Rob Lindsay **Assignment 2** **Machine Learning 6341**

EXECUTIVE SUMMARY

Data

This report serves to explain how two classification techniques, Support Vector Machines and Decisions Trees, were used to develop classification models. The data that was used was found in the SeoulBikeData.csv. The data contains hourly information on environment variables as well as seasonal and holiday data. The number of bikes that were rented over the course of each hour is what we will derive the classification target off of.

With the exception of the initial challenges faced with special characters in the column titles and the column titles not aligning with the data, both of which were addressed in Assignment 1, the data was exceptionally clean. Using Similar data manipulation techniques from the first assignment, target encoding was used over one hot encoding for the SVM model. The decision tree however required the use of one hot encoding and will be further discussed in that section.

Classification Target

For the classification model, I decided to go with the median value of the rented bikes for my train set. Having a balanced classification would allow me to choose Accuracy as my indication for how well each classification model was performing. Using the median as a target value reduced my concern about making sure that there was a representative distribution in my train and test sets as well, although this can easily be accomplished throughout the code. At the end of the report, I discuss how choosing a different target value may affect how we approach determining a "best" classification model and how a business scenario could dictate the need for a more flexible target approach.

Support Vector Machines

I chose to implement SVM's with three different kernels: Linear, Poly, and RBF. Taking advantage of the GridSearchCV capabilities, I was able to use hyperparameters to fine tune my models. I was able to utilize various visualization tools such as the ROC curve, heatmaps, and 3D functionality to assess the performance and the cost of increased performance. What I found was that SVM comes with a very high cost in regards to time. Adding GridSearchCV functionality takes considerable processing time. I tried to expand this even further and test for various CV fold values outside of the standard 5 and 10 fold approaches, but quickly realized that this would take way too long and provided little benefit to my classification model

Decision Trees

After completing the SVM analysis, my original intent was to continue to use the similar visualizations as my SVM approach to help fine tune my model. However, as I introduced Cross Validation into the model, I found that one of my parameters, ccp_alpha, was not having any effect. Upon investing this, I realized that that the Decision tree had a better visual approach that could be found through the Cost Complexity Pruning Path. By utilizing this, I discovered that I could fine tune the ccp_alpha variable which had the added benefit of self-pruning the tree. This served as the biggest takeaway for me for this project.

SUPPORT VECTOR MACHINE CLASSIFICATION MODELS

Out of the box testing:

To establish a starting point, I ran SVM models using Linear, Poly, and RBF kernels without any parameters other than setting the random state. The training set contained 70% of the available data and the rest was used for the test set. The accuracy results were pretty good out of the box, with both Poly and SVM performing above 91%, and the more restricted Linear model performing near 88%. AUC from all models was between .95 and .97 as seen in Figure 1. As you can see, all the curves are pretty close to the upper left corner and it is already difficult to differentiate between the different models, especially the Poly and RBF models. Using ROC curves for visual reference on improvement is going to be difficult when the curves are already so alike and near the upper left corner.

GridSearchCV:

Having established baseline models, I began to see if I could improve upon them by utilizing the parameters and introducing Cross Fold validation. I made use of GridSearchCV for each model in order to easily manage my hyperparameters. For the Linear model, I chose C scores that ranged from 10^{-3} to 10^{4} with 1 unit increments in the power of ten. I used the same C settings for my RBF model as well as running gamma from 10^{-4} to 10, again incrementing by powers of 10. For the Poly model, I ran the degree setting from 1 to 11.

Results:

- Linear - accuracy results for the training and testing set still around 88%.
- Poly - training accuracy under 91%, increased to over 92.1% on test data.
- RBF - training accuracy at 91% and the highest test accuracy at 92.7%.

As you can see in figure 2, the visual differences between the different ROC curves is not very distinguishable. If we want to take advantage of visualization, other aids will be needed. To get a better feel for where each model was performing outside of just the top performing parameters, I created a heat map for each hyperparameter setting. As you can see in Figure 3, there are hot spots that seem to allow for better areas. These could be used to further tune the Linear and RBF models if desired. In particular, the RBF heatmap shows that there are ranges for gamma and C that could be explored

