http://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a copy of your code (with comments) and any figures that you are asked to plot. If typing your solutions, include your code as text in your PDF. Do not submit extra files. (5) To account for late days, the due date listed on Gradescope is October 22 at 11 am. If you submit after October 19, you will begin consuming your late days. If you wish to submit on time, submit before October 19 at 11 am.

All students must submit an electronic PDF version. We highly recommend typesetting your solutions via latex. If you are scanning your document by cell phone, please check the Piazza forum for recommended scanning apps and best practices.

1. **[25 points] Logistic regression**

(a) [10 points] Consider the average empirical loss (the risk) for logistic regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \log(1 + e^{-y^{(i)}\theta^T x^{(i)}}) = -\frac{1}{m}\sum_{i=1}^{m} \log(h_\theta(y^{(i)}x^{(i)}))$$

where $h_\theta(x) = g(\theta^T x)$ and $g(z) = 1/(1 + e^{-z})$. Find the Hessian H of this function, and show that for any vector z, it holds true that

$$z^T H z \geq 0.$$

Hint: You might want to start by showing the fact that $\sum_i \sum_j z_i x_i x_j z_j = (x^T z)^2 \geq 0$.

Remark: This is one of the standard ways of showing that the matrix H is positive semi-definite, written "$H \succeq 0$." This implies that J is convex, and has no local minima other than the global one.[1] If you have some other way of showing $H \succeq 0$, you're also welcome to use your method instead of the one above.

Answer: (Note we do things in a slightly shorter way here; this solution does not use the hint.) Note that if $g(z) = 1/(1 + e^{-z})$, then $g'(z) = g(z)(1 - g(z))$. and thus for $h(x) = g(\theta^T x)$, we have $\frac{\partial h(x)}{\partial \theta_k} = h(x)(1-h(x))x_k$. This latter fact is very useful to make the following derivations. Remember we have shown in class:

$$\frac{\partial}{\partial \theta_k} \log(1 + e^{-yx^T\theta}) = -\frac{1}{1 + e^{yx^T\theta}} yx = -h_\theta(-yx)yx.$$

Thus we have

$$\frac{\partial}{\partial \theta_k} J(\theta) = \frac{1}{m}\sum_{i=1}^{m} -\frac{1}{1 + e^{y^{(i)}\theta^T x^{(i)}}} y^{(i)} x_k^{(i)} = -\frac{1}{m}\sum_{i=1}^{m} h_\theta(-y^{(i)}x^{(i)})y^{(i)}x_k^{(i)}.$$

[1] If you haven't seen this result before, please feel encouraged to ask us about it during office hours.

Consequently, we have the Hessian

$$\begin{aligned} H_{kl} &= \frac{\partial^2}{\partial\theta_k \partial\theta_l} J(\theta) \\ &= -\frac{1}{m}\sum_{i=1}^{m} \frac{\partial}{\partial\theta_l} h_\theta(-y^{(i)}x^{(i)}) y^{(i)} x_k^{(i)} \\ &= \frac{1}{m}\sum_{i=1}^{m} h_\theta(x^{(i)})(1-h_\theta(x^{(i)})) x_l^{(i)} x_k^{(i)}. \end{aligned}$$

So we have for the Hessian matrix H (using that for $X = xx^T$if and only if$X_{ij} = x_i x_j$):

$$H = \frac{1}{m}\sum_{i=1}^{m} h(x^{(i)})(1-h(x^{(i)})) x^{(i)} x^{(i)T}.$$

To prove H is positive semidefinite, we show $z^T H z \geq 0$ for all z:

$$\begin{aligned} z^T H z &= \frac{1}{m} z^T \left(\sum_{i=1}^{m} h(x^{(i)})(1-h(x^{(i)})) x^{(i)} x^{(i)T} \right) z \\ &= \frac{1}{m}\sum_{i=1}^{m} h(x^{(i)})(1-h(x^{(i)})) z^T x^{(i)} x^{(i)T} z \\ &= \frac{1}{m}\sum_{i=1}^{m} h(x^{(i)})(1-h(x^{(i)}))(z^T x^{(i)})^2 \geq 0. \end{aligned}$$

The last inequality holds, because $0 \leq h(x^{(i)}) \leq 1$, which implies $h(x^{(i)})(1-h(x^{(i)})) \geq 0$, and $(z^T x^{(i)})^2) \geq 0$.

(b) [10 points] We have provided two data files:

- `http://cs229.stanford.edu/ps/ps1/logistic_x.txt`
- `http://cs229.stanford.edu/ps/ps1/logistic_y.txt`

These files contain the inputs ($x^{(i)} \in \mathbb{R}^2$) and outputs ($y^{(i)} \in \{-1, 1\}$), respectively for a binary classification problem, with one training example per row. Implement[2] Newton's method for optimizing $J(\theta)$, and apply it to fit a logistic regression model to the data. Initialize Newton's method with $\theta = \vec{0}$ (the vector of all zeros). What are the coefficients θ resulting from your fit? (Remember to include the intercept term.)

Answer: $\theta = (-2.6205, 0.7604, 1.1719)$ with the first entry corresponding to the intercept term.

```
%%%%%%%   plot_log_regression.m   %%%%%%%

X = load('logistic_x.txt');
Y = load('logistic_y.txt');

X = [ones(size(X, 1), 1) X];
[theta, ll] = log_regression(X ,Y, 20);
```

[2]Write your own version, and do not call a built-in library function.

```
m=size(X,1);
figure; hold on;

plot(X(Y < 0, 2), X(Y < 0, 3), 'rx', 'linewidth', 2);
plot(X(Y > 0, 2), X(Y > 0, 3), 'go', 'linewidth', 2);

x1 = min(X(:,2)):.01:max(X(:,2));
x2 = -(theta(1) / theta(3)) - (theta(2) / theta(3)) * x1;

plot(x1,x2, 'linewidth', 2);
xlabel('x1');
ylabel('x2');


%%%%%%% log_regression.m %%%%%%%

function [theta,ll] = log_regression(X,Y, max_iters)

% rows of X are training samples
% rows of Y are corresponding -1/1 values

% newton raphson: theta = theta - inv(H)* grad;
% with H = hessian, grad = gradient

mm = size(X,1);
nn = size(X,2);

theta = zeros(nn,1);

ll = zeros(max_iters, 1);
for ii = 1:max_iters

  margins = Y .* (X * theta);
  ll(ii) = (1/mm) * sum(log(1 + exp(-margins)));
  probs = 1 ./ (1 + exp(margins));
  grad = -(1/mm) * (X' * (probs .* Y));
  H = (1/mm) * (X' * diag(probs .* (1 - probs)) * X);
  theta = theta - H \ grad;
end
```

(c) [5 points] Plot the training data (your axes should be x_1 and x_2, corresponding to the two coordinates of the inputs, and you should use a different symbol for each point plotted to indicate whether that example had label 1 or -1). Also plot on the same figure the decision boundary fit by logistic regression. (This should be a straight line showing the boundary separating the region where $h_\theta(x) > 0.5$ from where $h_\theta(x) \leq 0.5$.)

Answer:

2. **[15 points] Poisson regression and the exponential family**

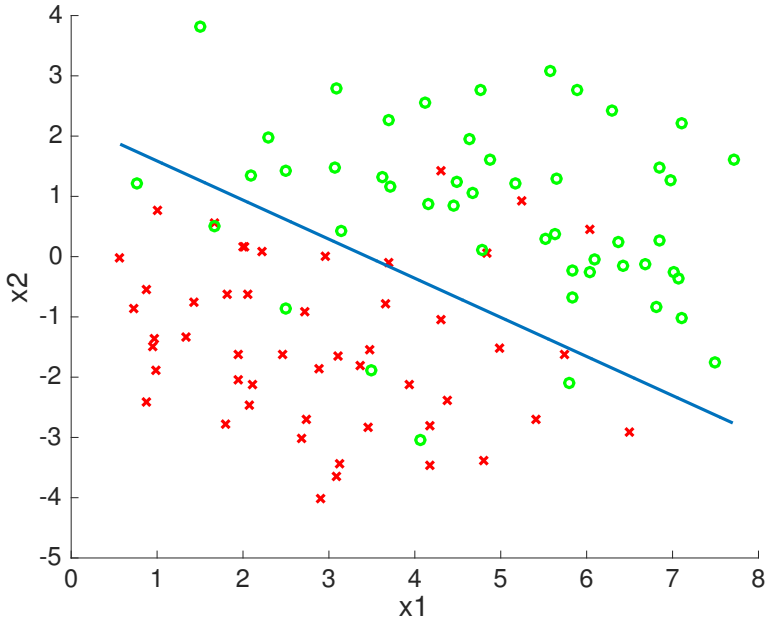


Figure 1: Separating hyperplane for logistic regression (question 1c).

(a) [5 points] Consider the Poisson distribution parameterized by λ:

$$p(y;\lambda) = \frac{e^{-\lambda}\lambda^y}{y!}.$$

Show that the Poisson distribution is in the exponential family, and clearly state what are $b(y)$, η, $T(y)$, and $a(\eta)$.

Answer: Rewrite the distribution function as:

$$\begin{aligned} p(y;\lambda) &= \frac{e^{-\lambda}e^{y\log\lambda}}{y!} \\ &= \frac{1}{y!}\exp(y\log\lambda - \lambda) \end{aligned}$$

Comparing with the standard form for the exponential family:

$$\begin{aligned} b(y) &= \frac{1}{y!} \\ \eta &= \log\lambda \\ T(y) &= y \\ a(\eta) &= e^{\eta} \end{aligned}$$

(b) [3 points] Consider performing regression using a GLM model with a Poisson response variable. What is the canonical response function for the family? (You may use the fact that a Poisson random variable with parameter λ has mean λ.)

Answer: The canonical response function for the GLM model will be:

$$\begin{aligned} g(\eta) &= E[y;\eta] \\ &= \lambda \\ &= e^{\eta} \end{aligned}$$

(c) [7 points] For a training set $\{(x^{(i)}, y^{(i)});\ i = 1, \ldots, m\}$, let the log-likelihood of an example be $\log p(y^{(i)}|x^{(i)};\theta)$. By taking the derivative of the log-likelihood with respect to θ_j, derive the stochastic gradient ascent rule for learning using a GLM model with Poisson responses y and the canonical response function.

Answer: The log-likelihood of an example $(x^{(i)}, y^{(i)})$ is defined as $\ell(\theta) = \log p(y^{(i)}|x^{(i)};\theta)$. To derive the stochastic gradient ascent rule, use the results in part (a) and the standard GLM assumption that $\eta = \theta^T x$.

$$\begin{aligned} \frac{\partial \ell(\theta)}{\partial \theta_j} &= \frac{\partial \log p(y^{(i)}|x^{(i)};\theta)}{\partial \theta_j} \\ &= \frac{\partial \log \left(\frac{1}{y^{(i)}!} \exp(\eta^T y^{(i)} - e^{\eta}) \right)}{\partial \theta_j} \\ &= \frac{\partial \log \left(\exp((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}}) \right)}{\partial \theta_j} + \frac{\partial \log \left(\frac{1}{y^{(i)}!} \right)}{\partial \theta_j} \\ &= \frac{\partial \left((\theta^T x^{(i)})^T y^{(i)} - e^{\theta^T x^{(i)}} \right)}{\partial \theta_j} \\ &= \frac{\partial \left((\sum_k \theta_k x_k^{(i)}) y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}} \right)}{\partial \theta_j} \\ &= x_j^{(i)} y^{(i)} - e^{\sum_k \theta_k x_k^{(i)}} x_j^{(i)} \\ &= (y^{(i)} - e^{\theta^T x^{(i)}}) x_j^{(i)} \end{aligned}$$

Thus the stochastic gradient ascent update rule should be:

$$\theta_j := \theta_j + \alpha \frac{\partial \ell(\theta)}{\partial \theta_j}$$

which reduces here to:

$$\theta_j := \theta_j + \alpha (y^{(i)} - e^{\theta^T x}) x_j^{(i)}$$

(d) **[3 extra credit points]** Consider using GLM with a response variable from any member of the exponential family in which $T(y) = y$, and the canonical response function $h(x)$ for the family. Show that stochastic gradient ascent on the log-likelihood $\log p(\vec{y}|X;\theta)$ results in the update rule $\theta_i := \theta_i - \alpha(h(x) - y)x_i$.

Answer: As in the previous part, consider the derivative of the likelihood of a training example

(x, y) with respect to the parameter θ_j:

$$\begin{aligned}
\frac{\partial \ell(\theta)}{\partial \theta_j} &= \frac{\partial \log p(y|x;\theta)}{\partial \theta_j} \\
&= \frac{\partial \log \left(b(y)\exp(\eta^T y - a(\eta))\right)}{\partial \theta_j} \\
&= \frac{\partial \left(\eta^T y - a(\eta))\right)}{\partial \theta_j} \\
&= x_j y - \frac{\partial a(\eta)}{\partial \eta} x_j \\
&= \left(y - \frac{\partial a(\eta)}{\partial \eta}\right) x_j
\end{aligned}$$

Thus, it only remains to show that $\frac{\partial a(\eta)}{\partial \eta} = h(x) = E[y|x;\theta]$. To prove this consider the fact that $p(y|x;\theta)$ is a probability distribution and must thus sum to 1.

$$\begin{aligned}
\int_y p(y|x;\theta)dy &= 1 \\
\int_y b(y)\exp(\eta^T y - a(\eta))dy &= 1 \\
\int_y b(y)\exp(\eta^T y)dy &= \exp(a(\eta))
\end{aligned}$$

Differentiating both sides with respect to η:

$$\begin{aligned}
\int_y b(y) y \exp(\eta^T y)dy &= \exp(a(\eta))\frac{\partial a(\eta)}{\partial \eta} \\
\frac{\partial a(\eta)}{\partial \eta} &= \int_y b(y) y \exp(\eta^T y - a(\eta))dy \\
&= \int_y y p(y|x;\theta)dy \\
&= E[y|x;\theta]
\end{aligned}$$

where the last step follows from the definition of the (conditional) expectation of a random variable. Substituting this into the expression for $\frac{\partial \ell(\theta)}{\partial \theta_j}$ gives the required gradient ascent update rule.

3. **[15 points] Gaussian discriminant analysis**

Suppose we are given a dataset $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ consisting of m independent examples, where $x^{(i)} \in \mathbb{R}^n$ are n-dimensional vectors, and $y^{(i)} \in \{-1, 1\}$. We will model the joint

distribution of (x, y) according to:

$$\begin{aligned} p(y) &= \begin{cases} \phi & \text{if } y = 1 \\ 1-\phi & \text{if } y = -1 \end{cases} \\ p(x|y=-1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1})\right) \\ p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right) \end{aligned}$$

Here, the parameters of our model are ϕ, Σ, μ_{-1} and μ_1. (Note that while there're two different mean vectors μ_{-1} and μ_1, there's only one covariance matrix Σ.)

(a) [5 points] Suppose we have already fit ϕ, Σ, μ_{-1} and μ_1, and now want to make a prediction at some new query point x. Show that the posterior distribution of the label at x takes the form of a logistic function, and can be written

$$p(y \mid x; \phi, \Sigma, \mu_{-1}, \mu_1) = \frac{1}{1+\exp(-y(\theta^T x + \theta_0))},$$

where $\theta \in \mathbb{R}^n$ and the bias term $\theta_0 \in \mathbb{R}$ are some appropriate functions of $\phi, \Sigma, \mu_{-1}, \mu_1$. (Note: the term θ_0 corresponds to introducing an extra coordinate $x_0^{(i)} = 1$, as we did in class.)

Answer: For shorthand, we let $\mathcal{H} = \{\phi, \Sigma, \mu_{-1}, \mu_1\}$ denote the parameters for the problem. Since the given formulae are conditioned on y, use Bayes rule to get:

$$\begin{aligned} p(y=1|x;\phi,\Sigma,\mu_{-1},\mu_1) &= \frac{p(x|y=1;\phi,\Sigma,\mu_{-1},\mu_1)p(y=1;\phi,\Sigma,\mu_{-1},\mu_1)}{p(x;\phi,\Sigma,\mu_{-1},\mu_1)} \\ &= \frac{p(x|y=1;\mathcal{H})p(y=1;\mathcal{H})}{p(x|y=1;\mathcal{H})p(y=1;\mathcal{H}) + p(x|y=-1;\mathcal{H})p(y=-1;\mathcal{H})} \\ &= \frac{\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\phi}{\exp\left(-\frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)\phi + \exp\left(-\frac{1}{2}(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1})\right)(1-\phi)} \\ &= \frac{1}{1+\frac{1-\phi}{\phi}\exp\left(-\frac{1}{2}(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1}) + \frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)} \\ &= \frac{1}{1+\exp\left(\log(\frac{1-\phi}{\phi}) - \frac{1}{2}(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1}) + \frac{1}{2}(x-\mu_1)^T\Sigma^{-1}(x-\mu_1)\right)}. \end{aligned}$$

Now, we expand and rearrange the difference of quadratic terms in the preceding expression, finding that

$$\begin{aligned} &(x-\mu_{-1})^T\Sigma^{-1}(x-\mu_{-1}) - (x-\mu_1)^T\Sigma^{-1}(x-\mu_1) \\ &= x^T\Sigma^{-1}x - \mu_{-1}^T\Sigma^{-1}x - x^T\Sigma^{-1}\mu_{-1} + \mu_{-1}^T\Sigma^{-1}\mu_{-1} - x^T\Sigma^{-1}x + \mu_1^T\Sigma^{-1}x + x^T\Sigma^{-1}\mu_1 - \mu_1^T\Sigma^{-1}\mu_1 \\ &= -2\mu_{-1}^T\Sigma^{-1}x + \mu_{-1}^T\Sigma^{-1}\mu_{-1} + 2\mu_1^T\Sigma^{-1}x - \mu_1^T\Sigma^{-1}\mu_1 \\ &= 2(\mu_1-\mu_{-1})^T\Sigma^{-1}x + \mu_{-1}^T\Sigma^{-1}\mu_{-1} - \mu_1^T\Sigma^{-1}\mu_1. \end{aligned}$$

Thus, we have

$$p(y=1 \mid x; \mathcal{H}) = \frac{1}{1+\exp\left(\log\frac{1-\phi}{\phi} + \frac{1}{2}\mu_1^T\Sigma^{-1}\mu_1 - \frac{1}{2}\mu_{-1}^T\Sigma^{-1}\mu_{-1} + (\mu_{-1}-\mu_1)^T\Sigma^{-1}x\right)}.$$

and setting

$$\theta = \Sigma^{-1}(\mu_1 - \mu_{-1}) \text{ and } \theta_0 = \frac{1}{2}(\mu_{-1}^T\Sigma^{-1}\mu_{-1} - \mu_1^T\Sigma^{-1}\mu_1) - \log\frac{1-\phi}{\phi}$$

gives that

$$p(y \mid x; \phi, \Sigma, \mu_{-1}, \mu_1) = \frac{1}{1+\exp(-y(\theta^T x + \theta_0))}.$$

(b) [10 points] For this part of the problem only, you may assume n (the dimension of x) is 1, so that $\Sigma = [\sigma^2]$ is just a real number, and likewise the determinant of Σ is given by $|\Sigma| = \sigma^2$. Given the dataset, we claim that the maximum likelihood estimates of the parameters are given by

$$\begin{aligned}
\phi &= \frac{1}{m}\sum_{i=1}^m 1\{y^{(i)} = 1\} \\
\mu_{-1} &= \frac{\sum_{i=1}^m 1\{y^{(i)} = -1\}x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = -1\}} \\
\mu_1 &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}x^{(i)}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}} \\
\Sigma &= \frac{1}{m}\sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T
\end{aligned}$$

The log-likelihood of the data is

$$\begin{aligned}
\ell(\phi, \mu_{-1}, \mu_1, \Sigma) &= \log\prod_{i=1}^m p(x^{(i)}, y^{(i)}; \phi, \mu_{-1}, \mu_1, \Sigma) \\
&= \log\prod_{i=1}^m p(x^{(i)}|y^{(i)}; \mu_{-1}, \mu_1, \Sigma)p(y^{(i)}; \phi).
\end{aligned}$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, μ_{-1}, μ_1, and Σ are indeed as given in the formulas above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_{-1} and μ_1 above are non-zero.)

Answer: The derivation follows from the more general one for the next part.

(c) **[3 extra credit points]** Without assuming that $n = 1$, show that the maximum likelihood estimates of ϕ, μ_{-1}, μ_1, and Σ are as given in the formulas in part (b). [Note: If you're fairly sure that you have the answer to this part right, you don't have to do part (b), since that's just a special case.]

Answer: First, derive the expression for the log-likelihood of the training data:

$$\begin{aligned}
\ell(\phi, \mu_{-1}, \mu_1, \Sigma) &= \log\prod_{i=1}^m p(x^{(i)}|y^{(i)}; \mu_{-1}, \mu_1, \Sigma)p(y^{(i)}; \phi) \\
&= \sum_{i=1}^m \log p(x^{(i)}|y^{(i)}; \mu_{-1}, \mu_1, \Sigma) + \sum_{i=1}^m \log p(y^{(i)}; \phi) \\
&\simeq \sum_{i=1}^m \big[\frac{1}{2}\log\frac{1}{|\Sigma|} - \frac{1}{2}(x^{(i)} - \mu_{y^{(i)}})^T\Sigma^{-1}(x^{(i)} - \mu_{y^{(i)}}) + y^{(i)}\log\phi + (1-y^{(i)})\log(1-\phi)\big]
\end{aligned}$$

where constant terms indepedent of the parameters have been ignored in the last expression.

Now, the likelihood is maximized by setting the derivative (or gradient) with respect to each of the parameters to zero.

$$\begin{aligned}\frac{\partial \ell}{\partial \phi} &= \sum_{i=1}^{m} \big[\frac{y^{(i)}}{\phi} - \frac{1-y^{(i)}}{1-\phi}\big] \\ &= \frac{\sum_{i=1}^{m} 1\{y^{(i)}=1\}}{\phi} - \frac{m - \sum_{i=1}^{m} 1\{y^{(i)}=1\}}{1-\phi}\end{aligned}$$

Setting this equal to zero and solving for ϕ gives the maximum likelihood estimate.

For μ_{-1}, take the gradient of the log-likelihood, and then use the same kinds of tricks as were used to analytically solve the linear regression problem.

$$\begin{aligned}\nabla_{\mu_{-1}} \ell &= -\frac{1}{2} \sum_{i:y^{(i)}=-1} \nabla_{\mu_{-1}} (x^{(i)} - \mu_{-1})^T \Sigma^{-1} (x^{(i)} - \mu_{-1}) \\ &= -\frac{1}{2} \sum_{i:y^{(i)}=-1} \nabla_{\mu_{-1}} \big[\mu_{-1}^T \Sigma^{-1} \mu_{-1} - {x^{(i)}}^T \Sigma^{-1} \mu_{-1} - \mu_{-1}^T \Sigma^{-1} x^{(i)}\big] \\ &= -\frac{1}{2} \sum_{i:y^{(i)}=-1} \nabla_{\mu_{-1}} tr\big[\mu_{-1}^T \Sigma^{-1} \mu_{-1} - {x^{(i)}}^T \Sigma^{-1} \mu_{-1} - \mu_{-1}^T \Sigma^{-1} x^{(i)}\big] \\ &= -\frac{1}{2} \sum_{i:y^{(i)}=-1} \big[2\Sigma^{-1}\mu_{-1} - 2\Sigma^{-1} x^{(i)}\big]\end{aligned}$$

The last step uses matrix calculus identities (specifically, those given in page 8 of the lecture notes), and also the fact that Σ (and thus Σ^{-1}) is symmetric.

Setting this gradient to zero gives the maximum likelihood estimate for μ_{-1}. The derivation for μ_1 is similar to the one above.

For Σ, we find the gradient with respect to $S = \Sigma^{-1}$ rather than Σ just to simplify the derivation (note that $|S| = \frac{1}{|\Sigma|}$). You should convince yourself that the maximum likelihood estimate S_m found in this way would correspond to the actual maximum likelihood estimate Σ_m as $S_m^{-1} = \Sigma_m$.

$$\begin{aligned}\nabla_S \ell &= \sum_{i=1}^{m} \nabla_S \big[\frac{1}{2}\log|S| - \frac{1}{2} \underbrace{(x^{(i)} - \mu_{y^{(i)}})^T}_{b_i^T} S \underbrace{(x^{(i)} - \mu_{y^{(i)}})}_{b_i}\big] \\ &= \sum_{i=1}^{m} \big[\frac{1}{2|S|}\nabla_S |S| - \frac{1}{2}\nabla_S b_i^T S b_i\big]\end{aligned}$$

But, we have the following identities:

$$\nabla_S |S| = |S|(S^{-1})^T$$

$$\nabla_S b_i^T S b_i = \nabla_S tr\left(b_i^T S b_i\right) = \nabla_S tr\left(S b_i b_i^T\right) = b_i b_i^T$$

In the above, we again used matrix calculus identities, and also the commutatitivity of the trace operator for square matrices. Putting these into the original equation, we get:

$$\begin{aligned}\nabla_S \ell &= \sum_{i=1}^{m} \big[\frac{1}{2}S^{-1} - \frac{1}{2}b_i b_i^T\big] \\ &= \frac{1}{2}\sum_{i=1}^{m} \big[\Sigma - b_i b_i^T\big]\end{aligned}$$

Setting this to zero gives the required maximum likelihood estimate for Σ.

4. **[10 points] Linear invariance of optimization algorithms**

Consider using an iterative optimization algorithm (such as Newton's method, or gradient descent) to minimize some continuously differentiable function $f(x)$. Suppose we initialize the algorithm at $x^{(0)} = \vec{0}$. When the algorithm is run, it will produce a value of $x \in \mathbb{R}^n$ for each iteration: $x^{(1)}, x^{(2)}, \ldots$.

Now, let some non-singular square matrix $A \in \mathbb{R}^{n\times n}$ be given, and define a new function $g(z) = f(Az)$. Consider using the same iterative optimization algorithm to optimize g (with initialization $z^{(0)} = \vec{0}$). If the values $z^{(1)}, z^{(2)}, \ldots$ produced by this method necessarily satisfy $z^{(i)} = A^{-1}x^{(i)}$ for all i, we say this optimization algorithm is **invariant to linear reparameterizations**.

(a) [7 points] Show that Newton's method (applied to find the minimum of a function) is invariant to linear reparameterizations. Note that since $z^{(0)} = \vec{0} = A^{-1}x^{(0)}$, it is sufficient to show that if Newton's method applied to $f(x)$ updates $x^{(i)}$ to $x^{(i+1)}$, then Newton's method applied to $g(z)$ will update $z^{(i)} = A^{-1}x^{(i)}$ to $z^{(i+1)} = A^{-1}x^{(i+1)}$.[3]

Answer: Let $g(z) = f(Az)$. We need to find $\nabla_z g(z)$ and its Hessian $\nabla_z^2 g(z)$.

By the chain rule:

$$\begin{aligned}\frac{\partial g(z)}{\partial z_i} &= \sum_{k=1}^{n} \frac{\partial f(Az)}{\partial (Az)_k}\frac{\partial (Az)_k}{\partial z_i} &(1)\\ &= \sum_{k=1}^{n} \frac{\partial f(Az)}{\partial (Az)_k} A_{ki} &(2)\\ &= \sum_{k=1}^{n} \frac{\partial f(Az)}{\partial x_k} A_{ki} &(3)\end{aligned}$$

Notice that the above is the same as :

$$\frac{\partial g(z)}{\partial z_i} = A_{\bullet i}^{\top}\nabla_x f(Az) \quad (4)$$

where $A_{\bullet i}$ is the i'th column of A. Then,

$$\nabla_z g(z) = A^{\top}\nabla_x f(Az) \quad (5)$$

[3] Note that for this problem, you must explicitly prove any matrix calculus identities that you wish to use that are not given in the lecture notes.

where $\nabla_x f(Az)$ is $\nabla_x f(\cdot)$ evaluated at Az.

Now we want to find the Hessian $\nabla_z^2 g(z)$.

$$\begin{aligned}\frac{\partial^2 g(z)}{\partial z_i \partial z_j} &= \frac{\partial}{\partial z_j} \sum_{k=1}^{n} \frac{\partial f(Az)}{\partial (Az)_k} A_{ki} &(6)\\ &= \sum_l \sum_k \frac{\partial^2 f(Az)}{\partial x_l \partial x_k} A_{ki} A_{lj} &(7)\end{aligned}$$

If we let $H_f(y)$ denote the Hessian of $f(\cdot)$ evaluated at some point y, and let $H_g(y)$ be the Hessian of $g(\cdot)$ evaluated at some point y, we have from the previous equation that:

$$H_g(z) = A^\top H_f(Az) A \tag{8}$$

We can now put this together and find the update rule for Newton's method on the function $f(Ax)$:

$$\begin{aligned}z^{(i+1)} &= z^{(i)} - H_g(z^{(i)})^{-1} \nabla_z g(z^{(i)}) &(9)\\ &= z^{(i)} - (A^\top H_f(Az^{(i)})A)^{-1} A^\top \nabla_x f(Az^{(i)}) &(10)\\ &= z^{(i)} - A^{-1} H_f(Az^{(i)})^{-1} (A^\top)^{-1} A^\top \nabla_x f(Az^{(i)}) &(11)\\ &= z^{(i)} - A^{-1} H_f(Az^{(i)})^{-1} \nabla_x f(Az^{(i)}) &(12)\end{aligned}$$

Now we have the update rule for $z^{(i+1)}$, we just need to verify that $z^{(i+1)} = A^{-1}x^{(i+1)}$ or equivalently that $Az^{(i+1)} = x^{(i+1)}$. From Eqn. (12) we have

$$\begin{aligned}Az^{(i+1)} &= A\left(z^{(i)} - A^{-1} H_f(Az^{(i)})^{-1} \nabla_x f(Az^{(i)})\right) &(13)\\ &= Az^{(i)} - H_f(Az^{(i)})^{-1} \nabla_x f(Az^{(i)}) &(14)\\ &= x^{(i)} - H_f(x^{(i)})^{-1} \nabla_x f(x^{(i)}) &(15)\\ &= x^{(i+1)}, &(16)\end{aligned}$$

where we used in order: Eqn. (12); rewriting terms; the inductive assumption $x^{(i)} = Az^{(i)}$; the update rule $x^{(i+1)} = x^{(i)} - H_f(x^{(i)})^{-1} \nabla_x f(x^{(i)})$.

(b) [3 points] Is gradient descent invariant to linear reparameterizations? Justify your answer.

Answer:

No. Using the notation from above, gradient descent on $g(z)$ results in the following update rule:

$$z^{(i+1)} = z^{(i)} - \alpha A^\top \nabla_x f(Az^{(i)}). \tag{17}$$

The update rule for $x^{(i+1)}$ is given by

$$x^{(i+1)} = x^{(i)} - \alpha \nabla_x f(x^{(i)}). \tag{18}$$

The invariance holds if and only if $x^{(i+1)} = Az^{(i+1)}$ given $x^{(i)} = Az^{(i)}$. However we have

$$\begin{aligned} Az^{(i+1)} &= Az^{(i)} - \alpha AA^\top \nabla_x f(Az^{(i)}) && (19) \\ &= x^{(i)} - \alpha AA^\top \nabla_x f(x^{(i)}). && (20) \end{aligned}$$

The two expressions in Eqn. (18) and Eqn. (20) are not necessarily equal ($AA^T = I$ requires that A be an orthogonal matrix), and thus gradient descent is not invariant to linear reparameterizations.

5. **[35 points] Regression for denoising quasar spectra**[4]

Introduction. In this problem, we will apply a supervised learning technique to estimate the light spectrum of *quasars*. Quasars are luminous distant galactic nuclei that are so bright, their light overwhelms that of stars in their galaxies. Understanding properties of the spectrum of light emitted by a quasar is useful for a number of tasks: first, a number of quasar properties can be estimated from the spectra, and second, properties of the regions of the universe through which the light passes can also be evaluated (for example, we can estimate the density of neutral and ionized particles in the universe, which helps cosmologists understand the evolution and fundamental laws governing its structure). The *light spectrum* is a curve that relates the light's intensity (formally, lumens per square meter), or *luminous flux*, to its wavelength. Figure 2 shows an example of a quasar light spectrum, where the wavelengths are measured in Angstroms (Å), where 1Å$= 10^{-10}$ meters.

The Lyman-α wavelength is a wavelength beyond which intervening particles at most negligibly interfere with light emitted from the quasar. (Interference generally occurs when a photon is absorbed by a neutral hydrogen atom, which only occurs for certain wavelengths of light.) For wavelengths greater than this Lyman-α wavelength, the observed light spectrum f_{obs} can be modeled as a smooth spectrum f plus noise:

$$f_{\text{obs}}(\lambda) = f(\lambda) + \text{noise}(\lambda)$$

For wavelengths below the Lyman-α wavelength, a region of the spectrum known as the Lyman-α forest, intervening matter causes attenuation of the observed signal. As light emitted by the quasar travels through regions of the universe richer in neutral hydrogen, some of it is absorbed, which we model as

$$f_{\text{obs}}(\lambda) = \text{absorption}(\lambda) \cdot f(\lambda) + \text{noise}(\lambda)$$

Astrophysicists and cosmologists wish to understand the absorption function, which gives information about the Lyman-α forest, and hence the distribution of neutral hydrogen in otherwise unreachable regions of the universe. This gives clues toward the formation and evolution of the universe. Thus, it is our goal to estimate the spectrum f of an observed quasar.

Getting the data. We will be using data generated from the Hubble Space Telescope Faint Object Spectrograph (HST-FOS), Spectra of Active Galactic Nuclei and Quasars.[5] We have provided two comma-separated data files located at:

- Training set: `http://cs229.stanford.edu/ps/ps1/quasar_train.csv`
- Test set: `http://cs229.stanford.edu/ps/ps1/quasar_test.csv`

[4] Ciollaro, Mattia, et al. "Functional regression for quasar spectra." arXiv:1404.3168 (2014).
[5] `https://hea-www.harvard.edu/FOSAGN/`

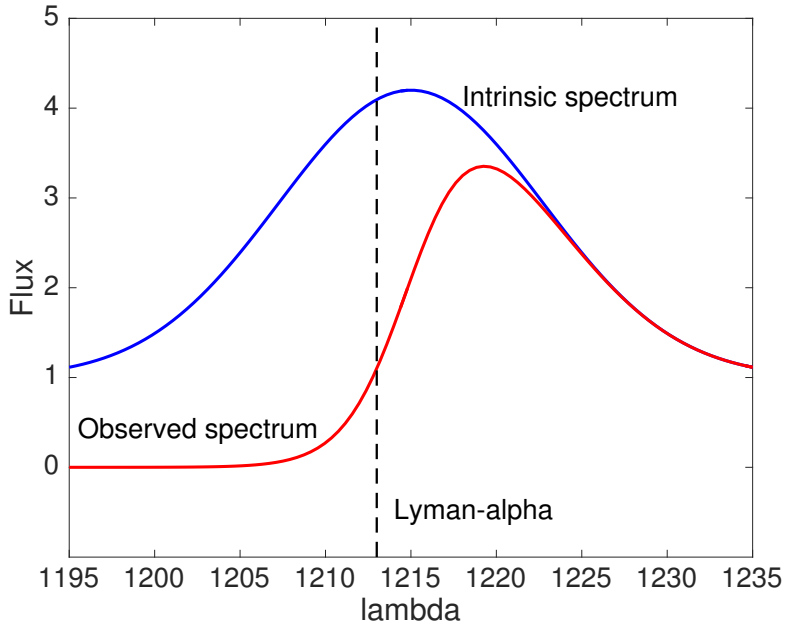


Figure 2: Light spectrum of a quasar. The blue line shows the intrinsic (i.e. original) flux spectrum emitted by the quasar. The red line denotes the observed spectrum here on Earth. To the left of the Lyman-α line, the observed flux is damped and the intrinsic (unabsorbed) flux continuum is not clearly recognizable (red line). To the right of the Lyman-α line, the observed flux approximates the intrinsic spectrum.

Each file contains a single header row containing 450 numbers corresponding integral wavelengths in the interval $[1150, 1600]$ Å. The remaining lines contain relative flux measurements for each wavelength. Specifically, `quasar_train.csv` contains 200 examples and `quasar_test.csv` contains 50 examples. You may use the helper file `load_quasar_data.m` to load the data in Matlab: `http://cs229.stanford.edu/ps/ps1/load_quasar_data.m`

(a) [10 points] Locally weighted linear regression

Consider a linear regression problem in which we want to "weight" different training examples differently. Specifically, suppose we want to minimize

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m} w^{(i)} \left(\theta^T x^{(i)} - y^{(i)}\right)^2$$

In class, we worked out what happens for the case where all the weights (the $w^{(i)}$'s) are the same. In this problem, we will generalize some of those ideas to the weighted setting.

i. [2 points] Show that $J(\theta)$ can also be written

$$J(\theta) = (X\theta - \vec{y})^T W (X\theta - \vec{y})$$

for an appropriate diagonal matrix W, and where X and $\vec{y}$ are as defined in class. State clearly what W is.

Answer: Let $W_{ii} = \frac{1}{2}w^{(i)}, W_{ij} = 0$ for $i \neq j$, let $\vec{z} = X\theta - \vec{y}$, i.e. $z_i = \theta^T x^{(i)} - y^{(i)}$. Then we have:

$$\begin{aligned}(X\theta - \vec{y})^T W(X\theta - \vec{y}) &= \vec{z}^T W\vec{z} \\ &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)} z_i^2 \\ &= \frac{1}{2}\sum_{i=1}^{m} w^{(i)}(\theta^T x^{(i)} - y^{(i)})^2 \\ &= J(\theta)\end{aligned}$$

ii. [4 points] If all the $w^{(i)}$'s equal 1, then we saw in class that the normal equation is

$$X^T X\theta = X^T\vec{y},$$

and that the value of θ that minimizes $J(\theta)$ is given by $(X^T X)^{-1}X^T\vec{y}$. By finding the derivative $\nabla_\theta J(\theta)$ and setting that to zero, generalize the normal equation to this weighted setting, and give the new value of θ that minimizes $J(\theta)$ in closed form as a function of X, W and $\vec{y}$.
Answer:

$$\nabla_\theta J(\theta) = \nabla_\theta(\theta^T X^T W X\theta + \vec{y}^T W\vec{y} - 2\vec{y}^T W X\theta) = 2(X^T W X\theta - X^T W\vec{y}),$$

so we have $\nabla_\theta J(\theta) = 0$ if and only if

$$X^T W X\theta = X^T W\vec{y}$$

These are the normal equations, from which we can get a closed form formula for θ.

$$\theta = (X^T W X)^{-1}X^T W\vec{y}$$

iii. [4 points] Suppose we have a training set $\{(x^{(i)}, y^{(i)}); i = 1\ldots, m\}$ of m independent examples, but in which the $y^{(i)}$'s were observed with differing variances. Specifically, suppose that

$$p(y^{(i)}|x^{(i)};\theta) = \frac{1}{\sqrt{2\pi}\sigma^{(i)}}\exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2}\right)$$

I.e., $y^{(i)}$ has mean $\theta^T x^{(i)}$ and variance $(\sigma^{(i)})^2$ (where the $\sigma^{(i)}$'s are fixed, known, constants). Show that finding the maximum likelihood estimate of θ reduces to solving a weighted linear regression problem. State clearly what the $w^{(i)}$'s are in terms of the $\sigma^{(i)}$'s.

Answer:

$$\begin{aligned}
\arg\max_\theta \prod_{i=1}^m p(y^{(i)}|x^{(i)};\theta) &= \arg\max_\theta \sum_{i=1}^m \log p(y^{(i)}|x^{(i)};\theta) \\
&= \arg\max_\theta \sum_{i=1}^m \left(\log \frac{1}{\sqrt{2\pi}\sigma^{(i)}} - \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2} \right) \\
&= \arg\max_\theta - \sum_{i=1}^m \frac{(y^{(i)} - \theta^T x^{(i)})^2}{2(\sigma^{(i)})^2} \\
&= \arg\min_\theta \frac{1}{2} \sum_{i=1}^m \frac{1}{(\sigma^{(i)})^2} (y^{(i)} - \theta^T x^{(i)})^2 \\
&= \arg\min_\theta \frac{1}{2} \sum_{i=1}^m w^{(i)} (y^{(i)} - \theta^T x^{(i)})^2
\end{aligned}$$

where in the last step, we substituted: $w^{(i)} = \frac{1}{(\sigma^{(i)})^2}$ to get the linear regression form.

(b) [6 points] Visualizing the data

i. [2 points] Use the normal equations to implement (unweighted) linear regression ($y = \theta^T x$) on the *first* training example (i.e. first non-header row). On one figure, plot both the raw data and the straight line resulting from your fit. State the optimal θ resulting from the linear regression.
Answer: $\theta^* = (2.5134, -0.0010)$

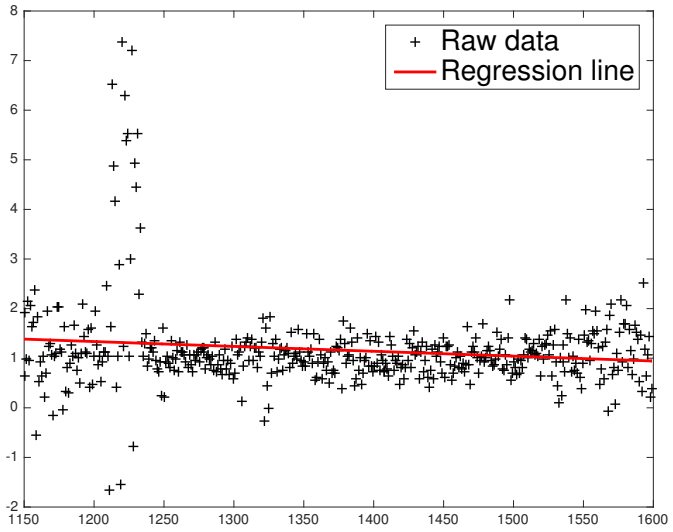


Figure 3: Unweighted linear regression

ii. [2 points] Implement locally weighted linear regression on the *first* training example. Use the normal equations you derived in part (a)(ii). On a different figure, plot both the raw data and the smooth curve resulting from your fit. When evaluating $h(\cdot)$ at a

query point x, use weights

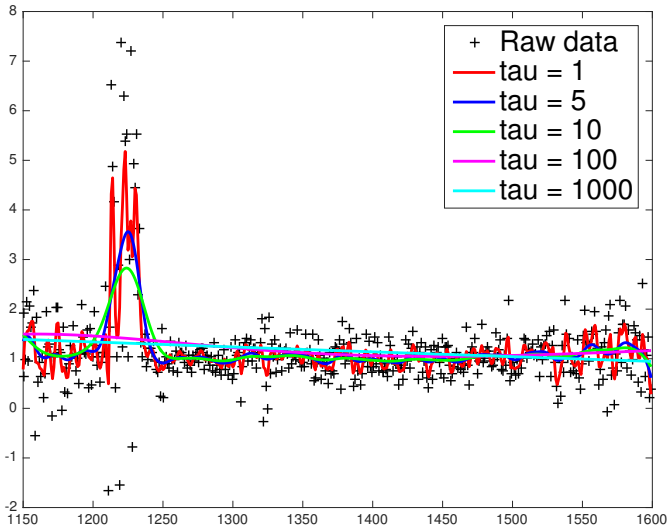
$$w^{(i)} = \exp\left(-\frac{(x - x^{(i)})^2}{2\tau^2}\right),$$

with bandwidth parameter $\tau = 5$.

