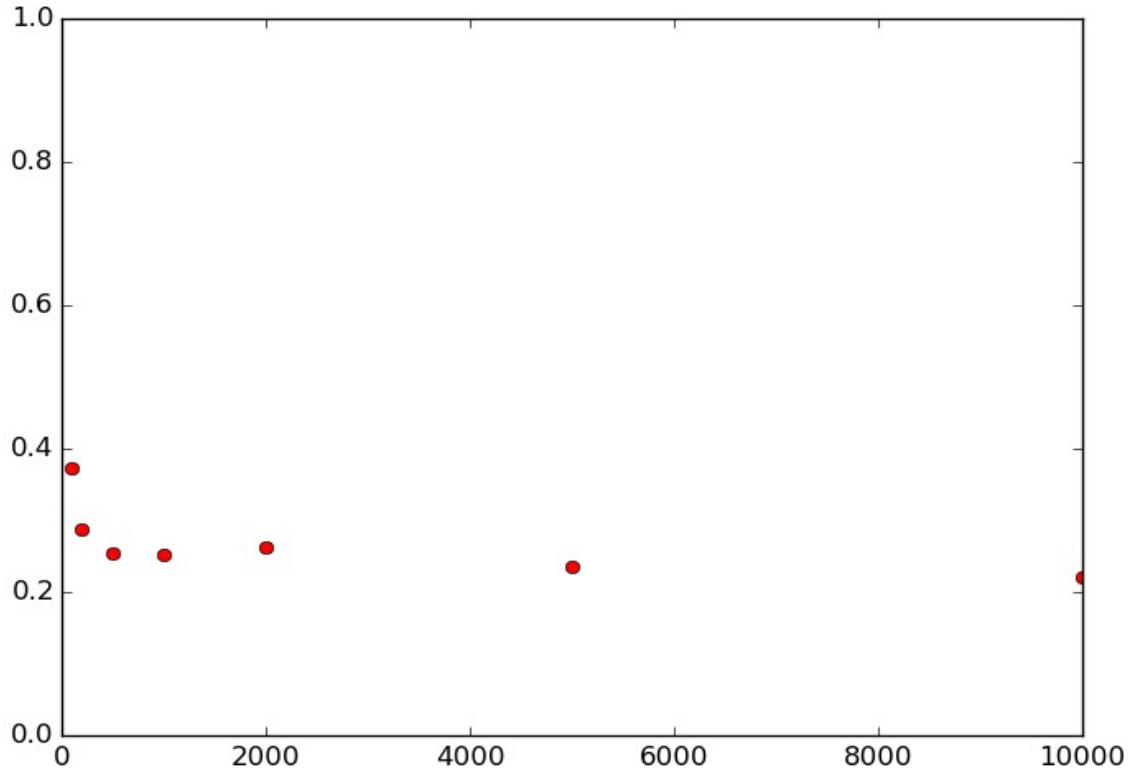
CS 189 Homework 1: Support Vector Machines
Kevin Chau
23816929

Running the code:
The code for problems 1, 2, and 3 can be found in **digitSVM.py**. To use *digitSVM.py*, make sure it is in the digit-dataset/ directory, or equivalently in the same folder as train.mat and test.mat. In the command line, execute "python -i digitSVM.py". For problem 1, call the function **plotRateVsExamples()** in the interpreter. For problem 2, call the function **makeConfusionMatrices()** to train the data and save multiple confusion matrices files. For problem 3, call the function **crossValidation10fold()**, which should output a list of cross-validation errors and the optimal value of the C parameter. The code for problem found can be found in **spamSVM.py.** The file should be adjacent to spam_data.mat. Like problem 3, you can call the function **crossValidation10fold()** to train the spam data.

Write-up
Problem 1: Train a linear SVM using raw pixels as features. Plot the error rate on a validation set versus the number of training exam- ples that you used to train your classifier. The choices of the number of training examples should be 100, 200, 500, 1,000, 2,000, 5,000 and 10,000. Make sure you set aside 10,000 other training points as a validation set. You should expect accuracies between 70% and 90% at this stage.

