Supervised Learning Models for Classification

CS 7641: Machine Learning Assignment I

Jay Kint, GTid: 903357015
2/14/2021

Introduction

The purpose of this assignment is to explore supervised machine learning algorithms within different contexts to understand how they behave under a variety of circumstances.

I have chosen two datasets, one for regression and another for classification. (Professor Isbell's response to a student on Piazza regarding regression vs. classification: "You can do regression if you want to. I appreciate the claim that's it's all classification under finite data."[i])

The first dataset is a common one I've seen used on the Internet; the *Finding Donors for Charity* data set from the UCI Machine Learning Repository[ii]. The task is to predict people with more than $50k annual income (ostensibly who would be more willing to donate to charity). There are 13 features, 8 categorical and 5 continuous, over 45k entries. This dataset is not very balanced. *This data set is simply called the "census" dataset from here on.*

The second dataset is a list of cars, their features, and their prices as scraped from Kaggle via Edmunds and Twitter[iii]. The task is to predict the price of a car given its features. There are 9 categorical, and 7 numeric, which we will explore in more detail below, with almost 12000 samples. This dataset is fairly well balanced, except for a couple of columns which are eventually eliminated. *This dataset is simply called the "cars" dataset from here on.*

These are interesting data sets due to their applicability as scaled down problems found in the real world. Price prediction is always en vogue, and the used car market is enormous. Predicting incomes is perhaps more problematic in today's world, as it could be used for various purposes; the purpose of donating to charity is noble and knowing how income predictions are performed is useful for combating nefarious activities (such as finding people to defraud).

Truth be told, there are so many good data sets, it was difficult to narrow it down to these two. And even these two types of datasets, prices and income classifications, there are several good ones available that are far larger and could have yielded better information. In the end, these two were chosen for their size and popularity. Otherwise, I would have likely chosen similar data sets but larger and more feature rich[1].

Python 3.9 was used to write the software. The algorithms and data transformations used were from the Scikit-Learn library, version 0.24.1. Pandas 1.2.2 was used for data loading and manipulation. Other libraries and their versions are listed in the requirements.txt referenced in the README accompanying this report (and useable by pip to install the same libraries).

[1] I would really liked to have tested the hypothesis of "The Unreasonable Effectiveness of Data" (as cited in Geron, Aurelien. "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow", 2nd ed. 2019. p. 24.) in this paper with a larger dataset but running times and storage/compute costs were prohibitive.

Exploratory Analysis and Preparation

These are the transformations done on the cars dataset:

- normalize the MSRP label distribution via log transformation,
- "leave one out" encoding for the categorical parameters except Market Segment,
- break out the Market Segment into one hot encoded features,
- standard scaling all other the features to a mean of 0 and std of 1,
- determined the most correlated (and inversely correlated) features in the dataset,
- PCA to determine how many features we could eliminate without reducing the dataset's variance too much[iv] (variance loss curve shown in code referenced in accompanying README),
- Eliminate 8 of the features that were either highly correlated or were highly imbalanced to decrease the number of features from 25 to 17 without reducing the variance too much (from the previous PCA run): exotic, city_mpg, highway_mpg (these two were collapsed into mpg), high_performance, luxury, factory_tuner, flex_fuel, diesel (last two covered by the engine_fuel_type),
- dividing the data into training and test sets of 80/20,
- run 5-fold cross validation of each algorithm for the training data and test data to compute baseline Root Mean Squared Error (RMSE).

These are the transformations done on the census dataset:

- normalize the capital-gain and capital-loss features via log transformation,
- scaled numeric features to 0 and 1,
- dropped native-country column due to imbalance (91% from a single country, the US),
- reified income to binary encoding (0 and 1),
- one hot encoded race and working class columns,
- examined correlations but decided to keep all the data.
- run default classifiers and compute the Area Under Curve (AUC) from a Receiver Operating Characteristic (ROC) curve.

Metrics Chosen

For the cars dataset, a regression problem, I use the Root Mean Squared Error (RMSE). For the census dataset, I used the Receiver Operating Curve (ROC) (the False Positive Rate (FPR) over the True Positive Rate (TPR)) with the Area Under Curve (AUC) metric since it is a classification problem.

Baseline and Learning Curves

The next item was to establish a baseline for how well the algorithms performed on the datasets.

The baseline consists of running the data through each algorithm with default parameters[2] and plotting the appropriate metric. This gives us something to improve on.

The ROC curve below shows the AUC metrics for each algorithm on the census dataset, with the following predictive results:

[2] For the kNN algorithm, the default I chose was k=5, and I used the non-default "ball-tree" algorithm to speed up testing over the default "brute force" method.

