SUPERVISED LEARNING

David Gong
Department of Computer Science
Georgia Institute of Technology, davidcgong@gatech.edu

February 11, 2019

ABSTRACT

This report seeks to analyze the performances, accuracy, and characteristics of namely five supervised learning techniques - Decision Trees, Neural Networks, Boosting, Support Vector Machines, and k-Nearest Neighbors. All implementations were done in Python using the scikit-learn library.

1 Datasets

For the application of the supervised learning techniques, two datasets were used - the Spambase Data Set and the Poker Hand Data Set from the UC Irvine Machine Learning Repository. Both data sets were split into 70% training cases and 30% testing cases.

The Spambase Data Set consists of a collection of spam-emails from the UC Irvine postmaster and individuals who had filed spam for e-mails, and a collection of non-spam emails from work and personal e-mails. With this data set, the goal is to train a model to generate a general purpose spam filter. There were 4,601 instances of data available.

For the attributes, the first 48 attributes consist of continuous real [0, 100] attributes for word frequency, which measure the percentage of words in the e-mail message that match a certain word. Then, we have 6 additional attributes for the matching percentage of characters in the e-mail, the average length of uninterrupted sequences of capital letters, the length of the longest uninterrupted sequences of capital letterse, and the sum of the length of uninterrupted sequences of capital letters. The last column represents binary classification, which serves to denote whether or not an e-mail was considered to be spam with a 1 or not spam with a 0.

The motivation for choosing this specific data set ties into real-life experience in which certain aspects of spam have managed to penetrate through a spam filter and into an inbox, even on Gmail. As a result, the need for creating a more effective spam filter would be greatly beneficial for improving the overall productivity of the electronic mailing system.

With regards to the Poker Hand Data Set, it consists of a collection of hands which consist of five playing cards drawn from a standard deck of 52 cards. The goal is to train a model that would be able to predict poker hands based on suit and rank attributes for each card in a hand, adding up to 10 predictive attributes for the data. The last attribute involves an ordinal number between 0-9, which signifies 0: Nothing in hand, 1: One pair, ..., 8: Straight flush, 9: Royal flush. There are 25,008 instances of data.

Motivation for picking this specific data set is based off of personal experience with playing poker for money. This data set also provides a unique level of difficulty, as even with many instances of data, it becomes difficult to give an exact classification for the hand based off of identifying patterns between suit and rank. For instance, the probability of a royal flush is at 0.000154% odds, a two pair is at 4.7539% odds, etc. As a result, it becomes doubtful that the model trained with this data set can give an accurate classification of hands.

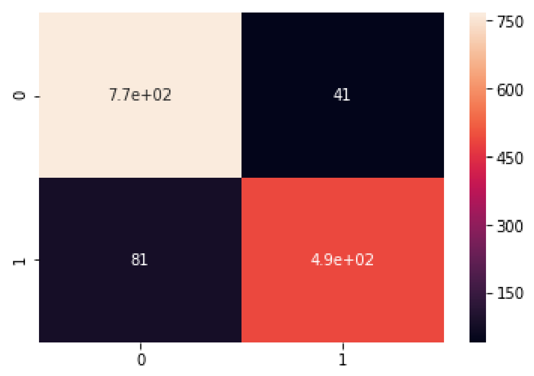


Figure 1: Confusion Matrix (Decision Tree, Spambase)

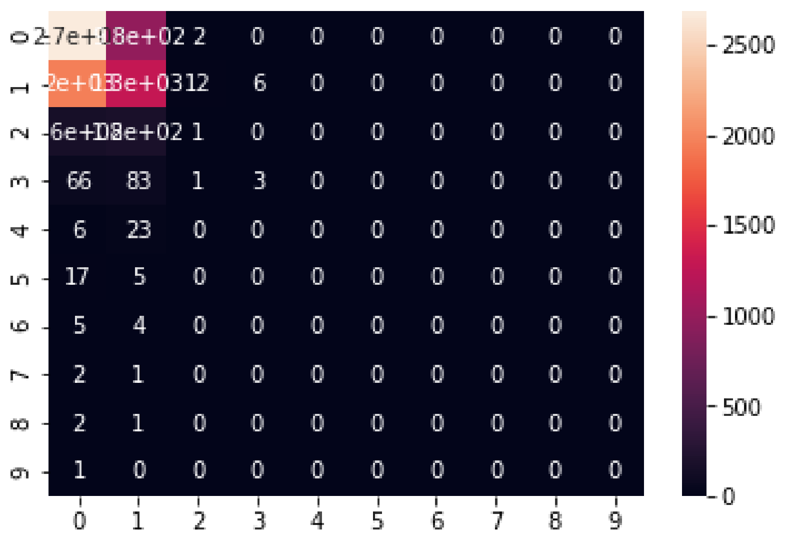


Figure 2: Confusion Matrix (Decision Tree, Poker Hand

2 Decision Tree

As will be the case with the other algorithms, we will first split the data into training and testing data (70% and 30% respectively). For this specific instance, we will be using RandomizedSearchCV from scikit-learn for randomized searching and k-fold in order to find the best parameters along with training data.

