An analysis of Supervised Learning Classification Models with imbalanced datasets

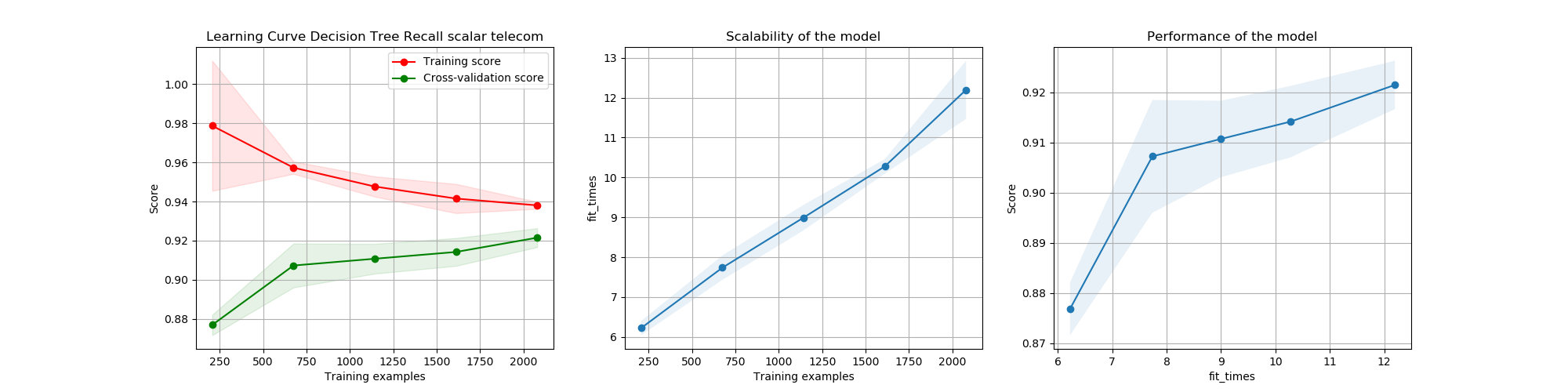
In the real world, customer retention is a valued and important aspect of any business. At the same hand, for some businesses, selecting to right customers is just as important given inherit risks to the services that they provide. The two classification problems I have selected each highlight these real world problems and attempt to predict when they occur. The first classification problem I have defined is utilizing telecom data for customers to determine if the customer will change providers or churn. The second classification problem that I have chosen is utilizing information about a banking client that has a loan to determine if the client will default on the loan. Both problems in the real world are the minority of cases for businesses, and as such, both problems have imbalanced data where only around 14.5% of the samples represent the class of interest. The rest represent the average customer who keeps their plan or pays their loan. A key difference in the classification problems are the number of features. The telecom dataset has 11 features and only 3,333 samples. On the other hand, the credit default problem contains just 3 features and 1,997 samples. The first problem suffers from the dreaded curse of dimensionality, and it will highlight how that impacts the learners. The second problem gives a good example of how having a good sample size in relation to the features can help a learner overcome imbalances in data. It should be noted that just guessing a class of 0 or the majority class value in each problem would result in 85.5% accuracy. This is why the graphs will also contain the recall score of the underrepresented class for the training and test data.

