Karthik Kumar
kkumar33@gatech.edu

Supervised Learning

INTRODUCTION

In the report, I will explore several techniques of Supervised Learning by applying it two interesting classification problems. The two classification problems I chose were: “Identify NBA All-Stars based on player statistics” and “Predict Student Performance (Pass/Fail)”. The techniques include: Pruned Decision Trees, K-Nearest Neighbors (k-NN), Boosted Decision Trees, Artificial Neural Networks (ANN), and Support Vector Machines (SVM). I will apply each algorithm to the two classification problems and will report on the behavior and results that I observed. To conclude, I will compare the performance of each algorithm on each problem and summarize my findings.

The dataset for the first problem is merged from two sources: NBA All Stars from 2000-2016 [1] and season stats for every player from 2000-2016 [2]. As a fan of basketball and the NBA, this dataset was interesting to me for obvious reasons. As a student of Machine Learning, this dataset has some interesting characteristics that made it an appealing choice to investigate. First, the data is heavily unbalanced; of the **8069 players** in the dataset, only **463 players** are actually All Stars (only ~ 0.05%). The number of true negatives greatly outweighed the number of true positives. This allowed me to explore each algorithm’s ability to cope with unbalanced data and to experiment with various techniques of accurately measuring performance. Second, the size of the data was small enough to allow me to quickly run experiments while still giving me results that I could analyze. There are **50 total features**, including total points/assists/rebounds, field goal/free throw/3 point percentages and more advanced stats derived from formulas (like PER, USAG% etc).

The dataset for the second problem comes from a Portuguese study that collected various student attributes (demographic, social and school related) along with their grades for two classes [7]. This data was interesting because of the challenges it posed. First, the dataset is small, containing only **1044 samples** and **33 features**. I wanted to chose a smaller dataset to learn about ways that different algorithms generalize over small amounts of data. Second, some of the features were nominally-typed (for example, the “reason” feature could have the values “home”, “reputation”, “course” or “other”). This required me to use an encode the categorical features that don’t have a quantitative relationship between them. I used scikit-learn’s `OneHotEncoder` to do this. This transformation results in **48 total features**. Lastly, the data contained a feature named ‘G3’, which was the student’s final grade (out of 20). Since I was investigating mostly classification problems, I decided to label all grades above the mean grade of (11.34) as Pass (1) and all grades below the mean as Fail (0). There were **534 samples labeled “Fail” and 510 samples labeled “Pass”**, so this dataset was more evenly balanced.

GENERAL APPROACH

For this project, I used Python’s scikit-learn [3] library for running the Supervised Learning algorithms. I used the pandas [4] library for data manipulation and analysis and matplotlib [5] for data visualization and plotting. For each of the five algorithms, I used the same general procedure for my analysis. My analysis included: a Model Complexity Curve (also called Validation Curve) to help tune hyperparameters, a Learning Curve to find the lower bound on the number of samples needed to learn this model and to investigate any issues due to high bias or high variance, a timing curve to investigate the runtime performance of training and testing and, for some algorithms, an iteration curve to investigate accuracy as the number of iterations increased. At a high level, here is the general methodology I followed when analyzing each algorithm:

1. Split 80% of the dataset into a training set and 20% into a testing set, using a stratified split
2. Standardize features by removing the mean and scaling to unit variance
3. Of the 80% of training set, use a Model Complexity curve to find the best hyperparameters for tuning the model (with 5-fold cross validation)
4. Of the same 80% training set and using the best estimator identified in Step 2, plot the Learning Curve for the model (with 5-fold cross validation, using 20% of data incrementally cumulated)
5. Calculate the training accuracy of the model on the 80% that dataset from Step 1

6. Calculate the test accuracy of the model on the 20% of the dataset held out in Step 1
7. Plot the iteration or timing curves, as necessary, depending on the model

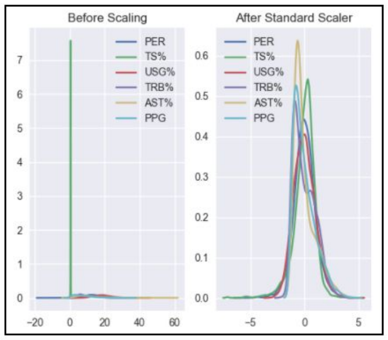


Figure 1: StandardScaler applied to sample attributes

In Step 1, the initial split is done using stratified sampling. This is done to ensure that the split contains the same percentage of each classification (for ex: All Stars and Non-All Stars) in both the training and test sets. Step 2 is done using scikit-learn's StandardScaler to ensure that each attribute's variance is in the same order and that the mean is centered around 0. This is required for certain models like SVMs and k-NNs (to ensure that a certain attribute with a large variance doesn't dominate the objective functions). See Figure 1 for an example of the StandardScaler applied to a few sample attributes from the NBA dataset. Steps 3 and 4 both use built-in functions for plotting the Validation and Learning Curves. My experiments used a custom sample weight scoring function that automatically adjusts weights inversely proportional to class frequencies in the input data. This is done to ensure that the unbalanced dataset can be accurately scored. Steps 5 and 6 also used a similar "balanced" sample weight scoring function to account for unbalanced data. Additionally, my experiments used 5-fold cross validation to prevent overfitting on the training data, at the cost of increased run times.

DECISION TREES

The first algorithm I investigated was Decision Trees (DT). DTs classify samples by inferring simple decision rules from the data features. They build a tree structure where each internal node represents a if/then rule based on some attribute. Classification of new samples is done by traversing the tree and using the leaf node values. Some advantages of DTs are that they are easy to understand, implement and visualize. One big disadvantage of DTs is that they are prone to overfitting on the training data; overfitting occurs when the algorithm builds a tree that performs really well on the training data, capturing all the noise and unwanted features. For my experiments, I avoided overfitting by pruning (specifying the `max_depth`) the tree. In scikit-learn's `DecisionTreeClassifier` class, I used the Gini Impurity as the function to measure the quality of a split and specified a balanced class weighting measure. All other parameters used the defaults provided by scikit-learn.

