Sheena Ganju
CS 4641 HW 1 Write-Up

Introduction

This report is structured in the following way: problem descriptions, algorithms used and their implementation, results, and overall results/analysis. All the data in this report and more can be replicated from running the code with the instructions from the readme file.

Please note that performance data for all the algorithms are put together in one large table in the Overall Results Analysis section.

Classification Problem 1: Predicting if Crime Was Violent in London Boroughs

Heretofore known as the "LSOA" dataset. Size: 6 columns, 1048576 rows

This Kaggle csv (London_crime_by_lsoa) contained the LSOA, what seems to be the British equivalent of a zip code of the city, the borough, the category and description of the crime, the month and year of the crime, and the number of total incidents in that month. There are a few interesting problems that you can tackle with this data, including prediction how many crimes will happen in a month given the other factors or how many crimes will happen in a month given that an area is prone to pretty versus violent crime.

The problem I chose to solve for this project was to use the lsoa, borough, year, and month to decide which general classification, or category, the crime went to. A practical use for the extrapolation on the test set is that depending on these four factors, a first responder to the emergency call could potentially predict the general classification of the crime, and be better prepared to face the situation.

This was also personally interesting to me because I want to see the effects of month and year as well as areas on crime rates and to know whether the time of the year/specific seasons affected the categories of crime by a significant amount. I also wanted to see how any trends over time affected the accuracy of this classification problem, as a result from May 2008 in a neighborhood that was revived in 2017 may or may not be an accurate predictor for future crime.

Classification Problem 2: Predicting if Alcohol is Served

Heretofore known as the "Restaurant" dataset. Size: 138 rows, 18 columns

The restaurant dataset was a list of data for Mexican restaurants, with general location data down to the latitude and longitude, but also franchise, price, environment and other service information. Problems that this data could solve range from determining handicap accessibility based on an area, determining the clustering of certain types of restaurants in Mexico, and determining franchise information based on the region/latitude.

This problem I chose was to classify if a restaurant served alcohol or not based on the rest of the factors in the sheet, none of which are necessarily highly correlated with the others. This problem interests me because it's important to me, for personal reasons, to be able to predict whether a restaurant I could one day dine at in Mexico is an establishment that serves alcohol (though, as a tourist, I'm sure the rate for the places I'd likely be around would be a lot higher.)

Decision Trees with Pruning

Test Accuracy: 99.978

Train Accuracy: 99.968

Sheena Ganju
CS 4641 HW 1 Write-Up

Details

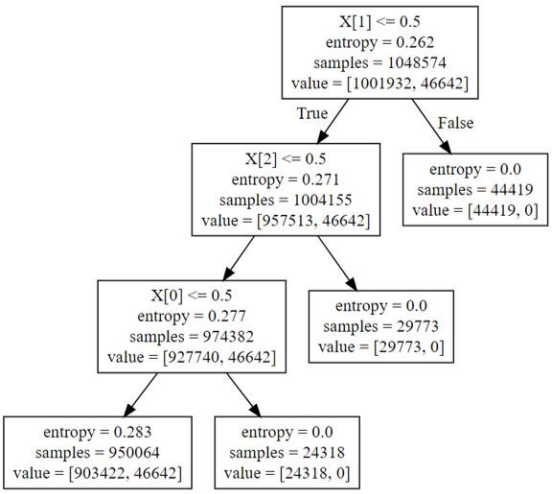
This code was run using scikit's Decision Tree Classifier, with the specification that attribute split for the restaurant data, module was created using **GINI,** while **entropy** was used as the attribute split for the LSOA. The GINI index is an attribute that measures how often an element would be falsely identified, so the branches with the lowest GINI indexes are preferred using this method. The GINI score for the restaurant dataset, as it was a very easily learned dataset, was 0. For the restaurant set, the GINI was mostly low, as the splits for this dataset were clear. Low entropy is also favored while deciding splits, and a user can see the LSOA decision tree graphs to see that the lowest entropy is used at the first split.

