CS416 – HW2

Introduction

All problems are designed by Dan Sheldon.

Due Friday 10/2 at 11:59pm. Please complete all exercises and problems below.

- All the files can be found in

  `http://www.cs.wm.edu/~liqun/teaching/cs416_20f/hw2/`

- You can also download the zip file:

  `http://www.cs.wm.edu/~liqun/teaching/cs416_20f/hw2/hw2.zip`

- You can also copy to your directory on a department machine by:

  `cp ~liqun/public_html/teaching/cs416_20f/hw2/* .`

Your submission consists of three steps:

1. Create hw2.pdf with your solutions to the following problems. Put your name and your login id in the file.
2. You'll need to create or edit these files in the directory hw2. Complete the requested code in these files.
   - logistic_regression.ipynb
   - logistic_regression.py
   - sms_classify.ipynb
   - digit-classification.ipynb
   - one_vs_all.py
3. Submit:

Problem 1

Logistic Regression

Let $g(z) = \frac{1}{1+e^{-z}}$ be the logistic function.

1.1 (5 points). Show that $\frac{d}{dz}g(z) = g(z)(1 - g(z))$.

1.2 (5 points). Show that $1 - g(z) = g(-z)$.

1.3 (5 points). Consider the log loss function for logistic regression simplified so there is only one training example:

$$J(\theta) = -y \log h_\theta(x) - (1 - y) \log(1 - h_\theta(x)),\ h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

Show that the partial derivative with respect to θ_j is:

$$\frac{\partial}{\partial \theta_j} J(\theta) = (h_\theta(x) - y)x_j$$

Problem 2 (10 points).

Logistic regression for book classification. In this problem, you will implement logistic regression for book classification. Open the jupyter notebook logistic_regression.ipynb and follow the instructions to complete the problem.

Problem 3 (10 points).

SMS spam classification. In this problem you will use your implementation of logistic regression to create a spam classifier for SMS messages. Open the jupyter notebook sms_classify.ipynb and follow the instructions to complete the problem.

In this assignment you will implement one-vs-all multiclass logistic regression to classify images of hand-written digits. Then you will explore the effects of regularization and training set size on training and test accuracy. Open the Jupyter notebook digit-classification.ipynb and follow the instructions to complete the problems.

4 Hand-Written Digit Classification

In this assignment you will implement multi-class classification for hand-written digits and run a few experiments. The file digits-py.mat is a data file containing the data set, which is split into a training set with 4000 examples, and a test set with 1000 examples.

You can import the data as a Python dictionary like this:

```
data = scipy.io.loadmat('digits-py.mat')
```

The code in the cell below first does some setup and then imports the data into the following variables for training and test data:

- X_train - 2d array shape 4000 x 400
- y_train - 1d array shape 4000
- X_test - 2d array shape 1000 x 400
- y_test - 1d array shape 1000

```
In [ ]: %matplotlib inline
        %reload_ext autoreload
        %autoreload 2

        import numpy as np
        import matplotlib.pyplot as plt

        # Load train and test data
        import scipy.io
        data = scipy.io.loadmat('digits-py.mat')
        X_train = data['X_train']
        y_train = data['y_train'].ravel()
        X_test = data['X_test']
        y_test = data['y_test'].ravel()
```

4.1 (2 points) Write code to visualize the data

Once you have loaded the data, it is helpful to understand how it represents images. Each row of X_train and X_test represents a 20 x 20 image as a vector of length 400 containing the pixel intensity values. To see the original image, you can reshape one row of the train or test data into a 20 x 20 matrix and then visualize it using the matlplotlib imshow command.

Write code using np.reshape and plt.imshow to display the 100th training example as an image. (Hint: use cmap='gray' in plt.imshow to view as a grayscale image.)

```
In [ ]: # Write code here
```

4.2 (2 points) Explore the data

A utility function display_data is given to you to further visualize the data by showing a mosaic of many digits at the same time. For example, you can display the first 25 training examples like this:

```
display_data( X_train[:25, :] )
```

Go ahead and do this to visualize the first 25 training examples. Then print the corresponding labels to see if they match.

```
In [ ]: from one_vs_all import display_data
        # Write code here
```

4.3 Alert: notation change!

Please read this carefully to understand the notation used in the assignment. We will use logistic regression to solve multi-class classification. For three reasons (ease of displaying parameters as images, compatibility with scikit learn, previewing notation for SVMs and neural networks), we will change the notation as described here.

