EECS 545 WN20 Machine Learning
Homework #2

Due date: 11:55pm, Tuesday, Feb 11, 2020

Reminder: While you are encouraged to discuss problems in a small group (up to 5 people), you should write your own solutions and source codes independently. In case you worked in a group for the homework, please include the names of your collaborators in the homework submission. Your answers should be as concise and clear as possible. Please address all questions to `http://piazza.com/class#winter2020/eecs545` with a reference to the specific question in the subject line (E.g. RE: Homework 1, Q2(c)).

Submission Instruction: For homework 2, you should submit both **solution** and **source code**. We may inspect your source code submission visaully and run the code to check the results. Please be aware your points can be deducted if you don't follow the instructions listed below:

- Submit **solution** to **Gradescope**
    - Solution should contain your answers to **all** questions (typed or hand-written).
- Submit **source code** to **Canvas**
    - Source code should contain your codes to programming questions.
    - Source code should be **1 zip** file named **'HW2_yourfirstname_yourlastname.zip'**
    - The zip file should contain **1 ipynb file per question** and should be named as 'qx.ipynb' (x is the question number). We do not allow .py files for HW2.
    - The zip file should also contain the **data** folder we provide. Please preserve the directory structure we set up.

In summary, your source code submission for HW2 should be 1 zip file which includes 3 ipynb files and 3 data folders we provided: q1.ipynb, q2.ipynb, q4.ipynb, q1_data, q2_data, and q4_data, as the programming questions are q1(a,b,d), q2(b), and q4(a,b,c) in HW2.

Source Code Instruction: Your source code should run under an environment with following libraries:

- Python 3.6+
- Numpy (for implementations of algorithms)
- Matplotlib (for plots)

Please do not use any other library unless otherwise instructed. You should be able to load the data and execute your code on someone else's computer with the environment described above. In other words, work with the setup we provide and do not change the directory structure. Use relative path instead of absolute path to load the data. Note, the outputs of your source code must match with your solution.

Credits

Stanford CS229 and Bishop PRML.

1 [22+4 points] k-Nearest Neighbors (kNN)

Given a labeled dataset $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), ..., (\mathbf{x}^{(N)}, y^{(N)})\}$, a KNN algorithm predicts the class $y^{(test)}$ of an unseen test data $\mathbf{x}^{(test)}$ by:

1. Finding the k closest examples to $\mathbf{x}^{(test)}$ from the dataset, i.e. $kNN(\mathbf{x}^{(test)}) = \{(\mathbf{x}^{(1)'}, y^{(1)'}), (\mathbf{x}^{(2)'}, y^{(2)'}), ..., (\mathbf{x}^{(N)'}, y^{(N)'})\}$, such that $\mathbf{x}^{(1)'}, \mathbf{x}^{(2)'}, ..., \mathbf{x}^{(N)'}$ are the best k points among all $\mathbf{x}^{(i)}$ in the training dataset at minimizing $d(\mathbf{x}^{(i)'}, \mathbf{x}^{(test)})$, where $d(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ is a distance measure between $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$.

2. The predicted class label $y^{(test)}$ is the result of a majority vote (Break ties either arbitrarily or randomly):

$$y^{(test)} = \underset{y}{\text{argmax}} \sum_{(\mathbf{x}', y') \in kNN(\mathbf{x}^{(test)})} \mathbf{1}[y' = y]$$

Here, you will use with a small portion of MNIST hand-written digits dataset. Given a test example which is an image of a digit, your end goal is to assign it to the correct class among the 10 classes of MNIST (0 to 9). Load the file **q1_digits.npz** which is a dictionary that contains keys **digits_train**, **labels_train**, **digits_test**, and **labels_test**. We provide **q1.ipynb**, a starter code for data loading and visualization. All the implementations of your algorithms should be done in **q1.ipynb**.

(a) **[11 points]** For the first 5 examples in **digits_test**, find the 8-Nearest neighbors. Assume that $d(x^{(i)}, x^{(j)})$ is the Euclidean distance between the two examples $x^{(i)}$ and $x^{(j)}$ (calculated on the 28 by 28 = 784 pixels). For each of the 5 test digits, write down the indices of the 8 nearest neighbors and their class labels (the index starts from 0). In addition, submit the images of each of the 5 images and its 8 nearest neighbors. Implement your code in **q1.ipynb**.

