# EECS 545 WN22 Machine Learning
# Homework #2

**Due date: 11:55pm, Tuesday, Feb 8, 2022**

**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 `https://piazza.com/class/kxyu0owhjju7gu` with a reference to the specific question in the subject line (E.g. RE: Homework 2, Q1(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 **Gradescope**

**Code Submission Instructions** Your solutions to Q1, Q2, and Q4 should each be written in a *single* python program `q1.py`, `q2.py`, and `q4.py` respectively. The program should be runnable with the following command line: `python3 q1.py` (or `python3 q2.py`, `python3 q4.py`) and produce outputs and plots for *all* subproblems. The program should read the data files (i.e., `*.npy`) **from the same (current) working directory**: for example, `X_train = np.load('q1x.npy')` and for Q4, `readMatrix('q4_data/MATRIX.TRAIN')`.

The program should print all necessary outputs (e.g. coefficients computed) into standard output (stdout). For plots, it is okay to save figures as multiple files (e.g. `q1-b.png`) as you want. There are no requirements on the filename or format, as long as it produces valid outputs. Be sure to include all required outputs in your writeup.

Please upload your code files (`q1.py`, `q2.py`, `q4.py`) to Gradescope. Please do **NOT** include data files (`*.npy`) into your Gradescope submission.
**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 [21 points] Logistic regression

(a) **[8 points]** Consider the log-likelihood function for logistic regression:

$$\ell(\mathbf{w}) = \sum_{i=1}^{N} y^{(i)} \log h(\mathbf{x}^{(i)}) + (1 - y^{(i)}) \log(1 - h(\mathbf{x}^{(i)})),$$

where $h(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+\exp(-\mathbf{w}^T\mathbf{x})}$.

Find the Hessian $H$ of this function, and show that is negative semi-definite and thus $\ell$ is concave and has no local maxima other than the global one. That is, show that

$$\mathbf{z}^T H \mathbf{z} \leq 0$$

for any vector $\mathbf{z}$. [*Hint: You might want to start by showing the fact that* $\sum_i \sum_j z_i x_i x_j z_j = (\mathbf{x}^T\mathbf{z})^2$. *Note that* $(\mathbf{x}^T\mathbf{z})^2 \geq 0$.]

(b) **[8 points]** Using the $H$ you calculated in part (a), write down the update rule implied by Newton's method for optimizing $\ell(\mathbf{w})$. Now use this rule (and not a library function) to implement Newton's method and apply it to binary classification problem specified in files `q1x.npy` and `q1y.npy`. The two columns of `q1x.npy` represent the inputs ($x^{(i)}$) and `q1y.npy` represents the outputs ($y^{(i)} \in \{0, 1\}$), with one training example per row. Initialize Newton's method with $\mathbf{w} = \mathbf{0}$ (the vector of all zeros). What are the coefficients $\mathbf{w}$, including the intercept term, resulting from your fit? [*Hint: You might refer to the code in* `q1_hint.py` *to start programming.*]

(c) **[5 points]** Plot the training data. (Your axes should be $x_1$ and $x_2$, corresponding to the two coordinates of the inputs, and you should use a different symbol for each point plotted to indicate whether that example had label 1 or 0.) Also plot on the same figure the decision boundary fit by logistic regression. (This should be a straight line showing the boundary separating the region where $h(\mathbf{x}) > 0.5$ from where $h(\mathbf{x}) \leq 0.5$.)

## 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, which 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 fixing $\mathbf{w}_K = 0$:

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

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

We define the likelihood of the $i$th 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 define 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* $y^{(i)} = k = m$*, and another for* $y^{(i)} = k \neq m$*, which is equivalent to using Kronecker delta* $\delta_{km}$*].*

(b) **[14 points]** Using the gradient computed in part (a), implement gradient ascent for softmax regression in **q2.py**. 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.py**. 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 $\phi, \Sigma, \mu_0$, and $\mu_1$. (Note that while there are two different mean vectors $\mu_0$ and $\mu_1$, there is only one covariance matrix $\Sigma$.)

(a) **[8 points]** Suppose we have already fit $\phi, \Sigma, \mu_0$, and $\mu_1$, 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 $\phi, \Sigma, \mu_0$, and $\mu_1$. *[Hint 1: 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 $\ell$ with respect to the four parameters, prove that the maximum likelihood estimates of $\phi, \Sigma, \mu_0$, and $\mu_1$ 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 $\mu_0$ and $\mu_1$ above are non-zero.)

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

# 4 [30 points] Naive Bayes for classifying SPAM

In this problem, we will use the naive Bayes algorithm to build a SPAM classifier. Our classifier will distinguish between "real" (non-spam) emails and spam emails. For this problem, we obtained a set of spam emails and a set of non-spam newsgroup messages. Using only the subject line and body of each message, the classifier will learn to distinguish between the two groups.

In this problem, we're going to call the features *tokens*.

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 special tokens to allow them to be considered properly in the classification process. If you are interested in the pre-processing, two examples for spam emails and their pre-processed forms and one example for a non-spam email and its pre-processed form are in the folder **samples_FYI**.

We have done the feature extraction for you, so you can just load the data matrices (called document-term matrices in text classification) 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 $(i, j)$th entry of this matrix represents the number of occurrences of the $j$th token in the $i$th document.

For this problem, we chose the set of tokens (also called a *vocabulary*) to only contain the medium frequency tokens, as the tokens that occur too often or too rarely do not have much classification 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 file **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_hint.py** 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_hint.py** 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.py**.

**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.py**.

(c) **[10 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.py**.

## Update log

- (Jan/27th/14:30): Added 1 point to Q4 (c) to make point total 100.