Introduction to Machine Learning **Fall Semester, 2018/19**

Homework 1: Oct 14th, 2018

Due: Oct 28 (See the submission guidelines in the course web site)

Linear Algebra

1. A matrix A over $\mathbb{R}$ is called *positive semidefinite* (PSD) if for every vector v, $v^\mathsf{T} Av \geq 0$. Show that a symmetric matrix A is PSD if and only if it can be written as $A = XX^T$.

   Hint: a real symmetric matrix A can be decomposed as $A = Q^T DQ$, where Q is an orthogonal matrix and D is a diagonal matrix with eigenvalues of A as its diagonal elements.

2. Show that the set of all symmetric positive semidefinite matrices is *convex*: Namely, that for any two symmetric positive semidefinite matrices A, B and a scalar $0 \leq \theta \leq 1$, $\theta A + (1-\theta)B$ is also symmetric positive semidefinite.

Calculus and Probability

1. *Matrix calculus* is the extension of notions from calculus to matrices and vectors. We define the derivative of a scalar y with respect to a vector $\mathbf{x}$ as the column vector which obeys:

$$\left(\frac{\partial y}{\partial \mathbf{x}}\right)_i = \frac{\partial y}{\partial x_i}$$

   for $i = 1, \ldots, n$. Let A be a $n \times n$ matrix. Prove that: $\frac{\partial \mathbf{x}^T A \mathbf{x}}{\partial \mathbf{x}} = (A + A^T)\mathbf{x}$

2. Let $\mathbf{p} = (p_1, \ldots, p_n)$ be a discrete distribution, with $\sum_{i=1}^n p_i = 1$ and $p_i \geq 0$ for $i = 1, \ldots, n$. The *entropy*, which measures the uncertainty of a distribution, is defined by:

$$H(\mathbf{p}) = -\sum_{i=1}^{n} p_i \log p_i$$

   where we define $0 \log 0 = 0$. Use Lagrange multipliers to show that the uniform distribution has the largest entropy (Tip: Ignore the inequality constraints $p_i \geq 0$ and show that the solution satisfies them regardless).

3. Let $X_0, ..., X_{n-1}$ be n positive ($\forall i, X_i \in [0, \infty)$) i.i.d random variables with a **continuous** probability function f_X. We will prove the following:

$$\mathbb{P}[X_0 \geq \max(X_1, ..., X_{n-1})] = \frac{1}{n}$$

   We will start with a side lemma that will help us with the proof.

   (a) Prove the following lemma: $\mathbb{P}[X_0 \geq \max(X_1, ..., X_{n-1})] = \int_0^\infty (F_{X_0}(a))^{n-1} f_{X_0}(a) da$. Where $F_X(a)$ is the cumulative distribution function (CDF) of X_0 at point a and $f_{X_0}(a)$ is the probability density function (PDF) of X_0 at a.

   (b) Recall that the CDF is an integral over the probability density function (f is continuous) and use this fact to complete the above theorem.

Decision Rules and Concentration Bounds

1. Let X and Y be random variables where Y can take values in $\{1, \ldots, L\}$. Let ℓ_{0-1} be the 0-1 loss function defined in class. Show that $h = \arg \min_{f:\mathcal{X}\to\mathbb{R}} \mathbb{E}\left[\ell_{0-1}(f(X), Y)\right]$ is given by

$$h(x) = \arg \max_{i\in\{1,\ldots,L\}} \mathbb{P}[Y = i | X = x]$$

2. Let $\mathbf{X} = (X_1, \ldots, X_n)^\mathsf{T}$ be a vector of random variables. $\mathbf{X}$ is said to have a **multivariate normal (or Gaussian) distribution** with mean $\boldsymbol{\mu} \in \mathbb{R}^n$ and a $n \times n$ positive definite covariance matrix Σ, if its probability density function is given by

$$f(\mathbf{x}; \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\mathsf{T}\Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$$

where $\mathrm{E}[X_i] = \mu_i$ and $cov(X_i, X_j) = \Sigma_{ij}$ for all $i, j = 1, \ldots, n$. We write this as $\mathbf{X} \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$.

In this question, we generalize the decision rule we have seen in the recitation to more than one dimension. Assume that the data is $\langle \mathbf{x}, y \rangle$ pairs, where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{0, 1\}$. Denote by $f_0(\mathbf{x})$ and $f_1(\mathbf{x})$ the probability density functions of $\mathbf{x}$ given each of the label values. It is known that f_0, f_1 are multivariate Gaussian:

$$f_0(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\mu}_0, \Sigma)$$
$$f_1(\mathbf{x}) = f(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma)$$

Note that the covariance matrix, Σ, is the same for both distributions, but the mean vectors, $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1$, are different. Finally, it is known that the probability to sample a positive sample (i.e. $y = 1$) is p.

   (a) We are given a point $\mathbf{x}$ and we need to label it with either $y = 0$ or $y = 1$. Suppose our decision rule is to decide $y = 1$ if and only if $\mathbb{P}[y = 1|\mathbf{X}] > \mathbb{P}[y = 0|\mathbf{X}]$. Find a simpler condition for $\mathbf{X}$ that is equivalent to this rule.