The training and test data sets were created from the total dataset and were optimized for size using the sklearn function **train_test_split**, which splits the data into random test and train sets and checks for over and under fitting of the data. Prior to the implementation of train_test_split, the SciKit algorithm had a training accuracy without pruning for the LSOA of 95.552. Post-implementation, this training accuracy increased to 99.978 for this dataset, indicating that the split helped optimize the ratio of the size of the train to test set.

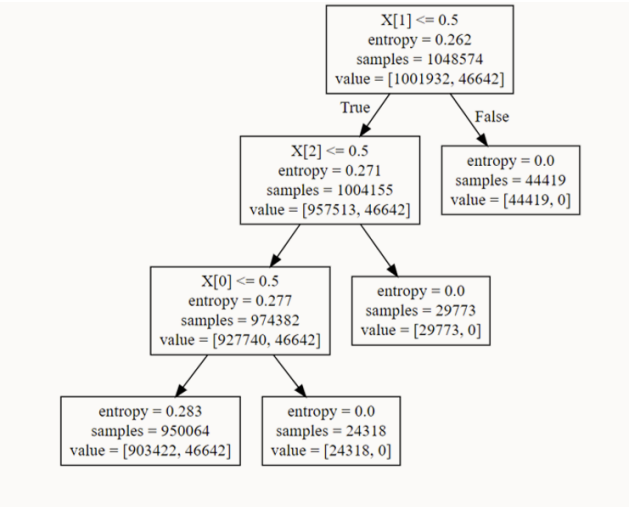
The **pruning** algorithm had the user specify max tree depth, but this was an ineffective strategy for both examples, as the trees were already shallow, despite the size of the LSOA dataset. For the large LSOA dataset, pruning changed absolutely nothing, but for the smaller restaurant dataset, the left branch of the first split had slightly different specifications, as can be seen in the graphs for both trees. The sheer size of the LSOA dataset compared to the size of the tree seems like it could be a case of overfitting, because although there were only 6 columns and close to 200,000 data points for each column, it doesn't seem likely that there would be so much repetition in the dataset. More investigation on the error of this model would need to be done were it to be selected.