I implemented the solution for this problem using the python SVM from sklearn. Running the function plotRateVsExamples() in the python interpreter produces the following results:

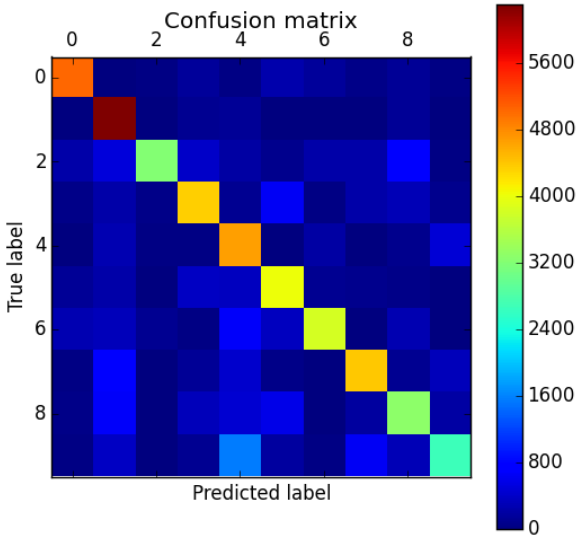


The X axis represents the number of training examples and the Y axis represents the error rate

(percentage of test cases that were misclassified).
The plot shows that the smallest error rate occurred when there was the largest number of training examples. More precisely, at 10000 training examples, there was an error of about 21%, which corresponds to an accuracy of 79%. In general, the error rate decreased as the number of training examples increased.

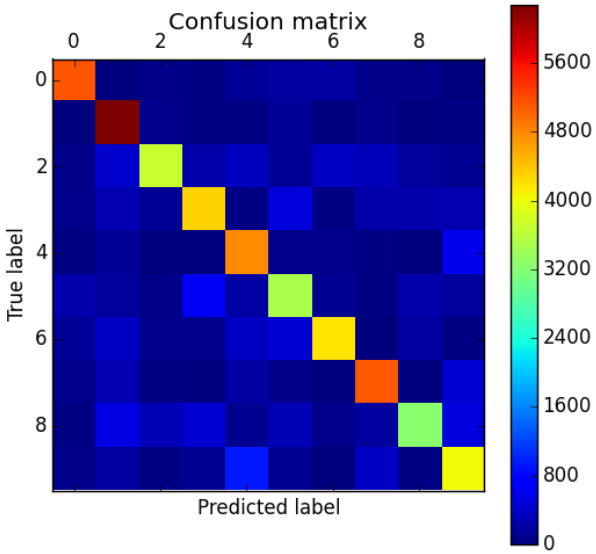
Problem 2*:* Create confusion matrices for your experiments in Problem 1. Color code your results and report them. You can use built-in implementations to generate confusion matrices. What in- sights can you get about the performance of your algorithm from looking at the confusion matrix?

Training Examples = 100

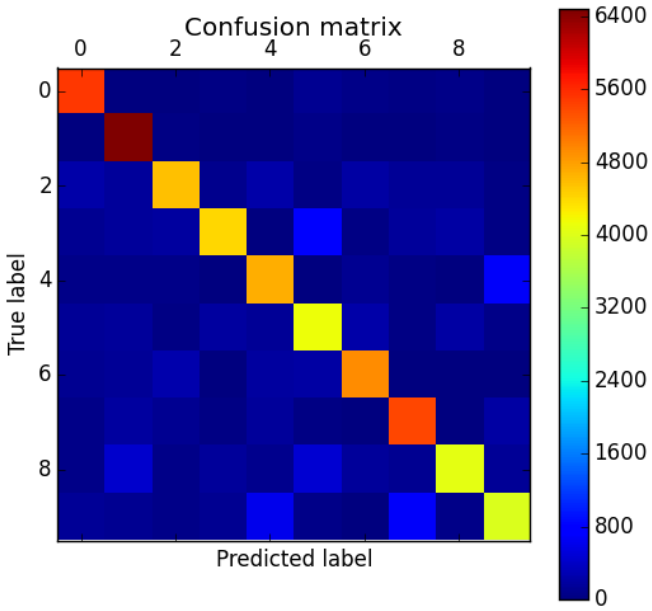


]

