Neil Barooah

Supervised Learning

Introduction

In this paper, I aim to analyze the strengths and weaknesses of various Machine Learning algorithms, namely decision trees, neural networks, boosting, support vector machines and k-nearest neighbors. I used scikit-learn to test these algorithms with appropriate tweaking to necessary parameters. I have chosen 2 datasets from the UCI repository for my analysis: Wisconsin Diagnostic Breast Cancer (WDBC) and Balance Scale.

Wisconsin Diagnostic Breast Cancer (WDBC)

This dataset extracts features from digitized image of a fine needle aspirate of a breast mass to describe the characteristics of the cell nuclei present in the image. I will be using this dataset to predict whether the diagnosis (label) is malignant or benign given the mean, standard error and worst (largest) value for the attributes: radius, texture, perimeter, area, smoothness, compactness, concavity, concave points, symmetry and fractal dimension for each cell nuclei. There are 569 instances, of which 357 are benign and 212 are malignant. All features are real-valued features.

The WDBC is an interesting and challenging binary classification problem as it contains a relatively high number of attributes for the limited number of instances available. The real-valued attributes make it a challenging problem as the range for each attribute widely varies and requires good feature engineering for successful analysis. It is also interesting because each instance contains statistical values like the mean, standard error and largest value for a group of cell nuclei; thus, provides greater insight into the nature of the dataset and adding meaning to the models built. As is the case with any dataset with a limited number of instances, we risk overfitting the training data. That further makes this an interesting challenge to tackle.

Balance Scale

The balance scale data set was generated to model psychological experimental results. Each instance is said to have classified the balance scale tip to the right, left or remain balanced. The features in this data set are left weight, left distance, right weight and right distance. All of these features have integer values ranging from 1 to 5. The target attribute here is the class, which represents the state of the scale: left (L), right (R) or balanced (B). The dataset represents 46.08% instances classified as L, 7.84% classified as B and 46.08% classified as R. Therefore, this a 3-category classification problem. In this dataset, there are 625 instances. I have taken an 80-20 split to obtain my training and test dataset.

The balance scale dataset is an interesting 3-way classification problem. What makes this dataset interesting is that it contrasts with the WDBC dataset. While WDBC has a large number of features (30), the balance scale only has 4 features. It is also interesting that while the WDBC dataset as continuous real-valued attributes, the Balance Scale dataset have integer features. Through analysis of this experiment, I will be able to understand the differences between analyzing problems with both discrete/continuous valued features and large/small number of features.

Neil Barooah

Decision Trees

Decision tree is a popular supervised learning algorithm that uses a tree to predict a target value based on observations of features. I pruned the decision tree by setting a maximum depth of the tree from 1 to 100. Restricting the maximum depth essentially prevents the tree from growing deeper than the specified depth to prevent overfitting the data. In order to determine the optimal maximum depth, I used a 7-fold cross validation strategy using Grid Search CV from scikit-learn. The results obtained are as shown in the Fig 1, 2.

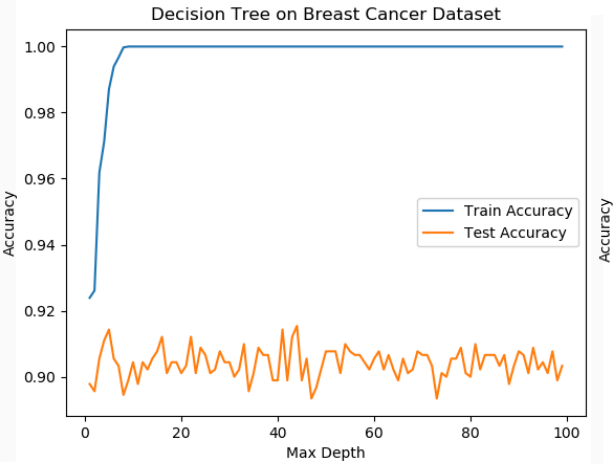

Fig 1: Accuracy vs Max Depth (WDBC)