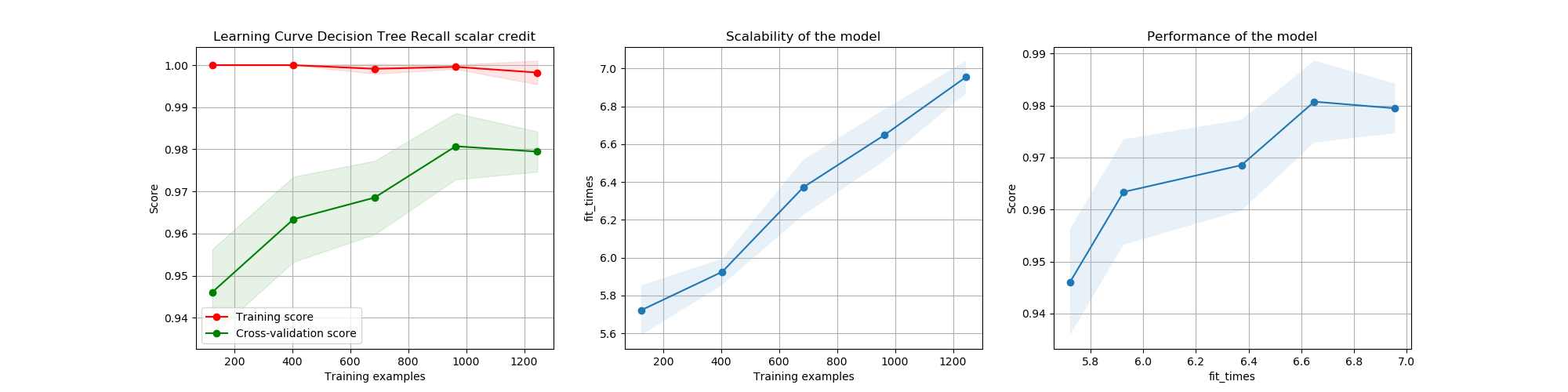
The Models

Decision Tree

For decision trees, I utilized the ‘best’ split with gini for determining what feature to split on. This method is the method by which a feature is selected based on how well it splits up the remaining data into the classification types. Randomization is used to pick between any given number of features that equally split the data. For tree pruning, I utilized two different kinds of pruning. pre-pruning and post pruning to assess the impact on the decision tree. I utilized both at the same time so that the tree would try not to overfit in the first place, but in cases where it was unable to be avoided, it would be able to prune the branches that were overfit. For pre-pruning, I utilized max depth, and for post pruning, I utilized cost-complexity pruning. I picked max depth because it is an efficient way to prevent trees from overfitting by limiting the actual shape of the tree itself, and the cost to vary it in hyperparameter tuning is minimal. I utilized cost-complexity pruning, because it is a calculation or how different the feature used to split the tree was and could clean up any overfitting leftover after max depth was considered.

The performance of the decision tree improved with more samples at first and then levels off. With only 200 samples, it has poor performance averaging around 87% on the cross validated set. As training samples are increased, an increase in performance can be seen up to around 92% The credit data increases to 98% accuracy and both level out. Overfitting is avoided in both cases which is the goal of the pruning. The training time of the decision tree is a linear relationship to the number of samples it is training on since the time for calculating the split is a direct result of how many samples are being used in the calculation. The max depth also impacts training time since more overall calculations need to be done for a greater depth of the tree. The scalability graphs below highlight both these relationships. The first graph is with a model with a max depth of 8 and the second for a model with a max depth of 6.



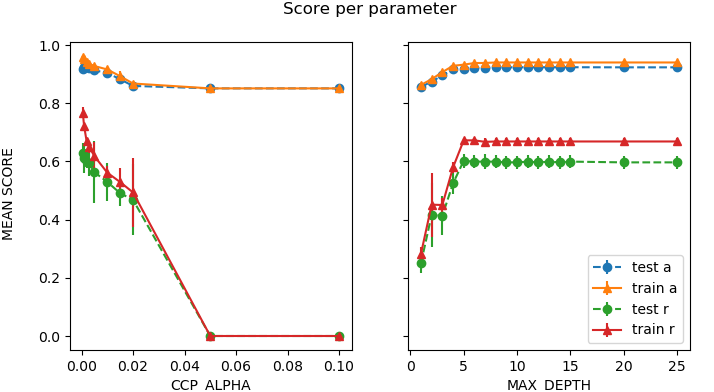


Decision tree results table

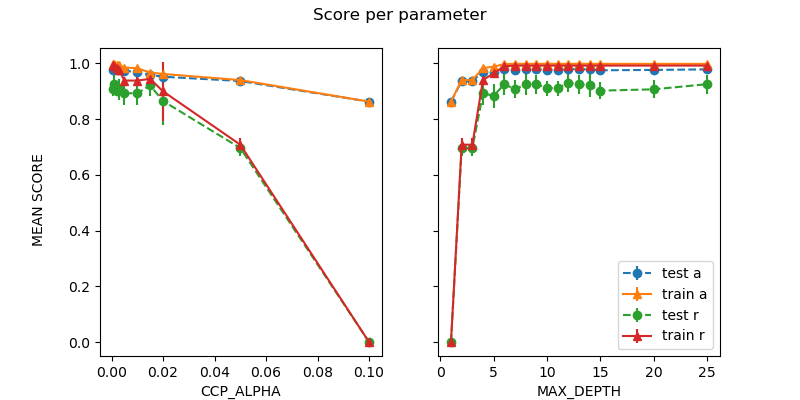
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy Train | Accuracy Test | Alpha for CCP | Max Depth |
| Telecom | 93.5% | 92.37% | .002 | 8 |
| Credit | 99.4% | 98.4% | 0.001 | 6 |