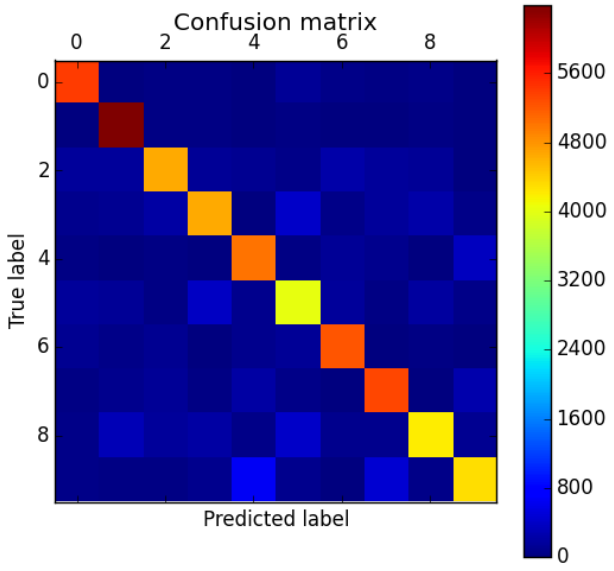
Training Examples = 200



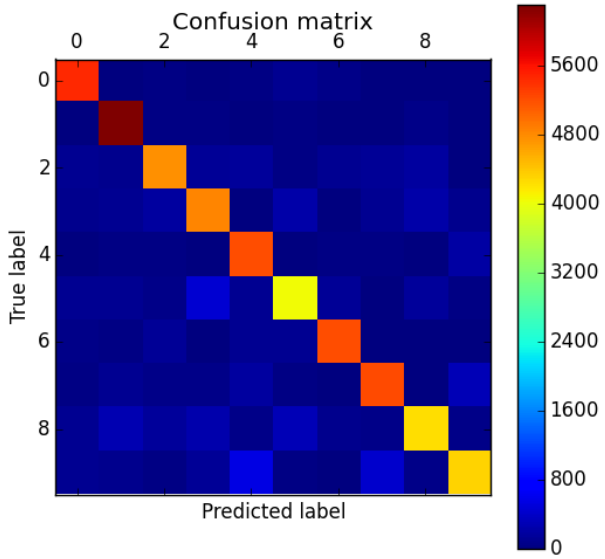
Training Examples = 500



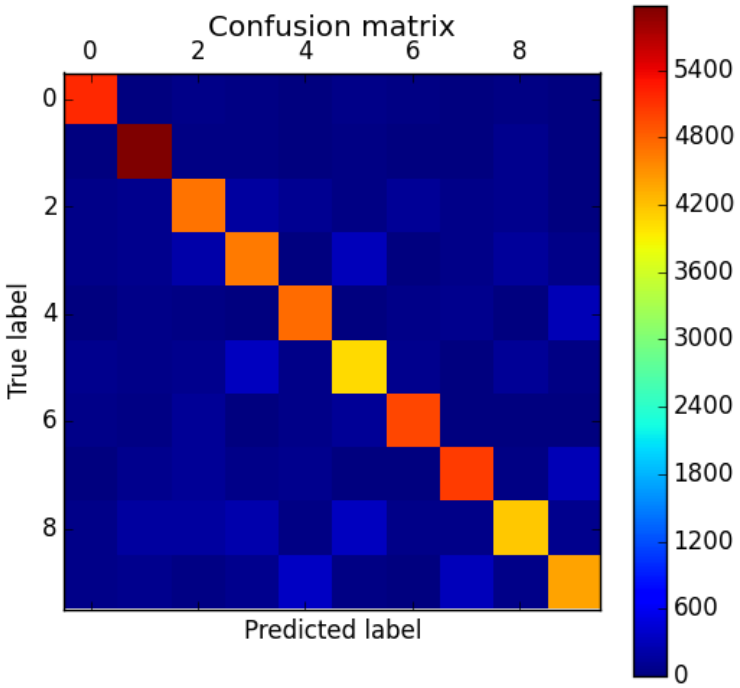
Training Examples = 1000



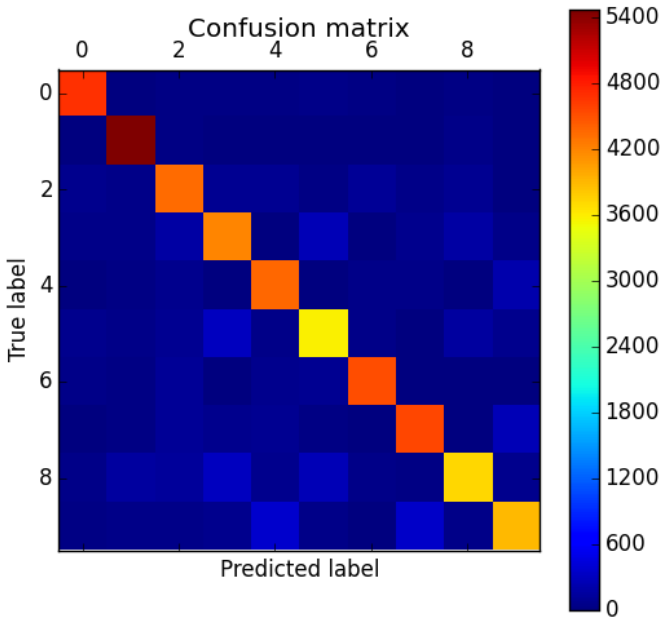
Training Examples = 2000



Training Examples = 5000



Training Examples = 10000



I created 7 confusion matrices, one for each choice of number of training examples: 100, 200, 500, 1000, 2000, 5000, and 10000. The matrices show us that we were consistently classifying the digits 0 and 1 fairly well, even when we had a small number of training examples. However, when there were few training examples, it was harder for the SVM to correctly classify and distinguish the other digits. Specifically for 100 training examples, the SVM poorly classified digits 2, 8, and 9. 200 training examples did not seem to yield significant improvements over 100. In general, as the number of training examples increased, the confusion matrices show more accurate digit clarification. At 10000 training examples, most of the diagonal entries have high values (more red in color). Overall, 1's were the most accurately classified.

Problem 3. Explain why cross-validation helps. Find the optimal value of the parameter *C* using *10-fold* cross-validation on the training set with 10,000 examples. Train an SVM with this value of *C* and report the validation error rate.