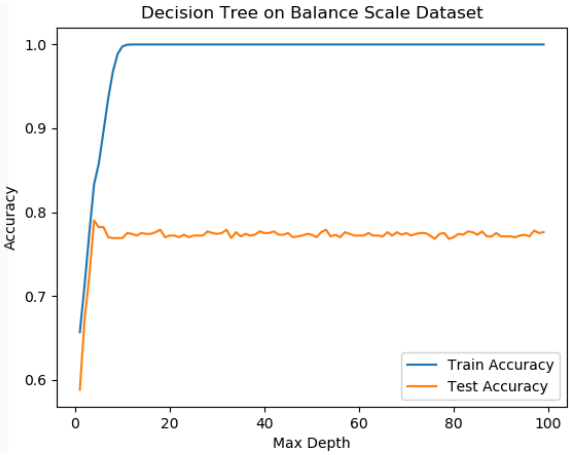

Fig 2: Accuracy vs Max Depth (Balance Scale)

As shown in Fig 1, it was found that the algorithm performed the best on the WDBC dataset by setting maximum depth to 44 with an accuracy of 91.9% while the lowest accuracy obtained was with maximum depth set to 45 with accuracy of 89.5%. It is important to note that the training accuracy sharply rises to 100% beyond a depth of 8. This shows signs of overfitting when the depth is larger than 8. At the same time, the test accuracy starts behaving erratically beyond a maximum depth of 4. Interestingly, the concave points had an overwhelmingly large feature importance: 0.0357 [concave points (mean)], 0.757 [concave points (standard error)] and 0.0055 [concave points (worst)]. As such, several attributes such as the area, perimeter and fractal dimension had close to 0 feature importance. Two other significant attributes were the radius (standard error) and texture (mean). Therefore, this is saying that it is usually enough to diagnose breast cancer by looking at the concave points, radius and texture of the nucleus.

On the other hand, Fig 2 shows that the performance of the decision tree on the balance scale dataset sharply improves from a maximum depth of 1 to 4, at which point it peaks and doesn't improve any further. Similar to the WDBC dataset, the training accuracy sharply rises as the maximum depth increases initially but reaches 100% accuracy at maximum depth of 10. At this point, the model starts to overtrain and shows signs of overfitting. Unlike the WDBC dataset, Balance Scale has far fewer features all of which seem to have a relatively equal importance according to the results of the decision tree. The left-weight, left-distance, right-weight and right-distance have feature importance values 0.226, 0.272, 0.241 and 0.260, respectively. Therefore, all of these features are equally important in predicting the position of the balance scale.

According to the cross validation done the WDBC dataset, we infer 44 to be the optimal maximum depth of the tree. However, due to the gross overfitting observed in Fig 1 beyond a maximum depth of 4, I have analyzed the model for both a maximum curve of 44 and 3 as shown in Fig 3 a, b. In case of a maximum depth of 44, the performance peaks at 92.5% test accuracy at 280 instances (75% of training dataset) while a depth of 3 peaks with the same accuracy at 120 instances (33% of training set). Beyond these peaks, the model doesn't show any improvements in test accuracy with more data. As mentioned earlier, the 100% training accuracy in both settings is indicative of overfitting of the training set.