(b) **[8 points]** Classify all the test images (1000 examples) into the 10 classes using $k = 10$. You are only required to report the classification accuracy in your solution. Implement your code in **q1.ipynb**

(c) **[3 points]** Based on your implementation of kNN, what are advantages and disadvantages of kNN? *[hint: what happens if we have over 100,000 training examples?]* How does the value of k affect classication accuracy? *[hint: run your code using many different values of k and observe how the accuracy changes.]*

(d) **[extra 4 points]** What are ways that can improve the accuracy of this kNN classifier? For full credits, implement your ideas in code, report your improved accuracy and implement your idea in **q1.ipynb**. *[hint: One thing you could do is to come up with a different/better distance function. It could help to view the examples that are incorrectly classified]*

2 [27 points] Softmax Regression via Gradient Ascent

Gradient Ascent is an algorithm used to find parameters that maximize a certain expression (contrary to Gradient Descent, that is used to minimize an expression). For some function $f(\mathbf{w})$, Gradient Ascent finds $\mathbf{w}^* = \text{argmax}_{\mathbf{w}} f(\mathbf{w})$ according to the following pseudo-code:

Algorithm 1 Gradient Ascent

$\mathbf{w}^* \leftarrow$ *random*
repeat
$\mathbf{w}^* \leftarrow \mathbf{w}^* + \alpha \nabla_{\mathbf{w}} f(\mathbf{w}^*)$
until *convergence*
return $\mathbf{w}^*$

Softmax regression is a multiclass classification algorithm. Given a labeled dataset: $D = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), ..., (\mathbf{x}^{(N)}, y^{(N)})\}$, where $y^{(i)} \in \{1, 2, ..., K\}$ (total K classes), Softmax regression computes the probability that an example $\mathbf{x}$ belongs to a class k:

$$p(y = k|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}$$

Where $\mathbf{w}_k$ is a weight vector for class k. The above expression is over-parametrized, meaning that there is more than one unique $\{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_K\}$ that gives identical probability measures for $p(y = k|\mathbf{x}, \mathbf{w})$. An unique solution can be obtained using only $K - 1$ weight vectors $\mathbf{w} = \{\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{K-1}\}$ and xing $\mathbf{w}_K = 0$:

$$p(y = k|\mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^T \phi(\mathbf{x}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}; \;\; \forall k = \{1, 2, ..., K-1\}$$

$$p(y = K|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}))}$$

We define the likelihood of the ith training example $p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w})$ as:

$$p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \prod_{k=1}^{K} \Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbf{I}(y^{(i)}=k)}$$

Where $\mathbf{I}(.)$ is the indicator function. We define the likelihood as:

$$L(\mathbf{w}) = \prod_{i=1}^{N} p(y^{(i)}|\mathbf{x}^{(i)}, \mathbf{w}) = \prod_{i=1}^{N} \prod_{k=1}^{K} \Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbf{I}(y^{(i)}=k)}$$

Finally, we dene the log-likelihood as:

$$l(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^{N} \sum_{k=1}^{K} \log \Big[p(y^{(i)} = k|\mathbf{x}^{(i)}, \mathbf{w})\Big]^{\mathbf{I}(y^{(i)}=k)}$$

(a) **[13 points]** Derive the gradient ascent update rule for the log-likelihood of the training data. In other words, derive the expression $\nabla_{\mathbf{w}_m} l(\mathbf{w})$ for $m = 1, ..., K - 1$. Show that:

$$\nabla_{\mathbf{w}_m} l(\mathbf{w}) = \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \left[\mathbf{I}(y^{(i)} = m) - \frac{\exp(\mathbf{w}_m^T \phi(\mathbf{x}^{(i)}))}{1 + \sum_{j=1}^{K-1} \exp(\mathbf{w}_j^T \phi(\mathbf{x}^{(i)}))}\right]$$

$$= \sum_{i=1}^{N} \phi(\mathbf{x}^{(i)}) \Big[\mathbf{I}(y^{(i)} = m) - p(y^{(i)} = m | \mathbf{x}^{(i)}, \mathbf{w}) \Big]$$

