Assignment #1: Supervised Learning

Introduction and Dataset Description

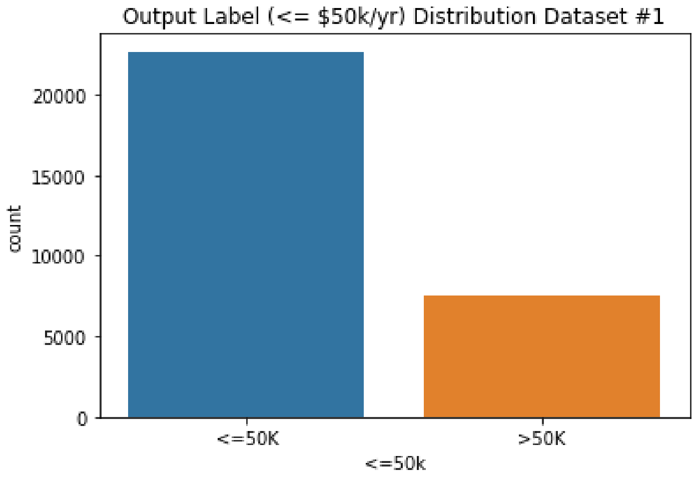
I have selected two datasets to compare the performance of various machine learning models on their output. Both datasets were collected from UCI's Machine Learning Repository and are classification problems with a non-trivial data size.

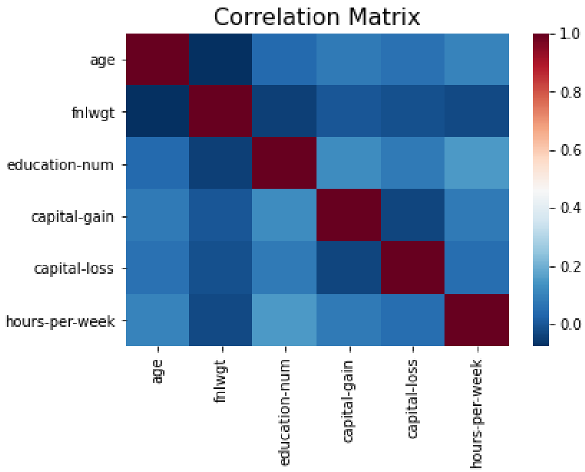
Dataset #1: Adult[1]

According to the dataset description[2] it contains features which attempt to classify a binary label which is whether or not the example "makes more than $50k USD" per year.

What makes this dataset/problem "interesting?"

This dataset is interesting due to its size (> 40k rows) which allowed me to cover a significant portion of the space made up of the various attributes. There are 14 attributes in this dataset, 7 continuous valued attributes and 7 categorical attributes. In regards to data distribution with respect to the target attribute I am classifying, there are approximately 3x as many samples in the category of "less than 50k per year" than in the other category. As such, I will be using a "stratified" sampling technique when I split my training and testing set for hyperparameter tuning and selection using k-fold cross validation. An investigation of the coefficient of correlation for the numerical attributes reveals that there is a very low correlation between any of them. Their relative linear independence from each other will most likely not help with classification but I intend to leave them in for now. The figure depicting the correlation matrix heatmap is in the figure to the right.





Dataset #2: Banking[3]

According to the dataset description this comes from the marketing campaign of a Portuguese bank in the 1990s. They were intending to sell their customers on a product called "bank term deposit" [4]. The attributes describe various aspects of each customer including details such as age and employment, as well as that

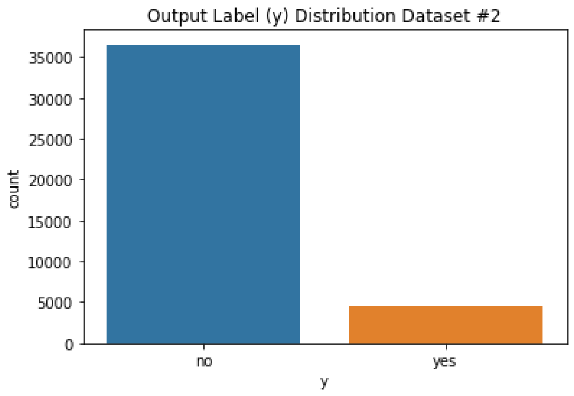
[1] Location: http://archive.ics.uci.edu/ml/datasets/Adult

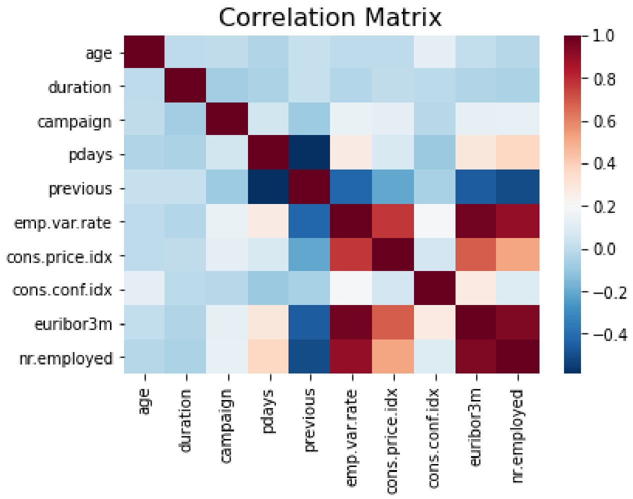
[2] Location: http://archive.ics.uci.edu/ml/datasets/Adult