Decision Tree Structures, pre- and post- pruning for each dataset

Decision Tree: LSOA pre-pruning

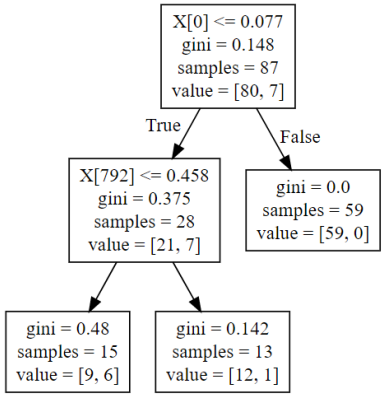


Decision Tree: LSOA post-pruning

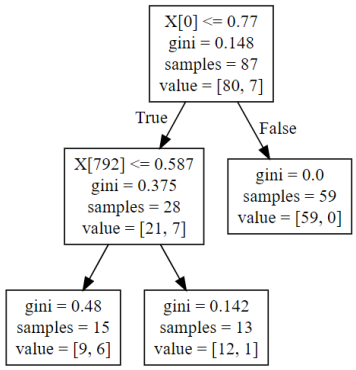


Sheena Ganju
CS 4641 HW 1 Write-Up

Decision Tree: Restaurants pre-pruning



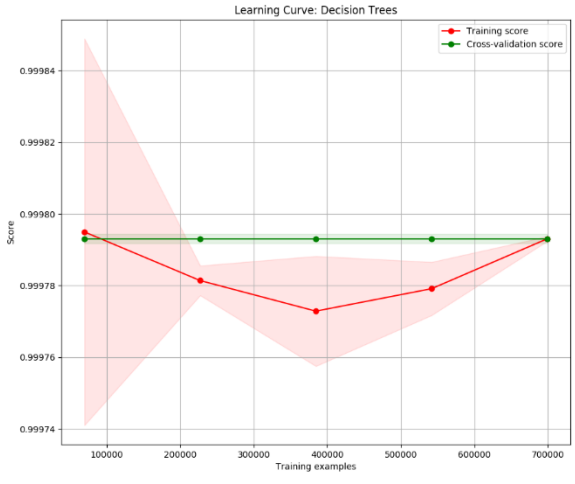
Decision Tree: Restaurants post-pruning



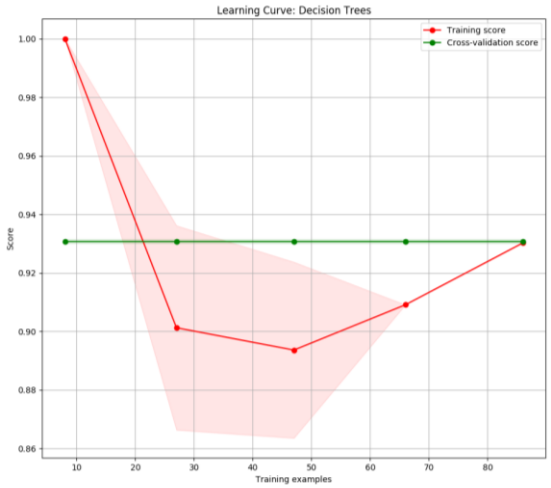
Repeated runs of this algorithm did not demonstrate any significant performance or accuracy changes, and the precision measure of these models were 99 and 90%, respectively to each model. The runtime score of this model was a 1506114335, which is a metric that will be explained later in the paper.

The below graphs are the LSOA learning curves, which show the cross-validation score of the model, with the standard error around the measure in green. This model did not deviate significantly in terms of cross-validation across sample size. Training size on the training model for LSOA had a significant effect on the fit, demonstrating that either a relatively smaller dataset or a very large one should be used for application of this model, but not a dataset in the middle.