The training and testing of the data sets using the Decision Tree method is extremely quick, with a training time of 0.43s and 0.61s for Spambase and Poker Hand data sets respectively.

2.1 Spambase

For this data set, it can be expected that we should be able to generate a somewhat accurate prediction, even with the moderate amount of data and somewhat large amount of attributes due to the end result being a binary classification between spam or not spam. As a result, our confusion matrix is 2x2 and the risk of failing to predict a classification is significantly reduced. See Figure 1.

Through the use of RandomizedSearchCV and 3-Fold, the pruning of the decision tree has also significantly reduced overfitting and reduced the overall complexity of our classifying model. The maximum depth of the tree was 12, with the minimum sample leaves amount being 1. As a result, a reasonable accuracy score of 91.88% was obtained. As seen in Figure 1, the model was successful in classifying 1,268 instances of e-mail (762 as spam and 506 as non-spam).

2.2 Poker Hand

With there being numerous possibilities and probabilities, when it comes to training a model to classify a poker hand, the problem becomes much more complicated. Instead of a binary classification, the model must be able to classify a certain poker hand correctly out of ten possiblities. The confusion matrix for this specific data set would be 10x10. See Figure 2.

A pattern of interest here is shown with regards to the tendency for the model to properly classify poker hands that are no pairs. With the chance of drawing a no pair being close to 50% and a one pair being at around 42%, most of the classification results and data seem to be concentrated around those two classifications. Even then, simply due to the added probability implications of the problem, the classifier was not able to predict as accurately as the Spambase model classifier.

The pruning also reduced the overall decision tree to a minimum sample of leaves to 2 with a max depth of 8. The end result is the classifier achieving a training score of 54.90% and accuracy score of 53.05% when it came to predicting poker hands.

3 Neural Networks

For neural networks on scikit-learn, we will be using the MLPClassifier, also known as the Multi-layer Perceptron classifier. This allows for the optimization of the log-loss function with the use of LBFGS or SGD. For the activation function for the hidden layers (default of 100 layers), three functions were used.

Logistic Sigmoid Function: $f(x) = \frac{1}{1+exp(-x)}$
Hyperbolic Tan Function: $f(x) = tanh(x)$
Rectified Linear Unit Function: $f(x) = max(0, x)$

These three functions generated significantly different results than expected in terms of performance, accuracy through overfitting, and cross-validation score over a certain number of epochs.

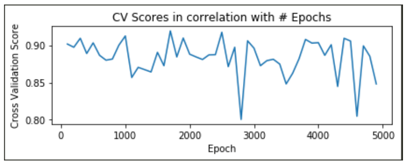
3.1 Spambase

See Table 1.

When it came to training time, the log function took the longest with 38.62s and the RELU function took the slowest with 20.22s. When it comes to convergence, the RELU typically out-performs the Hyperbolic Tan Function on average by a factor of 6, which in turn also outperformed the Logistic Sigmoid Function for this case. The training scores were similar across the board, but a specific metric of interest is the accuracy score for RELU, which is significantly lower than its counterparts even with similar training scores.

For some additional background, RELU is also used in most deep learning models as of today for its simplicity and performance, as can be observed in the training time. The question arises with regards to why in this instance the accuracy score was lower than even that of a decision tree with basic pruning.

An additional major upside of RELU is that it also avoids the vanishing gradient problem. However, in turn, some gradients can "become fragile" and not be activated during training. This results in a weight update which makes RELU not activate on data points. This is most likely the root cause of the low accuracy score for this specific case.