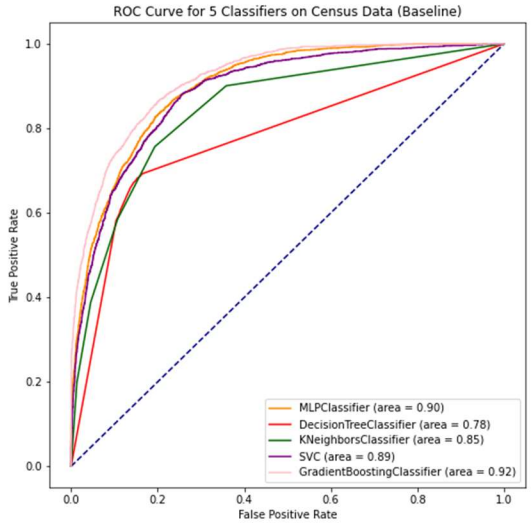
1) Gradient Boosted Trees
2) Neural Network (MLPClassifier)
3) SVM (SVC)
4) K Nearest Neighbor
5) Decision Tree

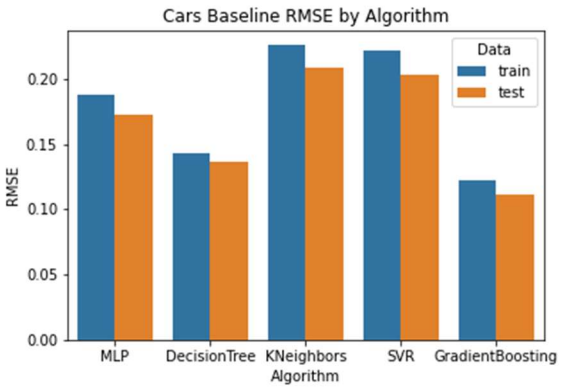
The relatively high results show that the problem isn't too hard, but the poor results of the Decision Tree indicate there might be overfitting.

The bar chart shows the Root Mean Squared Error for each algorithm on the cars dataset, with the following predictive results:

1) Gradient Boosted Trees
2) Decision Tree
3) Neural Network (MLP)
4) SVM (SVR)
5) K Nearest Neighbor

The RMSE numbers were all in a fairly tight range though, only ranging from 0.11 to 0.22, also showing that the problem isn't too complex. Converse to the census dataset, the Decision Tree was the second best. I would have thought that decision tree would be better at classification than regression, and





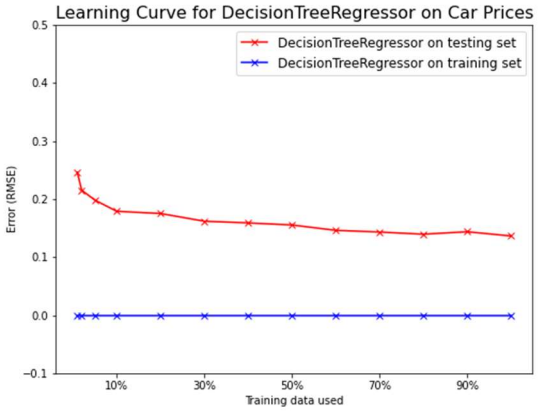
perhaps we shall see that later in our analysis.

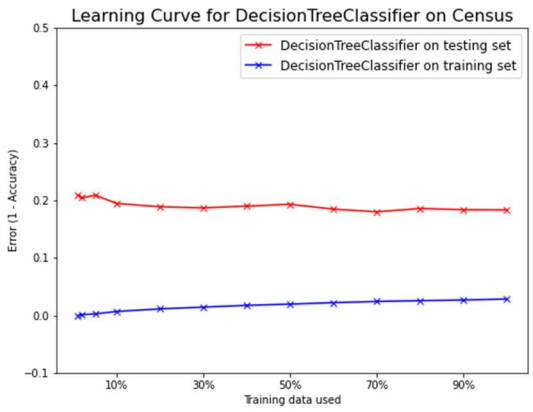
Each dataset has a different set of rankings for the classifiers. Gradient Boosting and Neural Networks are both in the top 3 of each dataset, and neither kNN nor SVM are in the top 3 of each dataset. Perhaps hyperparameter tuning can rearrange those rankings.

Following are the learning curves for each algorithm and data set. The primary goal of these learning curves is to examine how each algorithm behaves with more training data and see how that affects the error rates and bias/variance trade-off[v]. The curves are at the same scale for each dataset, to allow for easier comparison.

Learning Curves for Decision Tree

Sci-kit decision trees use the Gini metric instead of the information gain detailed in the class[vi]. No pruning is done either by default.





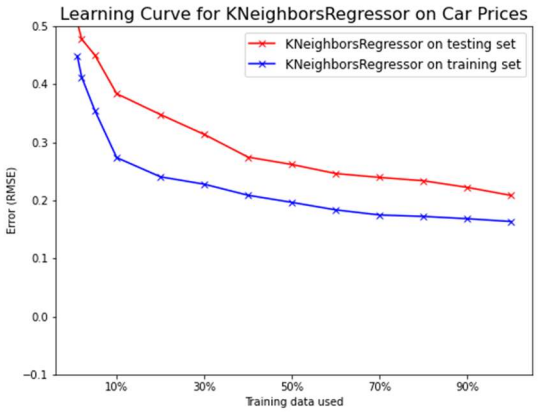
The graphs show that there is a pretty wide performance gap between the test and training sets. This indicates a high variance. The cars dataset looks like it might converge, but in the wrong direction. (Not really, since we've used 90% of the data.) Likewise, the census data doesn't converge and after 90%, it remains far apart despite a slight improvement. The slight improvement is likely due to 4 times the amount of data in the census dataset.

