Simple Classification Analysis

I. Classification Problem Description:

Dataset 1:

Mushroom Classification: This dataset includes description of hypothetical samples of mushroom from The Audubon Society Field Guide to North American Mushrooms. It includes in total 8124 instances and each instance of mushroom is identified as edible or poisonous (or suspicious). There are 22 string attributes describing the properties of these mushrooms and none of the attributes alone determines whether it is edible or poisonous.

Classification Problem: Using this dataset, I want to explore which machine learning models perform the best on this dataset in predicting if a particular type of mushroom is poisonous or not. This dataset is interesting because it appears to be a very typical classification dataset with clean data from a reputable source, indicating the probability of it containing error/noise is low. Considering the purpose is to better understand different machine learning models, it would be more reliable to have a dataset we have confidence on so analysis is not affected too much by uncontrollable factors. Knowing the high reliability of the dataset, it would also be interesting to artificially add in noises to see how it would affect each learning models which is not something we can easily do with an already noisy dataset. Also, this is a very practical inductive problem to look into since there is no single rule that can predict if a mushroom is poisonous or not and it is hard for a non-expert to analyze all 22 (or even more) different attributes in brain and come up with a conclusion, so machine learning can show its power in generalizing over complicated attributes to give a fairly accurate prediction from learning.

https://www.kaggle.com/uciml/mushroom-classification

Dataset 2:

World University Rankings: This dataset includes university ranking data from year 2012-2015 from *The Center for World University Rankings* is a listing that comes from Saudi Arabia founded in 2012. The raw data includes in total 2200 instances with 14 attributes. The attributes include information such as education and research power in numerical format.

Classification Problem: In this dataset, I want to use attribute score (0-100) as the target attribute. This is an interesting dataset as it can be considered both as a classification problem (if we divide the output as a large number of categories) and a regression problem (if our goal is to produce the score as a continuous value). Also, it has a time series attribute 'year' (from 2012-2015) which I plan to omit. It would be interesting to see how well different machine learning models handle the data when some part of the data is omitted. This dataset also interests me personally as college ranking has had a non-trivial impact on my choice of application and despite trust in the ranking, I have always been curious about if and how rigorously these rankings follow particular rule.

https://www.kaggle.com/mylesoneill/world-university-rankings

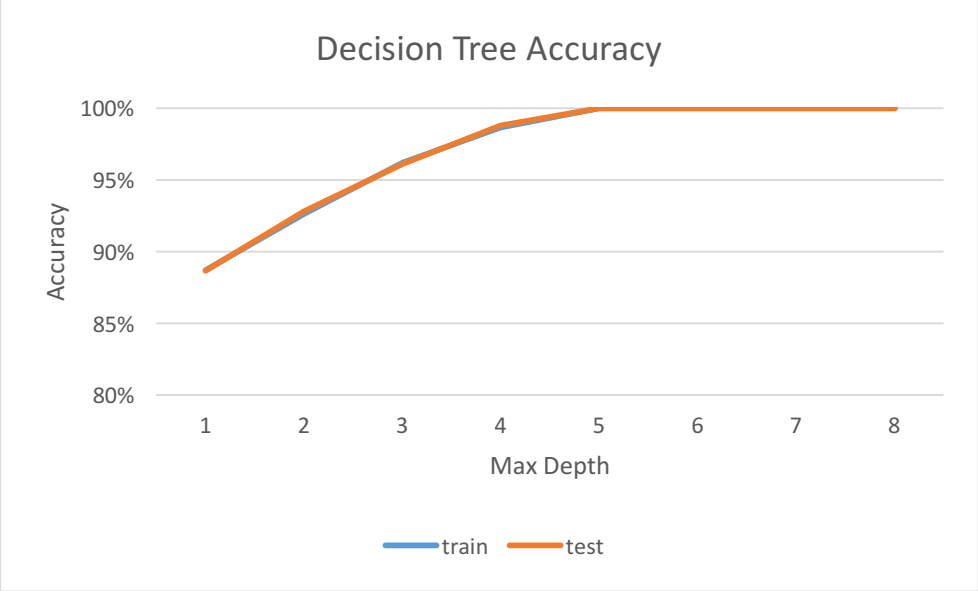
II: Machine Learning Methods:

Dataset 1 Mushroom Classification:

First, I want to run the highly reliable raw data which is believed to have very low noise. There is not much pre-processing to do for the data except all attributes are in string so I created dummy variables for all of them so they can be easily processed by any machine learning methods. After processing, the attribute number increases to 117 all with values {0,1}

