Matthew O’Brien
Homework 3
February 16, 2016

Homework 3 Overview

The purpose of this project was to have a better understanding of the different parameters, and techniques when using a support vector machine. Throughout this assignment, I used the SVM package SKlearn for Python. SKlearn had a lot of different possibly libraries that I could have imported into this assignment. SKlearn even had a K-Fold library, but according the assignment guidelines, I implemented my own version of that function. I did use the preprocessing library from SKlearn to easily skip the normalization by hand portion of the data processing. SKlearn also had the function calls for accuracy, precision, and recall. I only had to do the False Positive Rate calculation within my program, and I will go into more detail about that under experiment 2 and 3. This project covered the implementation and testing of K-Fold cross validation to find the best C parameter for the SVM, and two different types of feature selections. The first feature selection is a “best weight” or the most valuable features for predictions from greatest to smallest. And a randomized selection, to see if the number of features compared to higher importance would make a difference for the accuracies of predictions.

Experiment 1:

The same SVM package was used for experiment one, as stated previously, which was SKlearn. I actually ran this program quite a few times to see if I could notice any sort of differences between the C parameter selected and the accuracies coming out. The C* or best C parameter was normally between 0.8 and 1.0. The following table shows the different trials ran, the value of C* selected by the K-Fold cross validation, and the corresponding values for accuracy, precision, and recall. For this project, the value of K was 10. With 200 evenly placed thresholds. My implementation allows for these values to changed easily, but I did not run any different values because of time restraints.

Table 1. C* values and corresponding accuracy precision, and recall.

C*	Accuracy	Precision	Recall
1	93.82	94.01	93.6
0.9	93.82	94.11	93.49
0.8	93.76	93.91	93.6

The values 0.8, 0.9, and 1.0 were the most common C* values. This wasn’t necessarily proven to be the most common, just an observation. There were a couple of trials were the C* value was actually as low as 0.6. Though the values for accuracy, precision, and recall stayed above 90% for all of them. It would have been interesting to have set the function call in a way to loop through my cross validation, reporting the C* and corresponding values, then plot those to see the different values vary depending on the amount of times each C* value was seen. Since I didn't even have time to finish this report on time, I did not attempt to create any more testing scenarios, except for those that I stumbled onto with my own mistakes. The following figure is the ROC curve for the final learned model. I did not alter the x or y axis from 0 to 1 because that was always have the examples were shown.