LSOA Learning Curves



Restaurants Learning Curves



Neural Networks

Test Accuracy: 99.979

Train Accuracy: 99.969

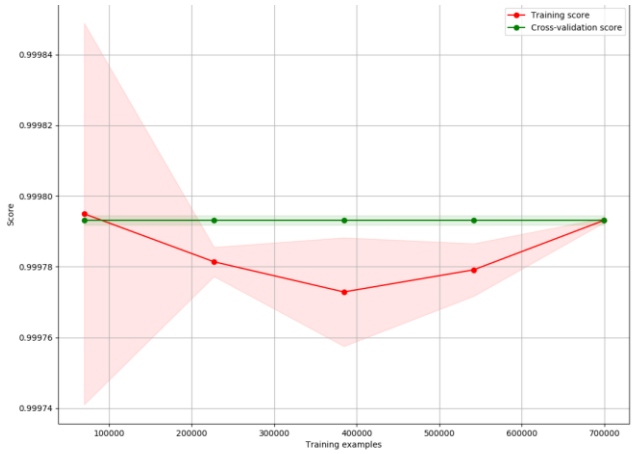
Details

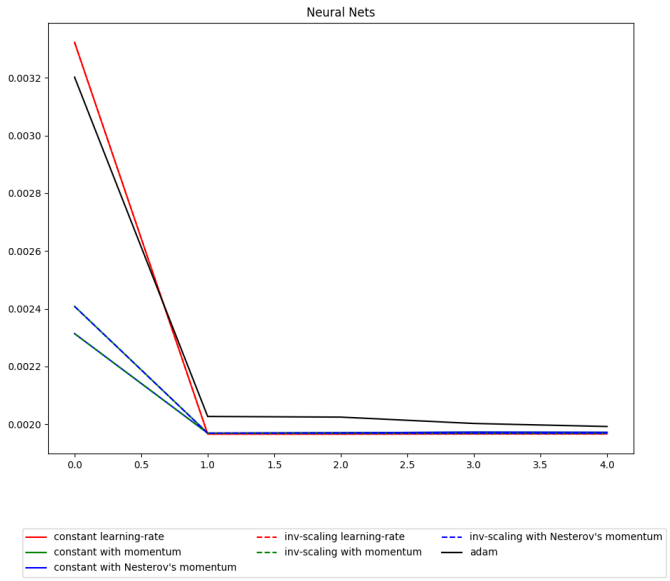
Scikit's Multilayer Perceptron (MLP) setting was used in this instance on both the restaurant and LSOA datasets to build a nueral network classifier. **Stochastic gradient descent**, which as we learned iteratively determines the negative gradient of a point in a function, was the algorithm specified for weight updates in this classifier. For time efficiency, the **max number of iterations** allowed for these networks was 50. The **learning rate** of this model was constant and the

default .001, and for the fits for these models, momentum was not used, although different momentum rates were graphed on the datasets to compare their usefulness.

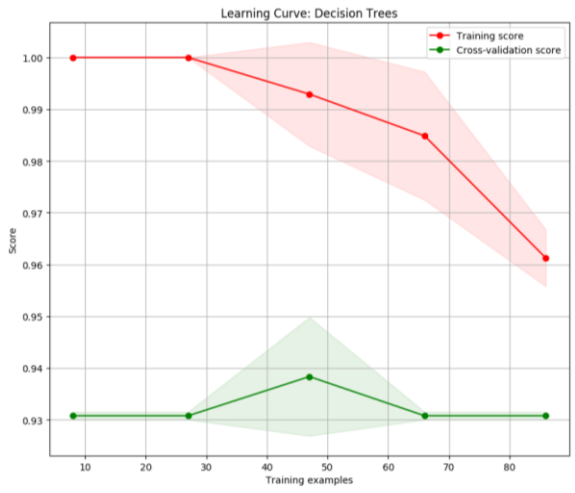
Plots and Explanations

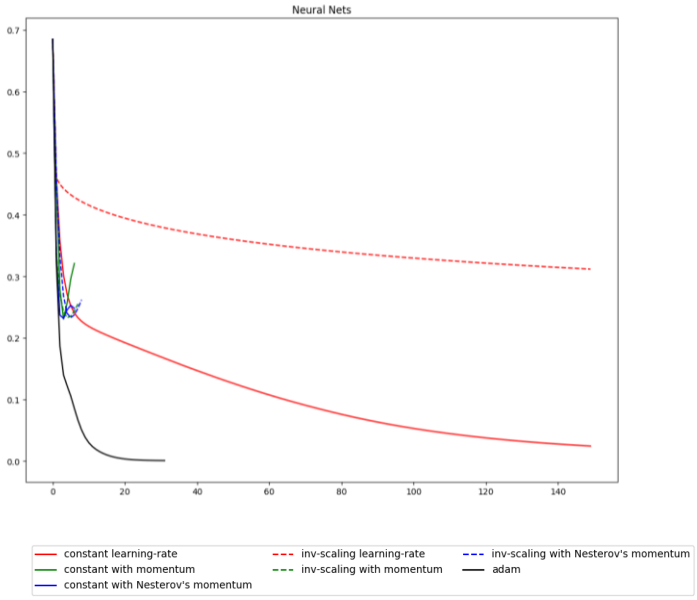
LSOA Learning Curve and Learning Rate Analysis





Restaurant Learning Curves and Learning Rate Analysis





Explanations

The learning curves and cross validation for the LSOA demonstrated the same things as they did in the decision tree example- this model was most accurate on a model size that was closer to either ends of the graph, and not the middle. The graph on the right demonstrated the potential effects of learning rate (X) on the error of a classifier (Y). For this example, it was clear that the constant learning rate implemented in the above model was actually one of the more error-prone, and that a less error-prone but time efficient method of calculation would have been implementing standard momentum or Nesterov's momentum, both of which are techniques to optimize error while decreasing

training time. I didn't implement these in the results, but would revise this technique if this was the model I were to move forward with for modeling on this dataset. These results had very high precision and medium to low loss, and were not expensive in terms of modeling time.