The table above highlights the accuracy and hyperparameter choices for the best model after cross validation was utilized on training data to select the hyperparameters. Note that for the credit data the decision tree can get a much higher overall accuracy than for the telecom data. This is likely since the credit data has only 3 features to choose from and when deciding how to split the data it would become easier to see what feature will split off the two classes. In comparison, when there were 11 features to split on, there were likely multiple features that split the data in different ways that had similar levels of the different classes in each group making it harder to select the feature that actually represents the relationship between the features and the final classification. Further analysis of the graphs below will give insights into the hyperparameter selection process for each dataset.

Decision Tree Hyperparameters for Telecom data

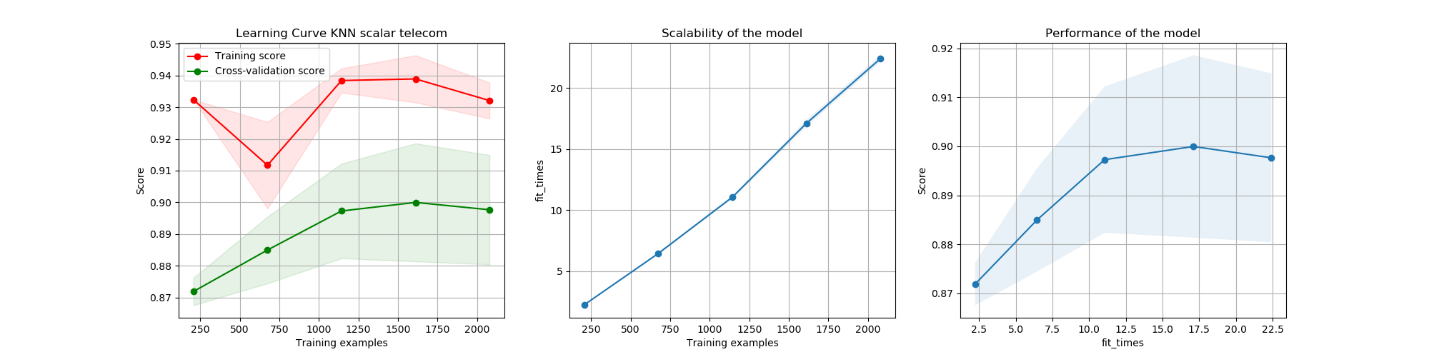


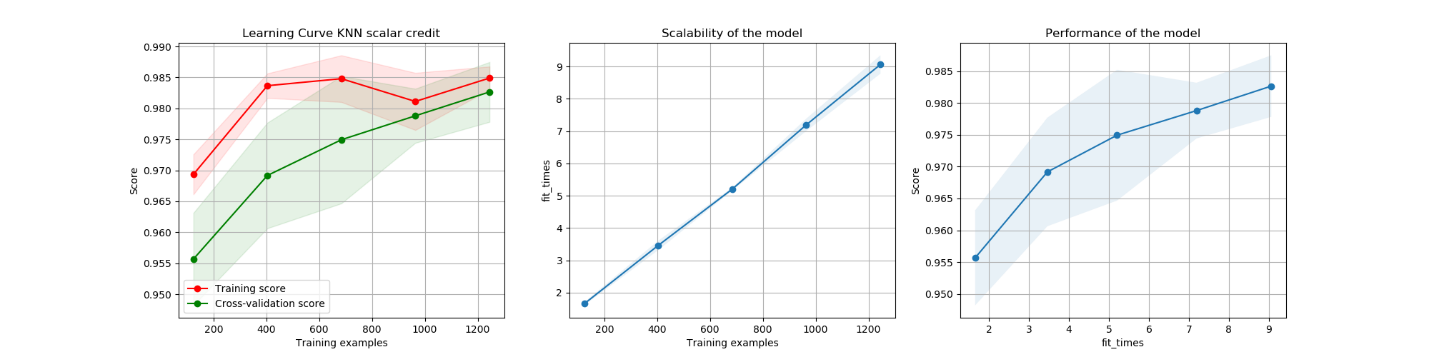
This graph is the results of hyperparameter tuning for the telecom data. It contains accuracy and recall scores for different values for alpha as well as max depth. As max depth increases there is an initial increase in both the models accuracy as well as the recall. This tapers off around 8 and slightly decreases from then on. The only reason it does not decrease steeper is because the cross complexity pruning is pruning the tree when it gets too deep. At first, the tree is underfit, and that is shown by the fact that the tree is barely performing better than 85%, which is the results that would be given on average for just guessing that a customer will not churn. This shows how allowing some growth of the tree allows it to fit the data better and that the sweet spot for this data lies between 6 and 10. For this sampling of data, 8 was selected by the hyperparameter tuning process that was utilized. The CCP alpha value tells a similar story but in an opposite relation. If the value is too small, then overfitting occurs. Of course, in this example, that is capped by the fact that a max depth of 8 was utilized while tuning the value for alpha. When alpha is too large, there is a sharp decrease in accuracy and a even sharper one in recall, this means that most of the accuracy loss is from misidentifying one class, and in this case, that class is the underrepresented one, churn. There is also a floor that is hit for the overall accuracy that can be observed when the recall for churn reaches 0. This is a point where the tree has been pruned so much that it only guesses no churn and has a baseline accuracy of 85%. It seems the best the decision tree can do on this model with the hyperparameters selected is around 92-93% with a recall of the churn class at around 60%. To improve the performance for this decision tree, other hyperparameters could have been messed with to deal with pruning. Additional data with examples of customer churn could have also helped to improve the performance of the trees. Randomness could have also been utilized to select features for splitting which might have allowed the tree to stumble upon correlations that were not show in the gini value. A different problem that was more balanced could have also been used. Also, a problem that was just as imbalanced with similar sample size but less features could have been used like in the next classification problem.