Figure 1: Out of The Box ROC Results

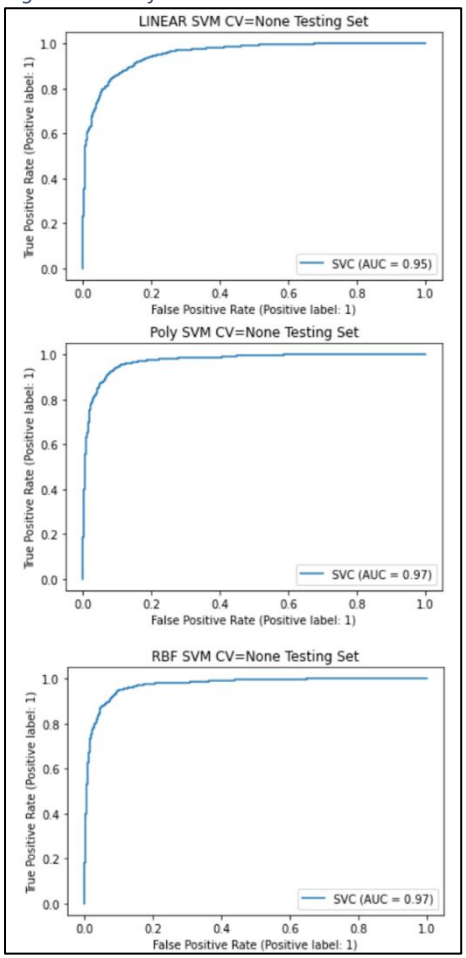


Figure 2: CV = 5-fold

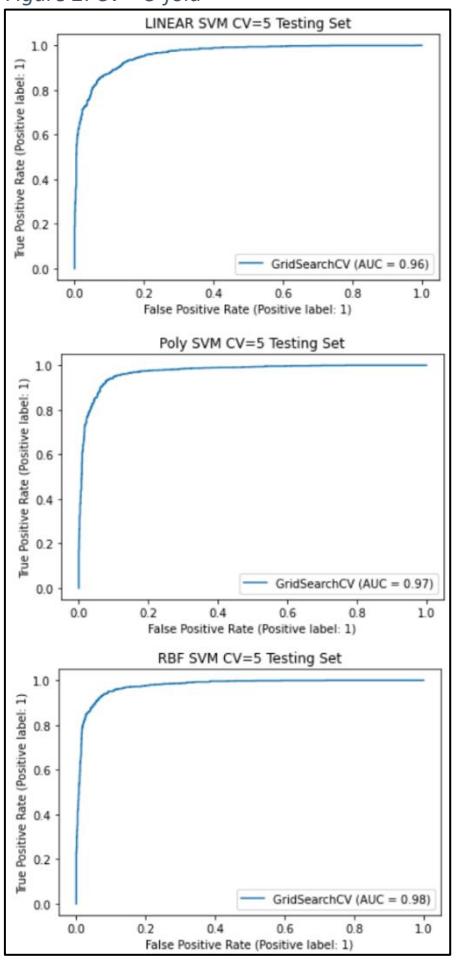


Figure 3: Heatmaps for CV = 5-fold

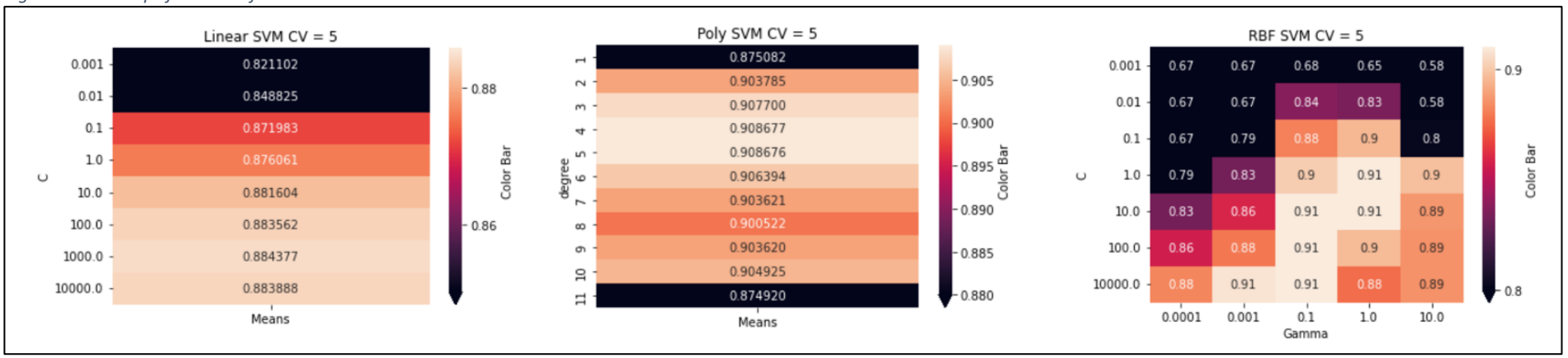
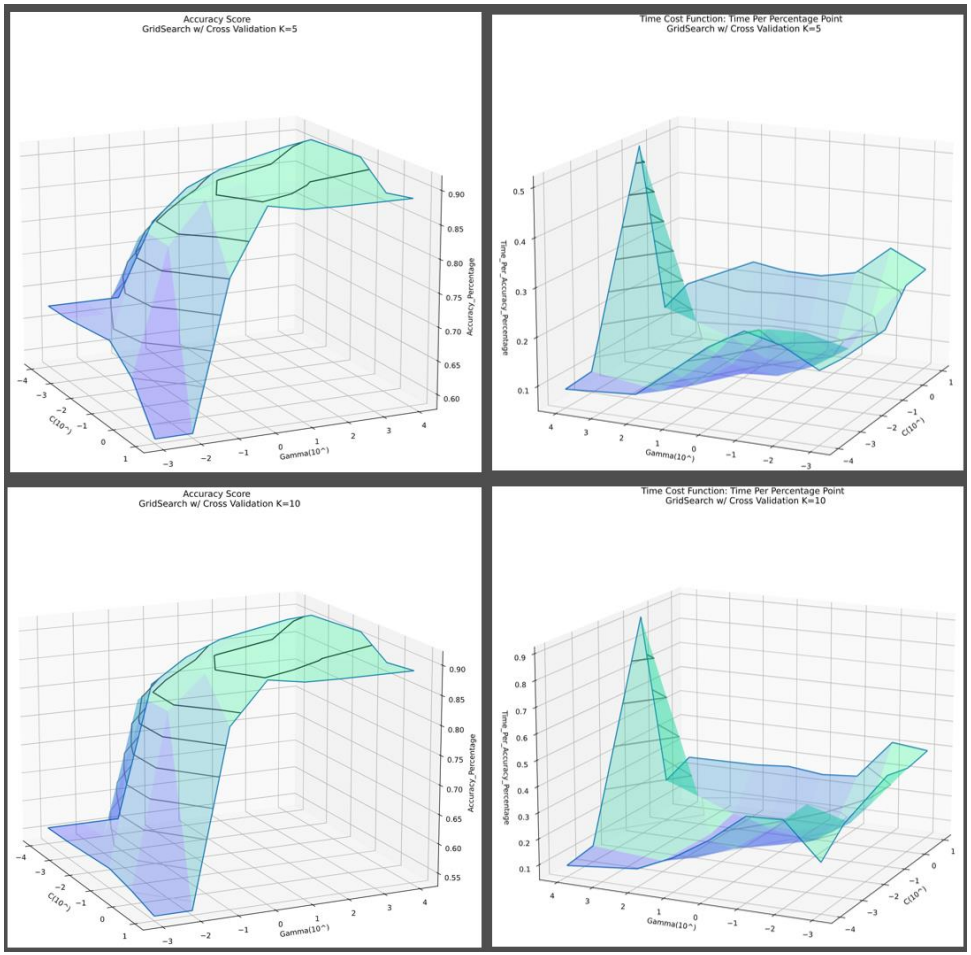


Figure 4: Accuracy and Cost Functions for 5 fold and 10 fold RBF SVM



Time is Expensive:

After running the three cross validation models, it was clear that cross validation was computationally expensive. In order to see just how expensive it was, I created a visualization that allowed me to visually compare the accuracy to the computation time. I used the results from the GridSearchCV to create a new calculated value that encoded the Time Per Percentage Point. The higher the value, the longer the algorithm would take. I then mapped both the accuracy spreads across the C and Gamma settings, as well as my new cost function based off Time Per Percentage Point. We will touch on this more in a moment.