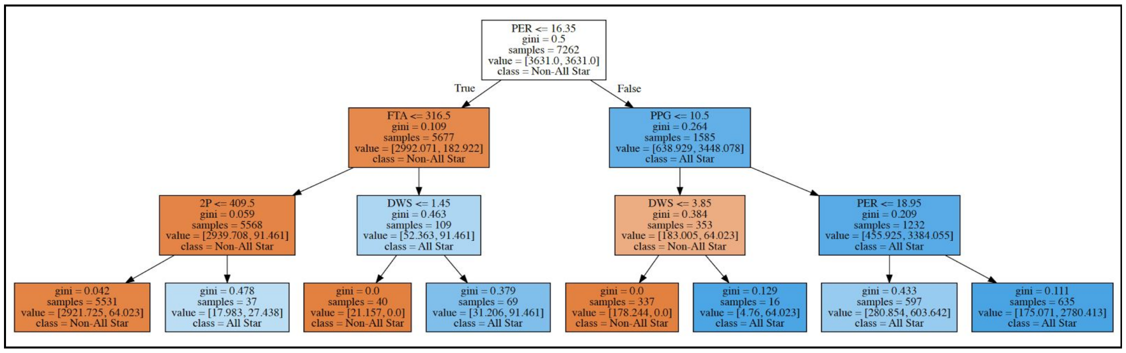


Figure 2: 3-level Decision Tree generated for classifying NBA All Stars

I applied DTs to the NBA All Star classification problem. See Figure 2 for the final 3-level DT generated for the dataset I supplied. I was surprised to see that only 5 attributes of the 50 were considered in the tree: PER (Player Efficiency Rating). FTA (Free Throws Attempted), PPG (Points Per Game), 2P (Two Pointers Made), DWS (Defensive Win Shares). I decided on using a tree with only 3 levels after generating the Model Complexity Curve. See Figure 3 for those results. The results in that graph demonstrate that using a tree of `max_depth` of 3 generated the highest cross validation score. As we increase the depth of the tree, we see that the training score continue to improve while the validation score declines. This demonstrates that **adding more levels to the tree leads to overfitting** on the training data, which causes the tree to not generalize well on unseen data.

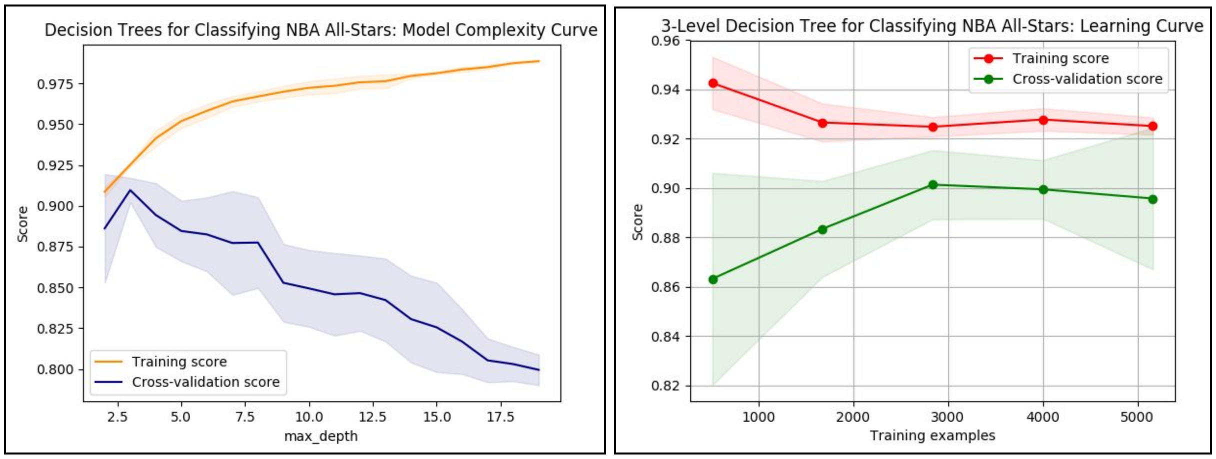


Figure 3: Decision Tree Model Complexity and Learning Curves for Classifying NBA All-Stars

Next, I ran a Learning Curve experiment on the 3-level Decision Tree. The results are captured in Figure 3. This graph shows that both the training and cross-validation scores converge to around the same value: 90-92% accuracy. This demonstrates that my Decision Tree model is ideal, in that it **does not suffer from high bias or variance**. Instead, this model will generalize well to previously-unseen testing data which is confirmed in the final testing results. **The final training accuracy was 92.4931% and testing accuracy was 89.2402%.** Additionally, this model required only about 3000 samples to achieve an accuracy of around 90%, which shows that this model generalizes well without requiring too much more data.

I applied the same algorithm and analysis to the Student Grades problem. See Figures 4 for the Model Complexity and Learning Curves. For this problem, the optimal `max_depth` of the DT was 5 levels. This was based on the validation accuracy and the desire to reduce the model's variance (since pruning the DT reduces variance). The learning curve shows that this model has **low variance but high bias**, since the training and validation curves converge towards a common accuracy value but that accuracy value is low (~70%). **The final training accuracy was 72.4468% and testing accuracy was 69.7269%**

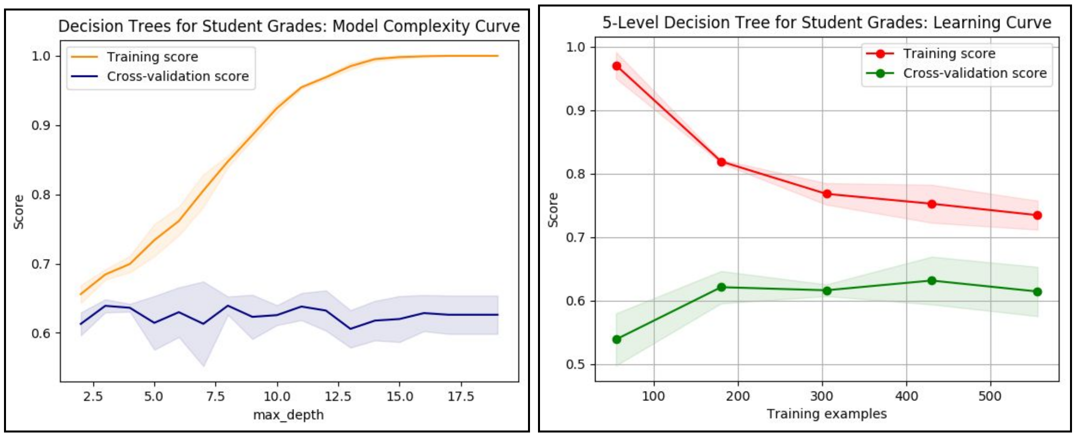


Figure 4: Decision Tree Model Complexity and Learning Curves for Classifying Student Grades

Figure 5 show the timing curves for training vs prediction. Both show that Decision Trees are an **eager learning** model, in that the model spends a linear amount of time learning from the model and returns classifications in constant time.