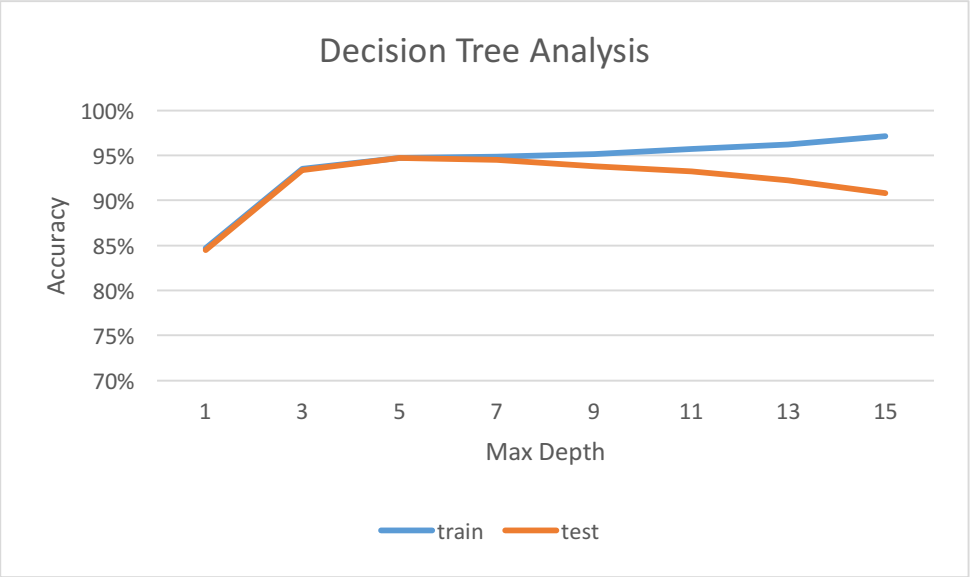
Decision Tree:

I used DecisionTreeClassifier in scikit-learn to implement the learning method. Two choices for criterion are GINI impurity and information gain. I used information gain('entropy') in this case. There are a few possible ways to achieve pruning based on the parameters provided, I can either restrict the max depth or set a min number of samples where split should stop or set a min number of samples for a leaf. All of these options can effectively stop the tree from growing too big to categorize every sample in the training set 'correctly' and cause over-fitting. In this case, I am pruning by restricting max depth of the tree. Below is the a graph showing how accuracy changes with change in max depth:

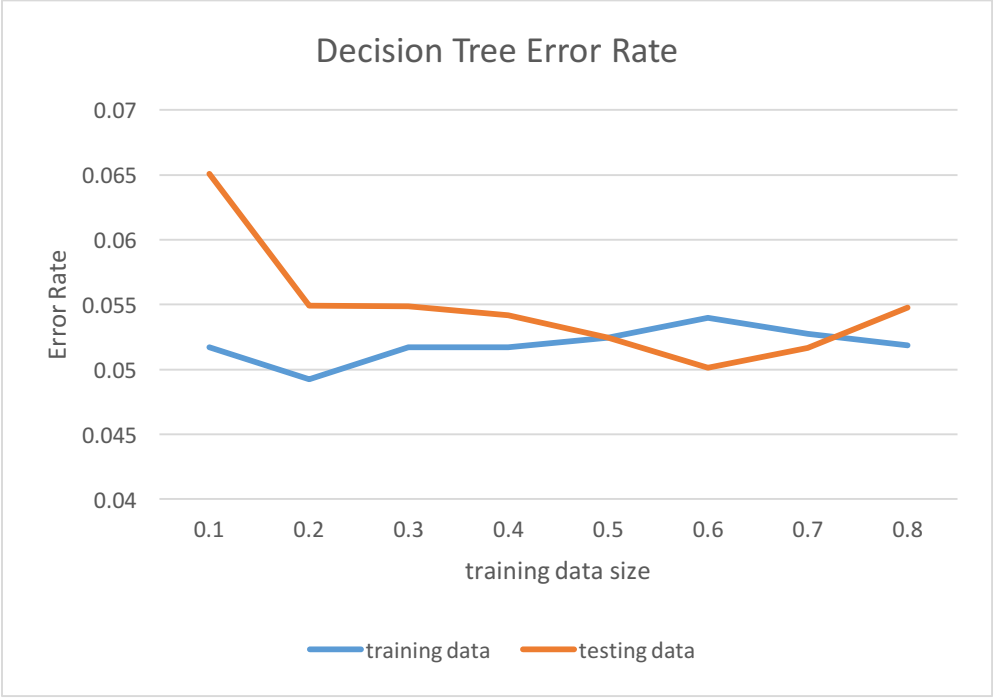


We can see in this graph that instead of the graph we've seen in lectures where training data used to over-fit and the accuracy in testing data drops when the tree grows larger and larger, we see here that both training and testing data reached accuracy of 100% and remains. This leads me to believe that this dataset is error-free, hence there was no over-fitting problem occurring. In order to understand how robust methods are with errors, I randomly chose a portion of the train data and altered their classification. We can see from the graph below that with the existence of noise, decision tree shows over-fitting problem. From pruning by

controlling max depth, we can see that the best max depth is 5 for the altered dataset. (number can vary based on the random instances chosen to alter but the overall trend should be the same.)