Cross validation helps us overcome the problem of over fitting a model that our SVM produces when we train the data. The reason is that by dividing the data into multiple sets and computing multiple error rates (one for each k-fold set), we get a better idea of what the error rate for a particular value of C actual is. This is because the variance in error rates for validating on each of the k-fold partitions is not negligible, so averaging them gives us a better picture than if we just did one error rate calculation on the whole entire training set.

Using 10-fold cross validation on the digit data set, I found the optimal parameter of C by trying out a wide range of orders of magnitudes and then tweaking the decimal values more precisely by hand. I found that values between 1e-7 and 5e-7 improved my accuracy a lot from my original value of C = 1 in problem 1. With a value of 2e-7, I got a cross validation error of .1428. When I trained the SVM with C = 2e-7, I got an even better accuracy with an error rate of 14.0%. So cross validation is indeed useful.

Problem 4. Train a linear SVM for your spam dataset. Please state your *C* value, your accuracy using cross validation, and (briefly) what features you added, removed, or modified, if any. Adding features is optional.

I copied the code from digitSVM and modified it so that it could work with the spam data in the file spamSVM.py. Running 10fold cross validation on the c values [.1,.2,.5,1,2,5,10,15,18,19,20,21,21.5,22.05,22.5,23,23.5,24,25,50,100,200, 500, 1000], I found that the optimum c for the spam data set was C = 50. The cross validation error was .18588. Training an SVM on the test data, I got a validation error rate of only .0736.

Kaggle: My Kaggle submissions can be found in the files **digits.csv** and **spam.csv.** My best digit score was .86780 with C = 1e-7 and my best spam score was .78309 with C = 50.