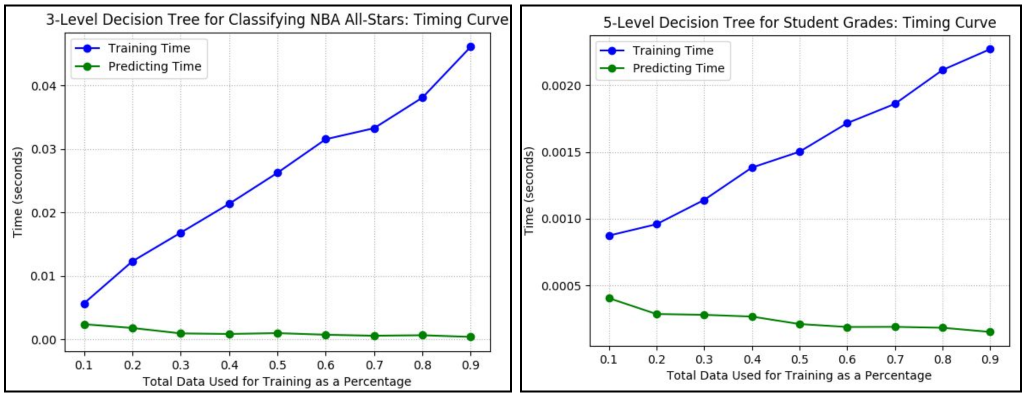


Figure 5: Training and Predicting Times of a 3-level Decision Tree on both problems

k-NEAREST NEIGHBORS

K-Nearest Neighbors is an instance-based learning algorithm. Rather than constructing a generalized model of the classification problem, it stores all training samples and classifies the testing data by taking a majority vote of the `k` nearest neighbors to the query point. Overfitting can occur in k-NN when we use a k value of 1. When k=1, each sample is in a neighborhood of its own and results in a model with **low bias and high variance**. As we increase k, we reduce the complexity in the model and we also reduce overfitting. In my experiments for k-NN, I varied the k parameter and used uniform weights for computing the majority votes for the nearest neighbors.

See Figure 6 for the Model Complexity curve for the NBA All-Stars problem, which shows that **overfitting occurs when k=1 and decreases as k increases**. The peak validation accuracy I saw that was when k=3, so I used this as the basis for the learning and for computing testing accuracy. **Increasing k decreases variance and increases bias**, so in choosing k=3, I prioritized bias (higher accuracy) over variance. Note that this model is also unaffected by the unbalanced nature of the NBA data, since it only considers a few closest samples for classification. The Learning Curve in Figure 6 shows that this model isn't as good at classifying NBA All-Stars as Decision Trees were. As we give the model more training examples, the accuracy scores continue to increase and there is a large gap between the training and validation scores. This shows that the model suffers from **high variance, which means more training data or a larger k value** will help improve the model's performance. **The final training accuracy was 84.5761% and testing accuracy was 76.4208%**. As expected, the accuracy is lower than that of Decision Trees.

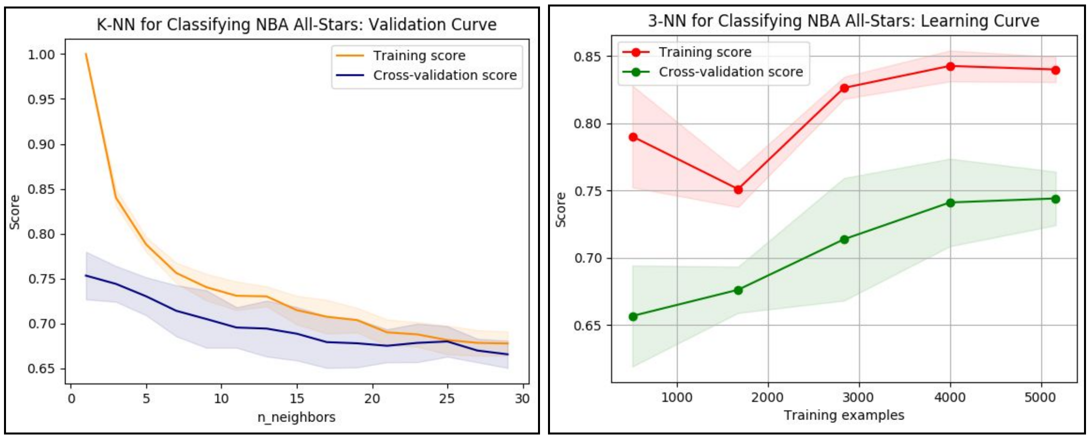


Figure 6: k-NN Model Complexity and Learning Curves for Classifying NBA All-Stars

Next, I ran the k-NN algorithm on the Student Grades data. See results in Figure 7. Once again, we see overfitting when k=1 and the accuracy score staying flat as k increases. I chose k=7 as the value for the Student

Karthik Kumar
kkumar33@gatech.edu

Grades model. Unfortunately, we didn't see very good performance with this dataset. As the learning curve in Figure 7 shows, this model suffers from **high bias** since the validation accuracy is low. The model also has **high variance** since the accuracies do not converge and there is a large difference in their values.

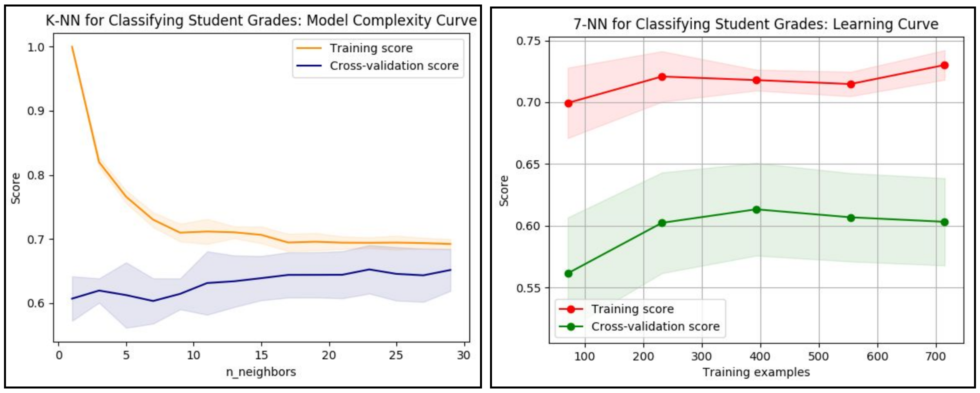


Figure 7: Model Complexity & Learning Curves for k-NN model Classifying Student Grades

The final training accuracy was 73.2226% and testing accuracy was 69.5437%. This model performs with about the same level of accuracy as the Decision Tree model for student grades. See Figure 8 for the timing curves, which show that k-NN spends a constant time learning and more time during prediction. This indicates that k-NN is **lazy learner**, which puts off the time-intensive work until query time.