[3] https://archive.ics.uci.edu/ml/datasets/bank+marketing

[4] https://archive.ics.uci.edu/ml/datasets/bank+marketing

customers financial history with the institution and some other details like previous marketing campaign results and economic indicators such as the euribor 3 month note daily rate and consumer price index. This is also a classification problem with the outcome being measured as whether or not they eventually purchased the product in question (binary classification.) This is a particularly interesting classification problem for a few reasons. Given the increased feature space and the decreased example count I expect certain algorithms to perform worse due to the “curse of dimensionality” mentioned before. There is an even split between categorical features (10/20) and continuous ones (10/20) in this dataset. In the figure to the left you can see that there are approximately 7x more examples of someone not signing up for this product (as indicated by the “y” value) than those that do. This is a fairly unbalanced dataset for binary classification, and similar to the above dataset we are going to be using a stratified technique when we do cross-validation to ensure that each “fold” has a roughly equivalent distribution of samples. The figure to the right depicts the correlation matrix between each of the numerical attributes in this dataset. Relative to the previous dataset there is a slightly higher correlation between some of the features in this dataset. This could prove to be noise that could cause my classifiers to overfit to the training data. Outside of the slightly increased correlation is one tangential observation. There seems to be a relatively high correlation between the Euribor 3 month note and the number of employees at the bank, probably indicating more about hiring practices than the actual variable I am trying to classify. This will most likely add noise that the binary classifiers will have to overcome.





Experiment Setup

For the purpose of this paper I am comparing 5 separate supervised learning algorithms: Decision Trees with Pruning, Neural Networks, Boosting, k Nearest Neighbors, and Support Vector Machines. This section will explain common experiment parameter definitions and approach to evaluating outcomes. There will be 10 separate experiments performed, one for each dataset and learning algorithm combination. Unless explicitly overwritten in the specific algorithm analysis, the output metric I will be optimizing for each binary classification problem's model is accuracy as defined by the Jaccard Index[5]:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Where A = The labeled samples by the classifier, and B = The correctly labeled samples (ground truth.) Since some algorithms in scikit learn do not support categorical attributes, I will be using Pandas' `get_dummys()` function to convert my categorical attributes to binary ones.[6] The reason for doing this and not the OrdinalEncoder() of scikit learn is because I don't want the algorithm to treat my ordinally encoded integer columns as continuously valued as that could produce potentially incorrect logic as output. This does significantly increase the number of attributes (105 in the case of Dataset #1) and may have impacts on training times/accuracies. In addition to expanding the categorical features to be binary encoded using pandas, I chose to use the StandardScaler() of scikit learn to ensure that numerical attributes would be normalized in order to improve the training time of some of the algorithms. Before each experiment I will split the data into a training and test set where the test set is 20% of the available samples and the training set is the remaining 80%. Performance on classification of the test set will be how each set of hyperparameters to the classifier are evaluated, and I will keep the set of parameters with the highest accuracy score as defined above. After preprocessing, for each algorithm I designed an exhaustive Cross-Validated GridSearch for the best hyper parameters to optimize output accuracy over the k different folds. This GridSearch was then performed n times to vary the number of folds to cross-validate where n is an integer between [3, 10). This allowed me to also analyze the impact that data size would have on training accuracy because the amount of data being withheld was different for each hyper parameter tuning GridSearch run. The rest of this paper is organized as follows: each learner will have a section dedicated to its description, its performance and some analysis related to its own parameters on each dataset. At the end of this paper I will provide some overall analysis comparing the algorithms' performance to each other and a final summary. For all computations in this paper, I am using Python's Scikit Learn library and associated data science tools for the implementation of each of the following algorithms.

Decision Trees with Pruning

The decision tree implementation I am using from SciKit Learn implements the CART algorithm[7] for how it constructs trees. According to the description it is an augmentation of the C4.5 algorithm (which is itself an improvement of ID3) in that it allows for regression trees to be built as well. It also doesn't compute rule sets (the second half of C4.5) so presumably we will have to achieve the pruning of trees another way since this is how C4.5 will prune its trees. In searching for hyper parameters for the Decision Tree, I have varied the following: the criterion

[5] https://en.wikipedia.org/wiki/Jaccard_index

[6] https://github.com/suvoooo/Machine_Learning/blob/master/DecsTree/notebooks/Bank_Data_Analysis.ipynb

[7] https://scikit-learn.org/stable/modules/tree.html

for choosing a next best feature to split on between Gini Impurity and Information Gain (as measured by Entropy,) whether to choose a random split in the data or computationally search for the best value to split on, and finally the maximum depth of the tree between 3 and 15 decision nodes, and finally the Cost of Complexity pruning parameter alpha between 0.0001, and 0.25. I achieved pruning through the use of two hyper parameters to control tree complexity, the max_depth parameter and the Cost Complexity Pruning Alpha variable.