Answer: See figure below for $\tau = 5$.

iii. [2 points] Repeat (b)(ii) four more times with $\tau = 1, 10, 100$ and 1000. Plot the resulting curves. You can submit one plot with all four τ values or submit four separate plots. If you submit one plot, make sure all curves are visible. Additionally, in **2-3 sentences**, comment on what happens to the locally weighted linear regression line as τ varies.

Answer: As τ becomes large, locally weighted linear regression becomes unweighted linear regression. As τ becomes small, it begins to overfit the data.



(c) [19 points] Predicting quasar spectra with functional regression

We now go a step beyond what we have covered explicitly in class, and we wish to predict an entire part of a spectrum—a curve—from noisy observed data. We begin by supposing that we observe a random sample of m absorption-free spectra, which is possible for quasars very close (in a sense relative to the size of the universe!) to Earth. For a given spectrum f, define f_{right} to be the spectrum to the right of the Lyman-α line. Let f_{left} be the spectrum within the Lyman-α forest region, that is, for lower wavelengths. To make the results cleaner, we define:

$$f(\lambda) = \begin{cases} f_{\text{left}}(\lambda) & \text{if } \lambda < 1200 \\ f_{\text{right}}(\lambda) & \text{if } \lambda \geq 1300 \end{cases}$$

We will learn a function r (for regression) that maps an observed f_{right} to an unobserved target f_{left}. This is useful in practice because we observe f_{right} with *only* random noise: there is no systematic absorption, which we cannot observe directly, because hydrogen does not absorb photons with higher wavelengths. By predicting f_{left} from a noisy version of f_{right}, we can estimate the unobservable spectrum of a quasar as well as the absorption function. Imaging systems collect data of the form

$$f_{\text{obs}}(\lambda) = \text{absorption}(\lambda) \cdot f(\lambda) + \text{noise}(\lambda)$$

for $\lambda \in \{\lambda_1, \ldots, \lambda_n\}$, a *finite* number of points λ, because they must quantize the information. That is, even in the quasars-close-to-Earth training data, our observations of f_{left} and f_{right} consist of noisy evaluations of the true spectrum f at multiple wavelengths. In our case, we have $n = 450$ and $\lambda_1 = 1150, \ldots, \lambda_n = 1599$.

We formulate the functional regression task as the goal of learning the function r mapping f_{left} to f_{right}:

$$r(f_{\text{right}})(\lambda) = \mathbb{E}(f_{\text{left}} \mid f_{\text{right}})(\lambda)$$

for λ in the Lyman-α forest.

i. [1 points] First, we must smooth the data in the training dataset to make it more useful for prediction. For each $i = 1, \ldots, m$, define $f^{(i)}(\lambda)$ to be the weighted linear regression estimate the i^{th} spectrum. Use your code from part (b)(ii) above to smooth all spectra in the training set using $\tau = 5$. Do the same for the test set. We will now operate on these smoothed spectra.

ii. [14 points] Using your estimated regression functions $f^{(i)}$ for $i = 1, \ldots, m$, we now wish to estimate the unobserved spectrum f_{left} of a quasar from its (noisy) observed spectrum f_{right}. To do so, we perform a weighted regression of the *locally weighted regressions*. In particular, given a new noisy spectrum observation:

$$f_{\text{obs}}(\lambda) = f(\lambda) + \text{noise}(\lambda) \quad \text{for } \lambda \in \{1300, \ldots, 1599\}.$$

We define a metric d which takes as input, two spectra f_1 and f_2, and outputs a scalar:

$$d(f_1, f_2) = \sum_i \Big(f_1(\lambda_i) - f_2(\lambda_i)\Big)^2.$$

The metric d computes squared distance between the new datapoint and previous datapoints. If f_1 and f_2 are right spectra, then we take the preceding sum only over $\lambda \in \{1300, \ldots, 1599\}$, rather than the entire spectrum.

Based on this distance function, we may define the nonparametric *functional* regression estimator, which is a locally weighted sum of *functions* f_{left} from the training data (this is like locally weighted linear regression, except that instead of predicting $y \in \mathbb{R}$ we predict a function f_{left}). Specifically, let f_{right} denote the right side of a spectrum, which we have smoothed using locally weighted linear regression (as you were told to do in the previous part of the problem). We wish to estimate the associated *left* spectrum f_{left}. Define the function $\ker(t) = \max\{1 - t, 0\}$ and let $\mathsf{neighb}_k(f_{\text{right}})$ denote the k indices $i \in \{1, 2, \ldots, m\}$ that are closest to f_{right}, that is

$$d(f^{(i)}_{\text{right}}, f_{\text{right}}) < d(f^{(j)}_{\text{right}}, f_{\text{right}}) \quad \text{for all } i \in \mathsf{neighb}_k(f_{\text{right}}),\ j \notin \mathsf{neighb}_k(f_{\text{right}})$$

and $\mathsf{neighb}_k(f_{\text{right}})$ contains exactly k indices. In addition, let

$$h := \max_{i \in \{1, \ldots, m\}} d(f^{(i)}_{\text{right}}, f_{\text{right}}).$$

Then define the estimated function $\widehat{f_{\text{left}}} : \mathbb{R} \to \mathbb{R}$ by

$$\widehat{f_{\text{left}}}(\lambda) = \frac{\sum_{i \in \mathsf{neighb}_k(f_{\text{right}})} \ker(d(f^{(i)}_{\text{right}}, f_{\text{right}})/h) f^{(i)}_{\text{left}}(\lambda)}{\sum_{i \in \mathsf{neighb}_k(f_{\text{right}})} \ker(d(f^{(i)}_{\text{right}}, f_{\text{right}})/h)}. \tag{21}$$

Recall that $f_{\text{right}}^{(i)}$ is the *smoothed* (weighted linear regression) estimate of the ith training spectrum.
Construct the functional regression estimate (21) for each spectrum in the entire training set using $k = 3$ nearest neighbors: for each $j = 1, \ldots, m$, construct the estimator $\widehat{f_{\text{left}}}$ from (21) using $f_{\text{right}} = f_{\text{right}}^{(j)}$. Then compute the error $d(f_{\text{left}}^{(j)}, \widehat{f_{\text{left}}})$ between the true spectrum $f_{\text{left}}^{(j)}$ and your estimated spectrum $\widehat{f_{\text{left}}}$ for each j, and return the average over the training data. What is your average training error?
Answer: We achieve a training set error of 1.0664.

iii. [4 points] Perform functional regression on the test set using the same procedure as in the previous subquestion. What is your average test error? For test examples 1 and 6, include a plot with both the entire smooth spectrum and the fitted curve $\widehat{f_{\text{left}}}$ curve on the same graph. You should submit two plots: one for test example 1 and one for test example 6.
Answer: We achieve a test set error of 2.7100.

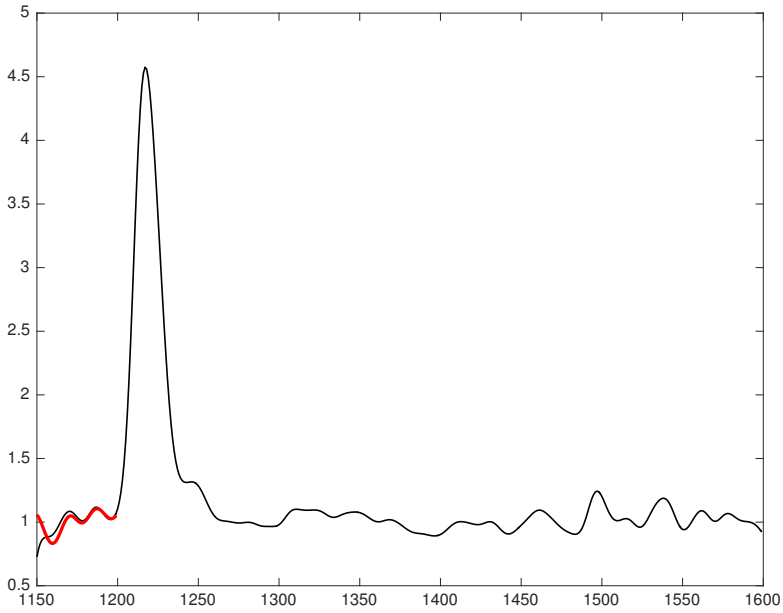


Figure 4: Resulting functional regression for test set example 1.

Code for the entire problem is included below
Answer:

```
function yhat = local_linear_regression(x, y, tau)
% LOCAL_LINEAR_REGRESSION Performs a local linear regression to smooth the
% given input signal.
%
% yhat = local_linear_regression(x, y, tau) takes as input the vectors x
% and y, both of the same dimension. Then, at each point x in the given
% vector, fits a local linear regression using the features (1, x) at that
```

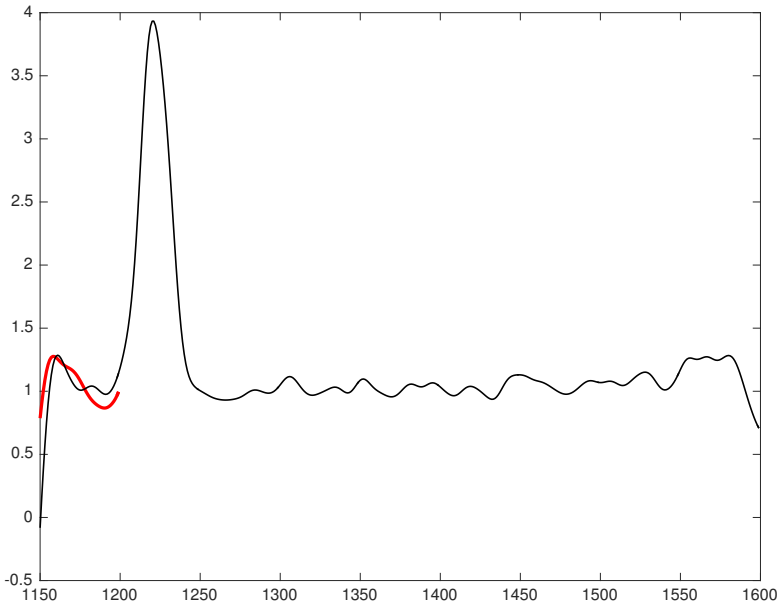


Figure 5: Resulting functional regression for test set example 6.

```
% point, with weights given by
%
% w^i(x) = exp(-(x - x^i)^2 / (2 * tau^2)),
%
% that is, transforms the input so that
%
% yhat(i) = [1, x(i)] * theta^(i)
%
% where theta^(i) minimizes
%
% J_i(theta) = sum_{j=1}^m w^j(x(i)) * (y(j) - [1 x(j)] * theta)^2.

if (length(x) ~= length(y))
  error('Length of x (%d) not same as y (%d)', length(x), length(y));
end
nn = length(x);

X = [ones(nn, 1), x];
yhat = zeros(nn, 1);

for ii = 1:nn
  w = exp(-(x - x(ii)).^2 / (2 * tau^2));
  XwX = X' * ([w, w] .* X);
  XtWy = X' * (w .* y);
  theta = XwX \ XtWy;
  yhat(ii) = [1 x(ii)] * theta;
```

```
end

% Full quasar solution, including plots

load_quasar_data;

[mm, nn] = size(train_qso);

mtest = size(test_qso, 1);

%% Part (a)i: Linear regression

y = train_qso(1, :)';
X = [ones(nn, 1), lambdas];
theta = X\y;  % Solves linear regression directly

figure;
h = plot(lambdas, train_qso(1, :), 'k+');
set(h, 'linewidth', 1);
hold on;
h = plot(lambdas, theta(1) + lambdas * theta(2), 'r-');
set(h, 'linewidth', 2);
h = legend('Raw data', 'Regression line');
set(h, 'fontsize', 20);

print -depsc2 quasar_linear_regression.eps;

%% Part (a)ii/iii: Smoothing the quasars with LWLR

figure;
X = [ones(nn, 1), lambdas];
y = train_qso(1, :)';
h = plot(lambdas, y, 'k+');
set(h, 'linewidth', 1);
hold on;
colors = {'r-', 'b-', 'g-', 'm-', 'c-'};
taus = [1, 5, 10, 100, 1000];
for tau_ind = 1:5
  tau = taus(tau_ind);
  y_smooth = local_linear_regression(lambdas, y, tau);
  h = plot(lambdas, y_smooth, char(colors(tau_ind)));
  set(h, 'linewidth', 2);
end

h = legend('Raw data', 'tau = 1', 'tau = 5', 'tau = 10', ...
           'tau = 100', 'tau = 1000');
set(h, 'fontsize', 20);

print -depsc2 sprintf('quasar_locally_taus.eps', tau);
```

```
%% Part (b)i: Smooth all quasars with LWLR

train_smooth = train_qso;
test_smooth = test_qso;
tau = 5;

X = [ones(nn, 1), lambdas];

for jj = 1:mm
  ytrain = train_qso(jj, :)';
  train_smooth(jj, :) = local_linear_regression(lambdas, ytrain, tau)';
end

for jj = 1:mtest
  ytest = test_qso(jj, :)';
  test_smooth(jj, :) = local_linear_regression(lambdas, ytest, tau)';
end

%% Find the right-most function parts
right_trains = train_smooth(:, 151:end);
left_trains = train_smooth(:, 1:50);
right_tests = test_smooth(:, 151:end);
left_tests = test_smooth(:, 1:50);

%% Construct matrix of all pairs of distances between training quasar spectra
train_dists = zeros(mm, mm);
for ii = 1:mm
  for jj = (ii + 1):mm
    train_dists(ii, jj) = norm(right_trains(ii, :) - right_trains(jj, :))^2;
  end
end
train_dists = train_dists + train_dists';
train_dists = train_dists / max(train_dists(:));

%% Reconstruct training curves

f_left_estimates = zeros(mm, 50);
num_nearest = 3;

for ii = 1:mm
  [train_dist_sort, inds] = sort(train_dists(:, ii), 1, 'ascend');
  close_inds = ones(mm, 1);
  close_inds(inds((num_nearest + 1):end)) = 0;
  h = max(train_dists(:, ii));
  kerns = max(1 - train_dists(:, ii) / h, 0); % An m-by-1 vector
  kerns = kerns .* close_inds;
  f_left_estimates(ii, :) = left_trains' * kerns / sum(kerns);
end
```

```
% Compute error rate of estimates
err = sum((left_trains(:) - f_left_estimates(:)).^2);
err = err / mm;
fprintf(1, 'Average training error: %1.4f\n', err);

%% Reconstruct test curves

% Construct matrix of all pairs of distances between training and testing
% quasar spectra

train_to_test_dists = zeros(mm, mtest);
for ii = 1:mm
  for jj = 1:mtest
    train_to_test_dists(ii, jj) = ...
        norm(right_trains(ii, :) - right_tests(jj, :))^2;
  end
end
train_to_test_dists = train_to_test_dists / max(train_to_test_dists(:));

f_left_estimates = zeros(mtest, 50);
for ii = 1:mtest
  [tttd_sorted, inds] = sort(train_to_test_dists(:, ii), 1, 'ascend');
  close_inds = ones(mm, 1);
  close_inds(inds((num_nearest + 1):end)) = 0;
  h = max(train_to_test_dists(:, ii));
  kerns = max(1 - train_to_test_dists(:, ii) / h, 0);
  kerns = kerns .* close_inds;
  f_left_estimates(ii, :) = left_trains' * kerns / sum(kerns);
end

%% Compute error rate of estimates
err = sum((left_tests(:) - f_left_estimates(:)).^2);
err = err / mtest;
fprintf(1, 'Average testing error: %1.4f\n', err);

%% Final plots
figure;
plot(lambdas, test_smooth(1, :), 'k-', 'linewidth', 1);
hold on;
plot(lambdas(1:50), f_left_estimates(1, :), 'r-', 'linewidth', 2);
print -depsc2 'quasar_test_1.eps';

figure;
plot(lambdas(1:50), f_left_estimates(6, :), 'r-', 'linewidth', 2)
hold on;
plot(lambdas, test_smooth(6, :), 'k-', 'linewidth', 1);
print -depsc2 'quasar_test_6.eps';
```

Reminder: Please include in your submission a printout of your code and figures for the programming questions.

CS 229, Autumn 2016
Problem Set #2: Naive Bayes, SVMs, and Theory

Due Wednesday, November 2 at 11:00 am on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[15 points] Constructing kernels**

   In class, we saw that by choosing a kernel $K(x, z) = \phi(x)^T \phi(z)$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding K.

   However in this question we are interested in direct construction of kernels. I.e., suppose we have a function $K(x, z)$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging K into the SVM as the kernel function. However for $K(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $K(x, z)$ is a (Mercer) kernel if and only if for any finite set $\{x^{(1)}, \ldots, x^{(m)}\}$, the matrix K is symmetric and positive semidefinite, where the square matrix $K \in \mathbb{R}^{m \times m}$ is given by $K_{ij} = K(x^{(i)}, x^{(j)})$.

   Now here comes the question: Let K_1, K_2 be kernels over $\mathbb{R}^n \times \mathbb{R}^n$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^n \to \mathbb{R}^d$ be a function mapping from $\mathbb{R}^n$ to $\mathbb{R}^d$, let K_3 be a kernel over $\mathbb{R}^d \times \mathbb{R}^d$, and let $p(x)$ a polynomial over x with *positive* coefficients.

   For each of the functions K below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

   (a) [1 points] $K(x, z) = K_1(x, z) + K_2(x, z)$

   (b) [1 points] $K(x, z) = K_1(x, z) - K_2(x, z)$

   (c) [1 points] $K(x, z) = aK_1(x, z)$

   (d) [1 points] $K(x, z) = -aK_1(x, z)$

   (e) [5 points] $K(x, z) = K_1(x, z)K_2(x, z)$

   (f) [2 points] $K(x, z) = f(x)f(z)$

   (g) [2 points] $K(x, z) = K_3(\phi(x), \phi(z))$

   (h) [2 points] $K(x, z) = p(K_1(x, z))$

[Hint: For part (e), the answer is that the K there *is* indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

2. **[10 points] Kernelizing the Perceptron**

   Let there be a binary classification problem with $y \in \{-1, 1\}$. The perceptron uses hypotheses of the form $h_\theta(x) = g(\theta^T x)$, where $g(z) = \text{sign}(z) = 1$ if $z \geq 0$, -1 otherwise. In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

   $$\theta^{(i+1)} := \begin{cases} \theta^{(i)} + \alpha y^{(i+1)} x^{(i+1)} & \text{if } h_{\theta^{(i)}}(x^{(i+1)}) y^{(i+1)} < 0 \\ \theta^{(i)} & \text{otherwise,} \end{cases}$$

   where $\theta^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to $\vec{0}$.

   Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, ∞-dimensional) that it's infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(x)$. [Note: You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(x) = 1$ so that this is taken care of.] Your description should specify

   (a) How you will (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$, including how the initial value $\theta^{(0)} = \vec{0}$ is represented (note that $\theta^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(x)$);

   (b) How you will efficiently make a prediction on a new input $x^{(i+1)}$. I.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = g({\theta^{(i)}}^T \phi(x^{(i+1)}))$, using your representation of $\theta^{(i)}$; and

   (c) How you will modify the update rule given above to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$; i.e., using the update rule corresponding to the feature mapping ϕ:

   $$\theta^{(i+1)} := \theta^{(i)} + \alpha \mathbf{1}\{{\theta^{(i)}}^T \phi(x^{(i+1)}) y^{(i+1)} < 0\} y^{(i+1)} \phi(x^{(i+1)}).$$

   [Hint: our discussion of the representer theorem may be useful.]

3. **[30 points] Spam classification**

   In this problem, we will use the naive Bayes algorithm and an SVM to build a spam classifier.

   In recent years, spam on electronic newsgroups has been an increasing problem. Here, we'll build a classifier to distinguish between "real" newsgroup messages, and spam messages. For this experiment, we obtained a set of spam emails, and a set of genuine newsgroup messages.[1] Using only the subject line and body of each message, we'll learn to distinguish between the spam and non-spam.

[1] Thanks to Christian Shelton for providing the spam email. The non-spam messages are from the 20 newsgroups data at `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html`.

All the files for the problem are in `http://cs229.stanford.edu/materials/spam_data.tgz`. **Note: Please do not circulate this data outside this class.** In order to get the text emails into a form usable by naive Bayes, we've already done some preprocessing on the messages. You can look at two sample spam emails in the files `spam_sample_original*`, and their preprocessed forms in the files `spam_sample_preprocessed*`. The first line in the preprocessed format is just the label and is not part of the message. The preprocessing ensures that only the message body and subject remain in the dataset; email addresses (EMAILADDR), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens" rather than "words," since some of the features will correspond to special values like EMAILADDR. You don't have to worry about the distinction.) The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam mail.

The work to extract feature vectors out of the documents has also been done for you, so you can just load in the design matrices (called document-word matrices in text classification) containing all the data. In a document-word matrix, the i^{th} row represents the i^{th} document/email, and the j^{th} column represents the j^{th} distinct token. Thus, the (i, j)-entry of this matrix represents the number of occurrences of the j^{th} token in the i^{th} document.

For this problem, we've chosen as our set of tokens considered (that is, as our vocabulary) only the medium frequency tokens. The intuition is that tokens that occur too often or too rarely do not have much classification value. (Examples tokens that occur very often are words like "the," "and," and "of," which occur in so many emails and are sufficiently content-free that they aren't worth modeling.) Also, words were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same word. For a list of the tokens used, see the file `TOKENS_LIST`.

Since the document-word matrix is extremely sparse (has lots of zero entries), we have stored it in our own efficient format to save space. You don't have to worry about this format.[2] The file `readMatrix.m` provides the `readMatrix` function that reads in the document-word matrix and the correct class labels for the various documents. Code in `nb_train.m` and `nb_test.m` shows how `readMatrix` should be called. The documentation at the top of these two files will tell you all you need to know about the setup.

(a) [11 points] Implement a naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing.

You should use the code outline provided in `nb_train.m` to train your parameters, and then use these parameters to classify the test set data by filling in the code in `nb_test.m`. You may assume that any parameters computed in `nb_train.m` are in memory when `nb_test.m` is executed, and do not need to be recomputed (i.e., that `nb_test.m` is executed immediately after `nb_train.m`) [3].

Train your parameters using the document-word matrix in `MATRIX.TRAIN`, and then report the test set error on `MATRIX.TEST`.

Remark. If you implement naive Bayes the straightforward way, you'll find that the computed $p(x|y) = \prod_i p(x_i|y)$ often equals zero. This is because $p(x|y)$, which is

[2] Unless you're not using Matlab/Octave, in which case feel free to ask us about it. We have provided Julia code to read the file in `MatrixReading.jl`.

[3] Matlab note: If a .m file doesn't begin with a function declaration, the file is a script. Variables in a script are put into the global namespace, unlike with functions.

the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow.") You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(x|y)$. [Hint: Think about using logarithms.]

(b) [3 points] Intuitively, some tokens may be particularly indicative of an email being in a particular class. We can try to get an informal sense of how indicative token i is for the SPAM class by looking at:

$$\log \frac{p(x_j = i|y = 1)}{p(x_j = i|y = 0)} = \log \left(\frac{P(\text{token } i|\text{email is SPAM})}{P(\text{token } i|\text{email is NOTSPAM})} \right).$$

Using the parameters fit in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., have the highest positive value on the measure above). The numbered list of tokens in the file `TOKENS_LIST` should be useful for identifying the words/tokens.

(c) [3 points] Repeat part (a), but with training sets of size ranging from 50, 100, 200, ..., up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error vs. training set size). You may need to change the call to `readMatrix` in `nb_train.m` to read the correct file each time. Which training-set size gives the best test set error?

(d) [11 points] Train an SVM on this dataset using stochastic gradient descent and the radial basis function (also known as the Gaussian) kernel, which sets

$$K(x, z) = \exp \left(-\frac{1}{2\tau^2} \|x - z\|_2^2 \right).$$

In this case, recall that (as proved in class) the objective with kernel matrix $K = [K^{(1)} \ \cdots \ K^{(m)}] \in \mathbb{R}^{m \times m}$ is given by

$$J(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \left[1 - y^{(i)} {K^{(i)}}^T \alpha \right]_+ + \frac{\lambda}{2} \alpha^T K \alpha$$

where $[t]_+ = \max\{t, 0\}$ is the positive part function. In this case, the gradient (actually, this is known as a *subgradient*) of the individual loss terms is

$$\nabla_\alpha \left[1 - y^{(i)} K^{(i)} \alpha \right]_+ = \begin{cases} -y^{(i)} K^{(i)} & \text{if } y^{(i)} {K^{(i)}}^T \alpha < 1 \\ 0 & \text{otherwise.} \end{cases}$$

In your SVM training, you should perform stochastic gradient descent, where in each iteration you choose an index $i \in \{1, \ldots, m\}$ uniformly at random, for a total of $40 \cdot m$ steps, where m is the training set size, and your kernel should use $\tau = 8$ and regularization multiplier $\lambda = \frac{1}{64m}$. For this part of the problem, you should also replace each training or test point $x^{(i)}$ with a zero-one vector $z^{(i)}$, where $z_j^{(i)} = 1$ if $x_j^{(i)} > 0$ and $z_j^{(i)} = 0$ if $x_j^{(i)} = 0$. Initialize your SGD procedure at $\alpha = 0$.
The output of your training code, which you should implement in `svm_test.m`, should be the α vector that is the *average* of all the α vectors that your iteration updates. At iteration t of stochastic gradient descent you should use stepsize $1/\sqrt{t}$.

Similar to part (c), train an SVM with training set sizes 50, 100, 200, ..., 1400, by using the file `MATRIX.TRAIN.50` and so on. Plot the test error each time, using `MATRIX.TEST` as the test data.

(A few hints for more efficient Matlab code: you should try to vectorize creation of the Kernel matrix, and you should call the method `full` to make the matrix non-sparse, which will make the method faster. In addition, the training data uses labels in $\{0, 1\}$, so you should change the output of the `readMatrix` method to have labels $y \in \{-1, 1\}$.)

(e) [2 points] How do naive Bayes and Support Vector Machines compare (in terms of generalization error) as a function of the training set size?

4. **[20 points] Properties of VC dimension**

In this problem, we investigate a few properties of the Vapnik-Chervonenkis dimension, mostly relating to how $\mathrm{VC}(H)$ increases as the set H increases. For each part of this problem, you should state whether the given statement is true, and justify your answer with either a formal proof or a counter-example.

(a) Let two hypothesis classes H_1 and H_2 satisfy $H_1 \subseteq H_2$. Prove or disprove: $\mathrm{VC}(H_1) \leq \mathrm{VC}(H_2)$.

(b) Let $H_1 = H_2 \cup \{h_1, \ldots, h_k\}$. (I.e., H_1 is the union of H_2 and some set of k additional hypotheses.) Prove or disprove: $\mathrm{VC}(H_1) \leq \mathrm{VC}(H_2) + k$. [Hint: You might want to start by considering the case of $k = 1$.]

(c) Let $H_1 = H_2 \cup H_3$. Prove or disprove: $\mathrm{VC}(H_1) \leq \mathrm{VC}(H_2) + \mathrm{VC}(H_3)$.

5. **[20 points] Training and testing on different distributions**

In the discussion in class about learning theory, a key assumption was that we trained and tested our learning algorithms on the same distribution $\mathcal{D}$. In this problem, we'll investigate one special case of training and testing on different distributions. Specifically, we will consider what happens when the training labels are *noisy*, but the test labels are not.

Consider a binary classification problem with labels $y \in \{0, 1\}$, and let $\mathcal{D}$ be a distribution over (x, y), that we'll think of as the original, "clean" or "uncorrupted" distribution. Define $\mathcal{D}_\tau$ to be a "corrupted" distribution over (x, y) which is the same as $\mathcal{D}$, except that the labels y have some probability $0 \leq \tau < 0.5$ of being flipped. Thus, to sample from $\mathcal{D}_\tau$, we would first sample (x, y) from $\mathcal{D}$, and then with probability τ (independently of the observed x and y) replace y with $1 - y$. Note that $\mathcal{D}_0 = \mathcal{D}$.

The distribution $\mathcal{D}_\tau$ models a setting in which an unreliable human (or other source) is labeling your training data for you, and on each example he/she has a probability τ of mislabeling it. Even though our training data is corrupted, we are still interested in evaluating our hypotheses with respect to the original, uncorrupted distribution $\mathcal{D}$.

We define the generalization error *with respect to* $\mathcal{D}_\tau$ to be

$$\varepsilon_\tau(h) = P_{(x,y)\sim\mathcal{D}_\tau}[h(x) \neq y].$$

Note that $\varepsilon_0(h)$ is the generalization error with respect to the "clean" distribution; it is with respect to ε_0 that we wish to evaluate our hypotheses.

(a) For any hypothesis h, the quantity $\varepsilon_0(h)$ can be calculated as a function of $\varepsilon_\tau(h)$ and τ. Write down a formula for $\varepsilon_0(h)$ in terms of $\varepsilon_\tau(h)$ and τ, and justify your answer.

(b) Let $|H|$ be finite, and suppose our training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ is obtained by drawing m examples IID from the corrupted distribution $\mathcal{D}_\tau$. Suppose we pick $h \in H$ using empirical risk minimization: $\hat{h} = \arg\min_{h \in H} \hat{\varepsilon}_S(h)$. Also, let $h^* = \arg\min_{h \in H} \varepsilon_0(h)$.

Let any $\delta, \gamma > 0$ be given. Prove that for

$$\varepsilon_0(\hat{h}) \leq \varepsilon_0(h^*) + 2\gamma$$

to hold with probability $1 - \delta$, it suffices that

$$m \geq \frac{1}{2(1-2\tau)^2\gamma^2} \log \frac{2|H|}{\delta}.$$

Remark. This result suggests that, roughly, m examples that have been corrupted at noise level τ are worth about as much as $(1-2\tau)^2 m$ uncorrupted training examples. This is a useful rule-of-thumb to know if you ever need to decide whether/how much to pay for a more reliable source of training data. (If you've taken a class in information theory, you may also have heard that $(1-\mathcal{H}(\tau))m$ is a good estimate of the information in the m corrupted examples, where $\mathcal{H}(\tau) = -(\tau \log_2 \tau + (1-\tau)\log_2(1-\tau))$ is the "binary entropy" function. And indeed, the functions $(1-2\tau)^2$ and $1-\mathcal{H}(\tau)$ are quite close to each other.)

(c) Comment **briefly** on what happens as τ approaches 0.5.

6. **[19 points] Boosting and high energy physics**

In class, we discussed boosting algorithms and decision stumps. In this problem, we explore applications of these ideas to detect particle emissions in a high-energy particle accelerator. In high energy physics, such as at the Large Hadron Collider (LHC), one accelerates small particles to relativistic speeds and smashes them into one another, tracking the emitted particles. The goal in these problems is to detect the emission of certain interesting particles based on other observed particles and energies.[4] In this problem, we explore the application of boosting to a high energy physics problem, where we use decision stumps applied to 18 low- and high-level physics-based features. All data for the problem is available at `http://cs229.stanford.edu/materials/boost_data.tgz`.

For the first part of the problem, we explore how decision stumps based on thresholding can provide a weak-learning guarantee. In particular, we show that for real-valued attributes x, there is an edge $\gamma > 0$ that decision stumps guarantee. Recall that thresholding-based decision stumps are functions indexed by a threshold s and sign $+/-$, such that

$$\phi_{s,+}(x) = \begin{cases} 1 & \text{if } x \geq s \\ -1 & \text{if } x < s \end{cases} \quad \text{and} \quad \phi_{s,-}(x) = \begin{cases} -1 & \text{if } x \geq s \\ 1 & \text{if } x < s. \end{cases}$$

That is, $\phi_{s,+}(x) = -\phi_{s,-}(x)$. We assume for simplicity in the theoretical parts of this exercise that our input attribute vectors $x \in \mathbb{R}$, that is, they are one-dimensional. Now, we would like guarantee that there is some $\gamma > 0$ and a threshold s such that, for *any*

[4] For more, see the following paper: Baldi, Sadowski, Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Communications* 5, Article 4308. `http://arxiv.org/abs/1402.4735`.

distribution p on the training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ (where $y^{(i)} \in \{-1, +1\}$ and $x^{(i)} \in \mathbb{R}$, and we recall that p is a distribution on the training set if $\sum_{i=1}^m p_i = 1$ and $p_i \geq 0$ for each i) we have

$$\sum_{i=1}^m p_i \mathbf{1}\left\{y^{(i)} \neq \phi_{s,+}(x^{(i)})\right\} \leq \frac{1}{2} - \gamma \quad \text{or} \quad \sum_{i=1}^m p_i \mathbf{1}\left\{y^{(i)} \neq \phi_{s,-}(x^{(i)})\right\} \leq \frac{1}{2} - \gamma.$$

For simplicity, we assume that all of the $x^{(i)}$ are *distinct*, so that none of them are equal. We also assume (without loss of generality, but this makes the problem notationally simpler) that

$$x^{(1)} > x^{(2)} > \cdots > x^{(m)}.$$

(a) [3 points] Show that for each threshold s, there is some $m_0(s) \in \{0, 1, \ldots, m\}$ such that

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,+}(x^{(i)}) \neq y^{(i)}\right\} = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m_0(s)} y^{(i)} p_i - \sum_{i=m_0(s)+1}^{m} y^{(i)} p_i\right)$$

and

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,-}(x^{(i)}) \neq y^{(i)}\right\} = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=m_0(s)+1}^{m} y^{(i)} p_i - \sum_{i=1}^{m_0(s)} y^{(i)} p_i\right)$$

Treat sums over empty sets of indices as zero, so that $\sum_{i=1}^0 a_i = 0$ for any a_i, and similarly $\sum_{i=m+1}^m a_i = 0$.

(b) [3 points] Define, for each $m_0 \in \{0, 1, \ldots, m\}$,

$$f(m_0) = \sum_{i=1}^{m_0} y^{(i)} p_i - \sum_{i=m_0+1}^{m} y^{(i)} p_i.$$

Show that there exists *some* $\gamma > 0$, which may depend on the training set size m (but should not depend on p), such that for any set of probabilities p on the training set, where $p_i \geq 0$ and $\sum_{i=1}^m p_i = 1$, we can find m_0 with

$$|f(m_0)| \geq 2\gamma.$$

What is your γ?

(*Hint:* Consider the difference $f(m_0) - f(m_0 + 1)$.)

(c) [2 points] Based on your answer to part (6b), what edge can thresholded decision stumps guarantee on any training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$, where the raw attributes $x^{(i)} \in \mathbb{R}$ are all distinct? Recall that the edge of a weak classifier $\phi : \mathbb{R} \to \{-1, 1\}$ is the constant $\gamma \in [0, \frac{1}{2}]$ such that

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi(x^{(i)}) \neq y^{(i)}\right\} \leq \frac{1}{2} - \gamma.$$

Can you give an upper bound on the number of thresholded decision stumps required to achieve zero error on a given training set?

(d) [11 points] Now you will implement boosting on data developed from a physics-based simulation of a high-energy particle accelerator. We provide two datasets, `boosting-train.csv` and `boosting-test.csv`, which consist of training data and test data for a binary classification problem on which you will apply boosting techniques. (For those not using `Matlab`, the files are comma-separated files, the first column of which consists of binary ± 1-labels $y^{(i)}$, the remaining 18 columns are the raw attribtues.) The file `load_data.m`, which we provide, loads the datasets into memory, storing training data and labels in appropriate vectors and matrices, and then performs boosting using *your* implemented code, and plots the results.

i. [5 points] Implement a method that finds the optimal thresholded decision stump for a training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ and distribution $p \in \mathbb{R}_+^m$ on the training set. In particular, fill out the code in the method `find_best_threshold.m`. Include your code in your solution.

ii. [2 points] Implement boosted decision stumps by filling out the code in the method `stump_booster.m`. Your code should implement the weight updating at each iteration $t = 1, 2, \ldots$ to find the optimal value θ_t given the feature index and threshold. Include your code in your solution.

iii. [2 points] Implement *random* boosting, where at each step the choice of decision stump is made completely randomly. In particular, at iteration t random boosting chooses a random index $j \in \{1, 2, \ldots, n\}$, then chooses a random threshold s from among the data values $\{x_j^{(i)}\}_{i=1}^m$, and then chooses the tth weight θ_t optimally for this (random) classifier $\phi_{s,+}(x) = \text{sign}(x_j - s)$. Implement this by filling out the code in `random_booster.m`.

iv. [2 points] Run the method `load_data.m` with your implemented boosting methods. Include the plots this method displays, which show the training and test error for boosting at each iteration $t = 1, 2, \ldots$. Which method is better?

[A few notes: we do not expect boosting to get classification accuracy better than approximately 80% for this problem.]

CS 229, Autumn 2016
Problem Set #2: Naive Bayes, SVMs, and Theory

Due Wednesday, November 2 at 11:00 am on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[15 points] Constructing kernels**

   In class, we saw that by choosing a kernel $K(x, z) = \phi(x)^T \phi(z)$, we can implicitly map data to a high dimensional space, and have the SVM algorithm work in that space. One way to generate kernels is to explicitly define the mapping ϕ to a higher dimensional space, and then work out the corresponding K.

   However in this question we are interested in direct construction of kernels. I.e., suppose we have a function $K(x, z)$ that we think gives an appropriate similarity measure for our learning problem, and we are considering plugging K into the SVM as the kernel function. However for $K(x, z)$ to be a valid kernel, it must correspond to an inner product in some higher dimensional space resulting from some feature mapping ϕ. Mercer's theorem tells us that $K(x, z)$ is a (Mercer) kernel if and only if for any finite set $\{x^{(1)}, \ldots, x^{(m)}\}$, the matrix K is symmetric and positive semidefinite, where the square matrix $K \in \mathbb{R}^{m \times m}$ is given by $K_{ij} = K(x^{(i)}, x^{(j)})$.

   Now here comes the question: Let K_1, K_2 be kernels over $\mathbb{R}^n \times \mathbb{R}^n$, let $a \in \mathbb{R}^+$ be a positive real number, let $f : \mathbb{R}^n \mapsto \mathbb{R}$ be a real-valued function, let $\phi : \mathbb{R}^n \to \mathbb{R}^d$ be a function mapping from $\mathbb{R}^n$ to $\mathbb{R}^d$, let K_3 be a kernel over $\mathbb{R}^d \times \mathbb{R}^d$, and let $p(x)$ a polynomial over x with *positive* coefficients.

   For each of the functions K below, state whether it is necessarily a kernel. If you think it is, prove it; if you think it isn't, give a counter-example.

   (a) [1 points] $K(x, z) = K_1(x, z) + K_2(x, z)$

   (b) [1 points] $K(x, z) = K_1(x, z) - K_2(x, z)$

   (c) [1 points] $K(x, z) = aK_1(x, z)$

   (d) [1 points] $K(x, z) = -aK_1(x, z)$

   (e) [5 points] $K(x, z) = K_1(x, z)K_2(x, z)$

   (f) [2 points] $K(x, z) = f(x)f(z)$

   (g) [2 points] $K(x, z) = K_3(\phi(x), \phi(z))$

   (h) [2 points] $K(x, z) = p(K_1(x, z))$

[Hint: For part (e), the answer is that the K there *is* indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

Answer: All 8 cases of proposed kernels K are trivially symmetric because K_1, K_2, K_3 are symmetric; and because the product of 2 real numbers is commutative (for (1f)). Thanks to Mercer's theorem, it is sufficient to prove the corresponding properties for positive semidefinite matrices. To differentiate between matrix and kernel function, we'll use G_i to denote a kernel matrix (Gram matrix) corresponding to a kernel function K_i.

(a) Kernel. The sum of 2 positive semidefinite matrices is a positive semidefinite matrix: $\forall z\ z^T G_1 z \geq 0, z^T G_2 z \geq 0$ since K_1, K_2 are kernels. This implies $\forall z\ z^T G z = z^T G_1 z + z^T G_2 z \geq 0$.

(b) Not a kernel. Counterexample: let $K_2 = 2K_1$ (we are using (1c) here to claim K_2 is a kernel). Then we have $\forall z\ z^T G z = z^T (G_1 - 2G_1) z = -z^T G_1 z \leq 0$.

(c) Kernel. $\forall z\ z^T G_1 z \geq 0$, which implies $\forall z\ a z^T G_1 z \geq 0$.

(d) Not a kernel. Counterexample: $a = 1$. Then we have $\forall z\ -z^T G_1 z \leq 0$.

(e) Kernel. K_1 is a kernel, thus $\exists \phi^{(1)}\ K_1(x,z) = \phi^{(1)}(x)^T \phi^{(1)}(z) = \sum_i \phi_i^{(1)}(x)\phi_i^{(1)}(z)$. Similarly, K_2 is a kernel, thus $\exists \phi^{(2)}\ K_2(x,z) = \phi^{(2)}(x)^T \phi^{(2)}(z) = \sum_j \phi_j^{(2)}(x)\phi_j^{(2)}(z)$.

$$
\begin{aligned}
K(x,z) &= K_1(x,z)K_2(x,z) && (1)\\
&= \sum_i \phi_i^{(1)}(x)\phi_i^{(1)}(z) \sum_i \phi_i^{(2)}(x)\phi_i^{(2)}(z) && (2)\\
&= \sum_i \sum_j \phi_i^{(1)}(x)\phi_i^{(1)}(z)\phi_j^{(2)}(x)\phi_j^{(2)}(z) && (3)\\
&= \sum_i \sum_j (\phi_i^{(1)}(x)\phi_j^{(2)}(x))(\phi_i^{(1)}(z)\phi_j^{(2)}(z)) && (4)\\
&= \sum_{(i,j)} \psi_{i,j}(x)\psi_{i,j}(z) && (5)
\end{aligned}
$$

Where the last equality holds because that's how we define ψ. We see K can be written in the form $K(x,z) = \psi(x)^T \psi(z)$ so it is a kernel.

Here is an alternate super-slick linear-algebraic proof. If G is the Gram matrix for the product $K_1 \times K_2$, then G is a principal submatrix of the Kronecker product $G_1 \otimes G_2$, where G_i is the Gram matrix for K_i. As the Kronecker product is positive semi-definite, so are its principal submatrices.

(f) Kernel. Just let $\psi(x) = f(x)$, and since $f(x)$ is a scalar, we have $K(x,z) = \phi(x)^T \phi(z)$ and we are done.