[Hints: $\log a^b = b \log(a)$*. Further, it helps to consider the two cases separately; a case for for* $y^{(i)} = k = m$*, and another for* $y^{(i)} = k \neq m$*, which is equivalent to using Kronecker delta* δ_{km}*].*

(b) **[14 points]** Using the gradient computed in part (a), implement Gradient Ascent for Softmax Regression in **q2.ipynb**. Use a learning rate $\alpha = 0.0005$. Load **q2_data.npz**, which is a dictionary that contains **q2x_train**, **q2y_train**, **q2x_test**, and **q2y_test**. Train your classifier on the training data and **report the accuracy** on the test data in your solution. Implement your code in **q2.ipynb**. Recall that Softmax Regression classifies an example $\mathbf{x}$ as:

$$y = \underset{y'}{\operatorname{argmax}}\, p(y' | \mathbf{x}, \mathbf{w})$$

While you must implement your own Softmax Regression from scratch, you can use the logistic regression function from sklearn (**sklearn.linear_model.LogisticRegression**) to validate your results. Your results should not be much less than the accuracy of the predictions from sklearn's **LogisticRegression**.

3 [22 points] Gaussian Discriminate Analysis

Suppose we are given a dataset $\{(\mathbf{x}^{(i)}, y^{(i)}); i = 1, ..., N\}$ consisting of N independent examples, where $\mathbf{x}^{(i)} \in \mathbb{R}^M$ are M-dimensional vectors, and $y^{(i)} \in \{0, 1\}$. We will model the joint distribution of $(\mathbf{x}^{(i)}, y^{(i)})$ using:

$$p(y^{(i)}) = \phi^{y^{(i)}}(1-\phi)^{1-y^{(i)}}$$

$$p(\mathbf{x}^{(i)}|y^{(i)}=0) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_0)^T\Sigma^{-1}(\mathbf{x}^{(i)}-\mu_0))$$

$$p(\mathbf{x}^{(i)}|y^{(i)}=1) = \frac{1}{(2\pi)^{\frac{M}{2}}|\Sigma|^{\frac{1}{2}}} \exp(-\frac{1}{2}(\mathbf{x}^{(i)}-\mu_1)^T\Sigma^{-1}(\mathbf{x}^{(i)}-\mu_1))$$

Here, the parameters of our model are ϕ, Σ, μ_0, and μ_1. (Note that while there are two different mean vectors μ_0 and μ_1, there is only one covariance matrix Σ.)

(a) **[8 points]** Suppose we have already fit ϕ, Σ, μ_0, and μ_1, and now want to make a prediction at some new query point $\mathbf{x}$. Show that the posterior distribution of the label (y) at $\mathbf{x}$ takes the form of a logistic function, and can be written as

$$p(y=1|\mathbf{x};\phi,\Sigma,\mu_0,\mu_1) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$$

where $\mathbf{w}$ is a function of ϕ, Σ, μ_0, and μ_1. *[Hint: For part (a) only, you may want to redefine* $\mathbf{x}$ *to be* $M+1$ *dimensional vectors by adding an extra coordinate* $\mathbf{x}_0 = 1$*.] [Hint 2:* $\mathbf{w}^T\mathbf{x}$ *is a scalar.]*

(b) **[8 points]** When M (the dimension of $\mathbf{x}^{(i)}$) is 1, then $\Sigma = \left[\sigma^2\right]$ becomes a scalar, and its determinant is $|\Sigma| = \sigma^2$. Accordingly, the maximum likelihood estimates of the parameters are given by

$$\phi = \frac{1}{N}\sum_{i=1}^{N} 1\{y^{(i)}=1\}$$

$$\mu_0 = \frac{\sum_{i=1}^{N} 1\{y^{(i)}=0\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} 1\{y^{(i)}=0\}}$$

$$\mu_1 = \frac{\sum_{i=1}^{N} 1\{y^{(i)}=1\}\mathbf{x}^{(i)}}{\sum_{i=1}^{N} 1\{y^{(i)}=1\}}$$