Decision tree credit hyperparameter tuning

A similar relationship between underfitting and overfitting can be seen in these graphs for max depth and alpha as previously described just in different degrees. So instead of redescribing the tuning process and results, I will discuss how having more data and less features affected the process. Right off the bat, it is apparent that the higher number of samples and lower number of features helps the decision tree to perform better. Even with the underfit models the recall score at 2 and 3 is at the ceiling for the recall score in the telecom data. The recall score starts higher and takes longer to reach the floor than in the previous example when it comes to the alpha values and underfitting as well. Overfitting is limited by the choice in hyperparameters as it was previously, but it is also prevented in this case because the splits are more likely to split the data in meaningful enough ways to classify it sooner so having greater depth may not result in overfitting if the splits are better. Overall, the performance increase is clear in both the recall score as well as the accuracy of the decision tree on this dataset and there is not anything I would really look to improve given the performance of the model.

K Nearest Neighbors





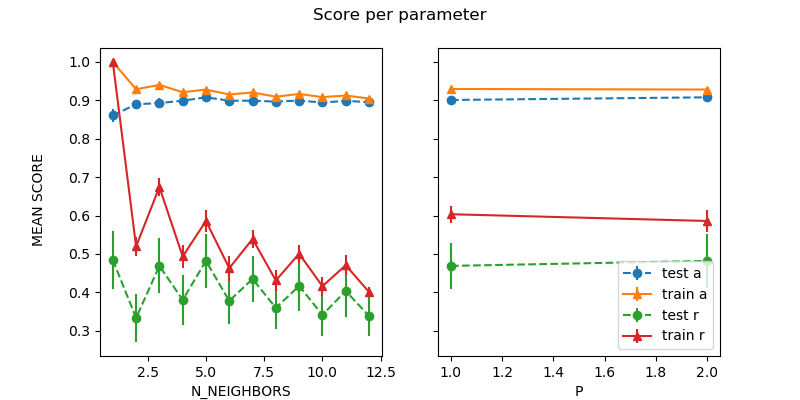
For KNN, we can see results that vary between the two datasets for the provided classification problems. As more samples are given to the KNN model for the telecom data, it shows that the overall improvement in accuracy starts to taper off and the standard deviation starts to greatly vary based on which samples were given for the cross validation and which were used for validation. This highlights that with a bunch of noise adding samples will improve the performance but only marginally for certain thresholds. To see a significant improvement to the levels of the decision trees, it would likely need potentially millions of samples for the numbers of features provided to get results like it did on the credit data. This can be compared to how well and consistently KNN improves with more samples on the credit classification problem since it only has 3 features. The greater number of samples and less features helps KNN improve. The performance for the training sample to fit times for KNN is surprisingly measured as linear. I would have to guess that this is due to the underlying implementation of the fit algorithm used for KNN as it would usually be expected to be a close to constant time operation. I would assume that the storage operations being done are making the overall time increase if a specific storage form is used for the implementation of the model.