(g) Kernel. Since K_3 is a kernel, the matrix G_3 obtained for *any* finite set $\{x^{(1)}, \ldots, x^{(m)}\}$ is positive semidefinite, and so it is also positive semidefinite for the sets $\{\phi(x^{(1)}), \ldots, \phi(x^{(m)})\}$.

(h) Kernel. By combining (1a) sum, (1c) scalar product, (1e) powers, (1f) constant term, we see that any polynomial of a kernel K_1 will again be a kernel.

2. **[10 points] Kernelizing the Perceptron**

Let there be a binary classification problem with $y \in \{-1, 1\}$. The perceptron uses hypotheses of the form $h_\theta(x) = g(\theta^T x)$, where $g(z) = \text{sign}(z) = 1$ if $z \geq 0$, -1 otherwise.

In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

$$\theta^{(i+1)} := \begin{cases} \theta^{(i)} + \alpha y^{(i+1)} x^{(i+1)} & \text{if } h_{\theta^{(i)}}(x^{(i+1)}) y^{(i+1)} < 0 \\ \theta^{(i)} & \text{otherwise,} \end{cases}$$

where $\theta^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to $\vec{0}$.

Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ. Suppose ϕ is so high-dimensional (say, ∞-dimensional) that it's infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the "kernel trick" to the perceptron to make it work in the high-dimensional feature space ϕ, but without ever explicitly computing $\phi(x)$. [Note: You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(x) = 1$ so that this is taken care of.] Your description should specify

(a) How you will (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$, including how the initial value $\theta^{(0)} = \vec{0}$ is represented (note that $\theta^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(x)$);

(b) How you will efficiently make a prediction on a new input $x^{(i+1)}$. I.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = g({\theta^{(i)}}^T \phi(x^{(i+1)}))$, using your representation of $\theta^{(i)}$; and

(c) How you will modify the update rule given above to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$; i.e., using the update rule corresponding to the feature mapping ϕ:

$$\theta^{(i+1)} := \theta^{(i)} + \alpha \mathbf{1}\{{\theta^{(i)}}^T \phi(x^{(i+1)}) y^{(i+1)} < 0\} y^{(i+1)} \phi(x^{(i+1)}).$$

[Hint: our discussion of the representer theorem may be useful.]

Answer:

In the high-dimensional space we update θ as follows:

$$\theta := \theta + \alpha(y^{(i)} - h_\theta(\phi(x^{(i)})))\phi(x^{(i)})$$

So (assuming we initialize $\theta^{(0)} = \vec{0}$) θ will always be a linear combination of the $\phi(x^{(i)})$, i.e., $\exists \beta_l$ such that $\theta^{(i)} = \sum_{l=1}^{i} \beta_l \phi(x^{(l)})$ after having incorporated i training points. Thus $\theta^{(i)}$ can be compactly represented by the coefficients β_l of this linear combination, i.e., i real numbers after having incorporated i training points $x^{(i)}$. The initial value $\theta^{(0)}$ simply corresponds to the case where the summation has no terms (i.e., an empty list of coefficients β_l).

We do not work explicitly in the high-dimensional space, but use the fact that $g({\theta^{(i)}}^T \phi(x^{(i+1)})) = g(\sum_{l=1}^{i} \beta_l \cdot \phi(x^{(l)})^T \phi(x^{i+1})) = g(\sum_{l=1}^{i} \beta_l K(x^{(l)}, x^{(i+1)}))$, which can be computed efficiently.

We can efficiently update θ. We just need to compute $\beta_i = \alpha(y^{(i)} - g({\theta^{(i-1)}}^T \phi(x^{(i)})))$ at iteration i. This can be computed efficiently, if we compute ${\theta^{(i-1)}}^T \phi(x^{(i)})$ efficiently as described above.

In an alternative approach, one can observe that, unless a sample $\phi(x^{(i)})$ is misclassified, $y^{(i)} - h_{\theta^{(i)}}(\phi(x^{(i)}))$ will be zero; otherwise, it will be ± 1 (or ± 2, if the convention $y, h \in \{-1, 1\}$ is taken). The vector θ, then, can be represented as the sum $\sum_{\{i:y^{(i)} \neq h_{\theta^{(i)}}(\phi(x^{(i)}))\}} \alpha(2y^{(i)} - 1)\phi(x^{(i)})$ under the $y, h \in \{0, 1\}$ convention, and containing $(2y^{(i)})$ under the other convention. This can then be expressed as $\theta^{(i)} = \sum_{i \in \text{Misclassified}} \beta_i \phi(x^{(i)})$ to be in more obvious congruence with the above. The efficient representation can now be said to be a list which stores only those indices that were misclassified, as the β_is can be recomputed from the $y^{(i)}$s and α on demand. The derivation for (b) is then only cosmetically different, and in (c) the update rule is to add $(i + 1)$ to the list if $\phi(x^{(i+1)})$ is misclassified.

3. **[30 points] Spam classification**

   In this problem, we will use the naive Bayes algorithm and an SVM to build a spam classifier.

   In recent years, spam on electronic newsgroups has been an increasing problem. Here, we'll build a classifier to distinguish between "real" newsgroup messages, and spam messages. For this experiment, we obtained a set of spam emails, and a set of genuine newsgroup messages.[1] Using only the subject line and body of each message, we'll learn to distinguish between the spam and non-spam.

   All the files for the problem are in `http://cs229.stanford.edu/materials/spam_data.tgz`. **Note: Please do not circulate this data outside this class.** In order to get the text emails into a form usable by naive Bayes, we've already done some preprocessing on the messages. You can look at two sample spam emails in the files `spam_sample_original*`, and their preprocessed forms in the files `spam_sample_preprocessed*`. The first line in the preprocessed format is just the label and is not part of the message. The preprocessing ensures that only the message body and subject remain in the dataset; email addresses (EMAILADDR), web addresses (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. (In this problem, we'll going to call the features "tokens" rather than "words," since some of the features will correspond to special values like EMAILADDR. You don't have to worry about the distinction.) The files `news_sample_original` and `news_sample_preprocessed` also give an example of a non-spam mail.

   The work to extract feature vectors out of the documents has also been done for you, so you can just load in the design matrices (called document-word matrices in text classification) containing all the data. In a document-word matrix, the i^{th} row represents the i^{th} document/email, and the j^{th} column represents the j^{th} distinct token. Thus, the (i, j)-entry of this matrix represents the number of occurrences of the j^{th} token in the i^{th} document.

   For this problem, we've chosen as our set of tokens considered (that is, as our vocabulary) only the medium frequency tokens. The intuition is that tokens that occur too often or too rarely do not have much classification value. (Examples tokens that occur very often are words like "the," "and," and "of," which occur in so many emails and are sufficiently content-free that they aren't worth modeling.) Also, words were stemmed using a standard stemming algorithm; basically, this means that "price," "prices" and "priced" have all been replaced with "price," so that they can be treated as the same word. For a list of the tokens used, see the file `TOKENS_LIST`.

[1] Thanks to Christian Shelton for providing the spam email. The non-spam messages are from the 20 newsgroups data at `http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/theo-20/www/data/news20.html`.

Since the document-word matrix is extremely sparse (has lots of zero entries), we have stored it in our own efficient format to save space. You don't have to worry about this format.[2] The file `readMatrix.m` provides the `readMatrix` function that reads in the document-word matrix and the correct class labels for the various documents. Code in `nb_train.m` and `nb_test.m` shows how `readMatrix` should be called. The documentation at the top of these two files will tell you all you need to know about the setup.

(a) [11 points] Implement a naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing.

You should use the code outline provided in `nb_train.m` to train your parameters, and then use these parameters to classify the test set data by filling in the code in `nb_test.m`. You may assume that any parameters computed in `nb_train.m` are in memory when `nb_test.m` is executed, and do not need to be recomputed (i.e., that `nb_test.m` is executed immediately after `nb_train.m`) [3].

Train your parameters using the document-word matrix in `MATRIX.TRAIN`, and then report the test set error on `MATRIX.TEST`.

Remark. If you implement naive Bayes the straightforward way, you'll find that the computed $p(x|y) = \prod_i p(x_i|y)$ often equals zero. This is because $p(x|y)$, which is the product of many numbers less than one, is a very small number. The standard computer representation of real numbers cannot handle numbers that are too small, and instead rounds them off to zero. (This is called "underflow.") You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(x|y)$. [Hint: Think about using logarithms.]

(b) [3 points] Intuitively, some tokens may be particularly indicative of an email being in a particular class. We can try to get an informal sense of how indicative token i is for the SPAM class by looking at:

$$\log \frac{p(x_j = i|y = 1)}{p(x_j = i|y = 0)} = \log \left(\frac{P(\text{token } i|\text{email is SPAM})}{P(\text{token } i|\text{email is NOTSPAM})} \right).$$

Using the parameters fit in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., have the highest positive value on the measure above). The numbered list of tokens in the file `TOKENS_LIST` should be useful for identifying the words/tokens.

(c) [3 points] Repeat part (a), but with training sets of size ranging from 50, 100, 200, ..., up to 1400, by using the files `MATRIX.TRAIN.*`. Plot the test error each time (use `MATRIX.TEST` as the test data) to obtain a learning curve (test set error vs. training set size). You may need to change the call to `readMatrix` in `nb_train.m` to read the correct file each time. Which training-set size gives the best test set error?

(d) [11 points] Train an SVM on this dataset using stochastic gradient descent and the radial basis function (also known as the Gaussian) kernel, which sets

$$K(x, z) = \exp \left(-\frac{1}{2\tau^2} \|x - z\|_2^2 \right).$$

[2] Unless you're not using Matlab/Octave, in which case feel free to ask us about it. We have provided Julia code to read the file in `MatrixReading.jl`.

[3] Matlab note: If a .m file doesn't begin with a function declaration, the file is a script. Variables in a script are put into the global namespace, unlike with functions.

In this case, recall that (as proved in class) the objective with kernel matrix $K = [K^{(1)} \ \cdots \ K^{(m)}] \in \mathbb{R}^{m \times m}$ is given by

$$J(\alpha) = \frac{1}{m} \sum_{i=1}^{m} \left[1 - y^{(i)} {K^{(i)}}^T \alpha\right]_+ + \frac{\lambda}{2} \alpha^T K \alpha$$

where $[t]_+ = \max\{t, 0\}$ is the positive part function. In this case, the gradient (actually, this is known as a *subgradient*) of the individual loss terms is

$$\nabla_\alpha \left[1 - y^{(i)} K^{(i)} \alpha\right]_+ = \begin{cases} -y^{(i)} K^{(i)} & \text{if } y^{(i)} {K^{(i)}}^T \alpha < 1 \\ 0 & \text{otherwise.} \end{cases}$$

In your SVM training, you should perform stochastic gradient descent, where in each iteration you choose an index $i \in \{1, \ldots, m\}$ uniformly at random, for a total of $40 \cdot m$ steps, where m is the training set size, and your kernel should use $\tau = 8$ and regularization multiplier $\lambda = \frac{1}{64m}$. For this part of the problem, you should also replace each training or test point $x^{(i)}$ with a zero-one vector $z^{(i)}$, where $z_j^{(i)} = 1$ if $x_j^{(i)} > 0$ and $z_j^{(i)} = 0$ if $x_j^{(i)} = 0$. Initialize your SGD procedure at $\alpha = 0$.

The output of your training code, which you should implement in `svm_test.m`, should be the α vector that is the *average* of all the α vectors that your iteration updates. At iteration t of stochastic gradient descent you should use stepsize $1/\sqrt{t}$.

Similar to part (c), train an SVM with training set sizes 50, 100, 200, ..., 1400, by using the file `MATRIX.TRAIN.50` and so on. Plot the test error each time, using `MATRIX.TEST` as the test data.

(A few hints for more efficient Matlab code: you should try to vectorize creation of the Kernel matrix, and you should call the method `full` to make the matrix non-sparse, which will make the method faster. In addition, the training data uses labels in $\{0, 1\}$, so you should change the output of the `readMatrix` method to have labels $y \in \{-1, 1\}$.)

(e) [2 points] How do naive Bayes and Support Vector Machines compare (in terms of generalization error) as a function of the training set size?

Answer:

(a) The test error when training on the full training set was 1.63%. If you got a different error (or if you got the words `website` and `lowest` for part b), you most probably implemented the wrong Naive Bayes model.

(b) The five most indicative words for the spam class were: `httpaddr`, `spam`, `unsubscrib`, `ebai` and `valet`.

(c) The test set error for different training set set sizes was:

i. Training set size 50: Test set error = 3.87%
ii. Training set size 100: Test set error = 2.62%
iii. Training set size 200: Test set error = 2.62%
iv. Training set size 400: Test set error = 1.87%
v. Training set size 800: Test set error = 1.75%
vi. Training set size 1400: Test set error = 1.63%

vii. Full training set: Test set error = 1.63%

(d) The test set error from the SVM for different training set sizes, averaged over 10 randomizations of the training data order, was:

i. Training set size 50: Test set error = 2.26%
ii. Training set size 100: Test set error = 1.47%
iii. Training set size 200: Test set error = .26%
iv. Training set size 400: Test set error = .14%
v. Training set size 800: Test set error = 0%
vi. Training set size 1400: Test set error = 0%

(e) The deduction that can be drawn is that for this dataset, Naive Bayes is simply not as good as the Kernelized SVM.

The Matlab code for the problem:

```
% ------------------------------------------------------------------------ %
% ---------------               Outer  loop                --------------- %
% ------------------------------------------------------------------------ %

training_set_size_list = [50, 100, 200, 400, 800, 1400];

%% SVM training %%

test_errors_svm = zeros(length(training_set_size_list), 1);
total_svms_to_avg = 10;
[M, tokenlist, category] = readMatrix('MATRIX.TEST');
Xtest = M;
ytest = (2 * category - 1)';

for train_ind = 1:length(training_set_size_list)
  for iter = 1:total_svms_to_avg
    num_train = training_set_size_list(train_ind);
    svm_train;
    svm_test;
    test_errors_svm(train_ind) = test_errors_svm(train_ind) + test_error;
  end
end

test_errors_svm = test_errors_svm / total_svms_to_avg;

%% Naive Bayes training %%

test_errors_nb = zeros(length(training_set_size_list), 1);
for train_ind = 1:length(training_set_size_list)
  num_train = training_set_size_list(train_ind);
  nb_train;
  nb_test;
  test_errors_nb(train_ind) = error;
```

```
end

figure;
semilogx(training_set_size_list, test_errors_svm, 'bs-', 'linewidth', 2);
hold on;
semilogx(training_set_size_list, test_errors_nb, 'ko-', 'linewidth', 2);
legend('SVM error', 'NB error');
set(gca, 'fontsize', 18);
axis([min(training_set_size_list), max(training_set_size_list), 0, .04]);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% svm_train.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[sparseTrainMatrix, tokenlist, trainCategory] = ...
    readMatrix(sprintf('MATRIX.TRAIN.%d', num_train));
m_train = size(Xtrain, 1);
ytrain = (2 * trainCategory - 1)';
Xtrain = 1.0 * (Xtrain > 0);

squared_X_train = sum(Xtrain.^2, 2);
gram_train = Xtrain * Xtrain';
tau = 8;

% Get full training matrix for kernels using vectorized code.
Ktrain = full(exp(-(repmat(squared_X_train, 1, m_train) ...
                    + repmat(squared_X_train', m_train, 1) ...
                    - 2 * gram_train) / (2 * tau^2)));

lambda = 1 / (64 * m_train);
num_outer_loops = 40;
alpha = zeros(m_train, 1);

avg_alpha = zeros(m_train, 1);
Imat = eye(m_train);

count = 0;
for ii = 1:(num_outer_loops * m_train)
  count = count + 1;
  ind = ceil(rand * m_train);
  margin = ytrain(ind) * Ktrain(ind, :) * alpha;
  g = -(margin < 1) * ytrain(ind) * Ktrain(:, ind) + ...
      m_train * lambda * (Ktrain(:, ind) * alpha(ind));
  % g(ind) = g(ind) + m_train * lambda * Ktrain(ind,:) * alpha;
  alpha = alpha - g / sqrt(count);
  avg_alpha = avg_alpha + alpha;
end
avg_alpha = avg_alpha / (num_outer_loops * m_train);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% svm_test.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Construct test and train matrices
Xtest = 1.0 * (Xtest > 0);
squared_X_test = sum(Xtest.^2, 2);
m_test = size(Xtest, 1);
gram_test = Xtest * Xtrain';
Ktest = full(exp(-(repmat(squared_X_test, 1, m_train) ...
                   + repmat(squared_X_train', m_test, 1) ...
                   - 2 * gram_test) / (2 * tau^2)));

% preds = Ktest * alpha;

% fprintf(1, 'Test error rate for final alpha: %1.4f\n', ...
%         sum(preds .* ytest <= 0) / length(ytest));

preds = Ktest * avg_alpha;
fprintf(1, 'Test error rate for average alpha: %1.4f\n', ...
        sum(preds .* ytest <= 0) / length(ytest));
test_error = sum(preds .* ytest <= 0) / length(ytest);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nb_test.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[spmatrix, tokenlist, category] = readMatrix('MATRIX.TEST');

testMatrix = full(spmatrix);
numTestDocs = size(testMatrix, 1);
numTokens = size(testMatrix, 2);

% ...
output = zeros(numTestDocs, 1);

%---------------
% YOUR CODE HERE

for k=1:numTestDocs,
  [i,j,v] = find(testMatrix(k,:));
  neg_posterior = sum(v .* neg_log_phi(j)) + neg_log_prior;
  pos_posterior = sum(v .* pos_log_phi(j)) + pos_log_prior;

  if (neg_posterior > pos_posterior)
    output(k) = 0;
  else
    output(k) = 1;
  end
```

```
end

%---------------

y = full(category);
y = y(:);
% Compute the error on the test set
error = sum(y ~= output) / numTestDocs;

%Print out the classification error on the test set
fprintf(1, 'Test error: %1.4f\n', error);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% nb_train.m
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


[spmatrix, tokenlist, trainCategory] = ...
    readMatrix(sprintf('MATRIX.TRAIN.%d', num_train));

trainMatrix = full(spmatrix);
numTrainDocs = size(trainMatrix, 1);
numTokens = size(trainMatrix, 2);

% ...
% YOUR CODE HERE

V = size(trainMatrix, 2);
neg = trainMatrix(find(trainCategory == 0), :);
pos = trainMatrix(find(trainCategory == 1), :);

neg_words = sum(sum(neg));
pos_words = sum(sum(pos));

neg_log_prior = log(size(neg,1) / numTrainDocs);
pos_log_prior = log(size(pos,1) / numTrainDocs);

for k=1:V,
  neg_log_phi(k) = log((sum(neg(:,k)) + 1) / (neg_words + V));
  pos_log_phi(k) = log((sum(pos(:,k)) + 1) / (pos_words + V));
end
```

4. **[20 points] Properties of VC dimension**

   In this problem, we investigate a few properties of the Vapnik-Chervonenkis dimension, mostly relating to how VC(H) increases as the set H increases. For each part of this problem, you should state whether the given statement is true, and justify your answer with either a formal proof or a counter-example.

(a) Let two hypothesis classes H_1 and H_2 satisfy $H_1 \subseteq H_2$. Prove or disprove: $\text{VC}(H_1) \leq \text{VC}(H_2)$.

(b) Let $H_1 = H_2 \cup \{h_1, \ldots, h_k\}$. (I.e., H_1 is the union of H_2 and some set of k additional hypotheses.) Prove or disprove: $\text{VC}(H_1) \leq \text{VC}(H_2) + k$. [Hint: You might want to start by considering the case of $k = 1$.]

(c) Let $H_1 = H_2 \cup H_3$. Prove or disprove: $\text{VC}(H_1) \leq \text{VC}(H_2) + \text{VC}(H_3)$.

Answer:

(a) True. Suppose that $VC(H_1) = d$. Then there exists a set of d points that is shattered by H_1 (i.e., for each possible labeling of the d points, there exists a hypothesis $h \in H_1$ which realizes that labeling). Now, since H_2 contains all hypotheses in H_1, then H_2 shatters the same set, and thus we have $VC(H_2) \geq d = VC(H_1)$.

(b) True. If we can prove the result for $k = 1$, then the result stated in the problem set follows immediately by applying the same logic inductively, one hypothesis at a time. So, let us prove that if $H_1 = H_2 \cup \{h\}$, then $VC(H_1) \leq VC(H_2) + 1$. Suppose that $VC(H_1) = d$, and let S_1 be a set of d points that is shattered by H_1. Now, pick an arbitrary $x \in S_1$. Since H_1 shatters S_1, there must be some $\bar{h} \in H_1$ such that h and $\bar{h}$ agree on labelings for all points in S_1 except x. This means that $H' := H_1 \setminus \{h\}$ achieves all possible labelings on $S' := S_1 \setminus \{x\}$ (i.e. H' shatters S'), so $VC(H') \geq |S'| = d - 1$. But $H' \subseteq H_2$, so from part (a), $VC(H') \leq VC(H_2)$. It follows that $VC(H_2) \geq d - 1$, or equivalently, $VC(H_1) \leq VC(H_2) + 1$, as desired.

For this problem, there were a number of possible correct proof methods; generally, to get full credit, you needed to argue formally that there exists no set of $(VC(H_2) + 2)$ points shattered by H_1, or equivalently, that there always exists a set of $(VC(H_1) - 1)$ points shattered by H_2. Here are a couple of the more common errors:

- Some submitted solutions stated that adding a single hypothesis to H_2 increases the VC dimension by at most one, since the new hypothesis can only realize a single labeling. While this statement is vaguely true, it is neither sufficiently precise, nor is its correctness immediately obvious.
- Some solutions made arguments relating to the cardinality of the sets H_1 and H_2. However, generally when we speak about VC dimension, the sets H_1 and H_2 often have infinite cardinality (e.g., the set of all linear classifiers in $\mathbb{R}^2$).

(c) False. Counterexample: let $H_1 = \{h_1\}, H_2 = \{h_2\}$, and $\forall x,\ h_1(x) = 0, h_2(x) = 1$. Then we have $VC(H_1) = VC(H_2) = 0$, but $VC(H_1 \cup H_2) = 1$.

5. **[20 points] Training and testing on different distributions**

In the discussion in class about learning theory, a key assumption was that we trained and tested our learning algorithms on the same distribution $\mathcal{D}$. In this problem, we'll investigate one special case of training and testing on different distributions. Specifically, we will consider what happens when the training labels are *noisy*, but the test labels are not.

Consider a binary classification problem with labels $y \in \{0, 1\}$, and let $\mathcal{D}$ be a distribution over (x, y), that we'll think of as the original, "clean" or "uncorrupted" distribution. Define $\mathcal{D}_\tau$ to be a "corrupted" distribution over (x, y) which is the same as $\mathcal{D}$, except that the labels y have some probability $0 \leq \tau < 0.5$ of being flipped. Thus, to sample from $\mathcal{D}_\tau$,

we would first sample (x, y) from $\mathcal{D}$, and then with probability τ (independently of the observed x and y) replace y with $1 - y$. Note that $\mathcal{D}_0 = \mathcal{D}$.

The distribution $\mathcal{D}_\tau$ models a setting in which an unreliable human (or other source) is labeling your training data for you, and on each example he/she has a probability τ of mislabeling it. Even though our training data is corrupted, we are still interested in evaluating our hypotheses with respect to the original, uncorrupted distribution $\mathcal{D}$.

We define the generalization error *with respect to* $\mathcal{D}_\tau$ to be

$$\varepsilon_\tau(h) = P_{(x,y)\sim\mathcal{D}_\tau}[h(x) \neq y].$$

Note that $\varepsilon_0(h)$ is the generalization error with respect to the "clean" distribution; it is with respect to ε_0 that we wish to evaluate our hypotheses.

(a) For any hypothesis h, the quantity $\varepsilon_0(h)$ can be calculated as a function of $\varepsilon_\tau(h)$ and τ. Write down a formula for $\varepsilon_0(h)$ in terms of $\varepsilon_\tau(h)$ and τ, and justify your answer.

(b) Let $|H|$ be finite, and suppose our training set $S = \{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ is obtained by drawing m examples IID from the corrupted distribution $\mathcal{D}_\tau$. Suppose we pick $h \in H$ using empirical risk minimization: $\hat{h} = \arg\min_{h\in H} \hat{\varepsilon}_S(h)$. Also, let $h^* = \arg\min_{h\in H} \varepsilon_0(h)$.

Let any $\delta, \gamma > 0$ be given. Prove that for

$$\varepsilon_0(\hat{h}) \leq \varepsilon_0(h^*) + 2\gamma$$

to hold with probability $1 - \delta$, it suffices that

$$m \geq \frac{1}{2(1-2\tau)^2\gamma^2} \log \frac{2|H|}{\delta}.$$

Remark. This result suggests that, roughly, m examples that have been corrupted at noise level τ are worth about as much as $(1-2\tau)^2 m$ uncorrupted training examples. This is a useful rule-of-thumb to know if you ever need to decide whether/how much to pay for a more reliable source of training data. (If you've taken a class in information theory, you may also have heard that $(1-\mathcal{H}(\tau))m$ is a good estimate of the information in the m corrupted examples, where $\mathcal{H}(\tau) = -(\tau \log_2 \tau + (1-\tau)\log_2(1-\tau))$ is the "binary entropy" function. And indeed, the functions $(1-2\tau)^2$ and $1-\mathcal{H}(\tau)$ are quite close to each other.)

(c) Comment **briefly** on what happens as τ approaches 0.5.

Answer:

(a) We compute ε_τ as a function of ε_0 and then invert the obtained expression. An error occurs on the corrupted distribution, if and only if, an error occurred for the original distribution and the point that was not corrupted, or no error occurred for the original distribution but the point was corrupted. So we have

$$\varepsilon_\tau = \varepsilon_0(1-\tau) + (1-\varepsilon_0)\tau$$

Solving for ε_0 gives

$$\varepsilon_0 = \frac{\varepsilon_\tau - \tau}{1-2\tau}$$

(b) We will need to apply the following (in the right order):

$$\forall h \in H,\ |\varepsilon_\tau(h) - \hat{\varepsilon}_\tau(h)| \leq \bar{\gamma} \quad \text{w.p.}(1-\delta), \quad \delta = 2K\exp(-2\bar{\gamma}^2 m) \tag{6}$$

$$\varepsilon_\tau = (1-2\tau)\varepsilon + \tau, \quad \varepsilon_0 = \frac{\varepsilon_\tau - \tau}{1-2\tau} \tag{7}$$

$$\forall h \in H,\ \hat{\varepsilon}_\tau(\hat{h}) \leq \hat{\varepsilon}_\tau(h), \quad \text{in particular for } h^* \tag{8}$$

Here is the derivation:

$$\varepsilon_0(\hat{h}) = \frac{\varepsilon_\tau(\hat{h}) - \tau}{1-2\tau} \tag{9}$$

$$\leq \frac{\hat{\varepsilon}_\tau(\hat{h}) + \bar{\gamma} - \tau}{1-2\tau} \quad \text{w.p.}(1-\delta) \tag{10}$$

$$\leq \frac{\hat{\varepsilon}_\tau(h^*) + \bar{\gamma} - \tau}{1-2\tau} \quad \text{w.p.}(1-\delta) \tag{11}$$

$$\leq \frac{\varepsilon_\tau(h^*) + 2\bar{\gamma} - \tau}{1-2\tau} \quad \text{w.p.}(1-\delta) \tag{12}$$

$$= \frac{(1-2\tau)\varepsilon_0(h^*) + \tau + 2\bar{\gamma} - \tau}{1-2\tau} \quad \text{w.p.}(1-\delta) \tag{13}$$

$$= \varepsilon_0(h^*) + \frac{2\bar{\gamma}}{1-2\tau} \quad \text{w.p.}(1-\delta) \tag{14}$$

$$= \varepsilon_0(h^*) + 2\gamma \quad \text{w.p.}(1-\delta) \tag{15}$$

Where we used in the following order: (7)(6)(8)(6)(7), and the last 2 steps are algebraic simplifications, and defining γ as a function of $\bar{\gamma}$. Now we can fill out $\bar{\gamma} = \gamma(1-2\tau)$ into δ of (6), solve for m and we are done.

Note: one could shorten the above derivation and go straight from (9) to (12) by using that result from class.

(c) The closer τ is to 0.5, the more samples are needed to get the same generalization error bound. For τ approaching 0.5, the training data becomes more and more random; having no information at all about the underlying distribution for $\tau = 0.5$.

6. **[19 points] Boosting and high energy physics**

In class, we discussed boosting algorithms and decision stumps. In this problem, we explore applications of these ideas to detect particle emissions in a high-energy particle accelerator. In high energy physics, such as at the Large Hadron Collider (LHC), one accelerates small particles to relativistic speeds and smashes them into one another, tracking the emitted particles. The goal in these problems is to detect the emission of certain interesting particles based on other observed particles and energies.[4] In this problem, we explore the application of boosting to a high energy physics problem, where we use decision stumps applied to 18 low- and high-level physics-based features. All data for the problem is available at `http://cs229.stanford.edu/materials/boost_data.tgz`.

[4]For more, see the following paper: Baldi, Sadowski, Whiteson. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Communications* 5, Article 4308. `http://arxiv.org/abs/1402.4735`.

For the first part of the problem, we explore how decision stumps based on thresholding can provide a weak-learning guarantee. In particular, we show that for real-valued attributes x, there is an edge $\gamma > 0$ that decision stumps guarantee. Recall that thresholding-based decision stumps are functions indexed by a threshold s and sign $+/-$, such that

$$\phi_{s,+}(x) = \begin{cases} 1 & \text{if } x \geq s \\ -1 & \text{if } x < s \end{cases} \quad \text{and} \quad \phi_{s,-}(x) = \begin{cases} -1 & \text{if } x \geq s \\ 1 & \text{if } x < s. \end{cases}$$

That is, $\phi_{s,+}(x) = -\phi_{s,-}(x)$. We assume for simplicity in the theoretical parts of this exercise that our input attribute vectors $x \in \mathbb{R}$, that is, they are one-dimensional. Now, we would like guarantee that there is some $\gamma > 0$ and a threshold s such that, for *any* distribution p on the training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ (where $y^{(i)} \in \{-1, +1\}$ and $x^{(i)} \in \mathbb{R}$, and we recall that p is a distribution on the training set if $\sum_{i=1}^m p_i = 1$ and $p_i \geq 0$ for each i) we have

$$\sum_{i=1}^m p_i \mathbf{1}\left\{y^{(i)} \neq \phi_{s,+}(x^{(i)})\right\} \leq \frac{1}{2} - \gamma \quad \text{or} \quad \sum_{i=1}^m p_i \mathbf{1}\left\{y^{(i)} \neq \phi_{s,-}(x^{(i)})\right\} \leq \frac{1}{2} - \gamma.$$

For simplicity, we assume that all of the $x^{(i)}$ are *distinct*, so that none of them are equal. We also assume (without loss of generality, but this makes the problem notationally simpler) that

$$x^{(1)} > x^{(2)} > \cdots > x^{(m)}.$$

(a) [3 points] Show that for each threshold s, there is some $m_0(s) \in \{0, 1, \ldots, m\}$ such that

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,+}(x^{(i)}) \neq y^{(i)}\right\} = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m_0(s)} y^{(i)} p_i - \sum_{i=m_0(s)+1}^m y^{(i)} p_i\right)$$

and

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,-}(x^{(i)}) \neq y^{(i)}\right\} = \frac{1}{2} - \frac{1}{2}\left(\sum_{i=m_0(s)+1}^m y^{(i)} p_i - \sum_{i=1}^{m_0(s)} y^{(i)} p_i\right)$$

Treat sums over empty sets of indices as zero, so that $\sum_{i=1}^0 a_i = 0$ for any a_i, and similarly $\sum_{i=m+1}^m a_i = 0$.

Answer:

We perform several algebraic steps. Let $\operatorname{sgn}(t) = 1$ if $t \geq 0$, and $\operatorname{sgn}(t) = -1$ otherwise. Then $\mathbf{1}\{\phi_{s,+}(x) \neq y\} = \mathbf{1}\{\operatorname{sgn}(x - s) \neq y\} = \mathbf{1}\{y \operatorname{sgn}(x - s) \leq 0\}$. Thus we have

$$\begin{aligned}\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,+}(x^{(i)}) \neq y^{(i)}\right\} &= \sum_{i=1}^m p_i \mathbf{1}\left\{y^{(i)} \cdot \operatorname{sgn}(x^{(i)} - s) \leq 0\right\} \\ &= \sum_{i:x^{(i)} \geq s} p_i \mathbf{1}\left\{y^{(i)} = -1\right\} + \sum_{i:x^{(i)} < s} p_i \mathbf{1}\left\{y^{(i)} = 1\right\}.\end{aligned}$$

Thus, if we let $m_0(s)$ be the index in $\{1, \ldots, m\}$ such that $x^{(i)} \geq s$ for $i \leq m_0(s)$ and $x^{(i)} < s$ for $i > m_0(s)$, which we know must exist because $x^{(1)} > x^{(2)} > \cdots$, we have

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi_{s,+}(x^{(i)}) \neq y^{(i)}\right\} = \sum_{i=1}^{m_0(s)} p_i \mathbf{1}\left\{y^{(i)} = -1\right\} + \sum_{i=m_0(s)+1}^m p_i \mathbf{1}\left\{y^{(i)} = 1\right\}.$$

Now we make the key observation: we have

$$\mathbf{1}\{y = -1\} = \frac{1-y}{2} \text{ and } \mathbf{1}\{y = 1\} = \frac{1+y}{2},$$

as $y \in \{-1, 1\}$. Consquently,

$$\begin{aligned}
\sum_{i=1}^{m} p_i \mathbf{1}\left\{\phi_{s,+}(x^{(i)}) \neq y^{(i)}\right\} &= \sum_{i=1}^{m_0(s)} p_i \mathbf{1}\left\{y^{(i)} = -1\right\} + \sum_{i=m_0(s)+1}^{m} p_i \mathbf{1}\left\{y^{(i)} = 1\right\} \\
&= \sum_{i=1}^{m_0(s)} p_i \frac{1-y^{(i)}}{2} + \sum_{i=m_0(s)+1}^{m} p_i \frac{1+y^{(i)}}{2} \\
&= \frac{1}{2}\sum_{i=1}^{m} p_i - \frac{1}{2}\sum_{i=1}^{m_0(s)} p_i y^{(i)} + \frac{1}{2}\sum_{i=m_0(s)+1}^{m} p_i y^{(i)} \\
&= \frac{1}{2} - \frac{1}{2}\left(\sum_{i=1}^{m_0(s)} p_i y^{(i)} - \sum_{i=m_0(s)+1}^{m} p_i y^{(i)}\right).
\end{aligned}$$

The last equality follows because $\sum_{i=1}^{m} p_i = 1$. The case for $\phi_{s,-}$ is symmetric to this one, so we omit the argument.

(b) [3 points] Define, for each $m_0 \in \{0, 1, \ldots, m\}$,

$$f(m_0) = \sum_{i=1}^{m_0} y^{(i)} p_i - \sum_{i=m_0+1}^{m} y^{(i)} p_i.$$

Show that there exists *some* $\gamma > 0$, which may depend on the training set size m (but should not depend on p), such that for any set of probabilities p on the training set, where $p_i \geq 0$ and $\sum_{i=1}^{m} p_i = 1$, we can find m_0 with

$$|f(m_0)| \geq 2\gamma.$$

What is your γ?

(*Hint:* Consider the difference $f(m_0) - f(m_0 + 1)$.)

Answer: We have for $m_0 \in \{1, \ldots, m\}$ that

$$f(m_0) - f(m_0 - 1) = \sum_{i=1}^{m_0} y^{(i)} p_i - \sum_{i=m_0+1}^{m} y^{(i)} p_i - \sum_{i=1}^{m_0-1} y^{(i)} p_i + \sum_{i=m_0}^{m} y^{(i)} p_i = 2y^{(m_0)} p_{m_0}.$$

In particular, we have $|f(m_0) - f(m_0 - 1)| = 2|y^{(m_0)}| p_{m_0} = 2p_{m_0}$ for all $m_0 \in \{1, \ldots, m\}$. Because $\sum_{i=1}^{m} p_i = 1$, there must be at least 1 index m_0 with $p_{m_0} \geq \frac{1}{m}$. Thus we have for some index m_0 that $|f(m_0) - f(m_0 - 1)| \geq \frac{2}{m}$, and so it must be the case that at least one of

$$|f(m_0)| \geq \frac{1}{m} \text{ or } |f(m_0 - 1)| \geq \frac{1}{m}$$

holds. We have $\gamma = \frac{1}{2m}$.

(c) [2 points] Based on your answer to part (6b), what edge can thresholded decision stumps guarantee on any training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$, where the raw attributes $x^{(i)} \in \mathbb{R}$ are all distinct? Recall that the edge of a weak classifier $\phi : \mathbb{R} \to \{-1, 1\}$ is the constant $\gamma \in [0, \frac{1}{2}]$ such that

$$\sum_{i=1}^m p_i \mathbf{1}\left\{\phi(x^{(i)}) \neq y^{(i)}\right\} \leq \frac{1}{2} - \gamma.$$

Can you give an upper bound on the number of thresholded decision stumps required to achieve zero error on a given training set?

Answer: Based on our answer to the first part of the question, the thresholded decision stumps are guaranteed to have edge at least $\gamma = \frac{1}{2m}$ over random guessing.

Boosting takes $\frac{\log m}{2\gamma^2}$ iterations to achieve zero error, as shown in class, so with decision stumps we will achieve zero error in at most $2m^2 \log m$ iterations of boosting. Each iteration of boosting introduces a single new weak classifier/hypothesis, so at most $2m^2 \log m$ thresholded decision stumps are necessary.

(d) [11 points] Now you will implement boosting on data developed from a physics-based simulation of a high-energy particle accelerator. We provide two datasets, `boosting-train.csv` and `boosting-test.csv`, which consist of training data and test data for a binary classification problem on which you will apply boosting techniques. (For those not using `Matlab`, the files are comma-separated files, the first column of which consists of binary ± 1-labels $y^{(i)}$, the remaining 18 columns are the raw attribtues.) The file `load_data.m`, which we provide, loads the datasets into memory, storing training data and labels in appropriate vectors and matrices, and then performs boosting using *your* implemented code, and plots the results.

i. [5 points] Implement a method that finds the optimal thresholded decision stump for a training set $\{x^{(i)}, y^{(i)}\}_{i=1}^m$ and distribution $p \in \mathbb{R}_+^m$ on the training set. In particular, fill out the code in the method `find_best_threshold.m`. Include your code in your solution.

ii. [2 points] Implement boosted decision stumps by filling out the code in the method `stump_booster.m`. Your code should implement the weight updating at each iteration $t = 1, 2, \ldots$ to find the optimal value θ_t given the feature index and threshold. Include your code in your solution.

iii. [2 points] Implement *random* boosting, where at each step the choice of decision stump is made completely randomly. In particular, at iteration t random boosting chooses a random index $j \in \{1, 2, \ldots, n\}$, then chooses a random threshold s from among the data values $\{x_j^{(i)}\}_{i=1}^m$, and then chooses the tth weight θ_t optimally for this (random) classifier $\phi_{s,+}(x) = \text{sign}(x_j - s)$. Implement this by filling out the code in `random_booster.m`.

iv. [2 points] Run the method `load_data.m` with your implemented boosting methods. Include the plots this method displays, which show the training and test error for boosting at each iteration $t = 1, 2, \ldots$. Which method is better?

Answer: Random decision stumps require about 200 iterations to get to error .22 or so, while regular boosting (with greedy decision stumps) requires about 15 iterations to get this error. See Fig. 1.

[A few notes: we do not expect boosting to get classification accuracy better than approximately 80% for this problem.]

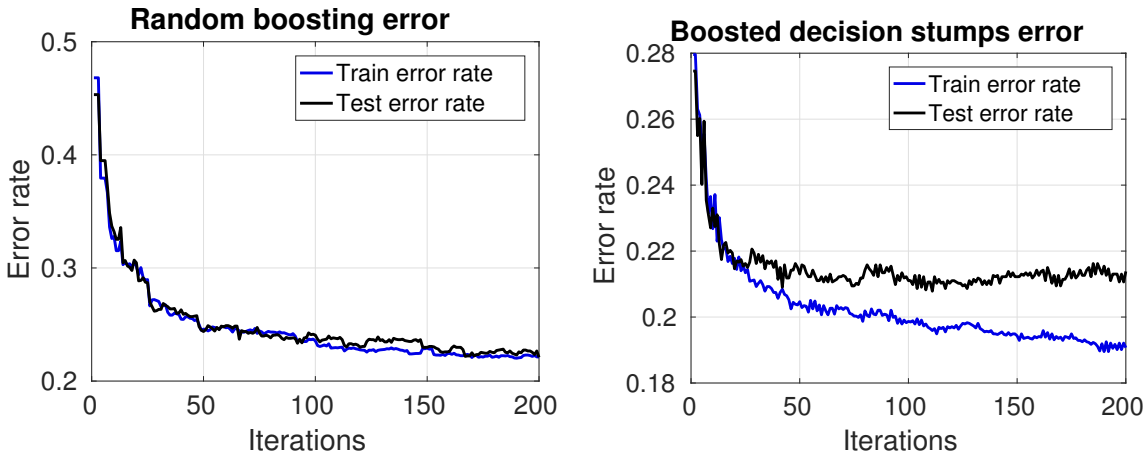


Figure 1: Boosting error for random selection of decision stumps and the greedy selection made by boosting.