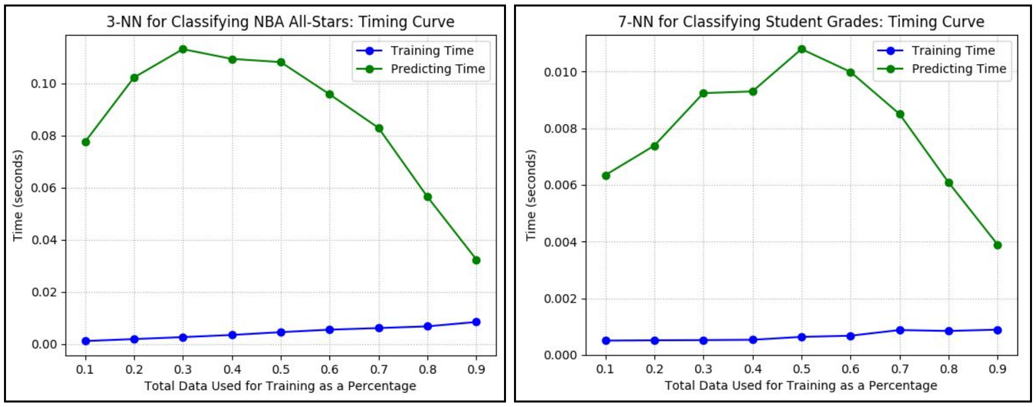


Figure 8: Training and Predicting Times of Nearest Neighbors Model on both problems

BOOSTING

Boosting is an ensemble learning technique that combines the predictions of several weak learners to produce a generalized prediction. Weak learners are models whose predictions are only slightly better than choosing at random. The final prediction is usually done by a weighted majority vote of all estimators. For my experiments, I used a specific boosting algorithm, AdaBoost. AdaBoost iteratively applies learning process to a distribution of weights. At each step, the training examples that were classified incorrectly are weighted higher for the next iteration and the examples that were classified correctly are weighted lower. For my experiments, I used a single-level Decision Tree (also known as a decision stump) as the base estimator for AdaBoost. I also experimented with various number of estimators to identify the best estimator count for my model. One interesting observation I had was that using a Decision Tree of greater depths (3 or greater) caused my boosting algorithm to underperform. After some research and reasoning, I discovered that the **larger, more complex underlying decision tree was overfitting, causing boosting to also overfit**. Using a less complex base estimator, one that was less likely to overfit, improved the performance of the boosted model.

See Figure 9 for the Model Complexity curve for Boosting. Here, we see that varying the `n_estimators` hyperparameter caused the training curve to continuously increase while keeping the validation curve relatively steady. This graph shows that overfitting is likely as the number of estimators grows. I chose 18 as the optimal number of estimators to use with the base estimator of a 1-level decision tree. In Figure 9, we also see the that the training and validation scores converge to similar values as the size of the training set increases. This shows that **our model has low bias and low variance so it generalizes well** to new data. **The final training accuracy was 92.3338% and testing accuracy was 87.7099%**. While it performed better than k-NN, the Boosting model had around the same accuracy score as a 3-level Decision Tree. This could be because of outliers (All-Stars that didn't have good stats but were voted in due to popularity) in the data of which Boosting is maximizing the importance.

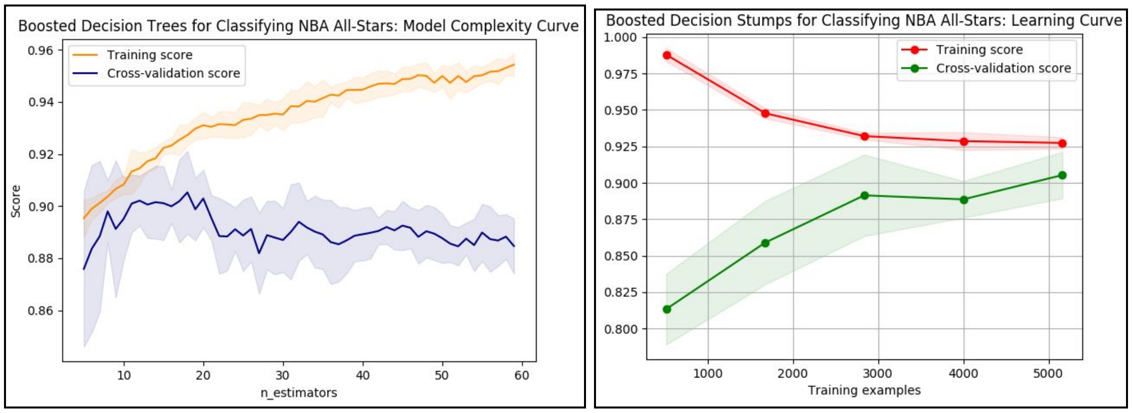


Figure 9: Boosting Model Complexity and Learning Curves for Classifying NBA All-Stars

Next, for the grades classification problem, I applied the Boosting model using similar process. The results are shown in Figure 10 below. The Model Complexity curve shows that overfitting occurs when the n_estimators is greater than 20. I chose a value of 13 for plotting the learning curve and measuring test accuracy of the model. The learning curve shows that this model has **low variance** (accuracy scores converge) but has **high bias** (low accuracy overall). **The final training accuracy was 71.2592% and testing accuracy was 74.8422%.**Note that the test accuracy is actually higher than training accuracy! This could be because the model is capturing some noise in the training data that is not present in the testing data.

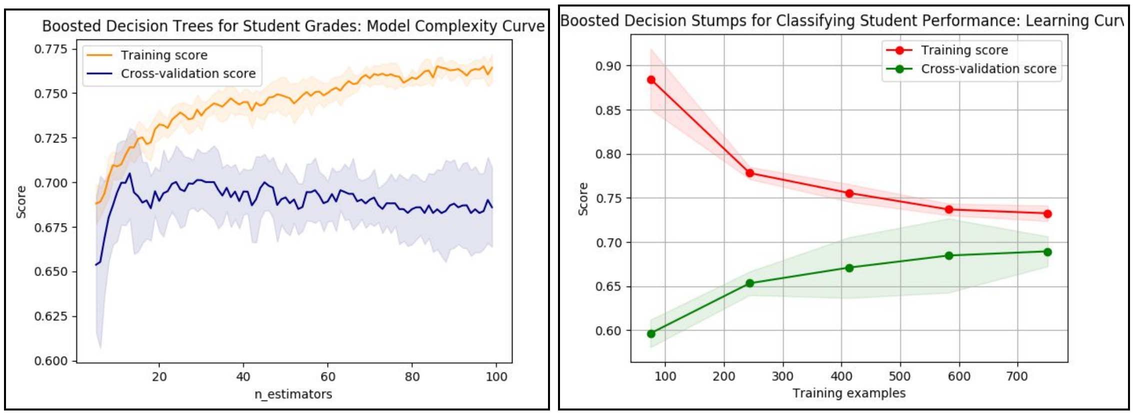


Figure 10: Model Complexity and Learning Curve for Boosting model for Classifying Student Grades

The timing curves in Figure 11 shows that boosting is a **eager learner** since prediction takes constant time and training takes is O(n). For the Student Grades problem, the dataset is so small that even training takes almost constant run time.

