Michael Tong Assignment 1 ID: 903357308
CS 7641 – Spring 2019 mtong31@gatech.edu

Abstract: The following report summarizes a decision tree, gradient boosted decision tree, multilayer perceptron, support vector machine, and k-nearest neighbor classifier results on two distinct datasets. One dataset involves records of students, high level characteristics about each of them, and their performance on an exam. Through these metrics, this study demonstrates the ability to implicitly identify a person's gender. The other dataset contains information on a large portion of residential houses in Washington, District of Columbia. With housing characteristics, location, and numerous other factors, classification is performed by predicting which price group a particular house belongs in. Each of the classifiers will be manually hypertuned and evaluated.

Selection Criteria

The datasets chosen for this assignment were selected to satisfy both the classification requirement, and arguably more important, an opportunity to explore a field of interest. Three separate datasets were selected, where two are merged together to increase the density of features. The first is a simple and clean, 1K x 8 dataset on student test performance with supplemental information on the student's characteristics.[1] Complementing this smaller dataset is two larger ones identifying Washington, DC property information, and the other detailing individual property characteristics. This data is 147K x 52 and 107K x 39 respectively. [2]

The student test performance dataset was selected to be a simpler test set for the algorithms prior to applying it to the larger dataset but is also interesting to see the possible relationships between a person's characteristics and their scores on three generic tests for math, reading, and writing. The dataset contains 5 categorical features, namely gender, race/ethnicity, parental level of education, their lunch that day, if they took a prep course, and three discrete values, namely scores for math, reading, and writing. After cleaning and categorical encoding, the dataset becomes 1K x 19. Binary gender classification is chosen as the target output for this dataset.

Moving to DC in July 2018 inspired the selection for the DC housing datasets as an opportunity for understanding more about my new home. The first and larger of the two contains high level information about a building, such as the lat/long, address, type of building, etc., containing 37 categorical, 12 continuous, and 3 discrete values. The second and smaller housing dataset contains building specific information, such as the type of heating, number of bedrooms, roof style, etc. containing 13 categorical, 19 continuous, and 7 discrete values. Both of these datasets are joined on the SSL feature, which is a designation for a specific building's location denoted as Square Suffix Lot. Post data cleaning and encoding results in a dataset of size 58K x 287. Price of the residential home is selected as the target output which is divided into three different bins, chosen specifically to separate the classes evenly; these values are $300K and $640K. Code for cleaning is provided but will not be explained in this analysis in the interest of time.

Michael Tong Assignment 1 ID: 903357308
CS 7641 – Spring 2019 mtong31@gatech.edu

General Analysis

For both these datasets, five classification algorithms, specifically a decision tree, neural network (MLP), boosted decision trees (XGB), support vector machine (SVM), and k-nearest neighbor (KNN), are applied to test their classification performance. Nearly all algorithms, except the boosted decision tree, is implemented through scikit-learn in Python, and the boosted decision tree is implemented through xgboost in Python.[3,4] Default parameter classification results are displayed in Table 1 below as a rough estimate for initial comparison.

K-fold cross validation (CV) is used as a means of reducing variance in the accuracy results, and the "best" model is chosen as the highest accuracy by this metric. For this particular study, since compute time is a non-factor, accuracy of our classification prediction is objectively of high importance. As such, the classifier with the highest cross validation accuracy is highlighted in green in Table 1, representing the metric we want to maximize

Student Dataset	Train time (s)	Test time (s)	10-Fold Cross Val	Training Acc	Training F1	Testing Acc	Testing F1
Decision Tree:	0.0043	0.0009	0.8199	1.0000	1.0000	0.8250	0.8250
XGBoost:	0.1126	0.0041	0.8759	0.9488	0.9487	0.8450	0.8451
SVM:	0.0312	0.0205	0.8669	0.9163	0.9162	0.8500	0.8499
MLP:	6.3323	0.0024	0.8770	0.9025	0.9025	0.8750	0.8752
KNN:	0.0017	0.1052	0.6560	1.0000	1.0000	0.6350	0.0.6351

Housing Dataset	Train time (s)	Test time (s)	10-Fold Cross Val	Training Acc	Training F1	Testing Acc	Testing F1
Decision Tree:	5.0000	0.1714	0.7388	1.0000	1.0000	0.7365	0.7383
XGBoost:	68.4875	0.5855	0.7912	0.8048	0.8048	0.7857	0.7862
SVM:	164.7695	125.9789	0.7546	0.8290	0.8297	0.7502	0.7517
MLP:	249.6069	0.1009	0.7591	1.0000	0.9990	0.7580	0.7579
KNN:	1.5665	114.3393	0.5935	1.0000	1.0000	0.5855	0.5854

Table 1: Default parameter classifier results

Each classifier is analyzed and discussed in detail in the follow sections, but preliminarily, we find that the two datasets respond very differently to their respective classifiers. Execution times do generally follow expectations, and it's interesting to see the contrast between the decision tree and SVM for the housing dataset, where the ~2% increase in 10-fold cross validation accuracy may not be worthwhile when considering the compute time difference. Interestingly, the F1 scores are consistent with the accuracy scores, meaning that the recall is fairly proportional with accuracy. Form these results however, it's clear that the housing dataset is more difficult to classify, even with significantly more features. Let's now analyze how these classifiers perform with tuned parameters!