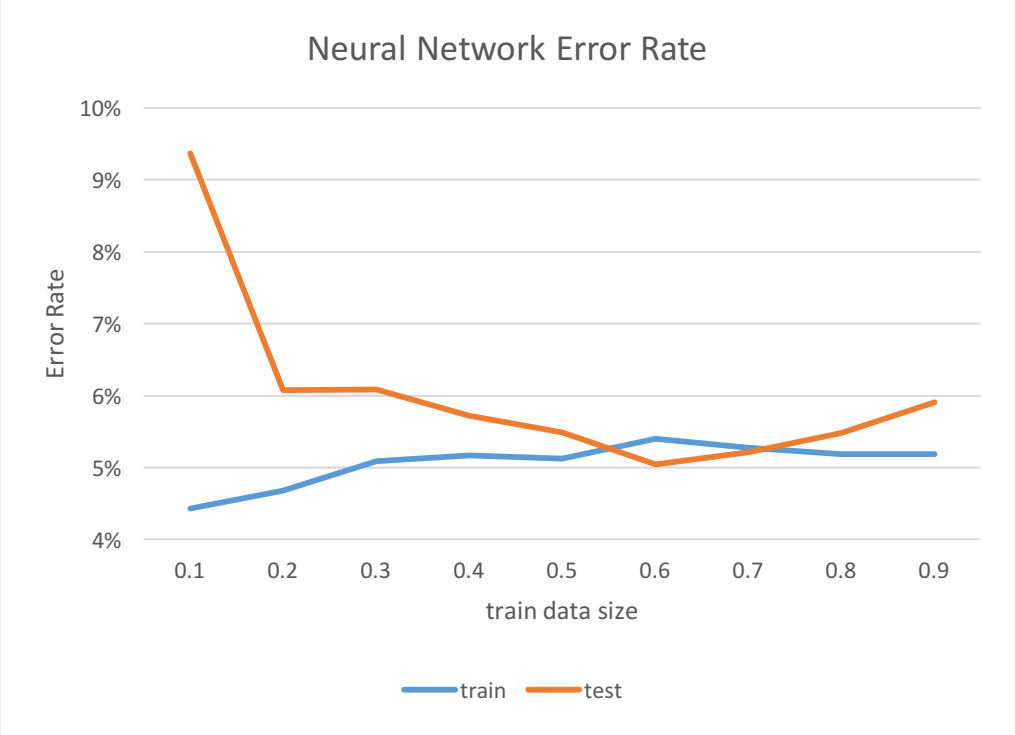
Now, let's look at accuracy of accuracy of training and testing data related to training size,



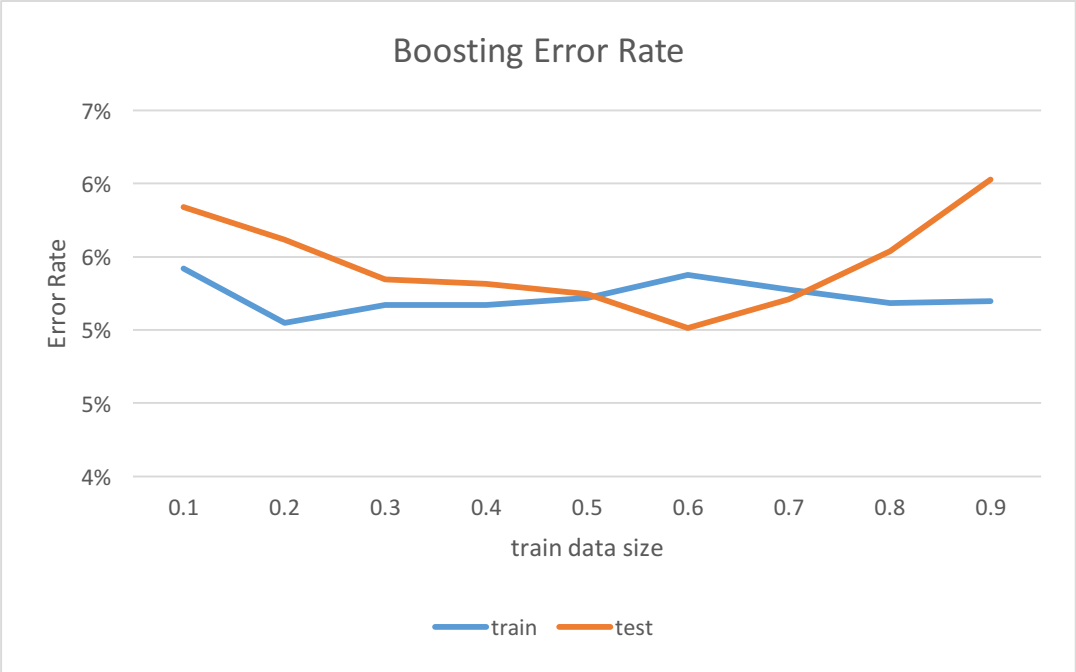
As we can see in the graph, when training data size is very small, the error rate is low for training set and high for test set which makes sense. On the one hand, few train data are easy to categorize, thus the error rate is low. However, the rule generalized from few train data is

