EECS 545 – Machine Learning - Homework #2

Due: 11:59pm 10/13/2017

Homework Policy: Working in groups is allowed, but each member must submit their own writeup. Please write the members of your group on your solutions (maximum allowed group size is 4). Homeworks will be submitted via Gradescope (https://gradescope.com/) as pdf files (including your code).

1) **Optimization (30 pts).**

(a) (15 pts) For each of the following functions, determine whether it is convex, strictly convex, concave. or strictly concave. Give the proof.

(i) (5 pts) $f(x) = e^x - 1. \quad x \in \mathbb{R}$.

(ii) (5 pts) $f(x_1, x_2) = x_1 x_2. \quad x_1, x_2 \in \mathbb{R}^2_{++}$.

(iii) (5 pts) $f(x_1, x_2) = x_1^{\alpha} x_2^{1-\alpha}. \quad 0 \leq \alpha \leq 1,\ x_1, x_2 \in \mathbb{R}^2_{++}$.

(b) (15 pts) The cost for **Linear Regression** is given by $J(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{A}^T\boldsymbol{\theta} - \mathbf{y}\|^2$, where $\mathbf{A}$ is the design matrix and $\mathbf{y}$ is the target vector.

(i) (8 pts) **Gradient Descent** is an iterative algorithm which starts with some initial $\boldsymbol{\theta}$ and performs the update: $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{(t)})$ in order to minimize $J(\boldsymbol{\theta})$. Derive the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ and give the expression of the update rule.

(ii) (7 pts) Gradient Descent is one way of updating the values of parameter $\boldsymbol{\theta}$ to arrive at a solution. One can also minimize J without restoring to an iterative algorithm, by finding the closed form solution that minimizes the cost function J. Solving the minimization problem for the cost function J, derive a closed form equation for $\boldsymbol{\theta}$ that minimizes J.

2) **Probability (15 pts).**
One fine day, you happen to stop by an Apple store. You meet a mathematician and are challenged to solve a problem. She tells you that there are 100 customers forming a line inside the Apple store and 50 of them have the **Rose Gold** iPhone X and the remaining 50 have the **Space Gray** iPhone X, shuffled randomly. The mathematician challenges you to predict the iPhone color of the next person exiting the Apple store, where they will exit one by one according to the line.

At any point in the game, if you can correctly predict the color of the iPhone X as **Rose Gold**, the mathematician will gift you a Rose Gold iPhone X. You can choose to skip any number of guesses during the game and see the color of their iPhone X until you decide to guess that next iPhone will be a **Rose Gold**. Note that you will only receive the gift if you correctly predict the color to be **Rose Gold** and the person

exiting the store carries a **Rose Gold** iPhone, *i.e.,* you will not win even if you correctly predict a person with **Space Gray** iPhone. You will have to guess once before all people leave the store.

Since you tell the mathematician that you are taking EECS 545, she challenges you to find an algorithm that has better chances of winning than $\frac{1}{2}$. If you are certain that such an algorithm does not exist, you must prove it. [*Note*: If you skip 99 people, the probability of correctly guessing the last person's iPhone color as **Rose Gold** is $\frac{1}{2}$] ***Problem***: Describe the algorithm or prove that such an algorithm does not exist.

3) **Naive Bayes Classifier (30 pts).**
In this problem, we will use the Naive Bayes Classifier to build a SPAM classifier.

All the files for this problem are in *spam_classification.zip*. The text emails have been preprocessed to get them into a form usable by Naive Bayes. 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 address (HTTPADDR), currency (DOLLAR) and numbers (NUMBER) were also replaced by the special tokens to allow them to be considered properly in the classification process. 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 read 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/message, and the j^{th} column represents the j^{th} distinct word/token. Thus, the (i, j)-entry of this matrix represents the number of occurrences of the j^{th} word in the i^{th} document.