Michael Tong
CS 7641 – Spring 2019

ID: 903357308
mtong31@gatech.edu

Decision Tree

Beginning with the decision tree implementation, analysis starts with the learning curve, where the default scikit-learn class parameters are used, splitting on Gini importance, with no max depth, and one sample per leaf. Results for both datasets of varying training sizes are displayed in Figure 1 below, ranging from 0.05 to 0.95 in steps of 0.10.

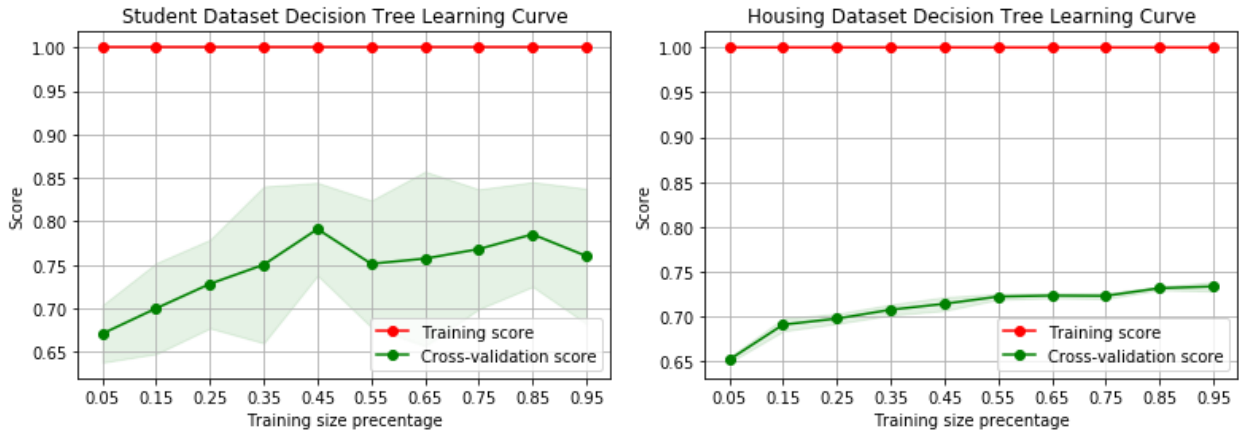

Figure 1: Decision Tree Learning Curves

Since the default parameters allows for unlimited depth and a single sample per leaf, the training accuracy of 100% is expected because the algorithm will split features until everything in the training set is perfectly fit. In the CV score however, it is found that both datasets exhibit similar trends, where accuracy increases as more data is available for training. For the student dataset however, this improvement diminishes around 0.45, likely due to the additional training data not adding any additional information or variance to the model. For the housing data, this stagnation is less apparent, and is likely due to the fact that there are significantly more features in this set, and/or the data used is not fully representative of the target class.

To see if performance can be improved, the analysis now shifts to pruning the decision trees; adjusting parameters in order to modify how they are created. The goal of this process is to improve the testing accuracy of the decision tree, which is 0.825 and 0.737 for the student and housing datasets respectively. For each of these pruning approaches, two splitting criteria are used, Gini importance and entropy. Beginning with the max tree depth parameter, this value limits how many splits the tree can make. By default, this value is infinite, allowing the tree to continuously split values until all the training data is perfectly classified, often leading to overfitting. Results for varying ranges of the max depth parameter are shown below in Figure 2 below.