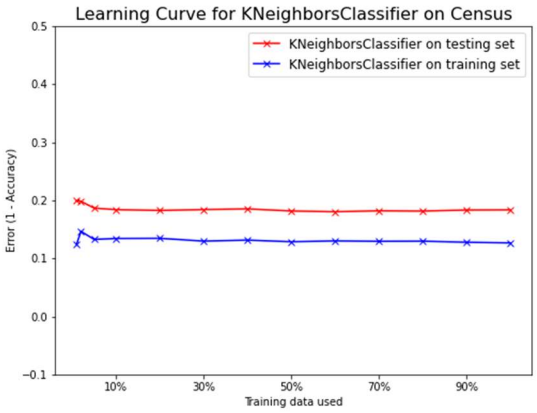
The low error metrics for the training set indicates a low bias and overfitting. The tree depth and number of leaves are very high for the amount of data processed (depth of 13 and 58 leaves at 2% data to depth of 51 and 6000+ leaves at 90% for the census dataset; depth of 15 and 86 leaves at 2% data to depth of 39 and 8000+ leaves at 90% for the cars dataset). (Diagrams for depth and leaf count omitted for space.)

Pruning should decrease the model complexity, increase the bias and lower the variance, and hopefully improve the response for both datasets.

Learning Curves for k-Nearest-Neighbors

k-Nearest Neighbors (kNN) is a straightforward algorithm for both regression and classification[vii]. The main hyperparameter is k, the number of neighbors to query. A small k (e.g., k=1), the model is susceptible to outliers and overfitting. For the census learning curves, I used the non-default spatial





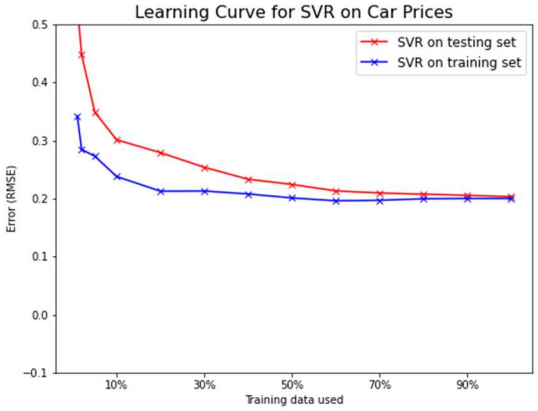
algorithm 'ball_tree', which uses points/radius combinations to create subsections of points to rapidly determine nearby points to compare. This improved the running times to acceptable amounts. The cars dataset is small enough (12k point) that brute force was sufficient.

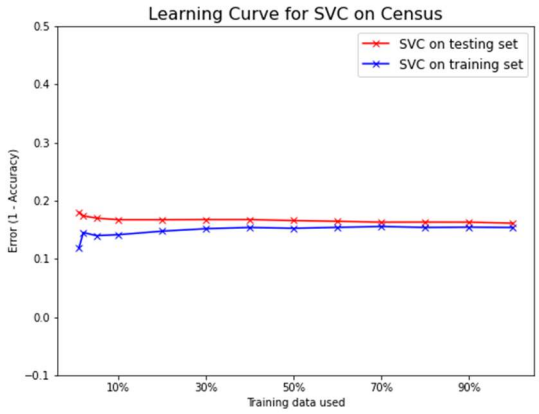
In our learning curves here, I used k=5, which is fairly small. The test error rates, and hence bias, aren't much higher than other algorithms, but they fail to converge on either dataset, and the variance between the training and test set is only worse for Decision Trees. This continuous gap implies a relatively high variance and even some overfitting, otherwise they should converge with more data.

The lack of convergence means adding instances won't likely help. Adding features to increase complexity would help, but that can be expensive. Regularization might help, though that could increase the variance, which is already higher than other algorithms except decision trees. Increasing the k should improve, but we'll see when the model complexity curves are constructed.

Learning Curves for Support Vector Machine (SVM)

The SVM (SVR for regression, SVC for classification), are right in the middle of the pack for performance on the learning curves. They converge well with more data, though the cars dataset has a rough start. The census dataset pretty much remains constant regardless of the amount of data.