Adjusting the three models to incorporate 10 fold cross validation yielded subtle changes from the results had in our 5 fold models. The best parameter for C in our Linear model went from 1000 to 10000, the Poly model shifted from 4 degrees to 5 degrees, and the RBF model shifted from {C:100, Gamma:.1} to {C:10, Gamma:1}. From the heat maps generated in our 5 fold cross validations, none of these shifts are shocking. However, the only test set improvement came from the Linear model but was minimal. The others two had a slight drop in testing set accuracy. However, the computational speed was considerably longer. As we see in figure 4, increasing the cross validation from 5 fold to 10 fold almost doubled our cost function values with very little change to the accuracy curve. The highest cost incurred for K=5 was around .5. On the other hand, K=10 incurred a value around .9. It is important to note that cost function varies from one run to the next, but these numbers are consistent with my experience over several iterations.

Further Testing:

Adjusting the target to the mean of the testing set, an increase of about 200 bikes, does not change the results by much. Our testing results did show a decrease of about 1% across the board for all variations of the models. For example, the final RBF testing accuracy value came in at 91.7% with Precision being the only metric that fell slightly below a 90% threshold. Interestingly, figure 15 shows the cost curve reduced for both he 5 and 10 fold cross validations.

Figure 16 shows the cost function results when resetting the target to the median value of the training set, but this time having that training set be split on a 60/40 ratio, produced provided not just cost savings in regards to the time per % point for accuracy, but also reduced the overall time as there were fewer observations to fit (the mean fitting time is larger than the mean scoring time). We continue to drop in accuracy, but we are still above 90% for both 5 and 10-fold cross validations on Poly and RBF. If for some reason we wanted to update this model on a daily or weekly basis, I would choose to sacrifice the percentage point and use a 60/40 split while running a condensed GridSearchCV over the known best producers in order to reduce the total training time and still feel confident that I was getting superior results.

Figure 15: Cost Functions for 5 fold and 10 fold RBF SVM with Mean value Target

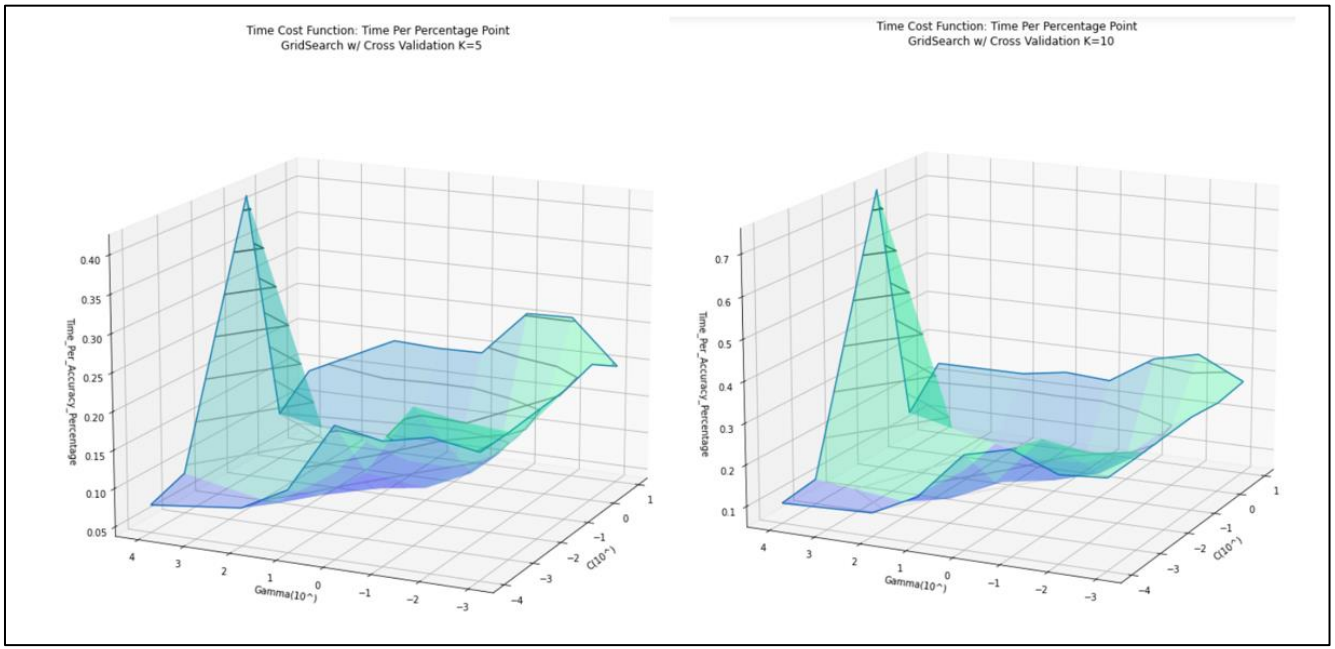
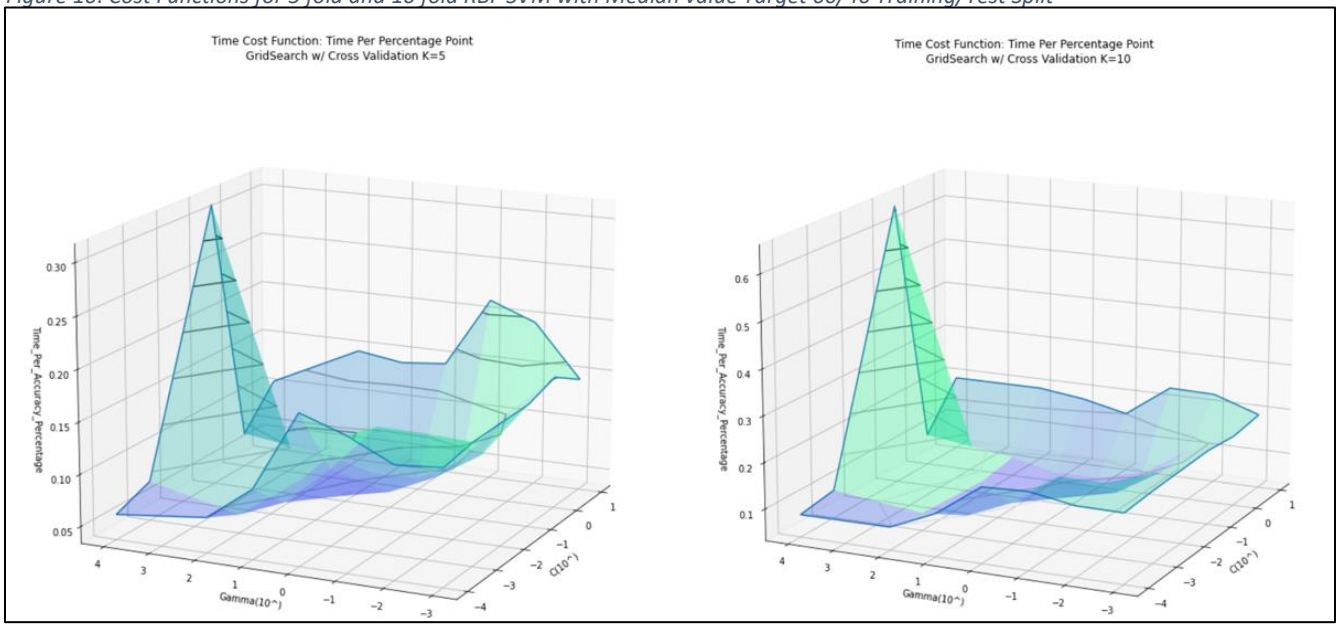


