ML Report 1: Supervised Learning (Spring 2019)

Zachary Pearson (User ID: zpearson7)

Experiment 1: Introduction and Description of Dataset

For my first experiment, I decided to use the UCI dataset "default of credit card clients". The dataset consists of the following attributes:

ATTRIBUTE	DESCRIPTION
LIMIT_BAL (integer)	Client Credit Limit
SEX (integer)	Gender (1 = male, 2 = female)
EDUCATION (integer)	Education (1 = graduate school; 2 = university; 3 = high school; 4 = others)
MARRIAGE (integer)	Marital status (1 = married; 2 = single; 3 = others)
AGE (integer)	Age (years)
PAY_N (integer)	N different columns (0 through 7) repayment status for 7 different months (Apr. 2005 to Sept. 2005). The measurement scale for the repayment status is: -1 = pay duly; 1 = payment delay for one month; 2 = payment delay for two months; . . .; 8 = payment delay for eight months; 9 = payment delay for nine months and above
BILL_AMT_N (integer)	N different columns representing amount of bill statement for N months from Apr. 2005 to Sept. 2005
PAY_AMT_N (integer)	N different columns representing payments made by month for N months from Apr. 2005 to Sept. 2005
DEFAULT PAYMENT NEXT MONTH (integer)	Whether or not the customer defaulted in the subsequent month (encoded as 1 for yes and 0 for no)

The dataset contains 23364 negative examples (where the client did not default) and 6636 positive examples (where the client did default).

My initial interest in this dataset was driven by two pieces of information in the description:

- First, the claim on the dataset website that only an artificial neural network (ANN) was the only effective method for "accurately estimating the real probability of default". Given the goal of evaluating multiple supervised learning methods against one another, a dataset that is purported to only have one effective method was of particular interest, because we can use this analysis to either confirm or deny that claim.
- Second, a cursory exploration of the data revealed that their description of the X attributes as integers was not entirely accurate; there was one categorical variable that, while encoded as integers, did not have any inherent ordering (marital status), as well as one that did (education level). My hypothesis was that one-hot encoding those categorical variables could potentially yield better results, because an ordinal representation could confuse the training algorithm. When marital status is encoded as increasing integers, a training algorithm would interpret marital status **other** (encoded as 3) as somehow being "three times more" than **married** (encoded as 1).
- Third, there is a relationship between many of the columns that is not made explicit in the structure of the data. Most notable are the **BILL_AMT_N**, **PAY_N**, and **PAY_AMT_N** columns - they are all related temporally (with each value of **N** representing a different calendar month). It was my hypothesis that some of the methods would pick up this relationship, while others would not - the decision tree, for example, can only split on one column at a time, so it would not be able to effectively pick up the relationship between these variables. The KNN, by contrast, would not struggle with this, because it would be able to find "corners" of the feature space where there were low payments and high outstanding balances. My hypothesis was that making these relationships explicit by replacing **BILL_AMT_N** and **PAY_AMT_N** with a new feature **DELTA_N** - representing their difference, which is the amount of the bill unpaid - would improve the performance of the decision tree and not impact the performance of the other algorithms.

Code Attribution and Methodology

The results presented below owe an enormous debt to the code provided by GitHub user cmaron - specifically the repository CS-7641-assignments. The assignment1 directory of that repository was used as the base code for this analysis, with minor modifications to allow it to run the modified datasets I used.

To produce this report, I worked with both the raw data from UCI and two different modifications to that dataset:

- I ran all 5 algorithms (SVM, ANN, KNN, DT, and Boosting) on the raw dataset, directly as provided from UCI.
- In addition, I ran all but the SVM (due to its significant training time costs) on a modified dataset that separated **EDUCATION** and **MARRIAGE** into their own indicator variable columns. This was done to test the hypothesis that their representation as ordered

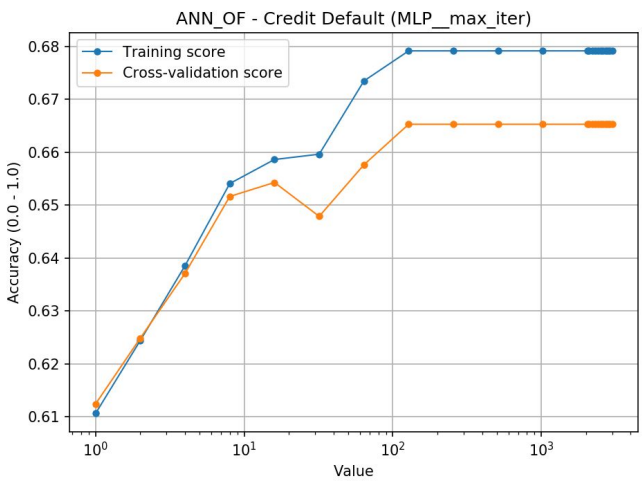
integers could cause unexpected results in some of the algorithms (as described in bullet #2 of the introduction).