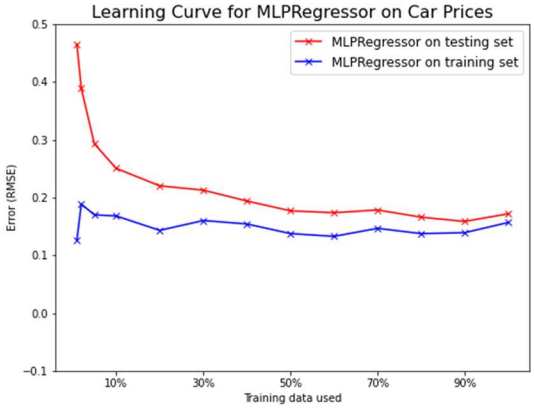


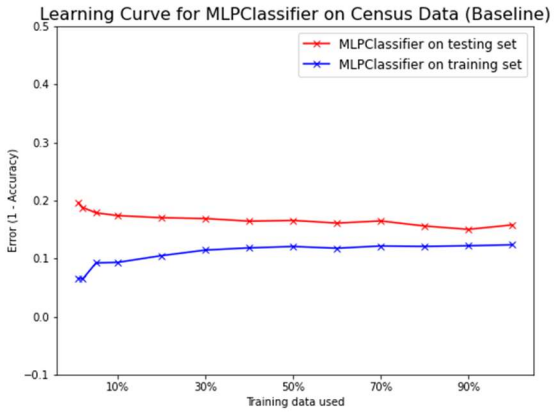


This flat learning curve and higher training error indicates high bias since more data does not decrease (or increase) the error rate. The thin gap means very low variance. Given the high bias and low variance, increasing the model complexity by tuning the hyperparameters is the way to change the error rates. The primary Scikit-learn parameters are C (which controls the margin sizes) and the kernels (equations that make the data separable when not naturally so). The default kernel is 'rbf', a non-linear kernel (projecting to a higher dimension space to make the data separable), and we used a default C of 1[viii].

Learning Curves for Neural Network

Sci-kit Learn calls neural networks "Multi-Layer Perceptrons"[ix]. The default is a single layer of 100 units.



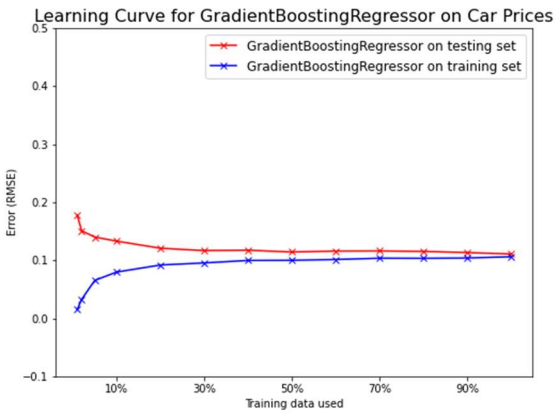


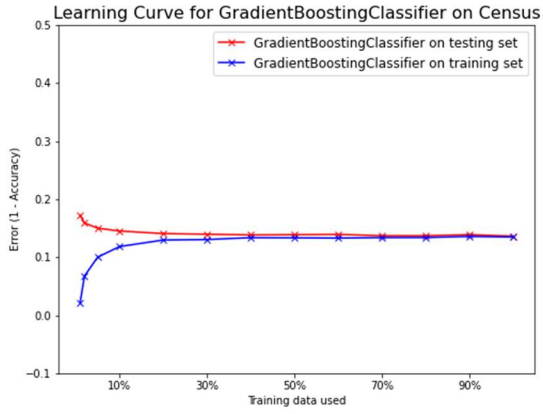
We see from the curves that the MLP has a couple of nice characteristics. The error rates are low for both algorithms, especially for the census dataset, and they converge with more data, especially the cars dataset. The error difference isn't too great, but it doesn't improve drastically either with more data, so there is likely a high bias/low variance contrast that I can attempt to correct with model complexity via the hyper parameters.

There are several parameters to tweak for neural networks, including hidden layer count, neuron count, and regularization alpha.

Learning Curve for Gradient Boosting Decision Trees

Gradient Boosted Decision Trees are unlike the other algorithms we've studied so far. GBDT's are amalgamations of decision trees rather than just a single model or "data structure". Several representations are kept and each one is consulted to get a response, and the responses are aggregated via one of several methods: mean for regressions usually, and mode for classification usually. This aggregation of the "ensemble" of representations generally allows GBDTs to avoid overfitting.





The error rates are the lowest of all the algorithms, and they converge rather quickly, which seems good. Their variance and bias are the lowest of all the algorithms (excepting Decision Trees overfitted

training). These learning curves are almost ideal[v]. I'm not sure there is much to be done here, though pruning might help a bit.

Model Complexity Curves

A model complexity curve charts the changes to errors over changes to parameters. In this section I change only a single parameter per curve to keep the analysis simplified, with the exception of SVMs, for which I provide two comparisons, one for kernels and another for the C parameter.

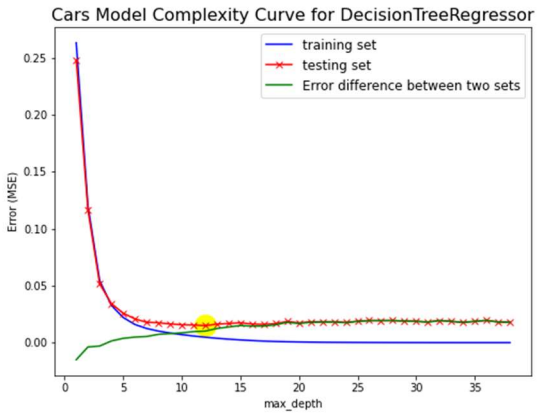
It's probably obvious, but I'll state it anyways. Given the 2-dimensional nature of these plots, a single parameter was the option to change and monitor the result.