CV Scores in Correlation with Epochs (RELU) for Spambase

Table 1: NN Metrics (Spambase)

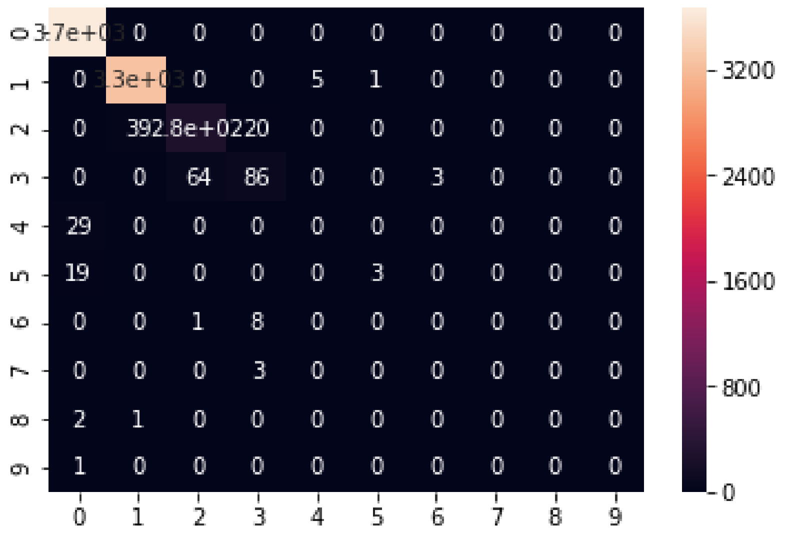
Activation Function	Training Time	Training Score	Accuracy Score
Logistic Sigmoid Function (log)	38.62s	92.30%	92.25%
Hyperbolic Tan Function (tanh)	26.97s	93.23%	93.48%
Rectified Linear Unit Function (RELU)	20.22s	91.96%	87.83%

Table 2: NN Metrics (Poker Hand)

Activation Function	Training Time	Training Score	Accuracy Score
Logistic Sigmoid Function (log)	262.95s	54.96%	72.94%
Hyperbolic Tan Function (tanh)	290.96s	63.21%	97.96%
Rectified Linear Unit Function (RELU)	222.03s	56.81%	59.91%

A glance at the cross-validation scores in correlation with the number of epochs also shows a worrying trend - a deep dip at around the 2800 epoch mark on the x-axis for the cross-validation score. RELU may not be the best activation function to use here and using neural networks for this specific data set instance either treads on the side of overkill or inaccuracy.

3.2 Poker Hand



Confusion Matrix (Hyperbolic Tan Function)

See Table 2.

Performance-wise, RELU once again showed its prowess. However, given a significantly more challenging problem for the classifier, the difference between results for each activation function was immediately shown. The metric that is of significant interest in this case is the accuracy score for the Hyperbolic Tan Function, with an accuracy of near 98%. The confusion matrix above shows that the classifier predicted correctly along the diagonal for almost every data instance, a surprising revelation to be questioned.

Two theories immediately came to mind - possibility of overfitting or a lucky drawing of a training set. Overfitting was ruled out due to the training score being low, as overfitting is vice versa where training score is high and accuracy score becomes low. When re-tested, an accuracy of 97% was obtained, although convergence took significantly longer at 435.62s.

In conclusion, for our present theory we can actually conclude that for the specific instance of the Poker

Hand data set, the Multi-layer Perceptron (100-layered) classifier with an activation function of Hyperbolic Tan Function produces extraordinarily accurate results.

4 Boosting

For boosting, Adaptive Boosting (AdaBoost) was used through the use of the AdaBoostClassifier from scikit-learn. The AdaBoost classifier fits a classifer on the original data set and fits more copies of the classifier on the data set where incorrectly classified data instances have their weights adjusted so future classifiers can become more accurate when it comes to predicting difficult cases.

AdaBoost was used in conjunction with the Decision Trees with Pruning model, which tries to find the best parameters through the use of randomized search.

Of immediate notice is the amount of time it took to train the data with Adaptive Boosting. Even with using the same Decision Tree model, the Adaptive Boosting model ended up taking 75.31s and 140.32s as opposed to 0.43s and 0.61s respectively for the Spambase and Poker Hand data sets.

4.1 Spambase

See Table 3.

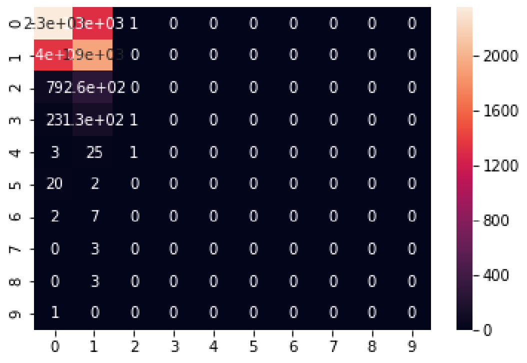
The Spambase data set classifier in particular saw a 2.11% increase, raising the accuracy score in classification from 91.88% to 93.99%.