Answer: Here is code for each of the three coding parts.

```
function [ind, thresh] = find_best_threshold(X, y, p_dist)
% FIND_BEST_THRESHOLD Finds the best threshold for the given data
%
% [ind, thresh] = find_best_threshold(X, y, p_dist) returns a threshold
%   thresh and index ind that gives the best thresholded classifier for the
%   weights p_dist on the training data. That is, the returned index ind
%   and threshold thresh minimize
%
%    sum_{i = 1}^m p(i) * 1{sign(X(i, ind) - thresh) ~= y(i)}
%
%   OR
%
%    sum_{i = 1}^m p(i) * 1{sign(thresh - X(i, ind)) ~= y(i)}.
%
%   We must check both signed directions, as it is possible that the best
%   decision stump (coordinate threshold classifier) is of the form
%   sign(threshold - x_j) rather than sign(x_j - threshold).
%
%   The data matrix X is of size m-by-n, where m is the training set size
%   and n is the dimension.
%
%   The solution version uses efficient sorting and data structures to perform
%   this calculation in time O(n m log(m)), where the size of the data matrix
%   X is m-by-n.

[mm, nn] = size(X);
best_err = inf;
ind = 1;
```

```
thresh = 0;

for jj = 1:nn
  [x_sort, inds] = sort(X(:, jj), 1, 'descend');
  p_sort = p_dist(inds);
  y_sort = y(inds);
  % We let the thresholds be s_0, s_1, ..., s_{m-1}, where s_k is between
  % x_sort(k-1) and x_sort(k) (so that s_0 > x_sort(1)). Then the empirical
  % error associated with threshold s_k is exactly
  %
  % err(k) = sum_{l = k + 1}^m p_sort(l) * 1(y_sort(l) == 1)
  %          + sum_{l = 1}^k p_sort(l) * 1(y_sort(l) == -1),
  %
  % because this is where the thresholds fall. Then we can sequentially
  % compute
  %
  % err(l) = err(l - 1) - p_sort(l) y_sort(l),
  %
  % where err(0) = p_sort' * (y_sort == 1).
  %
  % The code below actually performs this calculation with indices shifted by
  % one due to Matlab indexing.
  s = x_sort(1) + 1;
  possible_thresholds = x_sort;
  possible_thresholds = (x_sort + circshift(x_sort, 1)) / 2;
  possible_thresholds(1) = x_sort(1) + 1;
  increments = circshift(p_sort .* y_sort, 1);
  increments(1) = 0;
  emp_errs = ones(mm, 1) * (p_sort' * (y_sort == 1));
  emp_errs = emp_errs - cumsum(increments);
  [best_low, thresh_ind] = min(emp_errs);
  [best_high, thresh_high] = max(emp_errs);
  best_high = 1 - best_high;
  best_err_j = min(best_high, best_low);
  if (best_high < best_low)
    thresh_ind = thresh_high;
  end
  if (best_err_j < best_err)
    ind = jj;
    thresh = possible_thresholds(thresh_ind);
    best_err = best_err_j;
  end
end

function [theta, feature_inds, thresholds] = stump_booster(X, y, T)
% STUMP_BOOSTER Uses boosted decision stumps to train a classifier
%
% [theta, feature_inds, thresholds] = stump_booster(X, y, T)
%  performs T rounds of boosted decision stumps to classify the data X,
```

```
%  which is an m-by-n matrix of m training examples in dimension n,
%  to match y.
%
%  The returned parameters are theta, the parameter vector in T dimensions,
%  the feature_inds, which are indices of the features (a T dimensional
%  vector taking values in {1, 2, ..., n}), and thresholds, which are
%  real-valued thresholds. The resulting classifier may be computed on an
%  n-dimensional training example by
%
%   theta' * sign(x(feature_inds) - thresholds).
%
%  The resulting predictions may be computed simultaneously on an
%  n-dimensional dataset, represented as an m-by-n matrix X, by
%
%  sign(X(:, feature_inds) - repmat(thresholds', m, 1)) * theta.
%
%  This is an m-vector of the predicted margins.

[mm, nn] = size(X);
p_dist = ones(mm, 1);
p_dist = p_dist / sum(p_dist);

theta = [];
feature_inds = [];
thresholds = [];

for iter = 1:T
  [ind, thresh] = find_best_threshold(X, y, p_dist);
  Wplus = p_dist' * (sign(X(:, ind) - thresh) == y);
  Wminus = p_dist' * (sign(X(:, ind) - thresh) ~= y);
  theta = [theta; .5 * log(Wplus / Wminus)];
  feature_inds = [feature_inds; ind];
  thresholds = [thresholds; thresh];
  p_dist = exp(-y .* (...
    sign(X(:, feature_inds) - repmat(thresholds', mm, 1)) * theta));
  fprintf(1, 'Iter %d, empirical risk = %1.4f, empirical error = %1.4f\n', ...
          iter, sum(p_dist), sum(p_dist >= 1));
  p_dist = p_dist / sum(p_dist);
end

function [theta, feature_inds, thresholds] = random_booster(X, y, T)
% RANDOM_BOOSTER Uses random thresholds and indices to train a classifier
%
% [theta, feature_inds, thresholds] = random_booster(X, y, T)
%  performs T rounds of boosted decision stumps to classify the data X,
%  which is an m-by-n matrix of m training examples in dimension n.
%
%  The returned parameters are theta, the parameter vector in T dimensions,
%  the feature_inds, which are indices of the features (a T dimensional vector
```

```
%  taking values in {1, 2, ..., n}), and thresholds, which are real-valued
%  thresholds. The resulting classifier may be computed on an n-dimensional
%
%   theta' * sgn(x(feature_inds) - thresholds).

[mm, nn] = size(X);
p_dist = ones(mm, 1);
p_dist = p_dist / sum(p_dist);

theta = [];
feature_inds = [];
thresholds = [];

for iter = 1:T
  ind = ceil(rand * nn);
  thresh = X(ceil(rand * mm), ind) + 1e-8 * randn;
  Wplus = p_dist' * (sign(X(:, ind) - thresh) == y);
  Wminus = p_dist' * (sign(X(:, ind) - thresh) ~= y);
  theta = [theta; .5 * log(Wplus / Wminus)];
  feature_inds = [feature_inds; ind];
  thresholds = [thresholds; thresh];
  p_dist = exp(-y .* (...
    sign(X(:, feature_inds) - repmat(thresholds', mm, 1)) * theta));
  fprintf(1, 'Iter %d, empirical risk = %1.4f, empirical error = %1.4f\n', ...
          iter, sum(p_dist), sum(p_dist >= 1));
  p_dist = p_dist / sum(p_dist);
end

function s = sgn(v)
s = 2 * (v >= 0) - 1;
```

CS 229 Autumn 2016
Problem Set#3:Theory & Unsupervised learning

Due Wednesday, November 16 at 11:00 am on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout/scan of your code (with comments) and any figures that you are asked to plot.

If you are skipping a question, please include it on your PDF/photo, but leave the question blank and tag it appropriately on Gradescope. This includes extra credit problems. If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[23 points] Uniform convergence**

   You are hired by CNN to help design the sampling procedure for making their electoral predictions for the next presidential election in the (fictitious) country of Elbania.

   The country of Elbania is organized into states, and there are only two candidates running in this election: One from the Elbanian Democratic party, and another from the Labor Party of Elbania. The plan for making our electorial predictions is as follows: We'll sample m voters from each state, and ask whether they're voting democrat. We'll then publish, for each state, the estimated fraction of democrat voters. In this problem, we'll work out how many voters we need to sample in order to ensure that we get good predictions with high probability.

   One reasonable goal might be to set m large enough that, with high probability, we obtain uniformly accurate estimates of the fraction of democrat voters in every state. But this might require surveying very many people, which would be prohibitively expensive. So, we're instead going to demand only a slightly lower degree of accuracy.

   Specifically, we'll say that our prediction for a state is "highly inaccurate" if the estimated fraction of democrat voters differs from the actual fraction of democrat voters within that state by more than a tolerance factor γ. CNN knows that their viewers will tolerate some small number of states' estimates being highly inaccurate; however, their credibility would be damaged if they reported highly inaccurate estimates for too many states. So, rather than trying to ensure that all states' estimates are within γ of the true values (which would correspond to no state's estimate being highly inaccurate), we will instead try only to ensure that the number of states with highly inaccurate estimates is small.

   To formalize the problem, let there be n states, and let m voters be drawn IID from each state. Let the actual fraction of voters in state i that voted democrat be ϕ_i. Also let X_{ij} $(1 \leq i \leq n, 1 \leq j \leq m)$ be a binary random variable indicating whether the j-th randomly chosen voter from state i voted democrat:

   $$X_{ij} = \begin{cases} 1 & \text{if the } j^{th} \text{ example from the } i^{th} \text{ state voted democrat} \\ 0 & \text{otherwise} \end{cases}$$

We assume that the voters correctly disclose their vote during the survey. Thus, for each value of i, we have that X_{ij} are drawn IID from a Bernoulli(ϕ_i) distribution. Moreover, the X_{ij}'s (for all i, j) are all mutually independent.

After the survey, the fraction of democrat votes in state i is estimated as:

$$\hat{\phi}_i = \frac{1}{m} \sum_{j=1}^{m} X_{ij}$$

Also, let $Z_i = 1\{|\hat{\phi}_i - \phi_i| > \gamma\}$ be a binary random variable that indicates whether the prediction in state i was highly inaccurate.

(a) Let ψ_i be the probability that $Z_i = 1$. Using the Hoeffding inequality, find an upper bound on ψ_i.

(b) In this part, we prove a general result which will be useful for this problem. Let V_i and W_i $(1 \leq i \leq k)$ be Bernoulli random variables, and suppose

$$\mathrm{E}[V_i] = P(V_i = 1) \leq P(W_i = 1) = \mathrm{E}[W_i] \qquad \forall i \in \{1, 2, \ldots k\}$$

Let the V_i's be mutually independent, and similarly let the W_i's also be mutually independent. Prove that, for any value of t, the following holds:

$$P\left(\sum_{i=1}^{k} V_i > t\right) \leq P\left(\sum_{i=1}^{k} W_i > t\right)$$

[Hint: One way to do this is via induction on k. If you use a proof by induction, for the base case ($k = 1$), you must show that the inequality holds for $t < 0$, $0 \leq t < 1$, and $t \geq 1$.]

(c) The fraction of states on which our predictions are highly inaccurate is given by $\overline{Z} = \frac{1}{n} \sum_{i=1}^{n} Z_i$. Prove a reasonable closed form upper bound on the probability $P(\overline{Z} > \tau)$ of being highly inaccurate on more than a fraction τ of the states.

[Note: There are many possible answers, but to be considered reasonable, your bound must decrease to zero as $m \to \infty$ (for fixed n and $\tau > 0$). Also, your bound should either remain constant or decrease as $n \to \infty$ (for fixed m and $\tau > 0$). It is also fine if, for some values of τ, m and n, your bound just tells us that $P(\overline{Z} > \tau) \leq 1$ (the trivial bound).]

2. **[15 points] More VC dimension**

Let the domain of the inputs for a learning problem be $\mathcal{X} = \mathbb{R}$. Consider using hypotheses of the following form:

$$h_\theta(x) = 1\{\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d \geq 0\},$$

and let $\mathcal{H} = \{h_\theta : \theta \in \mathbb{R}^{d+1}\}$ be the corresponding hypothesis class. What is the VC dimension of $\mathcal{H}$? Justify your answer.

[Hint: You may use the fact that a polynomial of degree d has at most d real roots. When doing this problem, you should not assume any other non-trivial result (such as that the VC dimension of linear classifiers in d-dimensions is $d + 1$) that was not formally proved in class.]

3. **[12 points] MAP estimates and weight decay**

Consider using a logistic regression model $h_\theta(x) = g(\theta^T x)$ where g is the sigmoid function, and let a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ be given as usual. The maximum likelihood estimate of the parameters θ is given by

$$\theta_{\text{ML}} = \arg\max_\theta \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta).$$

If we wanted to regularize logistic regression, then we might put a Bayesian prior on the parameters. Suppose we chose the prior $\theta \sim \mathcal{N}(0, \tau^2 I)$ (here, $\tau > 0$, and I is the $n+1$-by-$n+1$ identity matrix), and then found the MAP estimate of θ as:

$$\theta_{\text{MAP}} = \arg\max_\theta p(\theta) \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta)$$

Prove that

$$||\theta_{\text{MAP}}||_2 \leq ||\theta_{\text{ML}}||_2$$

[Hint: Consider using a proof by contradiction.]
Remark. For this reason, this form of regularization is sometimes also called **weight decay**, since it encourages the weights (meaning parameters) to take on generally smaller values.

4. **[15 points] KL divergence and Maximum Likelihood**

The Kullback-Leibler (KL) divergence between two discrete-valued distributions $P(X), Q(X)$ is defined as follows:[1]

$$KL(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

For notational convenience, we assume $P(x) > 0, \forall x$. (Otherwise, one standard thing to do is to adopt the convention that "$0 \log 0 = 0$.") Sometimes, we also write the KL divergence as $KL(P||Q) = KL(P(X)||Q(X))$.

The KL divergence is an assymmetric measure of the distance between 2 probability distributions. In this problem we will prove some basic properties of KL divergence, and work out a relationship between minimizing KL divergence and the maximum likelihood estimation that we're familiar with.

(a) Nonnegativity. Prove the following:

$$\forall P, Q \quad KL(P\|Q) \geq 0$$

and

$$KL(P\|Q) = 0 \quad \text{if and only if } P = Q.$$

[1] If P and Q are densities for continuous-valued random variables, then the sum is replaced by an integral, and everything stated in this problem works fine as well. But for the sake of simplicity, in this problem we'll just work with this form of KL divergence for probability mass functions/discrete-valued distributions.

[Hint: You may use the following result, called **Jensen's inequality**. If f is a convex function, and X is a random variable, then $E[f(X)] \geq f(E[X])$. Moreover, if f is strictly convex (f is convex if its Hessian satisfies $H \geq 0$; it is *strictly* convex if $H > 0$; for instance $f(x) = -\log x$ is strictly convex), then $E[f(X)] = f(E[X])$ implies that $X = E[X]$ with probability 1; i.e., X is actually a constant.]

(b) **Chain rule for KL divergence.** The KL divergence between 2 conditional distributions $P(X|Y), Q(X|Y)$ is defined as follows:

$$KL(P(X|Y)\|Q(X|Y)) = \sum_y P(y) \left(\sum_x P(x|y) \log \frac{P(x|y)}{Q(x|y)} \right)$$

This can be thought of as the expected KL divergence between the corresponding conditional distributions on x (that is, between $P(X|Y = y)$ and $Q(X|Y = y)$), where the expectation is taken over the random y.

Prove the following chain rule for KL divergence:

$$KL(P(X,Y)\|Q(X,Y)) = KL(P(X)\|Q(X)) + KL(P(Y|X)\|Q(Y|X)).$$

(c) **KL and maximum likelihood.**

Consider a density estimation problem, and suppose we are given a training set $\{x^{(i)}; i = 1, \ldots, m\}$. Let the empirical distribution be $\hat{P}(x) = \frac{1}{m} \sum_{i=1}^{m} 1\{x^{(i)} = x\}$. ($\hat{P}$ is just the uniform distribution over the training set; i.e., sampling from the empirical distribution is the same as picking a random example from the training set.)

Suppose we have some family of distributions P_θ parameterized by θ. (If you like, think of $P_\theta(x)$ as an alternative notation for $P(x; \theta)$.) Prove that finding the maximum likelihood estimate for the parameter θ is equivalent to finding P_θ with minimal KL divergence from $\hat{P}$. I.e. prove:

$$\arg\min_\theta KL(\hat{P}\|P_\theta) = \arg\max_\theta \sum_{i=1}^{m} \log P_\theta(x^{(i)})$$

Remark. Consider the relationship between parts (b-c) and multi-variate Bernoulli Naive Bayes parameter estimation. In the Naive Bayes model we assumed P_θ is of the following form: $P_\theta(x, y) = p(y) \prod_{i=1}^{n} p(x_i|y)$. By the chain rule for KL divergence, we therefore have:

$$KL(\hat{P}\|P_\theta) = KL(\hat{P}(y)\|p(y)) + \sum_{i=1}^{n} KL(\hat{P}(x_i|y)\|p(x_i|y)).$$

This shows that finding the maximum likelihood/minimum KL-divergence estimate of the parameters decomposes into $2n + 1$ independent optimization problems: One for the class priors $p(y)$, and one for each of the conditional distributions $p(x_i|y)$ for each feature x_i given each of the two possible labels for y. Specifically, finding the maximum likelihood estimates for each of these problems individually results in also maximizing the likelihood of the joint distribution. (If you know what Bayesian networks are, a similar remark applies to parameter estimation for them.)

5. **[20 points] K-means for compression**

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image.

We will be using the following files:

- `http://cs229.stanford.edu/ps/ps3/mandrill-small.tiff`
- `http://cs229.stanford.edu/ps/ps3/mandrill-large.tiff`

The `mandrill-large.tiff` file contains a 512x512 image of a mandrill represented in 24-bit color. This means that, for each of the 262144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262144 \times 3 = 786432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $k = 16$ colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)![2]

(a) Start up MATLAB, and type `A = double(imread('mandrill-large.tiff'));` to read in the image. Now, `A` is a "three dimensional matrix," and `A(:,:,1)`, `A(:,:,2)` and `A(:,:,3)` are 512x512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `imshow(uint8(round(A)));` to display the image.

(b) Since the large image has 262144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat (a) with `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, run K-means[3] with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

(c) Take the matrix `A` from `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid. Display the new image, and compare it visually to the original image. Hand in all your code and a printout of your compressed image (printing on a black-and-white printer is fine).

(d) If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

[2] In order to use the `imread` and `imshow` commands in octave, you have to install the Image package from octave-forge. This package and installation instructions are available at: `http://octave.sourceforge.net`

[3] Please implement K-means yourself, rather than using built-in functions from, e.g., MATLAB or octave.

CS 229 Autumn 2016
Problem Set #3 Solutions: Theory & Unsupervised learning

Due Wednesday, May 18 at 11:00 pm on Gradescope.

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are skipping a question, please include it on your PDF/photo, but leave the question blank and tag it appropriately on Gradescope. This includes extra credit problems. If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[23 points] Uniform convergence**

   You are hired by CNN to help design the sampling procedure for making their electoral predictions for the next presidential election in the (fictitious) country of Elbania.

   The country of Elbania is organized into states, and there are only two candidates running in this election: One from the Elbanian Democratic party, and another from the Labor Party of Elbania. The plan for making our electorial predictions is as follows: We'll sample m voters from each state, and ask whether they're voting democrat. We'll then publish, for each state, the estimated fraction of democrat voters. In this problem, we'll work out how many voters we need to sample in order to ensure that we get good predictions with high probability.

   One reasonable goal might be to set m large enough that, with high probability, we obtain uniformly accurate estimates of the fraction of democrat voters in every state. But this might require surveying very many people, which would be prohibitively expensive. So, we're instead going to demand only a slightly lower degree of accuracy.

   Specifically, we'll say that our prediction for a state is "highly inaccurate" if the estimated fraction of democrat voters differs from the actual fraction of democrat voters within that state by more than a tolerance factor γ. CNN knows that their viewers will tolerate some small number of states' estimates being highly inaccurate; however, their credibility would be damaged if they reported highly inaccurate estimates for too many states. So, rather than trying to ensure that all states' estimates are within γ of the true values (which would correspond to no state's estimate being highly inaccurate), we will instead try only to ensure that the number of states with highly inaccurate estimates is small.

   To formalize the problem, let there be n states, and let m voters be drawn IID from each state. Let the actual fraction of voters in state i that voted democrat be ϕ_i. Also let X_{ij} $(1 \leq i \leq n, 1 \leq j \leq m)$ be a binary random variable indicating whether the j-th randomly

chosen voter from state i voted democrat:

$$X_{ij} = \begin{cases} 1 & \text{if the } j^{th} \text{ example from the } i^{th} \text{ state voted democrat} \\ 0 & \text{otherwise} \end{cases}$$

We assume that the voters correctly disclose their vote during the survey. Thus, for each value of i, we have that X_{ij} are drawn IID from a Bernoulli(ϕ_i) distribution. Moreover, the X_{ij}'s (for all i, j) are all mutually independent.

After the survey, the fraction of democrat votes in state i is estimated as:

$$\hat{\phi}_i = \frac{1}{m} \sum_{j=1}^{m} X_{ij}$$

Also, let $Z_i = 1\{|\hat{\phi}_i - \phi_i| > \gamma\}$ be a binary random variable that indicates whether the prediction in state i was highly inaccurate.

(a) Let ψ_i be the probability that $Z_i = 1$. Using the Hoeffding inequality, find an upper bound on ψ_i.

Answer: A direct application of the Hoeffding inequality yields

$$\psi_i \leq 2e^{-2\gamma^2 m}$$

(b) In this part, we prove a general result which will be useful for this problem. Let V_i and W_i $(1 \leq i \leq k)$ be Bernoulli random variables, and suppose

$$\mathrm{E}[V_i] = P(V_i = 1) \leq P(W_i = 1) = \mathrm{E}[W_i] \qquad \forall i \in \{1, 2, \ldots k\}$$

Let the V_i's be mutually independent, and similarly let the W_i's also be mutually independent. Prove that, for any value of t, the following holds:

$$P\left(\sum_{i=1}^{k} V_i > t\right) \leq P\left(\sum_{i=1}^{k} W_i > t\right)$$

[Hint: One way to do this is via induction on k. If you use a proof by induction, for the base case ($k = 1$), you must show that the inequality holds for $t < 0$, $0 \leq t < 1$, and $t \geq 1$.]

Answer: Prove it by induction.

Base case: Show

$$P\left(V_1 > t\right) \leq P\left(W_1 > t\right)$$

If $t < 0$, then both probalities are 1. If $t \geq 1$, then both probabilities are 0. Otherwise, the equation reduces to

$$P\left(V_1 = 1\right) \leq P\left(W_1 = 1\right)$$

which holds by our original assumptions.

Inductive step: Assume

$$P\left(\sum_{i=1}^{l} V_i > t\right) \leq P\left(\sum_{i=1}^{l} W_i > t\right), \forall t$$

Then,

$$\begin{aligned}
&P\left(\sum_{i=1}^{l+1} V_i > t\right) \\
&\quad = P\left(V_{l+1}=1\right) P\left(\sum_{i=1}^{l+1} V_i > t \middle| V_{l+1}=1\right) + P\left(V_{l+1}=0\right) P\left(\sum_{i=1}^{l+1} V_i > t \middle| V_{l+1}=0\right) \\
&\quad = P\left(V_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t-1 \middle| V_{l+1}=1\right) + P\left(V_{l+1}=0\right) P\left(\sum_{i=1}^{l} V_i > t \middle| V_{l+1}=0\right) \\
&\quad = P\left(V_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t-1\right) + P\left(V_{l+1}=0\right) P\left(\sum_{i=1}^{l} V_i > t\right) \\
&\quad = P\left(V_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t-1\right) + \left(1 - P\left(V_{l+1}=1\right)\right) P\left(\sum_{i=1}^{l} V_i > t\right) \\
&\quad = P\left(V_{l+1}=1\right) \left(P\left(\sum_{i=1}^{l} V_i > t-1\right) - P\left(\sum_{i=1}^{l} V_i > t\right)\right) + P\left(\sum_{i=1}^{l} V_i > t\right) \\
&\quad \leq P\left(W_{l+1}=1\right) \left(P\left(\sum_{i=1}^{l} V_i > t-1\right) - P\left(\sum_{i=1}^{l} V_i > t\right)\right) + P\left(\sum_{i=1}^{l} V_i > t\right) \\
&\quad = P\left(W_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t-1\right) + \left(1 - P\left(W_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t\right)\right. \\
&\quad = P\left(W_{l+1}=1\right) P\left(\sum_{i=1}^{l} V_i > t-1\right) + P\left(W_{l+1}=0\right) P\left(\sum_{i=1}^{l} V_i > t\right) \\
&\quad \leq P\left(W_{l+1}=1\right) P\left(\sum_{i=1}^{l} W_i > t-1\right) + P\left(W_{l+1}=0\right) P\left(\sum_{i=1}^{l} W_i > t\right) \\
&\quad = P\left(\sum_{i=1}^{l+1} W_i > t\right).
\end{aligned}$$

And the result is proved.

A second, and completely different way to prove this result, is by what is known as *coupling*. In particular, let $\xi_1, \xi_2, \ldots, \xi_k$ be i.i.d. random variables, each uniform on $[0,1]$. Define the variables

$$V_i^u = \begin{cases} 0 & \text{if } \xi_i > P(V_i=1) \\ 1 & \text{if } \xi_i \leq P(V_i=1). \end{cases} \quad \text{and} \quad W_i^u = \begin{cases} 0 & \text{if } \xi_i > P(W_i=1) \\ 1 & \text{if } \xi_i \leq P(W_i=1). \end{cases}$$

Then $\mathrm{E}[V_i^u] = P(\xi_i \leq P(V_i=1)) = P(V_i=1) = \mathrm{E}[V_i]$ and similarly $\mathrm{E}[W_i^u] = \mathrm{E}[W_i]$. But $W_i^u \geq V_i^u$ always and the sequence $\{V_1^u, \ldots, V_k^u\}$ is i.i.d. (and similarly for W_i^u).

Consequently, we have

$$P\Big(\sum_{i=1}^{k} V_i > t\Big) = P\Big(\sum_{i=1}^{k} V_i^u > t\Big) \quad \leq P\Big(\sum_{i=1}^{k} W_i^u > t\Big) = P\Big(\sum_{i=1}^{k} W_i > t\Big),$$

where the inequality follows because

$$\sum_{i=1}^{k} W_i^u \geq \sum_{i=1}^{k} V_i^u,$$

so that $\sum_{i=1}^{k} V_i^u > t$ implies $\sum_{i=1}^{k} W_i^u > t$.

(c) The fraction of states on which our predictions are highly inaccurate is given by $\overline{Z} = \frac{1}{n}\sum_{i=1}^{n} Z_i$. Prove a reasonable closed form upper bound on the probability $P(\overline{Z} > \tau)$ of being highly inaccurate on more than a fraction τ of the states.

[Note: There are many possible answers, but to be considered reasonable, your bound must decrease to zero as $m \to \infty$ (for fixed n and $\tau > 0$). Also, your bound should either remain constant or decrease as $n \to \infty$ (for fixed m and $\tau > 0$). It is also fine if, for some values of τ, m and n, your bound just tells us that $P(\overline{Z} > \tau) \leq 1$ (the trivial bound).]

Answer: There are multiple ways to do this problem. We list a couple of them below:

Using Chernoff's inequality

Let Y_i be new Bernoulli random variables with mean $\mu = 2e^{-2\gamma^2 m}$. Then we know from part (a) that $P(Z_i = 1) \leq \mu = P(Y_i = 1)$. Using the result from the previous part:

$$\begin{aligned} P(\overline{Z} > \tau) &\leq P\left(\frac{1}{n}\sum_{i=0}^{n} Y_i > \tau\right) \\ &= P\left(\frac{1}{n}\sum_{i=0}^{n} Y_i - \mu > \tau - \mu\right) \\ &\leq P\left(\left|\frac{1}{n}\sum_{i=0}^{n} Y_i - \mu\right| > \tau - \mu\right) \\ &\leq 2\exp\left(-2(\tau-\mu)^2 n\right), \end{aligned}$$

where the last step follows provided that $0 < \tau - \mu = \tau - 2e^{-2\gamma^2 m}$, or equivalently, $m > \frac{1}{2\gamma^2}\log\left(\frac{2}{\tau}\right)$. For fixed τ and m, this bound goes to zero as $n \to \infty$. Alternatively,

we can also just compute the right side directly, as in

$$\begin{aligned} P(\overline{Z} > \tau) &\leq P\left(\frac{1}{n}\sum_{i=0}^{n} Y_i > \tau\right) \\ &= P\left(\sum_{i=0}^{n} Y_i > n\tau\right) \\ &= \sum_{j=k}^{n} P\left(\sum_{i=0}^{n} Y_i = j\right) \\ &= \sum_{j=k}^{n} \binom{n}{j} \mu^j (1-\mu)^{1-j} \\ &\leq \sum_{j=k}^{n} \binom{n}{j} \mu^j \end{aligned}$$

where k is the smallest integer such that $k > n\tau$. For fixed τ and n, observe that as $m \to \infty$, $\mu \to 0$, so this bound goes to zero. Therefore,

$$P(\overline{Z} > \tau) \leq \min\left\{1, 2e^{-2(\tau-\mu)^2 n}, \sum_{j=k}^{n} \binom{n}{j} \mu^j\right\}$$

has the properties we want.

Using Markov's inequality

Markov's inequality states that for any nonnegative random variable X and $\tau > 0$, then $P(X > \tau) \leq \frac{E[X]}{\tau}$. From part (a), we have $E[Z_i] = P(Z_i = 1) \leq 2e^{-2\gamma^2 m}$, implying that

$$\begin{aligned} P(\overline{Z} > \tau) &= P\left(\frac{1}{n}\sum_{i=0}^{n} Z_i > \tau\right) \\ &\leq \frac{E\left[\frac{1}{n}\sum_{i=0}^{n} Z_i\right]}{\tau} \\ &\leq \frac{2}{\tau} e^{-2\gamma^2 m}. \end{aligned}$$

This bound satisfies the given requirements: as $m \to \infty$, the bound goes to zero; if $n \to \infty$, the bound stays constant.

Using Chebyshev's inequality

Chebyshev's inequality states that for any random variable X with expected value μ and finite variance σ^2, then for any constant $\tau > 0$, $P(|X - \mu| > \tau) \leq \frac{\sigma^2}{\tau}$. Let Y_i be new Bernoulli random variables with mean $\mu = 2e^{-2\gamma^2 m}$. Then we know from part (a) that

$P(Z_i = 1) \leq \mu = P(Y_i = 1)$. Using the result from the previous part:

$$\begin{aligned}
P(\overline{Z} > \tau) &\leq P\left(\frac{1}{n}\sum_{i=0}^{n} Y_i > \tau\right) \\
&= P\left(\frac{1}{n}\sum_{i=0}^{n} Y_i - \mu > \tau - \mu\right) \\
&\leq P\left(\left|\frac{1}{n}\sum_{i=0}^{n} Y_i - \mu\right| > \tau - \mu\right) \\
&\leq \frac{\mathsf{Var}\left[\frac{1}{n}\sum_{i=0}^{n} Y_i\right]}{(\tau-\mu)^2} \\
&= \frac{\frac{1}{n^2}\sum_{i=0}^{n} \mathsf{Var}\left[Y_i\right]}{(\tau-\mu)^2} \\
&= \frac{2e^{-2\gamma^2 m}(1 - 2e^{-2\gamma^2 m})}{n(\tau-\mu)^2} \\
&\leq \frac{2e^{-2\gamma^2 m}}{n(\tau-\mu)^2},
\end{aligned}$$

where we again require that $m > \frac{1}{2\gamma^2}\log\left(\frac{2}{\tau}\right)$. This version of the bound goes to zero both when $m \to \infty$ and when $n \to \infty$.

2. **[15 points] More VC dimension**

Let the domain of the inputs for a learning problem be $\mathcal{X} = \mathbb{R}$. Consider using hypotheses of the following form:

$$h_\theta(x) = 1\{\theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_d x^d \geq 0\},$$

and let $\mathcal{H} = \{h_\theta : \theta \in \mathbb{R}^{d+1}\}$ be the corresponding hypothesis class. What is the VC dimension of $\mathcal{H}$? Justify your answer.

[Hint: You may use the fact that a polynomial of degree d has at most d real roots. When doing this problem, you should not assume any other non-trivial result (such as that the VC dimension of linear classifiers in d-dimensions is $d + 1$) that was not formally proved in class.]

Answer: The key insight is that if the polynomial does not cross the x-axis (i.e. have a root) between two points, then it must give the two points the same label.

First, we need to show that there is a set of size $d + 1$ which $\mathcal{H}$ can shatter. We consider polynomials with d real roots. A subset of the polynomials in $\mathcal{H}$ can be written as

$$\pm\prod_{i=1}^{d}(x - r_i)$$

where r_i is the i^{th} real root. Consider any set of size $d+1$ which does not contain any duplicate points. For any labelling of these points, construct a function as follows: If two consecutive points are labelled differently, set one of the r_i to the average of those points. If two consecutive

points are labelled the same, don't put a root between them. If we haven't used up all of our d roots, place them beyond the last point. Finally, choose $\pm$ to get the desired labelling.

A more constructive proof of the above is the following: consider any set of distinct points $x^{(1)}, \ldots, x^{(d+1)}$, and let $y^{(1)}, \ldots, y^{(d+1)} \in \{-1, 1\}$ be any labeling of these points (where we have used -1 for points which would normally be labeled zero). Then, consider the following polynomial:

$$p(x) = \sum_{k=1}^{d+1} y^{(k)} \prod_{j \neq k} \left(\frac{x^{(j)} - x}{x^{(j)} - x^{(k)}} \right).$$

Here, observe that in the above expression, each term of the summation is a polynomial (in x) of degree d, and hence the overall expression is a polynomial of degree d. Furthermore, observe that when $x = x^{(i)}$, then the ith term of the summation evaluates to $y^{(i)}$, and all other terms of the summation evaluate to 0 (since all other terms have a factor $(x^{(i)} - x)$). Therefore, $p(x^{(i)}) = y^{(i)}$ for $i = 1, \ldots, d+1$. This construction is known as a "Lagrange interpolating polynomial." Therefore, any labeling of $d+1$ points can be realized using a degree d polynomial.

Second, we need to prove that $\mathcal{H}$ can't shatter a set of size $d+2$. If two points are identical, we can't realize any labelling that labels them differently. If all points are unique, we can't achieve an alternating labelling because we would need $d+1$ roots.

3. **[12 points] MAP estimates and weight decay**

Consider using a logistic regression model $h_\theta(x) = g(\theta^T x)$ where g is the sigmoid function, and let a training set $\{(x^{(i)}, y^{(i)}); i = 1, \ldots, m\}$ be given as usual. The maximum likelihood estimate of the parameters θ is given by

$$\theta_{\mathrm{ML}} = \arg\max_\theta \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}; \theta).$$

If we wanted to regularize logistic regression, then we might put a Bayesian prior on the parameters. Suppose we chose the prior $\theta \sim \mathcal{N}(0, \tau^2 I)$ (here, $\tau > 0$, and I is the $n+1$-by-$n+1$ identity matrix), and then found the MAP estimate of θ as:

$$\theta_{\mathrm{MAP}} = \arg\max_\theta p(\theta) \prod_{i=1}^{m} p(y^{(i)} | x^{(i)}, \theta)$$

Prove that

$$||\theta_{\mathrm{MAP}}||_2 \leq ||\theta_{\mathrm{ML}}||_2$$

[Hint: Consider using a proof by contradiction.]
Remark. For this reason, this form of regularization is sometimes also called **weight decay**, since it encourages the weights (meaning parameters) to take on generally smaller values.

Answer: Assume that

$$||\theta_{\mathrm{MAP}}||_2 > ||\theta_{\mathrm{ML}}||_2$$

Then, we have that

$$\begin{aligned} p(\theta_{\text{MAP}}) &= \frac{1}{(2\pi)^{\frac{n+1}{2}}|\tau^2 I|^{\frac{1}{2}}} e^{-\frac{1}{2\tau^2}(||\theta_{\text{MAP}}||_2)^2} \\ &< \frac{1}{(2\pi)^{\frac{n+1}{2}}|\tau^2 I|^{\frac{1}{2}}} e^{-\frac{1}{2\tau^2}(||\theta_{\text{ML}}||_2)^2} \\ &= p(\theta_{\text{ML}}) \end{aligned}$$

This yields

$$\begin{aligned} p(\theta_{\text{MAP}}) \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta_{\text{MAP}}) &< p(\theta_{\text{ML}}) \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta_{\text{MAP}}) \\ &\leq p(\theta_{\text{ML}}) \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta_{\text{ML}}) \end{aligned}$$

where the last inequality holds since θ_{ML} was chosen to maximize $\prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta)$. However, this result gives us a contradiction, since θ_{MAP} was chosen to maximize $\prod_{i=1}^{m} p(y^{(i)}|x^{(i)}, \theta)p(\theta)$

4. **[15 points] KL divergence and Maximum Likelihood**

   The Kullback-Leibler (KL) divergence between two discrete-valued distributions $P(X), Q(X)$ is defined as follows:[1]

   $$KL(P\|Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

   For notational convenience, we assume $P(x) > 0, \forall x$. (Otherwise, one standard thing to do is to adopt the convention that "$0 \log 0 = 0$.") Sometimes, we also write the KL divergence as $KL(P||Q) = KL(P(X)||Q(X))$.

   The KL divergence is an assymmetric measure of the distance between 2 probability distributions. In this problem we will prove some basic properties of KL divergence, and work out a relationship between minimizing KL divergence and the maximum likelihood estimation that we're familiar with.

   (a) Nonnegativity. Prove the following:

   $$\forall P, Q \quad KL(P\|Q) \geq 0$$

   and

   $$KL(P\|Q) = 0 \quad \text{if and only if } P = Q.$$

   [Hint: You may use the following result, called **Jensen's inequality**. If f is a convex function, and X is a random variable, then $E[f(X)] \geq f(E[X])$. Moreover, if f is

[1] If P and Q are densities for continuous-valued random variables, then the sum is replaced by an integral, and everything stated in this problem works fine as well. But for the sake of simplicity, in this problem we'll just work with this form of KL divergence for probability mass functions/discrete-valued distributions.

strictly convex (f is convex if its Hessian satisfies $H \geq 0$; it is *strictly* convex if $H > 0$; for instance $f(x) = -\log x$ is strictly convex), then $E[f(X)] = f(E[X])$ implies that $X = E[X]$ with probability 1; i.e., X is actually a constant.]

Answer:

$$\begin{aligned}
-KL(P\|Q) &= -\sum_x P(x) \log \frac{P(x)}{Q(x)} && (1)\\
&= \sum_x P(x) \log \frac{Q(x)}{P(x)} && (2)\\
&\leq \log \sum_x P(x) \frac{Q(x)}{P(x)} && (3)\\
&= \log \sum_x Q(x) && (4)\\
&= \log 1 && (5)\\
&= 0 && (6)
\end{aligned}$$

Where all equalities follow from straight forward algebraic manipulation. The inequality follows from Jensen's inequality.

To show the second part of the claim, note that $\log t$ is a strictly concave function of t. Using the form of Jensen's inequality given in the lecture notes, we have equality if and only if $\frac{Q(x)}{P(x)} = E[\frac{Q(x)}{P(x)}]$ for all x. But since $E[\frac{Q(x)}{P(x)}] = \sum_x P(x)\frac{Q(x)}{P(x)} = \sum_x Q(x) = 1$, it follows that $P(x) = Q(x)$. Hence we have $KL(P\|Q) = 0$ if and only if $P(x) = Q(x)$ for all x.

(b) **Chain rule for KL divergence.** The KL divergence between 2 conditional distributions $P(X|Y), Q(X|Y)$ is defined as follows:

$$KL(P(X|Y)\|Q(X|Y)) = \sum_y P(y) \left(\sum_x P(x|y) \log \frac{P(x|y)}{Q(x|y)} \right)$$

This can be thought of as the expected KL divergence between the corresponding conditional distributions on x (that is, between $P(X|Y = y)$ and $Q(X|Y = y)$), where the expectation is taken over the random y.

Prove the following chain rule for KL divergence:

$$KL(P(X,Y)\|Q(X,Y)) = KL(P(X)\|Q(X)) + KL(P(Y|X)\|Q(Y|X)).$$

Answer:

$$\begin{aligned}
KL(P(X,Y)\|Q(X,Y)) &= \sum_{x,y} P(x,y)\log\frac{P(x,y)}{Q(x,y)} && (7)\\
&= \sum_{x,y} P(x,y)\log\frac{P(x)P(y|x)}{Q(x)Q(y|x)} && (8)\\
&= \sum_{x,y} P(x,y)\log\frac{P(x)}{Q(x)} + P(x,y)\log\frac{P(y|x)}{Q(y|x)} && (9)\\
&= \sum_{x,y} P(x,y)\log\frac{P(x)}{Q(x)} + \sum_{x,y} P(x)P(y|x)\log\frac{P(y|x)}{Q(y|x)} && (10)\\
&= \sum_{x} P(x)\log\frac{P(x)}{Q(x)} + \sum_{x} P(x)\sum_{y} P(y|x)\log\frac{P(y|x)}{Q(y|x)} && (11)\\
&= KL(P(X)\|Q(X)) && (12)\\
&\quad + KL(P(Y|X)\|Q(Y|X)). && (13)
\end{aligned}$$

Where we applied (in order): definition of KL, definition of conditional probability, log of product is sum of logs, splitting the summation, $\sum_y P(x,y) = P(x)$, definition of KL.

(c) **KL and maximum likelihood.**

Consider a density estimation problem, and suppose we are given a training set $\{x^{(i)}; i = 1,\ldots,m\}$. Let the empirical distribution be $\hat{P}(x) = \frac{1}{m}\sum_{i=1}^{m} 1\{x^{(i)} = x\}$. ($\hat{P}$ is just the uniform distribution over the training set; i.e., sampling from the empirical distribution is the same as picking a random example from the training set.)

Suppose we have some family of distributions P_θ parameterized by θ. (If you like, think of $P_\theta(x)$ as an alternative notation for $P(x;\theta)$.) Prove that finding the maximum likelihood estimate for the parameter θ is equivalent to finding P_θ with minimal KL divergence from $\hat{P}$. I.e. prove:

$$\arg\min_\theta KL(\hat{P}\|P_\theta) = \arg\max_\theta \sum_{i=1}^{m} \log P_\theta(x^{(i)})$$

Remark. Consider the relationship between parts (b-c) and multi-variate Bernoulli Naive Bayes parameter estimation. In the Naive Bayes model we assumed P_θ is of the following form: $P_\theta(x,y) = p(y)\prod_{i=1}^{n} p(x_i|y)$. By the chain rule for KL divergence, we therefore have:

$$KL(\hat{P}\|P_\theta) = KL(\hat{P}(y)\|p(y)) + \sum_{i=1}^{n} KL(\hat{P}(x_i|y)\|p(x_i|y)).$$

This shows that finding the maximum likelihood/minimum KL-divergence estimate of the parameters decomposes into $2n+1$ independent optimization problems: One for the class priors $p(y)$, and one for each of the conditional distributions $p(x_i|y)$ for each feature x_i given each of the two possible labels for y. Specifically, finding the maximum likelihood estimates for each of these problems individually results in also maximizing the likelihood of the joint distribution. (If you know what Bayesian networks are, a similar remark applies to parameter estimation for them.)

Answer:

$$\begin{aligned}
\arg\min_\theta KL(\hat{P}\|P_\theta) &= \arg\min_\theta \sum_x \hat{P}(x)\log\hat{P}(x) - \hat{P}(x)\log P_\theta(x) && (14)\\
&= \arg\min_\theta \sum_x -\hat{P}(x)\log P_\theta(x) && (15)\\
&= \arg\max_\theta \sum_x \hat{P}(x)\log P_\theta(x) && (16)\\
&= \arg\max_\theta \sum_x \frac{1}{m}\sum_{i=1}^m 1\{x^{(i)}=x\}\log P_\theta(x) && (17)\\
&= \arg\max_\theta \frac{1}{m}\sum_{i=1}^m\sum_x 1\{x^{(i)}=x\}\log P_\theta(x) && (18)\\
&= \arg\max_\theta \frac{1}{m}\sum_{i=1}^m \log P_\theta(x^{(i)}) && (19)\\
&= \arg\max_\theta \sum_{i=1}^m \log P_\theta(x^{(i)}) && (20)
\end{aligned}$$

where we used in order: definition of KL, leaving out terms independent of θ, flip sign and correspondingly flip min-max, definition of $\hat{P}$, switching order of summation, definition of the indicator and simplification.

5. **[20 points] K-means for compression**

In this problem, we will apply the K-means algorithm to lossy image compression, by reducing the number of colors used in an image.