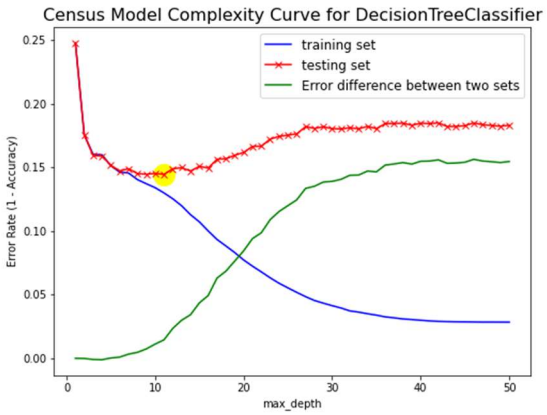
I did not keep the graphs at the same scale as with learning curves.

An interesting note about cross validation: I did not cross validate the training samples in this section. Well, I did, but the result was extremely puzzling. The testing sets consistently scored better than the training set in all the algorithms. This is highly dubious. Given the online explanations for such phenomenon were often about the distributions not being random enough[x] [xi], I decided to not use cross-validation and used a straight split of 80/20 from random samples. It bears further study, but is beyond the scope of this report given the time constraints.

Decision Trees

The parameter modified is the max_depth of the tree, ranging from 1 to the maximum depth of the unpruned depth. The other parameters to potentially modify included max_leaf_nodes, min_samples_split, min_samples_leaf, and min_impurity_decrease. max_depth was chosen for its obvious pruning association, but a combination found via grid search would probably be best. (The decision trees are extremely fast, so a wide grid search would be incredibly practical).



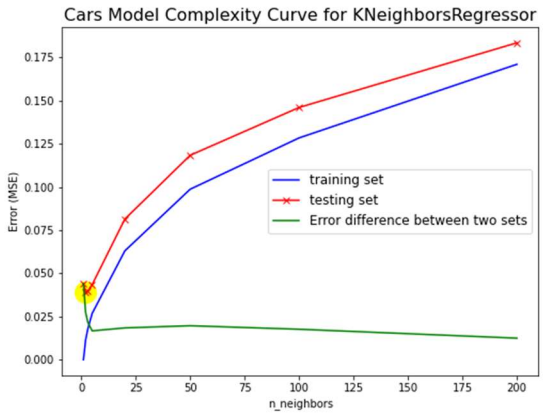


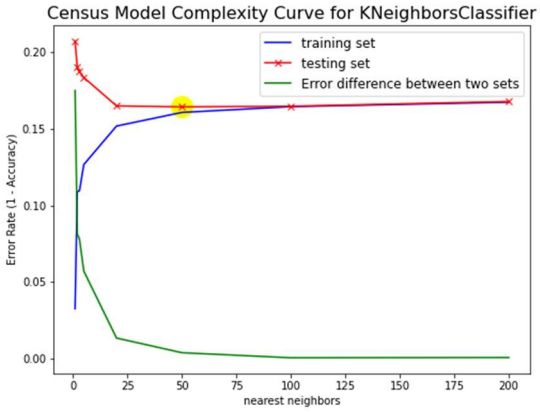
The cars dataset did what I figured both would do. It converged to an ideal depth around 12 and the test error stayed about the same. The error difference remained about the same after 12.

The census dataset performed somewhat similarly, but it diverged after its ideal depth around 12 and became a little worse, though it only deteriorated from 0.15 to ~0.19.

Model Complexity for k-Nearest Neighbors

The number of neighbors to factor (k in the name), n_neighbors, was the parameter modified for the model complexity for k-Nearest Neighbors. The expectation is that the greater the k, the more robust the estimates, up until perhaps some point of stabilization, after which increasing k does nothing to improve the error.





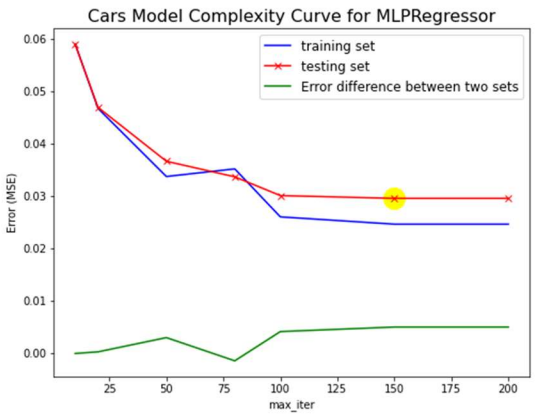
The baselines and learning curves used a value of k = 5. Varying k for the model complexity curves show each different results for each dataset.