   (b) The decision boundary for this problem is defined as the set of points for which $\mathbb{P}[y = 1|\mathbf{X}] = \mathbb{P}[y = 0|\mathbf{X}]$. What is the shape of the decision a general $d > 1$ (think of $d = 1$ and $d = 2$ for intuition)?

3. Let $X_1, ..., X_n$ be i.i.d random variables that are uniformly distributed over the interval $[-3, 5]$. Define $S = X_1 + ... + X_n$. Use Hoeffding's inequality to find $N \in \mathbb{N}$ such that for all $n \geq N$

$$\mathbb{P}[S > n^2 + 0.2n] < 0.1$$

4. Suppose we need to build a load balancing device to assign a set of n jobs to a set of m servers. Suppose the j-th job takes L_j time, $0 \leq L_j \leq 1$ (say, in seconds). The goal is to assign the n jobs to the m servers so that the load is as balanced as possible (i.e., so that the busiest server finishes as quickly as possible). Suppose each server works sequentially through the jobs that are assigned to it and finishes in time equal to the sum of job lengths assigned to the server. Let $L = \sum_{j=1}^{n} L_j$ be the total sum of job lengths (assume $L >> m$). With perfect load balancing, each server would take L/m time. There are some good algorithms for this scenario, but we are interested in analyzing the case of random assignment of jobs to servers.

Suppose we assign a random server for each job, with replacement. Denote by $R_{i,j}$ the load on server i from job j - that is, L_j if server i was assigned for job j, or 0 otherwise. Also, let $R_i = \sum_{j=1}^{n} R_{ij}$ be the total load on server i.

(a) What is $\mathrm{E}[R_i]$?

(b) We want to bound the probability that the load on the i-th server is more than $\delta = 10\%$ larger than the expected load. Use the multiplicative form of the Chernoff bound to bound

$$\mathbb{P}[R_i \geq (1+\delta) \cdot \mathrm{E}[R_i]]$$

Note that this form doesn't require the summed random variables to be identically distributed.

(c) Now, we want to bound the probability that **any** of the servers are overloaded by more than $\delta = 10\%$ of the expected load. Give a bound for:

$$\mathbb{P}\big[R_1 \geq (1+\delta) \cdot \mathrm{E}[R_1] \;\; \textit{or} \;\; \ldots \;\; \textit{or} \;\; R_m \geq (1+\delta) \cdot \mathrm{E}[R_m]\big]$$

using the results from (a) and (b) and using the union bound (reminder: for events $A_1, \ldots, A_k$, the union bound is $\mathbb{P}[A_1 \cup \ldots \cup A_k] \leq \sum_{i=1}^{k} \mathbb{P}[A_i]$).

Programming Assignment

1. **Nearest Neighbor.** In this question, we will study the performance of the Nearest Neighbor (NN) algorithm on the MNIST dataset. The MNIST dataset consists of images of handwritten digits, along with their labels. Each image has 28×28 pixels, where each pixel is in grayscale scale, and can get an integer value from 0 to 255. Each label is a digit between 0 and 9. The dataset has 70,000 images. Althought each image is square, we treat it as a vector of size 784.

   The MNIST dataset can be loaded with `sklearn` as follows:

   ```
   >>> from sklearn.datasets import fetch_mldata
   >>> mnist = fetch_mldata('MNIST original')
   >>> data = mnist['data']
   >>> labels = mnist['target']
   ```

   Loading the dataset might take a while when first run, but will be immediate later. See `http://scikit-learn.org/stable/datasets/mldata.html` for more details. Define the training and test set of images as follows:

   ```
   >>> import numpy.random
   >>> idx = numpy.random.RandomState(0).choice(70000, 11000)
   >>> train = data[idx[:10000], :].astype(int)
   >>> train_labels = labels[idx[:10000]]
   >>> test = data[idx[10000:], :].astype(int)
   >>> test_labels = labels[idx[10000:]]
   ```

   It is recommended to use `numpy` and `scipy` where possible for speed, especially in distance computations.

   (a) Write a function that accepts as input: (i) a set of images; (ii) a vector of labels, corresponding to the images (ii) a query image; and (iii) a number k. The function will implement the k-NN algorithm to return a prediction of the query image, given the given label set of images. The function will use the k nearest neighbors, using the Euclidean L2 metric. In case of a tie between the k labels of neighbors, it will choose an arbitrary option.

   (b) Run the algorithm using the first $n = 1000$ training images, on each of the test images, using $k = 10$. What is the accuracy of the prediction (measured by 0-1 loss; i.e. the percentage of correct classifications)? What would you expect from a completely random predictor?

   (c) Plot the prediction accuracy as a function of k, for $k = 1, \ldots, 100$ and $n = 1000$. Discuss the results. What is the best k?

   (d) Using $k = 1$, run the algorithm on an increasing number of training images. Plot the prediction accuracy as a function of $n = 100, 200, \ldots, 5000$. Discuss the results.