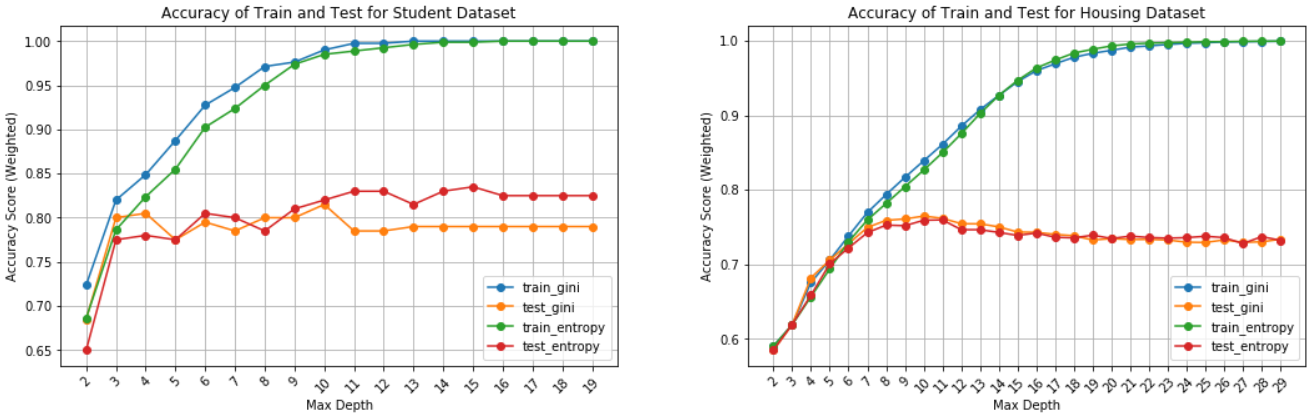

Figure 2: Decision Tree Max Depth Results

Note that the two ranges are different and were selected until both curves appeared to reach a visual asymptote. From this analysis it is found that both dataset's performance improved by limiting the tree depth. For the student data, at a max depth of 15 using the entropy split criteria, we find a test accuracy improvement of 1.2%, and at a max depth of 11 splitting on the Gini importance, we find a test accuracy improvement of 16.8% for the housing dataset.

In addition to pruning the decision tree by max depth, we also analyze the effects of adjusting the minimum number of samples for a leaf. Results are displayed in Figure 3 below.

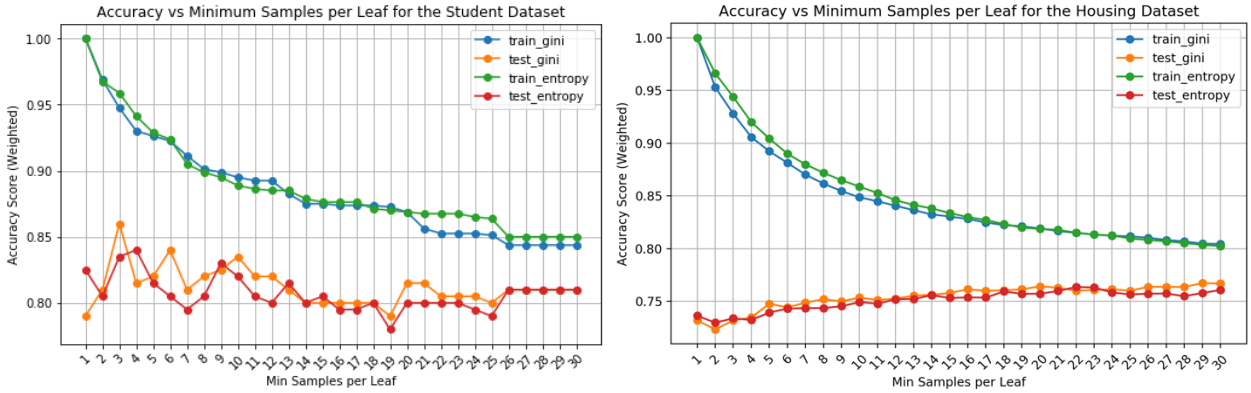

Figure 3: Decision Tree Minimum Samples per Leaf Results

Decrease in training accuracy is expected since we're reducing the algorithms ability to perfectly fit the training samples, but the testing accuracy contrast is particularly interesting. With the student data, we see a rapid increase in accuracy at a minimum sample of 3, then a dramatic decrease until it reaches 25, potentially the result of combating overfitting, then underfitting. With the housing data, the training accuracy exhibits a similar trend, however, testing accuracy improves steady over the entire range, which likely means overfitting was a significant issue prior.

With two pruning methods analyzed, it's fairly certain that the original classifier parameters are overfitting both datasets. Pruning for both datasets clearly shows accuracy improvements.

Boosted Decision Tree

In an attempt to improve upon the decision tree classifier, the gradient boosting method is implemented and analyzed here. This task is performed through the Python XGBoost library, which imputes the gradient boost algorithm through parallel computation, improving efficacy. As seen in the Table 1 comparison matrix, this method potentially increases training and testing time, but greatly improves the 10-fold cross validation error when compared to a traditional decision tree (without pruning). Let's now analyze the affect that different parameters have on performance.

