Using Predictive Modeling to Recognize Human Activity

Abraham Do

Introduction:

The goal of this report is to implement methods of predictive modelling in order to not only find but compare performances between the training and test data to discover the best possible techniques through the discovery of its accuracies.

Given a large dataset, it can be tricky to come to the realization of which predictive model to use to identify previously unrecognized patterns within structured and unstructured data. For a prediction model to be valuable, it must not only have predictive ability in the derivation cohort but must also perform well in a validation cohort [1].

The HAR dataset is a perfect test case for this investigation as it provides a relatively small number of test subjects while maintaining a respectable amount of collected data on each of the individuals. Because the data provided by the smartphone is relative nonsense when viewed on the outside, by applying statistical and data mining techniques, the use of predictive modeling can find such relations with minimal background knowledge to predict the future outcomes of a given dataset.

By implementing the Random Forest algorithm along with the Support Vector Machine algorithms, we hope to show that there is a significant enough accuracy to comfortably state any relations within the data as well as see if there is truly a method of discovery that is substantially better than the other, and discover why that may be the case.

Methods:

Data Collection

For my own data analysis, I was lucky to use a dataset already provided by the UCI Machine Learning Repository [2]. The Human Activity Recognition using Smartphones dataset was provided by Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge L. Reyes-Ortiz.

The HAR dataset provided by the UCI Repository gives the following information:

- 30 volunteers between the ages of 19-48
- 6 different activities
    1. WALKING
    2. WALKING_UPSTAIRS
    3. WALKING_DOWNSTAIRS
    4. SITTING
    5. STANDING
    6. LAYING
- Recorded data of 3-axial linear acceleration and 3-axial angular velocity at 50Hz.
- Data discovered by placing a smartphone (Samsung Galaxy S II) on the waists of the subjects.

The data was already comprised within the DataSet_HAR.csv file. Within the dataset, there were two columns. One labeled "Activity" which represented the type of activity being performed and "Subject" which represented the individual who was performing the activity. Because the dataset has already been normalized, the other attributes are all in the same column data format.

Random Forest

Random Forests have the ability to use both classification and regression predictions to improve on instability and accuracy. They naturally handle categorical predictors, are simple to fit, handle non-linear interactions, perform automatic variable selection, and can handle missing values [3]. For a dataset such as the one given in the HAR project, it can be very clear to see why a Random Forest algorithm could be extremely beneficial. As we are attempting to use this wide range of data and find relations between the 561 features with the users who are performing the features, we need to use an algorithm that would be able to make these connections with ease. The Random Forest algorithm has the ability to make connections with seemingly non-linear data as well as automatically select the correct variables(features), which is a large aspect of what we would need in order to efficiently and easily find and describe such relations with as accurate a result as possible.

Furthermore, Random Forest algorithms provide us with the out-of-bag (OOB) estimate. Within a Random Forest, there is no need for cross-validation or a separate test set to get an unbiased estimate of the test set error. The OOB estimate is extremely beneficial because it allows us to get a <u>running</u>, <u>unbiased</u> estimate of the classification error as trees are added to the algorithm, as well as get proper estimates of variable importance [4].

For the second objective of the project, the OOB estimate can be very helpful to us to find a proper accuracy between certain selected features. As the estimate will continue as the algorithm runs until it runs through every iteration, we can see a clear picture of what to expect and exactly what accuracies coincide with each selected feature. This is possible because every tree is constructed through the use of different bootstrap samples in comparison to the original HAR dataset [4]. As the program finishes, the “j” iteration within the kth tree can be related to the class that received the most amount of votes during the moments in which “n” was the OOB.

Essentially, the OOB is a crucial part of the Random Forest algorithm which allows us to easily see the results of the algorithm without the need to stop and continuously reset variables as the estimate will actually iterate through every single node separately and return different results. In the end, this estimator makes the entire process a bit simpler to not only use, but collect data from.

Support Vector Machines

Support Vector Machines are a bit trickier conceptually, but are extremely beneficial when dealing with support vectors as they are usually the most difficult to classify. Support Vector Machines maximize the margins around the separating hyperplanes and are extremely effective in high dimensional spaces [5].

For the HAR dataset, the SVM algorithm could be beneficial as we are dealing with a large number of variables. The ideal outcome would be that the algorithm can take the training pair samples of the dataset and effectively output a set of weights that can predict the value of y through linear combination [5]. The SVM is helpful because as the algorithm is maximizing the margins, it is actually reducing the total number of weights that aren’t empty values to as few as possible as to separate the hyperplane into just the significant features. This allows for analyzation that is not only more accurate, but also memory efficient [6].