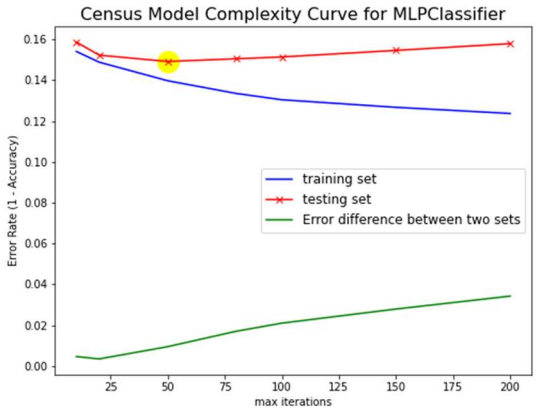
For the census, it aligns more with expectations, that more neighbors would provide a better estimate, in this case with k = ~55.

The cars dataset bucks that expectation, with k=2 being the best results for the test set. However, looking further at the census graph, it seems that the difference in error between the training and test sets is best at around 5-10, so I might recommend that value instead. After that, the error difference stays pretty much the same.

Model Complexity Curve for Neural Network

The parameter modified for the MLP (Multilayer Perceptron) neural networks was the number of iterations (max_iter). As with the learning curves, we used the adam solver, but didn't vary that during these experiments. Using other parameters such as hidden_layer_sizes and alpha didn't vary the models much, so max_iter was the parameter I settled on for these curves. Increasing hidden layers only increased the variance, and changing the alpha had minimal impact on the error rate/MSE.





The census graph shows that the census dataset overfits, since the training error decreases while the test error increases after max_iter=50 or so. Max_iter = 50 seems to be a good marker, and perhaps even 25 < max iterations < 50 is good as the error pretty low (high on the graph, but the values are low and the graph isn't scaled).

The car graph gives us some good data points that might be useful at various iterations. The error is lowest at max_iter > 100, but the variance increases a bit, but is pretty stable, so I would recommend a max iterations of 100.

Model Complexity for SVM

Based on the assignment criteria, "I'd like to see at least two" kernels is required. The margin parameter C is also interesting, so we evaluate it also.

Kernels

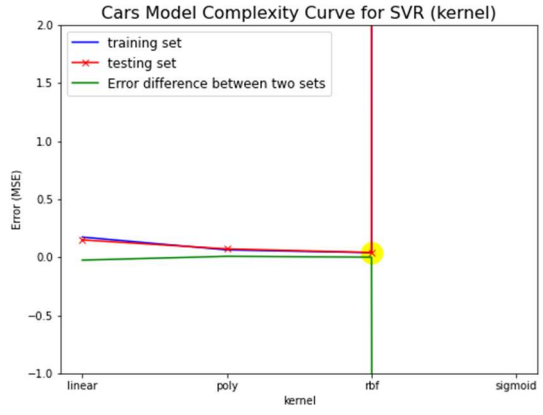
Scikit Learn has 4 kernels available: 'linear', 'poly', 'rbf', and 'sigmoid'. (There is also a precomputed available, but that was skipped.)

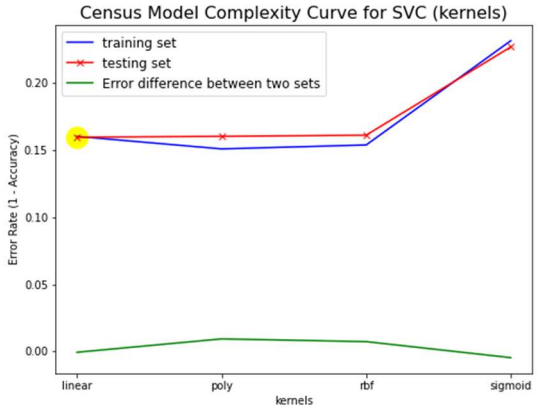
The linear, poly, and rbf kernels all gave very similar results on both datasets, to the point where they are almost a wash. The only real lesson is to avoid the sigmoid kernel. Even more puzzling is that the sigmoid kernel did better on the testing data than on the training data on both data sets, though again barely.

The cars graph deserves a special explanation. The error rate for the sigmoid kernel was in the hundreds, and like the census graph, the test set did better than the training set. To better show the differences between the 3 acceptable kernels, the range was constrained to -1 to 2.

If it weren't for the assignment asking to see kernels, I likely would have skipped this section as kernels don't seem to make an appreciable difference.

The basic conclusion is that the SVM kernels result in high bias models, similarly as with the learning curve. Except sigmoid. It's so whacked, I'd recommend sigmoid talk to Sigmund Freud. 😊