4.3.1 Old notation

Previously we defined our model as

$$h_\theta(x) = \text{logistic}(\theta_0 + \theta_1 x_1 + ... + \theta_n x_n) = \text{logistic}(\theta^T x)$$

where

- $x = [1, x_1, \cdots, x_n]$ was a feature vector with a 1 added in the first position
- $\theta = [\theta_0, \theta_1, \cdots, \theta_n]$ was a parameter vector with the intercept parameter θ_0 in the first position

4.3.2 New notation

We will now define our model as

$$h_w(x) = \text{logistic}(b + w_1 x_1 + ... + w_n x_n) = logistic(w^T x + b)$$

where

- $x \in \mathbb{R}^n$ is the **original feature vector** with no 1 added
- $w \in \mathbb{R}^n$ is a **weight vector** (equivalent to $\theta_1, \cdots, \theta_n$ in the old notation)
- b is a scalar **intercept parameter** (equivalent to θ_0 in our old notation)

4.4 (10 points) One-vs-All Logistic Regression

Now you will implement one vs. all multi-class classification using logistic regression. Recall the method presented in class. Suppose we are solving a K class problem given training examples in the data matrix $X \in \mathbb{R}^{m \times n}$ and label vector $y \in \mathbb{R}^m$ (the entries of y can be from 1 to K).

For each class $c = 1, \cdots, K$, fit a logistic regression model to distinguish class c from the others using the labels

$$y_c^{(i)} = \begin{cases} 1 & \text{if } y(i) = c \\ 0 & \text{otherwise.} \end{cases}$$

This training procedure will result in a weight vector w_c and an intercept parameter b_c that can be used to predict the probability that a new example x belongs to class c:

$\text{logistic}(w_c^T x + b_c)$ = probability that x belongs to class c.

The overall training procedure will yield one weight vector for each class. To make the final prediction for a new example, select the class with highest predicted probability:

predicted class = the value of c that maximizes $\text{logistic}(w_c^T x + b_c)$.

4.4.1 Training

Open the file one_vs_all.py and complete the function train_one_vs_all to train binary classifiers using the procedure outlined above. You are given a function for training a regularized logistic regression model, which you can call like this:

```
weight_vector, intercept = train_logistic_regression(X, y, lambda_val)
```

Follow the instructions in the file for more details. Once you are done, test your implementation by running the code below to train the model and display the weight vectors as images. You should see images that are recognizable as the digits 0 through 9 (some are only vague impressions of the digit).

```
In [ ]: from one_vs_all import train_one_vs_all

        lambda_val = 100
        weight_vectors, intercepts = train_one_vs_all(X_train, y_train, 10, lambda_val)
        display_data(weight_vectors.T) # display weight vectors as images
```

4.4.2 Predictions

Now complete the function predict_one_vs_all in one_vs_all.py and run the code below to make predictions on the train and test sets. You should see accuracy around 88% on the test set.

```
In [ ]: from one_vs_all import predict_one_vs_all

        pred_train = predict_one_vs_all(X_train, weight_vectors, intercepts)
        pred_test = predict_one_vs_all(X_test, weight_vectors, intercepts)

        print("Training Set Accuracy: %f" % (np.mean(pred_train == y_train) * 100))
        print(" Test Set Accuracy: %f" % (np.mean( pred_test == y_test) * 100))
```

4.5 (5 points) Regularization Experiment

Now you will experiment with different values of the regularization parameter λ to control overfitting. Write code to measure the training and test accuracy for values of λ that are powers of 10 ranging from 10^{-3} to 10^{5}.

- Display the weight vectors for each value of λ as an image using the display_data function
- Save the training and test accuracy for each value of λ
- Plot training and test accuracy versus lambda (in one plot).

```
In []: lambda_vals = 10**np.arange(-3., 5.)
       num_classes = 10

       # Write code here
       # In your final plot, use these commands to provide a legend and set
       # the horizontal axis to have a logarithmic scale so the value of lambda
       # appear evenly spaced.
       plt.legend(('train', 'test'))
       plt.xscale('log')
```

4.6 (5 points) Regularization Questions

1. Does the plot show any evidence of overfitting? If so, for what range of values (roughly) is the model overfit? What do the images of the weight vectors look when the model is overfit?

2. Does the plot show any evidence of underfitting? For what range of values (roughly) is the model underfit? What do the images of the weight vectors look like when the model is underfit?

3. If you had to choose one value of , what would you select?

4. Would it make sense to run any additional experiments to look for a better value of . If so, what values would you try?

** Your answers here **

4.7 (6 points) Learning Curve

A learning curve shows accuracy on the vertical axis vs. the amount of training data used to learn the model on the horizontal axis. To produce a learning curve, train a sequence of models using subsets of the available training data, starting with only a small fraction of the data and increasing the amount until all of the training data is used.

Write code below to train models on training sets of increasing size and then plot both training and test accuracy vs. the amount of training data used. (This time, you do not need to display the weight vectors as images and you will not set the horizontal axis to have log-scale.)

```
In [ ]: m, n = X_train.shape

        train_sizes = np.arange(250, 4000, 250)
        nvals = len(train_sizes)

        # Example: select a subset of 100 training examples
        p = np.random.permutation(m)
        selected_examples = p[0:100]
        X_train_small = X_train[selected_examples,:]
        y_train_small = y_train[selected_examples]

        # Write your code here
```

4.8 (4 points) Learning Curve Questions

1. Does the learning curve show evidence that additional training data might improve performance on the test set? Why or why not?

2. Is the any relationship between the amount of training data used and the propensity of the model to overfit? Explain what you can conclude from the plot.

** Your answers here *