The one downside with Support Vector Machines is the fact that their compute and storage requirements increase rather quickly as the total number of training vectors increases. As a result, in our experiment we will have to find a way to minimize this effect as much as possible to still create an effective method of data collection while not sacrificing overall performance [6]. The obvious answer to this would be to limit the total number of vectors, but it will be interesting to see just how accurate our outcome will be as a result of this downfall.

Reproducibility

All analyses performed in this document are reproduced in the Jupyter file “Mini Project.ipynb” file provided with the final submission of the project.

Analysis/Results:

The overall analysis of the HAR dataset is divided into 3 different parts:

1. Random Forest w/ all 561 Features
2. Random Forest w/ Reduced Features
3. Support Vector Machines

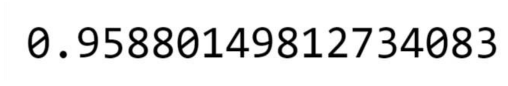
The Random Forest w/ Reduced Functions was an important step of the process as it allows us to see how the number of features provided to the algorithm could affect the overall accuracy of the dataset. We were trying to achieve ~80% and ~90% accuracy by altering the features selected, and this will be described in more detail following a thorough analysis of the first step.

<u>Random Forest w/ all 561 Features</u>

In order to determine the actual stability of our model, the OOB score was repeatedly calculated throughout the model’s building as to properly optimize and improve the accuracy of the dataset. Furthermore, to showcase a more accurate performance in the model I felt that it was best to create a separate testing set as a high accuracy on this set could show that the model was not overfitting, which would result in outcomes that are not satisfactory or off.

In order for the dataset to maintain consistent outcomes, I ensured to set the “random” variable to 0 so that all of the sets and features would not produce random results upon each run of the program. As we are working with data that is within a single CSV file, we separated the dataset into two different sets: a training set and a test set. The data was split 70%/30%, respectively, as to ensure that the data was properly partitioned according to the following statement in the Readme for the set: “The obtained dataset has been randomly partitioned into two sets, where 70% of the volunteers was selected for generating the training data and 30% the test data.” It was important to keep this consistency so that our results would not be affected by improperly partitioned data.

From this point on, we had to use the partitioned training set along with all 561 features and implement our Random Forest algorithm for the first time. With all the features present, this is the resulting accuracy given:



```
0.95880149812734083
```

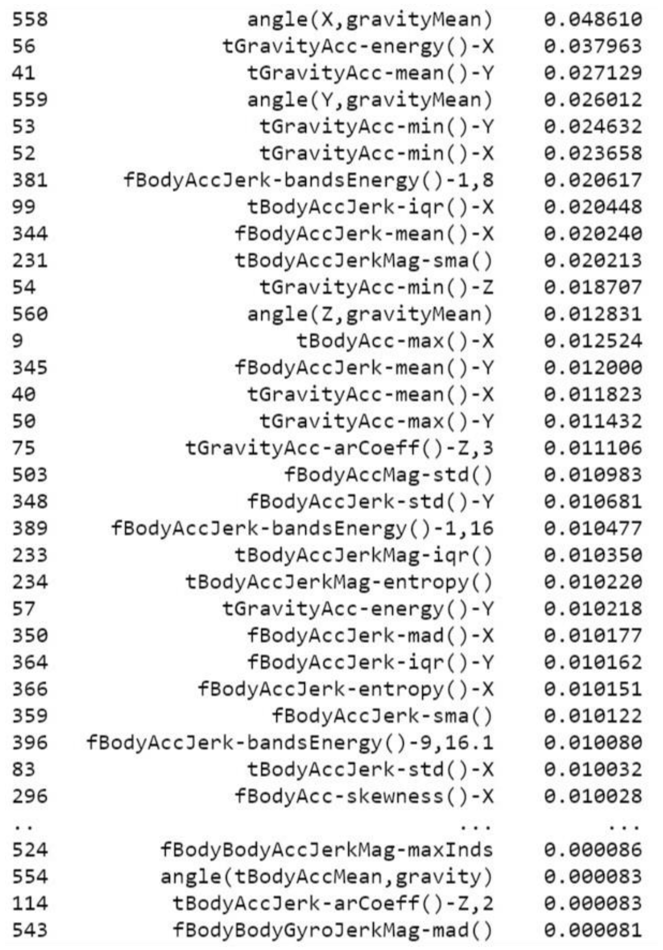
The high accuracy with the Random Forest algorithm is unsurprising as this particular algorithm tends to excel when given more data as it is able to connect more relations. Furthermore, the Random Forest algorithm does not overfit training data. This along with the fact that the training set is rather balanced and unbiased shows that this algorithm would be perfect to use with a dataset such as the HAR set.