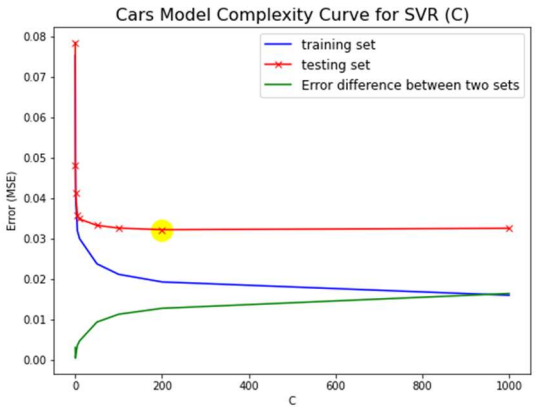


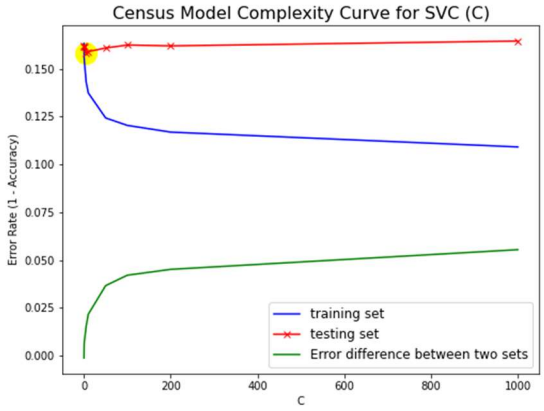


C

Given that the results of the kernel evaluation were not that interesting (aside from sigmoid being so devolved), the margin parameter C was also evaluated.

The learning curve default for C was 1, so we evaluate values both less than and greater than that to see how it affects the error rates, using the default 'rbf' kernel.

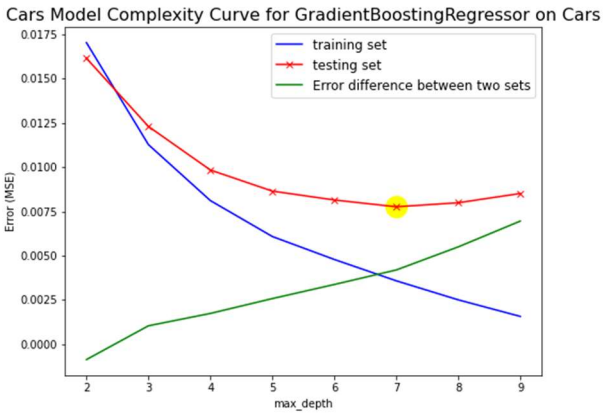


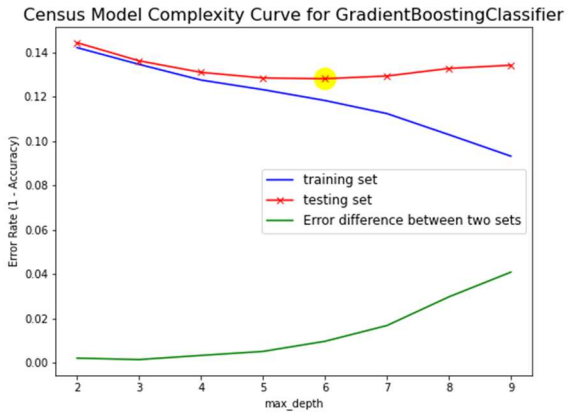


The first thing to notice is that both have similar general shapes. Controlling C gives a high variance, and that variance (and the error rate) doesn't substantially change as the C increases. Neither one would ever converge since the error rate keeps rising. C greater than single digits or low double digits seems to only make things worse. 5 < C < 10 seems perfectly reasonable, though there is a lot of leeway in the assignment for the cars dataset, which could probably go as high as 50.

Model Complexity Curve for Gradient Boosted Decision Trees

The learning curves showed that the GBDT's provided the best results. Will that continue if we change the hyper parameters? Given we only have 2 dimensions to plot, we will only alter one, and given the similarity to Decision Trees, the parameter will be max_depth.



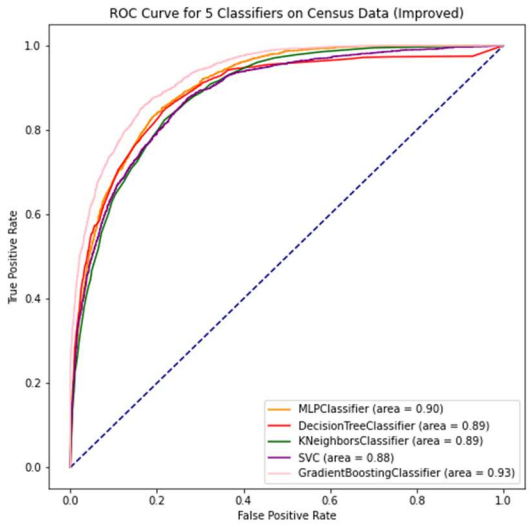


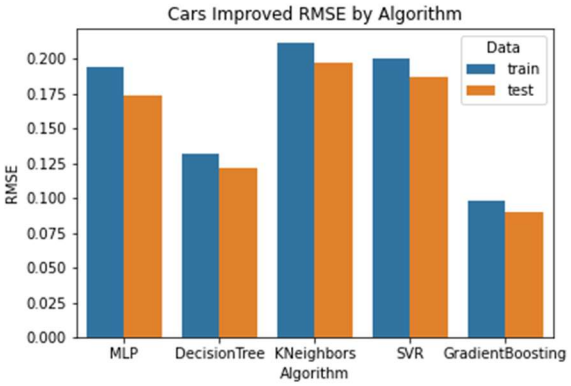
With Decision Trees we had the range of up to 50+ depth, but for the GBDT, the metric is much more aggressive, only ranging to 10. Both the cars and census data sets perform similarly, if not on the same scale.