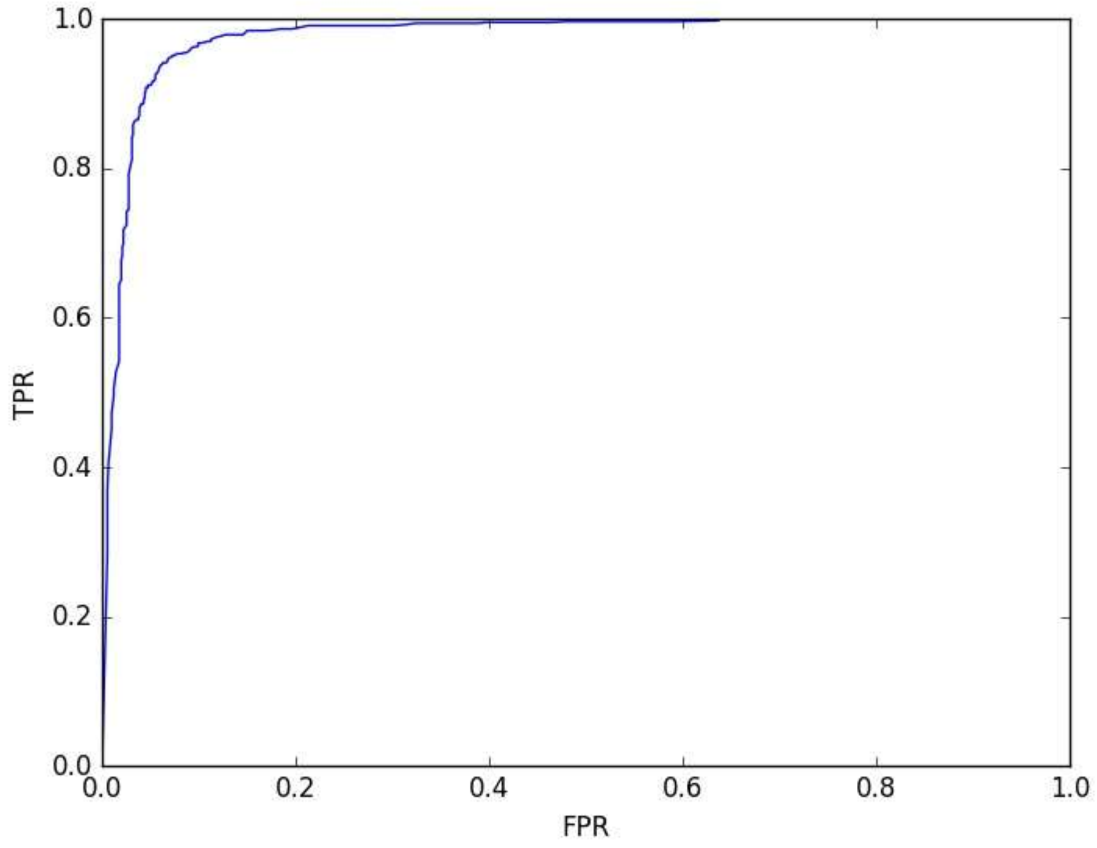

Figure 1. ROC Graph Experiment 1.

From the slides and lecture, this figure represents a very useless ROC curve. Since the area under the ROC curve represents the probability the SVM will choose positive over negative.

Experiment 2:

For experiment 2 in this project, the purpose was to test different styles of feature selections. Then to see if these feature selections had any impact on accuracy. I actually made a very interesting mistake when setting up my algorithm for this experiment. I originally use the argsort() function provided by the Python library. This would return the indices of an array from minimum to maximum, which was easily flipped to have the most important feature in the front as the experiment guidelines requested. What I did not realize originally was that the implementation of argsort considers the negative numbers to be smaller, and the experimented wanted the highest |w|, which could have been negative. This was solved with a very simple absolute function call to gain the correct values for weight vectors. Either way, with that I was able to compare these different values for the Top 5 features between the two implementations. The first set of five, is the incorrect implementation, or not using the weight vector, only the integer value.

Top 5 weight vectors for Experiment 2 Done Incorrectly: [55, 51, 52, 54, 23]

And the corresponding plot in the following figure.

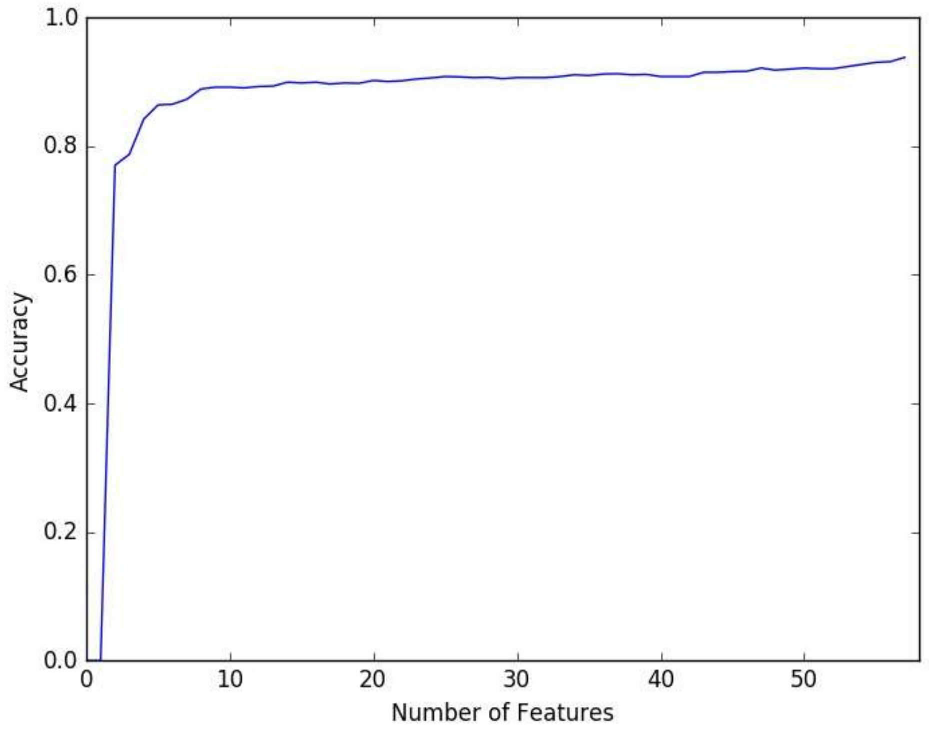

Figure 2. Experiment 2 Done Incorrectly Feature number and corresponding accuracies.