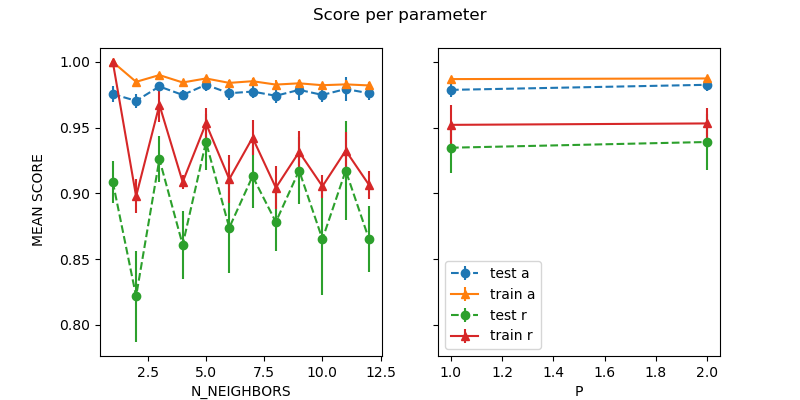
KNN Results

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy Train | Accuracy Test | Number of Neighbors | Distance Metric |
| Telecom | 92.9% | 91.96% | 5 | Euclidean distance |
| Credit | 99.1% | 97.27% | 5 | Manhattan distance |

The results for KNN are shown above. One interesting detail is how well KNN performed on the telecom dataset. From the learning curve for the telecom dataset, KNN can widely vary on this dataset due to the limited number of samples. The accuracy that it got was on the upper end of the standard deviation of the cross validated sets. That shows for this sampling KNN got lucky with the samples that it received to train and test on and that a different slice might have shown substantially worse performance. For the training data KNN did substantially worse than it did in the learning curves for the telecom data. This shows how KNN can be significantly impacted by what slices are given for training when there are a small number of samples compared with the number of features. The credit data on the other hand shows how well KNN can perform when there is a large number of samples and small amount of features.