- Lastly, I ran the KNN and DT on a further modified dataset that used the new feature **DELTA_N**, as described in the third bullet of the introduction.

All model performance was judged by its **F1 score** on a test set (chosen because the dataset is imbalanced in favor of negative results). 5-fold cross validation was also performed on all of the models and is reported as an additional metric in each overview section.

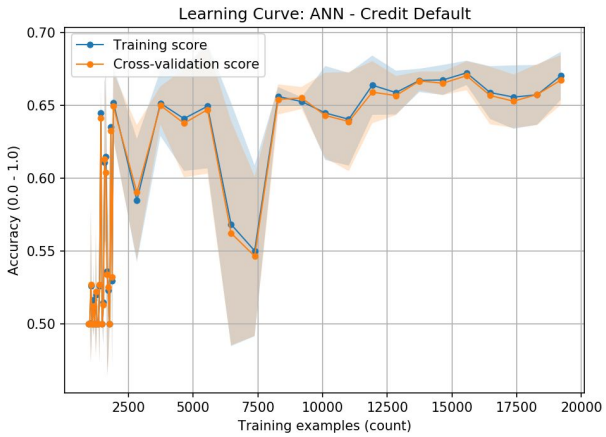
Overview - Artificial Neural Network

First an examination of the artificial neural network, as that was reported to be the best performing algorithm in the dataset page.



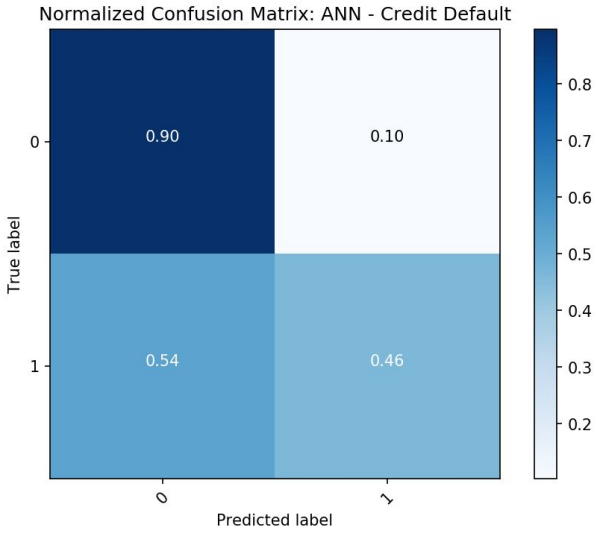
Many different networks were fit, over a range of network architectures and learning parameters. All were multilayer perceptrons with a binary classification output layer. Approximately 3400 neural networks were fit, grid searching over 1 to 3 hidden layers * 3 different layer node counts * 2 different activation functions (ReLU and sigmoid) * 9 different learning rates * 20 different values of alpha.

The plot to the left shows the accuracy of the best performing network (parameter details in the “Test Set Results” section) as a function of backpropagation iterations. It shows that there was considerable variance in the result of the fit before approximately 100 backpropagation iterations. After that point, however, all runs converged to the same value (approximately 68% accuracy on training and 66.5% on cross validation).



The plot on the left shows the learning curve for the all the ANNs fit (with the line being the average and the shaded region the standard deviation). Performance on both training and cross validation is highly variable at the low end of the training example size (below approximately 2500 examples). This makes sense, as small random samples have a higher probability of not being representative of the entire set. For this dataset that is particularly

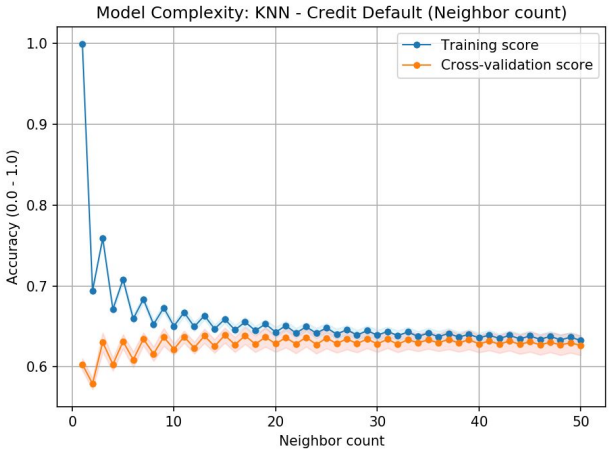
true, as there is an uneven distribution of positive and negative examples (approximately 3.5 / 1 ratio). As a result, small training samples can often contain very few positive examples, which can lead to overfitting if the sampled positive examples share a small set of similar features (which is more likely when the number of examples is small). The variance in the accuracy performance remains high until approximately 10000 examples is reached (which is 1/3 of the dataset). After that it levels out at approximately 66.5%.