This figure shows Accuracy as a function of weights with only their integer values being considered. I only kept this because when I did the experiment correctly, I noticed that three of the features were the same as before, and the graph looked extremely similar.

The Experiment done correctly, using the weight vectors instead gave the following results for the Top 5 features.
Top 5 weight vectors for Experiment 2: [26, 24, 55, 51, 54]
The features 55, 51, and 54 appear in both of these experiments. These are obviously very large positive numbers for the weights. So they are obviously very important features for the SVM.
The following figure shows the plot of the number of features against accuracy again, this time done correctly as the project guidelines suggest.

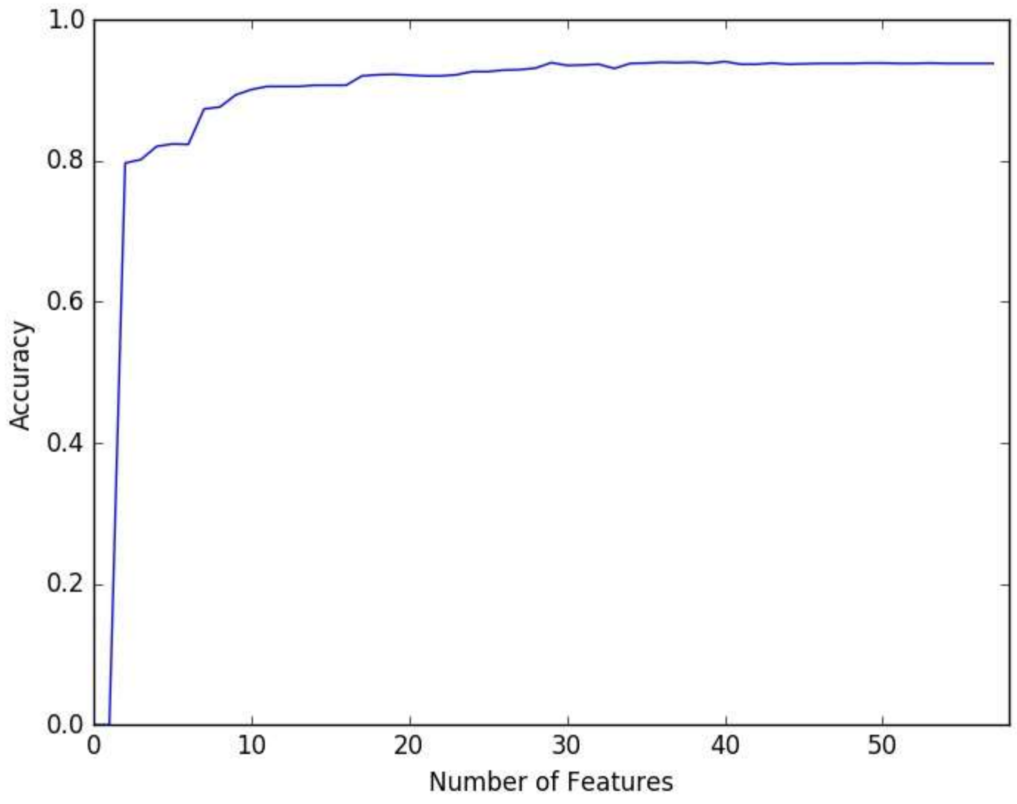

Figure 3. Experiment 2 Feature Number and Corresponding accuracies.

The two figures are extremely similar. If anything, during this iteration of my testing the incorrect graph actually seems to be better. But that is probably more just a random event, than concrete evidence suggesting it is better. The Top 5 features selected by importance were the following:

Feature 26: Word Frequency George.
Feature 24: Word Frequency 'hp'.
Feature 55: Capital letter run length for the longest string.
Feature 51: Character Frequency "!"
Feature 54: Capital letter run length for the average string.