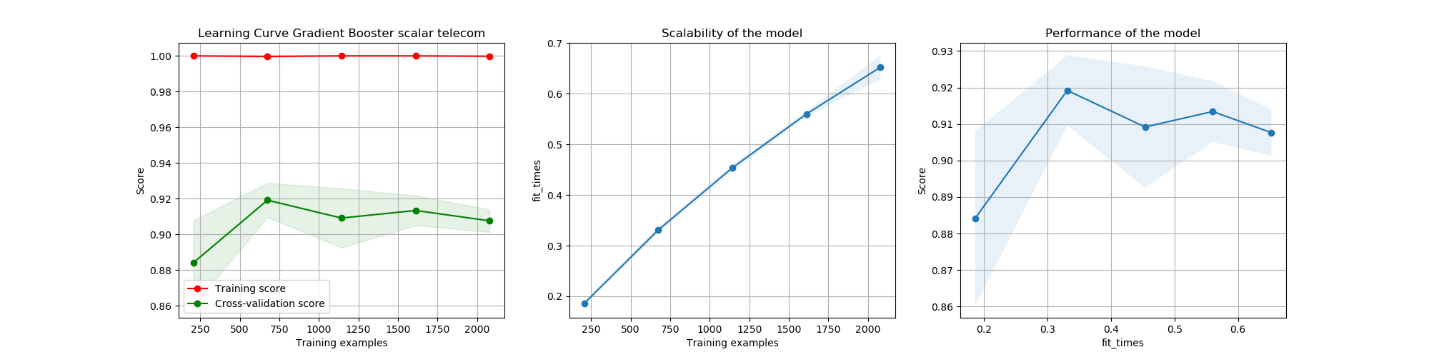
Telecom & Credit hyperparameter tuning

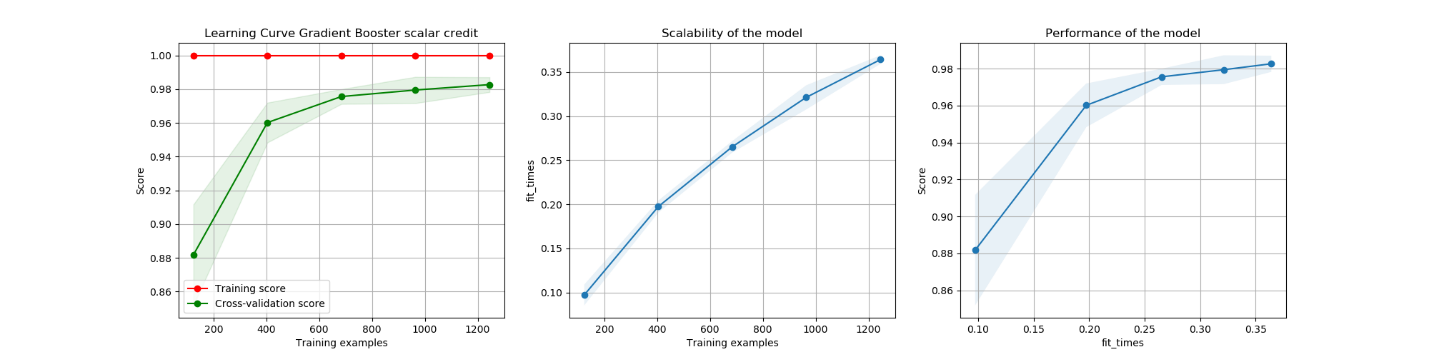




The hyperparameter tuning for KNN shows some interesting results. For both the telecom and credit problems, when KNN picked an odd number of neighbors, the recall score was on average higher. This makes sense given that there are so many of one type of class that picking an even number is more likely to end up with a split between the two class types. When an odd number of neighbors is utilized, it is more likely that the split will lean towards the correct and underrepresented class. Additionally, the higher number of samples and lower number of features for the credit problem shows for KNN a higher floor for recall in relation to the number of neighbors picked. It highlights the benefits of the second dataset’s less features more benefits property. The two also picked different distance metrics for slight improvements in performance. The standard scalar choice for data representation really helped KNN in this case. I tested normalization and the data as is for KNN and it performed significantly worse. For the sake of space those results are not highlighted here, but they were a part of the decision to utilize a scalar data format. With the raw data, KNN performed worse than just guessing.

Gradient Boosting



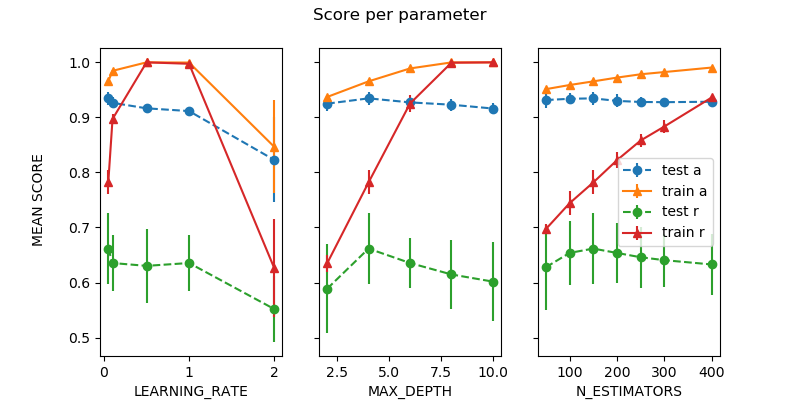


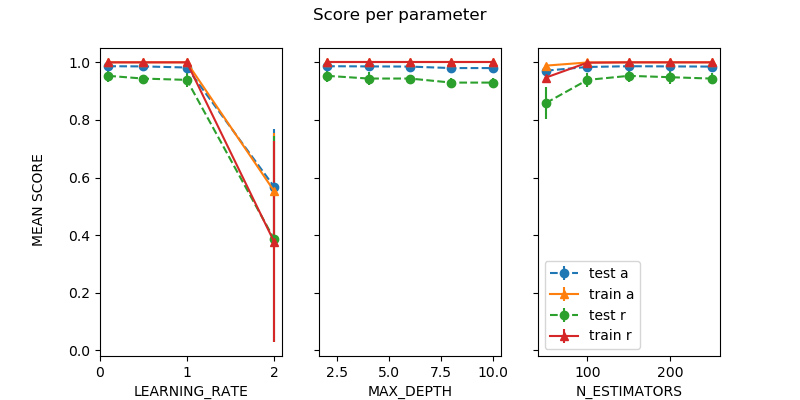
The first thing we can notice about the learning curve for both boosting models is that they do not overfit and always get the training set correct for whatever sample size is provided. The performance as the number of training samples is very similar to that of the decision tree overall, but on the test set, it always gets near 100 percent meaning overfitting will not occur. The scalability of the model is a little rough given that the scalability shown is for each estimator. That means it needs multiplied by the number of estimators to get a true value. It is a similar scalability to that of decision trees just multiplied by the underlying number. Each tree itself is faster to train due to a much smaller max depth, but overall more trees need trained.