Karthik Kumar
kkumar33@gatech.edu

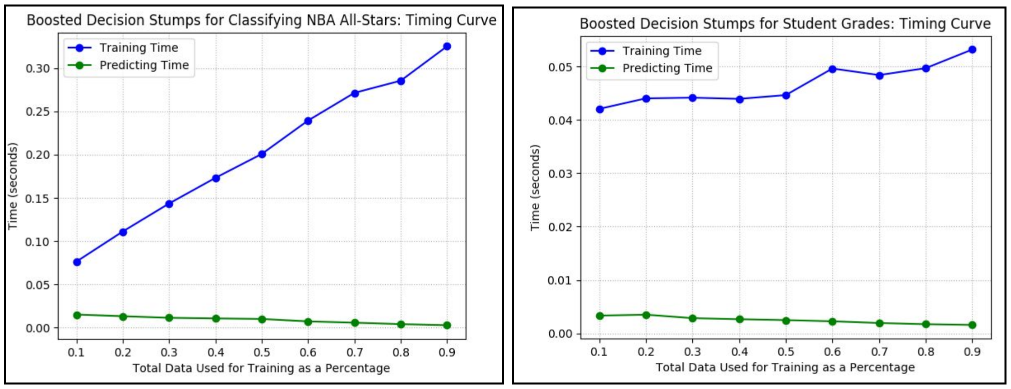


Figure 11: Timing curve for Boosted Decision Stumps on NBA All Star Classification Problem

ARTIFICIAL NEURAL NETWORKS

Artificial Neural Networks (ANN) learn a nonlinear function using three types of connected nodes: input, hidden and output. These nodes are connected in a manner similar to neurons in the human body. The algorithm takes input features in the leftmost input layer, each neuron in the hidden layer(s) transforms the output of the previous layer, then the hidden layer computes a weighted linear summation and applies a non-linear activation function. Lastly, the output layer gets the values from the last hidden layer and transforms it into output values. These output values are compared against the actual known output value and the error value is bubbled back through the layers so subsequent iterations can adjust the weights to continuously reduce error. In scikit-learn, I used the `MLPClassifier` (Multilayer Perceptron), which uses the Backpropagation algorithm described above. I experimented with changing the `alpha` hyperparameter. I used the default hidden layer configuration of one hidden layer with 100 units for the NBA dataset, since my classification problem was not too complex. For the Student Grades dataset, I used two hidden layers with 5 units each. This was because using the default 100 units caused my model to result in 100% training accuracy, which meant that large number of neurons caused drastic overfitting for this model.

`Alpha` is a penalty term that constrains the size of the weights. Increasing alpha can reduce high variance in the model, which results in fewer curves in the decision boundaries. Decreasing alpha can help reduce high bias in the model, resulting in a more complex, curvier decision boundary. Figure 12 below shows the Model Complexity curve for varying values of alpha for the NBA problem. As alpha increases, the accuracy scores remain consistent, but after alpha is greater than 0.1 the accuracy declines, which shows that using an alpha greater than 0.1 will result in overfitting.

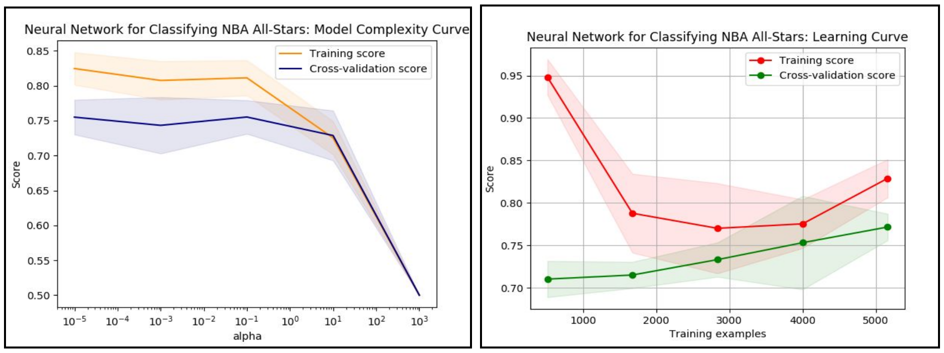


Figure 12: Model Complexity and Learning Curves for Neural Network for Classifying NBA All-Stars

For the learning curve, I used alpha=10^-5 as the hyperparameter, since having a less complex model is preferable to a more complex one (**Occam's Razor**). The learning curve shown in Figure 12 exhibits **high variance** (since the validation accuracy is increasing with more data). This suggests that either giving this model more data or reducing the number of features might improve accuracy. The model also shows **high bias**, since the accuracy scores are relatively low. One option to reduce the high bias might be to get new features from another source that might help improve accuracy. **The final training accuracy was 84.5597% and testing accuracy was 76.8251%.** This model showed similar accuracy scores as KNN but performed worse than decision trees or boosting.

For the Student Grades classification problem, I saw similarly shaped curves. For the Model Complexity curve for alpha, overfitting occurs after an alpha of 10**0. I decided to use an alpha of 10**-5 and plotting the learning curve. See Figure 13 for that result. Unfortunately, this model suffers from **high bias and high variance** (low accuracy and score does not converge). Perhaps adding more data would help improve accuracy. **The final training accuracy was 78.1168% and the testing accuracy was 76.2527%**. The testing scores for the Student grades problem was highest when using Neural Networks. This could be because Neural Networks with two hidden layers are able to capture complex relationships in the dataset.