<u>Random Forest w/ Reduced Features</u>

This particular step of the project was more interesting as it shows just how significant data can be in creating different levels of accuracy. There are several ways to approach this situation, but I personally wanted to work with features that had more importance in differentiating between certain categories and the users associated with them.

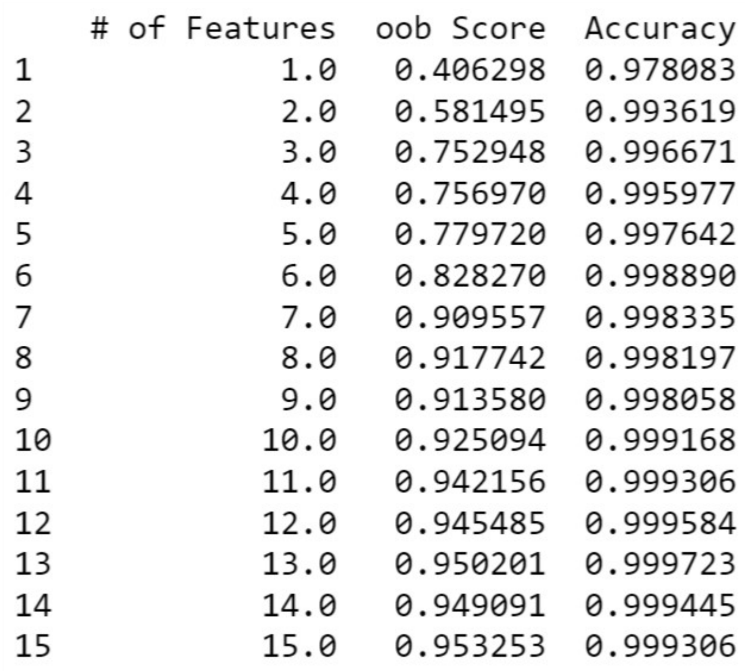
The reason that I decided to take this approach was because I felt that in a realistic test, it would be unnecessary to use data that provided little to no actual impact as it would just use up more memory. Granted, this may not be the case with all tests but for our example, I felt that it was something that would output a much better representation of the effects of the dataset.

In order to accomplish this, I took the features that were generated from the analysis of the 561 features and ranked them based off of their overall importances. The following are the results given from this analysis:


```
558                    angle(X,gravityMean)    0.048610
56                  tGravityAcc-energy()-X    0.037963
41                    tGravityAcc-mean()-Y    0.027129
559                    angle(Y,gravityMean)    0.026012
53                     tGravityAcc-min()-Y    0.024632
52                     tGravityAcc-min()-X    0.023658
381       fBodyAccJerk-bandsEnergy()-1,8    0.020617
99                    tBodyAccJerk-iqr()-X    0.020448
344                  fBodyAccJerk-mean()-X    0.020240
231                  tBodyAccJerkMag-sma()    0.020213
54                     tGravityAcc-min()-Z    0.018707
560                    angle(Z,gravityMean)    0.012831
9                          tBodyAcc-max()-X    0.012524
345                  fBodyAccJerk-mean()-Y    0.012000
40                    tGravityAcc-mean()-X    0.011823
50                     tGravityAcc-max()-Y    0.011432
75              tGravityAcc-arCoeff()-Z,3    0.011106
503                      fBodyAccMag-std()    0.010983
348                   fBodyAccJerk-std()-Y    0.010681
389      fBodyAccJerk-bandsEnergy()-1,16    0.010477
233                  tBodyAccJerkMag-iqr()    0.010350
234              tBodyAccJerkMag-entropy()    0.010220
57                  tGravityAcc-energy()-Y    0.010218
350                   fBodyAccJerk-mad()-X    0.010177
364                   fBodyAccJerk-iqr()-Y    0.010162
366               fBodyAccJerk-entropy()-X    0.010151
359                     fBodyAccJerk-sma()    0.010122
396    fBodyAccJerk-bandsEnergy()-9,16.1    0.010080
83                    tBodyAccJerk-std()-X    0.010032
296                  fBodyAcc-skewness()-X    0.010028
..                                     ...         ...
524           fBodyBodyAccJerkMag-maxInds    0.000086
554           angle(tBodyAccMean,gravity)    0.000083
114             tBodyAccJerk-arCoeff()-Z,2    0.000083
543             fBodyBodyGyroJerkMag-mad()    0.000081
```