Performance of the trained ANN can be further investigated using the normalized confusion matrix to the right (which shows performance on the training data). The ANNs worst performance was its false negative rate, which was 54% (as compared to a false positive rate of 10%). This may be due to the fact that the dataset is unbalanced in favor of negative results.

Overview - K Nearest Neighbors

Next we will examine the performance of KNN on the dataset. As a much simpler algorithm than ANN, its performance will provide a useful counterpoint to the performance of the more complicated ANN.

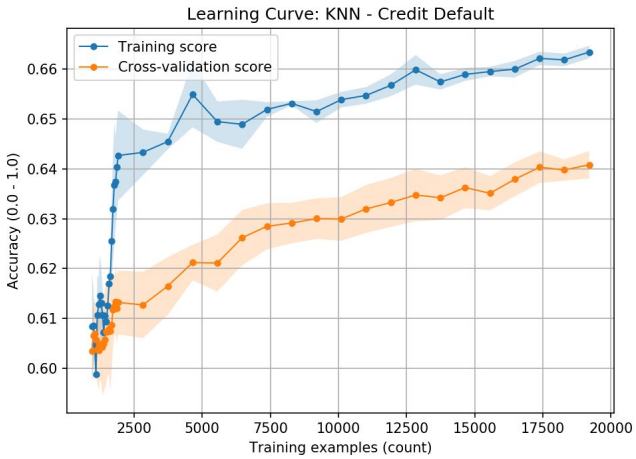


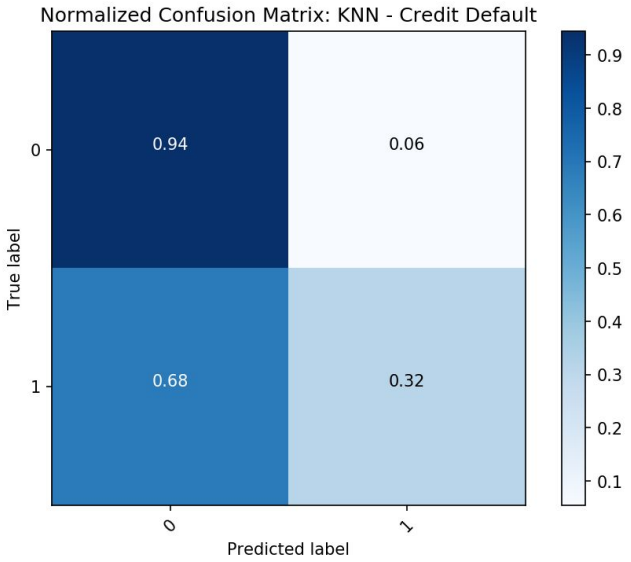
A grid search was performed over three distance metrics (manhattan, euclidian, and chebyshev) and 50 values for K (from 1 to 50). The best performing parameters were K = 3 with manhattan distance.

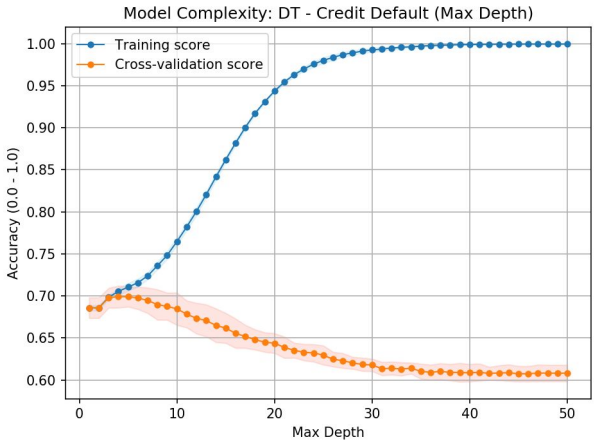
The plot on the left shows the accuracy of the KNN as a function of neighbor count.
This plot demonstrates the expected degenerate case where K = 1, meaning that the training set is essentially predicting itself. However, it does illustrate a couple interesting points:

- Values of K > 10 (and especially K > 20) have comparable performance on cross validation to training, and are nearly as accurate as the ANN.