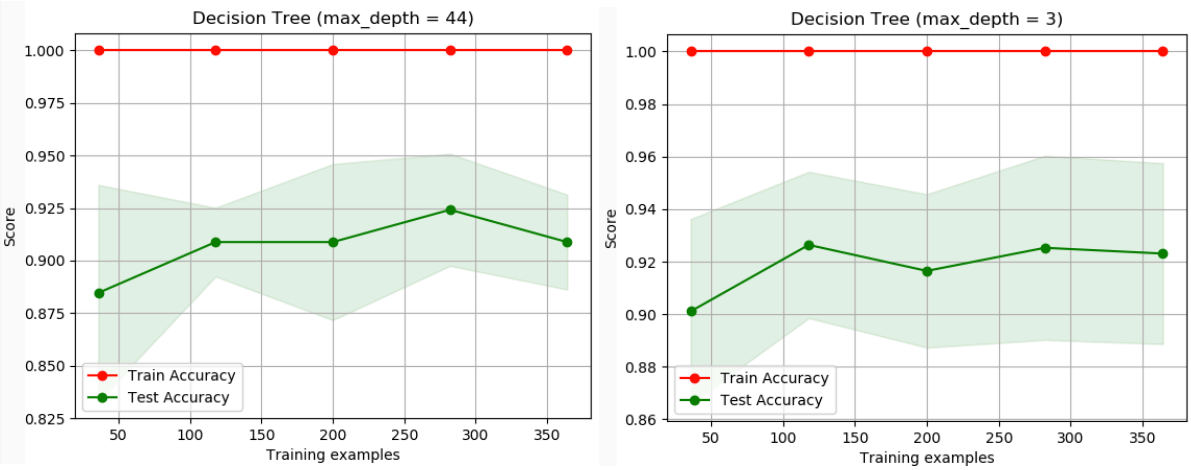

Fig 3 a, b: Learning Curve for Decision Tree on WDBC with max_depth = 4, max_depth = 3

On the other, a maximum depth of 4 is the optimal depth for the Balance Scale dataset as concluded by the cross validation done from Fig 2. With a maximum depth of 4 on the decision tree, we obtain the learning curve as shown in Fig 4. It is clear that the algorithm benefits from a larger training set. Unlike the WDBC dataset, the learning curve obtained shows that the model could potentially benefit from training on more data as the performance improves with more training examples.

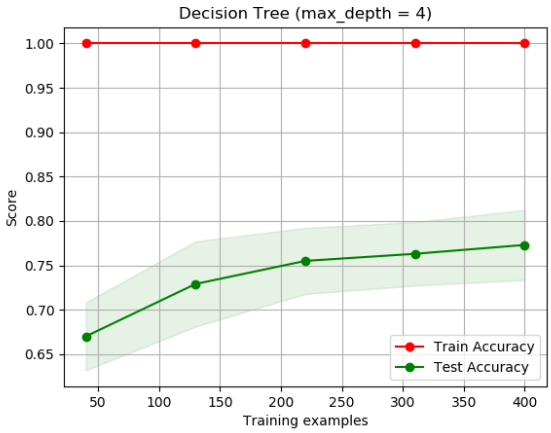

Fig 4: Learning Curve for Decision Tree on Balance Scale with maximum depth = 4

Boosting

Boosting is a useful supervised learning algorithm that converts weak learners to strong learners. Specifically, I used Adaboost for this purpose, which subsequently tweaks weak learners in favor of instances misclassified by previous classifiers. One reason why I chose Adaboost is because it is resilient to overfitting. I used a Decision Tree classifier as the estimator for Adaboost as it requires relatively less processing

time. I experimented with varying the number of estimators (weak learners) from 1 to 100. I conducted a 7-fold cross validation to determine the optimal number of estimators that produces a high accuracy on the test set for further analysis. The results are as shown in Fig 6, 7.

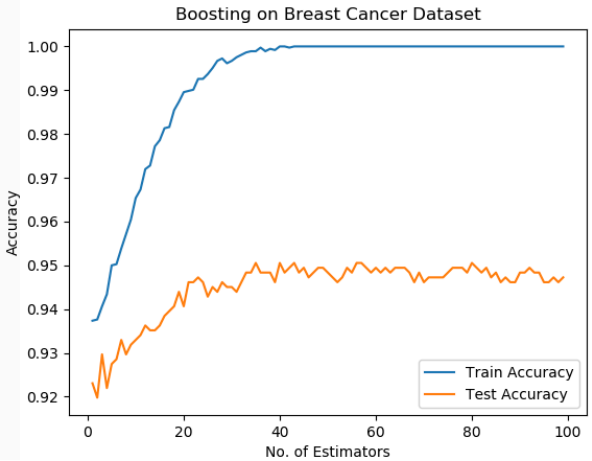

Fig 5: Accuracy vs Num Estimators (WDBC)