We will be using the following files:

- `http://cs229.stanford.edu/ps/ps3/mandrill-small.tiff`
- `http://cs229.stanford.edu/ps/ps3/mandrill-large.tiff`

The `mandrill-large.tiff` file contains a 512x512 image of a mandrill represented in 24-bit color. This means that, for each of the 262144 pixels in the image, there are three 8-bit numbers (each ranging from 0 to 255) that represent the red, green, and blue intensity values for that pixel. The straightforward representation of this image therefore takes about $262144 \times 3 = 786432$ bytes (a byte being 8 bits). To compress the image, we will use K-means to reduce the image to $k = 16$ colors. More specifically, each pixel in the image is considered a point in the three-dimensional (r, g, b)-space. To compress the image, we will cluster these points in color-space into 16 clusters, and replace each pixel with the closest cluster centroid.

Follow the instructions below. Be warned that some of these operations can take a while (several minutes even on a fast computer)![2]

(a) Start up MATLAB, and type `A = double(imread('mandrill-large.tiff'));` to read in the image. Now, `A` is a "three dimensional matrix," and `A(:,:,1)`, `A(:,:,2)` and `A(:,:,3)` are 512x512 arrays that respectively contain the red, green, and blue values for each pixel. Enter `imshow(uint8(round(A)));` to display the image.

(b) Since the large image has 262144 pixels and would take a while to cluster, we will instead run vector quantization on a smaller image. Repeat (a) with `mandrill-small.tiff`. Treating each pixel's (r, g, b) values as an element of $\mathbb{R}^3$, run K-means[3] with 16 clusters on the pixel data from this smaller image, iterating (preferably) to convergence, but in no case for less than 30 iterations. For initialization, set each cluster centroid to the (r, g, b)-values of a randomly chosen pixel in the image.

(c) Take the matrix `A` from `mandrill-large.tiff`, and replace each pixel's (r, g, b) values with the value of the closest cluster centroid. Display the new image, and compare it visually to the original image. Hand in all your code and a printout of your compressed image (printing on a black-and-white printer is fine).

(d) If we represent the image with these reduced (16) colors, by (approximately) what factor have we compressed the image?

Answer: Figure **??** shows the original image of the mandrill. Figure **??** shows the image compressed into 16 colors using K-means run to convergence, and shows the 16 colors used in the compressed image. (These solutions are given in a color PostScript file. To see the colors without a color printer, view them with a program that can display color PostScript, such as ghostview.) The original image used 24 bits per pixel. To represent one of 16 colors requires $log_2 16 = 4$ bits per pixel. We have therefore achieved a compression factor of about $24/4 = 6$ of the image. MATLAB code for this problem is given below.

[2] In order to use the `imread` and `imshow` commands in octave, you have to install the Image package from octave-forge. This package and installation instructions are available at: `http://octave.sourceforge.net`

[3] Please implement K-means yourself, rather than using built-in functions from, e.g., MATLAB or octave.

```
A = double(imread('mandrill-small.tiff'));
imshow(uint8(round(A)));

% K-means initialization
k = 16;
initmu = zeros(k,3);
for l=1:k,
    i = random('unid', size(A, 1), 1, 1);
    j = random('unid', size(A, 2), 1, 1);
    initmu(l,:) = double(permute(A(i,j,:), [3 2 1])');
end;

% Run K-means
mu = initmu;
for iter = 1:200, % usually converges long before 200 iterations
    newmu = zeros(k,3);
    nassign = zeros(k,1);
    for i=1:size(A,1),
        for j=1:size(A,2),
            dist = zeros(k,1);
            for l=1:k,
                d = mu(l,:)'-permute(A(i,j,:), [3 2 1]);
                dist(l) = d'*d;
            end;
            [value, assignment] = min(dist);
            nassign(assignment) = nassign(assignment) + 1;
            newmu(assignment,:) = newmu(assignment,:) + ...
                permute(A(i,j,:), [3 2 1])';
        end; end;
    for l=1:k,
        if (nassign(l) > 0)
            newmu(l,:) = newmu(l,:) / nassign(l);
        end;
    end;
    mu = newmu;
end;

% Assign new colors to large image
bigimage = double(imread('mandrill-large.tiff'));
imshow(uint8(round(bigimage)));
qimage = bigimage;
for i=1:size(bigimage,1), for j=1:size(bigimage,2),
        dist = zeros(k,1);
        for l=1:k,
            d = mu(l,:)'-permute(bigimage(i,j,:), [3 2 1]);
            dist(l) = d'*d;
        end;
        [value, assignment] = min(dist);
        qimage(i,j,:) = ipermute(mu(assignment,:), [3 2 1]);
```

```
    end; end;
imshow(uint8(round(qimage)));
```

Figure 1: The original image of the mandrill.

Figure 2: The compressed image of the mandrill.

CS 229, Autumn 2016
Problem Set #4: Unsupervised learning & RL

Due Wednesday, December 7 at 11:00 am on Gradescope

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[11 points] EM for MAP estimation**

   The EM algorithm that we talked about in class was for solving a maximum likelihood estimation problem in which we wished to maximize

   $$\prod_{i=1}^{m} p(x^{(i)}; \theta) = \prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta),$$

   where the $z^{(i)}$'s were latent random variables. Suppose we are working in a Bayesian framework, and wanted to find the MAP estimate of the parameters θ by maximizing

   $$\left(\prod_{i=1}^{m} p(x^{(i)}|\theta) \right) p(\theta) = \left(\prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}|\theta) \right) p(\theta).$$

   Here, $p(\theta)$ is our prior on the parameters. Generalize the EM algorithm to work for MAP estimation. You may assume that $\log p(x, z|\theta)$ and $\log p(\theta)$ are both concave in θ, so that the M-step is tractable if it requires only maximizing a linear combination of these quantities. (This roughly corresponds to assuming that MAP estimation is tractable when x, z is fully observed, just like in the frequentist case where we considered examples in which maximum likelihood estimation was easy if x, z was fully observed.)

   Make sure your M-step is tractable, and also prove that $\prod_{i=1}^{m} p(x^{(i)}|\theta)p(\theta)$ (viewed as a function of θ) monotonically increases with each iteration of your algorithm.

2. **[22 points] EM application**

   Consider the following problem. There are P papers submitted to a machine learning conference. Each of R reviewers reads each paper, and gives it a score indicating how good he/she thought that paper was. We let $x^{(pr)}$ denote the score that reviewer r gave to paper p. A high score means the reviewer liked the paper, and represents a recommendation from that reviewer that it be accepted for the conference. A low score means the reviewer did not like the paper.

We imagine that each paper has some "intrinsic," true value that we denote by μ_p, where a large value means it's a good paper. Each reviewer is trying to estimate, based on reading the paper, what μ_p is; the score reported $x^{(pr)}$ is then reviewer r's guess of μ_p.

However, some reviewers are just generally inclined to think all papers are good and tend to give all papers high scores; other reviewers may be particularly nasty and tend to give low scores to everything. (Similarly, different reviewers may have different amounts of variance in the way they review papers, making some reviewers more consistent/reliable than others.) We let ν_r denote the "bias" of reviewer r. A reviewer with bias ν_r is one whose scores generally tend to be ν_r higher than they should be.

All sorts of different random factors influence the reviewing process, and hence we will use a model that incorporates several sources of noise. Specifically, we assume that reviewers' scores are generated by a random process given as follows:

$$\begin{aligned} y^{(pr)} &\sim \mathcal{N}(\mu_p, \sigma_p^2), \\ z^{(pr)} &\sim \mathcal{N}(\nu_r, \tau_r^2), \\ x^{(pr)}|y^{(pr)}, z^{(pr)} &\sim \mathcal{N}(y^{(pr)} + z^{(pr)}, \sigma^2). \end{aligned}$$

The variables $y^{(pr)}$ and $z^{(pr)}$ are independent; the variables (x, y, z) for different paper-reviewer pairs are also jointly independent. Also, we only ever observe the $x^{(pr)}$'s; thus, the $y^{(pr)}$'s and $z^{(pr)}$'s are all latent random variables.

We would like to estimate the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$. If we obtain good estimates of the papers' "intrinsic values" μ_p, these can then be used to make acceptance/rejection decisions for the conference.

We will estimate the parameters by maximizing the marginal likelihood of the data $\{x^{(pr)}; p = 1, \ldots, P, r = 1, \ldots, R\}$. This problem has latent variables $y^{(pr)}$ and $z^{(pr)}$, and the maximum likelihood problem cannot be solved in closed form. So, we will use EM. Your task is to derive the EM update equations. Your final E and M step updates should consist only of addition/subtraction/multiplication/division/log/exp/sqrt of scalars; and addition/subtraction/multiplication/inverse/determinant of matrices. For simplicity, you need to treat only $\{\mu_p, \sigma_p^2; p = 1 \ldots P\}$ and $\{\nu_r, \tau_r^2; r = 1 \ldots R\}$ as parameters. I.e. treat σ^2 (the conditional variance of $x^{(pr)}$ given $y^{(pr)}$ and $z^{(pr)}$) as a fixed, known constant.

(a) In this part, we will derive the E-step:

(i) The joint distribution $p(y^{(pr)}, z^{(pr)}, x^{(pr)})$ has the form of a multivariate Gaussian density. Find its associated mean vector and covariance matrix in terms of the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$, and σ^2.

[Hint: Recognize that $x^{(pr)}$ can be written as $x^{(pr)} = y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}$, where $\epsilon^{(pr)} \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise.]

(ii) Derive an expression for $Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)}|x^{(pr)})$ (E-step), using the rules for conditioning on subsets of jointly Gaussian random variables (see the notes on Factor Analysis).

(b) Derive the M-step updates to the parameters $\{\mu_p, \nu_r, \sigma_p^2, \tau_r^2\}$. [Hint: It may help to express the lower bound on the likelihood in terms of an expectation with respect to $(y^{(pr)}, z^{(pr)})$ drawn from a distribution with density $Q_{pr}(y^{(pr)}, z^{(pr)})$.]

Remark. In a recent machine learning conference, John Platt (whose SMO algorithm you've seen) implemented a method quite similar to this one to estimate the papers' true scores μ_p. (There, the problem was a bit more complicated because not all reviewers reviewed every paper, but the essential ideas are the same.) Because the model tried to estimate and correct for reviewers' biases ν_r, its estimates of μ_p were significantly more useful for making accept/reject decisions than the reviewers' raw scores for a paper.

3. **[14 points] PCA**

   In class, we showed that PCA finds the "variance maximizing" directions onto which to project the data. In this problem, we find another interpretation of PCA.

   Suppose we are given a set of points $\{x^{(1)}, \ldots, x^{(m)}\}$. Let us assume that we have as usual preprocessed the data to have zero-mean and unit variance in each coordinate. For a given unit-length vector u, let $f_u(x)$ be the projection of point x onto the direction given by u. I.e., if $\mathcal{V} = \{\alpha u : \alpha \in \mathbb{R}\}$, then

   $$f_u(x) = \arg\min_{v \in \mathcal{V}} ||x - v||^2.$$

   Show that the unit-length vector u that minimizes the mean squared error between projected points and original points corresponds to the first principal component for the data. I.e., show that

   $$\arg\min_{u:u^Tu=1} \sum_{i=1}^{m} \|x^{(i)} - f_u(x^{(i)})\|_2^2 \ .$$

   gives the first principal component.

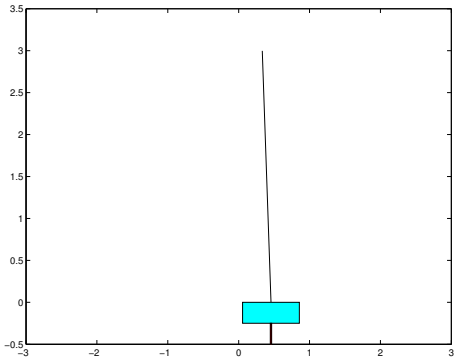
   Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squares distance between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first k principal components of the data. This problem shows that this result holds for the case of $k = 1$.

4. **[12 points] Independent components analysis**

   For this question you will implement the Bell and Sejnowski ICA algorithm, as covered in class. The files you'll need for this problem are in `/afs/ir/class/cs229/ps/ps4/q4`. The file `mix.dat` contains a matrix with 5 columns, with each column corresponding to one of the mixed signals x_i. The file `bellsej.m` contains starter code for your implementation.

   Implement and run ICA, and report what was the W matrix you found. Please make your code clean and very concise, and use symbol conventions as in class. To make sure your code is correct, you should listen to the resulting unmixed sources. (Some overlap in the sources may be present, but the different sources should be pretty clearly separated.)

   Note: In our implementation, we **annealed** the learning rate α (slowly decreased it over time) to speed up learning. We briefly describe in `bellsej.m` what we did, but you should feel free to play with things to make it work best for you. In addition to using the variable learning rate to speed up convergence, one thing that we also tried was choosing a random permutation of the training data, and running stochastic gradient ascent visiting the training data in that order (each of the specified learning rates was then used for one full pass through the data); this is something that you could try, too.



5. **[16 points] Markov decision processes**

   Consider an MDP with finite state and action spaces, and discount factor $\gamma < 1$. Let B be the Bellman update operator with V a vector of values for each state. I.e., if $V' = B(V)$, then

$$V'(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V(s').$$

   (a) **[12 points]** Prove that, for any two finite-valued vectors V_1, V_2, it holds true that

$$||B(V_1) - B(V_2)||_\infty \leq \gamma ||V_1 - V_2||_\infty.$$

   where

$$||V||_\infty = \max_{s \in S} |V(s)|.$$

   (This shows that the Bellman update operator is a "γ-contraction in the max-norm.")

   (b) **[4 points]** We say that V is a **fixed point** of B if $B(V) = V$. Using the fact that the Bellman update operator is a γ-contraction in the max-norm, prove that B has at most one fixed point—i.e., that there is at most one solution to the Bellman equations. You may assume that B has at least one fixed point.

6. **[25 points] Reinforcement Learning: The inverted pendulum**

   In this problem, you will apply reinforcement learning to automatically design a policy for a difficult control task, without ever using any explicit knowledge of the dynamics of the underlying system.

   The problem we will consider is the inverted pendulum or the pole-balancing problem.[1]

   Consider the figure shown. A thin pole is connected via a free hinge to a cart, which can move laterally on a smooth table surface. The controller is said to have failed if either the angle of the pole deviates by more than a certain amount from the vertical position (i.e., if the pole falls over), or if the cart's position goes out of bounds (i.e., if it falls off the end of the table). Our objective is to develop a controller to balance the pole with these constraints, by appropriately having the cart accelerate left and right.

   We have written a simple Matlab simulator for this problem. The simulation proceeds in discrete time cycles (steps). The state of the cart and pole at any time is completely characterized by 4 parameters: the cart position x, the cart velocity $\dot{x}$, the angle of the

[1] The dynamics are adapted from `http://www-anw.cs.umass.edu/rlr/domains.html`

pole θ measured as its deviation from the vertical position, and the angular velocity of the pole $\dot{\theta}$. Since it'd be simpler to consider reinforcement learning in a discrete state space, we have approximated the state space by a discretization that maps a state vector $(x, \dot{x}, \theta, \dot{\theta})$ into a number from 1 to `NUM_STATES`. Your learning algorithm will need to deal only with this discretized representation of the states.

At every time step, the controller must choose one of two actions - push (accelerate) the cart right, or push the cart left. (To keep the problem simple, there is no *do-nothing* action.) These are represented as actions 1 and 2 respectively in the code. When the action choice is made, the simulator updates the state parameters according to the underlying dynamics, and provides a new discretized state.

We will assume that the reward $R(s)$ is a function of the current state only. When the pole angle goes beyond a certain limit or when the cart goes too far out, a negative reward is given, and the system is reinitialized randomly. At all other times, the reward is zero. Your program must learn to balance the pole using only the state transitions and rewards observed.

The files for this problem are in `/afs/ir/class/cs229/ps/ps4/q6`. Most of the the code has already been written for you, and you need to make changes only to `control.m` in the places specified. This file can be run in Matlab to show a display and to plot a learning curve at the end. Read the comments at the top of the file for more details on the working of the simulation.[2]

(a) To solve the inverted pendulum problem, you will estimate a model (i.e., transition probabilities and rewards) for the underlying MDP, solve Bellman's equations for this estimated MDP to obtain a value function, and act greedily with respect to this value function.

Briefly, you will maintain a current model of the MDP and a current estimate of the value function. Initially, each state has estimated reward zero, and the estimated transition probabilities are uniform (equally likely to end up in any other state).

During the simulation, you must choose actions at each time step according to some current policy. As the program goes along taking actions, it will gather observations on transitions and rewards, which it can use to get a better estimate of the MDP model. Since it is inefficient to update the whole estimated MDP after every observation, we will store the state transitions and reward observations each time, and update the model and value function/policy only periodically. Thus, you must maintain counts of the total number of times the transition from state s_i to state s_j using action a has been observed (similarly for the rewards). Note that the rewards at any state are deterministic, but the state transitions are not because of the discretization of the state space (several different but close configurations may map onto the same discretized state).

Each time a failure occurs (such as if the pole falls over), you should re-estimate the transition probabilities and rewards as the average of the observed values (if any). Your program must then use value iteration to solve Bellman's equations on the estimated MDP, to get the value function and new optimal policy for the new model. For value iteration, use a convergence criterion that checks if the maximum absolute change in the value function on an iteration exceeds some specified tolerance.

[2] Note that the routine for drawing the cart does not work in Octave.

Finally, assume that the whole learning procedure has converged once several consecutive attempts (defined by the parameter `NO_LEARNING_THRESHOLD`) to solve Bellman's equation all converge in the first iteration. Intuitively, this indicates that the estimated model has stopped changing significantly.

The code outline for this problem is already in `control.m`, and you need to write code fragments only at the places specified in the file. There are several details (convergence criteria etc.) that are also explained inside the code. Use a discount factor of $\gamma = 0.995$.

Implement the reinforcement learning algorithm as specified, and run it. How many trials (how many times did the pole fall over or the cart fall off) did it take before the algorithm converged?

(b) Plot a learning curve showing the number of time-steps for which the pole was balanced on each trial. You just need to execute `plot_learning_curve.m` after `control.m` to get this plot.

CS 229, Autumn 2016
Problem Set #4 Solutions: Unsupervised learning & RL

Due Wednesday, December 7 at 11:00 am on Gradescope

Notes: (1) These questions require thought, but do not require long answers. Please be as concise as possible. (2) If you have a question about this homework, we encourage you to post your question on our Piazza forum, at `https://piazza.com/stanford/autumn2016/cs229`. (3) If you missed the first lecture or are unfamiliar with the collaboration or honor code policy, please read the policy on Handout #1 (available from the course website) before starting work. (4) For problems that require programming, please include in your submission a printout of your code (with comments) and any figures that you are asked to plot.

If you are scanning your document by cellphone, please check the Piazza forum for recommended cellphone scanning apps and best practices.

1. **[11 points] EM for MAP estimation**

   The EM algorithm that we talked about in class was for solving a maximum likelihood estimation problem in which we wished to maximize

   $$\prod_{i=1}^{m} p(x^{(i)};\theta) = \prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)};\theta),$$

   where the $z^{(i)}$'s were latent random variables. Suppose we are working in a Bayesian framework, and wanted to find the MAP estimate of the parameters θ by maximizing

   $$\left(\prod_{i=1}^{m} p(x^{(i)}|\theta)\right) p(\theta) = \left(\prod_{i=1}^{m} \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}|\theta)\right) p(\theta).$$

   Here, $p(\theta)$ is our prior on the parameters. Generalize the EM algorithm to work for MAP estimation. You may assume that $\log p(x, z|\theta)$ and $\log p(\theta)$ are both concave in θ, so that the M-step is tractable if it requires only maximizing a linear combination of these quantities. (This roughly corresponds to assuming that MAP estimation is tractable when x, z is fully observed, just like in the frequentist case where we considered examples in which maximum likelihood estimation was easy if x, z was fully observed.)

   Make sure your M-step is tractable, and also prove that $\prod_{i=1}^{m} p(x^{(i)}|\theta)p(\theta)$ (viewed as a function of θ) monotonically increases with each iteration of your algorithm.

   Answer: We will derive the EM updates the same way as done in class for maximum likelihood estimation. Monotonic increase with every iteration is guaranteed because of the same reason: in the E-step we compute a lower bound that is tight at the current estimate of θ, in the M-step we optimize θ for this lower bound, so we are guaranteed to improve the actual objective function.

$$\begin{aligned}
\log \prod_{i=1}^{m} p(x^{(i)}|\theta)p(\theta) &= \log p(\theta) + \sum_{i=1}^{m} \log p(x^{(i)}|\theta) \\
&= \log p(\theta) + \sum_{i=1}^{m} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}|\theta) \\
&= \log p(\theta) + \sum_{i=1}^{m} \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}|\theta)}{Q_i(z^{(i)})} \\
&\geq \log p(\theta) + \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}|\theta)}{Q_i(z^{(i)})},
\end{aligned}$$

where we just did straightforward substitutions and rewritings, and the last step is given by Jensen's inequality. Requiring the inequality to be tight, gives us the E-step:

$$Q_i(z^{(i)}) = p(z^{(i)}|x^{(i)}; \theta).$$

For the M-step we maximize the lower bound, i.e.

$$\theta = \arg\max_{\theta} \left[\log p(\theta) + \sum_{i=1}^{m} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}|\theta)}{Q_i(z^{(i)})} \right].$$

The M-step is tractable, since it only requires maximizing a linear combination of tractable concave terms $\log p(x, z|\theta)$ and $\log p(\theta)$.

2. **[22 points] EM application**

   Consider the following problem. There are P papers submitted to a machine learning conference. Each of R reviewers reads each paper, and gives it a score indicating how good he/she thought that paper was. We let $x^{(pr)}$ denote the score that reviewer r gave to paper p. A high score means the reviewer liked the paper, and represents a recommendation from that reviewer that it be accepted for the conference. A low score means the reviewer did not like the paper.

   We imagine that each paper has some "intrinsic," true value that we denote by μ_p, where a large value means it's a good paper. Each reviewer is trying to estimate, based on reading the paper, what μ_p is; the score reported $x^{(pr)}$ is then reviewer r's guess of μ_p.

   However, some reviewers are just generally inclined to think all papers are good and tend to give all papers high scores; other reviewers may be particularly nasty and tend to give low scores to everything. (Similarly, different reviewers may have different amounts of variance in the way they review papers, making some reviewers more consistent/reliable than others.) We let ν_r denote the "bias" of reviewer r. A reviewer with bias ν_r is one whose scores generally tend to be ν_r higher than they should be.

   All sorts of different random factors influence the reviewing process, and hence we will use a model that incorporates several sources of noise. Specifically, we assume that reviewers' scores are generated by a random process given as follows:

$$\begin{aligned}
y^{(pr)} &\sim \mathcal{N}(\mu_p, \sigma_p^2), \\
z^{(pr)} &\sim \mathcal{N}(\nu_r, \tau_r^2), \\
x^{(pr)}|y^{(pr)}, z^{(pr)} &\sim \mathcal{N}(y^{(pr)} + z^{(pr)}, \sigma^2).
\end{aligned}$$

The variables $y^{(pr)}$ and $z^{(pr)}$ are independent; the variables (x, y, z) for different paper-reviewer pairs are also jointly independent. Also, we only ever observe the $x^{(pr)}$'s; thus, the $y^{(pr)}$'s and $z^{(pr)}$'s are all latent random variables.

We would like to estimate the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$. If we obtain good estimates of the papers' "intrinsic values" μ_p, these can then be used to make acceptance/rejection decisions for the conference.

We will estimate the parameters by maximizing the marginal likelihood of the data $\{x^{(pr)}; p = 1, \ldots, P, r = 1, \ldots, R\}$. This problem has latent variables $y^{(pr)}$ and $z^{(pr)}$, and the maximum likelihood problem cannot be solved in closed form. So, we will use EM. Your task is to derive the EM update equations. Your final E and M step updates should consist only of addition/subtraction/multiplication/division/log/exp/sqrt of scalars; and addition/subtraction/multiplication/inverse/determinant of matrices. For simplicity, you need to treat only $\{\mu_p, \sigma_p^2; p = 1 \ldots P\}$ and $\{\nu_r, \tau_r^2; r = 1 \ldots R\}$ as parameters. I.e. treat σ^2 (the conditional variance of $x^{(pr)}$ given $y^{(pr)}$ and $z^{(pr)}$) as a fixed, known constant.

(a) In this part, we will derive the E-step:

(i) The joint distribution $p(y^{(pr)}, z^{(pr)}, x^{(pr)})$ has the form of a multivariate Gaussian density. Find its associated mean vector and covariance matrix in terms of the parameters $\mu_p, \sigma_p^2, \nu_r, \tau_r^2$, and σ^2.

[Hint: Recognize that $x^{(pr)}$ can be written as $x^{(pr)} = y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}$, where $\epsilon^{(pr)} \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise.]

(ii) Derive an expression for $Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)}|x^{(pr)})$ (E-step), using the rules for conditioning on subsets of jointly Gaussian random variables (see the notes on Factor Analysis).

(b) Derive the M-step updates to the parameters $\{\mu_p, \nu_r, \sigma_p^2, \tau_r^2\}$. [Hint: It may help to express the lower bound on the likelihood in terms of an expectation with respect to $(y^{(pr)}, z^{(pr)})$ drawn from a distribution with density $Q_{pr}(y^{(pr)}, z^{(pr)})$.]

Remark. In a recent machine learning conference, John Platt (whose SMO algorithm you've seen) implemented a method quite similar to this one to estimate the papers' true scores μ_p. (There, the problem was a bit more complicated because not all reviewers reviewed every paper, but the essential ideas are the same.) Because the model tried to estimate and correct for reviewers' biases ν_r, its estimates of μ_p were significantly more useful for making accept/reject decisions than the reviewers' raw scores for a paper.

Answer:

Let Θ denote the whole set of parameters we are estimating, then the EM steps for our problem are (at a high level):

(a) (E-step) For each p, r, set $Q_{pr}(y^{(pr)}, z^{(pr)}) = p(y^{(pr)}, z^{(pr)}|x^{(pr)}; \theta)$.

(b) (M-step) Set $\Theta = \arg\max_\Theta \sum_{p=1}^{P} \sum_{r=1}^{R} E_{Q_{pr}(Y^{(pr)}, Z^{(pr)})} \log p(x^{(pr)}, Y^{(pr)}, Z^{(pr)}; \Theta)$.

Now it's a matter of working out how these updates can actually be computed.

For the E-step, if we use Bayes's Rule to compute $p(y^{(pr)}, z^{(pr)}|x^{(pr)})$, then we'll get integrals of Gaussians in the denominator, which are tough to compute. Instead, observe that

$$p(y^{(pr)}, z^{(pr)}, x^{(pr)}) = p(y^{(pr)}, z^{(pr)}) p(x^{(pr)}|y^{(pr)}, z^{(pr)}) = p(y^{(pr)}) p(z^{(pr)}) p(x^{(pr)}|y^{(pr)}, z^{(pr)})$$

is the product of three Gaussian densities, so it is itself a multivariate Gaussian density. Therefore, the joint distribution $p(y^{(pr)}, z^{(pr)}, x^{(pr)})$ is some type of normal distribution so we can use the rules for conditioning Gaussians to compute the conditional. To get a form for the joint density, we'll exploit the fact that a multivariate Gaussian density is fully parameterized by its mean vector and covariance matrix.

- To compute the mean vector, we'll rewrite the $x^{(pr)}$ in the following way: $x^{(pr)} = y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}$, where $\epsilon^{(pr)} \sim \mathcal{N}(0, \sigma^2)$ is independent Gaussian noise.[1] Then, $E[y^{(pr)}] = \mu_p$, $E[z^{(pr)}] = \nu_r$ and

$$\begin{aligned} E[x^{(pr)}] &= E[y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}] = E[y^{(pr)}] + E[z^{(pr)}] + E[\epsilon^{(pr)}] \\ &= \mu_p + \nu_r + 0 = \mu_p + \nu_r. \end{aligned}$$

- To compute the covariance matrix, observe that $\mathrm{Var}(y^{(pr)}) = \sigma_p^2$, $\mathrm{Var}(z^{(pr)}) = \tau_r^2$, and $\mathrm{Cov}(y^{(pr)}, z^{(pr)}) = \mathrm{Cov}(z^{(pr)}, y^{(pr)}) = 0$ (since $y^{(pr)}$ and $z^{(pr)}$ are independent). Also, since $y^{(pr)}$, $z^{(pr)}$, and $\epsilon^{(pr)}$ are independent, we have

$$\begin{aligned} \mathrm{Var}(x^{(pr)}) &= \mathrm{Var}(y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}) = \mathrm{Var}(y^{(pr)}) + \mathrm{Var}(z^{(pr)}) + \mathrm{Var}(\epsilon^{(pr)}) \\ &= \sigma_p^2 + \tau_r^2 + \sigma^2. \end{aligned}$$

  Finally,

$$\begin{aligned} \mathrm{Cov}(y^{(pr)}, x^{(pr)}) &= \mathrm{Cov}(x^{(pr)}, y^{(pr)}) \\ &= \mathrm{Cov}(y^{(pr)} + z^{(pr)} + \epsilon^{(pr)}, y^{(pr)}) \\ &= \mathrm{Cov}(y^{(pr)}, y^{(pr)}) + \mathrm{Cov}(z^{(pr)}, y^{(pr)}) + \mathrm{Cov}(\epsilon^{(pr)}, y^{(pr)}) \\ &= \sigma_p^2 + 0 + 0 = \sigma_p^2. \end{aligned}$$

  where the second to last equality follows from independence of $y^{(pr)}$, $z^{(pr)}$ and $\epsilon^{(pr)}$. Similarly, we can show that $\mathrm{Cov}(z^{(pr)}, x^{(pr)}) = \mathrm{Cov}(x^{(pr)}, z^{(pr)}) = \tau_r^2$.

This allows us to write

$$y^{(pr)}, z^{(pr)}, x^{(pr)} \sim \mathcal{N}\left(\begin{bmatrix} \mu_p \\ \nu_r \\ \mu_p + \nu_r \end{bmatrix}, \begin{bmatrix} \sigma_p^2 & 0 & \sigma_p^2 \\ 0 & \tau_r^2 & \tau_r^2 \\ \sigma_p^2 & \tau_r^2 & \sigma_p^2 + \tau_r^2 + \sigma^2 \end{bmatrix} \right)$$

Now we can use the standard results for conditioning on subsets of variables for Gaussians (from the Factor Analysis notes) to obtain:

$$Q_{pr}(y^{(pr)}, z^{(pr)}) = \mathcal{N}\left(\begin{bmatrix} \mu_{pr,Y} \\ \mu_{pr,Z} \end{bmatrix}, \begin{bmatrix} \Sigma_{pr,YY} & \Sigma_{pr,YZ} \\ \Sigma_{pr,ZY} & \Sigma_{pr,ZZ} \end{bmatrix} \right)$$

[1] To see why this follows from the definition in the problem statement, observe that the probability that $\epsilon^{(pr)} = x^{(pr)} - y^{(pr)} - z^{(pr)}$ takes on any specific value ϵ is $p(\epsilon^{(pr)} = \epsilon | y^{(pr)}, z^{(pr)}) = p(x^{(pr)} - y^{(pr)} - z^{(pr)} = \epsilon | y^{(pr)}, z^{(pr)}) = p(x^{(pr)} = \epsilon + y^{(pr)} + z^{(pr)} | y^{(pr)}, z^{(pr)}) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}\epsilon^2)$ which does not depend on either $y^{(pr)}$ or $z^{(pr)}$; hence $\epsilon^{(pr)}$ can be regarded as independent zero-mean Gaussian noise with σ^2 variance.

where

$$\mu_{pr} = \begin{bmatrix} \mu_{pr,Y} \\ \mu_{pr,Z} \end{bmatrix} = \begin{bmatrix} \mu_p + \frac{\sigma_p^2}{\sigma^2 + \sigma_p^2 + \tau_r^2}(x^{(pr)} - \mu_p - \nu_r) \\ \nu_r + \frac{\tau_r^2}{\sigma^2 + \sigma_p^2 + \tau_r^2}(x^{(pr)} - \mu_p - \nu_r) \end{bmatrix} \tag{1}$$

$$\Sigma_{pr} = \begin{bmatrix} \Sigma_{pr,YY} & \Sigma_{pr,YZ} \\ \Sigma_{pr,ZY} & \Sigma_{pr,ZZ} \end{bmatrix} = \frac{1}{\sigma_p^2 + \tau_r^2 + \sigma^2} \begin{bmatrix} \sigma_p^2(\tau_r^2 + \sigma^2) & -\sigma_p^2\tau_r^2 \\ -\sigma_p^2\tau_r^2 & \tau_r^2(\sigma_p^2 + \sigma^2) \end{bmatrix}. \tag{2}$$

For the M-step, an important realization is that the Q_{pr} distribution is defined in terms of Θ^t, while we want to choose the parameters for the next time step, Θ^{t+1}. This means that the parameters of the Q_{pr} distributions are constant in terms of the parameters we wish to maximize. Maximizing the expected log-likelihood, we have (letting $E_Q[\cdot]$ denote expectations with respect to $Q_{pr}(y^{(pr)}, z^{(pr)})$ for each p and r, respectively),

$$\begin{aligned}
\Theta &= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} E_Q \log p(x^{(pr)}, y^{(pr)}, z^{(pr)}; \Theta) \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} E_Q \log \left[\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2\sigma^2}(x^{(pr)} - y^{(pr)} - z^{(pr)})^2} \frac{1}{\sqrt{2\pi}\sigma_p} e^{-\frac{1}{2\sigma_p^2}(y^{(pr)} - \mu_p)^2} \frac{1}{\sqrt{2\pi}\tau_r} e^{-\frac{1}{2\tau_r^2}(z^{(pr)} - \nu_r)^2} \right] \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} E_Q \left[\log \frac{1}{(2\pi)^{3/2}\sigma\sigma_p\tau_r} - \frac{1}{2\sigma^2}(x^{(pr)} - y^{(pr)} - z^{(pr)})^2 - \frac{1}{2\sigma_p^2}(y^{(pr)} - \mu_p)^2 - \frac{1}{2\tau_r^2}(y^{(pr)} - \nu_r)^2 \right] \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} E_Q \left[\log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}(y^{(pr)} - \mu_p)^2 - \frac{1}{2\tau_r^2}(z^{(pr)} - \nu_r)^2 \right] \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} E_Q \left[\log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}((y^{(pr)})^2 - 2y^{(pr)}\mu_p + \mu_p^2) - \frac{1}{2\tau_r^2}((z^{(pr)})^2 - 2z^{(pr)}\nu_r + \nu_r^2) \right] \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} \left[\log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}(E_Q[(y^{(pr)})^2] - 2E_Q[y^{(pr)}]\mu_p + \mu_p^2) - \frac{1}{2\tau_r^2}(E_Q[(z^{(pr)})^2] - 2E_Q[z^{(pr)}]\nu_r + \nu_r^2) \right] \\
&= \arg\max_\Theta \sum_{p=1}^{P}\sum_{r=1}^{R} \left[\log \frac{1}{\sigma_p\tau_r} - \frac{1}{2\sigma_p^2}(\Sigma_{pr,YY} + \mu_{pr,Y}^2 - 2\mu_{pr,Y}\mu_p + \mu_p^2) - \frac{1}{2\tau_r^2}(\Sigma_{pr,ZZ} + \mu_{pr,Z}^2 - 2\mu_{pr,Z}\nu_r + \nu_r^2) \right].
\end{aligned}$$

where the equality in the last line follows from $E_Q[y^{(pr)}] = \mu_{pr,Y}$ and $E_Q[(y^{(pr)})^2] = (E_Q[(y^{(pr)})^2] - E_Q[y^{(pr)}]^2) + E_Q[y^{(pr)}]^2 = \Sigma_{pr,YY} + \mu_{pr,Y}^2$ (and similarly for $E_Q[z^{(pr)}]$ and

$E_Q[(z^{(pr)})^2]$). Setting derivatives w.r.t. parameters $\mu_p, \nu_r, \sigma_p, \tau_r$ to 0,

$$-\frac{1}{2\sigma_p^2}\sum_{r=1}^{R}(2\mu_p - 2\mu_{pr,Y}) = 0 \quad \Longrightarrow \quad \mu_p = \frac{1}{R}\sum_{r=1}^{R}\mu_{pr,Y} \tag{3}$$

$$-\frac{1}{2\tau_r^2}\sum_{p=1}^{P}(2\nu_r - 2\mu_{pr,Z}) = 0 \quad \Longrightarrow \quad \nu_r = \frac{1}{P}\sum_{p=1}^{P}\mu_{pr,Z} \tag{4}$$

$$\sum_{r=1}^{R}\left[-\frac{1}{\sigma_p} + \frac{1}{\sigma_p^3}(\Sigma_{pr,YY} + \mu_{pr,Y}^2 - 2\mu_{pr,Y}\mu_p + \mu_p^2)\right] = 0 \quad \Longrightarrow \quad \sigma_p^2 = \frac{1}{R}\sum_{r=1}^{R}(\Sigma_{pr,YY} + \mu_{pr,Y}^2 - 2\mu_{pr,Y}\mu_p + \mu_p^2) \tag{5}$$

$$\sum_{p=1}^{P}\left[-\frac{1}{\tau_r} + \frac{1}{\tau_r^3}(\Sigma_{pr,ZZ} + \mu_{pr,Z}^2 - 2\mu_{pr,Z}\nu_r + \nu_r^2)\right] = 0 \quad \Longrightarrow \quad \tau_r^2 = \frac{1}{P}\sum_{p=1}^{P}(\Sigma_{pr,ZZ} + \mu_{pr,Z}^2 - 2\mu_{pr,Z}\nu_r + \nu_r^2) \tag{6}$$

Using the above results, we can restate our E and M steps in terms of actual computations:

(a) (E-step) For each p, r, compute μ_{pr}, Σ_{pr} using equations (1),(2)

(b) (M-step) Compute $\mu_p, \nu_r, \sigma_p^2, \tau_r^2$ using equations (3), (4), (5), (6).

3. **[14 points] PCA**

In class, we showed that PCA finds the "variance maximizing" directions onto which to project the data. In this problem, we find another interpretation of PCA.

Suppose we are given a set of points $\{x^{(1)}, \ldots, x^{(m)}\}$. Let us assume that we have as usual preprocessed the data to have zero-mean and unit variance in each coordinate. For a given unit-length vector u, let $f_u(x)$ be the projection of point x onto the direction given by u. I.e., if $\mathcal{V} = \{\alpha u : \alpha \in \mathbb{R}\}$, then

$$f_u(x) = \arg\min_{v \in \mathcal{V}} ||x - v||^2.$$

Show that the unit-length vector u that minimizes the mean squared error between projected points and original points corresponds to the first principal component for the data. I.e., show that

$$\arg\min_{u:u^Tu=1} \sum_{i=1}^{m} \|x^{(i)} - f_u(x^{(i)})\|_2^2 \ .$$

gives the first principal component.

Remark. If we are asked to find a k-dimensional subspace onto which to project the data so as to minimize the sum of squares distance between the original data and their projections, then we should choose the k-dimensional subspace spanned by the first k principal components of the data. This problem shows that this result holds for the case of $k = 1$.

Answer: First note we have $f_u(x^{(i)}) = u^T x u$.[2] So we have to solve the following problem:

$$\begin{aligned}
\arg\min_{u:u^Tu=1} \sum_{i=1}^m \|x^{(i)} - f_u(x^{(i)})\|_2^2 &= \arg\min_{u:u^Tu=1} \sum_{i=1}^m \|x^{(i)} - u^T x^{(i)} u\|_2^2 \\
&= \arg\min_{u:u^Tu=1} \sum_{i=1}^m (x^{(i)} - u^T x^{(i)} u)^T (x^{(i)} - u^T x^{(i)} u) \\
&= \arg\min_{u:u^Tu=1} \sum_{i=1}^m (x^{(i)T} x^{(i)} - 2(u^T x^{(i)})^2 + u^T u (u^T x^{(i)})^2) \\
&= \arg\min_{u:u^Tu=1} \sum_{i=1}^m (x^{(i)T} x^{(i)} - 2(u^T x^{(i)})^2 + (u^T x^{(i)})^2) \\
&= \arg\min_{u:u^Tu=1} \sum_{i=1}^m -(u^T x^{(i)})^2 \\
&= \arg\max_{u:u^Tu=1} u^T \left(\sum_{i=1}^m x^{(i)} x^{(i)T} \right) u
\end{aligned}$$

And the last line corresponds to the optimization problem that defines the first principal component.

4. **[12 points] Independent components analysis**

   For this question you will implement the Bell and Sejnowski ICA algorithm, as covered in class. The files you'll need for this problem are in `/afs/ir/class/cs229/ps/ps4/q4`. The file `mix.dat` contains a matrix with 5 columns, with each column corresponding to one of the mixed signals x_i. The file `bellsej.m` contains starter code for your implementation.

   Implement and run ICA, and report what was the W matrix you found. Please make your code clean and very concise, and use symbol conventions as in class. To make sure your code is correct, you should listen to the resulting unmixed sources. (Some overlap in the sources may be present, but the different sources should be pretty clearly separated.)

   Note: In our implementation, we **annealed** the learning rate α (slowly decreased it over time) to speed up learning. We briefly describe in `bellsej.m` what we did, but you should feel free to play with things to make it work best for you. In addition to using the variable learning rate to speed up convergence, one thing that we also tried was choosing a random permutation of the training data, and running stochastic gradient ascent visiting the training data in that order (each of the specified learning rates was then used for one full pass through the data); this is something that you could try, too.