usually poor resulting in high testing data error rate. With the increase in train data size, error rate for train data gradually increases as it gets harder to categorize these data. However, the learning becomes more generalized and and fits testing data better, therefore, error rate drops.

Neural Networks:
I used MLPClassifier (Multi-layer Perceptron) from scikit-learn package for neural networks implementation. It trains using Backpropagation. I set hidden layer size to be 8 which gives me a pretty good prediction on the test set when train data grows. When we look at the error rate with change in train data size, we see the same pattern as in decision tree. With the increase in train data size, error rate increases slightly for train data and decreases significantly for test data.

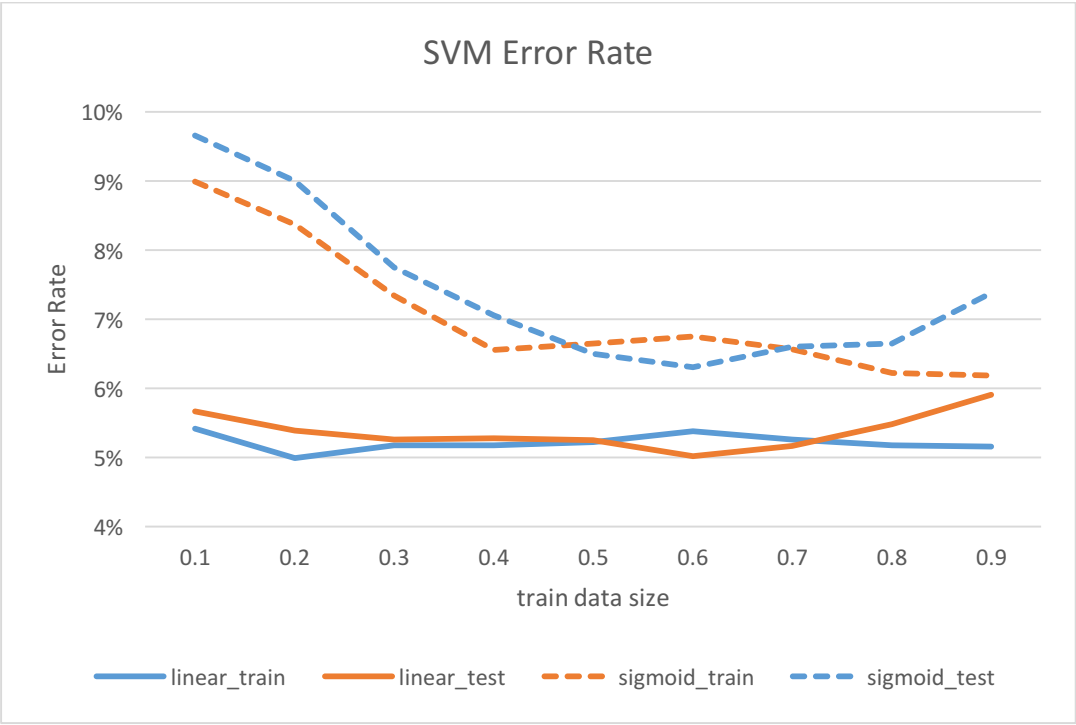


Boosting:
For boosting, I am using AdaBoostClassifier from scikit-learn where I define the base estimator to be decision tree classifier we used in the above training. Now, in turns of pruning, we can either adjust the base estimator or the number of estimators which is the number of weak learner. In this case, I am keeping the number of estimator fixed at 100 and adjust the max depth of the base estimator to achieve the optimal prediction. Using the similar method used in decision tree pruning, I can see the optimal prediction achieved for test set at max depth = 1.