Similar to the decision tree, this analysis begins by analyzing the effect of the maximum tree depth, with results in Figure 4 below. Recall that with a standard decision tree, the accuracy positively increased rapidly, then the testing accuracy begins to decrease gradually, while the training accuracy continues until reaching 100%.

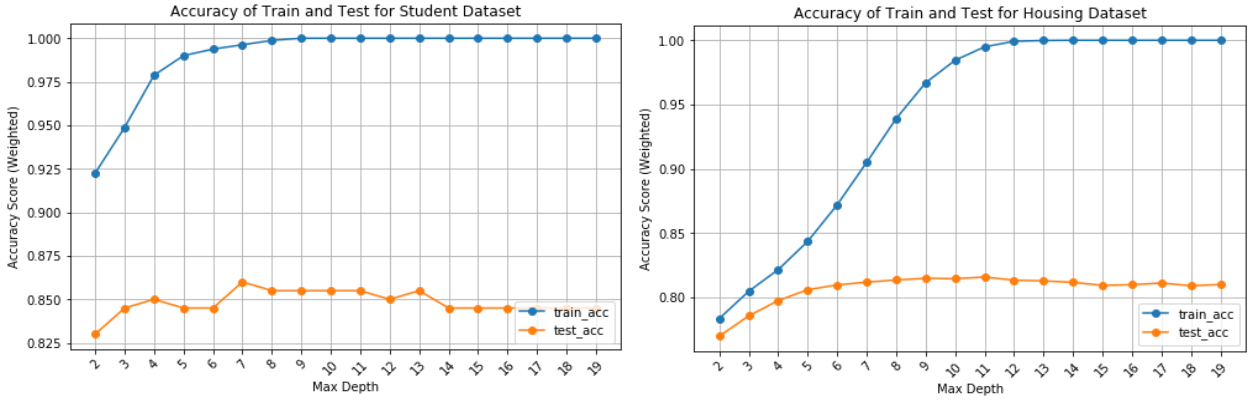

Figure 4: XGB Max Depth Results

To no surprise, we find that the max depth analysis for the boosted decision trees results in a similar trend, finding that the test accuracy improves then drops off as depth increases. This trend is fairly indicative of improving generalization and then overfitting to the training data, especially with the 100% training accuracy indication. It is important to note however, that the decrease in accuracy in the housing dataset is much less prominent than the decision tree analysis. Here we find the difference to be around ~.006%, whereas before it was ~0.3%.

The next parameter analyzed is the number of trees to ensemble, or the *n_estimators* parameter in the *XGBClassifier* class; Figure 5 displays the results. What's immediately noticeable is that both datasets follow similar logarithmic trends increasing rapidly from 10, then plateauing off. These plots are somewhat expected, as the addition of more and more learners will push the average prediction towards some average value, hopefully the true value. What is unexpected is the sudden increase in test accuracy for the student dataset between 450 and 560, before staying constant again.

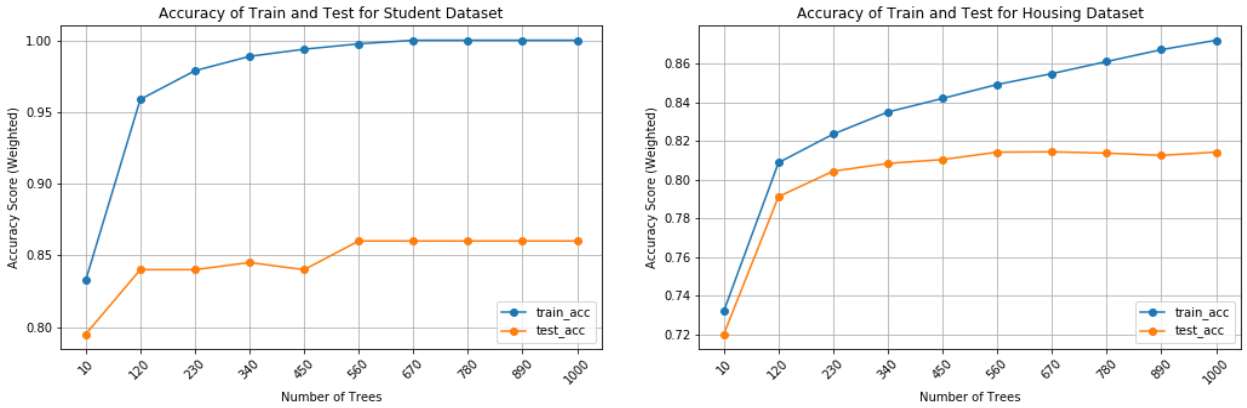

Figure 5: XGB Number of Learners Analysis

From hyperparameter turning, we find that slight improvements can be made to the XGB classification model, but they are not as significant as those found in the decision tree. This is likely due to this model being less prone to overfitting than the decision tree.