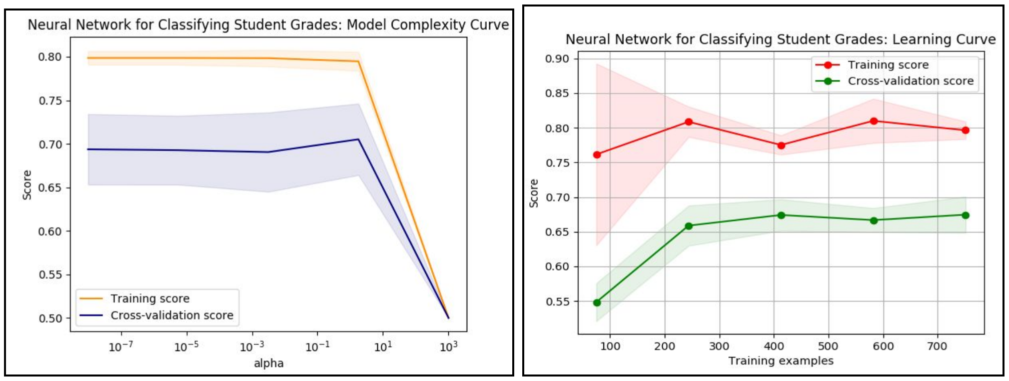
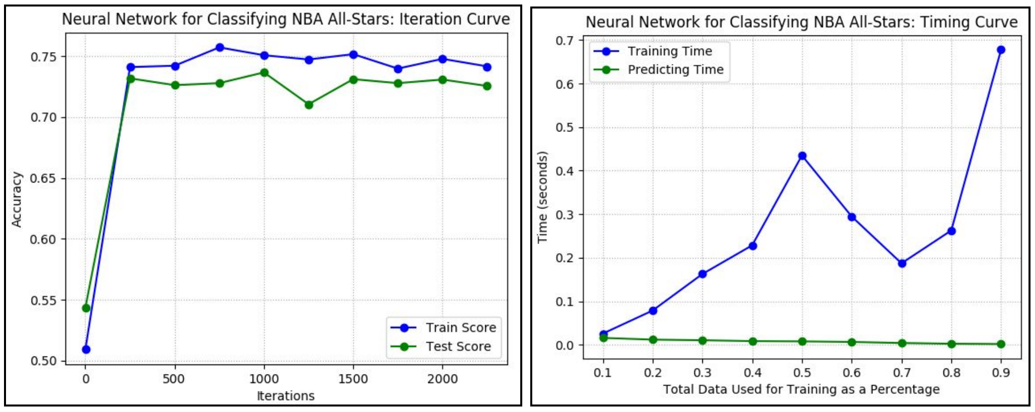
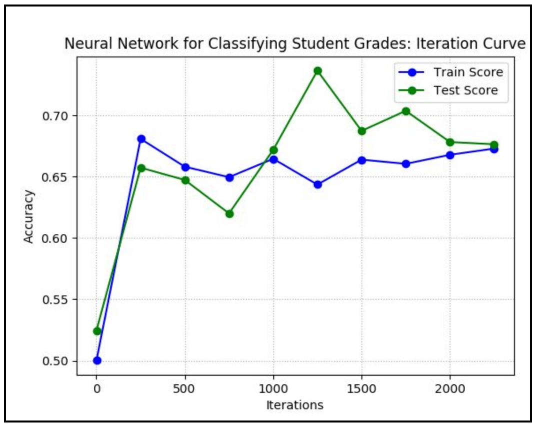


Figure 13: Model Complexity and Learning Curves for Neural Network model of Classifying

Figure 14 below shows the iteration and timing curves for Neural Networks. Here we see at we need at least 500 iterations of the algorithm to be able to classify data with a better accuracy than 50% for the NBA problem. Similarly, for the Student Grades problem, we need more than 500 samples before making semi-accurate predictions. We also see that since Neural Networks are a **eager learner**, more time is spent in training (O(n)) than during prediction (O(1)) for both problems.



Karthik Kumar
kkumar33@gatech.edu



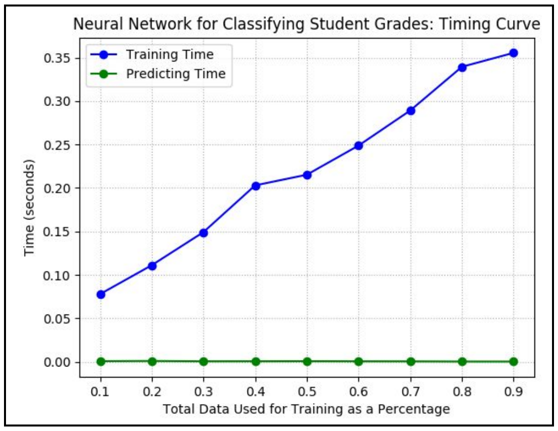
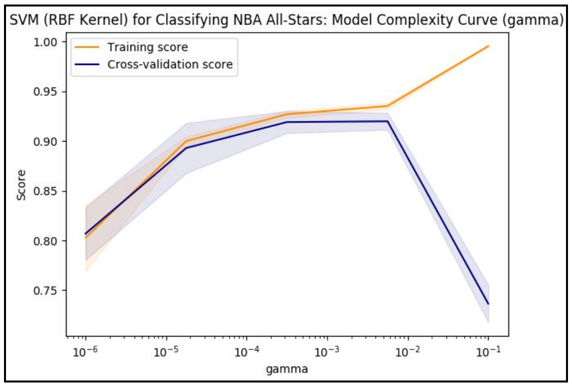


Figure 14: Iteration and Timing Curves for Neural Networks for both problems

SUPPORT VECTOR MACHINES

Support Vector Machines (SVMs) are a method of supervised learning used for classification problems. It classifies new samples into categories by finding a decision boundary that maximizes the **margin** between two classes. The motivation behind this is that larger margins between the classes reduce the generalization error of the classifier. For my experiments, I used the `SVC` class in scikit-learn and tested with two different kernels: `sigmoid` and `rbf`. Kernel functions are used to measure similarity between two data points. Kernel functions allow us to find boundaries between two classifications, sometimes in higher dimensions (using the Kernel trick). I experimented with the `rbf` (radial basis function) and the `sigmoid` functions which are good when the data isn't linearly separable. I found optimal values for both the `C` and `gamma` hyperparameters. `C` is the penalty for a misclassification, as C increases, variance increases and bias decreases. `Gamma` is a property for the kernel function and decides the "curviness" or the shape of the decision boundary. See Figure 15 for the model complexity curves for the rbf kernel function when varying both C and gamma.



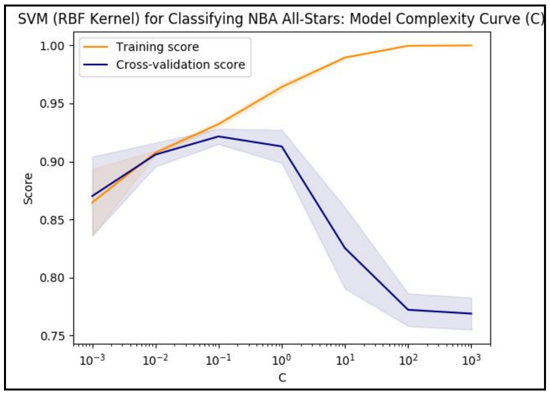


Figure 15: Model Complexity curves for SVM (with RBF Kernel) for increasing values of gamma and C

The above models show that for rbf, overfitting occurs when gamma is greater than 0.005 and when C is greater than 0.1. This is apparent because the training scores continue to grow while the validation score declines sharply after these points. See Figure 16 for similar curves for the sigmoid function. In the sigmoid graphs, there was some weirdness observed. Both the training and cross-validation scores mirror each other almost exactly. Also, the model complexity curve for varying C closely matches that of an inverse sigmoid function. Through my (limited) research, I wasn't able to find a satisfactory explanation for this behavior. I decided to use the rbf kernel function with C=0.1 and gamma= 0.005 for my learning curves.