As seen from the results, it is evident that each iteration of the features contains a rather significant drop-offs of importance. This is exactly the result that I wanted to see as I wanted to ensure that the difference was large enough to justify using the points with the top importances instead of iterating through every single point to create a more efficient algorithm.

Once this was resolved, I proceeded to further optimize my algorithm with the number of trees and features to get as close to the 80% and 90% thresholds as possible. I ended up choosing 15 features to see how the iteration process went with the OOB estimations, and the results were pleasantly surprising:



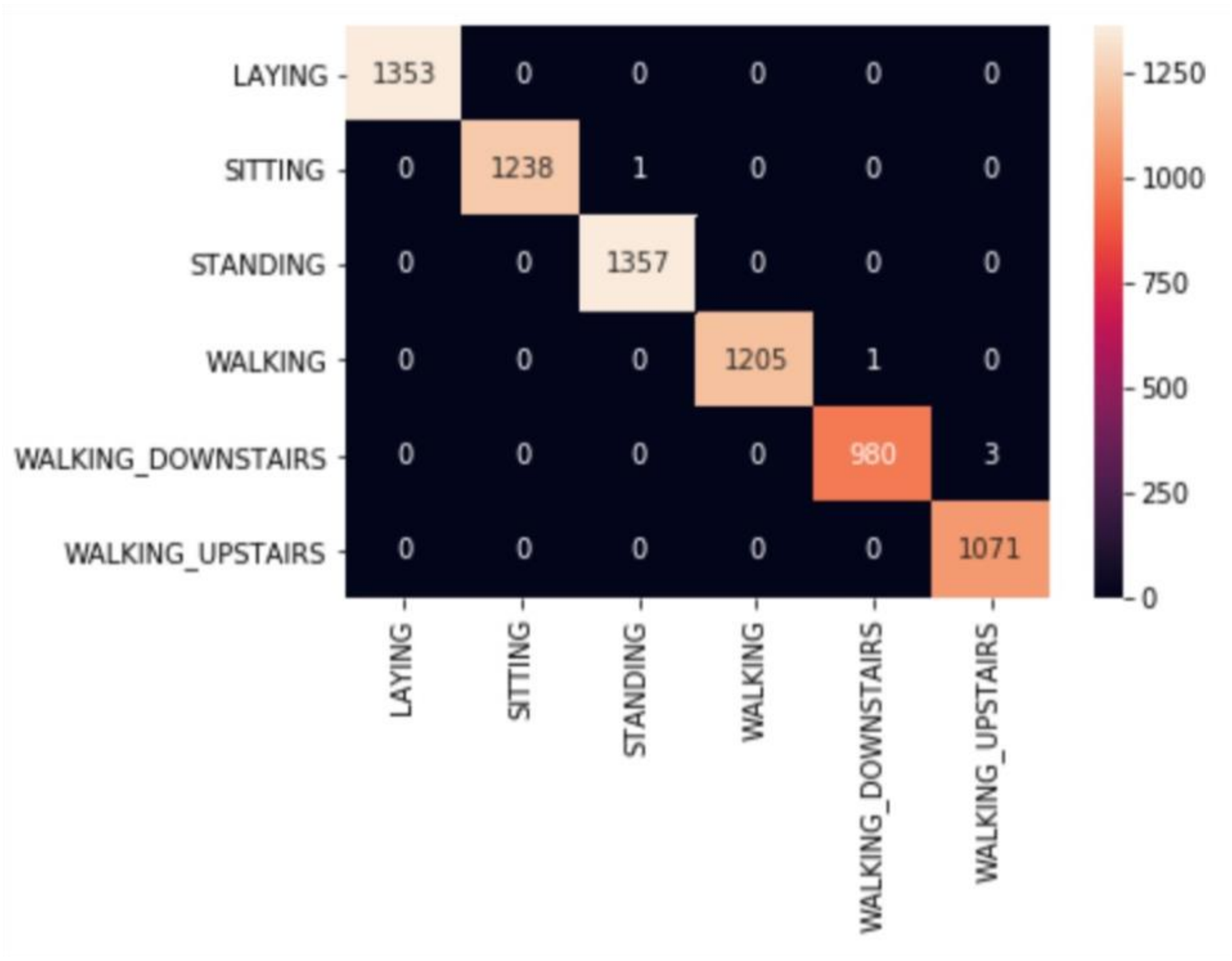
```
    # of Features  oob Score  Accuracy
1             1.0   0.406298  0.978083
2             2.0   0.581495  0.993619
3             3.0   0.752948  0.996671
4             4.0   0.756970  0.995977
5             5.0   0.779720  0.997642
6             6.0   0.828270  0.998890
7             7.0   0.909557  0.998335
8             8.0   0.917742  0.998197
9             9.0   0.913580  0.998058
10           10.0   0.925094  0.999168
11           11.0   0.942156  0.999306
12           12.0   0.945485  0.999584
13           13.0   0.950201  0.999723
14           14.0   0.949091  0.999445
15           15.0   0.953253  0.999306
```