Neural Network

Following the decision tree analysis, a neural network, specifically a multilayer perceptron, is implemented and analyzed. Beginning with the learning curves, they are displayed below in Figure 6, which implement the classifier with its default parameters except for the max iterations, specifically 100 hidden units at 1 layer deep, rectified linear unit activation, ADAM solver starting at a learning rate of 0.001, 0.0001 L2 regularization, and auto batch sizing. In order assist in convergence, the number of maximum iterations is set to 1000, which consumes more compute time but improves convergence. Additionally, the input vector is normalized prior to prediction to avoid biases from different value ranges, using the scikit-learn *StandardScaler* class.

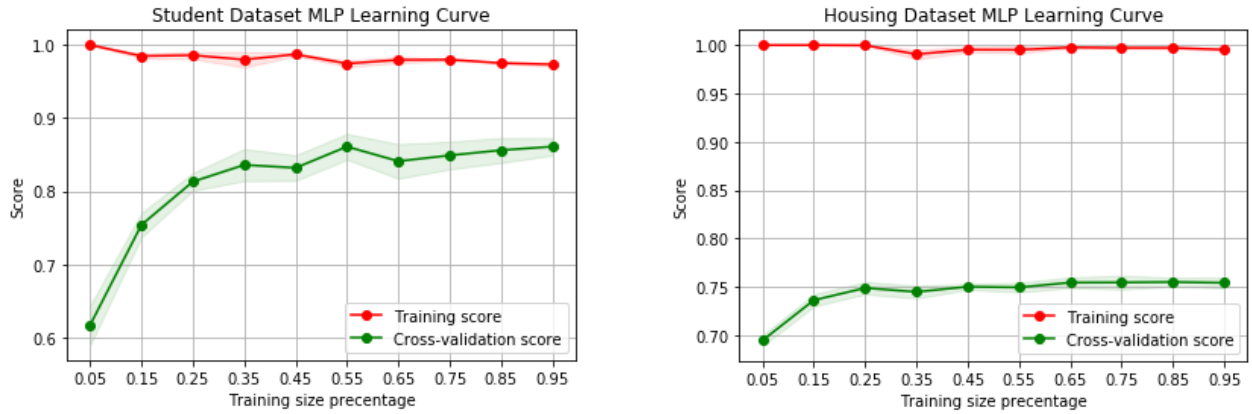

Figure 6: MLP Learning Curves

Analysis continues with a study of hidden layers, specifically the affect that the size of each hidden layer. To perform this task, a highly referenced equation is used to determine potential sizes provided by Oklahoma State University.[5]

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

Where:

$N_h = Number\ of\ hidden\ units.$
$N_s = Number\ of\ training\ samples.$
$\alpha = Hyper\ parameter\ scaling\ factor\ (typically\ 2 - 10).$
$N_i = Number\ of\ input\ neurons.$
$N_o = Number\ of\ output\ neurons.$

The results from varying α are displayed below in Figure 7. Hidden layer depth was additionally change from 1 to 2 to improve the network's ability to capture non-linearity. Alteration of the number of hidden units clearly had significant affects. Improvements while the values are low is not surprising, but the dramatic decrease in accuracy for both training and testing sets in the student data is unexpected. It's difficult to identify why this is the case, but it the increase in hidden units may be leading to non-convergence of the perceptron, or not reaching the global minimum in the error function.