The model fit for the Restaurant data was much less clear cut in terms of a learning curve, as the accuracy trended downwards with the increase in training examples, but started at 100. This means that the model was very accurate for the test data we had, but would not extrapolate well to other data. The errors from the constant learning rate implementation were relatively high, but it seems that the only other reasonable learning rate algorithm for this data was adam.

This model and graph were created very quickly and had a training accuracy of 100, possibly because the dataset was smaller, but had a very high loss score of 160.64, which indicates overfitting.

Boosting

Test Accuracy: 99.994

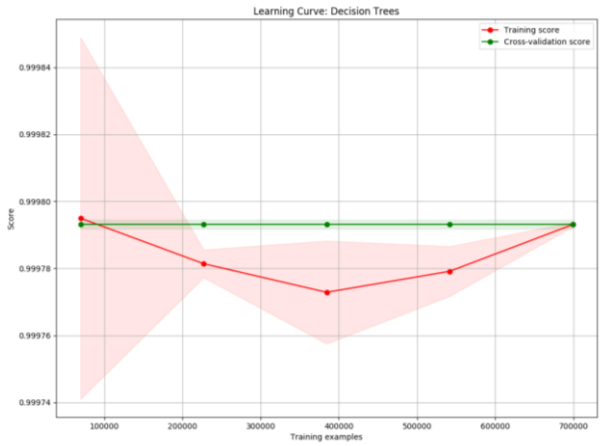
Train Accuracy: 99.989

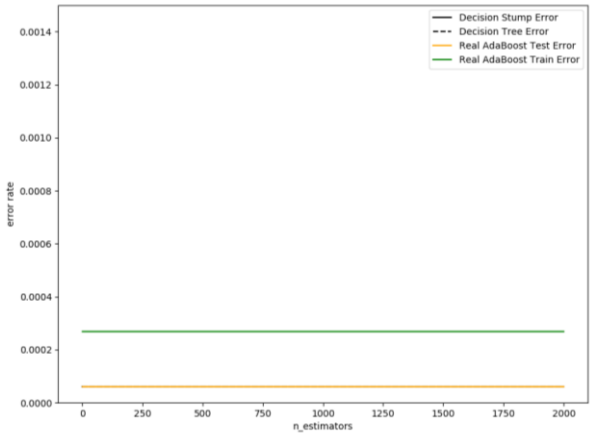
Details

The Adaboost algorithm from Python Scikit, which fits copies of the same classifier onto a dataset and weights the incorrect dataset higher, was applied to both decision trees from the first example with the gini splitter. and a max depth of 3 applied for pruning. The number of **estimators** used for both examples was 100, but all other parameters remained the same from the first example.

