ECE 6254, Spring 2024

Homework # 4

Due Sunday, March 3, at 11:59pm EST.

Suggested reading:

- *Elements of Statistical Learning* (by Hastie, Tibshirani, and Friedman): Section 4.3 (pages 106–111) has a discussion of LDA; Section 4.4 (pages 119–128) contains a good discussion of logistic regression; Section 4.5 (pages 129–135) discusses the perceptron learning algorithm and optimal separating hyperplanes; Sections 12.1–12.3 (pages 417–438) discuss support vector machines and kernels in more detail.
- *Learning from Data* (by Abu-Mostafa, Magdon-Ismail, Lin): Section 3.1 (pages 77–82) discusses linear classification and the perceptron learning algorithm; Section 3.3 (pages 88–99) discusses logistic regression, gradient descent, and stochastic gradient descent.

Problems:

1. **Logistic regression and maximum likelihood estimation.** In class, we discussed logistic regression. This problem will derive he gradient of the log-likelihood function, then show three ways to solve for the parameters $\boldsymbol{w}$ and b. Suppose we have n training points $(\boldsymbol{x}_1, y_1), \ldots (\boldsymbol{x}_n, y_n)$ where each $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$ is a "label" that indicates which of two classes $\boldsymbol{x}_i$ corresponds to.

   The key question in fitting a logistic regression model is deciding how to set the parameters $\boldsymbol{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ based on the training data. In this problem, we derive all the necessary quantities to estimate the parameters using gradient descent. For this problem, let g be the logistic function, i.e., $g(t) = \frac{1}{1+e^{-t}}$.

   (a) If $y_i = 1$, we would like to have $g(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \approx 1$, in which case

   $$\log(g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) \approx 0.$$

   Similarly, if $y_i = 0$, we would like $g(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \approx 0$, or said differently, $1 - g(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \approx 1$, in which case

   $$\log(1 - g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) \approx 0.$$

   Note in both cases, the terms on the left-hand side are always negative, so that to make them ≈ 0 corresponds to making these terms *as large* as possible. Combining these desired criteria together, we can express the problem of fitting $\boldsymbol{w}$ and b as

   $$\underset{\boldsymbol{w},b}{\text{maximize}} \sum_{i:y_i=1} \log(g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) + \sum_{i:y_i=0} \log(1 - g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)).$$

   It is somewhat more convenient to re-express this as the equivalent problem:

   $$\underset{\boldsymbol{w},b}{\text{maximize}} \sum_{i=1}^{n} y_i \log(g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)) + (1 - y_i) \log(1 - g(\boldsymbol{w}^\top \boldsymbol{x}_i + b)).$$

Show that, by plugging in the formula for g, this problem reduces to

$$\underset{\boldsymbol{w},b}{\text{maximize}} \sum_{i=1}^{n} y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - \log(1 + e^{\boldsymbol{w}^\top \boldsymbol{x}_i + b}).$$

(b) We would now like to use an iterative algorithm like gradient descent to solve this optimization problem. To do this, we will need to calculate the gradient. Note that if we use the notation

$$\tilde{\boldsymbol{x}}_i = \begin{bmatrix} \boldsymbol{x}_i \\ 1 \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \boldsymbol{w} \\ b \end{bmatrix}$$

then we can re-write our objective function as

$$f(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \boldsymbol{\theta}^\top \tilde{\boldsymbol{x}}_i - \log(1 + e^{\boldsymbol{\theta}^\top \tilde{\boldsymbol{x}}_i})$$

With this notation, compute a formula for the gradient $\nabla f(\boldsymbol{\theta})$.

(c) We will now implement logistic regression using standard gradient descent for performing the maximum likelihood estimation step. Review the slides and lecture notes and implement the functions `log_grad` and `grad_desc` in `logistic_regression.py`. Test out the file by experimenting a bit with the various parameters – especially the "step size" (or learning rate) α – to obtain a good convergence rate. I would recommend trying a wide variety starting with $\alpha \approx 1e-3$ and ranging to $\alpha \approx 1e-6$. You should also feel free to play around with other stopping criteria than those provided. For this problem, report the value of α you used and the number of iterations required for convergence for these parameters.