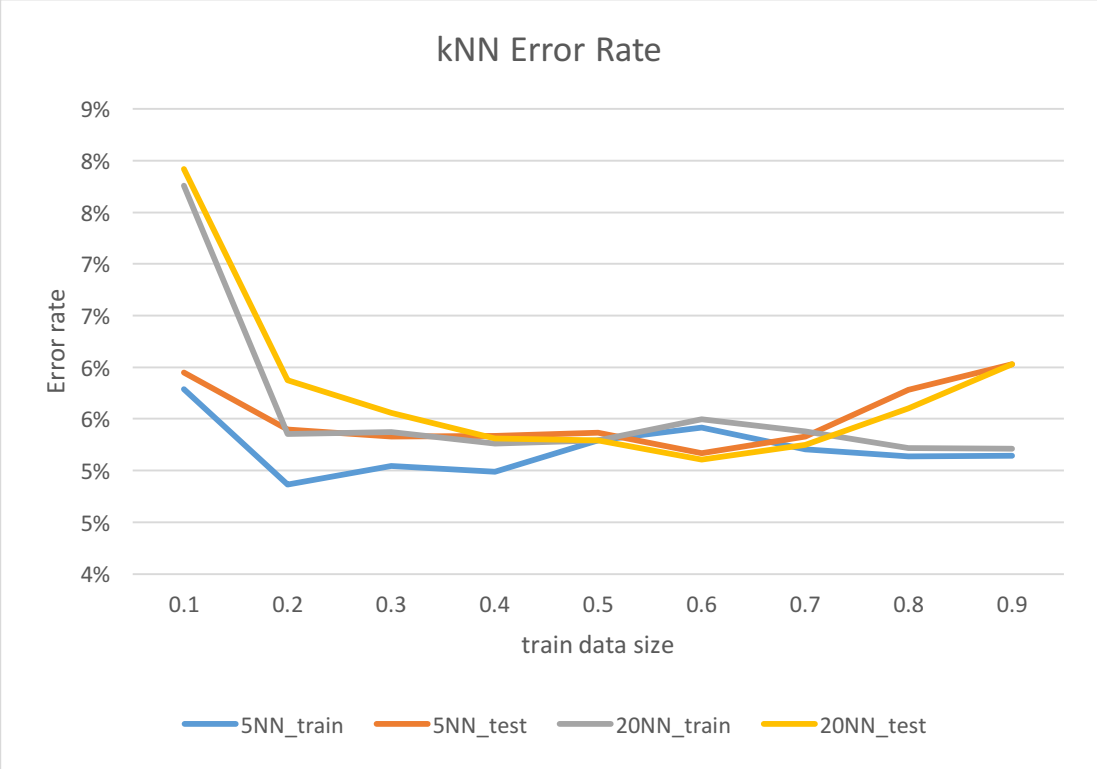
We can see that boosting actually does a pretty good job across all train data size and it tends to not over-fit if the base estimator does not over-fit which matched exactly what we talked about in the lecture that it tends to not over-fit as it place different weight on based on the difficulty of different instances.

Support Vector Machine:



I used SVM from scikit-learn package. I am using two kernel functions, linear and sigmoid. We can see that for linear kernel function, the error rate is pretty steady, indicating the data may be linearly separable. On the other hand, when looking at the sigmoid kernel, we can see initially the error rate drops for both test and train data set but later, when error rate continues to drop for train dataset, it starts to increase for test dataset. This may be a sign of over-fitting as more and more data are used for train and more noise are counted into the classification.