The OOB estimator allows us to see each iteration of the algorithm from the top features. As expected, the OOB Scores for the first few iterations are extremely low as a single feature cannot be expected to accurately represent the relations between the points. However, it only took 6 iterations for us to get to within 2-3 percent of our desired 80% threshold, and it took 7 iterations to get within 1-2 percent of our desired 90% threshold.

If we were to have iterated through the data normally, it would have undoubtedly taken many more features to get to our desired scores of accuracies. Because we took the extra step to rank the features accordingly, we were able to achieve both thresholds while maintaining a very low memory usage.

While this may not be absolutely necessary as we only have 561 features in this particular dataset, I wanted to showcase how beneficial this method of approach could be once you start dealing with situations in which there are thousands of points that you have to deal with. Selecting features based off of their importances can help immensely in the long run, and it is rather impressive to see that it only took our algorithm 6 steps to reach an 80% accuracy and 7 steps to cross the 90% threshold.

The following confusion matrix further shows how accurate and consistent the results of the Random Forest algorithm are:



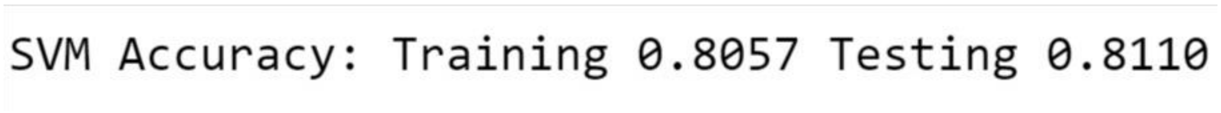
As evident from the above figure, the Random Forest classifier has little to no noise outside of the linear results, showcasing that this particular algorithm is extremely efficient in identifying the connections between a user and its activities.

Support Vector Machines

Because of Python's accessibility, the implementation of the Support Vector Machine was fairly similar to that of the Random Forest classifier. To begin, I set the "random" variable to 0 once again to ensure that there were no random results upon each run of the algorithm.

Once again, I decided to iterate this particular dataset with the sets of the highest importances. Whereas this process was optional above, I felt that this step was a bit more necessary for the SVM model as I wanted to ensure that we were using as little of the memory process as possible. Furthermore, the SVM model tends to take up more processing power as the number of vectors increase as states in the original description of the methods.