The most likely explanation for this phenomenon is that the adjusted weights for incorrectly classified data instances helped for the prediction of difficult cases, but there were still outliers in the data that could still not have been caught.

Regardless, this is a noticeable improvement over the decision trees counterpart and is duly noted.

4.2 Poker Hand

The Poker Hand data set classifier saw a 3.85% increase over its Spambase counterpart, raising the accuracy score in classification from 53.05% to 56.90%.



Confusion Matrix (Adaptive Boosting)

Similarly to the Spambase data set, the Adaptive Boosting ultimately increased the classifier's ability to accurately predict hands in general. As can be seen in the confusion matrix above, the classification model saw a good improvement in predicting 0 or 1 (no pair or one pair), but not much anywhere else due to decreasing probabilities and statistics around the board.

Table 3: Boosting Metrics (Spambase + Poker Hand)

Dataset	Training Time	Training Score	Accuracy Score
Spambase	75.31s	94.16%	93.99%
Poker Hand	140.32s	56.25%	56.90%

5 Support Vector Machines

For Support Vector Machines, the C-Support Vector Classification (SVC) was used in order to aid our implementation. Two kernels were used: linear and RBF.

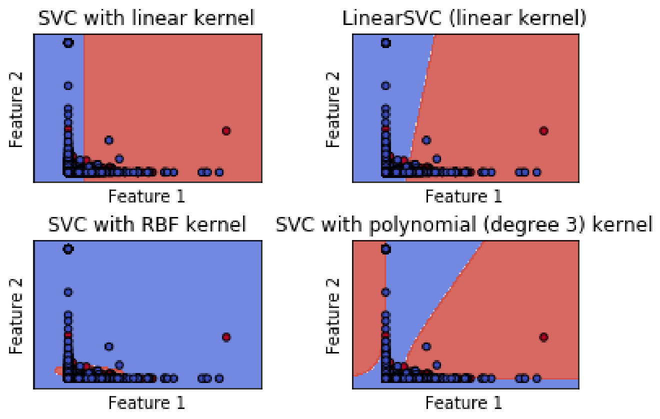
Before running the SVM, one important note to make is that one of our datasets, Poker Hand, has over 25,000 instances of data. The fit time complexity of SVC is more than quadratic, which makes it harder to scale to the data set if there are over 20,000 samples of data.

5.1 Spambase

See Table 4.

Upon viewing of the SVM metrics for Spambase, the difference between performance and accuracy on the SVM kernel becomes evident. RBF, also known as radial basis function, outperforms the Linear kernel in terms of accuracy by 13.27%, but also takes ten times longer to run. Though the RBF kernel performed up to expectations and slightly better, the linear kernel falters when compared to other algorithms.

A likely theory of how these results came into existence could be due to the gaussian distribution of the data for Spambase. This allowed the radial basis function to better fit the concentrated data within a radial area. Linear, on the other hand, falters in this specific scenario and when there are numerous classes and attributes to consider. This becomes evident once visualized on the graph below.



5.2 Poker Hand

See Table 5.

Similarly to the results for Spambase, running the SVM on the Poker Hand dataset also took longer but was more accurate for RBF and vice versa for Linear kernels. However, RBF outperforms the Linear kernel in terms of accuracy by 33.35%, much more than in Spambase's SVM.

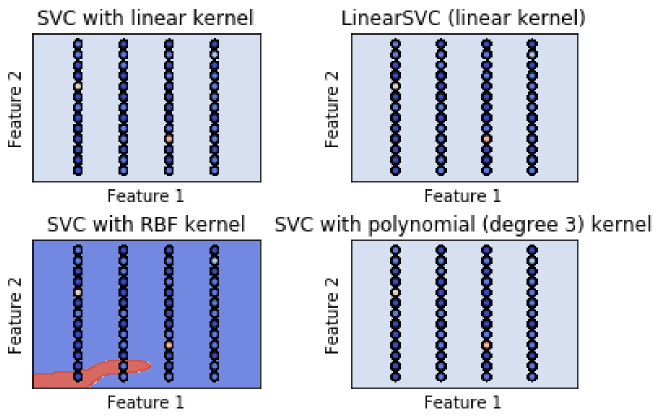
The same theory exists, that a gaussian distribution has allowed the RBF to outperform the Linear kernel. Notice the graph below.

