CS 335: Machine Learning
Homework 4

Dan Sheldon

October 25, 2014

Instructions

Due Friday 10/31 by midnight

- Write your name on your submission
- Write the name of all students with whom you collaborated
- Cite any sources you used other than the textbook, course notes, or Coursera

Files

The zip archive `hw4-files.zip` contains the files needed for this assignment. When unzipped, it will expand into a directory `hw4-files`. These are the files you will have to edit as instructed in the assignment:

- `problem_1.m`
- `one_vs_all.m`
- `predict_one_vs_all.m`
- `problem_2.m`
- `problem_3.m`

These are the ones you won't have to change:

- `fit_logistic_regression.m`
- `regularized_cost_function.m`
- `logistic.m`
- `fmincg.m`
- `display_data.m`
- `resize.m`
- `digits.mat`
- `digits2.mat`

What to Submit

Here is what to submit:

- `hw4-files-<yourlastname>.zip`. A single zip file containing the directory with all of the code files, including those you edited as well as those you did not edit. (Please preserve the original filenames of all files.)
- `report.pdf`. A single pdf file containing the plots and answers to questions you are asked below about the experiments. Please submit your report in pdf format instead of docx, etc.

There are no written problems on this assignment.

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.mat` is a MATLAB 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 into your MATLAB workspace by typing:

```
>> load digits.mat
```

This will create the following four variables:

- `X_train` - size 4000×400
- `y_train` - size 4000×1
- `X_test` - size 1000×400
- `y_test` - size 1000×1

Recall that the X matrices are arranged so that each row is a training example, i.e.,

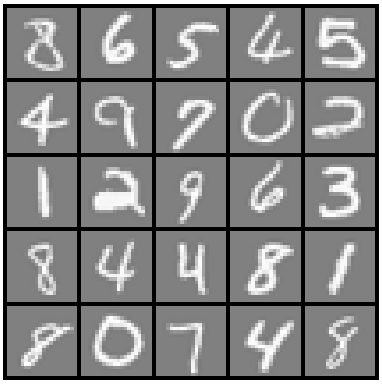
$$X = \begin{bmatrix} — & (\mathbf{x}^{(1)})^T & — \\ — & (\mathbf{x}^{(2)})^T & — \\ & \vdots & \\ — & (\mathbf{x}^{(m)})^T & — \end{bmatrix},$$

and the vector $\mathbf{x}^{(i)} \in \mathbb{R}^{400}$ is a vector containing all of the pixel values of the 20×20 grayscale image from the ith training example.

Because this is a multi-class classification problem, the `y_test` and `y_train` vectors have entries from 1 to 10, where the labels from 1 to 9 represent the digits from 1 to 9, and the label 10 represents the digit 0. You can visualize the data using the `display_data.m` function provided with the assignment. For example, to display the first 25 training examples, type the following

```
>> display_data(X_train(1:25,:));
```

You will see the following



You can check that the labels for the first two rows are indeed correct:

```
>> y_train(1:10)'

ans =

     8     6     5     4     5     4     9     7    10     2
```

Problem 1: One-vs-All Logistic Regression (10 points)

In this problem 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 $\boldsymbol{\theta}_c$ that can be used to predict the probability that a new example $\mathbf{x}$ belongs to class c:

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

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:

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

Open the script `problem_1.m`. It will load the data and call the procedure `one_vs_all()` to train a multi-class logistic regression model on the training set, and then call `predict_one_vs_all.m` to make predictions on the test set. It will then print some statistics and display the learned weight vectors as images. However, it will not work properly until you implement the functions as instructed below.

Training

Complete the code in `one_vs_all.m` to train the binary classifiers using the procedure outline above. You should return the weight vectors all together in one matrix, so that `weight_vectors(:,c)` gives the fitted weight vector for class c. I have included an optimized routine called `fit_logistic_regression()` to fit a *regularized* logistic regression model, which you can invoke as follows:

```
theta = fit_logistic_regression( X, y, lambda, theta_init );
```

Look inside the file for more documentation. You need to call this function with the appropriate input arguments. The input parameter `lambda` (λ) controls how much we penalize large weights in the regularized cost function. You will experiment with different values of λ in the next problem.

Hint: You may find the expression `y == c` useful, where `y` is a vector and `c` is a scalar. This compares each entry of `y` to `c`, and produces the *vector* of 0/1 values that result from the individual comparisons.

Checking that training is working properly When you have correctly implemented the training procedure, you should see the following number of the training accuracy:

```
Training Set Accuracy: 92.775000
```

(If you are within one or two percent of this but do not get the exact number, you are probably doing OK, but it might be worth checking with me—say, by posting on Piazza!)

The code will also display the weight vectors as images. You should also be able to recognize the digits, but they may not be perfect looking specimens of the digits.

Prediction

Now complete the code in the file `predict_one_vs_all.m` to make predictions using the learned weight vectors. Once you have implmemented both training and prediction correctly, you should now see the following number for test set accuracy:

```
Test Set Accuracy: 91.000000
```

Problem 2: Regularization (15 points)

In this problem, you will experiment with different values of the regularization parameter λ to control overfitting.

The file `problem_2.m` is currently a stub that load the data and sets a few variables. Write code in this file to measure the training and test accuracy for values of λ that are powers of 10 ranging from 10^{-7} to 10^{0}. For each value of λ, save an image of the weight vectors to file (see comments in the file for instructions).

When you are done, plot the training and test accuracy vs. λ. To plot two different vectors on the vertical axis versus one set of values on the horizontal axis, you can use the `plot()` command as follows:

```
plot(lambda_vals, [train_acc test_acc]);
```

Then, provide a legend and set the horizontal axis to be log-scale so the λ values appear evenly spaced:

```
legend('train', 'test');
set(gca, 'xscale', 'log');
```

Save your figure using the either the `print` command or the GUI, and include it in your writeup. Then answer the following questions.

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

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

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

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

Problem 3: Learning Curve (10 points)

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.

Complete the code in the script `problem_2.m` to produce a learning curve. Read the comments in the file for more instructions. Use commands similar to those in Problem 2 to plot both training and test accuracy vs. the amount of training data used. (This time, you will not set the horizontal axis to have log-scale.) Include the plot in your writeup, and answer the following questions:

1. Does the plot 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.

For Fun

Confusion Matrix

Accuracy measures the percentage of correctly classified digits, but fails to let us know where our classifier is making its mistakes. For example, perhaps we always correctly classify examples of the digit 1, bu make many more mistakes on examples of the digit 4. Or perhaps we commonly confuse the digits 3 and 6.

A "confusion matrix" can help explore these questions. Read about confusion matrices (on Wikipedia or elsewhere) and create a confusion matrix for one of the classifiers from Problems 1-3. Use the confusion matrix to answer the question: which are the two most commonly confused digits by the classifier?

SVM vs. Logistic Regression Comparison

Would SVMs perform better than logistic regression on this problem? `libsvm` is a freely available implementation of SVM training and prediction with a MATLAB interface. It can be downloaded from `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`. Install `libsvm` and test it out on a small example problem so you understand the usage. Then try SVMs instead of logistic regression as the base learner for this multiclass classification problem and run experiments to see if you can make it perform better than logistic regression.

(**Note:** Pay attention to the `libsvm` interface for making predictions after you train a model—while it is possible to extract a weight vector and make predictions yourself, it is usually less error-prone to treat their model data structure as a black box and use it in conjunction with the provided prediction methods to make predictions.)