CS 335 Homework 4, Fall 2018

Dan Sheldon

Instructions

Due Thursday 3/7 at 11:55pm.

How to submit

In this assignment you will edit the following files included with the homework in the directory `hw4-files`:

- `digit-classification.ipynb`
- `hw4.py`

There is an additional data file and script that you will not need to edit:

- `digits-py.mat`
- `pprint_hw4.py`

Complete the requested code in these files, then follow these steps:

1. **Upload code listing to Gradescope**. Make sure you set your path to include the anaconda binary directory. On the lab computers, this is the correct command:

   ```
   $ export PATH=/anaconda/bin:$PATH
   ```

   When you are done editing and ready to submit your code listing, run the `pprint_hw4.py` script from the `hw4-files` directory:

   ```
   $ python pprint_hw4.py
   ```

   This will create a new file called `hw4-code.pdf` with a listing of all of your code and results. Open the pdf to make sure it is correct and includes all of your code and plots. You can run this multiple times if you update your code. Upload this to Gradescope.

2. **Submit a single zip file containing source code to Moodle.** Make sure your code is complete and files are included in the directory. Also include any auxiliary data or code files you created that are needed to run your code.

   Rename your code directory from `hw4-files` to `hw4-<your last name>` and zip it:

   ```
   $ mv hw4-files hw4-sheldon
   $ zip -r hw4-sheldon.zip hw4-sheldon
   ```

   Submit the single zip file to Moodle.

Problems

In this assigment 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.

1 NOTEBOOK LISTING: `digit-classification.ipynb`

2 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()
```

2.1 (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
```

2.2 (2 points) Explore the data

I wrote a utility function `display_data` for 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 hw4 import display_data

        # Write code here
```

2.3 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.

2.3.1 Old notation

Previously we defined our model as

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

where

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

2.3.2 New notation

We will now define our model as

$$h_{\mathbf{w}}(\mathbf{x}) = \text{logistic}(b + w_1 x_1 + \ldots + w_n x_n) = \text{logistic}(\mathbf{w}^T \mathbf{x} + b)$$

where

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

2.4 (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 $\mathbf{y} \in \mathbb{R}^m$ (the entries of $\mathbf{y}$ can be from 1 to K).

For each class $c = 1, \ldots, K$, 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 $\mathbf{w}_c$ and an intercept parameter b_c that can be used to predict the probability that a new example $\mathbf{x}$ belongs to class c:

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

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:

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

2.4.1 Training

Open the file `hw4.py` and complete the function `train_one_vs_all` to train binary classifiers using the procedure outlined above. I have included a function for training a regularized logistic regression model, which you can call like this:

```
weight_vector, intercept = fit_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 hw4 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
```

2.4.2 Predictions

Now complete the function `predict_one_vs_all` in `hw4.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 hw4 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))
```

2.5 (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')
```

2.6 (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* **

2.7 (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
```

3 (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* **