The KNN was subject to the same failures of generalization as the ANN when the number of training examples was small. Performance continued to improve with sample size, which makes







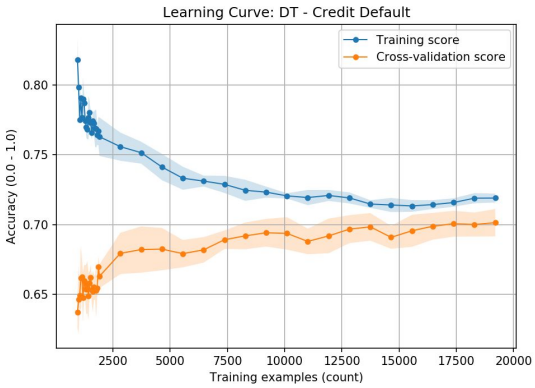
sense - as a KNN training set covers more of the feature space, it becomes less likely that it will struggle with outlier points (assuming, of course, that the training and test sets are drawn from identical distributions). This is illustrated in the learning curve plot to the left.

Lastly we look at the confusion matrix for the KNN training set (left). Here we see a reduction in false negatives but a significant increase in false positives (as compared to the ANN). The KNN has a true negative rate nearly double the ANN (1/10 vs. 1/20) - depending on the application, this could make the KNN the better choice (for example, if it were used for screening out clients with low risk of default before sending the rest to an analyst or other system).

Overview - Decision Tree

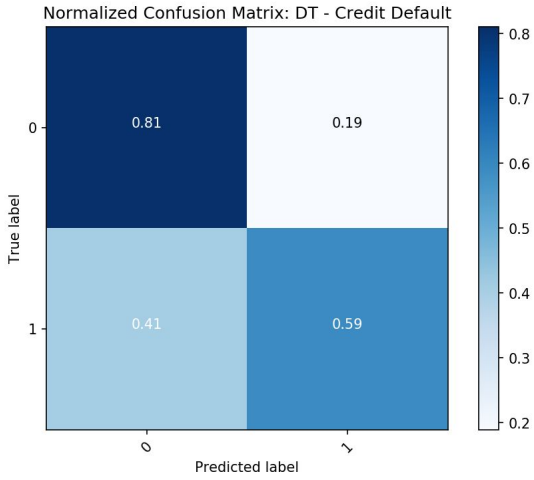
A decision tree approach was the next algorithm tested. A grid search was performed to find the best parameters - two criteria were used (gini impurity and information gain) as well as 50 different max depths. The plot on the left shows the obvious importance of using max depth as a pruning method - its effect on overfitting is clear. Without a max depth to prune the tree and keep it from overfitting, performance on cross-validation falls and prediction on the training set reaches 100% (the starkest example of overfitting of all the algorithms tested).

The training size learning curve for the DT (next page) shows very different response to sample size vs. the KNN and the ANN, particularly with respect to the training set. For small training set sizes, we observe significant overfitting that was not there for the ANN and KNN. This makes sense given how the DT works - for small training sizes and large max



depths, it is easier for the tree to overfit. This is the opposite case for the previous two algorithms, which improve both training score and cross validation score as training data size increases. This is because their accuracy on training is dependent on the data being a representative sample, whereas the DT benefits from small training sizes because it can overfit).

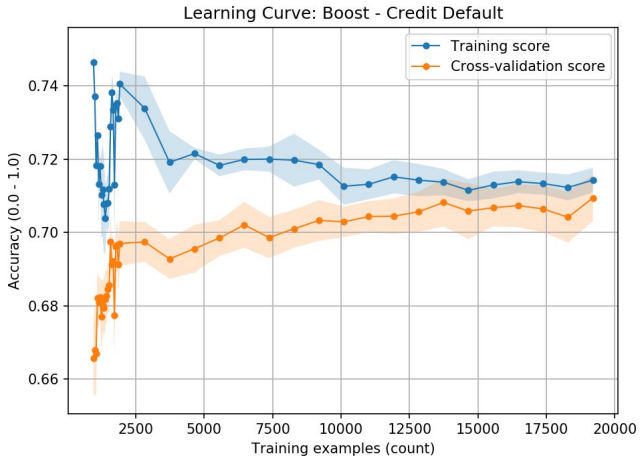
Lastly, the DT training set confusion matrix shows our best performance yet on the true positive case, but worst performance on the true negatives.



Overview - Boosting

Several different boosted versions of the DTs above were similarly fit using a grid search over tree max depth and number of boosted estimators. The best parameters found from that search were max depth = 4 and estimator count = 45.