The first feature is counting the frequency of the word George. I'm assuming that's a mixture of a common name, possibly referring to George Washington for the dollar bill, or just focusing in on a certain set of people. I really felt like that was a very random feature, and couldn't understand the difference between "George" and any other common male name. I was unable to find any real importance of the next feature for "hp". I'm going to assume it's because it's not really a word. Any none real word used over and over again in an email would definitely be a consideration for a spam filter. I just wasn't sure how or why it was set on that "hp" word in particular. I also noticed the feature for "hpl", and I wasn't sure why that was there either. Though it wasn't one of the type features. Skipping a feature, and going to 51 or the "!" makes perfect sense. An over use of that character shows someone who is clearly too excited or not a real person, trying to scam users. This feature definitely fits in for what I would consider a key towards spam identification. The last two features are for capital letter strings. Both average and longest were top features. These two make sense as well, since the spam I've seen in my email has a habit of being in all or mostly all capital letters. Those two features showed up in both sets of the top 5 features I made in experiment 2. The incorrect and correct version, so clearly they have a very high importance for this model.

Looking at Figure 3, showing the plot of feature selection against the accuracy of each SVM test, it shows a very clear and steady increase as more important features are added to the SVM. Though as more features are added, the model does not become much more accurate compared the first few additions of the more important features. The shows the little amount of importance these other features place on the model. It is possible with more testing data and more trials that these features could be helpful, but with the given data set, the lower priority features are not the most helpful when it comes to improving the accuracy of the model. In figure 3, there is a very large jump right around the addition of a 9th feature. And again around the 18th feature. That could just be a random event, but it is interesting to see those features play such a big part in improving accuracy for the model. But the greatest improvement to accuracy comes from the first roughly 20 features added into the model. After that, there is very little improvement overall.

Experiment 3:

Experiment three used a different style of feature selection. The indices of the features were randomized and added into the model one at a time. This was to see how much the weight vectors actually mattered for accuracies compared to just how many features were in the model. The following figure shows the random features added compared to their accuracies.

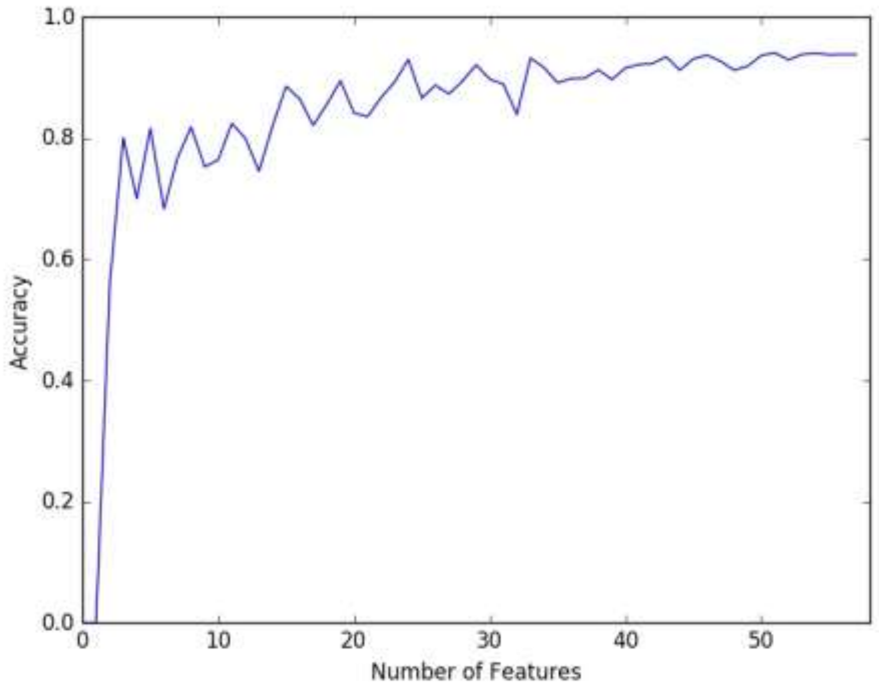

Figure 4. Experiment 3 Double Random Feature Number and Corresponding accuracies.

Looking at Figure 4 it is very different from the plot of weight vector feature selection. The initial accuracy is quite higher from this trial than seen in other attempts. But it is still varies with the accuracies as more features are added into the model. In my implementation of the function, I randomized the indices of the features to be added into the model, then I would randomize the model again. I did this because I wanted to make sure the added features were actually random every time. I was unsure if this was actually necessary for the experiment, so I ran it without the extra random call. So instead, I would only randomize the index array once, then added one feature at a time. This should still keep the feature selection random. I realized during my testing that my original implementation of randomizing twice could allow for the same feature to be added to the model more than once, I think. The next figure shows Experiment 3 ran with only randomizing the index array once, and I feel like this plot is more accurate for the experiment.