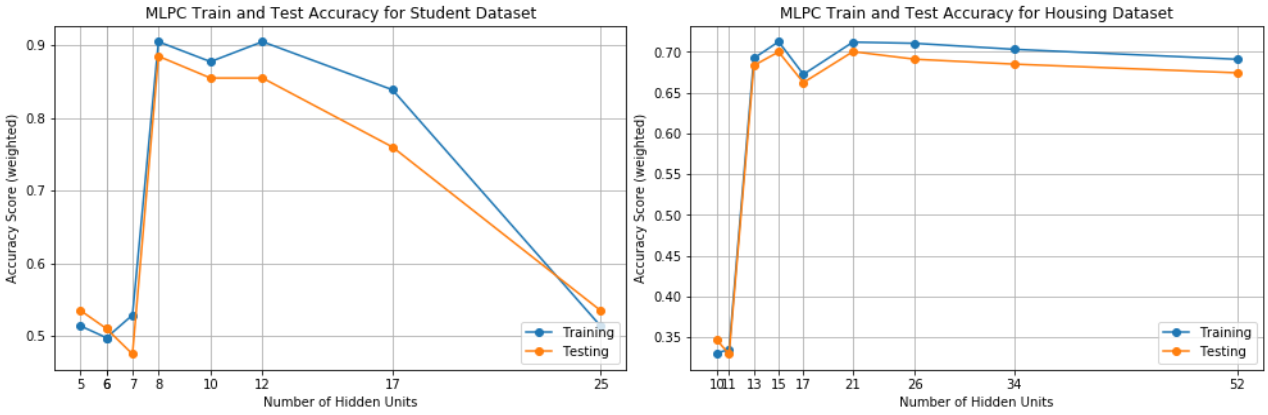

Figure 7: MLP Hidden Unit Results

With a better understanding of the affect that the hidden layer has on performance, we now shift our analysis onto the different activation functions. The sci-kit learn MLPC class contains four activation function options, *identity*, *logistic, tanh,* and *relu.* Using the optimal number of hidden units found in the previous analysis, the resulting testing accuracies are displayed below in Table 2.

Student	10-Fold Acc	Std
Identity:	0.8750	0.0685
Logistic:	0.5491	0.0818
Tanh:	0.8719	0.0596
Relu:	0.7573	0.2455

Housing	10-Fold Acc	Std
Identity:	0.7489	0.0091
Logistic:	0.7691	0.0151
Tanh:	0.7532	0.0141
Relu:	0.7653	0.0091

Table 2: MLP Activation Function Results

Interestingly, we find some pretty large disparities both within and between the datasets. For the student dataset, the logistic activation function suffers over a 30% accuracy decrease to the identity function. The housing data is a lot more consistent in terms of the performance from each activation function. A reason for this disparity may be the size of the datasets. The neural network for the housing dataset is allowed to train for longer with more variation in its data whereas the student data is limited in comparison.

In conclusion, fairly significant improvements to the MLP classifier on both datasets can be found with the simple parameter tuning performed above. It would be interesting to compare the performance of a perceptron network against a backpropagation flavor network.

Support Vector Machine

Before jumping into the SVM analysis, it's important to notice the extremely long train and test times shown in Table 1 for the housing dataset in particular, which is likely due to the large feature space expressed by this dataset, at 287 after one hot encoding. SVM learning curves are first constructed and displayed in Figure 8 below. Default parameters use a penalty value of 1, radial basis kernel function with a coefficient of 1 / *n_features*, a stopping tolerance of 0.001, and unlimited iterations. Input feature vectors are once again normalized with the scikit-learn *StandardScaler* class.

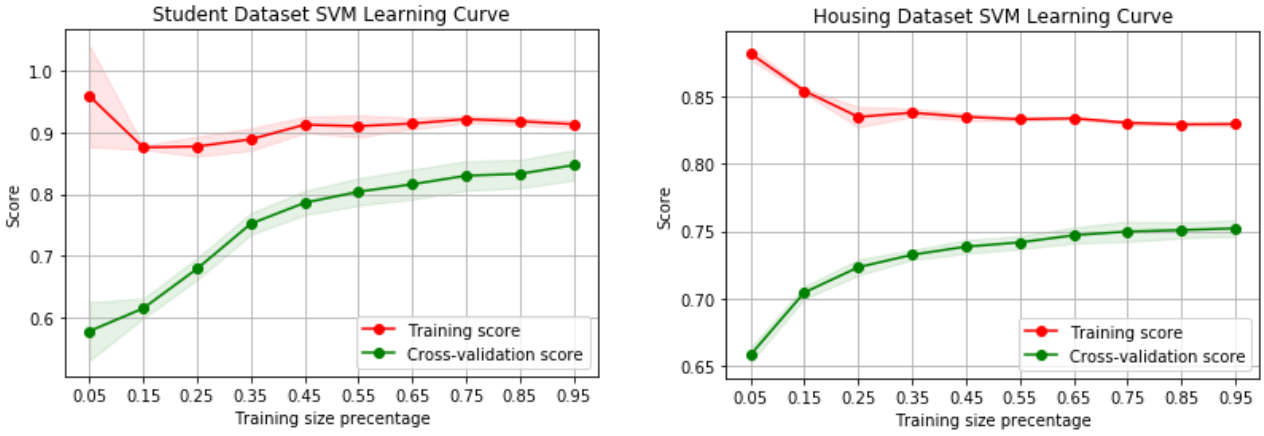

Figure 8: SVM Learning Curves

Follow the learning curves, the first analysis performed is the effect of different kernel functions, namely linear, poly, rbf, and sigmoid. Their performance accuracies are displayed in Table 3 below.