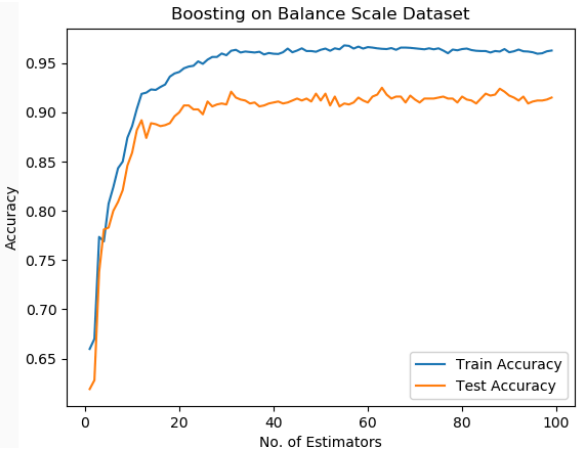

Fig 6: Accuracy vs Num Estimators (Balance Scale)

As shown in Fig 5, it was found that Adaboost performed the best on the WDBC dataset with 35 estimators at 95.05% test accuracy while it performed the worst with 2 estimators yielding 91.98% test accuracy. At 40 estimators and beyond, the model starts to overtrain as the training accuracy nears 100%. In contrast, Fig 6 shows that the optimal number of estimators for the Balance Scale dataset was found to be 63 yielding a 90% test accuracy and performed the worst with 1 estimator yielding 62% test accuracy. In both datasets, we observe that increasing the number of estimators beyond the optimal point doesn't lead to any improvements. In case of the Balance Scale dataset, improvements are minimal beyond 25 estimators. However, unlike in the case of the WDBC dataset, the training accuracy doesn't go above 95%.

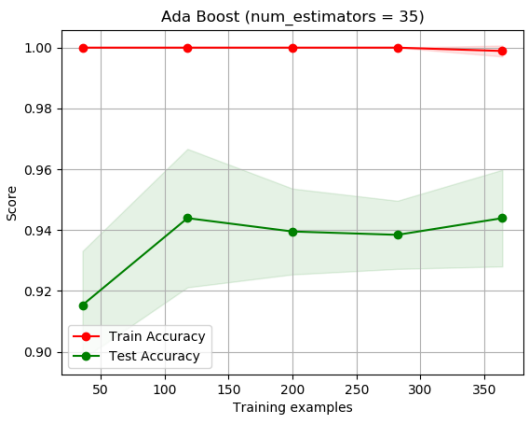

Fig 7: Adaboost on WDBC with 35 estimators

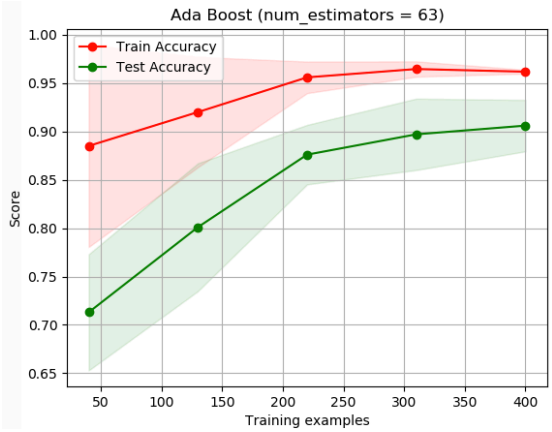

Fig 8: Adaboost on Balance Scale with 63 estimators

From the learning curve of Adaboost on WDBC dataset as shown on Fig 7, we see signs that the model has overtrained on the training set; thus leading to training accuracies of near 100%. This is likely due to the small number of samples in the WDBC dataset coupled with the numerous features at play. At the same time, we see that the model performs reasonably well on the test set with an accuracy of 94% at its peak on 110 instances (30% of training set). Although the model initially benefits from a growing dataset, the accuracy doesn't increase beyond this point. However, we observe a generally high test accuracy of above 90% on the WDBC dataset is due to the binary

classification nature of the problem; Adaboost generally tends to perform better on such problems.