(d) We now want to implement Newton's method. This algorithm requires computing both the gradient and the Hessian at each iteration. Show that the Hessian in this case is given by

$$\nabla^2 f(\boldsymbol{\theta}) = -\sum_i \tilde{\boldsymbol{x}}_i \tilde{\boldsymbol{x}}_i^\top \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \tilde{\boldsymbol{x}}_i}} \left(1 - \frac{1}{1 + e^{-\boldsymbol{\theta}^\top \tilde{\boldsymbol{x}}_i}}\right).$$

(e) Implement the functions in `log_hess` and `newton` in `logistic_regression.py` to implement Newton's method for solving this problem. Pay attention to the following:

- Take extra care in your implementation in making sure you have the correct signs in your algorithm – remember we are trying to minimize the negative log-likelihood. If your algorithm is not converging, this is a likely suspect.
- In the notes I assume that the $\tilde{\boldsymbol{x}}_i$ are column vectors, but in the Python code they are treated as row vectors.
- Make sure you use the proper operations in python. In particular `A*B` is not a matrix multiplication.

Report the number of iterations required for convergence. In addition, provide a plot showing the "cost" decreasing with iterations for both gradient descent and Newton's method. Compare this to the results from part (c).

(f) We will now implement yet another variant of gradient descent called *stochastic gradient descent* (SGD) to perform the optimization step in logistic regression. In standard gradient descent, in order to compute the gradient we must compute the sum

$$\sum_{i=1}^{n} \widetilde{\boldsymbol{x}}_i(y_i - g(\boldsymbol{\theta}^T \widetilde{\boldsymbol{x}}_i)) \tag{1}$$

In the data set we're using here, this is not much of a challenge, but if our data set is extremely massive (i.e., if n is extraordinarily large) then even simply computing the gradient can be computationally intractable. One possibility in this case is known as *stochastic gradient descent*, and consists of simply selecting one $\widetilde{\boldsymbol{x}}_i$ at random and treating $\widetilde{\boldsymbol{x}}_i(y_i - g(\boldsymbol{\theta}^T \widetilde{\boldsymbol{x}}_i))$ as a rough approximation to (1) and proceeding with the standard gradient descent approach as before. This will (in general) require more iterations, but each iteration can be much cheaper when n is large, so can be very effective when dealing with extremely large data sets.

More generally, we can select a block of k samples $\{\widetilde{\boldsymbol{x}}_{i_1}, \ldots, \widetilde{\boldsymbol{x}}_{i_k}\}$ at random and treat

$$\sum_{j=1}^{k} \widetilde{\boldsymbol{x}}_{i_j}(y_{i_j} - g(\boldsymbol{\theta}^T \widetilde{\boldsymbol{x}}_{i_j})) \tag{2}$$

as a rough approximation to (1).

Implement the general version of stochastic gradient descent by completing the `stoc_grad_desc` function in `lr-sgd.py`. Keep the following in mind:

- **Do not** change the parameters `alpha` , `maxiter`, and `tol`. These have been pretuned for you to ensure relatively smooth convergence.
- Set a stopping criterion based on the change in $\boldsymbol{\theta}$. In particular, you should stop if $\|\boldsymbol{\theta}_k - \boldsymbol{\theta}_{k-1}\|_2 \leq$ `tol`. This is to be consistent with the idea expressed in the footnote.
- You may find the command `np.random.randint(0,n,k)` to be of use.

Provide a plot showing the "cost"[1] decreasing with iterations for block sizes of $k = 1, 5, 10$ and report the number of iterations required for convergence.