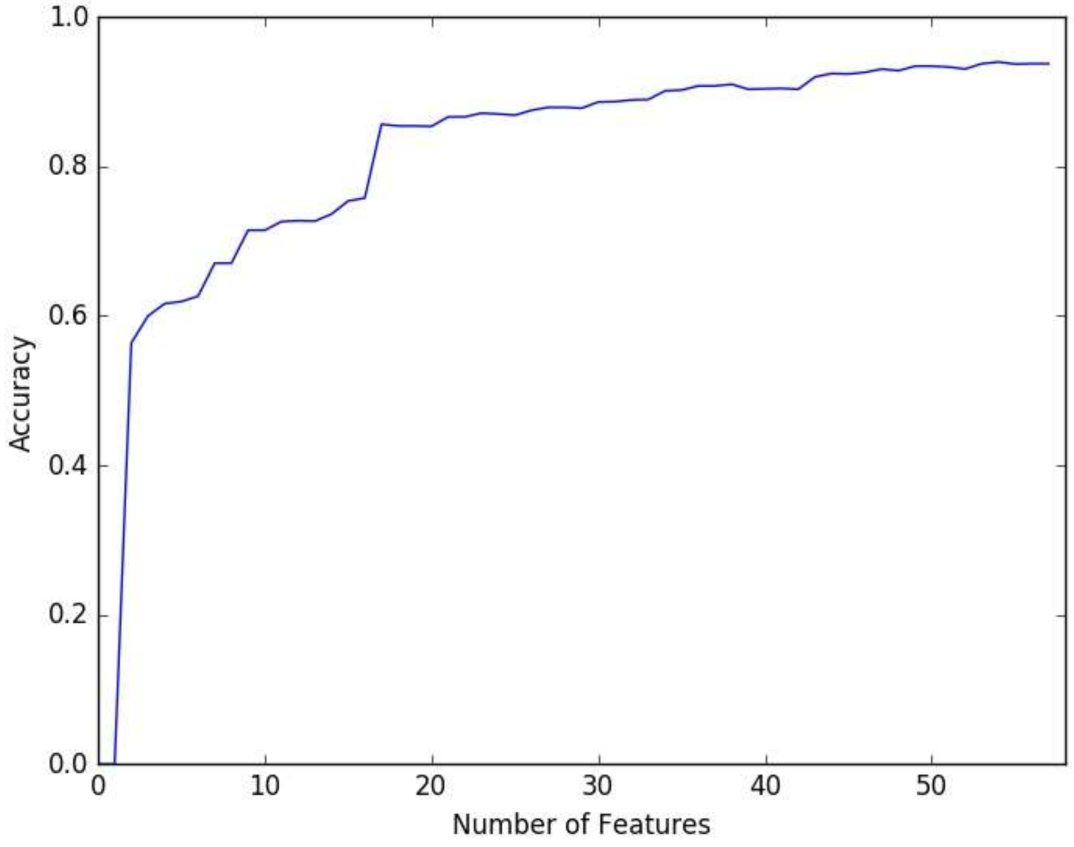

Figure 5. Experiment 3 Single Random Feature Number and Corresponding accuracies.

I believe this figure shows a more accurate representation of what a randomized feature selection should look like for the SVM. Right around the 19th feature added, there is a massive spike in the accuracy, which would suggest a higher importance feature was added into the model. But the slow gain of accuracy shows that the model is only becoming more accurate as each feature is added without any sort of real order to which is added. When a higher weight vector is added, the accuracy takes a larger jump like the 19th feature or the 8th feature. But the accuracy gain is a much slower process compared to the ordered weight vector from experiment 2. And the initial accuracy is much smaller as well. Which suggests that the first two features for the SVM were not really the most helpful of features. With experiment two, it immediately started at 80% because it used the highest two weight vectors. It would've been interesting to actual show which feature was added with this experiment. This way the actual importance of each feature could be tested with this randomized selection, and the impact on the model could be seen in a different way for each feature.

Obviously there were a lot of different ideas that would have been fun to test, but time was limited. The feature selection based on the weight vectors is obviously more useful for the SVM. Though, the varying values of C* but still having similar precision, accuracy, and recall was an interesting part of this project. It would be interesting to play with this program more and see what could happen.