Boosting increased F1 score by nearly 2 points vs. the single DT approach (more detail in the "Test Set Results" section). It was also the best performing of all the algorithms on the test data. This may be because there are a lot of positive cases in the dataset that look very much like negative ones. This is made clear by the KNN results, which have a very low true positive rate, indicating that most of the positive results are dispersed broadly among negative cases. Boosting is naturally an effective solution for a dataset like that, because it will de-emphasize those easy true positives and focus more on finding rules that can differentiate the difficult cases, which the KNN results indicate are the majority.



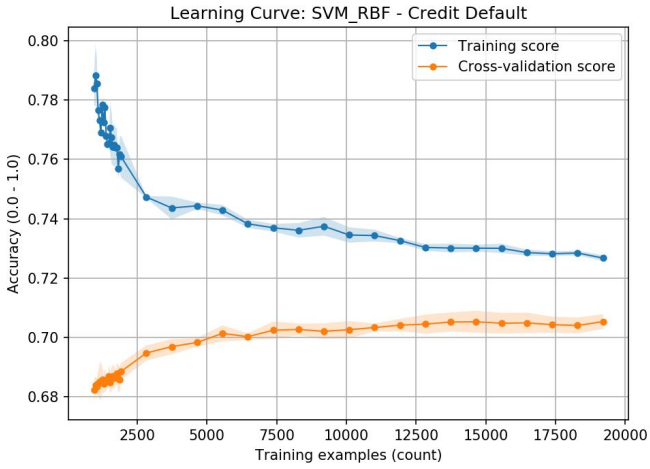
In the Boosting learning curve plot on the right, we see that much of the overperformance on the training data set that was observed in the single DT case is corrected by using boosting. This is expected and should be the case with any ensemble method (since ensembles reduce extreme overfitting by using a summary

statistic calculated over several different model results to provide a prediction). However, for training set sizes, it is still subject to the same poor generalizability of the single DT.

Overview - SVM (Linear and RBF)

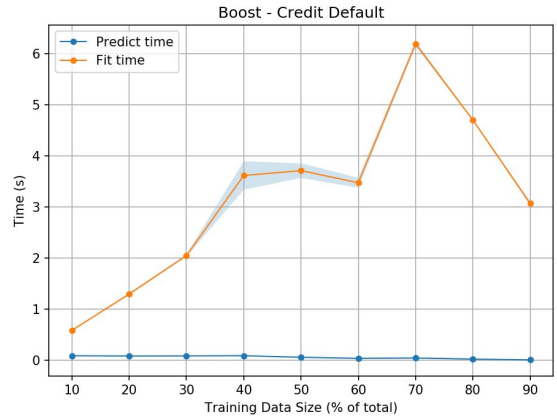
Two different SVMs were fit in this experiment - both a linear one (with a hinge loss function) and one using a radial basis function (RBF) as a kernel. A grid search over fit tolerance, C (a "penalty" parameter that increases the importance of accurate classifications vs. wide margins), and gamma / kernel coefficient (for the RBF) parameters was performed for both of these methods. The F1 score for the RBF kernel outperformed the linear kernel by approximately 0.03 (see "Test Set Results"), which is expected as the dataset clearly does not have a clean linear separation and the RBF gives the model an additional degree of freedom to fit the relationship.

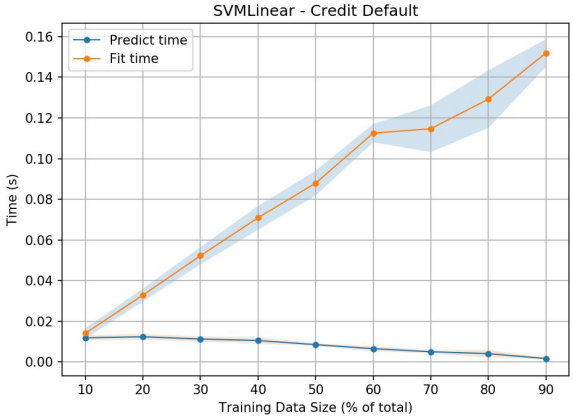
This is further supported by the difference in optimal C for the Linear and RBF fits - the optimal C for the linear was 0.001, while the optimal C for the RBF was 1.001. This indicates that the Linear SVM was forced to accept much smaller margins in order to correctly classify the data, while the RBF was able to take advantage of the additional dimension and degree of freedom (gamma) introduced by the kernel and use that to make the data more separable (and a wider margin more acceptable).