Table 4: SVM Metrics (Spambase)

Kernel	Training Time	Training Score	Accuracy Score
RBF	3.32s	94.88%	94.86%
Linear	0.33s	81.49%	81.59%

Table 5: SVM Metrics (Poker Hand)

Kernel	Training Time	Training Score	Accuracy Score
RBF	73.91s	79.67%	79.05%
Linear	21.07s	46.23%	45.70%



The SVC with the RBF kernel was able to capture much of the data towards the bottom of the graph, which consists of the highest concentration of data (0's and 1's) in the data set. Ultimately, this allowed the classifier to be trained to be better at classifying those specific data instances in general, resulting in a steep increase in accuracy score.

Linear, on the other hand, rarely catches the trend in the data and should be excluded even with fast performance.

6 k-Nearest Neighbors

For kNN, the KNeighborsClassifier from scikit-learn was used, and a range of k values were tested, ranging from 1 to 14. Similarly to decision trees, the model trained extremely quickly albeit somewhat slower by a factor of 2-6.

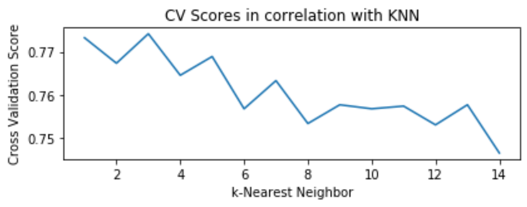
6.1 Spambase

See Table 6.

A surprising revelation is that the accuracy score for the Spambase data set is significantly lower than any other supervised learning technique. The graph shown here also displays how as the number of neighbors went up, the cross-validation score actually decreased. Though upon further analysis, it can be concluded that there is one main reason for this case.

Table 6: kNN Metrics (Spambase + Poker Hand)

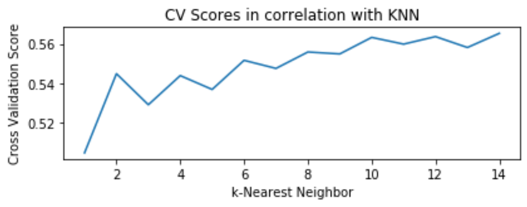
Dataset	Training Time	Training Score	Accuracy Score
Spambase	0.89s	77.42%	78.70%
Poker Hand	8.96s	56.53%	56.88%



High dimensionality in general can greatly lower the accuracy of kNN, and class distributions are skewed. In this case, with the amount of attributes given for the each data instance, paired with only 4,600 instances of data, kNN is not the optimal supervised learning technique for this specific case. As a result, kNN did not perform as well as expected.

6.2 Poker Hand

On the other hand, the accuracy score for the Poker Hand data set was increased through the use of kNN. The cross-validation scores increase as the number of neighbors increase, as shown in the graph here.



The main reason why kNN worked well for this specific case was because the data was often concentrated in certain areas. Due to the overall probability statistics of poker, most of the data were concentrated on 0s and 1s which were no pairs and one pairs over 2s to 9s. As a result, when it came to better being able to generalize the data and predict, there was a visible increase in the classifier's ability to predict properly.

7 Conclusion & Recap

See Table 7 & 8.

As can be seen, there is a no glove fits all solution to different data sets due to the differing preferences in attributes, performance, and accuracy.

For the Spambase data set, our model performed best when it incorporated boosting with decision trees, though the training time took longer. Neural networks with the Hyperbolic Tan Function comes in at second place, albeit with a better performance in terms of training time.

For the Poker Hand data set, our model performed best when using the neural networks approach with Hyperbolic Tan Function for the activation function for the Multi-layered Perceptron Classifier. Though, the training time was costly.

This wraps up the analysis of five supervised learning techniques on Spambase and Poker Hand data sets, which has ultimately revealed some interesting behaviors of the algorithms when implemented.

Table 7: Results of Supervised Learning Techniques (Spambase Dataset)

Technique	Training Time	Training Score	Accuracy Score
Decision Trees	0.43s	92.08%	91.88%
Neural Networks - tanh	26.97s	93.23%	93.48%
Boosting (Winner)	75.31s	94.16%	93.99%
SVM	8.96s	56.53%	56.88%
kNN	0.89s	77.42%	78.70%

Table 8: Results of Supervised Learning Techniques (Spambase Dataset)

Technique	Training Time	Training Score	Accuracy Score
Decision Trees	0.61s	54.90%	53.05%
Neural Networks - tanh (Winner)	290.96s	63.21%	97.96%
Boosting	140.32s	56.25%	56.90%
SVM	8.96s	56.53%	56.88%
kNN	8.96s	56.53%	56.88%

References

[1] *Scikit-learn: Machine Learning in Python*, Pedregosa et al., Journal of Machine Learning Research, JMLR 12, pp. 2825–2830, 2011.