   Answer:

```
%---------------------------------------------------------
% ICA
```

[2] To see why, observe that

$$f_u(x) = u \cdot \left(\arg\min_\alpha ||x - \alpha u||^2 \right) = u \cdot \left(\arg\min_\alpha (x^T x - 2\alpha x^T u + \alpha^2 u^T u) \right) = u \cdot \left(\frac{2x^T u}{2u^T u} \right) = u x^T u$$

where the third equality follows from the fact that the minimum of a convex quadratic function $ax^2 + bx + c$ is given by $x = -\frac{b}{2a}$, and the last equality follows from the fact that u is a unit-length vector.

```
load mix.dat % load mixed sources
Fs = 11025; %sampling frequency being used

% listen to the mixed sources
normalizedMix = 0.99 * mix ./ (ones(size(mix,1),1)*max(abs(mix)));

% handle writing in both matlab and octave
v = version;
if (v(1) <= '3') % assume this is octave
  wavwrite('mix1.wav', normalizedMix(:, 1), Fs, 16);
  wavwrite('mix2.wav', normalizedMix(:, 2), Fs, 16);
  wavwrite('mix3.wav', normalizedMix(:, 3), Fs, 16);
  wavwrite('mix4.wav', normalizedMix(:, 4), Fs, 16);
  wavwrite('mix5.wav', normalizedMix(:, 5), Fs, 16);
else
  wavwrite(normalizedMix(:, 1), Fs, 16, 'mix1.wav');
  wavwrite(normalizedMix(:, 2), Fs, 16, 'mix2.wav');
  wavwrite(normalizedMix(:, 3), Fs, 16, 'mix3.wav');
  wavwrite(normalizedMix(:, 4), Fs, 16, 'mix4.wav');
  wavwrite(normalizedMix(:, 5), Fs, 16, 'mix5.wav');
end

W=eye(5); % initialize unmixing matrix

% this is the annealing schedule I used for the learning rate.
% (We used stochastic gradient descent, where each value in the
% array was used as the learning rate for one pass through the data.)
% Note: If this doesn't work for you, feel free to fiddle with learning
%  rates, etc. to make it work.
anneal = [0.1 0.1 0.1 0.05 0.05 0.05 0.02 0.02 0.01 0.01 ...
      0.005 0.005 0.002 0.002 0.001 0.001];

for iter=1:length(anneal)

   %%%% here comes your code part

   m = size(mix, 1);
   order = randperm(m);
   for i = 1:m
      x = mix(order(i), :)';
      g = 1 ./ (1 + exp(-W * x));
      W = W + anneal(iter) * ((1 - 2 * g) * x' + inv(W'));
   end

end;

%%%% After finding W, use it to unmix the sources.  Place the unmixed sources
%%%% in the matrix S (one source per column).  (Your code.)
```

```
S = mix * W';

S=0.99 * S./(ones(size(mix,1),1)*max(abs(S)));  % rescale each column to have maximum absolute

% now have a listen --- You should have the following five samples:
% * Godfather
% * Southpark
% * Beethoven 5th
% * Austin Powers
% * Matrix (the movie, not the linear algebra construct :-)

v = version;
if (v(1) <= '3') % assume this is octave
  wavwrite('unmix1.wav', S(:, 1), Fs, 16);
  wavwrite('unmix2.wav', S(:, 2), Fs, 16);
  wavwrite('unmix3.wav', S(:, 3), Fs, 16);
  wavwrite('unmix4.wav', S(:, 4), Fs, 16);
  wavwrite('unmix5.wav', S(:, 5), Fs, 16);
else
  wavwrite(S(:, 1), Fs, 16, 'unmix1.wav');
  wavwrite(S(:, 2), Fs, 16, 'unmix2.wav');
  wavwrite(S(:, 3), Fs, 16, 'unmix3.wav');
  wavwrite(S(:, 4), Fs, 16, 'unmix4.wav');
  wavwrite(S(:, 5), Fs, 16, 'unmix5.wav');
end
```

5. **[16 points] Markov decision processes**

Consider an MDP with finite state and action spaces, and discount factor $\gamma < 1$. Let B be the Bellman update operator with V a vector of values for each state. I.e., if $V' = B(V)$, then

$$V'(s) = R(s) + \gamma \max_{a \in A} \sum_{s' \in S} P_{sa}(s')V(s').$$

(a) **[12 points]** Prove that, for any two finite-valued vectors V_1, V_2, it holds true that

$$||B(V_1) - B(V_2)||_\infty \leq \gamma ||V_1 - V_2||_\infty.$$

where

$$||V||_\infty = \max_{s \in S} |V(s)|.$$

(This shows that the Bellman update operator is a "γ-contraction in the max-norm.")

Answer: First we observe that $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$. To see why, define $a_f = \arg\max_a f(a)$ and $a_g = \arg\max_a g(a)$, respectively. Then,

$$f(a_f) - g(a_g) \leq f(a_f) - g(a_f) \leq |f(a_f) - g(a_f)| \leq \max_a |f(a) - g(a)|$$
$$g(a_g) - f(a_f) \leq g(a_g) - f(a_g) \leq |g(a_g) - f(a_g)| \leq \max_a |g(a) - f(a)|,$$

where the first inequality in each line follows from the fact that a_g and a_f are the maximizers of g and f, respectively. Combining the results from the two lines, it follows

that $|f(a_f) - g(a_g)| \leq \max_a |f(a) - g(a)|$, which is the equivalent to $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$.

Then, we have

$$\begin{aligned}
\|B(V_1) - B(V_2)\|_\infty &= \max_{s\in S} \left| \gamma \max_{a\in A} \sum_{s'\in S} P_{sa}(s')V_1(s') - \gamma \max_{a\in A} \sum_{s''\in S} P_{sa}(s'')V_2(s'') \right| \\
&\leq \gamma \max_{s\in S} \max_{a\in A} \left| \sum_{s'\in S} P_{sa}(s')(V_1(s') - V_2(s')) \right| \\
&\leq \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} |P_{sa}(s')(V_1(s') - V_2(s'))| \\
&= \gamma \max_{s\in S} \max_{a\in A} \sum_{s'\in S} P_{sa}(s') |(V_1(s') - V_2(s'))| \\
&\leq \gamma \max_{s\in S} \max_{a\in A} \max_{s'\in S} |(V_1(s') - V_2(s'))| \\
&= \gamma \max_{s'\in S} |V_1(s') - V_2(s')| \\
&= \gamma \|V_1 - V_2\|_\infty.
\end{aligned}$$

The first equality uses the definition of the Bellman operator (after noticing that $R(s)$ cancels). The second inequality comes from the fact that $|\max_a f(a) - \max_a g(a)| \leq \max_a |f(a) - g(a)|$ (and some simplification). The third inequality comes from the triangle inequality. The fourth equality comes from the fact that probabilities are nonnegative. The fifth equality follows from the fact that an expectation of a random variable is necessarily less than its maximum value. The sixth equality involves removing maximizations which play no role, and the final equality uses the definition of the max-norm.

(b) **[4 points]** We say that V is a **fixed point** of B if $B(V) = V$. Using the fact that the Bellman update operator is a γ-contraction in the max-norm, prove that B has at most one fixed point—i.e., that there is at most one solution to the Bellman equations. You may assume that B has at least one fixed point.

Answer: Suppose that V_1 and V_2 are 2 fixed points of B. We proved that $\|B(V_1) - B(V_2)\|_\infty \leq \gamma \|V_1 - V_2\|_\infty$, but $B(V_1) = V_1$ and $B(V_2) = V_2$ so

$$\|V_1 - V_2\|_\infty \leq \gamma \|V_1 - V_2\|_\infty \quad \Longrightarrow \quad (1-\gamma)\|V_1 - V_2\|_\infty \leq 0.$$

Since $0 \leq \gamma < 1$, then the coefficient $1 - \gamma$ is positive. Dividing through by $1 - \gamma$, and observing that the max-norm is always nonnegative, it follows that $\|V_1 - V_2\|_\infty = 0$, i.e. $V_1 = V_2$.

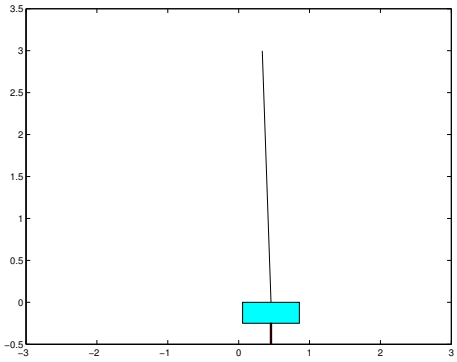
6. **[25 points] Reinforcement Learning: The inverted pendulum**

In this problem, you will apply reinforcement learning to automatically design a policy for a difficult control task, without ever using any explicit knowledge of the dynamics of the underlying system.

The problem we will consider is the inverted pendulum or the pole-balancing problem.[3]

Consider the figure shown. A thin pole is connected via a free hinge to a cart, which can move laterally on a smooth table surface. The controller is said to have failed if either the

[3]The dynamics are adapted from `http://www-anw.cs.umass.edu/rlr/domains.html`



angle of the pole deviates by more than a certain amount from the vertical position (i.e., if the pole falls over), or if the cart's position goes out of bounds (i.e., if it falls off the end of the table). Our objective is to develop a controller to balance the pole with these constraints, by appropriately having the cart accelerate left and right.

We have written a simple Matlab simulator for this problem. The simulation proceeds in discrete time cycles (steps). The state of the cart and pole at any time is completely characterized by 4 parameters: the cart position x, the cart velocity $\dot{x}$, the angle of the pole θ measured as its deviation from the vertical position, and the angular velocity of the pole $\dot{\theta}$. Since it'd be simpler to consider reinforcement learning in a discrete state space, we have approximated the state space by a discretization that maps a state vector $(x, \dot{x}, \theta, \dot{\theta})$ into a number from 1 to `NUM_STATES`. Your learning algorithm will need to deal only with this discretized representation of the states.

At every time step, the controller must choose one of two actions - push (accelerate) the cart right, or push the cart left. (To keep the problem simple, there is no *do-nothing* action.) These are represented as actions 1 and 2 respectively in the code. When the action choice is made, the simulator updates the state parameters according to the underlying dynamics, and provides a new discretized state.

We will assume that the reward $R(s)$ is a function of the current state only. When the pole angle goes beyond a certain limit or when the cart goes too far out, a negative reward is given, and the system is reinitialized randomly. At all other times, the reward is zero. Your program must learn to balance the pole using only the state transitions and rewards observed.

The files for this problem are in `/afs/ir/class/cs229/ps/ps4/q6`. Most of the the code has already been written for you, and you need to make changes only to `control.m` in the places specified. This file can be run in Matlab to show a display and to plot a learning curve at the end. Read the comments at the top of the file for more details on the working of the simulation.[4]

(a) To solve the inverted pendulum problem, you will estimate a model (i.e., transition probabilities and rewards) for the underlying MDP, solve Bellman's equations for this estimated MDP to obtain a value function, and act greedily with respect to this value function.

[4]Note that the routine for drawing the cart does not work in Octave.

Briefly, you will maintain a current model of the MDP and a current estimate of the value function. Initially, each state has estimated reward zero, and the estimated transition probabilities are uniform (equally likely to end up in any other state).

During the simulation, you must choose actions at each time step according to some current policy. As the program goes along taking actions, it will gather observations on transitions and rewards, which it can use to get a better estimate of the MDP model. Since it is inefficient to update the whole estimated MDP after every observation, we will store the state transitions and reward observations each time, and update the model and value function/policy only periodically. Thus, you must maintain counts of the total number of times the transition from state s_i to state s_j using action a has been observed (similarly for the rewards). Note that the rewards at any state are deterministic, but the state transitions are not because of the discretization of the state space (several different but close configurations may map onto the same discretized state).

Each time a failure occurs (such as if the pole falls over), you should re-estimate the transition probabilities and rewards as the average of the observed values (if any). Your program must then use value iteration to solve Bellman's equations on the estimated MDP, to get the value function and new optimal policy for the new model. For value iteration, use a convergence criterion that checks if the maximum absolute change in the value function on an iteration exceeds some specified tolerance.

Finally, assume that the whole learning procedure has converged once several consecutive attempts (defined by the parameter `NO_LEARNING_THRESHOLD`) to solve Bellman's equation all converge in the first iteration. Intuitively, this indicates that the estimated model has stopped changing significantly.

The code outline for this problem is already in `control.m`, and you need to write code fragments only at the places specified in the file. There are several details (convergence criteria etc.) that are also explained inside the code. Use a discount factor of $\gamma = 0.995$.

Implement the reinforcement learning algorithm as specified, and run it. How many trials (how many times did the pole fall over or the cart fall off) did it take before the algorithm converged?

Answer: The number of trials needed varies a good deal, but in the example run shown in the reference solution answer to part (b), 160 trials were needed.

```
%%%%%%% CS 229 Machine Learning %%%%%%%%%%
%%%%%%% Programming Assignment 4 %%%%%%%%%%
%%%
%%% Parts of the code (cart and pole dynamics, and the state
%%% discretization) are adapted from code available at the RL repository
%%% http://www-anw.cs.umass.edu/rlr/domains.html
%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This file controls the pole-balancing simulation. You need to write
% code in places marked "CODE HERE" only.
```

```
% Briefly, the main simulation loop in this file calls cart_pole.m for
% simulating the pole dynamics, get_state.m for discretizing the
% otherwise continuous state space in discrete states, and show_cart.m
% for display.

% Some useful parameters are listed below.

% NUM_STATES: Number of states in the discretized state space
% You must assume that states are numbered 1 through NUM_STATES. The
% state numbered NUM_STATES (the last one) is a special state that marks
% the state when the pole has been judged to have fallen (or when the
% cart is out of bounds). However, you should NOT treat this state any
% differently in your code. Any distinctions you need to make between
% states should come automatically from your learning algorithm.

% After each simulation cycle, you are supposed to update the transition
% counts and rewards observed. However, you should not change either
% your value function or the transition probability matrix at each
% cycle.

% Whenever the pole falls, a section of your code below will be
% executed. At this point, you must use the transition counts and reward
% observations that you have gathered to generate a new model for the MDP
% (i.e., transition probabilities and state rewards). After that, you
% must use value iteration to get the optimal value function for this MDP
% model.

% TOLERANCE: Controls the convergence criteria for each value iteration
% run
% In the value iteration, you can assume convergence when the maximum
% absolute change in the value function at any state in an iteration
% becomes lower than TOLERANCE.

% You need to write code that chooses the best action according
% to your current value function, and the current model of the MDP. The
% action must be either 1 or 2 (corresponding to possible directions of
% pushing the cart).

% Finally, we assume that the simulation has converged when
% 'NO_LEARNING_THRESHOLD' consecutive value function computations all
% converged within one value function iteration. Intuitively, it seems
% like there will be little learning after this, so we end the simulation
% here, and say the overall algorithm has converged.


% Learning curves can be generated by calling plot_learning_curve.m (it
% assumes that the learning was just executed, and the array
% time_steps_to_failure that records the time for which the pole was
% balanced before each failure are in memory). num_failures is a variable
```

```
% that stores the number of failures (pole drops / cart out of bounds)
% till now.

% Other parameters in the code are described below:

% GAMMA: Discount factor to be used

% The following parameters control the simulation display; you dont
% really need to know about them:

% pause_time: Controls the pause between successive frames of the
% display. Higher values make your simulation slower.
% min_trial_length_to_start_display: Allows you to start the display only
% after the pole has been successfully balanced for at least this many
% trials. Setting this to zero starts the display immediately. Choosing a
% reasonably high value (around 100) can allow you to rush through the
% initial learning quickly, and start the display only after the
% performance is reasonable.

%%%%%%%%% Simulation parameters %%%%%%%%%

pause_time = 0.001;
min_trial_length_to_start_display = 0;
display_started = min_trial_length_to_start_display == 0;

NUM_STATES = 163;

GAMMA=0.995;

TOLERANCE=0.01;

NO_LEARNING_THRESHOLD = 20;

%%%%%%%%%  End parameter list   %%%%%%%%%

% Time cycle of the simulation
time=0;

% These variables perform bookkeeping (how many cycles was the pole
% balanced for before it fell). Useful for plotting learning curves.
time_steps_to_failure=[];
num_failures=0;
time_at_start_of_current_trial=0;

max_failures=500; % You should reach convergence well before this.

% Starting state is (0 0 0 0)
% x, x_dot, theta, theta_dot represents the actual continuous state vector
x = 0.0; x_dot = 0.0; theta = 0.0; theta_dot = 0.0;
```

```
% state is the number given to this state - you only need to consider
% this representation of the state
state = get_state(x, x_dot, theta, theta_dot);

if display_started==1
  show_cart(x, x_dot, theta, theta_dot, pause_time);
end

%%% CODE HERE: Perform all your initializations here %%%

% Assume no transitions or rewards have been observed
% Initialize the value function array to small random values (0 to 0.10,
% say)
% Initialize the transition probabilities uniformly (ie, probability of
% transitioning for state x to state y using action a is exactly
% 1/NUM_STATES). Initialize all state rewards to zero.

transition_counts = zeros(NUM_STATES, NUM_STATES, 2);
transition_probs = ones(NUM_STATES, NUM_STATES, 2) / NUM_STATES;
reward_counts = zeros(NUM_STATES, 2);
reward = zeros(NUM_STATES, 1);
value = rand(NUM_STATES, 1) * 0.1;

%%%% END YOUR CODE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%% CODE HERE (while loop condition) %%%
% This is the criterion to end the simulation
% You should change it to terminate when the previous
% 'NO_LEARNING_THRESHOLD' consecutive value function computations all
% converged within one value function iteration. Intuitively, it seems
% like there will be little learning after this, so end the simulation
% here, and say the overall algorithm has converged.

consecutive_no_learning_trials = 0;
while (consecutive_no_learning_trials < NO_LEARNING_THRESHOLD)

  %%% CODE HERE: Write code to choose action (1 or 2) %%%

  % This action choice algorithm is just for illustration. It may
  % convince you that reinforcement learning is nice for control
  % problems!  Replace it with your code to choose an action that is
  % optimal according to the current value function, and the current MDP
  % model.
```

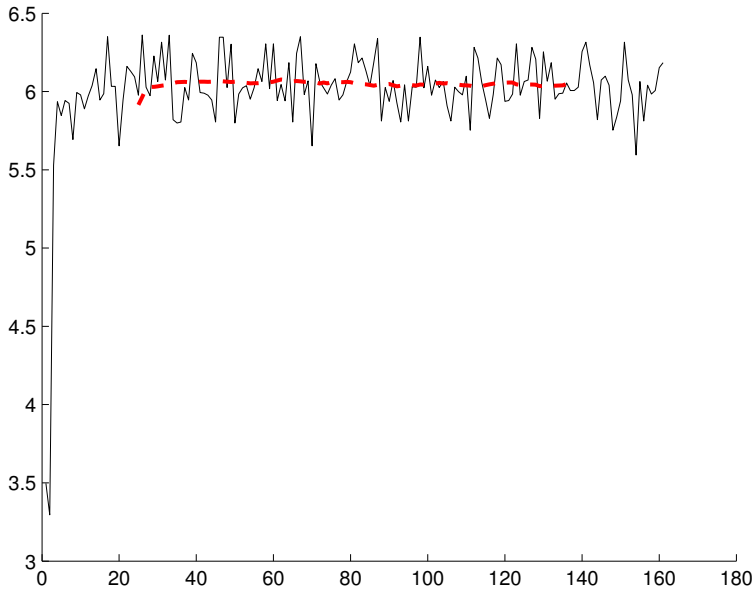
```
score1 = transition_probs(state, :, 1) * value;
score2 = transition_probs(state, :, 2) * value;
if (score1 > score2)
  action = 1;
elseif (score2 > score1)
  action = 2;
else
  if (rand < 0.5)
    action = 1;
  else
    action = 2;
  end
end

%%% END YOUR CODE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Get the next state by simulating the dynamics
[x, x_dot, theta, theta_dot] = cart_pole(action, x, x_dot, theta, theta_dot);

% Increment simulation time
time = time + 1;

% Get the state number corresponding to new state vector
new_state = get_state(x, x_dot, theta, theta_dot);

if display_started==1
  show_cart(x, x_dot, theta, theta_dot, pause_time);
end

% Reward function to use - do not change this!
if (new_state==NUM_STATES)
  R=-1;
else
  %R=-abs(theta)/2.0;
  R=0;
end


%%% CODE HERE: Perform updates %%%%%%%%%

% A transition from 'state' to 'new_state' has just been made using
% 'action'. The reward observed in 'new_state' (note) is 'R'.
% Write code to update your statistics about the MDP - i.e., the
% information you are storing on the transitions and on the rewards
% observed. Do not change the actual MDP parameters, except when the
% pole falls (the next if block)!

transition_counts(state, new_state, action) = ...
```

```
  transition_counts(state, new_state, action) + 1;
reward_counts(new_state, 1) = reward_counts(new_state, 1) + R;
reward_counts(new_state, 2) = reward_counts(new_state, 2) + 1;

% Recompute MDP model whenever pole falls
% Compute the value function V for the new model
if (new_state==NUM_STATES)

  % Update MDP model using the current accumulated statistics about the
  % MDP - transitions and rewards.
  % Make sure you account for the case when total_count is 0, i.e., a
  % state-action pair has never been tried before, or the state has
  % never been visited before. In that case, you must not change that
  % component (and thus keep it at the initialized uniform distribution).

  for a = 1:2
    for s = 1:NUM_STATES
      den = sum(transition_counts(s, :, a));
      if (den > 0)
        transition_probs(s, :, a) = transition_counts(s, :, a) / den;
      end
    end
  end

  for s = 1:NUM_STATES
    if (reward_counts(s, 2) > 0)
      reward(s) = reward_counts(s, 1) / reward_counts(s, 2);
    end
  end

  % Perform value iteration using the new estimated model for the MDP
  % The convergence criterion should be based on TOLERANCE as described
  % at the top of the file.
  % If it converges within one iteration, you may want to update your
  % variable that checks when the whole simulation must end

  iterations = 0;
  new_value = zeros(NUM_STATES, 1);
  while true
    iterations = iterations + 1;
    for s = 1:NUM_STATES
      value1 = transition_probs(s, :, 1) * value;
      value2 = transition_probs(s, :, 2) * value;
      new_value(s) = max(value1, value2);
    end
    new_value = reward + GAMMA * new_value;
    diff = max(abs(value - new_value));
    value = new_value;
    if (diff < TOLERANCE)
```

```
      break;
    end
  end

  if (iterations == 1)
    consecutive_no_learning_trials = consecutive_no_learning_trials + 1;
  else
    consecutive_no_learning_trials = 0;
  end

  % pause(0.2); % You can use this to stop for a while!

end


%%% END YOUR CODE %%%%%%%%%%%%%%%%%%%


% Dont change this code: Controls the simulation, and handles the case
% when the pole fell and the state must be reinitialized
if (new_state == NUM_STATES)
  num_failures = num_failures+1
  time_steps_to_failure(num_failures) = time - time_at_start_of_current_trial;
  time_at_start_of_current_trial = time;

  time_steps_to_failure(num_failures)

  if (time_steps_to_failure(num_failures) > ...
      min_trial_length_to_start_display)
    display_started=1;
  end

  % Reinitialize state
  x = -1.1 + rand(1)*2.2
  %x=0.0;
  x_dot = 0.0; theta = 0.0; theta_dot = 0.0;
  state = get_state(x, x_dot, theta, theta_dot);
else
  state=new_state;
end
end

% Plot the learning curve (time balanced vs trial)
plot_learning_curve
```

(b) Plot a learning curve showing the number of time-steps for which the pole was balanced on each trial. You just need to execute `plot_learning_curve.m` after `control.m` to get this plot.

Answer:



STANFORD UNIVERSITY
CS 229, Autumn 2014
Midterm Examination

Wednesday, November 5, 6:00pm-9:00pm

Question	Points
1 Least Squares	/16
2 Generative Learning	/16
3 Generalized Linear Models	/18
4 Support Vector Regression	/16
5 Learning Theory	/20
6 Short Answers	/24
Total	/110

Name of Student: _____

SUNetID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signed: _____

1. **[16 points] Least Squares**

   As described in class, in least squares regression we have a cost function:

   $$J(\theta) = \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = (X\theta - \vec{y})^T (X\theta - \vec{y})$$

   The goal of least squares regression is to find θ such that we minimize $J(\theta)$ given the training data.

   Let's say that we had an original set of n features, so that the training inputs were represented by the design matrix $X \in \mathbb{R}^{m \times (n+1)}$. However, we now gain access to one additional feature for every example. As a result, we now have an additional vector of features $\vec{v} \in \mathbb{R}^{m \times 1}$ for our training set that we wish to include in our regression. We can do this by creating a new design matrix: $\widetilde{X} = [X\ \vec{v}] \in \mathbb{R}^{m \times (n+2)}$.

   Therefore the new parameter vector is $\theta_{new} = \begin{pmatrix} \theta \\ p \end{pmatrix}$ where $p \in \mathbb{R}$ is the parameter corresponding to the new feature vector $\vec{v}$.

   Note: For mathematical simplicity, throughout this problem you can assume that $X^T X = I \in \mathbb{R}^{(n+1)\times(n+1)}$ and $\widetilde{X}^T \widetilde{X} = I \in \mathbb{R}^{(n+2)\times(n+2)}, \vec{v}^T \vec{v} = 1$. This is called an *orthonormality* assumption – specifically, the columns of $\widetilde{X}$ are orthonormal. The conclusions of the problem hold even if we do not make this assumption, but this will make your derivations easier.

   (a) [2 points] Let $\hat{\theta} = \arg\min_\theta J(\theta)$ be the minimizer of the original least squares objective (using the original design matrix X). Using the orthornormality assumption, show that $J(\hat{\theta}) = (XX^T\vec{y} - \vec{y})^T (XX^T\vec{y} - \vec{y})$. I.e., show that this is the value of $\min_\theta J(\theta)$ (the value of the objective at the minimum).

(b) [5 points] Now let $\hat{\theta}_{new}$ be the minimizer for $\widetilde{J}(\theta_{new}) = (\widetilde{X}\theta_{new} - \vec{y})^T(\widetilde{X}\theta_{new} - \vec{y})$. Find the new minimized objective $\widetilde{J}(\hat{\theta}_{new})$ and write this expression in the form: $\widetilde{J}(\hat{\theta}_{new}) = J(\hat{\theta}) + f(X, \vec{v}, \vec{y})$ where $J(\hat{\theta})$ is as derived in part (a) and f is some function of $X, \vec{v}$, and $\vec{y}$.

(c) [6 points] Prove that the optimal objective value does not increase upon adding a feature to the design matrix. That is, show $\widetilde{J}(\hat{\theta}_{new}) \leq J(\hat{\theta})$.

(d) [3 points] Does the above result show that if we keep increasing the number of features, we can always get a model that generalizes better than a model with fewer features? Explain why or why not.

2. **[16 points] Decision Boundaries for Generative Models**

   (a) [8 points] Consider the *multinomial event model* of Naive Bayes. Our goal in this problem is to show that this is a linear classifier.

   For a given text document x, let $c_1, ..., c_V$ indicate the number of times each word (out of V words) appears in the document. Thus, $c_i \in \{0, 1, 2, \ldots\}$ counts the occurrences of word i. Recall that the Naive Bayes model uses parameters $\phi_y = p(y = 1)$, $\phi_{i|y=1} = p(\text{word i appears in a specific document position} \mid y = 1)$ and $\phi_{i|y=0} = p(\text{word i appears in a specific document position} \mid y = 0)$.

   We say a classifier is linear if it assigns a label $y = 1$ using a decision rule of the form

$$\sum_{i=1}^{V} w_i c_i + b \geq 0$$

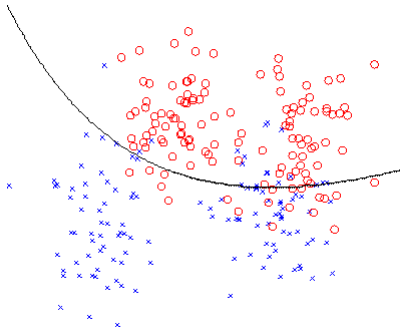
   I.e., the classifier predicts "$y = 1$" if $\sum_{i=1}^{V} w_i c_i + b \geq 0$, and predicts $y = 0$ otherwise.

   Show that Naive Bayes is a linear classifier, and clearly state the values of w_i and b in terms of the Naive Bayes parameters. (Don't worry about whether the decision rule uses "$\geq$" or "$>$.") Hint: consider using log-probabilities.

[extra space for 2 (a)]

(b) [8 points] In Problem Set 1, you showed that Gaussian Discriminant Analysis (GDA) is a linear classifier. In this problem, we will show that a modified version of GDA has a quadratic decision boundary.

Recall that GDA models $p(x|y)$ using a multivariate normal distribution, where $(x|y=0) \sim \mathcal{N}(\mu_0, \Sigma)$ and $(x|y=1) \sim \mathcal{N}(\mu_1, \Sigma)$, where we used the same Σ for both Gaussians. For this question, we will instead use two covariance matrices Σ_0, Σ_1 for the two labels. So, $(x|y=0) \sim \mathcal{N}(\mu_0, \Sigma_0)$ and $(x|y=1) \sim \mathcal{N}(\mu_1, \Sigma_1)$.



The model distributions can now be written as:

$$
\begin{aligned}
p(y) &= \phi^y (1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma_0|^{1/2}} \exp\Big(-\frac{1}{2}(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)\Big) \\
p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma_1|^{1/2}} \exp\Big(-\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\Big)
\end{aligned}
$$

Let's follow a binary decision rule, where we predict $y=1$ if $p(y=1|x) \geq p(y=0|x)$, and $y=0$ otherwise. Show that if $\Sigma_0 \neq \Sigma_1$, then the separating hyperplane is quadratic in x.

That is, simplify the decision rule "$p(y=1|x) \geq p(y=0|x)$" to the form "$x^T A x + B^T x + C \geq 0$" (supposing that $x \in \mathbb{R}^{n+1}$), for some $A \in \mathbb{R}^{(n+1)\times(n+1)}$, $B \in \mathbb{R}^{n+1}$, $C \in \mathbb{R}$ and $A \neq 0$. Please clearly state your values for A, B and C.

[extra space for 2 (b)]

3. **[18 points] Generalized Linear Models**
   In this problem you will build a Generalized Linear Model (GLM) for a response variable y, whose distribution (parameterized by ϕ) is modeled as:

$$p(y;\phi) = (1-\phi)^{y-1}\phi$$

   This distribution is known as the *geometric distribution*, and is used to model network connections and many other problems.

   (a) i. [5 points] Show that the geometric distribution is an exponential family distribution. You should explicitly specify $b(y)$, η, $T(y)$, $\alpha(\eta)$. Also specify what ϕ is in terms of η.

ii. [5 points] Suppose that we have an IID training set $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$ and we wish to model this using a GLM based on a geometric distribution. Find the log-likelihood $\log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta)$ defined with respect to the entire training set.

(b) [6 points] Derive the Hessian H and the gradient vector of the log likelihood, and state what one step of Newton's method for maximizing the log likelihood would be.

(c) [2 points] Show that the Hessian is negative semi-definite. This shows the optimization objective is concave, and hence Newton's method is maximizing log-likelihood.

4. **[16 points] Support Vector Regression**

   In class, we showed how the SVM can be used for classification. In this problem, we will develop a modified algorithm, called the Support Vector Regression algorithm, which can instead be used for regression, with continuous valued labels $y \in \mathbb{R}$.

   Suppose we are given a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, where $x^{(i)} \in \mathbb{R}^{(n+1)}$ and $y^{(i)} \in \mathbb{R}$. We would like to find a hypothesis of the form $h_{w,b}(x) = w^T x + b$ with a small value of w. Our (convex) optimization problem is:

$$\begin{array}{lll} \min_{w,b} & \frac{1}{2}\|w\|^2 & \\ \text{s.t.} & y^{(i)} - w^T x^{(i)} - b \leq \epsilon \qquad & i = 1, \ldots, m \quad (1) \\ & w^T x^{(i)} + b - y^{(i)} \leq \epsilon \qquad & i = 1, \ldots, m \quad (2) \end{array}$$

   where $\epsilon > 0$ is a given, fixed value. Notice how the original functional margin constraint has been modified to now represent the distance between the continuous y and our hypothesis' output.

   (a) [3 points] Write down the Lagrangian for the optimization problem above. We suggest you use two sets of Lagrange multipliers α_i and α_i^*, corresponding to the two inequality constraints (labeled (1) and (2) above), so that the Lagrangian would be written $\mathcal{L}(w, b, \alpha, \alpha^*)$.

(b) [9 points] Derive the dual optimization problem. You will have to take derivatives of the Lagrangian with respect to w and b.

[more space for problem 4 (b)]

(c) [4 points] Show that this algorithm can be kernelized. For this, you have to show that (i) the dual optimization objective can be written in terms of inner-products of training examples; and (ii) at test time, given a new x the hypothesis $h_{w,b}(x)$ can also be computed in terms of inner products.

5. **[20 points] Learning Theory**

Suppose you are given a hypothesis $h_0 \in \mathcal{H}$, and your goal is to determine whether h_0 has generalization error within $\eta > 0$ of the best hypothesis, $h^* = \arg\min_{h \in \mathcal{H}} \varepsilon(h)$. More specifically, we say that a hypothesis h is η**-optimal** if $\varepsilon(h) \leq \varepsilon(h^*) + \eta$. Here, we wish to answer the following question:

Given a hypothesis h_0, is h_0 η-optimal?

Let $\delta > 0$ be some fixed constant, and consider a finite hypothesis class $\mathcal{H}$ of size $|\mathcal{H}| = k$. For each $h \in \mathcal{H}$, let $\hat{\varepsilon}(h)$ denote the training error of h with respect to some training set of m IID examples, and let $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$ denote the hypothesis that minimizes training error.

Now, consider the following algorithm:

1. Set
$$\gamma := \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$
2. If $\hat{\varepsilon}(h_0) > \hat{\varepsilon}(\hat{h}) + \eta + 2\gamma$, then return NO.
3. If $\hat{\varepsilon}(h_0) < \hat{\varepsilon}(\hat{h}) + \eta - 2\gamma$, then return YES.
4. Otherwise, return UNSURE.

Intuitively, the algorithm works by comparing the training error of h_0 to the training error of the hypothesis $\hat{h}$ with the minimum training error, and returns NO or YES only when $\hat{\varepsilon}(h_0)$ is either significantly larger than or significantly smaller than $\hat{\varepsilon}(\hat{h})+\eta$.

(a) [6 points] First, show that if $\varepsilon(h_0) \leq \varepsilon(h^*) + \eta$ (i.e., h_0 is η-optimal), then the probability that the algorithm returns NO is at most δ.

[extra space for 5 (a)]

(b) [6 points] Second, show that if $\varepsilon(h_0) > \varepsilon(h^*) + \eta$ (i.e., h_0 is not η-optimal), then the probability that the algorithm returns YES is at most δ.

(c) [8 points] Finally, suppose that $h_0 = h^*$, and let $\eta > 0$ and $\delta > 0$ be fixed. Show that if m is sufficiently large, then the probability that the algorithm returns YES is at least $1 - \delta$.

Hint: observe that for fixed η and δ, as $m \to \infty$, we have

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}} \to 0.$$

This means that there are values of m for which $2\gamma < \eta - 2\gamma$.

6. **[24 points] Short answers**

   The following questions require a reasonably short answer (usually at most 2-3 sentences or a figure, though some questions may require longer or shorter explanations).

   To discourage random guessing, one point will be deducted for a wrong answer on true/false or multiple choice questions! Also, no credit will be given for answers without a correct explanation.

   (a) [3 points] You have an implementation of Newton's method and gradient descent. Suppose that one iteration of Newton's method takes twice as long as one iteration of gradient descent. Then, this implies that gradient descent will converge to the optimal objective faster. True/False?

   (b) [3 points] A stochastic gradient descent algorithm for training logistic regression with a fixed learning rate will always converge to exactly the optimal setting of the parameters $\theta^* = \arg\max_\theta \prod_{i=1}^m p(y^{(i)}|x^{(i)};\theta)$, assuming a reasonable choice of the learning rate. True/False?

(c) [3 points] Given a valid kernel $K(x, y)$ over $\mathbb{R}^m$, is $K_{norm}(x, y) = \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}}$ a valid kernel?

(d) [3 points] Consider a 2 class classification problem with a dataset of inputs $\{x^{(1)} = (-1, -1), x^{(2)} = (-1, +1), x^{(3)} = (+1, -1), x^{(4)} = (+1, +1)\}$. Can a linear SVM (with no kernel trick) shatter this set of 4 points?

(e) [3 points] For linear hypotheses (i.e. of the form $h(x) = w^T x + b$), the vector of learned weights w is always perpendicular to the separating hyperplane. True/False? Provide a counterexample if False, or a brief explanation if True.

(f) [3 points] Let $\mathcal{H}$ be a set of classifiers whose VC-dimension is 5. Suppose we have four training examples and labels, $\{(x^{(1)}, y^{(1)}), ..., (x^{(4)}, y^{(4)})\}$, and we select a classifier h from $\mathcal{H}$ by minimizing the classification error on the training set. In the absence of any other information about the set of classifiers $\mathcal{H}$, can we say that

i. $x^{(5)}$ will certainly be classified correctly?
ii. $x^{(5)}$ will certainly be classified incorrectly?
iii. we cannot tell?

Briefly justify your answer.

(g) [6 points] Suppose you would like to use a linear regression model in order to predict the price of houses. In your model, you use the features $x_0 = 1$, $x_1 =$ size in square meters, $x_2 =$ height of roof in meters. Now, suppose a friend repeats the same analysis using exactly the same training set, only he represents the data instead using features $x'_0 = 1$, $x'_1 = x_1$, and $x'_2 =$ height in cm (so $x'_2 = 100x_2$).

i. [3 points] Suppose both of you run linear regression, solving for the parameters via the Normal equations. (Assume there are no degeneracies, so this gives a unique solution to the parameters.) You get parameters θ_0, θ_1, θ_2; your friend gets θ'_0, θ'_1, θ'_2. Then $\theta'_0 = \theta_0$, $\theta'_1 = \theta_1$, $\theta'_2 = \frac{1}{100}\theta_2$. True/False?

ii. [3 points] Suppose both of you run linear regression, initializing the parameters to 0, and compare your results after running just *one* iteration of batch gradient descent. You get parameters θ_0, θ_1, θ_2; your friend gets θ'_0, θ'_1, θ'_2. Then $\theta'_0 = \theta_0$, $\theta'_1 = \theta_1$, $\theta'_2 = \frac{1}{100}\theta_2$. True/False?

STANFORD UNIVERSITY
CS 229, Autumn 2014
Midterm Examination

Wednesday, November 5, 6:00pm-9:00pm

Question	Points
1 Least Squares	/16
2 Generative Learning	/16
3 Generalized Linear Models	/18
4 Support Vector Regression	/16
5 Learning Theory	/20
6 Short Answers	/24
Total	/110

Name of Student: _____

SUNetID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signed: _____

1. **[16 points] Least Squares**

   As described in class, in least squares regression we have a cost function:

   $$J(\theta) = \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 = (X\theta - \vec{y})^T (X\theta - \vec{y})$$

   The goal of least squares regression is to find θ such that we minimize $J(\theta)$ given the training data.

   Let's say that we had an original set of n features, so that the training inputs were represented by the design matrix $X \in \mathbb{R}^{m \times (n+1)}$. However, we now gain access to one additional feature for every example. As a result, we now have an additional vector of features $\vec{v} \in \mathbb{R}^{m \times 1}$ for our training set that we wish to include in our regression. We can do this by creating a new design matrix: $\widetilde{X} = [X \ \vec{v}] \in \mathbb{R}^{m \times (n+2)}$.

   Therefore the new parameter vector is $\theta_{new} = \begin{pmatrix} \theta \\ p \end{pmatrix}$ where $p \in \mathbb{R}$ is the parameter corresponding to the new feature vector $\vec{v}$.

   Note: For mathematical simplicity, throughout this problem you can assume that $X^T X = I \in \mathbb{R}^{(n+1)\times(n+1)}$ and $\widetilde{X}^T \widetilde{X} = I \in \mathbb{R}^{(n+2)\times(n+2)}, \vec{v}^T \vec{v} = 1$. This is called an *orthonormality* assumption – specifically, the columns of $\widetilde{X}$ are orthonormal. The conclusions of the problem hold even if we do not make this assumption, but this will make your derivations easier.

   (a) [2 points] Let $\hat{\theta} = \arg\min_\theta J(\theta)$ be the minimizer of the original least squares objective (using the original design matrix X). Using the orthornormality assumption, show that $J(\hat{\theta}) = (XX^T\vec{y} - \vec{y})^T(XX^T\vec{y} - \vec{y})$. I.e., show that this is the value of $\min_\theta J(\theta)$ (the value of the objective at the minimum).

   Answer: We know from lecture that the least squares minimizer is $\hat{\theta} = (X^T X)^{-1} X^T \vec{y}$ but because of the orthonormality assumption, this simplifies to $\hat{\theta} = X^T \vec{y}$. Substituting this expression into the normal equation for $J(\theta)$ gives the final expression $J(\hat{\theta}) = (XX^T\vec{y} - \vec{y})^T(XX^T\vec{y} - \vec{y})$.

(b) [5 points] Now let $\hat{\theta}_{new}$ be the minimizer for $\widetilde{J}(\theta_{new}) = (\widetilde{X}\theta_{new} - \vec{y})^T(\widetilde{X}\theta_{new} - \vec{y})$. Find the new minimized objective $\widetilde{J}(\hat{\theta}_{new})$ and write this expression in the form: $\widetilde{J}(\hat{\theta}_{new}) = J(\hat{\theta}) + f(X, \vec{v}, \vec{y})$ where $J(\hat{\theta})$ is as derived in part (a) and f is some function of $X, \vec{v}$, and $\vec{y}$.

Answer: Just like we had in part (a), the minimizer for the new objective is $\hat{\theta}_{new} = \widetilde{X}^T\vec{y}$. Now we solve for the new minimized objective:

$$\begin{aligned}
\widetilde{J}(\hat{\theta}_{new}) &= (\widetilde{X}\hat{\theta}_{new} - \vec{y})^T(\widetilde{X}\hat{\theta}_{new} - \vec{y}) \\
&= (\widetilde{X}\widetilde{X}^T\vec{y} - \vec{y})^T(\widetilde{X}\widetilde{X}^T\vec{y} - \vec{y}) \\
&= ((XX^T + \vec{v}\vec{v}^T)\vec{y} - \vec{y})^T((XX^T + \vec{v}\vec{v}^T)\vec{y} - \vec{y}) \\
&= ((XX^T\vec{y} - \vec{y}) + \vec{v}\vec{v}^T\vec{y})^T((XX^T\vec{y} - \vec{y}) + \vec{v}\vec{v}^T\vec{y}) \\
&= (XX^T\vec{y} - \vec{y})^T(XX^T\vec{y} - \vec{y}) + 2(XX^T\vec{y} - \vec{y})^T(\vec{v}\vec{v}^T\vec{y}) + (\vec{v}\vec{v}^T\vec{y})^T(\vec{v}\vec{v}^T\vec{y}) \\
&= J(\hat{\theta}) + 2(XX^T\vec{y} - \vec{y})^T(\vec{v}\vec{v}^T\vec{y}) + (\vec{v}\vec{v}^T\vec{y})^T(\vec{v}\vec{v}^T\vec{y})
\end{aligned} \tag{1}$$

(c) [6 points] Prove that the optimal objective value does not increase upon adding a feature to the design matrix. That is, show $\widetilde{J}(\hat{\theta}_{new}) \leq J(\hat{\theta})$.

Answer: Using the final result of part (b), we can continue simplifying the expression for $J(\hat{\theta}_{new})$ as follows:

$$\begin{aligned}
\widetilde{J}(\hat{\theta}_{new}) &= J(\hat{\theta}) + 2(XX^T\vec{y} - \vec{y})^T(vv^T\vec{y}) + (vv^T\vec{y})^T(vv^T\vec{y}) \\
&= J(\hat{\theta}) + 2(XX^T\vec{y})^T(vv^T\vec{y}) - 2\vec{y}^T(vv^T\vec{y}) + (vv^T\vec{y})^T(vv^T\vec{y}) \\
&= J(\hat{\theta}) + 2(\vec{y}^TXX^Tvv^T\vec{y}) - 2(\vec{y}^Tvv^T\vec{y}) + (\vec{y}^Tvv^Tvv^T\vec{y}) \\
&= J(\hat{\theta}) - \vec{y}^Tvv^T\vec{y} \\
&= J(\hat{\theta}) - (v^T\vec{y})^2 \\
&\leq J(\hat{\theta})
\end{aligned} \tag{2}$$

From the third to last equality to the second to last equality, we use the two facts that $X^Tv = 0$ and $v^Tv = 1$.

