EECS 545 HW 2

Due Wednesday, Sept. 23, by 11:59 PM Eastern Time via Gradescope

When you upload your solutions to gradescope, please indicate which pages of your solution are associated with each problem.

1. Worst Bayes Risk (5 points)

Consider multiclass classification with K classes. As a function of K, what is the largest possible value of the Bayes Risk R^*? Justify your answer.

2. Naïve Bayes for Spam Filtering

In this problem you will apply the naïve Bayes classifier to the problem of spam detection, using a benchmark database assembled by researchers at Hewlett-Packard. Download the HW2 files found fom Canvas under Files → Homework → HW2. The file spambase.data contains the data. The file loadspam.py contains the following code for loading the training data, and extracting feature vectors and labels.

```
import numpy as np
z = np.genfromtxt('spambase.data', dtype=float, delimiter=',')
np.random.seed(0) #Seed the random number generator
rp = np.random.permutation(z.shape[0]) #random permutation of indices
z = z[rp,:] #shuffle the rows of z
x = z[:,:-1]
y = z[:,-1]
```

Note: If you copy and paste the above, you may need to delete and retype the single quotes to avoid an error. This has to do with the optical character recognition of pdfs on some systems.

Here `x` is $n \times d$, where $n = 4601$ and $d = 57$. The different features correspond to different properties of an email, such as the frequency with which certain characters appear. `y` is a vector of labels indicating spam or not spam. For a detailed description of the dataset, visit the UCI Machine Learning Repository, or Google 'spambase'.

To evaluate the method, treat the first 2000 examples as training data, and the rest as test data. Fit the naïve Bayes model using the training data (i.e., estimate the class-conditional marginals), and compute the misclassification rate (i.e., the test error) on the test data. The code above randomly permutes the data, so that the proportion of each class is about the same in both training and test data.

You should implement this algorithm yourself. Do not use existing implementations.

(a) (10 pts) Quantize each variable to one of two values, say 1 and 2, so that values below the median map to 1, and those above map to 2. In Python, use `np.median(a,axis=0)` to calculate the columnwise medians of an array `a`.

Report the test error. As a sanity check, what would be the test error if you always predicted the same class, namely, the majority class from the training data?

Your code should follow best coding practices including the use of comments, indentation, and descriptive variable names. Submit your code as a single .py file to the corresponding assignment on *Canvas* designated for this purpose. In addition, please also submit a .pdf version of your code (just print the .py file to pdf) and upload to gradescope. This will make it easier for graders who want to take a quick look at your code without running it.

Note: On the spam detection problem, please note that you will get a different test error depending on how you quantize values that are **equal** *to the median. It makes a difference whether you quantize values equal to the median to 1 or 2. You should quantize all medians the same way – I'm not suggesting that you try all* 2^d *combinations. So just make sure you try both options, and report the one that works better.*

Note: The notes mention a technique ("Laplacian smoothing") that adds 1 to the numerator and L to the denominator, where L is number of class. In this problem you **do not** need to implement this technique.

3. Logistic regression objective function (5 points each)

Consider the regularized logistic regression objective (penalized negative log-likelihood)

cost function $$J(\boldsymbol{\theta}) = -\ell(\boldsymbol{\theta}) + \lambda\|\boldsymbol{\theta}\|^2$$

where

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left[y_i \log\left(\frac{1}{1+e^{-\boldsymbol{\theta}^T\tilde{\boldsymbol{x}}_i}}\right) + (1-y_i)\log\left(\frac{e^{-\boldsymbol{\theta}^T\tilde{\boldsymbol{x}}_i}}{1+e^{-\boldsymbol{\theta}^T\tilde{\boldsymbol{x}}_i}}\right)\right].$$

Here $\tilde{\boldsymbol{x}}_i = [1\ x_{i1}\ \cdots\ x_{id}]^T$ and $y_i \in \{0,1\}$.

(a) Show that if we change the label convention in logistic regression from $y \in \{0,1\}$ to $y \in \{-1,1\}$, then

$$-\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log\left(1+\exp\left(-y_i\boldsymbol{\theta}^T\tilde{\boldsymbol{x}}_i\right)\right).$$

Introduce the notation $\phi(t) = \log(1+\exp(-t))$ so that, by the previous problem, the logistic regression regularized negative log-likelihood may be written

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{n} \phi(y_i\boldsymbol{\theta}^T\tilde{\boldsymbol{x}}_i) + \lambda\|\boldsymbol{\theta}\|^2. \tag{1}$$

(b) Calculate the gradient of $J(\boldsymbol{\theta})$ using (1). Your answer may be in the form of a summation. Simplify your result so that it involves only the full vectors $\boldsymbol{x}_i$ (or $\tilde{\boldsymbol{x}}_i$) and not their components.