On the other hand, Fig 8 shows that learning curve of the Adaboost algorithm on the Balance Scale dataset. We observe that both the train and test accuracy improve as the training set size increases. The test accuracy increases from about 72% to 90%. Clearly, the model benefits from a large set of data. The improvement in accuracy with larger training set shows that the model is able to generalize better with more data. Unlike the WDBC dataset, the model is not overfitting as the training accuracy is not at 100%.

k-Nearest Neighbors (KNN)

KNN is a type of non-generalizing learning algorithm. Here, we classify instances by majority vote of its neighbors, with the instance being classified to the class most common amongst its k neighbors. To determine the best k, I ran the algorithm on both datasets with k ranging from 1 to 20.

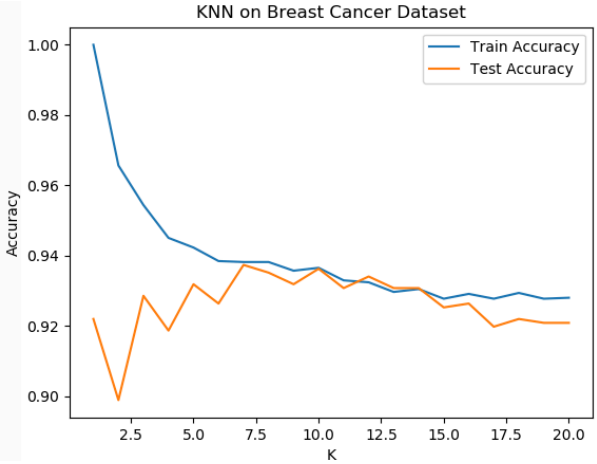

Fig 9: Accuracy vs k in KNN on WDBC

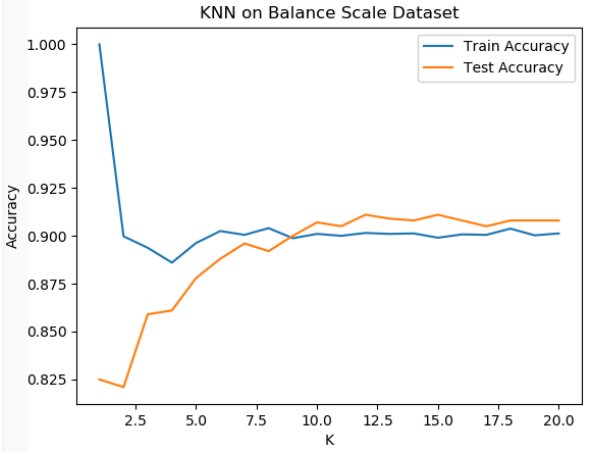

Fig 10: Accuracy vs k in KNN on Balance Scale

I conducted a 7-fold cross validation on both datasets to obtain the obtain value of k that leads to a high test accuracy. On the WDBC dataset, it was found that k = 7 led to the highest test accuracy at 93.74% while the worst case was k=2 with 89.89% as shown in Fig 9. As the performance peaks at k = 7, the test accuracy starts to decline beyond that and so does the training accuracy. Fig 10 shows that k = 12 performed the best on the Balance Scale dataset with a test accuracy of about 90% and performs worst when k = 2 with accuracy 82%. Interestingly, the test accuracy is higher than the training accuracy when k is set to 8 or higher, meaning that the model has correctly fit the data. Not surprisingly, in both cases, KNN achieves very high training accuracy with a low value of k. This is true because a small k value represents a complex model in which the granularity is too fine leading to overfitting. As we increase k, we reduce variance but increase bias and risk under-fitting. Therefore, I tested both datasets with a large range of k values to obtain sensible k values for the model.

Fig 11 that KNN performs reasonably well on the WDBC dataset with optimal k = 7. The training accuracy and test accuracy converge at 94% and stops improving after once it trains on about 200 instances (55% of training set size) meaning that it is unlikely to benefit from a larger training set. In contrast, Fig 12 shows the learning curve for KNN on the Balance Scale dataset with optimal k = 12. The test accuracy rises from about 77.5% to 90% as we increase the size of the training set. The test accuracy peaks at

about 215 (60% of training set size) instances and slightly suffers from this point onwards. However, the rate at which it's test accuracy rises is faster than that on the WDBC dataset. The larger k for Balance Scale indicates that the dataset benefits from more smoothing. However, peak accuracy at 90%, the accuracy going down is likely due to under-fitting that may have been caused by the large number of neighbors in the model.