The proof is complete.

(d) [3 points] Does the above result show that if we keep increasing the number of features, we can always get a model that generalizes better than a model with fewer features? Explain why or why not.

Answer: The result shows that we can either maintain or decrease the minimized square error objective by adding more features. However, remember that the error objective is computed only on the training samples and not the true data distribution. As a result, reducing training error does not guarantee a reduction in error on the true distribution. In fact, after a certain point adding features will likely lead to overfitting, increasing our generalization error. Therefore, adding features does not actually always result in a model that generalizes better.

2. **[16 points] Decision Boundaries for Generative Models**

(a) [8 points] Consider the *multinomial event model* of Naive Bayes. Our goal in this problem is to show that this is a linear classifier.

For a given text document x, let $c_1, ..., c_V$ indicate the number of times each word (out of V words) appears in the document. Thus, $c_i \in \{0, 1, 2, \ldots\}$ counts the occurrences of word i. Recall that the Naive Bayes model uses parameters $\phi_y = p(y = 1)$, $\phi_{i|y=1} = p(\text{word i appears in a specific document position} \mid y = 1)$ and $\phi_{i|y=0} = p(\text{word i appears in a specific document position} \mid y = 0)$.

We say a classifier is linear if it assigns a label $y = 1$ using a decision rule of the form

$$\sum_{i=1}^{V} w_i c_i + b \geq 0$$

I.e., the classifier predicts "$y = 1$" if $\sum_{i=1}^{V} w_i c_i + b \geq 0$, and predicts $y = 0$ otherwise.

Show that Naive Bayes is a linear classifier, and clearly state the values of w_i and b in terms of the Naive Bayes parameters. (Don't worry about whether the decision rule uses "$\geq$" or "$>$.") Hint: consider using log-probabilities.

Answer: The decision boundary for Naive Bayes can be stated as

$$\begin{aligned}
P(y = 1|c; \Phi) &> P(y = 0|c; \Phi) \\
\log p(y = 1|c; \Phi) &> \log p(y = 0|c; \Phi) \\
\log p(y = 1|c; \Phi) - \log p(y = 0|c; \Phi) &> 0 \\
\log \frac{p(y = 1|c; \Phi)}{p(y = 0|c; \Phi)} &> 0 \\
\log \frac{p(y = 1) \prod_{i=1}^{V} p(E_i|y = 1)^{c_i}}{p(y = 0) \prod_{i=1}^{V} p(E_i|y = 0)^{c_i}} &> 0 \\
\log \frac{p(y = 1)}{p(y = 0)} + \sum_{i=1}^{V} \log \frac{p(E_i|y = 1)^{c_i}}{p(E_i|y = 0)^{c_i}} &> 0 \\
\log \frac{\phi_y}{1 - \phi_y} + \sum_{i=1}^{V} c_i \log \frac{\phi_{i|y=1}}{\phi_{i|y=0}} &> 0
\end{aligned}$$

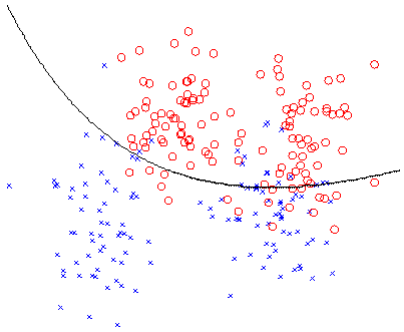
Thus, Naive Bayes is a linear classifier with

$$\begin{aligned}
w_i &= \log \frac{\phi_{i|y=1}}{\phi_{i|y=0}} \\
b &= \log \frac{\phi_y}{1 - \phi_y}
\end{aligned}$$

[extra space for 2 (a)]

(b) [8 points] In Problem Set 1, you showed that Gaussian Discriminant Analysis (GDA) is a linear classifier. In this problem, we will show that a modified version of GDA has a quadratic decision boundary.

Recall that GDA models $p(x|y)$ using a multivariate normal distribution, where $(x|y=0) \sim \mathcal{N}(\mu_0, \Sigma)$ and $(x|y=1) \sim \mathcal{N}(\mu_1, \Sigma)$, where we used the same Σ for both Gaussians. For this question, we will instead use two covariance matrices Σ_0, Σ_1 for the two labels. So, $(x|y=0) \sim \mathcal{N}(\mu_0, \Sigma_0)$ and $(x|y=1) \sim \mathcal{N}(\mu_1, \Sigma_1)$.



The model distributions can now be written as:

$$
\begin{aligned}
p(y) &= \phi^y (1-\phi)^{1-y} \\
p(x|y=0) &= \frac{1}{(2\pi)^{n/2}|\Sigma_0|^{1/2}} \exp\Big(-\frac{1}{2}(x-\mu_0)^T \Sigma_0^{-1}(x-\mu_0)\Big) \\
p(x|y=1) &= \frac{1}{(2\pi)^{n/2}|\Sigma_1|^{1/2}} \exp\Big(-\frac{1}{2}(x-\mu_1)^T \Sigma_1^{-1}(x-\mu_1)\Big)
\end{aligned}
$$

Let's follow a binary decision rule, where we predict $y=1$ if $p(y=1|x) \geq p(y=0|x)$, and $y=0$ otherwise. Show that if $\Sigma_0 \neq \Sigma_1$, then the separating hyperplane is quadratic in x.

That is, simplify the decision rule "$p(y=1|x) \geq p(y=0|x)$" to the form "$x^T A x + B^T x + C \geq 0$" (supposing that $x \in \mathbb{R}^{n+1}$), for some $A \in \mathbb{R}^{(n+1)\times(n+1)}$, $B \in \mathbb{R}^{n+1}$, $C \in \mathbb{R}$ and $A \neq 0$. Please clearly state your values for A, B and C.

[extra space for 2 (b)]

Answer: Examining the log-probabilities yields

$\log p(y=1|x) \geq \log p(y=0|x)$

$$
\begin{aligned}
0 \leq & \log\Big(\frac{p(y=1|x)}{p(y=0|x)}\Big) \\
0 \leq & \log\Big(\frac{p(y=1)p(x|y=1)}{p(y=0)p(x|y=0)}\Big) \\
0 \leq & \log\Big(\frac{\phi}{1-\phi}\Big) - \log\Big(\frac{|\Sigma_1|^{1/2}}{|\Sigma_0|^{1/2}}\Big) \\
& - \frac{1}{2}\Big((x-\mu_1)^T\Sigma_1^{-1}(x-\mu_1) - (x-\mu_0)^T\Sigma_0^{-1}(x-\mu_0)\Big) \\
0 \leq & -\frac{1}{2}\Big(x^T(\Sigma_1^{-1}-\Sigma_0^{-1})x - 2(\mu_1^T\Sigma_1^{-1} - \mu_0^T\Sigma_0^{-1})x \\
& + \mu_1^T\Sigma_1^{-1}\mu_1 - \mu_0^T\Sigma_0^{-1}\mu_0\Big) + \log\Big(\frac{\phi}{1-\phi}\Big) - \log\Big(\frac{|\Sigma_1|^{1/2}}{|\Sigma_0|^{1/2}}\Big) \\
0 \leq & x^T\Big(\frac{1}{2}(\Sigma_0^{-1}-\Sigma_1^{-1})\Big)x + \Big(\mu_1^T\Sigma_1^{-1} - \mu_0^T\Sigma_0^{-1}\Big)x \\
& + \log\Big(\frac{\phi}{1-\phi}\Big) + \log\Big(\frac{|\Sigma_0|^{1/2}}{|\Sigma_1|^{1/2}}\Big) + \frac{1}{2}\Big(\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1\Big)
\end{aligned}
$$

From the above, we see that $A = \frac{1}{2}(\Sigma_0^{-1} - \Sigma_1^{-1})$, $B = \mu_1^T\Sigma_1^{-1} - \mu_0^T\Sigma_0^{-1}$, and $C = \log(\frac{\phi}{1-\phi}) + \log(\frac{|\Sigma_0|^{1/2}}{|\Sigma_1|^{1/2}}) + \frac{1}{2}(\mu_0^T\Sigma_0^{-1}\mu_0 - \mu_1^T\Sigma_1^{-1}\mu_1)$. Furthermore, $A \neq 0$ since $\Sigma_0 \neq \Sigma_1$ implies that $\Sigma_0^{-1} - \Sigma_1^{-1} \neq 0$. Therefore, the decision boundary is quadratic.

3. **[18 points] Generalized Linear Models**
In this problem you will build a Generalized Linear Model (GLM) for a response variable y, whose distribution (parameterized by ϕ) is modeled as:

$$p(y;\phi) = (1-\phi)^{y-1}\phi$$

This distribution is known as the *geometric distribution*, and is used to model network connections and many other problems.

(a) i. [5 points] Show that the geometric distribution is an exponential family distribution. You should explicitly specify $b(y)$, η, $T(y)$, $\alpha(\eta)$. Also specify what ϕ is in terms of η.
Answer:

$$\begin{aligned} p(y;\phi) &= (1-\phi)^{y-1}\phi \\ &= \exp\Big((y-1)\log(1-\phi) + \log\phi\Big) \\ &= \exp\Big((\log(1-\phi))y + \log\phi - \log(1-\phi)\Big) \\ &= \exp\Big((\log(1-\phi))y - \log\frac{1-\phi}{\phi}\Big) \end{aligned}$$

$$\begin{aligned} b(y) &= 1, \\ \eta &= \log(1-\phi), \\ T(y) &= y, \\ \alpha(\eta) &= \log\frac{1-\phi}{\phi}, \\ \phi &= 1-e^{\eta} \end{aligned}$$

ii. [5 points] Suppose that we have an IID training set $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$ and we wish to model this using a GLM based on a geometric distribution. Find the log-likelihood $\log \prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta)$ defined with respect to the entire training set.

Answer: We calculate the log-likelihood for 1 sample as well as for the entire training set:

$$\begin{aligned}
\log p(y^{(i)}|x^{(i)}; \theta) &= \log \Big((1-\phi)^{y^{(i)}-1} \phi \Big) \\
&= (y^{(i)} - 1) \log (1-\phi) + \log \phi \\
&= (\log (1-\phi)) y^{(i)} - \log \frac{1-\phi}{\phi} \\
&= y^{(i)} \log e^{\eta} - \log \frac{e^{\eta}}{1-e^{\eta}} \\
&= \eta y^{(i)} - \eta + \log (1 - e^{\eta}) \\
&= \theta^T x^{(i)} y^{(i)} - \theta^T x^{(i)} + \log (1 - \exp (\theta^T x^{(i)})) \\
&= \theta^T x^{(i)} (y^{(i)} - 1) + \log (1 - \exp (\theta^T x^{(i)}))
\end{aligned}$$

$$\begin{aligned}
l(\theta) &= \log p(y|x; \theta) \\
&= \log \Big(\prod_{i=1}^{m} p(y^{(i)}|x^{(i)}; \theta) \Big) \\
&= \sum_{i=1}^{m} \log \Big(p(y^{(i)}|x^{(i)}; \theta) \Big) \\
&= \sum_{i=1}^{m} \Big(\theta^T x^{(i)} (y^{(i)} - 1) + \log (1 - \exp (\theta^T x^{(i)})) \Big)
\end{aligned}$$

(b) [6 points] Derive the Hessian H and the gradient vector of the log likelihood, and state what one step of Newton's method for maximizing the log likelihood would be.

Answer: To apply Newton's method, we need to find the gradient and Hessian of the log-likelihood:

$$\begin{aligned}
\nabla_\theta l(\theta) &= \nabla_\theta \sum_{i=1}^{m} \Big(\theta^T x^{(i)} y^{(i)} - \theta^T x^{(i)} + \log\left(1 - \exp\left(\theta^T x^{(i)}\right)\right) \Big) \\
&= \sum_{i=1}^{m} \Big(x^{(i)}(y^{(i)} - 1) - \frac{x^{(i)} \exp\left(\theta^T x^{(i)}\right)}{\left(1 - \exp\left(\theta^T x^{(i)}\right)\right)} \Big) \\
&= \sum_{i=1}^{m} \Big(y^{(i)} - \frac{1}{\left(1 - \exp\left(\theta^T x^{(i)}\right)\right)} \Big) x^{(i)} \\
H &= \nabla_\theta \Big(\nabla_\theta (l(\theta))^T \Big) \\
&= -\nabla_\theta \sum_{i=1}^{m} \frac{1}{\left(1 - \exp\left(\theta^T x^{(i)}\right)\right)} x^{(i)T} \\
&= -\sum_{i=1}^{m} \frac{\exp\left(\theta^T x^{(i)}\right)}{\left(1 - \exp\left(\theta^T x^{(i)}\right)\right)^2} x^{(i)} x^{(i)T}
\end{aligned}$$

The Newton's method update rule is then: $\theta := \theta - H^{-1} \nabla_\theta l(\theta)$

(c) [2 points] Show that the Hessian is negative semi-definite. This shows the optimization objective is concave, and hence Newton's method is maximizing log-likelihood.

Answer:

$$\begin{aligned} z^T H z &= -\sum_{i=1}^{m} \frac{\exp\left(\theta^T x^{(i)}\right)}{\left(1-\exp\left(\theta^T x^{(i)}\right)\right)^2} z^T x^{(i)} x^{(i)T} z \\ &= -\sum_{i=1}^{m} \frac{\exp\left(\theta^T x^{(i)}\right)}{\left(1-\exp\left(\theta^T x^{(i)}\right)\right)^2} \|z^T x^{(i)}\|^2 \leq 0 \end{aligned}$$

Therefore, H is negative semidefinite, which means $l(\theta)$ is concave. So we are maximizing it.

4. **[16 points] Support Vector Regression**

In class, we showed how the SVM can be used for classification. In this problem, we will develop a modified algorithm, called the Support Vector Regression algorithm, which can instead be used for regression, with continuous valued labels $y \in \mathbb{R}$.

Suppose we are given a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$, where $x^{(i)} \in \mathbb{R}^{(n+1)}$ and $y^{(i)} \in \mathbb{R}$. We would like to find a hypothesis of the form $h_{w,b}(x) = w^T x + b$ with a small value of w. Our (convex) optimization problem is:

$$\begin{array}{lll} \min_{w,b} & \frac{1}{2}\|w\|^2 & \\ \text{s.t.} & y^{(i)} - w^T x^{(i)} - b \leq \epsilon \qquad & i = 1, \ldots, m \quad (1) \\ & w^T x^{(i)} + b - y^{(i)} \leq \epsilon \qquad & i = 1, \ldots, m \quad (2) \end{array}$$

where $\epsilon > 0$ is a given, fixed value. Notice how the original functional margin constraint has been modified to now represent the distance between the continuous y and our hypothesis' output.

(a) [3 points] Write down the Lagrangian for the optimization problem above. We suggest you use two sets of Lagrange multipliers α_i and α_i^*, corresponding to the two inequality constraints (labeled (1) and (2) above), so that the Lagrangian would be written $\mathcal{L}(w, b, \alpha, \alpha^*)$.
Answer:

$$\begin{array}{lll} \min_{w,b} & \frac{1}{2}\|w\|^2 & \\ \text{s.t.} & y^{(i)} - w^T x^{(i)} - b \leq \epsilon \qquad & i = 1, \ldots, m \quad (1) \\ & w^T x^{(i)} + b - y^{(i)} \leq \epsilon \qquad & i = 1, \ldots, m \quad (2) \end{array}$$

Let $\alpha_i, \alpha_i^* \geq 0$ $(i = 1, \ldots, m)$ be the Lagrange multiplier for (1)-(4) respectively. Then, the Lagrangian can be written as:

$$\begin{aligned} & L(w, b, \alpha, \alpha^*) \\ & = \tfrac{1}{2}\|w\|^2 \\ & - \sum_{i=1}^{m} \alpha_i (\epsilon - y^{(i)} + w^T x^{(i)} + b) \\ & - \sum_{i=1}^{m} \alpha_i^* (\epsilon + y^{(i)} - w^T x^{(i)} - b) \end{aligned}$$

(b) [9 points] Derive the dual optimization problem. You will have to take derivatives of the Lagrangian with respect to w and b.

Answer:

First, the dual objective function can be written as:

$$\theta_D(\alpha, \alpha^*) = \min_{w,b} L(w, b, \alpha, \alpha^*)$$

Now, taking the derivatives of Lagrangian with respect to all primal variables, we have:

$$\partial_w L = w - \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)x^{(i)} = 0$$
$$\partial_b L = \sum_{i=1}^{m}(\alpha_i^* - \alpha_i) = 0$$

Substituting the above four relations back into the Lagrangian, we have:

$$\begin{aligned}
\theta_D(\alpha, \alpha^*) &= \tfrac{1}{2}\|w\|^2 - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) \\
&\quad + b\sum_{i=1}^{m}(\alpha_i^* - \alpha_i) + \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)w^T x^{(i)} \\
\theta_D(\alpha, \alpha^*) &= \tfrac{1}{2}\|w\|^2 - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) + \sum_{i=1}^{m}(\alpha_i^* - \alpha_i)w^T x^{(i)} \\
&= \tfrac{1}{2}\|\sum_{i=1}^{m}(\alpha_i - \alpha_i^*)x^{(i)}\|^2 - \sum_{i=1}^{m}(\alpha_i - \alpha_i^*)\Big(\sum_{j=1}^{m}(\alpha_j - \alpha_j^*)x^{(j)T}x^{(i)}\Big) \\
&\quad - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) \\
&= -\tfrac{1}{2}\sum_{i=1,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)x^{(i)T}x^{(j)} - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*)
\end{aligned}$$

Now the dual problem can be formulated as:

$$\begin{aligned}
\max_{\alpha_i, \alpha_i^*} \quad & -\tfrac{1}{2}\sum_{i=1,j=1}^{m}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)x^{(i)T}x^{(j)} - \epsilon\sum_{i=1}^{m}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{m} y^{(i)}(\alpha_i - \alpha_i^*) \\
\text{s.t.} \quad & \sum_{i=1}^{m}(\alpha_i^* - \alpha_i) = 0 \\
& \alpha_i, \alpha_i^* \geq 0
\end{aligned}$$

[more space for problem 4 (b)]

(c) [4 points] Show that this algorithm can be kernelized. For this, you have to show that (i) the dual optimization objective can be written in terms of inner-products of training examples; and (ii) at test time, given a new x the hypothesis $h_{w,b}(x)$ can also be computed in terms of inner products.

Answer:

This algorithm can be kernelized because when making prediction at x, we have:

$$f(w,x) = w^T x + b = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) x^{(i)T} x + b = \sum_{i=1}^{m} (\alpha_i - \alpha_i^*) k(x^{(i)}, x) + b$$

This shows that predicting function can be written in a kernel form.

5. **[20 points] Learning Theory**

Suppose you are given a hypothesis $h_0 \in \mathcal{H}$, and your goal is to determine whether h_0 has generalization error within $\eta > 0$ of the best hypothesis, $h^* = \arg\min_{h \in \mathcal{H}} \varepsilon(h)$. More specifically, we say that a hypothesis h is η**-optimal** if $\varepsilon(h) \leq \varepsilon(h^*) + \eta$. Here, we wish to answer the following question:

Given a hypothesis h_0, is h_0 η-optimal?

Let $\delta > 0$ be some fixed constant, and consider a finite hypothesis class $\mathcal{H}$ of size $|\mathcal{H}| = k$. For each $h \in \mathcal{H}$, let $\hat{\varepsilon}(h)$ denote the training error of h with respect to some training set of m IID examples, and let $\hat{h} = \arg\min_{h \in \mathcal{H}} \hat{\varepsilon}(h)$ denote the hypothesis that minimizes training error.

Now, consider the following algorithm:

1. Set
$$\gamma := \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}.$$
2. If $\hat{\varepsilon}(h_0) > \hat{\varepsilon}(\hat{h}) + \eta + 2\gamma$, then return NO.
3. If $\hat{\varepsilon}(h_0) < \hat{\varepsilon}(\hat{h}) + \eta - 2\gamma$, then return YES.
4. Otherwise, return UNSURE.

Intuitively, the algorithm works by comparing the training error of h_0 to the training error of the hypothesis $\hat{h}$ with the minimum training error, and returns NO or YES only when $\hat{\varepsilon}(h_0)$ is either significantly larger than or significantly smaller than $\hat{\varepsilon}(\hat{h})+\eta$.

(a) [6 points] First, show that if $\varepsilon(h_0) \leq \varepsilon(h^*) + \eta$ (i.e., h_0 is η-optimal), then the probability that the algorithm returns NO is at most δ.

[extra space for 5 (a)]

Answer:

Suppose that $\varepsilon(h_0) \leq \varepsilon(h^*) + \eta$. Using the Hoeffding inequality, we have that for

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

then with probability at least $1 - \delta$,

$$\begin{aligned}
\hat{\varepsilon}(h_0) &\leq \varepsilon(h_0) + \gamma \\
&\leq \varepsilon(h^*) + \eta + \gamma \\
&\leq \varepsilon(\hat{h}) + \eta + \gamma \\
&\leq \hat{\varepsilon}(\hat{h}) + \eta + 2\gamma.
\end{aligned}$$

Here, the first and last inequalities follow from the fact that under the stated uniform convergence conditions, all hypotheses in $\mathcal{H}$ have empirical errors within γ of their true generalization errors. The second inequality follows from our assumption, and the third inequality follows from the fact that h^* minimizes the true generalization error. Therefore, the reverse condition, $\hat{\varepsilon}(h_0) > \hat{\varepsilon}(\hat{h}) + \eta + 2\gamma$, occurs with probability at most δ.

(b) [6 points] Second, show that if $\varepsilon(h_0) > \varepsilon(h^*) + \eta$ (i.e., h_0 is not η-optimal), then the probability that the algorithm returns YES is at most δ.

Answer:
Suppose that $\varepsilon(h_0) > \varepsilon(h^*) + \eta$. Using the Hoeffding inequality, we have that for

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

then with probability at least $1 - \delta$,

$$\begin{aligned}
\hat{\varepsilon}(h_0) &\geq \varepsilon(h_0) - \gamma \\
&> \varepsilon(h^*) + \eta - \gamma \\
&\geq \hat{\varepsilon}(h^*) + \eta - 2\gamma \\
&\geq \hat{\varepsilon}(\hat{h}) + \eta - 2\gamma.
\end{aligned}$$

Here, the first and third inequalities follow from the fact that under the stated uniform convergence conditions, all hypotheses in $\mathcal{H}$ have empirical errors within γ of their true generalization errors. The second inequality follows from our assumption, and the last inequality follows from the fact that $\hat{h}$ minimizes the empirical error. Therefore, the reverse condition, $\hat{\varepsilon}(h_0) < \hat{\varepsilon}(\hat{h}) + \eta - 2\gamma$ occurs with probability at most δ.

(c) [8 points] Finally, suppose that $h_0 = h^*$, and let $\eta > 0$ and $\delta > 0$ be fixed. Show that if m is sufficiently large, then the probability that the algorithm returns YES is at least $1 - \delta$.

Hint: observe that for fixed η and δ, as $m \to \infty$, we have

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}} \to 0.$$

This means that there are values of m for which $2\gamma < \eta - 2\gamma$.

Answer:

Suppose that $h_0 = h^*$. Using the Hoeffding inequality, we have that for

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

then with probability at least $1 - \delta$,

$$\begin{aligned} \hat{\varepsilon}(h_0) &\leq \varepsilon(h_0) + \gamma \\ &= \varepsilon(h^*) + \gamma \\ &\leq \varepsilon(\hat{h}) + \gamma \\ &\leq \hat{\varepsilon}(\hat{h}) + 2\gamma. \end{aligned}$$

Here, the first and last inequalities follow from the fact that under the stated uniform convergence conditions, all hypotheses in $\mathcal{H}$ have empirical errors within γ of their true generalization errors. The equality in the second step follows from our assumption, and the inequality in the third step follows from the fact that h^* minimizes the true generalization error. But, observe that for fixed η and δ, as $m \to \infty$, we have

$$\gamma = \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}} \to 0.$$

This implies that for m sufficiently large, $4\gamma < \eta$, or equivalently, $2\gamma < \eta - 2\gamma$. It follows that with probability at least $1 - \delta$, if m is sufficiently large, then

$$\hat{\varepsilon}(h_0) \leq \hat{\varepsilon}(\hat{h}) + \eta - 2\gamma,$$

so the algorithm returns YES.

6. **[24 points] Short answers**

   The following questions require a reasonably short answer (usually at most 2-3 sentences or a figure, though some questions may require longer or shorter explanations).

   To discourage random guessing, one point will be deducted for a wrong answer on true/false or multiple choice questions! Also, no credit will be given for answers without a correct explanation.

   (a) [3 points] You have an implementation of Newton's method and gradient descent. Suppose that one iteration of Newton's method takes twice as long as one iteration of gradient descent. Then, this implies that gradient descent will converge to the optimal objective faster. True/False?

   Answer: False. Newton's method may take fewer steps.

   (b) [3 points] A stochastic gradient descent algorithm for training logistic regression with a fixed learning rate will always converge to exactly the optimal setting of the parameters $\theta^* = \arg\max_\theta \prod_{i=1}^m p(y^{(i)}|x^{(i)};\theta)$, assuming a reasonable choice of the learning rate. True/False?

   Answer: False. A fixed learning rate means that we are always taking a finite step towards improving the log-probability of any single training example in the update equation. Unless the examples are somehow aligned, we will continue jumping from side to side of the optimal solution, and will not be able to get arbitrarily close to it. The learning rate has to approach to zero in the course of the updates for the weights to converge robustly.

(c) [3 points] Given a valid kernel $K(x, y)$ over $\mathbb{R}^m$, is $K_{norm}(x, y) = \frac{K(x,y)}{\sqrt{K(x,x)K(y,y)}}$ a valid kernel?

Answer: Yes. If we write $K(x, y) = \Phi(x)^T\Phi(y)$, then we have:

$$K_{norm}(x, y) = \frac{\Phi(x)^T\Phi(y)}{\sqrt{(\Phi(x)^T\Phi(x))(\Phi(y)^T\Phi(y))}} = \Psi(x)^T\Psi(y)$$

with

$$\Psi(x) = \frac{\Phi(x)}{\sqrt{\Phi(x)^T\Phi(x)}}$$

So K_{norm} (as normalized) is a valid kernel.

(d) [3 points] Consider a 2 class classification problem with a dataset of inputs $\{x^{(1)} = (-1, -1), x^{(2)} = (-1, +1), x^{(3)} = (+1, -1), x^{(4)} = (+1, +1)\}$. Can a linear SVM (with no kernel trick) shatter this set of 4 points?

Answer: No we cannot, since the decision boundary is linear. Let the labels be represented by y. See that we cannot classify in the case $y^{(1)} = +1, y^{(2)} = y^{(3)} = -1, y^{(4)} = +1$.

(e) [3 points] For linear hypotheses (i.e. of the form $h(x) = w^T x + b$), the vector of learned weights w is always perpendicular to the separating hyperplane. True/False? Provide a counterexample if False, or a brief explanation if True.

Answer: True. For a linear separating boundary, the hyperplane is defined by the set $\{x|w^T x = -b\}$. The inner product $w^T x$ geometrically represents the projection of x onto w. The set of all points whose projection onto w is constant ($-b$) forms a line that must be perpendicular to w. So h is perpendicular to w. This fact is necessary in the formulation of geometric margins for the linear SVM.

(f) [3 points] Let $\mathcal{H}$ be a set of classifiers whose VC-dimension is 5. Suppose we have four training examples and labels, $\{(x^{(1)}, y^{(1)}), ..., (x^{(4)}, y^{(4)})\}$, and we select a classifier h from $\mathcal{H}$ by minimizing the classification error on the training set. In the absence of any other information about the set of classifiers $\mathcal{H}$, can we say that

i. $x^{(5)}$ will certainly be classified correctly?
ii. $x^{(5)}$ will certainly be classified incorrectly?
iii. we cannot tell?

Briefly justify your answer.

Answer: We cannot tell. Since the VC-dimension is 5, $\mathcal{F}$ can shatter (some) five points. These points could be $x^{(1)}, \ldots, x^{(5)}$. Thus we can find $f_1 \in \mathcal{F}$ consistent with the four training examples and $f_1(x^{(5)}) = 1$, as well as another classifier $f_2 \in \mathcal{F}$ also consistent with the training examples for which $f_2(x^{(5)}) = -1$. The training set therefore does not constrain the prediction at $x^{(5)}$.

(g) [6 points] Suppose you would like to use a linear regression model in order to predict the price of houses. In your model, you use the features $x_0 = 1$, $x_1 =$ size in square meters, $x_2 =$ height of roof in meters. Now, suppose a friend repeats the same analysis using exactly the same training set, only he represents the data instead using features $x'_0 = 1$, $x'_1 = x_1$, and $x'_2 =$ height in cm (so $x'_2 = 100x_2$).

i. [3 points] Suppose both of you run linear regression, solving for the parameters via the Normal equations. (Assume there are no degeneracies, so this gives a unique solution to the parameters.) You get parameters θ_0, θ_1, θ_2; your friend gets θ'_0, θ'_1, θ'_2. Then $\theta'_0 = \theta_0$, $\theta'_1 = \theta_1$, $\theta'_2 = \frac{1}{100}\theta_2$. True/False?

Answer: True. Observe that running a single step of Newton's method, for a linear regression problem, is equivalent to solving the Normal equations. The result then follows from the invariance of Newton's method to linear reparameterizations.

ii. [3 points] Suppose both of you run linear regression, initializing the parameters to 0, and compare your results after running just *one* iteration of batch gradient descent. You get parameters θ_0, θ_1, θ_2; your friend gets θ'_0, θ'_1, θ'_2. Then $\theta'_0 = \theta_0$, $\theta'_1 = \theta_1$, $\theta'_2 = \frac{1}{100}\theta_2$. True/False?

Answer: False. Recall that gradient descent is not invariant to linear reparameterizations.

STANFORD UNIVERSITY
CS 229, Autumn 2015
Midterm Examination

Wednesday, November 4, 6:00pm-9:00pm

Question	Points
1 Short Answers	/26
2 More Linear Regression	/10
3 Generalized Linear Models	/17
4 Naive Bayes and Logistic Regression	/17
5 Anomaly Detection	/15
6 Learning Theory	/15
Total	/100

Name of Student: _____

SUNetID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signed: _____

1. **[26 points] Short answers**

   The following questions require a reasonably short answer (usually at most 2-3 sentences or a figure, though some questions may require longer or shorter explanations).

   To discourage random guessing, one point will be deducted for a wrong answer on true/false or multiple choice questions! Also, no credit will be given for answers without a correct explanation.

   (a) [6 points] Suppose you are fitting a fixed dataset with m training examples using linear regression, $h_\theta(x) = \theta^T x$, where $\theta, x \in \mathbb{R}^{n+1}$. After training, you realize that the variance of your model is relatively high (i.e. you are overfitting). For the following methods, indicate true if the method can mitigate your overfitting problem and false otherwise. Briefly explain why.

   i. [3 points] Add additional features to your feature vector.

   ii. [3 points] Impose a prior distribution on θ, where the distribution of θ is of the form $\mathcal{N}(0, \tau^2 I)$, and we derive θ via maximum a posteriori estimation.

(b) [3 points] Choosing the parameter C is often a challenge when using SVMs. Suppose we choose C as follows: First, train a model for a wide range of values of C. Then, evaluate each model on the test set. Choose the C whose model has the best performance on the test set. Is the performance of the chosen model on the test set a good estimate of the model's generalization error?

(c) [11 points] For the following, provide the VC-dimension of the described hypothesis classes and briefly explain your answer.

i. [3 points] Assume $\mathcal{X} = \mathbb{R}^2$. $\mathcal{H}$ is a hypothesis class containing a single hypothesis h_1 (i.e. $\mathcal{H} = \{h_1\}$)

ii. [4 points] Assume $\mathcal{X} = \mathbb{R}^2$. Consider $\mathcal{A}$ to be the set of all convex polygons in $\mathcal{X}$. $\mathcal{H}$ is the class of all hypotheses $h_P(x)$ (for $P \in \mathcal{A}$) such that

$$h_P(x) = \begin{cases} 1 & \text{if } x \text{ is contained within polygon P} \\ 0 & \text{otherwise} \end{cases}$$

Hint: Points on the edges or vertices of P are included in P

iii. [4 points] $\mathcal{H}$ is the class of hypotheses $h_{(a,b)}(x)$ such that each hypothesis is represented by a single open interval in $\mathcal{X} = \mathbb{R}$ as follows:

$$h_{(a,b)}(x) = \begin{cases} 1 & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

(d) [3 points] Consider a sine function $f(x) = \sin(x)$ such that $x \in [-\pi, \pi]$. We use two different hypothesis classes such that $\mathcal{H}_0$ contains all constant hypotheses of the form, $h(x) = b$ and $\mathcal{H}_1$ contains all linear hypotheses of the form $h(x) = ax + b$. Consider taking a very large number of training sets, $S_i, i = 1, ..., N$ such that each S_i contains only two points $\{(x_1, y_1), (x_2, y_2)\}$ sampled iid from $f(x)$. In other words, each (x, y) pair is drawn from a distribution such that $y = f(x) = \sin(x)$ is satisfied. We train a model from each hypothesis class using each training set such that we have a collection of N models from each class. We then compute a mean-squared error between each model and the function $f(x)$.

It turns out that the average expected error of all models from $\mathcal{H}_0$ is significantly lower than the average expected error of models from $\mathcal{H}_1$ even though $\mathcal{H}_1$ is a more complex hypothesis class. Using the concepts of bias and variance, provide an explanation for why this is the case.

(e) [3 points] In class when we discussed the decision boundary for logistic regression $h_\theta(x) = g(\theta^T x)$, we did not require an explicit intercept term because we could define $x_0 = 1$ and let θ_0 be the intercept. When discussing SVMs, we dropped this convention and had $h_{w,b}(x) = g(w^T x+b)$ with b as an explicit intercept term. Consider an SVM where we now write $h_w(x) = g(w^T x)$ and define $x_0 = 1$ such that w_0 is the intercept. If the primal optimization objective remains $\frac{1}{2}\,||w||^2$, can we change the intercept in this way without changing the decision boundary found by the SVM? Justify your answer.

2. **[10 + 3 Extra Credit points] More Linear Regression**
In our homework, we saw a variant of linear regression called locally-weighted linear regression. In the problem below, we consider a regularized form of locally-weighted linear regression where we favor smaller parameter vectors by adding a complexity penalty term to the cost function. Additionally, we consider the case where we are trying to predict multiple outputs for each training example. Our dataset is:

$$\mathcal{S} = \{(x^{(i)}, y^{(i)}), i = 1, ..., m\}, x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}^p$$

Thus for each training example, $y^{(i)}$ is a real-valued vector with p entries. We wish to use a linear model to predict the outputs by specifying the parameter matrix θ, where $\theta \in \mathbb{R}^{n \times p}$. You can assume $x^{(i)}$ contains the intercept term (i.e. $x_0 = 1$). The cost function for this model is:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} w^{(i)} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2 + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2 \tag{1}$$

As before, $w^{(i)}$ is the "weight" for a specific training example i.

(a) [2 points] Show that $J(\theta)$ can be written as

$$J(\theta) = \frac{1}{2} \text{tr} \left((X\theta - Y)^T W (X\theta - Y) \right) + \frac{1}{2} \text{tr}(\theta^T \theta)$$

(b) [5 points] Derive a closed form expression for the minimizer θ^* that minimizes $J(\theta)$ from part (a).

(c) [3 points] Given the dataset $\mathcal{S}$ above, which of the following cost functions will lead to higher accuracy on the training set? Briefly explain why this is the case. If there is insufficient information, explain what details are needed to make a decision.

i. $J_1(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2$

ii. $J_2(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2 + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2$

iii. $J_3(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2 + 100 \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2$

(d) [3 Extra Credit points] Suppose we want to weight the regularization penalty on a per element basis. For this problem, we use the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p} w^{(i)}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}((\Gamma\theta)_{ij})^2 \tag{2}$$

Here, $\Gamma \in \mathbb{R}^{n\times n}$ where $\Gamma_{ij} > 0$ for all i, j. Derive a closed form solution for $J(\theta)$ and θ^* using this new cost function.

3. **[17 points] Generalized Linear Models**

   In class we showed that the Gaussian distribution is in the Exponential Family. However, a simplification we made to make the derivation easier was to set the variance term $\sigma^2 = 1$. This problem will investigate a more general form for the Exponential Family. First, recall that the Gaussian distribution can be written as follows:

$$p(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\} \tag{3}$$

   (a) [6 points] Show that the Gaussian distribution (without assuming unit variance) is an exponential family distribution. In particular, please specify $b(y)$, η, $T(y)$, $a(\eta)$. Recall that the standard form for the exponential family is given by

$$p(y;\eta) = b(y)\exp\{\eta^\top T(y) - a(\eta)\} \tag{4}$$

   Hint: since σ^2 is now a variable, η and $T(y)$ will now be two dimensional vectors; for consistent notation denote $\eta = \begin{bmatrix}\eta_1 & \eta_2\end{bmatrix}^\top$. For full credit, please ensure $a(\eta)$ is expressed in terms of η_1 and η_2.

(b) [4 points] Suppose you are given an IID training set $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$. Starting with the expression in (4) for $p(y; \eta)$, derive the general expression for the Hessian of the log-likelihood $\ell(\theta) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}; \theta)$. Your answer should be in terms of x, η_1 and η_2.

(c) [5 points] Using your result from the part (b), show that the Hessian is negative semi-definite, i.e., $z^\top Hz \leq 0$.

(d) [2 points] It turns out there is a more general definition for the exponential family given by

$$p(y;\eta,\tau) = b(a,\tau)\exp\left\{\frac{\eta^\top T(y) - a(\eta)}{c(\tau)}\right\}$$

In particular $c(\tau)$ is the dispersion function, where τ is called the *dispersion parameter*. Show that the Gaussian distribution can be written in this more general form with $c(\tau) = \sigma^2$.

4. **[17 points] Naive Bayes and Logistic Regression**
   For this entire problem assume that the input features x_j, $j = 1, ..., n$ are discrete binary-valued variables such that $x_j \in \{0, 1\}$ and $x = [x_1\ x_2\ ...\ x_n]$. For each training example $x^{(i)}$, assume that the output target variable $y^{(i)} \in \{0, 1\}$.

   (a) [2 points] Consider the Naive Bayes model, given the above context. This model can be parameterized by $\phi_{j|y=0} = p(x_j = 1|y = 0)$, $\phi_{j|y=1} = p(x_j = 1|y = 1)$ and $\phi_y = p(y = 1)$. Write down the expression for $p(y = 1|x)$ in terms of $\phi_{j|y=0}, \phi_{j|y=1}$, and ϕ_y.

(b) [7 points] Show that the conditional likelihood expression you obtained in part (a) can be simplified to the same form as the hypothesis for logistic regression:

$$p(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}. \tag{5}$$

Hint: Modify the definition of x to include the intercept term $x_0 = 1$

[More space for (b)]

(c) [6 points] In part (b) you showed that the discrete Naive Bayes decision boundary has the same form as that of the logistic regression. Now consider a dataset S_1 with m training examples of the form: $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$ with each $x^{(i)} \in \mathbb{R}^{n+1}$. Note that for this problem, S_1 satisfies the Naive Bayes assumption: $p(x_1, \ldots, x_n|y) = \prod_{j=1}^{n} p(x_j|y)$.
Suppose a second dataset, S_2, is given to you again with m training examples $\{(x^{(i)}, y^{(i)}), i = 1, \cdots, m\}$, but now each $x^{(i)} \in \mathbb{R}^{n+2}$ because each $x^{(i)}$ contains the same n conditionally-independent features and an additional feature x_{n+1} such that $x_{n+1} = x_n$. Each $x^{(i)}$ contains the intercept term $x_0 = 1$.

i. [2 points] You train two Naive Bayes classifiers independently on S_1 and S_2. Test data is generated according to the true distribution (i.e. $p(x_1, ..., x_n, y) = p(x_1, ..., x_n, x_{n+1}, y) = p(y)p(x_1, ..., x_n|y)$, where $x_{n+1} = x_n$). Would you expect the test error of the classifier trained on S_1 to be larger or smaller than that trained on S_2? You may assume that m is very large. Briefly justify your answer.

ii. [4 points] Now we will look at a similar situation regarding how logistic regression is affected by copies of features. In order to simplify the math, let's assume a more basic case where S_1 still has m training examples, but now has one feature x_1. S_2 has m training examples but has two features x_1 and x_2 where $x_2 = x_1$. The logistic regression model trained on S_1 therefore has associated parameters $\{\theta_0, \theta_1\}$ and the model trained on S_2 has parameters $\{\theta_0, \theta_1, \theta_2\}$. Here, θ_0 is associated with the intercept term $x_0 = 1$. Testing data is generated the same way (from the original true distribution). How will the error of the classifier trained on S_1 compare to that of the classifier trained on S_2? For this question you need to prove your result mathematically. (Hint: compare the forms of the log-likelihood for each classifier)

(d) [2 points] In general, if we assume that the number of training examples m is very large, which classifier will have a lower generalization error? Briefly justify why.

5. **[15 points] Anomaly Detection**

   Consider the following optimization problem:

$$\begin{aligned} \underset{r,z,\xi}{\text{minimize}} \quad & r^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & \left|\left|x^{(i)} - z\right|\right|_2^2 \leq r^2 + \xi_i \;\; i = 1,\ldots,m. \\ & \xi_i \geq 0, \;\; i = 1,\ldots,m. \end{aligned} \tag{6}$$

   where ξ_i are the slack variables.

   (a) [2 points] Write down the Lagrangian for the optimization problem above. We suggest using two sets of Lagrange multipliers α_i and η_i corresponding to the two inequality constraints so that the Lagrangian would be written as $\mathcal{L}(r, z, \xi, \alpha, \eta)$.

(b) [7 points] Assuming a non-trivial solution ($r > 0$), derive the dual optimization problem using the Lagrangian from part (a).

(c) [3 points] Show that the dual problem from (b) can be kernelized.

(d) [3 points] Now consider the following dual optimization problem

$$\begin{aligned} \max_{\alpha} \quad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t} \quad & \sum_{i=1}^{m}\alpha_i = 1, \quad i = 1, \ldots, m. \end{aligned} \tag{7}$$

Assume that we choose K such that it is a Gaussian Kernel. How does this dual compare with the dual you derived in part (c)?.

