EE512: Machine Learning

Homework 2

Due 12, March, 2019, Before Class at 07:15 pm *Spring 2019*

1. Make sure to read the questions carefully before attempting them.
2. You are allowed to discuss among yourselves and with TAs for general advice, but you must submit your own work.
3. Plagiarism will be not tolerated.
4. The write-ups must be very clear, to-the-point, and presentable. We will not just be looking for correct answers rather highest grades will be awarded to write-ups that demonstrate a clear understanding of the material. Write your solutions as if you were explaining your answer to a colleague. Style matters and will be a factor in the grade.
5. Codes and their results must be submitted with the homework in hard. Ideally, we would like you to submit a very well documented printout of Jupyter Notebook.

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.
- *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. 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. If you need help, review the slides from lecture 4 (especially pages 11–15, which I updated after the lecture to hopefully help clarify how this approach should work).

   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.)

2. In this problem you will explore implementations of three different solvers for logistic regression. To get started, download the file `lr-gd.py` from the course website.

   (a) We will begin by implementing logistic regression using standard gradient descent for performing the maximum likelihood estimation step. Review the slides from lecture 5 (especially pages 15–22) and make sure you understand what `lr-gd.py` is doing. 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 $\alpha \approx 1$ and ranging to $\alpha \ll 1$. You should also feel free to play around with other stopping criteria than those provided. For this problem, all you need to do is to report the value of α you used and the number of iterations required for convergence for these parameters.

   (b) Using the notes from lecture 5 (page 26), adapt `lr-gd.py` to implement Newton's method for solving this problem. This algorithm requires computing both the gradient and the Hessian at each iteration. The Hessian in this case is given by

$$\frac{\nabla^2 \ell(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{n} \widetilde{\mathbf{x}}_i \widetilde{\mathbf{x}}_i^T g(\boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i)(1 - g(\boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i)),$$

   where g is defined as in the notes. (You may verify this for yourself, but you do not need to.) [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. Note also that in the notes I assume that the $\widetilde{\mathbf{x}}_i$ are column vectors, but in the Python code they are treated as row vectors.]

   Just as a reminder, in Python `A*B` computes the element-wise multiplication, while `A.dot(B)` computes matrix multiplication. You may also want to compute a matrix inverse of `A` using `np.linalg.inv(A)`.

   You should complete this assignment by writing a new function to replace `grad_desc`. Submit the code you produce for this function (do not include the plotting commands or any functions provided in `lr-gd.py`)), and report the number of iterations required for convergence. Compare this to the results from part (a).

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

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

   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 $\widetilde{\mathbf{x}}_i$ at random and treating $\widetilde{\mathbf{x}}_i(y_i - g(\boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i))$ 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.

Implement a version of stochastic gradient descent by by writing a new function to replace `grad_desc` in `lr-gd.py`. Submit your code with your assignment, and report the number of iterations required for convergence. Compare this to the results from part (a). You may find the command `np.random.permutation(n)` to be of use.

(d) Finally, try comparing the results from parts (a), (b), and (c) when applied to a much larger dataset. Try changing `n_samples` to 100 000. You will want to comment out the plotting portion of the code for this part (or at least the part that plots the training data). Record both the running time and the number of iterations for all three algorithms when applied to this dataset. Note that in order to run traditional gradient descent on this large of a dataset, you will need to make α much smaller (think about why this would be the case). If you see warnings regarding a divide by zero, you have set α to be too large. Also, note that you can get the system time in Python by adding `import time` to your file and then using the `time.time()` command.

3. 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.]