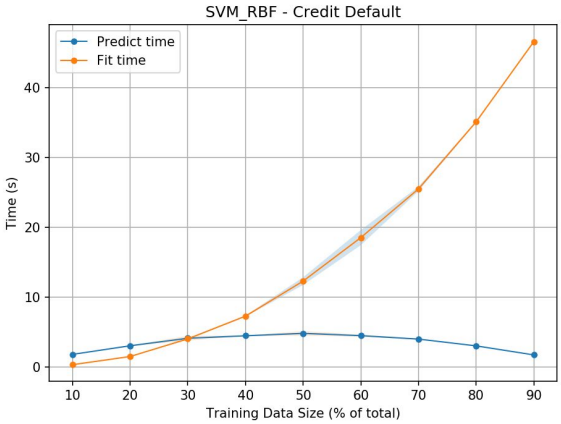


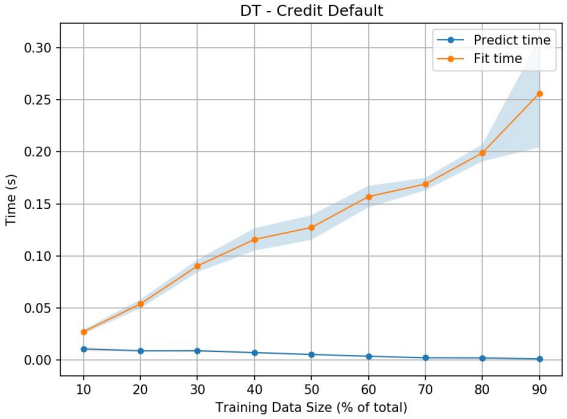
Learning curves for the SVM RBF show poor generalization for small training data sizes, as expected. This is for the same reason as the DT approach - small training samples are less likely to be representative of the entire general distribution, which naturally leads to overfitting.

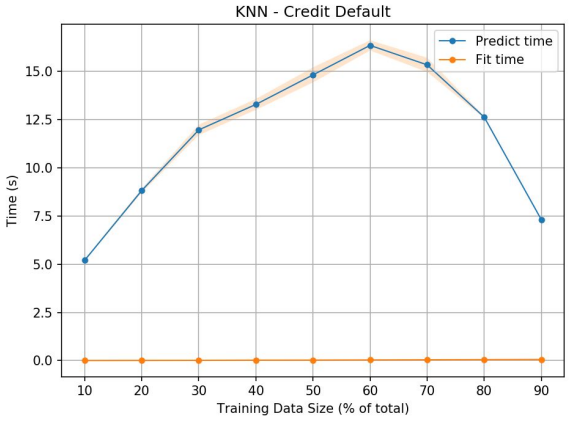
Training Times

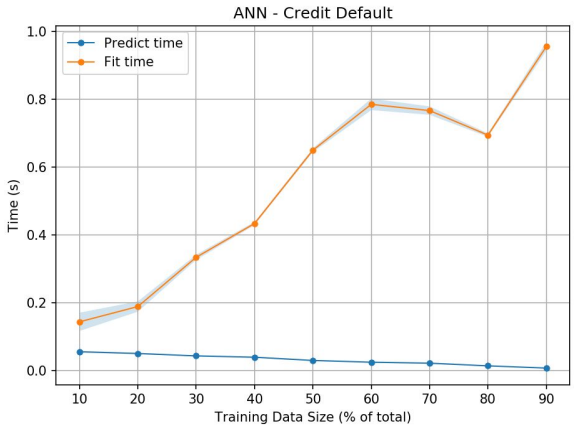












The plots above show the fit and predict times for each of the six algorithms tested (ANN, KNN, DT, Boosting, and two SVM variants - Linear and RBF) as a function of the training data size. All training times increase with training data size with the exception of the KNN, which has a constant training time of 0 (as expected).

Training times are under a second for all algorithms with the exception of boosting and SVM (RBF). The RBF example is particularly extreme, with training times in the 10s of seconds after training data size goes above 50%. This is expected, however, as the documentation for sklearn's implementation of SVM explains:

"The fit time complexity is more than quadratic with the number of samples which makes it hard to scale to dataset with more than a couple of 10000 samples."

Test Set Results

The results for each algorithm on the test data are shown in the table below. Each parameter set represents the best performing model of all the various model hyperparameters that were grid searched over. The F1 score for each variation on the dataset is shown("base" for the raw dataset, "one-hot" for categorical variables one-hot encoded, and "delta" for both one-hot and the difference between **BILL_AMT_N** and **PAY_AMT_N** as a single **DELTA_N** feature).