Plots and Explanations

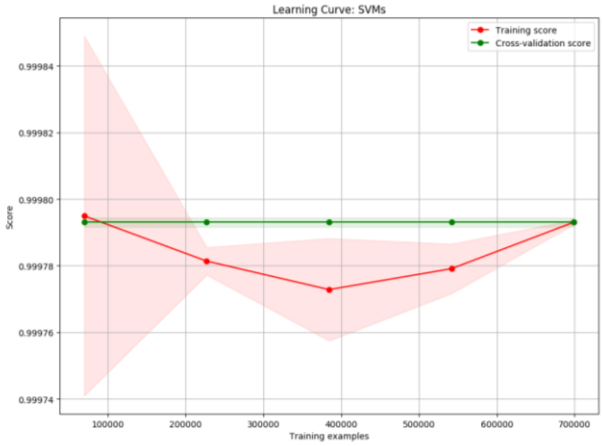
LSOA dataset

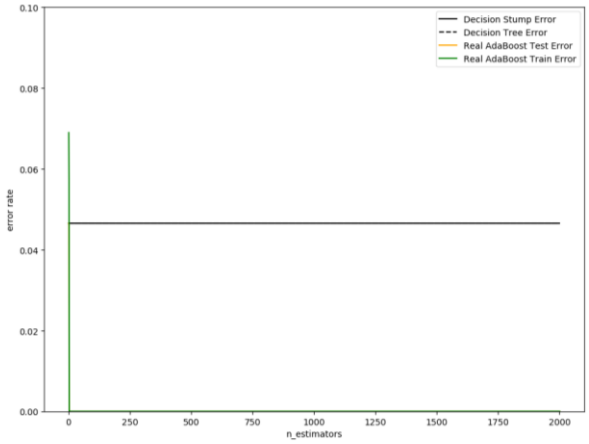




Sheena Ganju
CS 4641 HW 1 Write-Up

Restaurant dataset





Explanations

The boosting learning curve demonstrated similar results as discussed before. The charts on the right are a measure of training error to the number of estimators being used. For both charts, this error was constant over any reasonable number of estimators, so 100 estimators were used. As the model performed particularly well on the restaurant dataset and was too close to zero, pyplot was not able to accurately demonstrate the relationship between error and estimator number, resulting in the graph showing few trends. Performing this analysis on another less easily classified dataset would produce better error results.

Boosting did increase the accuracy of the test classifications by .02 percent on the large dataset for a rest accuracy of 99.97%, and by .1 percent on the smaller restaurant dataset to a test accuracy of 95.4%. The loss function for this measure was also very low, although it took a considerably larger amount of time than all but one of the other algorithms.

Support Vector Machines

Test Accuracy: 99.978

Train Accuracy: 99.956

Details

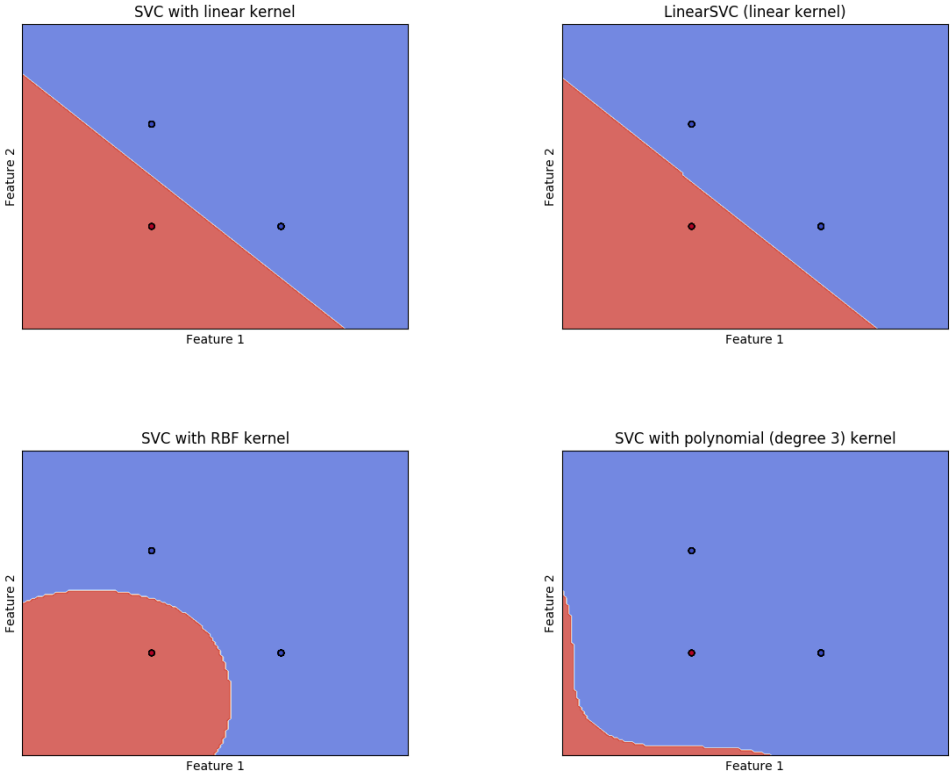
The **two kernels** selected for this section in Scikit were linearSVC and RBF, as they both allow for multi-class classification. Linear SVC is one method for sectioning the space into classifiers that uses straight lines. In contrast, the radial basis function (RBF), sections the space based on radial and not linear distances.