Figure 16: Cost Functions for 5 fold and 10 fold RBF SVM with Median value Target 60/40 Training/Test Split



DECISION TREE CLASSIFICATION MODELS

DATA PREPERATION

The first thing to consider was whether the data was in a format that was optimal for the Decision Tree. This did not just include if the right data was included, but also how we could interpret the results of the Decision Tree. The data that was used for the SVM algorithm was the same format as the normalized data created for the linear regression assignment. One advantage of the decision tree is that it can provide clarity as to how decisions are being made. With normalized data, this advantage is severely weekend.

So I decided to reload the data removing normalization techniques. I also went with one-hot encoding for the seasons, including all four seasons. The reason I chose to go with all four seasons is because it can result in shallower trees. For instance, if Autumn had been left out, what we will later see is the fifth most important feature of our tree, the tree would have to first test the other three seasons before coming to a conclusion that all three seasons as "0" resulted in a significant contribution.

The other two major variable changes came in the form of choosing to switch PeakHours out for Hours and switching out DOWAdjust for its nonnormalized version of DayOfWeek. The PeakHours decision was made because we no longer needed to take advantage of the PeakHours benefit for regression as it served to indicate a variation from the linearity that the other hours in the day displayed. However, I suspected that there could be benefit in allowing the tree to decided if a split should be performed on Hour or HourRank. Ultimately, Hour would play a factor in the final tree, however, further pruning would probably eliminate the need for having two variables. Further consideration could be made into the interpretability of having both forms of representation and if it overcomplicates the tree for a user, but if the goal is purely to find the best performing tree, then including does no harm.

TWO SEPARATE APPROACHES – Gridsearch vs Cost Complexity Pruning

I chose to use the DecisionTreeClassifier in sklearn. My initial thought was that I could follow a similar process as what was performed with the SVM algorithm and utilize ROC curves that were built off a Gridsearch approach. The results that we will discuss ended up pushing me in a totally different direction though and I began investigating Cost Complexity Pruning and ultimately decided that this was worth implementing.

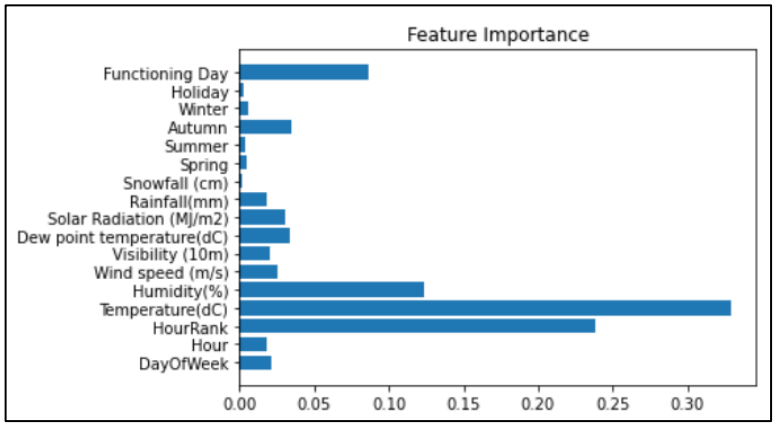
GRIDSEARCH APPROACH

Running the out of the box classification with nothing except for the random_state control in place provided the baseline results with an impressive Test Score of

Test Score : 0.91324200913242

Our most important features as seen in Figure 5 are Temperature, HourRank, Functioning Day, and Humidity with several more intermediate importance features and several that barely contribute to the model. However, the decision tree, whose shape is seen in Figure 6, would be impossible for any user to quickly navigate to understand the drivers of the decisions that are being made.