Algorithm	**F1 - Base**	**F1 - One Hot**	**F1 - Delta**	**Parameters**
Boosting	0.7186	NA	NA	MaxDepth = 4, Estimators = 45
DT	0.7007	0.6992	0.7117	Criteria = gini, MaxDepth = 4
SVM (RBF)	0.7067	NA	NA	C = 1.001, gamma = 0.0435, Tolerance = 0.06
ANN	0.6798	0.6597	NA	Activation =ReLU, Layers = (23, 23, 23), LearningRate = 0.128
SVM (Linear)	0.6707	NA	NA	C = 0.001, Tolerance = 0.03
KNN	0.6303	0.6211	0.6405	Metric = chebyshev, Neighbors = 7

Conclusions And Potential Future Experiments

In my testing, the ANN was significantly outperformed by Boosting, DT, and SVM (RBF). This runs counter to the argument made on the page that hosts the dataset, which said that only the

ANN was able to achieve acceptable performance. In future experiments I would want to try more complex neural network architectures to see if F1 scores above ~72 could be achieved. Given that the best performing ANN was not the one with the most nodes (my ANN grid search covered hidden layer architectures as high as (46, 46, 46)), I would think that more domain knowledge is required in order to prevent simply overfitting to the training data (as those more complex versions of the grid search did).

All algorithms also were much better at detecting true negatives than detecting true positives. This could be a function of the fact that the dataset was so imbalanced - as previously mentioned, there were approximately 3 and a half times as many negative examples as positive ones. The sharp fall in true positives for the KNN results (discussed in detail in “Overview - K Nearest Neighbors”) also demonstrate that most of the positive examples are well dispersed throughout the feature space, which makes accurate detection of true positives more difficult.

This does not necessarily mean that the fitted models are not useful, however; they all had quite good true negative detection, which could allow them to be used in scenarios where filtering true negatives has value (assigning cases to be monitored by an analyst or fed into another algorithm, for example).

Curiously, the results were slightly *worse* for all algorithms on the one-hot encoded dataset, which was unexpected given my previously stated hypothesis. My guess for why this happened is that I was thinking incorrectly about how education is represented - even though it is a categorical variable, its ordering is actually meaningful (in that *grad school* > *college* > *high school*). My changes to the dataset removed that relationship, which may have resulted in the algorithms not being able to take advantage of all the information in that feature.

The decision tree on the delta feature version of the data performed as well as the boosted trees on the raw data, which supports the hypothesis that the decision tree performance would improve if those features were merged (since DTs can only consider one at a time). It also produced more parsimonious models on the merged data (with a max depth of 4 being the best performing version). As a result, in future work I would like to try the boosted version on that modified data set and see if it can increase performance even further.

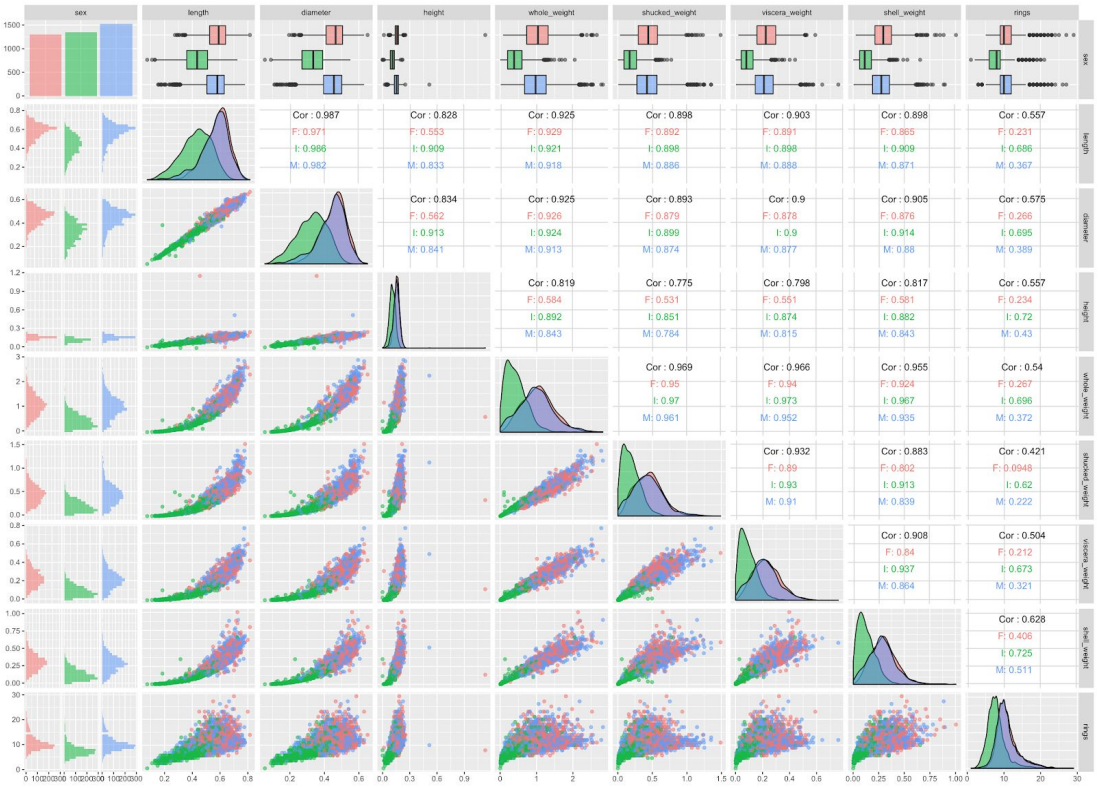
I would also try normalizing the the features before passing them into the KNN. This is a standard practice for clustering / classification tasks that make use of a distance metric, which ensures that all features are using the same units (standard deviations). This prevents features that have units that are naturally higher (e.g., one feature that ranges from 0 to 10 grams and another from 1000 to 10000 grams) from being overweighted, since minimizing them provides a much larger distance gain than minimizing the features that are on smaller orders of magnitude. This would be especially interesting given that the best performing KNN version (on the “delta” dataset) used the chebyshev distance, which further overweights reductions to features with higher orders of magnitude (by virtue of it taking the max of all distances).