Student	Accuracy	std
Linear:	0.9010	0.0562
Poly:	0.8279	0.0815
RBF:	0.8669	0.0583
Sigmoid:	0.8461	0.0514

Housing	Accuracy	std
Linear:	0.7534	0.0116
Poly:	0.7014	0.0169
RBF:	0.7546	0.0089
Sigmoid:	0.7296	0.0118

Table 3: Kernel function analysis results

While it's difficult to understand why specific kernel functions perform better than others, it's interesting that the linear function was much more accurate than the others in the student dataset and nearly was the best in the housing. Theoretically, the radial basis function can be tuned to perform exactly the same as a linear kernel function, so it is theoretically the more accurate choice from this analysis, but the linear kernel will always compute faster.[6]

In addition to the kernel functions, the penalty value is also analyzed, and results are displayed in Table 4 below, which fits the SVM with the optimal kernel function determined from above. From the default value of 1, values are chosen for various values of 2^k, where k are arbitrarily selected values from [-7, -5, -3, -2, -1, 2, 3, 5, 7, 9]. Unfortunately, significant improvements are not found for either dataset. It could be by chance that the default value of 1 creates the optimal hyperplane margin.

Student	Accuracy	Std
0.0078125	0.8669	0.0551
0.03125	0.8970	0.0582
0.125	0.9050	0.0641
0.25	0.9020	0.0606
0.5	0.9020	0.0552
4	0.9020	0.0473
8	0.8990	0.0556
32	0.9000	0.0515
128	0.9000	0.0515
512	0.9000	0.0515

Housing	Accuracy	Std
0.0078125	0.6186	0.0122
0.03125	0.6803	0.0124
0.125	0.7267	0.0153
0.25	0.7399	0.0131
0.5	0.7477	0.0097
4	0.7551	0.0134
8	0.7515	0.0150
32	0.7382	0.0155
128	0.7213	0.0146
512	0.7117	0.0151

Table 4: Penalty value analysis.

From this analysis, we find that the type of kernel function has a significant impact on the accuracy of the classifier, while the penalty value does not. Unfortunately, it's difficult to assess why one kernel performs better than the other from a mathematical level, but it is interesting to see how a simple linear function nearly performed best on both sets.

K-Nearest Neighbors

Last but not least, the instance-based KNN approach is analyzed, which arguably performs the worse in terms of accuracy and exhibits a high testing time. As expected in the execution times shown in Table 1, the test time is significantly longer than the training time, since as an instance-based classifier, training time is simply loading in the dataset to memory. Finally, we conclude with our last learning curves shown below in Figure 9. Both uniform and distance weight functions are used and performed similarly in terms of CV accuracy, but only the uniform weight is shown below. By default, the number of neighbors is set to 5.

Immediately, for the student dataset, the learning curves may be exposing overfitting. With additional training data, the CV error appears to be decreasing, whereas with the housing data, it steadily increases with more training. Likely, this is due to the contrast in feature space sizes.

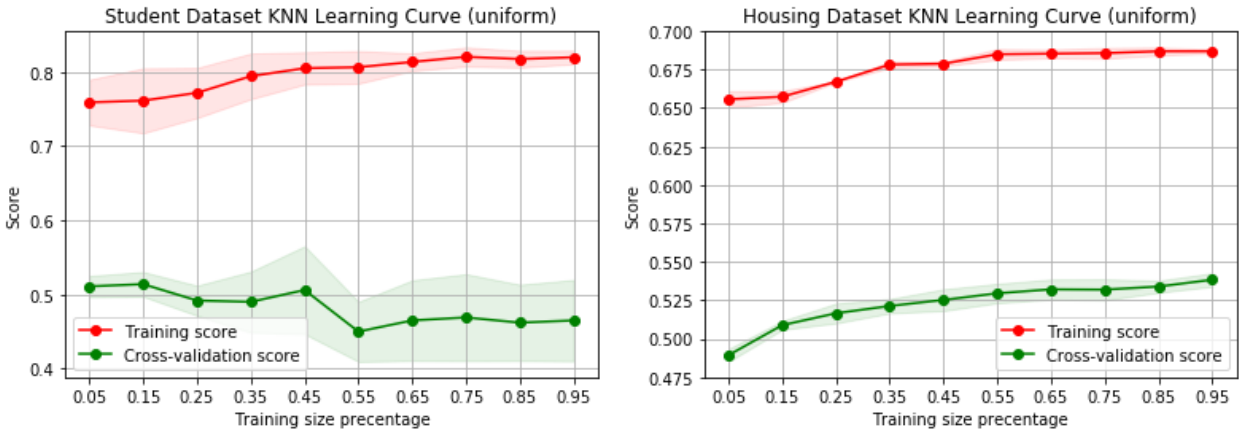

Figure 9: KNN Learning Curves

Our analysis now investigates the effect of *N* on performance. Shifting from the default 5, the student dataset sees a slight improvement in test accuracy as the number of neighbors increases, but the improvement is less than 1%; a similar but smoother trend is found with the housing dataset.