Figure 5: Feature Importance for Out of the Box Decision Tree



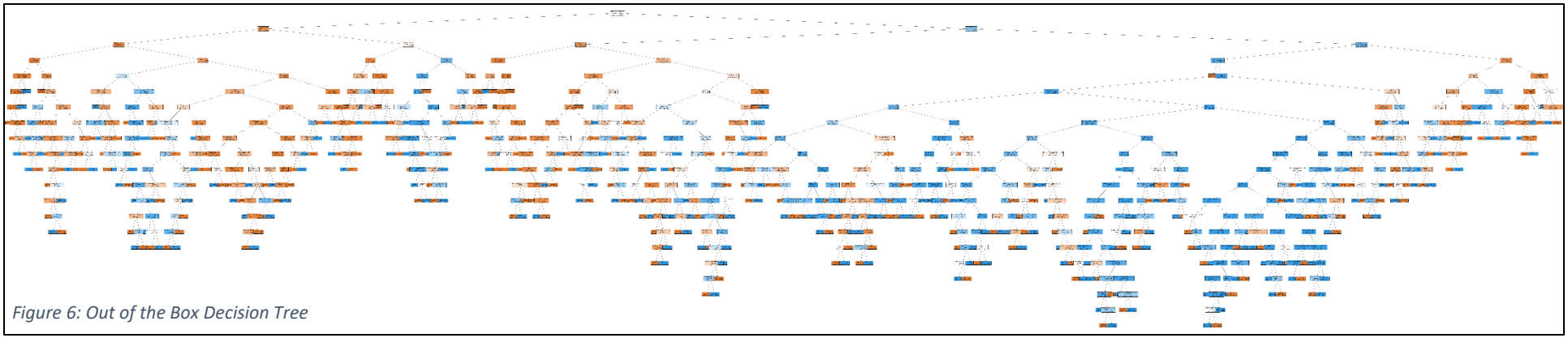


Figure 6: Out of the Box Decision Tree

CV = 5

I then began investigating how changing the parameters for the decision tree would allow me to optimize my tree. Returning to the GridSearchCV approach I employed with the SVM models, I used max_depth, min_samples_split, max_leaf_nodes, criterion, and ccp_alpha as my hyperparameters. First using a CV of 5 folds, I began investigating how the different values interacted with each other and what their results looked like. The training and testing time was nowhere near as long as the SVM models, so I was not as concerned with the time cost function for the decision tree. To help frame things, I plotted a histogram of my results, focusing on Criterion, Depth, and Leaf Nodes. These were all hyperparameters that were relatively easy to control and as the plots below show, lead to the conclusion that the default Gini criterion was likely the better performer and unsurprisingly, more leaves and deeper trees provided better results.

Figure 7: Gini vs Entropy , CV = 5

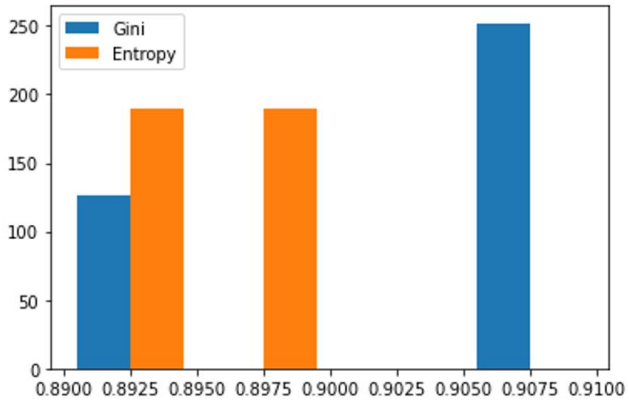


Figure 8: Distribution of Depth tuning , CV = 5

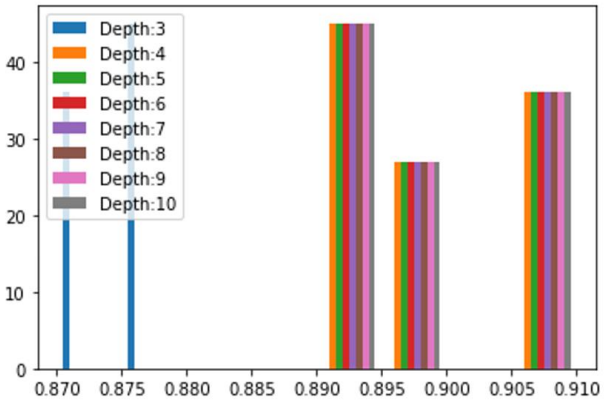
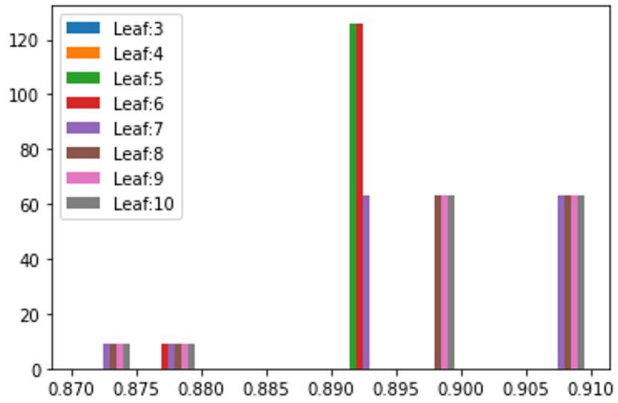


Figure 9: Distribution of Leaf tuning , CV = 5



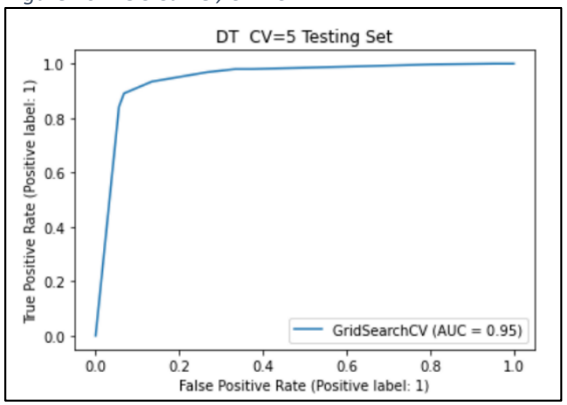
The more difficult criterion to tune was the ccp_alpha. I did not have an idea of where to start and the values that I choose for my gridsearch [0,.01,.02,.03,.04], all showed that 0 was my best option when run through the GridSearchCV.