The variance increases as the depth increases, but the error keeps reducing until about 6 or 7 depth for both training and test sets. After 6 for census data and 7 for cars data the test error increases. 6 seems to be the reasonable depth to assume for the retest.

Summary

Armed with our new parameters, how much better did we make our models?





Census Dataset Algorithm	**AUC Change**
Decision Tree	+.11
K-Nearest Neighbor	+.04
Neural Network (MLP)	None
SVM (SVC)	-.01
Gradient Boosted Trees	+.01

Cars Dataset Algorithm	**RMSE Change**
Decision Tree	+.02
K-Nearest Neighbor	+.02
Neural Network	None (+.002)
SVM (SVR)	+.02
Gradient Boosted Trees	+.02

We see improvement across almost the entire set. However, the increase is generally very small. The worst performing algorithm on the census, decision trees, received the best improvement ratio, 14% improvement. The GBDT remain the best algorithm for these datasets by quite a bit, with the RMSE being less than half of the worst performing (kNN) on the cars dataset, and the highest AUC on the census dataset. The only regression was for the SVM on the census dataset, which only degraded a very small amount (-.01).

Our final rankings for each dataset were:

Cars

1) Gradient Boosted Decision Trees
2) Decision Tree
3) Neural Net (MLPRegressor)
4) SVM (SVR)
5) K-Nearest Neighbors

Census

1) Gradient Boosted Decision Tree
2) Neural Net (MLPClassifier)
3) Tie: Decision Tree and k-Nearest Neighbors
4) SVM (SVC)

The only change was that the Decision Tree and SVM swapped places on the census dataset. Otherwise, the rankings remained the same. This would indicate that Scikit-Learn comes with reasonable defaults for their classifiers and regressors.

The final point I observed is that the error rates were fairly uniform across each dataset regardless of algorithm, once sufficient data had been trained. I can't help but wonder if this is an observation of the phenomenon described in "The Unreasonable Effectiveness of Data"[xii].

Things I Wish I Had Time Left To Do

While analyzing the learning curves, model complexity was often mentioned as a way to improve the error rates. For this report I opted to use the hyperparameters to see what could be done, rather than augmenting the data. As can be seen, there was only slight improvement.

These small improvements could be built on by expanding on the hyperparameters used to tune and using something similar to Grid Search with Cross-Validation[xiii] to try several at once.

I also could have increased the complexity by adding features, generally through the adding polynomial features[xiv].

---

[i] Isbell, Charles. FAQ: What are interesting datasets?. https://piazza.com/class/kjwtraqiv511du?cid=51_f6. Retrieved February 21, 2021.

[ii] Kohavi, Ronny, and Becker, Barry. UCI Machine Learning Repository: Census Income Data Set. https://archive.ics.uci.edu/ml/datasets/Census+Income. Retrieved January 30, 2021.

[iii] CooperUnion. Car Features and MSRP. https://www.kaggle.com/CooperUnion/cardataset. Retrieved on February 1, 2021.

[iv] Tasos. "Curse of Dimensionality: How many dimensions is too many dimensions?" https://datascience.stackexchange.com/a/57712. Retrieved February 14, 2021.

[v] OIteanu, Alex. "Tutorial: Learning Curves for Machine Learning in Python". https://www.dataquest.io/blog/learning-curves-machine-learning/. Retrieved February 14th, 2021.

[vi] Scikit-Learn. sklearn.tree.DecisionTreeClassifier. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html?highlight=decisiontreeclassifier#sklearn.tree.DecisionTreeClassifier. Retrieved February 21, 2021.

[vii] Scikit-Learn. Nearest Neighbors. https://scikit-learn.org/stable/modules/neighbors.html. Retrieved on February 10, 2021.

[viii] Scikit-Learn. sklearn.svm.SVR. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html. Retrieved on February 8, 2021.

[ix] Scikit-Learn. Neural Network Models. https://scikit-learn.org/stable/modules/neural_networks_supervised.html. Retrieved on February 14, 2021.

[x] Match Maker EE. Test accuracy much higher than training accuracy. https://stats.stackexchange.com/questions/345656/test-accuracy-much-higher-than-training-accuracy. Retrieved on Feb 20, 2021.

[xi] Boystsov, Leonid. Is it possible to have a higher train error than a test error in machine learning?. https://qr.ae/pNEzX7. Retrieved February 20, 2021.

[xii] Geron, Aurelien. "Hands-On Machine Learning with Scikit-Learn, Keras & Tensorflow", 2nd ed. 2019. p. 24.

[xiii] Scikit-Learn. 3.2 Tuning the hyper-parameters of an estimator. https://scikit-learn.org/stable/modules/grid_search.html#grid-search. Retrieved February 21, 2021.

[xiv] Scikit-Learn. 6.3 Preprocessing Data. https://scikit-learn.org/stable/modules/preprocessing.html#generating-polynomial-features. Retrieved on February 21, 2021