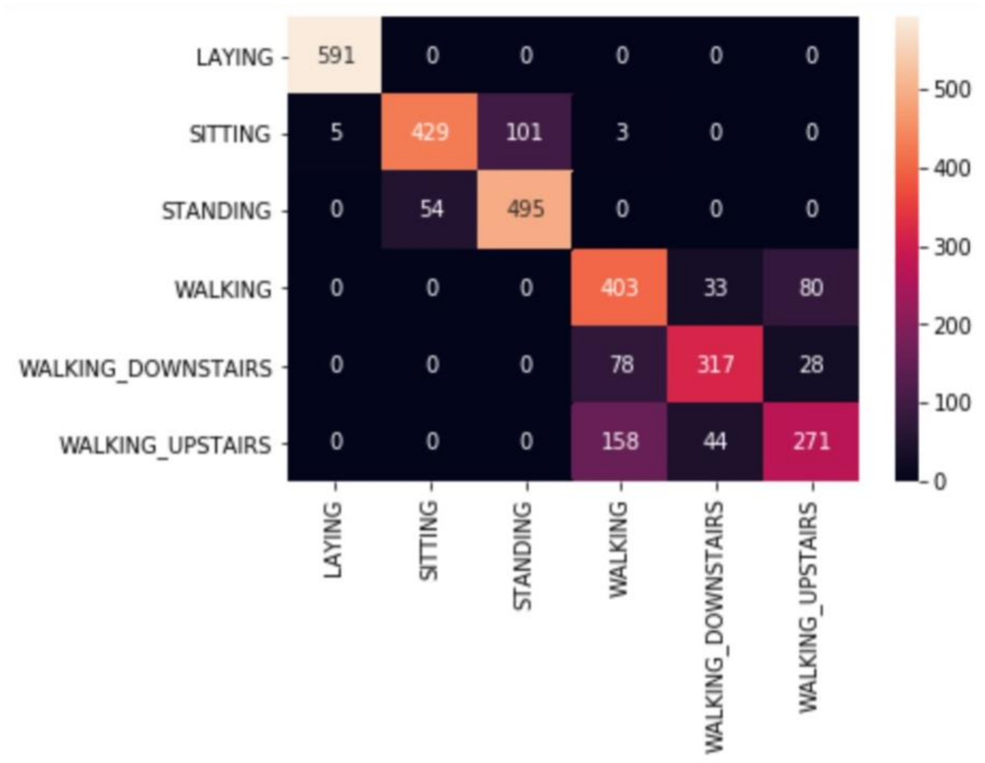
With this in mind, these are the accuracy scores of the training and testing points with SVM:



```
SVM Accuracy: Training 0.8057 Testing 0.8110
```

My immediate reaction was that the accuracy scores were much lower than those of the Random Forest classifier. This could be for several reasons, but I believe that the biggest factor is because of the somewhat non-linear interactions of the HAR dataset.

The following confusion matrix also showcases the algorithm’s inconsistencies:



The confusion matrix shows that this particular algorithm is not nearly as linear as the Random Forest algorithm. In fact, there seems to be a significant more amount of clustering as the relations take shape, which is causing the lower accuracy.

While an 80.57% accuracy for the training data is not too low, it is most definitely lower than the 95.88% accuracy found from the original analyzation of the 561 data points from the Random Forest algorithm.

Conclusions:

By the end of the data analysis, we were able to come up with a few solutions. Firstly, the Random Forest classifier had a substantial increase in accuracy in comparison to the SVM predictive model. While there could be several different reason for this occurrence, it does help my case that certain predictive models are simply better for particular datasets. The Random Forest classifier was expected to perform better than the SVM algorithm, but the difference was certainly more than I had anticipated.

Secondly, the effect of iterating a dataset through its importances showcases how it can be beneficial to take more of an algorithmic approach to predictive modelling by looking at the OOB estimations to help determine the most optimal amount of features to include. By doing this, you could possibly save a significant amount of processing memory and power once you start dealing with much larger datasets.

Once again, the results show that we can be very confident in using a Random Forest algorithm for a dataset such as the HAR set. The original 95.88% accuracy in the first test showcases the model's efficiency in handling such data, and it is most definitely a better approach than the SVM model.

The one issue that I had with this assignment was trying to figure out the best approach to select certain features to use for the second objective. While I did have some issues figuring out how to properly do this, I wanted to do everything I could to make the process as efficient as possible. Eventually I did come up with the solution of iterating the points through the top importances but it took a significant amount of time to get to that point.

One task I do wish I could try in the future is to see how other approaches would work on this dataset. Because I was more focused on the report and trying to get everything to work properly, I do feel like I missed out on the opportunity to see how accurately I could get results. The 95.88% accuracy from the Random Forest algorithm is definitely higher than what I original anticipated, but that does make me wonder if there is a different method that would produce an even higher level of accuracy.

References

1. NBCI – A Primer on Predictive Models. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3912317/
2. UCI Machine Learning Repository – Human Activity Recognition using Smartphones. URL: https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones
3. Utah State University – A Comparison of R, SAS, and Python Implementations of Random Forests. URL: https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=2295&context=gradreports
4. Berkeley (Salford Systems) – Random Forests. URL: https://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm
5. MIT - An Idiot's guide to Support vector machines (SVMs). URL: http://web.mit.edu/6.034/wwwbob/svm.pdf
6. Scikit Learn – Support Vector Machines. URL: https://scikit-learn.org/stable/modules/svm.html