Experiment 2: Introduction and Description of Dataset

The next experiment uses the abalone dataset from UCI, which is much smaller in terms of both number of observations and feature count than the previous dataset. The dataset concerns abalones (a type of shellfish), providing 8 features (sex, length, diameter, height, whole weight, shucked weight, viscera weight, and shell weight) for predicting the abalone's number of rings.

I modified this data slightly before training my algorithms - I one hot encoded the sex feature and also transformed the number of rings into a binary variable indicating whether or not the abalone had less than 10 rings (encoded as 0 for false and 1 for true). The choice of splitting on 10 or less was made because it split the dataset as evenly as possible (2081 false cases and 2096 true cases). The goal here was to see if there would be better performance could be achieved with a more balanced dataset (as compared to the credit default dataset).

This dataset is particularly interesting because very clear linear relationships exist between the length, diameter, and height, as well as between the four weight features. This means that there is very little additional information added by each. Further, the relationships between the first set (length, diameter and weight) and the second set (weights) were more like exponential curves, so I was interested to see if that relationship could help lead to better models. Lastly, none of the individual features were particularly good predictors of the rings variable (with the exception of sex=infant), so I thought it would be a good challenge for the algorithms to tease out. The relationships between the variables are shown in more detail in the pairs plot below:



Because this dataset is simpler (and I covered the details of the algorithms while presenting the previous results) I will present an abbreviated analysis of the second problem.

Test Set Results

The results for each algorithm on the test data are shown in the table below. Each parameter set represents the best performing model of all the various model hyperparameters that were grid searched over.

Algorithm	F1 Score	Parameters
ANN	0.797	ReLU, Alpha = 0.0316, LayerSizes = (20, 20, 20), LearningRate = 0.064
Boosting	0.7944	MaxDepth = 6, Estimators = 80
SVM (RBF)	0.7896	C = 2.001, Tolerance = 0.06, Gamma = 0.1
DT	0.7849	Criterion = gini, MaxDepth = 5
SVM (Linear)	0.7787	C = 0.251, Tolerance = 0.03
KNN	0.7574	Metric = Euclidean, Neighbors = 40

Conclusions

The ANN and Boosting algorithms work best for the abalone dataset, though the range of F1 scores is half that of what was observed on the credit default dataset. My hypothesis for why they work the best is that they are the methods that are best able to take advantage of the nonlinear relationship between the length/diameter/height variables and the four weight measurements. The density plots (colored by sex) along the trace of the pairs plot shown in the introduction to this experiment show that each individual variable in each of the two linearly correlated sets of features shares nearly identical distributions for each sex, with only the infant variable being significantly different. Absent some information gained from that relationship, the feature with by far the most information to add to the prediction of ring count is sex (specifically sex=infant, colored in green in the pairs plot).

The ANN and Boosting approaches both performed best on the training set at the lower ends of their model complexity grid search space, indicating that more complex approaches failed to generalize from training to testing. This could be a potential area where additional domain knowledge could help inform the modeling approaches.