6. **[15 points] Learning Theory**

   Consider a finite hypothesis class $\mathcal{H}$ with size $k = |\mathcal{H}|$ and $h^\star = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ \epsilon(h)$.

   (a) [7 points] Assume that the best hypothesis $h^\star$ has generalization error $\epsilon(h^\star) = B$ such that B is a constant with $0 \leq B \leq 1$. Prove that the joint probability of the expected risk minimizer $\hat{h}$ having large generalization error and the best hypothesis h^* having small training error can be bounded as:

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq \sum_{h \in \mathcal{H}} P(\epsilon(h) > B + 2\gamma, \hat{\epsilon}(h) \leq B + \gamma) \quad (8)$$

For any hypothesis $h' \in \mathcal{H}$ with high generalization error (i.e. $\epsilon(h') > B' + \tau$), the probability that it has low training error (i.e. $\hat{\epsilon}(h') \leq B'$) is bounded by:

$$P(\hat{\epsilon}(h') \leq B' \mid \epsilon(h') > B' + \tau) \leq \exp\left\{\frac{-m\tau^2}{2(B' + 4\tau/3)}\right\} \tag{9}$$

for any $B' \in (0, 1)$ and $\tau > 0$.

(b) [8 points] Using (9) and the result from part (a), show that:

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq k \exp\left\{\frac{-m\gamma^2}{2(B + 7\gamma/3)}\right\}. \tag{10}$$

STANFORD UNIVERSITY
CS 229, Autumn 2015
Midterm Examination

Wednesday, November 4, 6:00pm-9:00pm

Question	Points
1 Short Answers	/26
2 More Linear Regression	/10
3 Generalized Linear Models	/17
4 Naive Bayes and Logistic Regression	/17
5 Anomaly Detection	/15
6 Learning Theory	/15
Total	/100

Name of Student: _____

SUNetID: _____@stanford.edu

The Stanford University Honor Code:

I attest that I have not given or received aid in this examination, and that I have done my share and taken an active part in seeing to it that others as well as myself uphold the spirit and letter of the Honor Code.

Signed: _____

1. **[26 points] Short answers**

   The following questions require a reasonably short answer (usually at most 2-3 sentences or a figure, though some questions may require longer or shorter explanations).

   To discourage random guessing, one point will be deducted for a wrong answer on true/false or multiple choice questions! Also, no credit will be given for answers without a correct explanation.

   (a) [6 points] Suppose you are fitting a fixed dataset with m training examples using linear regression, $h_\theta(x) = \theta^T x$, where $\theta, x \in \mathbb{R}^{n+1}$. After training, you realize that the variance of your model is relatively high (i.e. you are overfitting). For the following methods, indicate true if the method can mitigate your overfitting problem and false otherwise. Briefly explain why.

   i. [3 points] Add additional features to your feature vector.
   Answer: False. More features will make our model more complex, which will capture more outliers in the training set and overfit more.

   ii. [3 points] Impose a prior distribution on θ, where the distribution of θ is of the form $\mathcal{N}(0, \tau^2 I)$, and we derive θ via maximum a posteriori estimation.
   Answer: True. By imposing a prior belief on the distribution of θ, we are effectively limiting the norm of θ, since larger norm will have a lower probability. Thus, it makes our model less susceptible to overfitting.

(b) [3 points] Choosing the parameter C is often a challenge when using SVMs. Suppose we choose C as follows: First, train a model for a wide range of values of C. Then, evaluate each model on the test set. Choose the C whose model has the best performance on the test set. Is the performance of the chosen model on the test set a good estimate of the model's generalization error?
Answer: No it is not because C will be selected using the test set, meaning that the test set is no longer separate from model development. As a result, the choice of C might be over-fit to the test set and therefore might not generalize well on a new example, but there will be no way to figure this out because the test set was used to choose C.

(c) [11 points] For the following, provide the VC-dimension of the described hypothesis classes and briefly explain your answer.

i. [3 points] Assume $\mathcal{X} = \mathbb{R}^2$. $\mathcal{H}$ is a hypothesis class containing a single hypothesis h_1 (i.e. $\mathcal{H} = \{h_1\}$)
Answer: $VC(\mathcal{H}) = 0$. The VC dimension of a single hypothesis is always zero because a single hypothesis can only assign one labeling to a set of points.

ii. [4 points] Assume $\mathcal{X} = \mathbb{R}^2$. Consider $\mathcal{A}$ to be the set of all convex polygons in $\mathcal{X}$. $\mathcal{H}$ is the class of all hypotheses $h_P(x)$ (for $P \in \mathcal{A}$) such that

$$h_P(x) = \begin{cases} 1 & \text{if } x \text{ is contained within polygon P} \\ 0 & \text{otherwise} \end{cases}$$

Hint: Points on the edges or vertices of P are included in P

Answer: $VC(\mathcal{H}) = \infty$. For any positive integer n, take n points from $\mathcal{A}$. Suppose we place the n points $\{x_1, x_2, ..., x_n\}$ uniformly spaced on the unit circle. Then for each of the 2^n subsets of this data set, there is a convex polygon with vertices at these n points. For each subset, the convex polygon contains the set and excludes its complement. Therefore, $\forall n$, the shattering coefficient is 2^n and thus the VC dimension is infinite.

iii. [4 points] $\mathcal{H}$ is the class of hypotheses $h_{(a,b)}(x)$ such that each hypothesis is represented by a single open interval in $\mathcal{X} = \mathbb{R}$ as follows:

$$h_{(a,b)}(x) = \begin{cases} 1 & \text{if } a < x < b \\ 0 & \text{otherwise} \end{cases}$$

Answer: $VC(\mathcal{H}) = 2$. Take for example two points $\{0, 2\}$. We can shatter these two points by choosing the following set of intervals for our hypotheses $\{(3, 5), (-1, 1), (1, 3), (-1, 3)\}$. These correspond to the labellings: $\{(0, 0), (1, 0), (0, 1), (1, 1)\}$. We cannot shatter any set of three points $\{x_1, x_2, x_3\}$ such that $x_1 < x_2 < x_3$ because the labelling $x_1 = x_3 = 1, x_2 = 0$ cannot be realized. More generally, alternate labellings of consecutive points cannot be realized.

(d) [3 points] Consider a sine function $f(x) = \sin(x)$ such that $x \in [-\pi, \pi]$. We use two different hypothesis classes such that $\mathcal{H}_0$ contains all constant hypotheses of the form, $h(x) = b$ and $\mathcal{H}_1$ contains all linear hypotheses of the form $h(x) = ax + b$. Consider taking a very large number of training sets, $S_i, i = 1, ..., N$ such that each S_i contains only two points $\{(x_1, y_1), (x_2, y_2)\}$ sampled iid from $f(x)$. In other words, each (x, y) pair is drawn from a distribution such that $y = f(x) = \sin(x)$ is satisfied. We train a model from each hypothesis class using each training set such that we have a collection of N models from each class. We then compute a mean-squared error between each model and the function $f(x)$.

It turns out that the average expected error of all models from $\mathcal{H}_0$ is significantly lower than the average expected error of models from $\mathcal{H}_1$ even though $\mathcal{H}_1$ is a more complex hypothesis class. Using the concepts of bias and variance, provide an explanation for why this is the case.

Answer: Consider what happens when we plot all of the possible hypotheses on top of the function $f(x)$. This can be seen in Figure 1. We can see that because our training set only consists of two points, the variance in linear hypotheses is far greater than that of the constant hypotheses. Even though the constant hypotheses have higher bias, the overall average expected error is less than for the linear hypotheses because of the huge difference in variance.

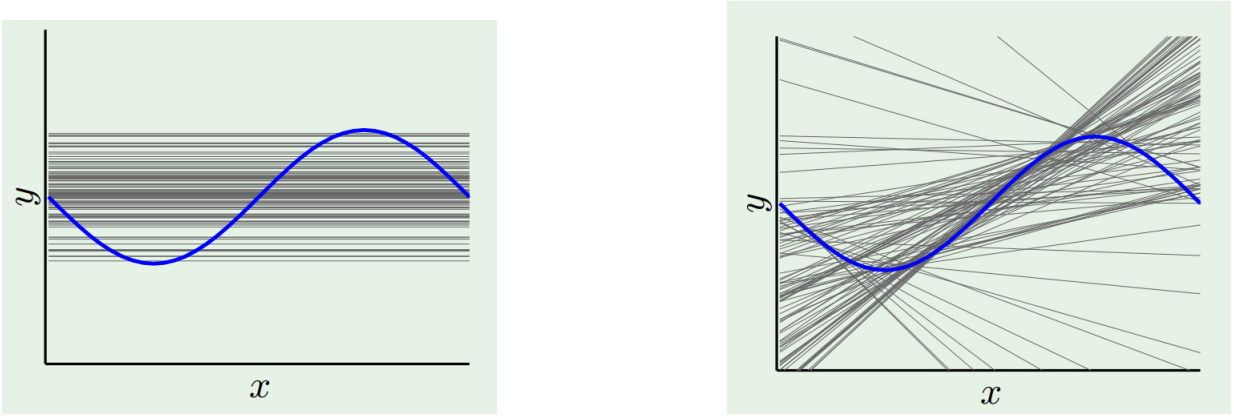


Figure 1: Many hypotheses from $\mathcal{H}_0$ and $\mathcal{H}_1$ plotted on top of $f(x)$

(e) [3 points] In class when we discussed the decision boundary for logistic regression $h_\theta(x) = g(\theta^T x)$, we did not require an explicit intercept term because we could define $x_0 = 1$ and let θ_0 be the intercept. When discussing SVMs, we dropped this convention and had $h_{w,b}(x) = g(w^T x + b)$ with b as an explicit intercept term. Consider an SVM where we now write $h_w(x) = g(w^T x)$ and define $x_0 = 1$ such that w_0 is the intercept. If the primal optimization objective remains $\frac{1}{2}||w||^2$, can we change the intercept in this way without changing the decision boundary found by the SVM? Justify your answer.

Answer: We cannot make this change, because it will change the decision boundary. In the original SVM, the $\frac{1}{2}||w||^2$ term did not penalize the intercept, so b could be chosen as large as possible to satisfy the constraints. If we treat, w_0 as the intercept, it will be regularized, thus changing the optimal solution.

2. **[10 + 3 Extra Credit points] More Linear Regression**
In our homework, we saw a variant of linear regression called locally-weighted linear regression. In the problem below, we consider a regularized form of locally-weighted linear regression where we favor smaller parameter vectors by adding a complexity penalty term to the cost function. Additionally, we consider the case where we are trying to predict multiple outputs for each training example. Our dataset is:

$$\mathcal{S} = \{(x^{(i)}, y^{(i)}), i = 1, ..., m\}, x^{(i)} \in \mathbb{R}^n, y^{(i)} \in \mathbb{R}^p$$

Thus for each training example, $y^{(i)}$ is a real-valued vector with p entries. We wish to use a linear model to predict the outputs by specifying the parameter matrix θ, where $\theta \in \mathbb{R}^{n \times p}$. You can assume $x^{(i)}$ contains the intercept term (i.e. $x_0 = 1$). The cost function for this model is:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} w^{(i)} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2 + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2 \tag{1}$$

As before, $w^{(i)}$ is the "weight" for a specific training example i.

(a) [2 points] Show that $J(\theta)$ can be written as

$$J(\theta) = \frac{1}{2} \text{tr} \left((X\theta - Y)^T W (X\theta - Y) \right) + \frac{1}{2} \text{tr}(\theta^T \theta)$$

Answer:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} w^{(i)} \left((\theta^T x^{(i)})_j - y_j^{(i)} \right)^2 + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2 \tag{2}$$

$$= \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{p} w^{(i)} (X\theta - Y)_{ij}^2 + \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{p} (\theta_{ij})^2 \tag{3}$$

$$= \frac{1}{2} \sum_{i=1}^{m} w^{(i)} \left((X\theta - Y)^T (X\theta - Y) \right)_{ii} + \frac{1}{2} \sum_{i=1}^{n} (\theta^T \theta)_{ii} \tag{4}$$

$$= \frac{1}{2} \text{tr} \left((X\theta - Y)^T W (X\theta - Y) \right) + \frac{1}{2} \text{tr}(\theta^T \theta) \tag{5}$$

(b) [5 points] Derive a closed form expression for the minimizer θ^* that minimizes $J(\theta)$ from part (a).

Answer: We compute the gradient of the first and second term separately. We start with the first term, $J_1(\theta) = \frac{1}{2}\text{tr}\left((X\theta - Y)^T W(X\theta - Y)\right)$.

$$\begin{aligned}
\nabla_\theta J_1(\theta) &= \nabla_\theta \frac{1}{2}\text{tr}\left((X\theta - Y)^T W(X\theta - Y)\right) && (6)\\
&= \frac{1}{2}\nabla_\theta \text{tr}\Big(\theta^T X^T W X\theta - \theta^T X^T W Y - Y^T W X\theta - Y^T W Y\Big) && (7)\\
&= \frac{1}{2}\nabla_\theta \left[\text{tr}(\theta^T X^T W X\theta) - \text{tr}(\theta^T X^T W Y) - \text{tr}(Y^T W X\theta) - \text{tr}(Y^T W Y)\right] && (8)\\
&= \frac{1}{2}\nabla_\theta \left[\text{tr}(\theta^T X^T W X\theta) - 2\text{tr}(Y^T W X\theta) - \text{tr}(Y^T W Y)\right] && (9)\\
&= \frac{1}{2}(X^T W X\theta - 2X^T W Y + X^T W X\theta) && (10)\\
&= X^T W X\theta - X^T W Y && (11)
\end{aligned}$$

Now we find the gradient of the second term, $J_2(\theta) = \frac{1}{2}\text{tr}(\theta^T\theta)$:

$$\nabla_\theta J_2(\theta) = \nabla_\theta \frac{1}{2}\text{tr}(\theta^T\theta) = \frac{1}{2}\nabla_\theta \text{tr}(\theta^T\theta) = \frac{1}{2}(2\theta) = \theta \tag{12}$$

Combining the gradient of both terms gives us the final gradient:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta J_1(\theta) + \nabla_\theta J_2(\theta)\\
&= X^T W X\theta - X^T W Y + \theta
\end{aligned} \tag{13}$$

We can then set this equal to zero and find the optimal θ which optimizes $J(\theta)$.

$$\begin{aligned}
0 &= X^T W X\theta - X^T W Y + \theta\\
X^T W Y &= X^T W X\theta + \theta\\
X^T W Y &= (X^T W X + I)\theta\\
\theta^* &= (X^T W X + I)^{-1} X^T W Y
\end{aligned} \tag{14}$$

where I is the $n \times n$ identity matrix.

(c) [3 points] Given the dataset $\mathcal{S}$ above, which of the following cost functions will lead to higher accuracy on the training set? Briefly explain why this is the case. If there is insufficient information, explain what details are needed to make a decision.

i. $J_1(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2$

ii. $J_2(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}(\theta_{ij})^2$

iii. $J_3(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2 + 100\sum_{i=1}^{n}\sum_{j=1}^{p}(\theta_{ij})^2$

Answer: (i) The regularization terms prevents overfitting by penalizing high weights. Excluding the regularization terms will allow the model to overfit on the training set and achieve a higher training accuracy.

(d) [3 Extra Credit points] Suppose we want to weight the regularization penalty on a per element basis. For this problem, we use the following cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p} w^{(i)}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}((\Gamma\theta)_{ij})^2 \tag{15}$$

Here, $\Gamma \in \mathbb{R}^{n\times n}$ where $\Gamma_{ij} > 0$ for all i, j. Derive a closed form solution for $J(\theta)$ and θ^* using this new cost function.

Answer: We first write $J(\theta)$ in matrix-vector notation to make the gradient derivation easier.

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p} w^{(i)}\left((\theta^T x^{(i)})_j - y_j^{(i)}\right)^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}((\Gamma\theta)_{ij})^2 \tag{16}$$

$$= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{p} w^{(i)}(X\theta - Y)_{ij}^2 + \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{p}((\Gamma\theta)_{ij})^2 \tag{17}$$

$$= \frac{1}{2}\sum_{i=1}^{m} w^{(i)}\left((X\theta - Y)^T(X\theta - Y)\right)_{ii} + \frac{1}{2}\sum_{i=1}^{n}\left((\Gamma\theta)^T(\Gamma\theta)\right)_{ii} \tag{18}$$

$$= \frac{1}{2}\mathrm{tr}\left((X\theta - Y)^T W(X\theta - Y)\right) + \frac{1}{2}\mathrm{tr}\left((\Gamma\theta)^T(\Gamma\theta)\right) \tag{19}$$

The gradient of the first term is the same as part (a):

$$\nabla_\theta J_1(\theta) = \nabla_\theta \frac{1}{2}\mathrm{tr}\left((X\theta - Y)^T W(X\theta - Y)\right) \tag{20}$$

$$= X^T W X\theta - X^T W Y \tag{21}$$

Now we find the gradient of the second term, denoted as $J_2(\theta)$:

$$\nabla_\theta J_2(\theta) = \nabla_\theta \frac{1}{2}\mathrm{tr}\left((\Gamma\theta)^T(\Gamma\theta)\right) \tag{22}$$

$$= \frac{1}{2}\nabla_\theta \mathrm{tr}\left(\theta^T\Gamma^T\Gamma\theta\right) \tag{23}$$

$$= \frac{1}{2}\Big((\Gamma^T\Gamma)^T\theta + \Gamma^T\Gamma\theta\Big) \tag{24}$$

$$= \frac{1}{2}\Big(2\Gamma^T\Gamma\theta\Big) \tag{25}$$

$$= \Gamma^T\Gamma\theta \tag{26}$$

The jump from (23) to (24) can be made using Equation 5 from lecture notes 1, where $A^T = \theta, A = \theta^T, B = \Gamma^T\Gamma$ and C be the identity matrix. Combining the gradient of both terms gives us the final gradient:

$$\nabla_\theta J(\theta) = X^T W X\theta - X^T W Y + \Gamma^T\Gamma\theta \tag{27}$$

We can then set this equal to zero and find the optimal θ which optimizes $J(\theta)$.

$$\begin{aligned} 0 &= X^T W X\theta - X^T W Y + \Gamma^T \Gamma\theta \\ X^T W Y &= X^T W X\theta + \Gamma^T \Gamma\theta \\ X^T W Y &= (X^T W X + \Gamma^T \Gamma)\theta \\ \theta^* &= (X^T W X + \Gamma^T \Gamma)^{-1} X^T W Y \end{aligned} \tag{28}$$

3. **[17 points] Generalized Linear Models**

In class we showed that the Gaussian distribution is in the Exponential Family. However, a simplification we made to make the derivation easier was to set the variance term $\sigma^2 = 1$. This problem will investigate a more general form for the Exponential Family. First, recall that the Gaussian distribution can be written as follows:

$$p(y|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\} \tag{29}$$

(a) [6 points] Show that the Gaussian distribution (without assuming unit variance) is an exponential family distribution. In particular, please specify $b(y)$, η, $T(y)$, $a(\eta)$. Recall that the standard form for the exponential family is given by

$$p(y;\eta) = b(y)\exp\{\eta^\top T(y) - a(\eta)\} \tag{30}$$

Hint: since σ^2 is now a variable, η and $T(y)$ will now be two dimensional vectors; for consistent notation denote $\eta = \begin{bmatrix}\eta_1 & \eta_2\end{bmatrix}^\top$. For full credit, please ensure $a(\eta)$ is expressed in terms of η_1 and η_2.

Answer: We can rearrange the Gaussian distribution as follows:

$$\begin{aligned} p(y|\mu, \sigma^2) &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left\{-\frac{1}{2\sigma^2}(y-\mu)^2\right\} \\ &= \frac{1}{\sqrt{2\pi}}\exp\left\{\frac{\mu}{\sigma^2}y - \frac{1}{2\sigma^2}y^2 - \frac{1}{2\sigma^2}\mu^2 - \ln\,\sigma\right\} \end{aligned}$$

This is now in the exponential family form, and we can note that:

$$\begin{aligned} \eta &= \begin{bmatrix}\eta_1 \\ \eta_2\end{bmatrix} = \begin{bmatrix}\mu/\sigma^2 \\ -1/2\sigma^2\end{bmatrix} \\ T(y) &= \begin{bmatrix}y \\ y^2\end{bmatrix} \\ a(\eta) &= \frac{\mu^2}{2\sigma^2} + \ln\,\sigma = -\frac{\eta_1^2}{4\eta_2} - \frac{1}{2}\ln(-2\eta_2) \\ b(y) &= \frac{1}{\sqrt{2\pi}} \end{aligned}$$

(b) [4 points] Suppose you are given an IID training set $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$. Starting with the expression in (30) for $p(y; \eta)$, derive the general expression for the Hessian of the log-likelihood $\ell(\theta) = \sum_{i=1}^{m} \log p(y^{(i)}|x^{(i)}; \theta)$. Your answer should be in terms of x, η_1 and η_2.

Answer: The log-likelihood is given by

$$\begin{aligned}\ell(\theta) &= \sum_{i=1}^{m} \log\, p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^{m} \log(b(y)) + \eta^{(i)} T(y) - a(\eta^{(i)})\end{aligned}$$

We now take partials with respect to θ_i and θ_j as follows. Recall that our standard GLM assumption is that $\eta = \theta^\top x$.

$$\begin{aligned}\frac{\partial}{\partial \theta_j}\ell(\theta) &= \sum_{i=1}^{M} T(y)x_j^{(i)} - \frac{\partial}{\partial \eta} a(\eta^{(i)})x_j^{(i)} \\ \frac{\partial^2}{\partial \theta_j \partial \theta_k}\ell(\theta) &= \sum_{i=1}^{M} -\frac{\partial^2}{\partial \eta^2} a(\eta^{(i)})x_j^{(i)} x_k^{(j)} \\ &= H_{jk}\end{aligned}$$

Thus the Hessian is described by each element H_{jk} which is itself a 2x2 symmetric matrix obtained by expanding the partial derivative with respect to η as follows:

$$\begin{aligned}\frac{\partial^2}{\partial \eta_1^2} a(\eta^{(i)}) &= \frac{\partial}{\partial \eta_1}\left(-\frac{2\eta_1}{4\eta_2}\right) = -\frac{1}{2\eta_2} \\ \frac{\partial^2}{\partial \eta_2^2} a(\eta^{(i)}) &= \frac{\partial}{\partial \eta_2}\left(\frac{-2\eta_2 + \eta_1^2}{4\eta_2^2}\right) = \frac{\eta_2 - \eta_1^2}{2\eta_2^3} \\ \frac{\partial^2}{\partial \eta_1 \partial \eta_2} a(\eta^{(i)}) &= \frac{\partial}{\partial \eta_2}\left(-\frac{2\eta_1}{4\eta_2}\right) = \frac{\eta_1}{2\eta_2^2} \\ \frac{\partial^2}{\partial \eta_2 \partial \eta_1} a(\eta^{(i)}) &= \frac{\partial}{\partial \eta_1}\left(\frac{-2\eta_2 + \eta_1^2}{4\eta_2^2}\right) = \frac{\eta_1}{2\eta_2^2}\end{aligned}$$

(c) [5 points] Using your result from the part (b), show that the Hessian is negative semi-definite, i.e., $z^\top Hz \leq 0$.

Answer: 5 points given to everyone because difficulty was harder than originally anticipated.

(d) [2 points] It turns out there is a more general definition for the exponential family given by

$$p(y;\eta,\tau) = b(a,\tau)\exp\left\{\frac{\eta^\top T(y) - a(\eta)}{c(\tau)}\right\}$$

In particular $c(\tau)$ is the dispersion function, where τ is called the *dispersion parameter*. Show that the Gaussian distribution can be written in this more general form with $c(\tau) = \sigma^2$.

Answer: In part a, we wrote the Gaussian distribution in the exponential family form as:

$$p(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}}\exp\left\{\frac{\mu}{\sigma^2}y - \frac{1}{2\sigma^2}y^2 - \frac{1}{2\sigma^2}\mu^2 - \ln\,\sigma\right\}$$

We can manipulate this slightly to get it into the general exponential family form, as follows:

$$p(y|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi}}\exp\left\{\frac{\mu}{\sigma^2}y - \frac{1/2}{\sigma^2}y^2 - \frac{1/2}{\sigma^2}\mu^2 - \frac{\sigma^2}{\sigma^2}\ln\,\sigma\right\}$$

Thus we have that the dispersion function $c(\tau) = \sigma^2$, as desired.

4. **[17 points] Naive Bayes and Logistic Regression**
For this entire problem assume that the input features x_j, $j = 1, ..., n$ are discrete binary-valued variables such that $x_j \in \{0, 1\}$ and $x = [x_1\ x_2\ ...\ x_n]$. For each training example $x^{(i)}$, assume that the output target variable $y^{(i)} \in \{0, 1\}$.

   (a) [2 points] Consider the Naive Bayes model, given the above context. This model can be parameterized by $\phi_{j|y=0} = p(x_j = 1|y = 0)$, $\phi_{j|y=1} = p(x_j = 1|y = 1)$ and $\phi_y = p(y = 1)$. Write down the expression for $p(y = 1|x)$ in terms of $\phi_{j|y=0}, \phi_{j|y=1}$, and ϕ_y.
   Answer:
   We first use the fact that each x_j has a binomial distribution:

$$\begin{aligned} p(x|y = 0) &= \prod_{j=1}^{n} p(x_j|y = 0) \\ &= \prod_{j=1}^{n} (\phi_{j|y=0})^{x_j} (1 - \phi_{j|y=0})^{1-x_j} \end{aligned} \tag{31}$$

   $p(x|y = 1)$ can be written the same way with $y = 0$ replaced by $y = 1$. Now we can derive $p(y = 1|x)$ using Bayes rule:

$$p(y = 1|x) = \frac{\phi_y(\prod_{j=1}^{n}(\phi_{j|y=1})^{x_j}(1 - \phi_{j|y=1})^{1-x_j})}{\phi_y(\prod_{j=1}^{n}(\phi_{j|y=1})^{x_j}(1 - \phi_{j|y=1})^{1-x_j}) + (1 - \phi_y)(\prod_{j=1}^{n}(\phi_{j|y=0})^{x_j}(1 - \phi_{j|y=0})^{1-x_j})} \tag{32}$$

   Points are given for this or any equivalent solution.

(b) [7 points] Show that the conditional likelihood expression you obtained in part (a) can be simplified to the same form as the hypothesis for logistic regression:

$$p(y=1|x) = \frac{1}{1+e^{-\theta^T x}}. \tag{33}$$

Hint: Modify the definition of x to include the intercept term $x_0 = 1$
Answer:
We begin by dividing the numerator and denominator in the expression from part (a) by the numerator:

$$\begin{aligned} p(y=1|x) &= \frac{\phi_y(\prod_{j=1}^n(\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{1-x_j})}{\phi_y(\prod_{j=1}^n(\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{1-x_j}) + (1-\phi_y)(\prod_{j=1}^n(\phi_{j|y=0})^{x_j}(1-\phi_{j|y=0})^{1-x_j})} \\ &= \frac{1}{1+\frac{(1-\phi_y)(\prod_{j=1}^n(\phi_{j|y=0})^{x_j}(1-\phi_{j|y=0})^{1-x_j})}{\phi_y(\prod_{j=1}^n(\phi_{j|y=1})^{x_j}(1-\phi_{j|y=1})^{1-x_j})}} \\ &= \frac{1}{1+\exp(\log\frac{1-\phi_y}{\phi_y} + \sum_{j=1}^n x_j(\log\frac{\phi_{j|y=0}}{\phi_{j|y=1}}) + (1-x_j)(\log\frac{1-\phi_{j|y=0}}{1-\phi_{j|y=1}}))} \\ &= \frac{1}{1+\exp(\log(\frac{1-\phi_y}{\phi_y} + \frac{1-\phi_{j|y=0}}{1-\phi_{j|y=1}}) + \sum_{j=1}^n x_j(\log\frac{\phi_{j|y=0}}{\phi_{j|y=1}} - \log\frac{1-\phi_{j|y=0}}{1-\phi_{j|y=1}}))} \end{aligned} \tag{34}$$

Now we remember that $x_0 = 1$, so if we set:

$$\begin{aligned} \theta_0 &= -\log\frac{1-\phi_y}{\phi_y} - \sum_{j=1}^n \log\frac{1-\phi_{j|y=0}}{1-\phi_{j|y=1}} \\ \theta_j &= -\log\frac{\phi_{j|y=0}}{\phi_{j|y=1}} + \log\frac{1-\phi_{j|y=0}}{1-\phi_{j|y=1}} \qquad \forall j = 1...n \end{aligned} \tag{35}$$

.
we arrive at the final form:

$$p(y=1|x) = \frac{1}{1+e^{-\theta^T x}} \tag{36}$$

[More space for (b)]

(c) [6 points] In part (b) you showed that the discrete Naive Bayes decision boundary has the same form as that of the logistic regression. Now consider a dataset S_1 with m training examples of the form: $\{(x^{(i)}, y^{(i)}), i = 1, ..., m\}$ with each $x^{(i)} \in \mathbb{R}^{n+1}$. Note that for this problem, S_1 satisfies the Naive Bayes assumption: $p(x_1, \ldots, x_n|y) = \prod_{j=1}^{n} p(x_j|y)$.
Suppose a second dataset, S_2, is given to you again with m training examples $\{(x^{(i)}, y^{(i)}), i = 1, \cdots, m\}$, but now each $x^{(i)} \in \mathbb{R}^{n+2}$ because each $x^{(i)}$ contains the same n conditionally-independent features and an additional feature x_{n+1} such that $x_{n+1} = x_n$. Each $x^{(i)}$ contains the intercept term $x_0 = 1$.

i. [2 points] You train two Naive Bayes classifiers independently on S_1 and S_2. Test data is generated according to the true distribution (i.e. $p(x_1, ..., x_n, y) = p(x_1, ..., x_n, x_{n+1}, y) = p(y)p(x_1, ..., x_n|y)$, where $x_{n+1} = x_n$). Would you expect the test error of the classifier trained on S_1 to be larger or smaller than that trained on S_2? You may assume that m is very large. Briefly justify your answer.

Answer: The expected testing error of the classifier trained on S_1 will be less than the error of the classifier trained on S_2. This is because S_2 violates the conditional independence assumption made by Naive Bayes between features x_n and x_{n+1}, so the model will learn an incorrect joint distribution.

ii. [4 points] Now we will look at a similar situation regarding how logistic regression is affected by copies of features. In order to simplify the math, let's assume a more basic case where S_1 still has m training examples, but now has one feature x_1. S_2 has m training examples but has two features x_1 and x_2 where $x_2 = x_1$. The logistic regression model trained on S_1 therefore has associated parameters $\{\theta_0, \theta_1\}$ and the model trained on S_2 has parameters $\{\theta_0, \theta_1, \theta_2\}$. Here, θ_0 is associated with the intercept term $x_0 = 1$. Testing data is generated the same way (from the original true distribution). How will the error of the classifier trained on S_1 compare to that of the classifier trained on S_2? For this question you need to prove your result mathematically. (Hint: compare the forms of the log-likelihood for each classifier)

Answer: The log-likelihood for the model trained on D_1 can be written as follows: (for simplicity we substitute $g(\theta^T x)$ for $h_\theta(x)$:

$$L_1(\theta) = \sum_i y^{(i)} \log\, g(\theta_0 + \theta_1 x_1^{(i)}) + (1 - y^{(i)})(\log\, (1 - g(\theta_0 + \theta_1 x_1^{(i)}))) \quad (37)$$

The log-likelihood for the model trained on D_2 can be similarly written as:

$$L_2(\theta) = \sum_i y^{(i)} \log\, g(\theta_0' + \theta_1' x_1^{(i)} + \theta_2' x_2^{(i)}) + (1 - y^{(i)})(\log\, (1 - g(\theta_0' + \theta_1' x_1^{(i)} + \theta_2' x_2^{(i)}))) \quad (38)$$

But we know that x_2 is a duplication of x_1, so when we maximize $L_2(\theta)$. It is equivalent to maximizing:

$$\begin{aligned} L_2(\theta) &= \sum_i y^{(i)} \log\, g(\theta_0' + \theta_1' x_1^{(i)} + \theta_2' x_1^{(i)}) + (1 - y^{(i)})(\log\, (1 - g(\theta_0' + \theta_1' x_1^{(i)} + \theta_2' x_1^{(i)}))) \\ &= \sum_i y^{(i)} \log\, g(\theta_0' + (\theta_1' + \theta_2') x_1^{(i)}) + (1 - y^{(i)})(\log\, (1 - g(\theta_0' + (\theta_1' + \theta_2') x_1^{(i)}))) \end{aligned} \quad (39)$$

Looking at this, we see that $L_2(\theta) = L_1(\theta)$ if we choose $\theta_0 = \theta_0'$ and $\theta_1 = \theta_1' + \theta_2'$. This basically means that the logistic regression trained on D_2 will split the weight of θ_1 between θ_1' and θ_2', but the decision boundary will be the same. Therefore, the accuracies of both classifiers will be the same.

(d) [2 points] In general, if we assume that the number of training examples m is very large, which classifier will have a lower generalization error? Briefly justify why.

Answer: The logistic regression will in general have a lower asymptotic error rate because the Naive Bayes classifier makes the conditional independence assumption about the data. Therefore, the logistic regression can learn the Naive Bayes decision boundary; however, the reverse is not true. It turns out that this generalization error relationship is true for any generative-discriminative pair because of the fact that the generative model makes stronger modeling assumptions.

In lecture we showed that if we make assumptions about the distribution of $p(x|y)$ (specifically that this follows a multivariate gaussian), then we can either perform classification using Gaussian Discriminant Analysis or Logistic Regression, and still arrive at a logistic form for the conditional distribution $p(y|x)$. Therefore, we can think of GDA and Logistic Regression as being a generative-discriminative pair where the discriminative model directly estimates the boundary between the class-conditionals that is learned by the generative classifier through Bayes' rule. In this problem, you have shown that a similar generative-discriminative pair property can be derived between Naive Bayes and Logistic Regression for binary classification. This kind of case analysis can provide a lot of insights into the intricate differences between generative and discriminative models.

5. **[15 points] Anomaly Detection**

Consider the following optimization problem:

$$\begin{aligned} \underset{r,z,\xi}{\text{minimize}} \quad & r^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t.} \quad & \left|\left|x^{(i)} - z\right|\right|_2^2 \leq r^2 + \xi_i \;\; i = 1,\ldots,m. \\ & \xi_i \geq 0, \;\; i = 1,\ldots,m. \end{aligned} \tag{40}$$

where ξ_i are the slack variables.

(a) [2 points] Write down the Lagrangian for the optimization problem above. We suggest using two sets of Lagrange multipliers α_i and η_i corresponding to the two inequality constraints so that the Lagrangian would be written as $\mathcal{L}(r, z, \xi, \alpha, \eta)$.
Answer: Using the definition of the Lagrangian, we obtain:

$$\mathcal{L}(r, z, \xi, \alpha, \eta) = r^2 + C\sum_{i=1}^{m}\xi_i - \sum_{i=1}^{m}\alpha_i(r^2 - \left|\left|x^{(i)} - z\right|\right|_2^2 + \xi_i) - \sum_{i=1}^{m}\eta_i\xi_i \tag{41}$$

with $\alpha, \eta \geq 0$.

(b) [7 points] Assuming a non-trivial solution ($r > 0$), derive the dual optimization problem using the Lagrangian from part (a).

Answer: We start by taking derivatives of the Lagrangian with respect to r, z, and ξ and set them to 0:

$$\partial_r \mathcal{L} = 2r - 2\sum_{i=1}^{m} \alpha_i r = 0 \implies \sum_{i=1}^{m} \alpha_i = 1 \tag{42}$$

$$\partial_z \mathcal{L} = \sum_{i=1}^{m} \alpha_i 2(x^{(i)} - z) = 0 \implies z = \frac{\sum_{i=1}^{m} \alpha_i x^{(i)}}{\sum_{i=1}^{m} \alpha_i} = \sum_{i=1}^{m} \alpha_i x^{(i)} \tag{43}$$

$$\partial_\xi \mathcal{L} = C - \alpha_i - \eta_i = 0 \implies \eta_i = C - \alpha_i \tag{44}$$

The result in (43) was obtained by using the result from (42). The last equality (44) shows that we can eliminate η_i by substituting for α_i and instead add the constraint $0 \leq \alpha_i \leq C, \quad \forall i$. We now substitute these values back into the Lagrangian and simplify as follows:

$$\mathcal{L}(r, z, \xi, \alpha, \eta) = r^2 - \sum_{i=1}^{m} \alpha_i r^2 + C\sum_{i=1}^{m} \xi_i - \sum_{i=1}^{m} \alpha_i \xi_i - \sum_{i=1}^{m} (C - \alpha_i)\xi_i + \sum_{i=1}^{m} \alpha_i \left|\left| x^{(i)} - \sum_{j=1}^{m} \alpha_j x_j \right|\right|_2^2 \tag{45}$$

Using (42), we see that the first two terms cancel, and because of our substitution from (44), all of the next 3 terms cancel leaving us with the dual problem:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{m} \alpha_i \left|\left| x^{(i)} - \sum_{j=1}^{m} \alpha_j x_j \right|\right|_2^2, \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1 \ldots m. \\ & \sum_{i=1}^{m} \alpha_i = 1, \quad i = 1 \ldots m. \end{aligned} \tag{46}$$

Points are given for this or any simplified version of this.

(c) [3 points] Show that the dual problem from (b) can be kernelized.

Answer: If we look at the dual from (b) we can simplify the objective further as follows:

$$\sum_{i=1}^{m} \alpha_i \left|\left| x^{(i)} - \sum_{j=1}^{m} \alpha_j x^{(j)} \right|\right|_2^2 = \sum_{i=1}^{m} \left(\alpha_i \langle x^{(i)}, x^{(i)} \rangle - 2\alpha_i \langle x^{(i)}, \sum_{j=1}^{m} \alpha_j x^{(j)} \rangle + \alpha_i \langle \sum_{j=1}^{m} \alpha_j x^{(j)}, \sum_{j=1}^{m} \alpha_j x^{(j)} \rangle \right) \tag{47}$$

$$= \sum_{i=1}^{m} \alpha_i \langle x^{(i)}, x^{(i)} \rangle - \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \tag{48}$$

The last equality was simplified by using (42). Now we can see that the dual objective can be written in terms of inner products of the training data, so the problem can be kernelized by mapping the input data into a higher dimensional feature space using a function $\phi(x)$ and computing the kernel sas $K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$.

(d) [3 points] Now consider the following dual optimization problem

$$\begin{aligned} \max_{\alpha} \quad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j\langle x^{(i)}, x^{(j)}\rangle \\ \text{s.t} \quad & \sum_{i=1}^{m}\alpha_i = 1, \quad i = 1, \ldots, m. \end{aligned} \tag{49}$$

Assume that we choose K such that it is a Gaussian Kernel. How does this dual compare with the dual you derived in part (c)?.

Answer: The objective in (48) only differs from the dual in this part by a constant $\frac{1}{2}$ in front of the second term and the first term $\sum_{i=1}^{m}\alpha_i\langle x^{(i)}x^{(i)}\rangle$ term. However, this inner product is a constant because for a gaussian kernel, $K(x,x) = \kappa$ (a constant), and by the constraint (42) this first term is just a constant. Therefore, the two optimization problems only differ by constant factors and are actually equivalent optimization problems.

The optimization problem defined in part (a) is known as the minimum enclosing ball (MEB) problem, which is commonly used for anomaly detection. The optimization is used to learn a minimum enclosing "ball" defined by a radius r which packs most of the training data close to its center defined by z. A new data point will be considered anomalous if it is outside this ball. The dual in part (d) is actually the dual of the much less intuitive but much more commonly used one-class SVM which is also used for anomaly detection. As you have shown in part (d), it turns out that these optimization problems are actually equivalent when using an isotropic kernel like the Gaussian Kernel.

6. **[15 points] Learning Theory**

   Consider a finite hypothesis class $\mathcal{H}$ with size $k = |\mathcal{H}|$ and $h^\star = \underset{h \in \mathcal{H}}{\operatorname{argmin}}\ \epsilon(h)$.

   (a) [7 points] Assume that the best hypothesis $h^\star$ has generalization error $\epsilon(h^\star) = B$ such that B is a constant with $0 \leq B \leq 1$. Prove that the joint probability of the expected risk minimizer $\hat{h}$ having large generalization error and the best hypothesis h^* having small training error can be bounded as:

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq \sum_{h \in \mathcal{H}} P(\epsilon(h) > B + 2\gamma, \hat{\epsilon}(h) \leq B + \gamma) \quad (50)$$

   Answer: By the definition of ERM, we know that $\hat{\epsilon}(h^\star) \geq \hat{\epsilon}(\hat{h})$, so that

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(\hat{h}) \leq B + \gamma).$$

   Using union bound, we can bound the RHS of last expression as

$$\begin{aligned} P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(\hat{h}) \leq B + \gamma) &\leq P(\exists h \text{ s.t. } \epsilon(h) > B + 2\gamma, \hat{\epsilon}(h) \leq B + \gamma) \\ &\leq \sum_{h \in \mathcal{H}} P(\epsilon(h) > B + 2\gamma, \hat{\epsilon}(h) \leq B + \gamma). \end{aligned}$$

For any hypothesis $h' \in \mathcal{H}$ with high generalization error (i.e. $\epsilon(h') > B' + \tau$), the probability that it has low training error (i.e. $\hat{\epsilon}(h') \leq B'$) is bounded by:

$$P(\hat{\epsilon}(h') \leq B' \mid \epsilon(h') > B' + \tau) \leq \exp\left\{\frac{-m\tau^2}{2(B' + 4\tau/3)}\right\} \tag{51}$$

for any $B' \in (0, 1)$ and $\tau > 0$.

(b) [8 points] Using (51) and the result from part (a), show that:

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq k \exp\left\{\frac{-m\gamma^2}{2(B + 7\gamma/3)}\right\}. \tag{52}$$

Answer: Using the definition of conditional probability and applying (51) with $B' = B + \gamma$ and $\tau = \gamma$, we get

$$\begin{aligned} P(\epsilon(h) > B + 2\gamma, \hat{\epsilon}(h) \leq B + \gamma) &= P(\hat{\epsilon}(h) \leq B + \gamma | \epsilon(h) > B + 2\gamma) P(\epsilon(h) > B + 2\gamma) \\ &\leq P(\hat{\epsilon}(h) \leq B + \gamma | \epsilon(h) > B + 2\gamma) \\ &\leq \exp\left\{\frac{-m\tau^2}{2(B + 7\tau/3)}\right\}. \end{aligned}$$

We can then use the fact that the summation over all hypotheses in $\mathcal{H}$ is from 1 to k to show that

$$P(\epsilon(\hat{h}) > B + 2\gamma, \hat{\epsilon}(h^\star) \leq B + \gamma) \leq k \exp\left\{\frac{-m\gamma^2}{2(B + 7\gamma/3)}\right\}.$$