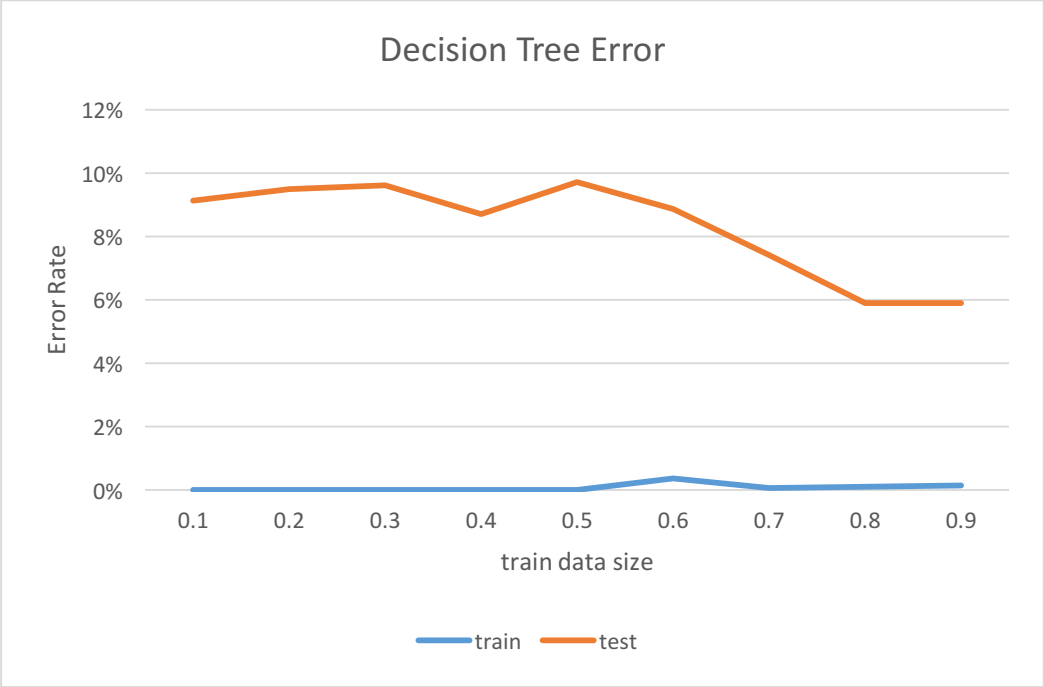
k-Nearest Neighbors:
I used KNeighborsClassifier from scikit-learn package. I have tried k = 1, 5, 20, 50 with weights set to uniform.



We can see from the graph that the error rate for k = 5 is constantly low while it is very high for k = 20 and gradually drops down with increase in train data size. The error rate also starts to rise with larger train data size. This makes sense for k = 20. As it includes more data points 'close' to the point we are interested into in uniform weight, on the one hand, it requires more data points to not make it simply the average of all data points (in the extreme case). On the other hand, when more train data is introduced, it is inevitable to have more noise that is valued uniformly as the valuable data points in k = 20 and cause the error rate to go up again.

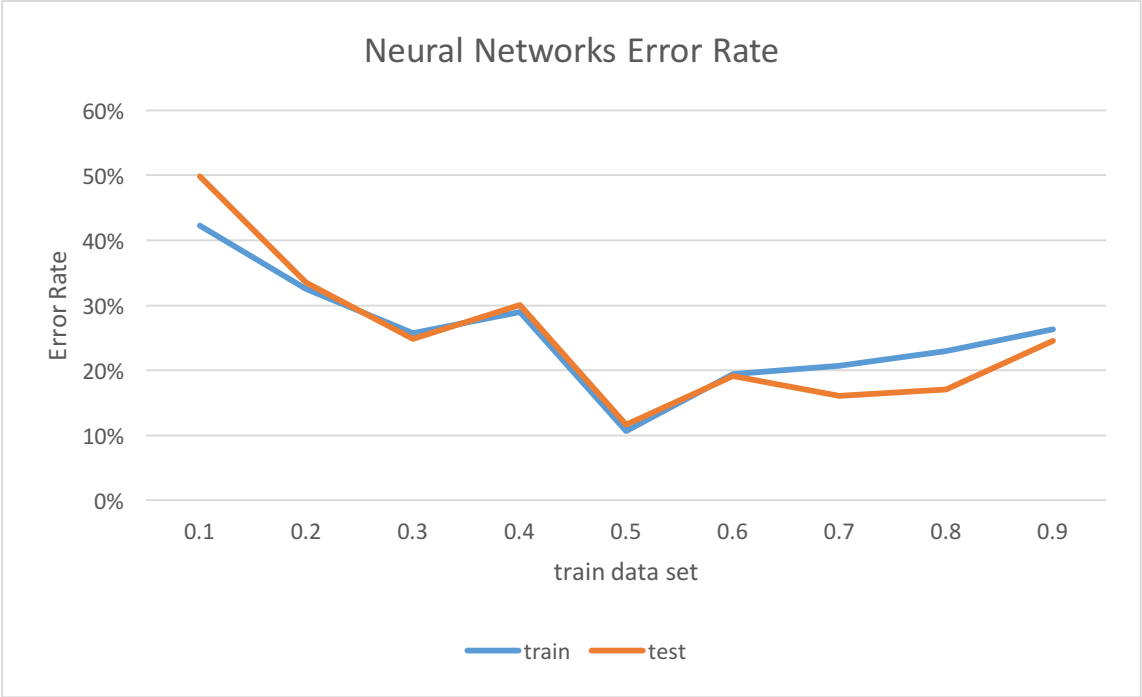
Dataset 2 World University Rankings:
Attributes for this dataset are all numerical except attribute country, therefore, in pre-processing the data, we create dummy variables for the country attribute. For the target value, it is ranging from 0 to 100, we want to make is binary for classification so we divide it at 45 which is the mid-point to divide the dataset in half.