Learning Results:

Dataset	Accuracy **(Train)**	Error Rate **(Train)**	MCC **(Train)**	Accuracy **(Test)**	Error Rate **(Test)**	MCC **(Test)**
Dataset 1 (Adult)	85.43%	14.56%	0.5813	84.78%	15.21%	0.5617
Dataset 2 (Banking)	92.02%	7.97%	0.5798	91.56%	8.43%	0.5464

Optimal Hyper Parameter Values:

Dataset	Criterion	Max Depth	Splitter	CCP Alpha
Dataset 1 (Adult)	Gini	7	Best	0.0001
Dataset 2 (Banking)	Gini	7	Best	0.0001

Discussion:

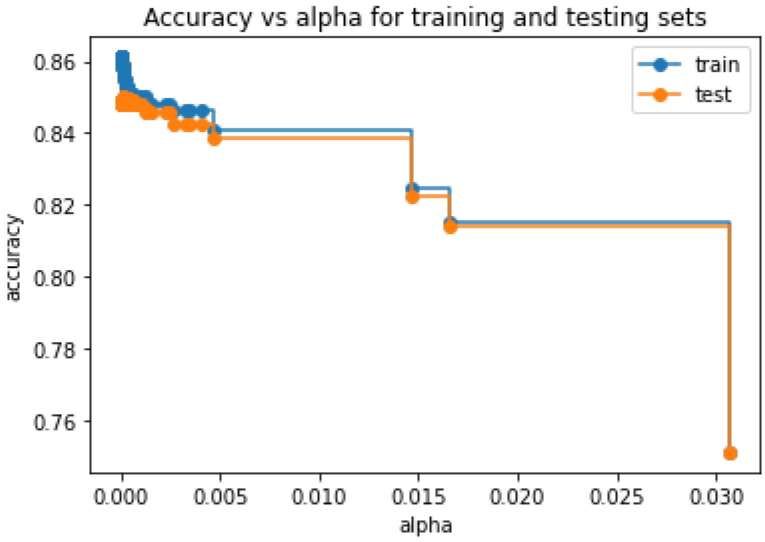
In addition to accuracy rate I also chose to measure the Matthews Correlation Coefficient[8] because it provides a better performance indicator than the traditional F1 score because it gives a better measure when there is a disproportionate number of examples in each class (as is the case in both of my datasets.) Given that, both classifiers did better than the average random guess (MCC=0) and training and test values were close together indicating a good overall generalization with respect to the testing data. Between the two datasets the learners had remarkably similar best hyper parameter values. Given the ground truth I had for each dataset from their description about the best accuracy achieved, I am satisfied with the result I got that it was the same as the best possible values known at the time. The parameter that had the largest impact on tree accuracy was the Cost Complexity Pruning parameter. Cost complexity calculates a value at each node of the complexity of its subtree using this formula:

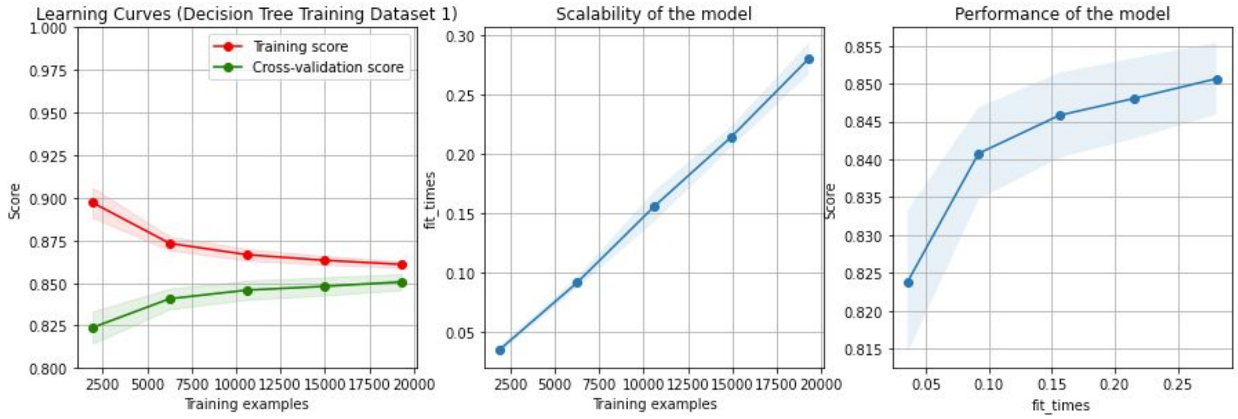
$$R_\alpha(T) = R(T) + \alpha|T|$$ [9]