(c) Calculate the Hessian of $J(\boldsymbol{\theta})$. This will be a $(d+1)\times(d+1)$ matrix. Again your result may be in the form of a summation. Simply your result so that it involves only the full vectors $\boldsymbol{x}_i$ (or $\tilde{\boldsymbol{x}}_i$) and not their components.

(d) Use the previous result to argue that J is a convex function of $\boldsymbol{\theta}$ when $\lambda \geq 0$, and strictly convex when $\lambda > 0$.

4. Logistic Regression for Handwritten Digit Recognition (15 points)

Download the file `mnist_49_3000.mat` from Canvas under Files → Homework → HW2. This is a subset of the MNIST handwritten digit database, which is a well-known benchmark database for classification algorithms. This subset contains examples of the digits 4 and 9.

The data file contains variables `x` and `y`, with the former containing patterns and the latter labels. The images are stored as column vectors. To visualize an image, in Python run

```
import numpy as np
import scipy.io as sio
import matplotlib.pyplot as plt

mnist_49_3000 = sio.loadmat('mnist_49_3000.mat')
x = mnist_49_3000['x']
y = mnist_49_3000['y']
d,n= x.shape

i = 0 #Index of the image to be visualized
plt.imshow(np.reshape(x[:,i], (int(np.sqrt(d)),int(np.sqrt(d)))))
plt.show()
```

The above code can be found in the file `loadmnist.py`.

Note: If you copy and paste the above, you may need to delete and retype the single quote to avoid an error. This has to do with the optical character recognition of pdfs on some systems.

Implement Newton's method (a.k.a. Newton-Raphson) to find a minimizer of the regularized negative log likelihood $J(\boldsymbol{\theta})$ from the previous problem. Set $\lambda = 10$. Use the first 2000 examples as training data, and the last 1000 as test data. As a termination criterion, let i be the final iteration as soon as $|J(\boldsymbol{\theta}_i) - J(\boldsymbol{\theta}_{i-1})|/J(\boldsymbol{\theta}_{i-1}) \leq \epsilon := 10^{-6}$. Let the initial iterate $\boldsymbol{\theta}_0$ be the zero vector.

- Report the test error, the number of iterations run, and the value of the objective function after convergence.
- In addition, generate a figure displaying 20 images in a 4 x 5 array. These images should be the 20 misclassified images for which the logistic regression classifier was most confident about its prediction (you will have to define a notion of confidence in a reasonable way – explain what this is). In the title of each subplot, indicate the true label of the image. What you should expect to see is some 4s that look kind of like 9s and 9s that look kind of like 4s.
- Your code should follow best coding practices including the use of comments, indentation, and descriptive variable names. Submit your code as a single .py file to the corresponding assignment on *Canvas* designated for this purpose. In addition, please also submit a .pdf version of your code (just print the .py file to pdf) and upload to gradescope. This will make it easier for graders who want to take a quick look at your code without running it.

Some additional remarks:

- Helpful Python commands and libraries: numpy.log, numpy.exp, numpy.sum, numpy.sign, numpy.zeros, numpy.repeat, str(), matplotlib.pyplot.
- Note that the labels in the data are ± 1, whereas the notes (at times) assume that the labels are 0 and 1.
- It is possible to "vectorize" Newton's method, that is, implement it without any additional loops besides the main loop. If you would prefer to do this, please consult the book by Hastie, Tibshirani, and Friedman and look for the iterative reweighted least squares (IRLS) implementation. Do note that they may have different notation than what was presented in class. That said, if you just use an additional loop to calculate the Hessian at each iteration, and it shouldn't take too long.

5. Weighted Least Squares Regression (5 points)

Consider linear regression and let $c_1, \ldots, c_n > 0$ be known weights. Determine the solution of

$$\min_{\boldsymbol{w},b} \sum_{i=1}^{n} c_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i - b)^2.$$

Express your solution in terms of the matrix $\boldsymbol{C} = \mathrm{diag}(c_1, \ldots, c_n)$ an appropriate data matrix $\boldsymbol{X}$, and other notation as needed.

Hint: As a sanity check, your solution should reduce to the ordinary least squares solution when C is the identity matrix.