Decision Tree:
With pruning using different max depth for the tree, the optimal max depth is around 15.



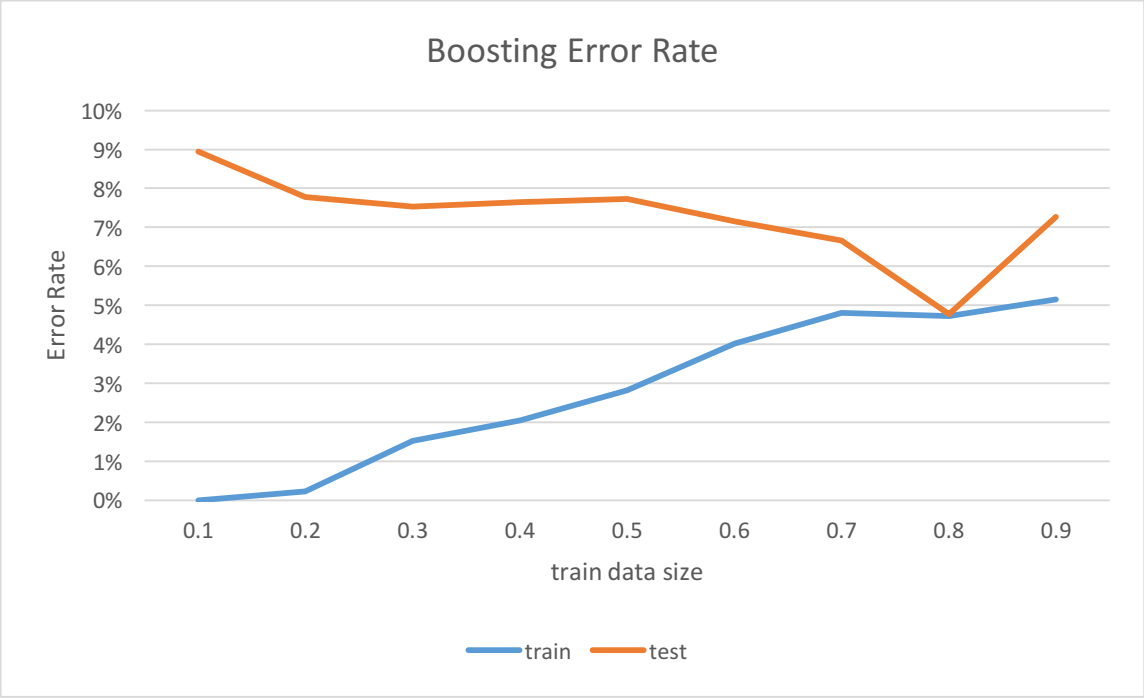
From the graph, we can see that test error rate drops significantly with the increase in train data size, indicating that learning is happening fast with increase in train data. It seems that the learning is not saturated yet on the graph, we may need more data to get a more generalized regression for this dataset.

Neural Networks:
For neural networks, I set hidden layer size to be 3. The result is similar to the above where error rate for test set decreases with increase in train data size for the same reason.



Looking from the graph, error rate for train and test data were both decreasing at the beginning and slightly increasing at the end. It is rare but it could be that the train and test data happen to be distributed uniformly that they share similar errors. Therefore when train data is affected by errors, test data is also affected by in the same way when learning over-fits

Boosting:

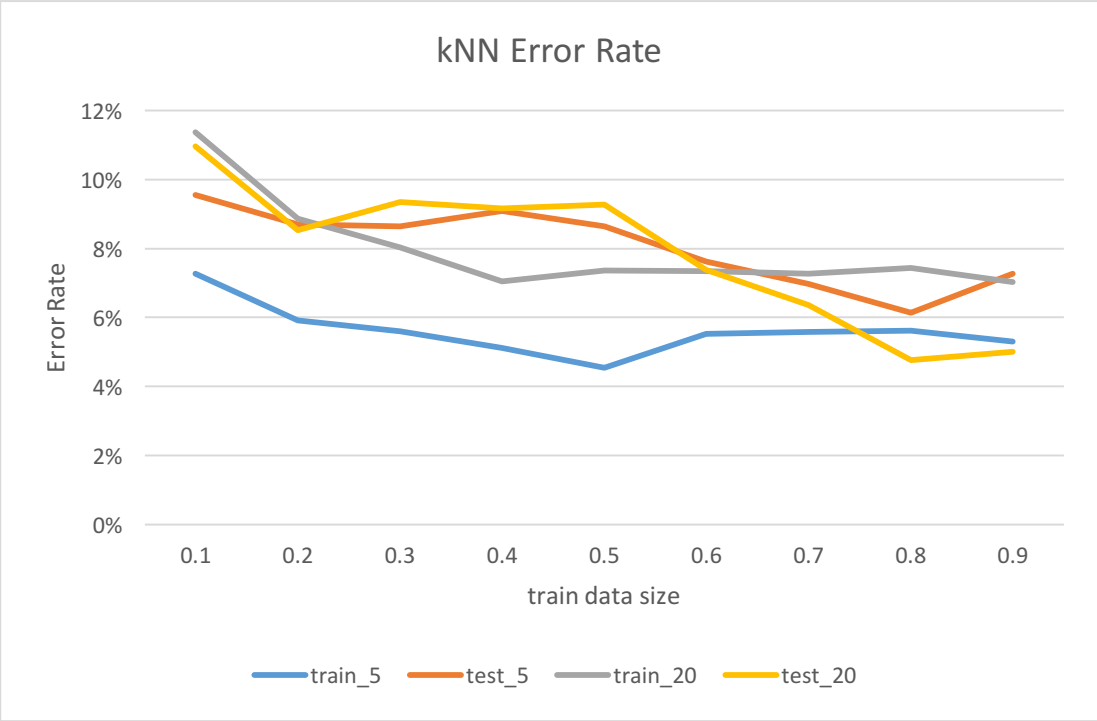


For boosting, we reused the same setting with pruning on the base estimator. We can see from the graph that it well represented the typical phenomenon where train error increases with more data introducing harder instances and more noise and test data error rate decreases with more generalized rules.

Support Vector Machine:

In this case, we are not seeing any learning occurring for this data as error rate is almost 50% for test data. For some reason, VSM was not able to learn general rules and predict test data well. My assumption is that since most of all raw score are above 43 and we set our threshold at 45 (because 45 cut the original data in half), it is possible that VSM could not well distinguish data points between 43 and 45 and hence categorized a large percentage of these data into adjusted_score = 1, resulting in error rate almost equal to 50%.

k-Nearest Neighbors:



Both train and test set error rate decreases with increase in train data size. However, we can see that test_5 and test_20 actually shares similar error rate even though they take very different k number. Therefore, large k number is not an optimal choice if smaller k number can achieve the same performance depending on how closely similar instances gather together.

IV Overall Comparison:

	Mushrooms		***College Ranking***	
Learning Method	***Train Error***	***Test Error***	***Train Error***	***Test Error***
Decision Tree	0.051702913	0.05904059	0.001515152	0.059090909
Neural Networks	0.051839694	0.05904059	0.263636364	0.245454545
Boosting	0.051976474	0.060270603	0.051515152	0.072727273
SVM	0.051566133	0.05904059	0.077272727	0.476767677
kNN	0.051429353	0.060270603	0.07020202	0.05

Looking at the overall error rate comparison at train data size = 0.9, it is interesting to see that Decision Tree, Boosted Decision Tree has been stable in low error rate across the two datasets. Neural Network and SVM has yield a pretty high error rate on the second dataset possibly due to the way the data is manually processed before learning. As there are ~50% of the data

located between 43 and 45, we set the threshold to result in 1 to be 45. It could be hard for Neural Network and SVM in this case distinguish 43-45 with numbers right above 45 and therefore incorrectly categorized many of them.

	Mushrooms		***College Ranking***	
Learning Method	***Train Time***	***Predict Time***	***Train Time***	***Predict Time***
Decision Tree	0.015264034	0.008008003	0.005944967	0.001874924
Neural Networks	0.541469097	0.007729053	0.13390708	0.002049923
Boosting	0.991137028	0.079893827	0.399791956	0.022406101
SVM	0.706838846	0.404135942	78.2111129761	0.0191690921783
kNN	0.038267136	2.883059978	0.00427508354187	0.0335490703583

This table shows the execution time of each learning method on these two datasets when train data size = 0.5/ We can see that decision tree is both fast in train time and predict time for same amount of data. Combining with the stable performance from the above table, it is makes sense to me to first try out the simple decision tree when getting a new dataset to have a rough idea on how the bottom-line learning cost would be like.

V: Programming language and software packages used:

Packages for machine learning methods used are borrowed from scikit-learn in Python. I used jupyter notebook to test code and relevant modifications and put them in python file to submit.