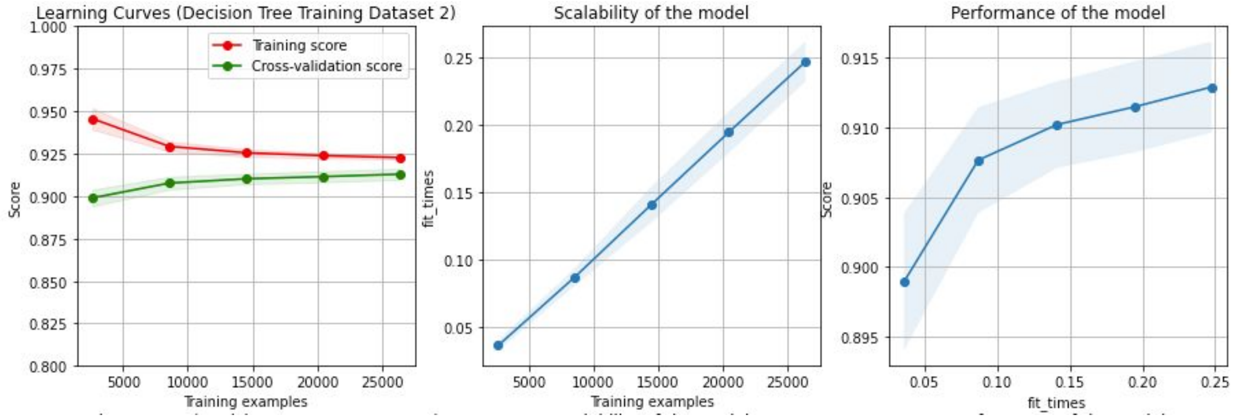
Where R(T) is the misclassification rate of all the terminal nodes T in the subtree at this particular node, and where |T| is the number of terminal nodes multiplied by some non-negative real valued parameter we vary called “alpha.” This value allows us to penalize a tree for its complexity and therefore force the learner to arrive at a decision node in a shorter distance, the

[8] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.matthews_corrcoef.html
[9] https://scikit-learn.org/stable/modules/tree.html#minimal-cost-complexity-pruning

goal being to generalize this decision tree to unseen data. Here is a graph that shows varying this parameter alpha and the effect on the overall accuracy of the mode. (note that larger values of alpha should produce a more inaccurate tree because it is penalizing complexity more heavily.) As you can see below, as alpha increases, the complexity decreases, but so does the tree size, the final value for alpha was 0.073 and resulted in a tree with only three nodes and an accuracy barely better than a coin flip. Below is how the classifier performed relative to cross validation and "wall clock time" in both testing and training sets. The classifier appeared to be growing in roughly linear time with the number of training examples and logarithmic in terms of accuracy. Even on a large number of samples (10's of thousands), training times usually took roughly 300ms.







As you can see from the leftmost above graph, the red line depicts training accuracy going down as the classifier gets more examples during a cross validation with the testing set. As examples increase the accuracy of the model with respect to the testing data is going up. For each dataset the learner used GINI impurity to pick the next best attribute to split on. GINI assigns a value between [0, 1] to each attribute where the value is how often a randomly chosen element from that attribute would be misclassified.[10] In both datasets, the learner chose GINI impurity as the better splitter when it came to choosing the next attribute.

Artificial Neural Network

The ANN implementation[11] I am using is a multi layered perceptron network that trains in an iterative manner optimizing a log-loss function using one of two approaches (SGD or L-BGFS) at each step to provide new values to the perceptrons. I performed a grid search over the following hyper parameters to search for optimal values: Activation function (Logistic Sigmoid, Hyperbolid tan(x), Rectified unit linear), Learning Rate Initial value (0.001-0.1), the weight optimization solver (ADAM-SGD, and SGD), and Max iterations (50-200). The results of hyper parameter tuning can be found in the table below for each dataset.

Learning Results:

Dataset	Accuracy **(Train)**	Error Rate **(Train)**	MCC **(Train)**	Accuracy **(Test)**	Error Rate **(Test)**	MCC **(Test)**
Dataset 1 (Adult)	75.14%	12.20%	0.0316	75.18%	24.81%	0.0500
Dataset 2 (Banking)	89.57%	10.42%	0.4883	89.25%	10.74%	0.4734

Optimal Hyper Parameter Values:

Dataset	Activation Function	Learning Rate Init	Solver	Max Iteration
Dataset 1 (Adult)	Rectified Linear Unit	0.1	Stochastic Gradient Descent	50
Dataset 2 (Banking)	Logistic Sigmoid	0.005	ADAM Stochastic Gradient Optimizer	50