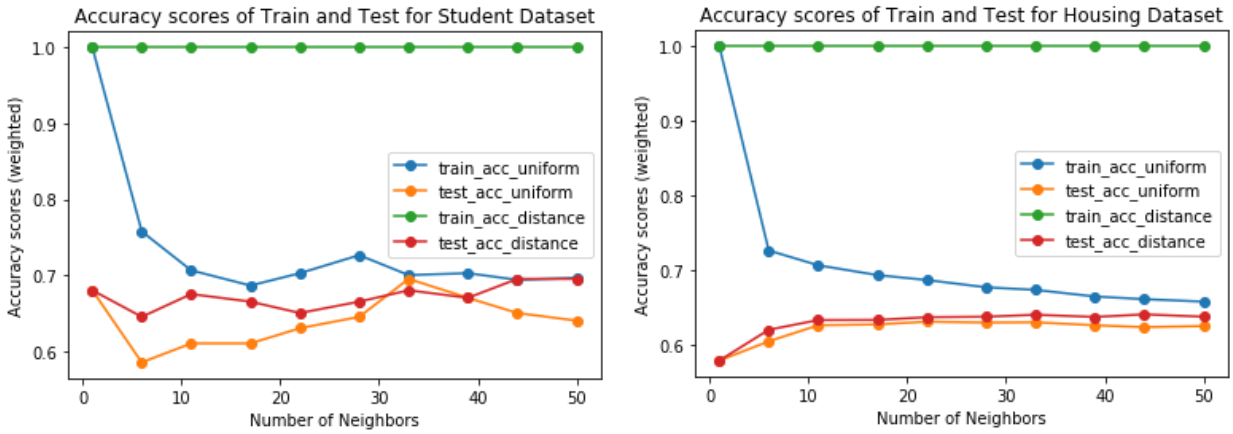

Figure 10: KNN N-Neighbor Results

Analysis in both the weight function as well as the number of neighbors does not appear to improve the performance of the KNN by a noticeable amount. Additionally, the testing time for the housing dataset shows one of the larger flaws of an instance-based learner, where compute time does not scale well with data size.

Summary

With our exploration and analysis into parameter tuning of our various classifiers, a comparison between how they have improved is displayed in Table 5 below using the best tuned model for 10-fold cross validation accuracy.

Student	10-Fold Cross Acc		
	Before	After	Increase
Decision Tree:	0.8199	0.8300	1.23%
XGBoost:	0.8759	0.8809	0.57%
MLP:	0.8770	0.8859	1.01%
SVM:	0.8669	0.8830	3.01%
KNN:	0.6560	0.7050	7.47%

Housing	10-Fold Cross Acc		
	Before	After	Increase
Decision Tree:	0.7388	0.7618	3.11%
XGBoost:	0.7912	0.8150	3.01%
MLP:	0.7591	0.7690	1.30%
SVM:	0.7546	0.7546	0.00%
KNN:	0.5935	0.6327	6.60%

Table 5: Parameter tuning improvement results

We find that after parameter tuning, nearly every classifier for both datasets improved in performance! With this increase however, we find that the highest 10-fold cross validation accuracy classifier remains the same for the respective datasets, but by much slimmer margins. These performance comparisons show the importance of parameter adjustment, as shown best by the KNN improvements; accomplished through simply adjusting the number of neighbors.

Overall, this analysis has shown that classification of gender based on seemingly unrelated information, and the price group of a house based on a large number of features can be accomplished relatively successfully through various machine learning techniques. Through a simple default implementation for each classifier, we've found fairly good results, which can often be improved through basic hyperparameter tuning. Lastly, I've personally learned more about the residential environment of my city, which will hopefully provide insights for a future purchase.

References

1) SPScientist. "Students Performance in Exams." RSNA Pneumonia Detection Challenge | Kaggle, 9 Nov. 201
2) "Address Residential Units." http://opendata.dc.gov/datasets/address-residential-units.
3) Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011
4) XGBoost: A Scalable Tree Boosting System, Chen *et al.*, KDD 16, pp. 785-794, 2016
5) Hagan, M., Demuth, H., Beale, M. and De Jesús, O. (2016). *Neural network design*. 2nd ed. Stillwater, Oklahoma: Oklahoma State University.
6) Keerthi, S. and Lin, C. (2003). Asymptotic Behaviors of Support Vector Machines with Gaussian Kernel. *Neural Computation*, 15(7), pp.1667-1689.