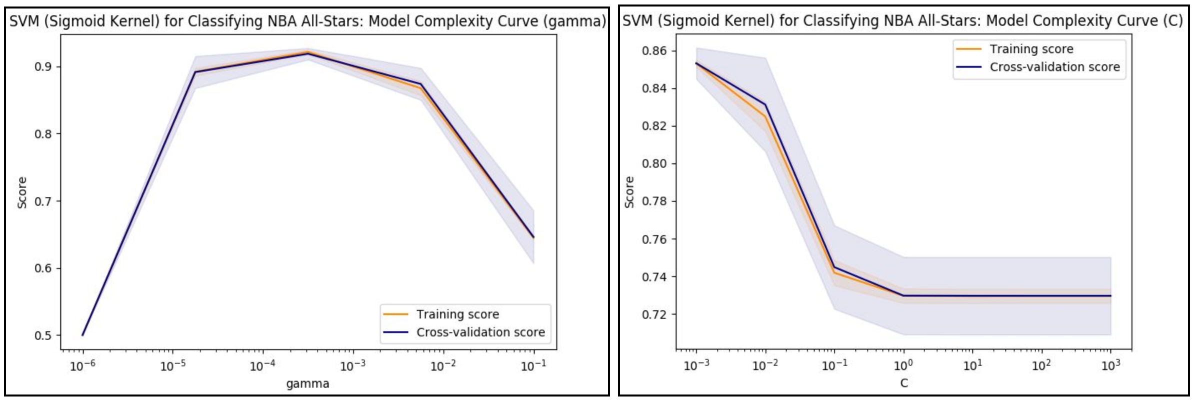


Figure 16: Model Complexity curve for SVM (with Sigmoid Kernel) for varying values of gamma and C

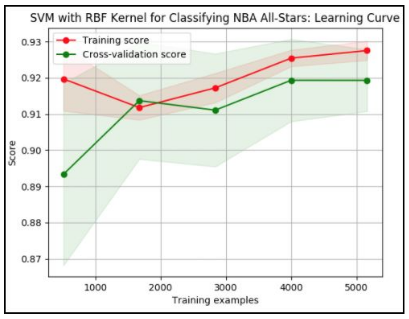


Figure 17: SVM (rbf kernel) Learning Curve for Classifying NBA All-Star

In Figure 17, we see the learning curve observed for the SVM model. This model has **low variance** (small gap between training and validation scores) and **low bias** (high accuracy score). **The final training accuracy was 92.8699% and the testing accuracy was 89.3930%**. This model shows about the same performance as the 3-level decision tree model. One interesting feature of this learning curve is that around 2000 training samples, the validation score actually exceeds the training score. This could be due to a coincidence from the random splits generated by the cross validation process that generated this graph.

For the student grades problem, I once again used the rbf and sigmoid kernel functions, since it wasn't clear that the data was linearly separable. One difference for this experiment was that I only varied C, instead of gamma. I relied on the default value of 'auto' for gamma, which uses 1/n features. This was done because I didn't have enough domain knowledge to know what the shape of the decision boundary should be like. See Figure 18 for the two Model Complexity curves. For the rbf kernels, overfitting occurs when C is greater than 10. The sigmoid kernel shows an unexpected curve since the validation accuracy is higher than the training accuracy. I attributed this to the lack of data for this problem and decided to use the rbf kernel for plotting the learning curves and measuring overall accuracy. Figure 19 shows the learning curve for the SVM model for predicting student grades. This model suffers from high variance, since there is a large gap in the errors observed; getting more data might help alleviate this issue. **The final training accuracy was 86.1442% and the testing accuracy was 78.4680%.**

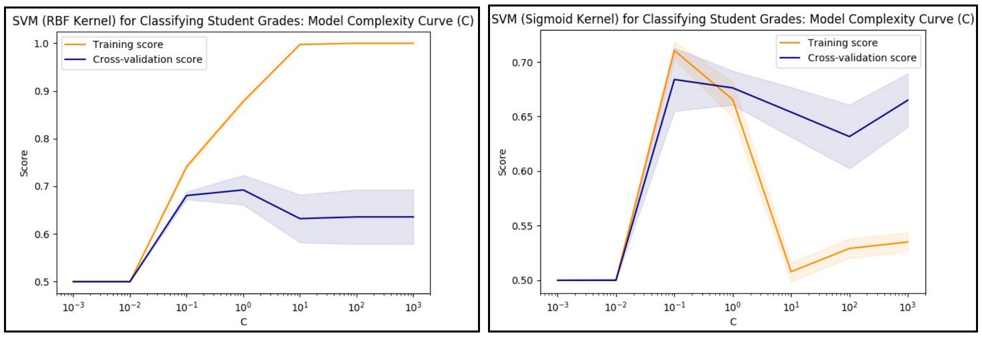


Figure 18: Model Complexity Curves for rbf and sigmoid kernels for Student Grades problem