Discussion

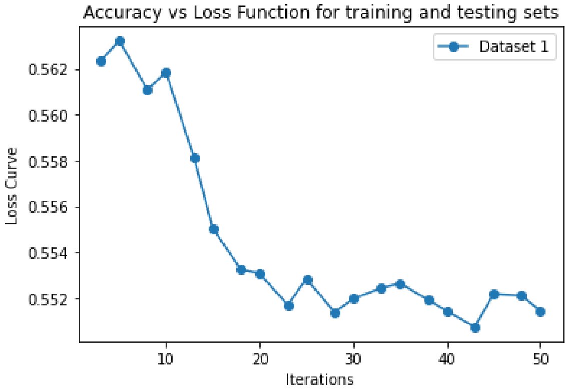
Out of all of the learners I trained, the ANN performed the worst in terms of both accuracy on both the training and test sets. After varying other parameters like hidden layer size and node count in each layer, performance decreased. A look at the MCC values indicate this mode was simply random guessing, and a look at the confusion matrix generated by dataset one reveals the most obvious thing this classifier did:

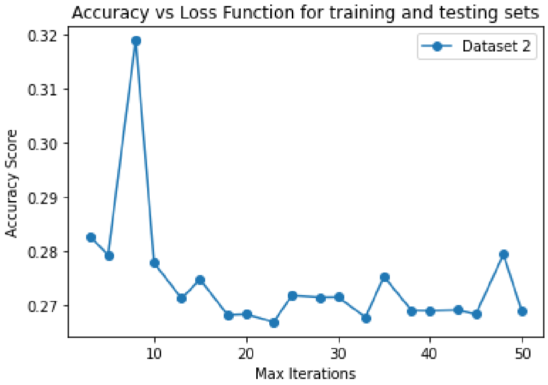
Training Confusion Matrix	Testing Confusion Matrix

[10] https://medium.com/@jason9389/gini-impurity-and-entropy-16116e754b27
[11] https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html#sklearn.neural_network.MLPClassifier

0 1
0 8 5998
1 0 18123
	0 1
	0 5 1497
	1 0 4531

With the exception of a few examples, the classifier basically only guessed 1 effectively misclassifying all of the values for which the label “makes less than 50k per year” was false.
Based on my observation I believe this learner has higher bias and is therefore underfitting relative to the training data. To combat this, I will either need to add more training data, alter the feature space (by adding/dropping/manipulating features), or manipulate the hyperparameters more[12]. Given that MLPs are sensitive to feature scaling I have scaled the numerical attributes before beginning training.





On the left hand graph above we have Dataset 1’s max iteration values for the neural network vs its loss value (the function it is seeking to minimize) as you can see, the loss function is decreasing but not by significant amounts and at about 20 iterations it doesn’t improve significantly beyond that. On the other graph, we can see that the loss changes more dramatically for lower iterations, and there are two outliers at about 5 and 20 iterations of this learner. Given that the overall trend is towards loss minimization, I assume that what is happening is that due to the “stochastic” part of gradient descent, we can see the learner select a direction and move in a non-globally optimal direction and have to change again. Since iterations after a large spike seem to still be moving downwards we can assume that the learner is still moving in the optimal direction. This level of zoom highlights SGD/ADAM, but if we were to zoom out we would see an overall trend towards the best the network could do in terms of accuracy. Varying the number of nodes and the number of hidden layers did not appear to make a significant difference on the accuracy of either dataset, and in the case of dataset 1 it made the accuracy worse in certain configurations. The two classifiers ended up using different activation functions for which performed better. Dataset 1 used the Rectified Linear Unit Function and Dataset 2 performed better with the Logistic Sigmoid. The Rectified Unit Linear function is gaining popularity in the Deep Learning community due to the fact that for network weight values that need to be truly zero this is supported (unlike the sigmoid which can be a

[12] https://www.quora.com/Why-and-what-to-do-when-neural-networks-perform-poorly-on-the-training-set

value close to, but not exactly zero.)[13] It is also better due to the fact that it is computationally faster to use because it only requires the use of a max() function. I would take the ReLU result for Dataset 1 with a grain of salt though looking at the accuracy results. It seems this problem wasn't well suited for a neural network. As for dataset 2, the sigmoid has been the default activation function in the supervised neural network community for a while.[14] As for why it performed better on this dataset, based on my research, the search for a good activation function is more about computational load than it is accuracy, so realistically all functions did reasonably well on this dataset but this one did slightly better than the others.

Boosting

For bootstrap aggregation I chose to use the AdaBoost ensemble learner made up of many DecisionTreeClassifiers (the class from the section above on decision trees.) I tuned the following parameters: the base estimator (i.e the decision tree params), the algorithm used for boosting (SAMME, SAMME.R), the number of trees to add to the estimation, and the learning rate between 0.1 and 1.0.