2. **Uniform Naïve Bayes.**

(a) Assume that you have access to n i.i.d. realization of $\{x_i\}_{i=1}^n$ of a random variable X distributed uniformly on the interval $[a, b]$. Show that the maximum likelihood estimators for parameters a and b with $a < b$ are

$$\hat{a} = \min_i x_i \qquad \hat{b} = \max_i x_i. \tag{3}$$

[1]Note that when trying to scale the problem to larger scales, you will not want to compute the cost/log-likelihood at every iteration as this alone wipes out any computational savings from using a stochastic estimate of the gradient. It is only for visualization purposes that we compute the cost at every iteration in this problem

(b) Suppose now that you are building a Naïve Bayes classifier for data containing two independent continuous attributes, x_1 and x_2. That is, the data $\boldsymbol{x} = [x_1, x_2]^\top$. Each data point has a corresponding label $y \in \{1, \ldots, K\}$. All classes are assumed to be equally likely. You decide to represent the conditional distributions $f_{X|Y}(x_i|y)$ as uniform distributions and build a Uniform Naive Bayes Classifier. Your conditional distributions would be represented using the parameters $a_{i,k}$ and $b_{i,k}$, with $a_{i,k} < b_{i,k}$, as follows.

$$f_{X|Y}(x_i|y=k) = \begin{cases} \frac{1}{b_{i,k}-a_{i,k}} & \text{if } a_{i,k} \leq x_i \leq b_{i,k} \\ 0 & \text{else.} \end{cases}$$

Here, the index i refers here to the component (1 or 2) of the vector, not to the index of a point in the dataset. Given a set of data, what are the MLEs for $a_{i,k}$ and $b_{i,k}$?

(c) Now suppose that you have estimates $\hat{a}_{i,k}$ and $\hat{b}_{i,k}$ for $a_{i,k}$ and $b_{i,k}$, respectively. Provide a sketch of the decision regions for the case when $K = 2$ (binary classification). Make sure to consider all potential cases.

3. **The Federalist Papers and Naïve Bayes.** In this problem you will explore the use of Naïve Bayes classification applied to a classic text processing problem. Specifically, one of the first usages of the Naïve Bayes approach concerned what is known as the *author attribution problem*. Here we will tackle a particularly famous instance: *who wrote the Federalist Papers?*

   The Federalist Papers were a series of essays written in 1787–1788 meant to persuade the citizens of the State of New York to ratify the Constitution and which were published anonymously under the pseudonym "Publius". In later years the authors were revealed as Alexander Hamilton, John Jay, and James Madison. However, there is some disagreement as to who wrote which essays. Hamilton wrote a list of which essays he had authored only days before being killed in a duel with then Vice President Aaron Burr. Madison wrote his own list many years later, which is in conflict with Hamilton's list on 12 of the essays. Since by this point the two (who were once close friends) had become bitter rivals, historians have long been unsure as to the reliability of both lists.

   We will try to settle this dispute using a simple Naïve Bayes classifier. You will need to download the documents which are in the file `fedpapers_split.txt` as well as some starter code in `fedpapers.py`, both located on the course website. The file `fedpapers.py` loads the documents and builds a "bag of words" representation of each document. Your task is to complete the missing portions of the code and to determine your best guess as to who wrote each of the 12 disputed essays.

   Submit your code along with your answer as to how many of the essays you think were written by Hamilton and how many were by Madison. (Note that there isn't actually a verifiably correct answer here, but there is an answer that has gained broad acceptance among historians.)

4. **Perceptron learning algorithm.** In this problem we are going to prove that the perceptron learning algorithm (PLA) will eventually converge to a linear separator for a separable data set. We will analyze the algorithm assuming for simplicity that the starting point for the

algorithm is given by $\boldsymbol{\theta}^0 = 0$. In the version that we will analyze here, we will suppose that for iterations $j \geq 1$, the algorithm proceeds by setting

$$\boldsymbol{\theta}^j = \boldsymbol{\theta}^{j-1} + y_{i_j} \widetilde{\mathbf{x}}_{i_j},$$

where $(\widetilde{\mathbf{x}}_{i_j}, y_{i_j})$ is any input/output pair in the training data that is mislabeled by the classifier defined by $\boldsymbol{\theta}^{j-1}$. The general approach of the proof will be to argue that for any $\boldsymbol{\theta}^*$ which separates the data, we can show that in a sense $\boldsymbol{\theta}^j$ and $\boldsymbol{\theta}^*$ get more "aligned" as j grows, and that this ultimately yields an upper bound on how many iterations the algorithm can take.