The resulting tree from my the Gridsearch with CV =5 had the following parameter values:

{'criterion': 'gini', 'max_depth': 5, 'max_leaf_nodes': 10, 'min_samples_split': 2}.

The test score outperformed the train score with a value of 91.17% vs 90.79% but this was a lower performance than what we were able to achieve from the out of the box tree. Figure 10 shows the ROC curve but I didn't feel like this curve was adding much value to my hyperparameter tuning.

Figure 10: ROC Curve , CV = 5



CV = 10

As I moved on to 10 fold cross validation, I began to experiment with much larger values for min_sample_split and max_leaf_nodes. Additionally, I reduced my ccp_alpha all the way to .005 but it still seemed to have no effect on my gridsearch results. My efforts did however lead to a best tree with the following parameter values:

{ccp_alpha=0, max_depth=9, max_leaf_nodes=40, min_samples_split=10, random_state=42}

Both the train and test scores improved to 92.09% and 92.047%. However, I still had not identified a ccp_alpha that would improve the tree. Trying to figure out why led me to start over with a new approach.

ALPHA ONLY APPROACH

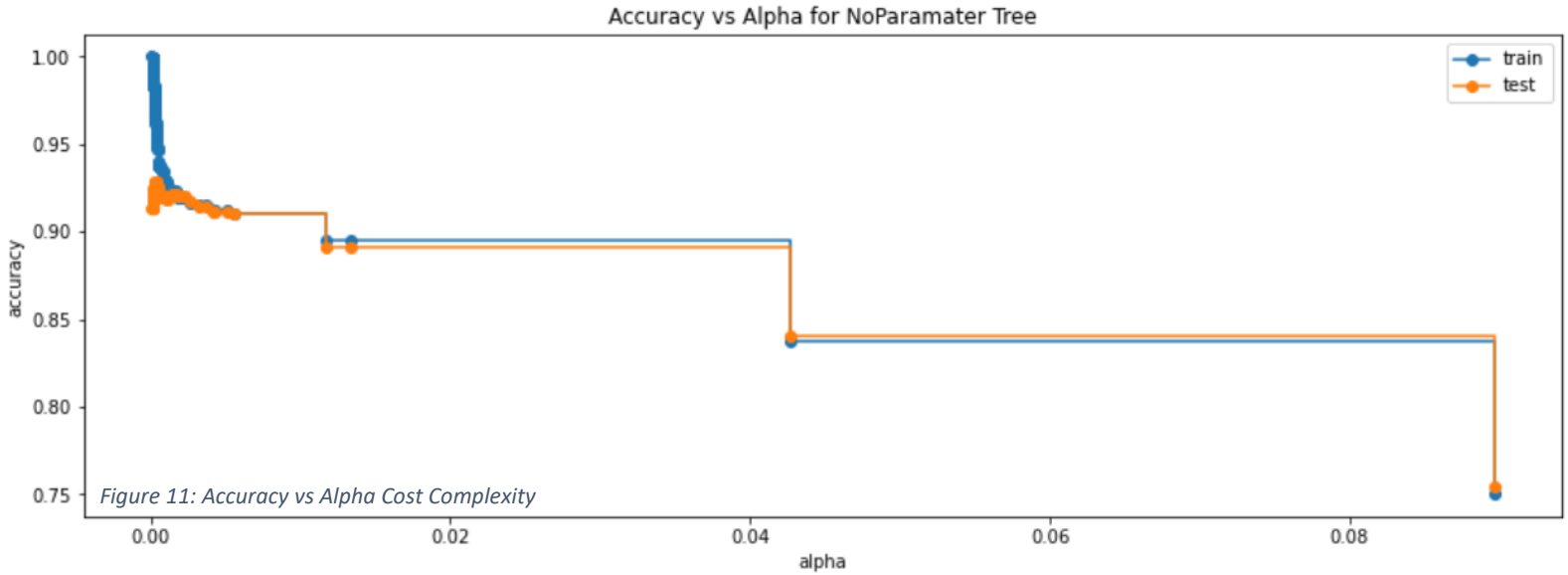


Figure 11: Accuracy vs Alpha Cost Complexity

After doing some research on the ccp_alpha parameter, I discovered that it provided several benefits to building out a decision tree. The first was that it was a self-pruning approach that reduced the size of the tree in a scientific manner that reduced the chances of overfitting. The other benefit was that it provided a better way of looking at a cost curve for the purpose of Decision Trees by utilizing the cost_complexity_pruning_path feature. I decided to applying it to my base decision tree and quickly realized why my ccp_alphas kept returning zero in my GridSearchCV results: The best alphas were below my lowest setting of .005 and were more likely in the range of .0005 as seen in the zoomed in chart in Figure 12.

I could now shift to a cross validation approach that prioritized maximizing my accuracy by manipulating the ccp_alpha. Without adding any other parameters besides my ccp_alphas that I had already generated, I reran my decision tree within a loop and calculated the training scores with 5-fold cross validation. This led to me being able to identify that my top ccp_alpha was .0004375260954892027 4. This can be seen in the figure 12 below as well as a visual representation of the Standard Deviation produced from the 5-fold CV. The Training data produced an accuracy of .9186 but the Test Data actually performed better, producing at .925 accuracy. The feature importance for our pruned tree is found in figure 13.