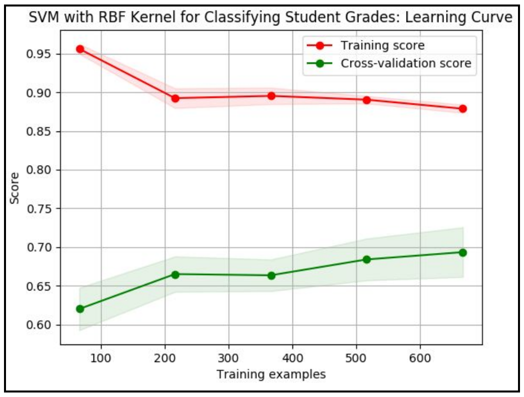


Figure 19: Learning curve for SVM model for Predicting student grades

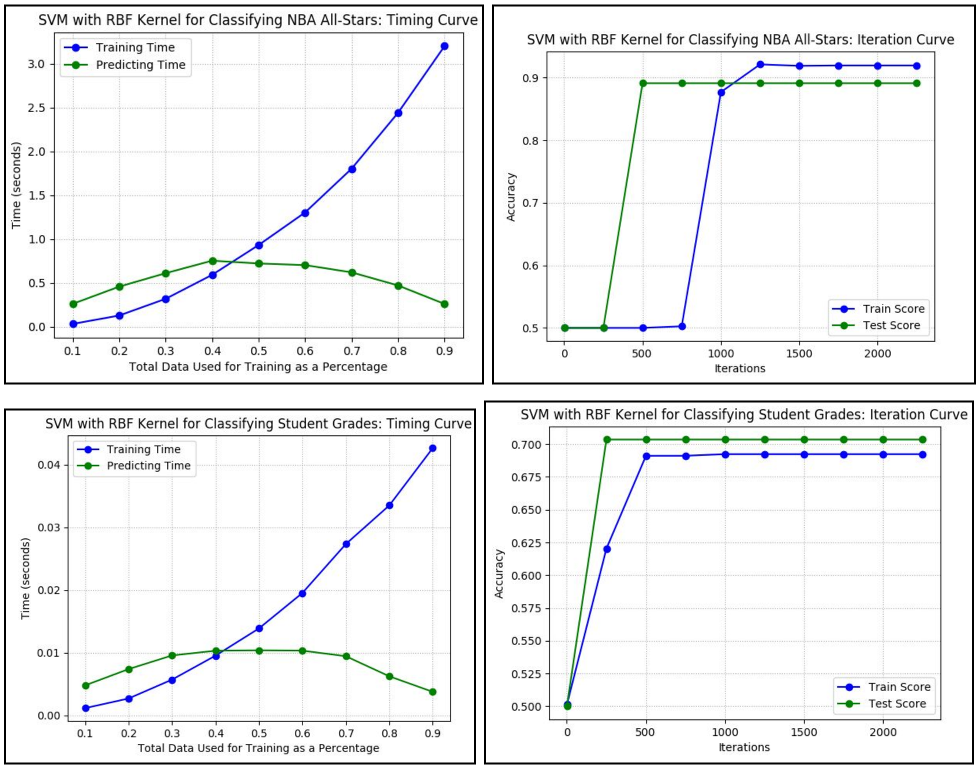


Figure 20: Timing and Iteration Curves for SVM with RBF Kernel for both problems

The timing curves for SVM in Figure 20 show a O(n^2) algorithm for training, which matches scikit-learn's documentation on the complexity of SVMs [6]. Since the core of an SVM model is a Quadratic Programming problem, this runtime is to be expected. As for the iteration curve, it shows that SVM performs poorly for the NBA problem (around 50% accuracy) when the number of iteration is lower than 1000. Once we exceed 1000 iterations of the algorithm, the model reaches a steady state for the accuracy. Also, it's interesting to note that the test score requires fewer iterations to reach >88% accuracy and the training score requires more iterations. This could be because of the disparity in the size of the training vs testing samples. For the Student grades problem, at least 500 iterations are needed to achieve a steady state accuracy measure.

CONCLUSION

To conclude, in this paper, I analyzed the performance of five different algorithm on two classification problems. I saw varying results for each problem and I learned more about the tools available in a specific library (scikit-learn). One improvement I could have made to my analysis is to use a grid search method of finding optimal hyperparameters for a certain model. In my analysis above, I only modified one or two hyperparameters for each algorithm. If I had more time (or more pages to work with :)), I would have explored using more non-default hyperparameters for each algorithm

For the NBA All-Star classification problem, I saw a high accuracy score across all the models. Decision Trees, Boosting and SVMs had the best performance, while k-NN and Neural networks were slightly worse at predicting All-Stars. DTs and Boosting have the ability to pick the most relevant features when generalizing, whereas k-NN and Neural Networks weigh each feature equally; noise from irrelevant features might be one reason to explain this reduced accuracy. Additionally, since NBA All-Stars are voted in by fans, a player's popularity also plays a factor in predicting whether or not they would selected as an All-Star. If we had new feature that quantified their popularity, like "Twitter Followers" or something similar, we might be able to achieve higher accuracy across all the models. See Figure 21 for a summary of the results.

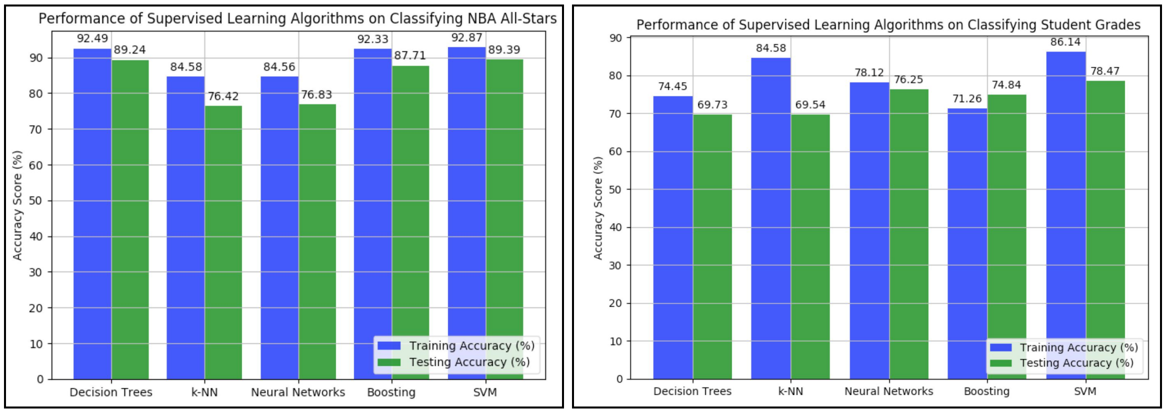


Figure 21: Comparing different models for both problems

For the Student Grades classification problem, we saw much lower accuracy across all models. Neural Networks and SVMs performed best for this problem, which might indicate that the relationships in this dataset require a complex model to generalize. Another reason for the low accuracy could be because the amount of data we had was not enough to accurately generalize. Since the number of samples was only around 1000 and we had 48 total features (including encoded features), this problem might be suffering from the **Curse of Dimensionality.** Providing more data or reducing the number of features might help increase accuracy. Additionally, there might actually be no correlation between the features we were given and a student's grade. While we did see accuracy scores around 75%, we need more domain knowledge to be able to verify that the models we created are actually generalizing on the dataset.

REFERENCES

[1] https://data.world/gmoney/nba-all-stars-2000-2016
[2] https://www.kaggle.com/drgilermo/nba-players-stats/data
[3] http://scikit-learn.org/stable/
[4] https://pandas.pydata.org/
[5] https://matplotlib.org/
[6] http://scikit-learn.org/stable/modules/svm.html#complexity
[7] P. Cortez and A. Silva. Using Data Mining to Predict Secondary School Student Performance. In A. Brito and J. Teixeira Eds., Proceedings of 5th FUture BUsiness TEChnology Conference (FUBUTEC 2008) pp. 5-12, Porto, Portugal, April, 2008, EUROSIS, ISBN 978-9077381-39-7.