Learning Results:

Dataset	Accuracy **(Train)**	Error Rate **(Train)**	MCC **(Train)**	Accuracy **(Test)**	Error Rate **(Test)**	MCC **(Test)**
Dataset 1 (Adult)	87.79%	12.20%	0.6593	87.27%	12.72%	0.6593
Dataset 2 (Banking)	91.92%	8.07%	0.5626	91.09%	8.90%	0.5230

Optimal Hyper Parameter Values:

Dataset	Algorithm	Base Estimator	Learning Rate	N Estimators
Dataset 1 (Adult)	SAMME.R	DTC(Max Depth of 3)	0.5	60
Dataset 2 (Banking)	SAMME.R	DTC(Max Depth of 2)	0.5	50

Discussion:

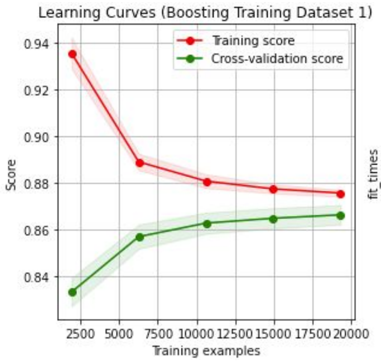
The Boosting did the best on dataset 1 and actually outperformed what was listed in the dataset description[15] as the best possible achieved accuracy. This really speaks to the power of the combination of many weaker learners who over time focus on harder to classify samples. Below is the learning curve and fit times of the training examples for Dataset 1, next to a graph of the varied hyperparameter "N Estimators" for both decision trees between [3, 60]. As you can see the converging graphs on the far left make it appear that the learner is generalizing well over the data and that there isn't much variability between the different k-folds in the cross-validation

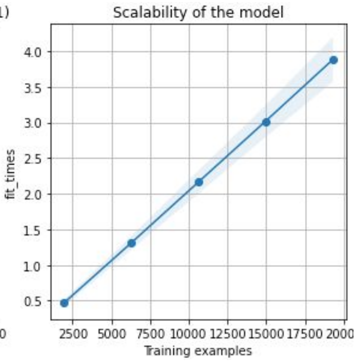
[13] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

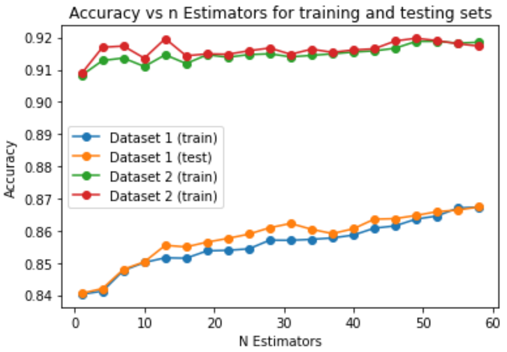
[14] https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[15] https://archive.ics.uci.edu/ml/machine-learning-databases/adult/

(given as the hue around the lines.) The learner appears to scale linearly with the number of examples in both datasets (even though only one is pictured here.) On the right hand graph you can see that as n estimators grew, the accuracy grew for both learners and there was not much divergence between the testing and training accuracies suggesting a good generalization. The more interesting thing to note is that Dataset 1 appeared to benefit from ensemble learning more than Dataset 2, suggesting that the number of difficult to classify instances could be higher, or the function attempting to be learned did better with many different trees as opposed to the previous neural network where it only guessed one class label. To achieve pruning I initialized the DecisionTreeClassifier passed to each Boosting function to have a max_tree height. By varying this parameter in a grid search for the best values I was able to find how aggressive I could be and still achieve high accuracy with many weak learners.







KNN

For k-Nearest Neighbors, I varied the following parameters to search for an optimal learner to fit my dataset: N Neighbors (the number neighbors to query), and the weights of each point on the prediction between uniform and distance weighted such that each contribution only had as much influence on the prediction proportional to the amount of distance it was from the point being predicted. I purposefully chose not to vary the algorithm in this case between Ball Trees and KD Trees after doing significant research into the differences between the two. Because KD Trees struggle with high dimensional spaces in terms of computational efficiency and storage, I assumed that Ball Trees would be used on both datasets given their size and dimensionality (105 features after dummying in the case of Dataset 1 and 63 in the case of Dataset 2).

Learning Results:

Dataset	Accuracy **(Train)**	Error Rate **(Train)**	MCC **(Train)**	Accuracy **(Test)**	Error Rate **(Test)**	MCC **(Test)**
Dataset 1 (Adult)	80.80%	19.19%	0.4135	78.05%	21.94%	0.3073
Dataset 2 (Banking)	100%	0%	1.0	90.71%	9.28%	0.4857

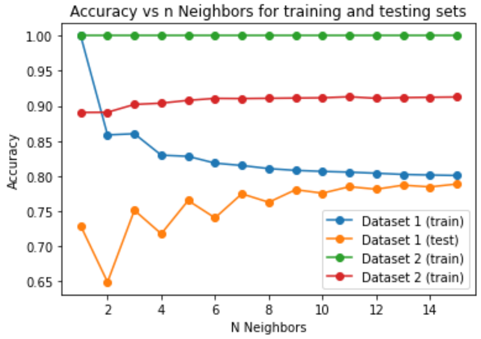
Optimal Hyper Parameter Values:

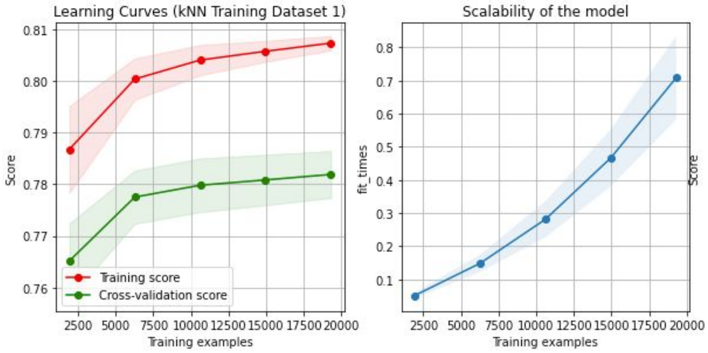
Dataset	N Neighbors	Weights

Dataset 1 (Adult)	15	Uniform
Dataset 2 (Banking)	4	Distance

Discussion:

It seems based on initial results that the kNN overfit to the training data in the case of dataset number 1 and dataset 2 kNN did slightly worse than other “eager” learners such as Boosting and Decision Trees. This could be due either to the data structure the kNN were stored in the computer or due to the data distribution itself. Let’s look at both datasets' accuracy when we vary the number of neighbors.





As you can see, the first dataset benefits the most from increasing the number of neighbors as the training accuracy decreases from having more overfitting to a better fit as it eventually converges with the testing set accuracy at n=15. Dataset 2 however remains at a perfect training accuracy level for n > 3 and its testing accuracy remains constant below no matter how many cross-validation runs are done. Looking at the learning curve we can see that the standard deviation of the folds is a lot wider than other algorithms suggesting that there is higher variability in the predictions given by this learning algorithm for both datasets. In terms of wall clock time, this is the only algorithm that has a non linear scalability with the amount of data and instead looks exponential. This is most likely due to the fact that kNN is a “lazy learner” and needs to do all work at query time. Interestingly the datasets had different values for how weights were assigned to queries suggesting that dataset 1 benefited from each point contributing an equal vote to the class value of a point to classify where as dataset 2 benefited more from each point only contributing the portion of its vote proportional to its distance. In order to get better results in the future I will most likely have to increase the dataset size or vary the types of examples.

SVM

For Support Vector Machines, I used the Support Vector Classifier implementation in Sckit Learn. I varied the following hyper parameters in search for the best fit in the data: Kernel (Radial Basis Function, Poly, and Sigmoid), Gamma (scale, auto) where it scales the feature by either 1/n number of features or 1/n times the training data variance, Criterion for stopping tolerance between 0.001 and 0.1, and finally whether or not to balance the class weights. Below are the results for both datasets.

Learning Results:

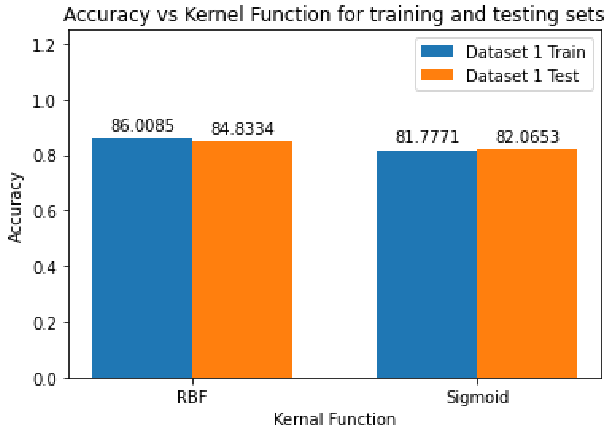
Dataset	Accuracy **(Train)**	Error Rate **(Train)**	MCC **(Train)**	Accuracy **(Test)**	Error Rate **(Test)**	MCC **(Test)**
Dataset 1 (Adult)	96.49%	3.50%	0.9057	75.00%	25.00%	0.0962
Dataset 2 (Banking)	89.73%	10.26%	0.3262	89.93%	10.06%	0.3444

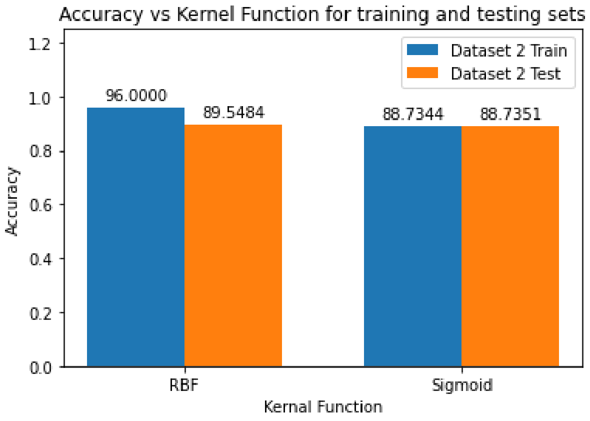
Optimal Hyper Parameter Values:

Dataset	Kernel	Tolerance for Stopping Criteria	C	Class Weight	Gamma
Dataset 1 (Adult)	RBF	0.1	1.0	None	Scale
Dataset 2 (Banking)	RBF	0.001	1.0	None	Auto

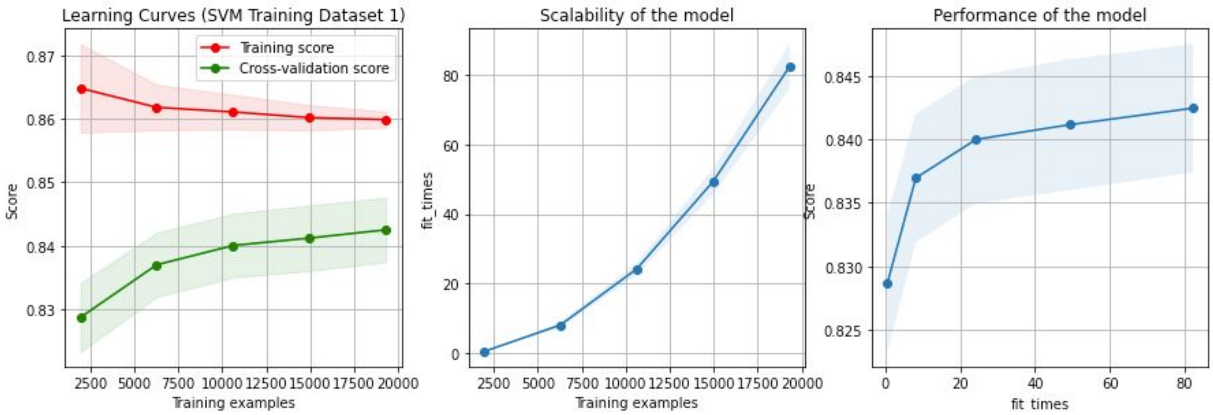
Discussion:

The SVM took by far the most time to train out of all of the learners, taking well over 5 hours to do a grid search to find the best parameters. Above in the table are the values that I varied in a grid search. Scaling the features dramatically improved performance in terms of CPU time. Outside of the tolerance value for stopping criteria, the next parameter that had a significant impact on accuracy during my experimentation was the kernel function. Below is a graph for both datasets and the accuracy difference between the Radial Basis function and Sigmoid function.





As you can see, there is a small difference in accuracy when we use the sigmoid (slightly worse) than the radial basis function. Presumably this has to do with the decision boundary presented by the data and the examples. Looking at the cross validation of the SVM should reveal more about the computational complexity of arriving at a solution. It is worth noting that that the SVM performed almost as well as other learning algorithms but with significantly increased computational complexity and time required to train them.



From the above learning curve you can see that more data is needed for the SVM to converge and that in testing sets the different folds in k-fold cross validation do not appear to get better as more examples are provided. The model scales exponentially in training time with the number of samples and only logarithmic in terms of amount of time and accuracy.

Analysis, Comparisons, and Summary

Using the Jaccard index as the accuracy measure, the algorithm that performed the best was the Boosting learner on Dataset 1 which used the AdaBoost implementation on top of the decision tree classifier whereas for Dataset 2 the pure Decision Tree classifier performed slightly better than all others. Dataset 1's best learner had an overall accuracy of 87.27% on the testing dataset not used to train the classifier and Dataset 2 scored 91.56%. I suspect this is due to the nature of ensemble learning. The many individual trees were able to focus due to the weighting of incorrect samples towards the end of training on "harder" to classify samples. The large datasets we used had many samples which likely produced edge cases on the decision tree boundary, therefore by having many and using voting they achieved a better testing accuracy than even that provided with the dataset as ground truth. Overall all of the algorithms came within reasonable levels of tolerance (+/- a few percentage points) from the best values known to the owners of the dataset. The ANN algorithm performed the worst on dataset 1, where the learner learned network weights that essentially forced all predictions to be one value regardless of input except in a few cases. The classifier which did the worst on the testing dataset for dataset 2 was the artificial neural network. In terms of performance attribution both accuracy and computational I will talk about the problems I chose and their complexity last. Since both problems were binary classification problems whose data was skewed towards one class on output, a not insignificant amount of computation had to be taken to stratify data to ensure that the proportions were kept in all phases of training and evaluation. Since on the whole binary classification problems are easier due to the fact that there are two possible class values, likely it was easier to get to optimal parameters for the learner and a casual observer of the graphs above could make the case that there are other optimal combinations of parameters and not just one to achieve the most accurate cross-validated result where this may not equally be the case in a regression problem. In summary, we have explored two different datasets/problems and run multiple machine learners on them searching for the best results as well as discussing parameters and their impact on the overall accuracy of the learner.