Figure 12: Accuracy vs Alpha Cost Complexity

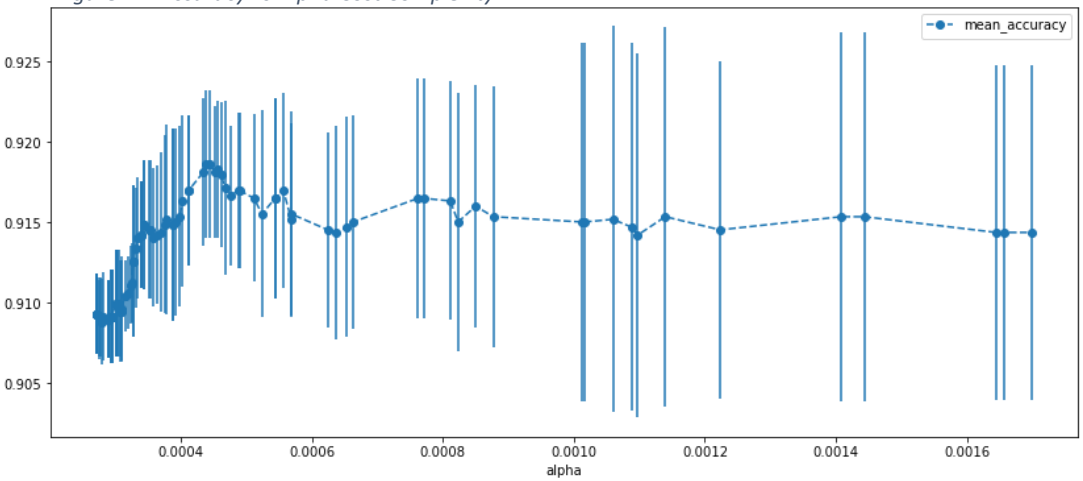
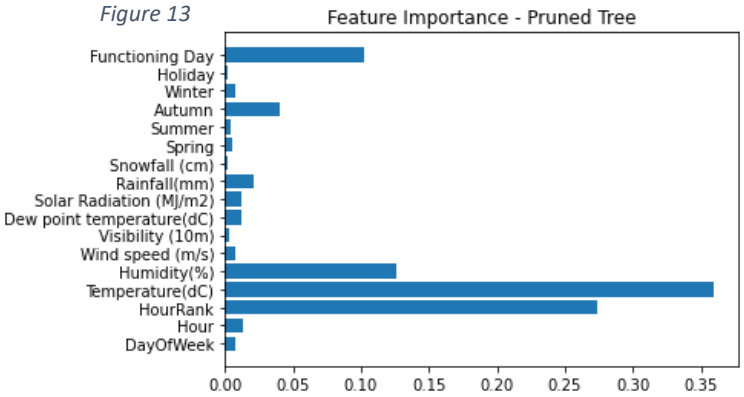
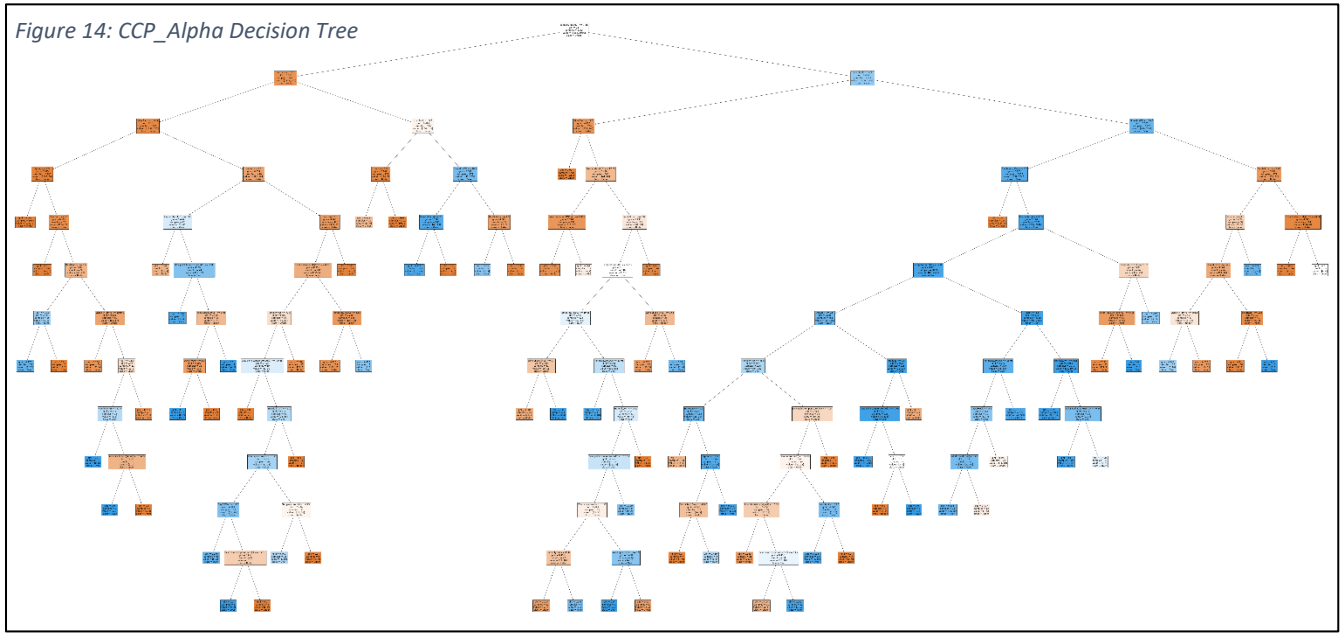


Figure 13



As seen in Figure 14, the pruned decision tree shape is significantly shallower as we have reduced our depth to just 13 levels as opposed to 21. If from an interpretability perspective we wanted to have a shallower tree, we could easily limit the depth through the use of the parameters and undertake the same approach as we just performed with an understanding that the accuracy results will not be expected to perform as well as this model.