For the restaurant data, the use of the linearSVC kernel resulted in a reasonably precise 95% accuracy, and 99.78 for the LSOA data. Both kernels had the same precision and loss functions.

Sheena Ganju
CS 4641 HW 1 Write-Up

Plots and Explanations

Restaurant Dataset



Explanation

The plot above is the classification of the first two features in the restaurant dataset, and the possible results. All but SVC with polynomial degree correctly classified all of the true means, and it seems that either the linear or the RBF kernels were appropriate choices for classification.

Though the SVC model had a very high training accuracy that was too high to be true, it was impossible in terms of computation time to plot the gradients for the different kernels of the LSOA dataset, so this is the only model for which we are sure that both of our kernels were good fits.

For the purposes of this homework, I used the default C and gammas, as I did not implement any functions that would endorse any specific values.

k-Nearest Neighbors

Test Accuracy: 99.999

Train Accuracy: 99.987

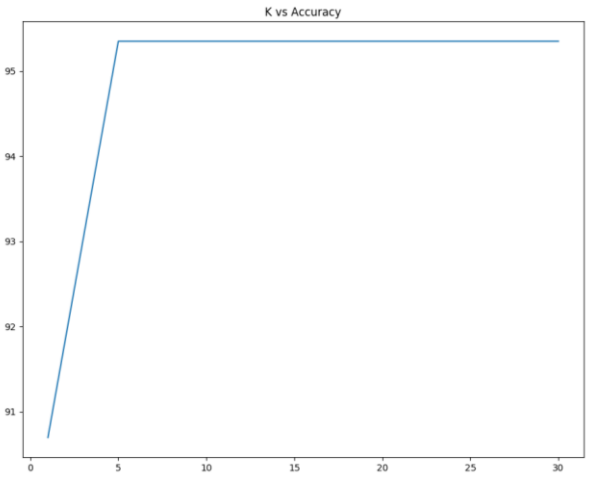
Details

For k-Nearest neighbor implementation, the k-Neighbors Classifier option in Scikit was implemented. To find the optimal k for each dataset, reasonable values of k were plotted against their proposed accuracy on the model. K for both of

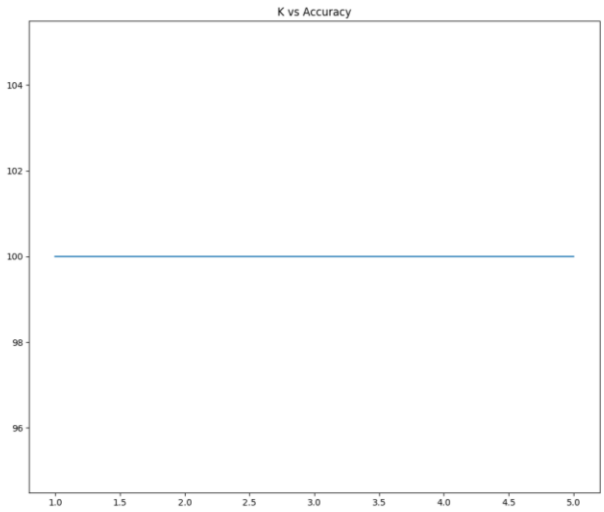
these datasets was set at 5 because of the values plotted below, and the rest of the options to vary were set at default, including a uniformly distributed weight function.

Plots and Explanations

LSOA kNN



Restaurant kNN



Explanation

For the LSOA, there was a clear correlation between k size and accuracy for the range chosen. However, for the restaurant kNN, there was zero graphed difference, and 100 percent accuracy reported, which does not represent a correct estimation. This model would need to be re-evaluated with different parameters and a better accuracy metric.