Boosting Results

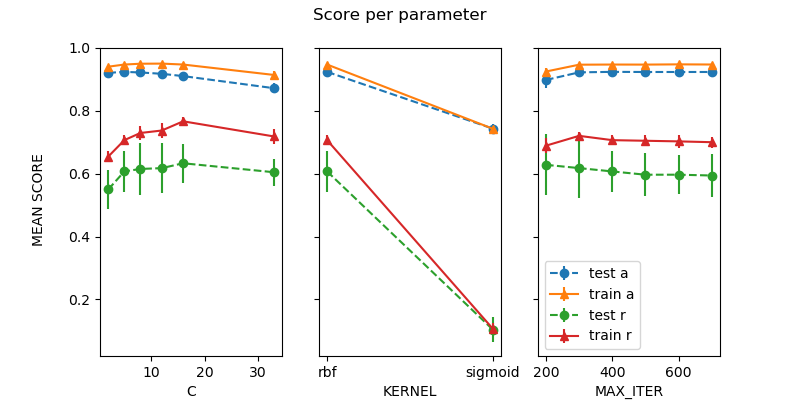
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy Train | Accuracy Test | Learning Rate | Max Depth | Estimators |
| Telecom | 95.5% | 94% | .005 | 2 | 150 |
| Credit | 100% | 98.2% | .01 | 2 | 150 |

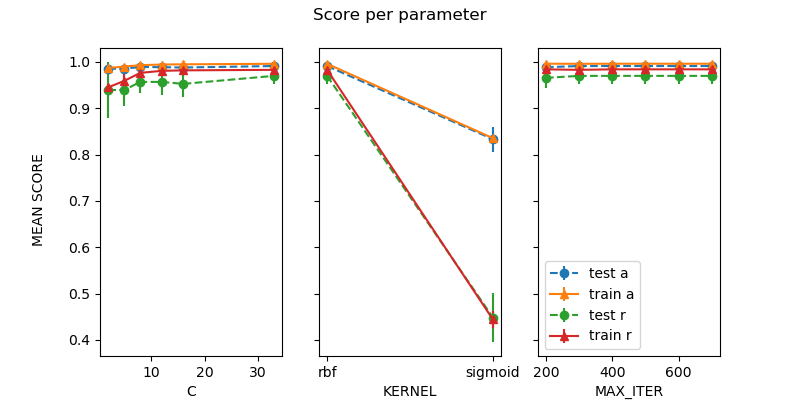
Overall, the boosting algorithm did as well as any of the learners and better than most for both sets of data. It easily did the best for the telecom data. It did as well as any of the other learners on the credit data. It was not given the opportunity to run with a learning rate as low as .005 since that would take a good chunk of time and computing power that was not available at the time this was run. This highlights the pros and cons of boosting algorithms. They can perform as well as any of the other learners given the right learning rate and number of estimators, but it comes at the cost of time and computing space.



 For boosting, the max depth both hyperparameter tuning experiments chose was 2. As max depth increased, the test recall was reduced for both learners. This makes sense since that would likely lead to more results leaning towards the overrepresented classes in each of the problems. Each booster decided on 150 estimators to use. For the telecom data set, it had the option to go up to 400 learners. Something interesting started to happen beyond 150 learners that results in performance degradation overall. Even though the recall score was increasing on the test set, the training set was getting worse. This is definitely a result of imbalance in the data, and it shows that with more learners the boosting algorithm does better on the training set, but with a lack of sufficient data for the number of features, it can cause overfitting to occur. Learning rate was picked to be the lowest value available by each of the models for each problem. The boosting model for credit data had little issues with optimizing parameters and finding a model that performed extremely well. If it had been given the option of more estimators or a smaller learning rate, it likely would have performed even better. I did not implement this due to the performance implications of doing so and the already stellar performance of the booster on the credit classification data. Learning rates of 1 or above performed horribly as would be expected.