(a) Suppose that $\boldsymbol{\theta}^*$ is normalized so that

$$\rho = \min_i \ |\langle \boldsymbol{\theta}^*, \widetilde{\mathbf{x}}_i \rangle|$$

calculates the distance from the hyperplane defined by $\boldsymbol{\theta}^*$ to the closest $\mathbf{x}_i$ in the training data. Argue that

$$\min_i \ y_i \langle \boldsymbol{\theta}^*, \widetilde{\mathbf{x}}_i \rangle = \rho > 0.$$

(b) Show that $\langle \boldsymbol{\theta}^j, \boldsymbol{\theta}^* \rangle \geq \langle \boldsymbol{\theta}^{j-1}, \boldsymbol{\theta}^* \rangle + \rho$, and conclude that $\langle \boldsymbol{\theta}^j, \boldsymbol{\theta}^* \rangle \geq j\rho$. [Hint: Use induction.]

(c) Show that $\|\boldsymbol{\theta}^j\|_2^2 \leq \|\boldsymbol{\theta}^{j-1}\|_2^2 + \|\widetilde{\mathbf{x}}_{i_j}\|_2^2$. [Hint: Use the fact that $\widetilde{\mathbf{x}}_{i_j}$ was misclassified by $\boldsymbol{\theta}^{j-1}$.]

(d) Show by induction that $\|\boldsymbol{\theta}^j\|_2^2 \leq j(1 + R^2)$, where $R = \max_i \ \|\mathbf{x}_i\|_2$.

(e) Show that (b) and (d) imply that

$$j \leq \frac{(1 + R^2)\|\boldsymbol{\theta}^*\|_2^2}{\rho^2}.$$

[Hint: Use the Cauchy-Schwartz inequality.]

5. **Unconstrained soft margin classifier.** Let $L(x) = \max(0, c - x)$ for some $c > 0$.

(a) Sketch $L(x)$; make sure that c is labeled clearly on your plot.

(b) We have been finding the optimal soft margin classifier by solving a constrained quadratic program.[2] In this problem, I want you to argue that there is an equivalent unconstrained formulation of the form:

$$\underset{\boldsymbol{w}, b}{\text{minimize}} \ \frac{1}{2}\|\boldsymbol{w}\|_2^2 + \sum_{i=1}^{n} L(?).$$

In other words, fill in the question mark, indicate a value for c, and then argue that this unconstrained problem is equivalent to the original constrained quadratic program we used to describe the optimal soft margin classifier in Lecture 12.

[2] Both the original primal problem and the dual problem can be expressed as particular instances of the general class of constrained quadratic programs. Read the supplemental notes for more details.

6. **Kernels.** In the notes I provided on kernels, we proved that if the function $k(\boldsymbol{u}, \boldsymbol{v})$ is a symmetric and positive semi-definite kernel, then it is an "inner product kernel" (i.e., there is a Hilbert space $\mathcal{H}$ with inner product $\langle \cdot, \cdot \rangle$ and a map $\boldsymbol{\Phi} : \mathbb{R}^d \to \mathcal{H}$ such that $k(\boldsymbol{u}, \boldsymbol{v}) = \langle \boldsymbol{\Phi}(\boldsymbol{u}), \boldsymbol{\Phi}(\boldsymbol{v}) \rangle$). Show that the converse is also true: an inner product kernel must be positive semi-definite. (This direction is much easier.)