Overall Results Analysis

All of the statistics, including training accuracy and precision measures, appear in the code when run, but these are the best test runs I was able to achieve. Here, the "run time score" comes from a glitch in my time tracking system and are not measures in a time unit, but can be compared to each other to determine run time for the models.

LSOA Answers Comparison

Method	Best Test Accuracy (%)	Model Precision (%)	Model Loss	Run Time Score
Decision Trees with Pruning	99.978	99.956	.75859	114335
Neural Networks	99.979	99.956	.75859	189539
Support Vector Machines	99.978	99.956	.75859	212701
k- Nearest Neighbors	99.999	99.987	.20932	n.a. (but very long)
Boosting	99.994	99.989	.20932	194383

Restaurant Answers Comparison

Method	Best Test Accuracy	Model Precision	Model Metrics	Run Time Score
Decision Trees with Pruning	95.349	90.914	160.645	207213
Neural Networks	95.349	90.914	160.645	208314
Support Vector Machines	95.349	90.91	160.645	211800
k- Nearest Neighbors	100	100	0.000	216379
Boosting	95.412	95.349	160.647	210219

Sheena Ganju
CS 4641 HW 1 Write-Up

Interpretation

Overall, each algorithm performed the way it was supposed to, with lower training accuracy than test accuracy for every model except for k-NN for the LSOA approximation. The overall high performance of all algorithms, particularly for the LSOA dataset, was due to the simplicity of the classification problems, as they were well-equipped datasets for this type of experimentation. The random baseline for the restaurants dataset was one in three, as there were three answers: all alcohol was sold, it wasn't, or only beer and wine were sold. The random baseline for the lsoa dataset was higher, about 1 in 12. There were only five predicting factors to train on in the lsoa dataset, and about 17 factors to train on in the restaurant dataset.

The similarity of test accuracy and precision was striking, particularly in the LSOA dataset. My impression is that since the dataset was so big and the choices were so limited in comparison to the size of the dataset, that most of the models could achieve very strong fits to the training set. Although **K-Nearest neighbors** performed the best in terms of test accuracy, it was an extremely expensive algorithm on my large dataset, taking almost 12 minutes. Therefore, although the test accuracy was high and the loss was low, I would not say this algorithm performed the best, especially since I was not able to produce a learning curve on the data and do not know how these results would extrapolate, or how much more time this algorithm would take given an increase in dataset size. I would select kNN as a candidate for the best model for smaller datasets given the same result, though.

For the restaurant answers, **kNN** also stood out as the highest performer, but the fact that my model claimed zero loss and 100% precision makes me suspect that I incorrectly implemented the model. For this reason, kNN is not the best model for this classification. Rather, for this model**, boosting** seems like the most viable method, as it was the most precise method of classification, was time efficient, and produced the best result. This method also makes the most practical sense, as there were 17 rows of data used to make the decisions, most of which are likely weak classifiers, used to make one of three simple decisions, which is a good situation for implementing boosting a tree.

In general, for both problems, **decision trees** were very small because of the simplicity of both of my classification problems, so pruning and **boosting** were not as effective measures as they could have been in a situation with weaker learners. This algorithm performed very well in terms of time and was applied with high accuracy to both models, so it is a strong candidate for both problems. I am suspicious that my simple decision tree model overfit the restaurant data, however, so additional testing would need to be done to confirm that this classifier was producing accurate and precise results.

To improve the performance and runtime of **neural networks**, I would experiment more with different weight selection methods and implement momentum, so long as these changes didn't cause more overfitting. I believe this would have also been a very strong candidate for both problems I selected had I specified more of the right parameters because of the nature of the classification problems. I would also try to improve the performance of my **SVM**s, as they gave little reward for their expense with the settings I fed in, but perhaps they would have performed better had I done proper gamma analysis instead of zeroing in on solely kernel specification.