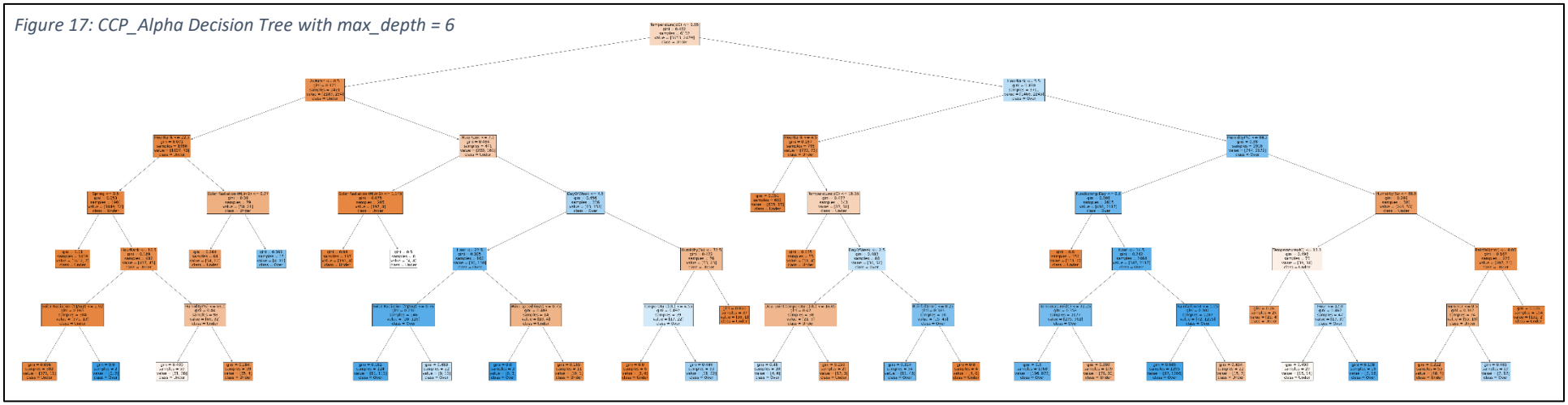
Figure 14: CCP_Alpha Decision Tree



FURTHER TESTING:

Running the Decision Tree classifier on the Mean Value of the training set still produces a final tree with test accuracy of 91.17%. However, this tree is still deep with 18 levels. It appears that our ccp_alpha approach pruned the tree, but not significantly. Further pruning the tree manually allows us to maintain a 90.5% test score all the way down to a max depth of 6. This is impressive and very interpretable as you can see in Figure 17.

Figure 17: CCP_Alpha Decision Tree with max_depth = 6



I have chosen to not run the 60/40 train test split for the decision tree model. The advantage to running it on the SVM models was that it provided a means of reducing the time it took to train. Since the Decision Tree Classifier does not suffer from the same cost issues regarding time, it does not make sense to deviate from the 70/30 split.

I did however run a temporary file in which I removed all of the transformation data that I created for my regression model and ran the tree. The resulting ccp_alpha model had a test score of 90.06. So, it looks like my transformed data creation had a 2.5% added value for this train/test split when you compare this models 90.06% accuracy to the 92.5% from the ccp_alpha tree pictured in Figure 14.

Final Evaluation:

Overall, I think that developing classification tools with over 90% accuracy is pretty good. It would be interesting to see if this was a result of the data transformations that I performed or if the results would change if we chose a target value that was further away from the median and mean values that I experimented with. Perhaps a better classification model would come from performing an estimation at several different target values that spanned at least two Standard Deviations of our data, but this would only be needed if you were in a business scenario in which a variation in the target was needed. One such scenario might be if you were to expand the model to include locations within the city and you wanted to see if the location had enough available bikes for anticipated demand. In this case, adding the location in and of itself would increase the complexity of the classification tool. You would also have to consider if you needed to change the way you measured success. Having a median target value was ideal for using Accuracy as the measurement for "best" in our current model. Accuracy continued to serve as a good indicator while the other statistical measures did not appear to suffer in our testing. However, there could be different results if we are testing more extreme values. Returning to the business implications, predictions that lead you to believe that you have enough bikes at a location when you actually do not may cost the company not in just missed profits but could lead to churn in your customer base. This would force you to reevaluate what you considered to be "best".

There were several other things that popped into my mind as I composed these algorithms. It would be interesting to run a linear regression model on the data and then use hypothesis testing to eliminate variables and see if the remaining variables were similar to the prominent variables found in our decision tree model. After seeing how my transformed data outperformed the original data, it would be interesting to also see if you could identify significant interaction variables for the linear regression model and if adding those variables led to even better results for our tested classification models. I would think that interaction variables might make for shallower decision trees, but I suspect that it would not have a strong effect on the SVM classification.

One thing that could also be done is to explore different CV settings. At one point, I began to test the effect various CV settings during my SVM analysis and plotted the accuracy as gamma and C parameters changed. I found that I was spending more time waiting for the model to run than I was on actually doing any work. For instance, running CV over 4 and 10 folds took anywhere from 1 hour to 2 hours. I eventually had to abandon this idea and limit my CV settings to the more traditional 5- and 10-fold approach. You could also argue that adding the additional CVs with these models would not have produced drastic enough change in the results for the increased amount of time. Perhaps if this were a model in which you were constantly refreshing on a daily or weekly basis, you might be able to vary which CV method you used and you could track the results through alternation. But for another project, revisiting this route might make the difference between a profitable venture or a loss.

Finally, I think the biggest takeaway that I have from this project is how valuable the ccp_alpha setting can be when creating decision trees. This will definitely be a starting point for me in the future to see what a best performing tree looks like under test conditions and then, depending on the scope of the project, any further refinements can be added through the other parameters. If I had to choose which of the models I would prefer if this were my business, I would want to go with the Decision Tree using Cost Complexity Pruning. Both SVM and decision tree approaches have given very similar classification results. As such, I would prefer to not have the black box approach of the SVM in favor of the clear rules provided by the decision tree classification and the added benefit that it performs at a much quicker pace.