Support Vector Machines

Hyperparameter tuning results for SVMs



For SVMs, max iterations will be looked at in relation to accuracy instead of sample size. The learning curve for the telecom problem shows that smaller iterations result in less accuracy and as more iterations are allowed the accuracy increases. This happens at first at least, but the curse of dimensionality strikes again. Further separating the data beyond 400 iterations results in our radial basis function slowly undoing its own progress while trying to find a better way to separate the remaining classes. The model does relatively fine for the credit data for any number of iterations beyond the initial 200. This data is much simpler with plenty of examples and values to play with. The hyperparameter tuning only picked 300 iterations likely due to the fact that there was not much difference at higher numbers, and it likely caused some misclassifications of the overrepresented class of not defaulting on a loan which would impact the overall accuracy more than any slight gains in classifying the underrepresented class. The values for C greatly varied for the two classification problems. C was only 5 for the telecom data where it was 33 for the credit data. The data must have been more easily separated by the radial basis function for the credit problem than it was for the telecom data. A value that large for C indicates that there are a lot of support vectors that could be picked and utilized to split the data up. It likely indicates that the data is not easily overfit. The number of samples and features can help to make those difference clearer, but this also shows the value in hyperparameter tuning for different data for a learner as a value of 33 seems high on first look. It does show a significant improvement though on the graph.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Accuracy Train | Accuracy Test | C | Kernel | Iterations |
| Telecom | 96.5% | 92.5% | 5 | Radial basis function | 400 |
| Credit | 99.7% | 98.8% | 33 | Radial basis function | 300 |
| Telecom | 85.5% | 85.5% | 5 | Sigmoid | 500 |
| Credit | 87.5% | 86.5% | 8 | Sigmoid | 400 |

In terms of kernel functions, the radial basis function was the function that performed best for both models. That is likely because the two classification problems were not linearly separable and sigmoid also did not fair well on them. It pretty much performed at the rate of just picking all 1. That makes sense as the sigmoid function predicts the probability of an outcome and it likely was mainly predicting the probability to be 0 since most of the features would lean that way in the two imbalanced classification problems that were utilized.

Looking at the results for the SVMs, the radial basis function did well on the credit data. It did decently well on the telecom data. It is interesting, because if we look at the recall, in a similar way to the boosting model, when the SVM does better on recalling the churn samples from the telecom dataset, it does not select those hyperparameter, because it decreases the accuracy overall too much. A way that the SVM performance could have been improved would have been with different kernel functions such as polynomials. Another way, could have been to reshape the data using normalization or leaving the data as is without the scaling to see if any of the kernels would do better with the data in that form. Higher values for C could have been tried as well to see if the credit data had improved with even more support vectors.

Neural Nets Results

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Layers | Accuracy Train | Accuracy Test | Nodes | Activation |
| Telecom | 1 | 89.8 | 90.1 | 4 | sigmoid |
| Credit | 1 | 1.0 | 99.3 | 12 | sigmoid |
| Telecom | 2 | 95.1 | 91.6 | 12 | Relu |
| Credit | 2 | 1.0 | 99.8 | 96 | tanh |
| Telecom | 3 | 92.7 | 90.7 | 12 | Relu |
| Credit | 3 | 99.9 | 99.5 | 4 | sigmoid |

For Neural Nets, all the results in the table above were limited to 200 iterations. While working with imbalanced data and in one case limited samples compared to the number of features, I thought it would be a good experiment to see how different activation function, node, and layer combinations would be able to do. For the credit data, the neural nets seemed to be easily able to figure out the relationship in the features and their weights and map them appropriately. The credit data utilized different node numbers for each number of layers. It used 12 then 96 and last 4 for layers 1, 2, and 3 respectively. The activation functions also differ for the different layers. On the layers with fewer nodes, the activation function was the sigmoid function. This is interesting because it smooths the data and turns it into a relative probability of whether to fire off. If the neural net was able to figure out weights for the nodes that properly weighted the features it was training on, then the sigmoid function becomes a quick validation of whether those are mapped properly or not. When the iterations are limited to 200, this could easily result in a better chance at converging than the other activation functions provided. Interestingly though for the 2 layer neural net, the hyperbolic tangent function was utilized with a whopping 96 nodes. The hyperbolic tangent allows for a greater variance in the values for the activation function outputs as well as a slightly different derivative for the back propagation of the weights. This function must have outperformed the sigmoid function on the 2 layer neural net.

Criticism balance accuracy could have been used for hyperparameter tuning and even learning curve