Since the document-word matrix is extremely sparse (has lots of zero entries), we have stored it in a sparse format to save space. These matrices are in SPARSE.* files. Each line of the file corresponds to a row in the document-word matrix, with the following format:

<label> <feature>:<value> <feature>:<value> ...

where label = 1 means the document is a spam; label = -1 means the document is not a spam. Each feature is the index of a word w_j in TOKENS LIST; each value is the count of the word w_j in that document d_i, denoted $c(d_i, w_j)$. The sparse format skips those entries where $c(d_i, w_j) = 0$. Note that the order of words is lost in this feature representation, hence the name "bag-of-words" representation.

(a) (25 points) Implement Naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing (add 1 to the counts). In Naive Bayes classifier, the predicted probability of d_i being a spam follows the Bayes rule:

$$p(y_i = 1|d_i) = \frac{p(d_i|y_i = 1)p(y_i = 1)}{\sum_{y\epsilon\{-1,1\}} p(d_i|y_i = y)p(y_i = y)} \tag{1}$$

Each $p(w_j|y_i = y)$ is estimated as the chance of seeing word w_j if we concatenate all documents of label y:

$$p(w_j|y_i = y) = \frac{1 + \sum_{k\epsilon I_y} c(d_k, w_j)}{\sum_{j=1}^{m} \left(1 + \sum_{k\epsilon I_y} c(d_k, w_j)\right)} \tag{2}$$

where $I_y = \{i : 1 \leq i \leq n, y_i = y\}$ is the index set of all documents with label y. In words, $\sum_{k \epsilon I_y} c(d_k, w_j)$ is the total count of word w_j in all documents with label y. "1+" here is to prevent $p(w_j|y_i = y) = 0$ for any w_j, which is called the *Laplace Smoothing*.

Estimate parameters $\{p(y_i = y), p(w_j|y_i = y), \forall y \epsilon \{-1, 1\}, \forall w_j, j = 1, ..., m\}$ using SPARSE.TRAIN, and then report the error rate on test set SPARSE.TEST.

$$\text{error rate} = \frac{\text{No. of wrongly classified documents in SPARSE.TEST}}{\text{No. of documents in SPARSE.TEST}} \times 100\% \tag{3}$$

(b) (5 pts) 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 j is for the SPAM class by looking at:

$$log\left(\frac{p(w_j|y = 1)}{p(w_j|y = -1)}\right) \tag{4}$$

Using the parameters learned 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 tokens.

4) **K-Nearest Neighbors (25 pts).**

In this problem, you will implement the K-nearest neighbors algorithm and apply it on MNIST dataset for handwritten digit classification. You can learn more about this dataset from this webpage. We provide the dataset in **.mat** so that you can easily import it into Matlab or python.

In Matlab,type

```
load mnist_data
```

In Python, type

```
import numpy as np
import scipy.io as sio
mnist_data = sio.loadmat('mnist_data.mat')
```

In the .mat file, there are two matrices **train** $\in \mathbb{R}^{60000 \times 785}$ and **test** $\in \mathbb{R}^{10000 \times 785}$. Train matrix contains 60000 training images and their labels while test matrix contains 10000 training images and their labels. Each row of these matrices represent a training or testing sample. The first column in each row is the label $\in \{0, 1, 2, \ldots, 9\}$, and the next 784 columns represent flattened 28×28 images.

(a) (15 pts) Tuning hyperparameter K. Implement the K-nearest neighbor algorithm with ℓ_2-norm Random sample 100 test images and classify the selected images with $K \in \{1, 5, 9, 13\}$. For random sampling: in Matlab, you can use `randsample()` while in Python `np.random.choice()` is helpful. Give the classification accuracies for all Ks and give the K value with the best performance. The classification accuracy is defined as: number of correctly classified test images / number of total test images. (**Please write your KNN program in vectorized code. Nested for-loops are very slow! If you compute the distance metric in a vectorized manner, the whole classification process shouldn't take more than several minutes.**).

(b) (10 pts) Change the distance metric to ℓ_1-norm and repeat the experiment. Compare the classification accuracies obtained with ℓ_1-norm and ℓ_2-norm. Which distance metric will you choose for MNIST? Briefly explain the reason.