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}^{(i)}-\mu_{y^{(i)}})(\mathbf{x}^{(i)}-\mu_{y^{(i)}})^T.$$

The log-likelihood of the data is

$$\ell(\phi,\mu_0,\mu_1,\Sigma) = \log\prod_{i=1}^{N} p(\mathbf{x}^{(i)}, y^{(i)};\phi,\mu_0,\mu_1,\Sigma)$$

$$= \log\prod_{i=1}^{N} p(\mathbf{x}^{(i)}|y^{(i)};\phi,\mu_0,\mu_1,\Sigma)p(y^{(i)};\phi)$$

By maximizing ℓ with respect to the four parameters, prove that the maximum likelihood estimates of ϕ, Σ, μ_0, and μ_1 are indeed the ones described above. (You may assume that there is at least one positive and one negative example, so that the denominators in the definitions of μ_0 and μ_1 above are non-zero.)

(c) **[6 points]** Redo part (b) for $M \neq 1$, excluding the maximum likelihood estimate for Σ.

4 [29 points] Naive Bayes for classifying SPAM

In this problem, we will use the naive Bayes algorithm to build a SPAM classifier.

In recent years, email spam has been an increasing problem. Here, we'll build a classifier to distinguish between "real (non-spam)" emails, and spam emails. For this problem, we obtained a set of spam emails, and a set of genuine newsgroup messages. Using only the subject line and body of each message, the classifier will learn to distinguish between the two groups of emails.

In our data, the text emails have been pre-processed so that they can be used for naive Bayes. The pre-processing ensures that only the email 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 classication process. (In this problem, we'll going to call the features "tokens".) For whomever interested in the pre-processing, two examples for spam emails and their pre-processed forms and one example for non-spam email and its pre-processed form are in the folder **samples_FYI**.

We have done the feature extraction works for you, so you can just load the data matrices (called document-term matrices in text classication) which contain all the data. In a document-term 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 jth token in the ith document.

For this problem, we chose the set of tokens (vocabulary) to only contain the medium frequency tokens, as the tokens that occur too often or too rarely do not have much classication value. (Examples: tokens that occur very often are terms like "the," "and," and "of," which occur in any spam and non-spam emails.) Also, terms 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 token. For a list of the tokens used, see the le **TOKENS_LIST.txt** in the **samples_FYI** folder.

Since the document-term matrix is sparse (has lots of zero entries), we store it in an efficient format to save space. We provide a starter code **q4.ipynb** which contains the **readMatrix** function that reads in the document-term matrix, the correct class labels for all emails, and the full list of tokens.

(a) **[12 points]** Implement a naive Bayes classifier for spam classification, using the multinomial event model and Laplace smoothing. You should use functions provided in **q4.ipynb** to train your parameters with the training dataset **MATRIX.TRAIN**. Then, use these parameters to classify the test dataset **MATRIX.TEST** and report the error using the **evaluate** function. Implement your code in **q4.ipynb**.

Remark. If you implement naive Bayes in the straightforward way, you'll note that the computed $p(\mathbf{x}|y) = \prod_j p(x_j|y)$ often equals zero. This is because $p(\mathbf{x}|y)$, 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. You'll have to find a way to compute naive Bayes' predicted class labels without explicitly representing very small numbers such as $p(\mathbf{x}|y)$. *[Hint: Think about using logarithms.]*

(b) **[8 points]** Some tokens may be particularly indicative of an email being spam or non-spam. One way to measure how indicative a token i is for the SPAM class by looking at:

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

Using the parameters you obtained in part (a), find the 5 tokens that are most indicative of the SPAM class (i.e., 5 tokens that have the highest positive value on the measure above). Implement your code in **q4.ipynb**.

(c) **[9 points]** Repeat part (a), but with different naive Bayes classifiers each trained with different training sets **MATRIX.TRAIN.***. Evaluate the classifiers with **MATRIX.TEST** and report the classification error for each classifier. Plot a graph of the test error with respect to size of training sets. Which training-set size gives you the best classification error? Implement your code in **q4.ipynb**.