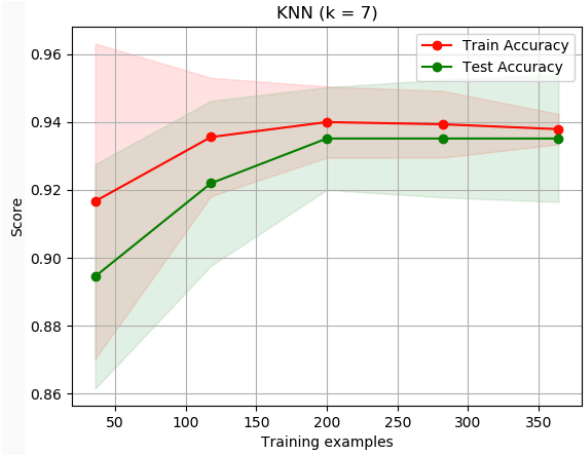

Fig 11: KNN on WDBC with k = 7

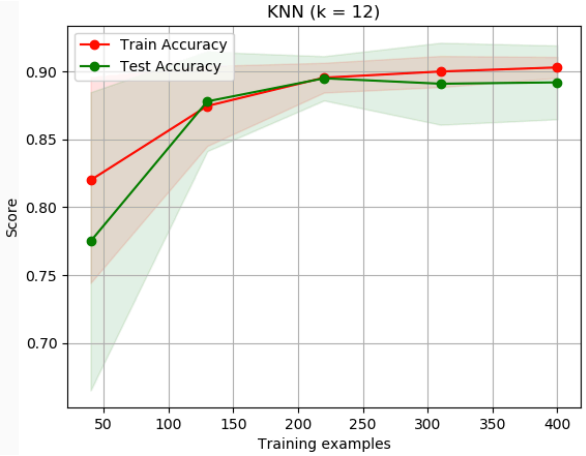

Fig 12: KNN on Balance Scale with k = 10

Support Vector Machine (SVM)

In addition to linear classification problems, SVM performs also well on non-linear classification problems by mapping features into high dimensional feature spaces. This is a particularly valuable algorithm for this experiment as SVM as we are primarily working on classification problems. I explored the performance of Linear, Sigmoid and Radial Basis Function (RBF) kernels for both datasets. Figure 13 represents the performance of these kernels using SVM on the WDBC dataset. While the RBF and linear kernels yield nearly 100% accuracy on the training set, the linear kernel performs significantly better than the other 2 kernels with a test accuracy of 95%. One that stands out in this case is the poor performance of the sigmoid kernel both on the training set and test set.

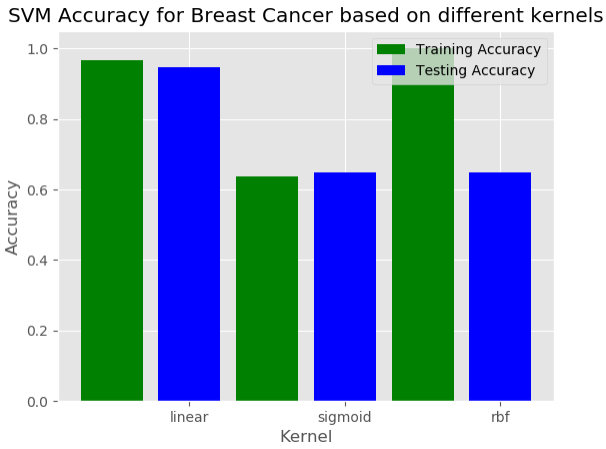

Fig 13: Accuracy of linear, sigmoid and RBF kernels on WDBC

As shown in the learning curve of the SVM on WDBC dataset using the optimal kernel (linear) based on Fig 14, we see observe that the model performs well with an increasing training set as the test accuracy rises from 92% on 10% of the training set to about 95% on the full training set. The training accuracy, on the other hand, decreases meaning that the model overfits the training set when the training set is small. As the set grows in size, the model is able to generalize better.

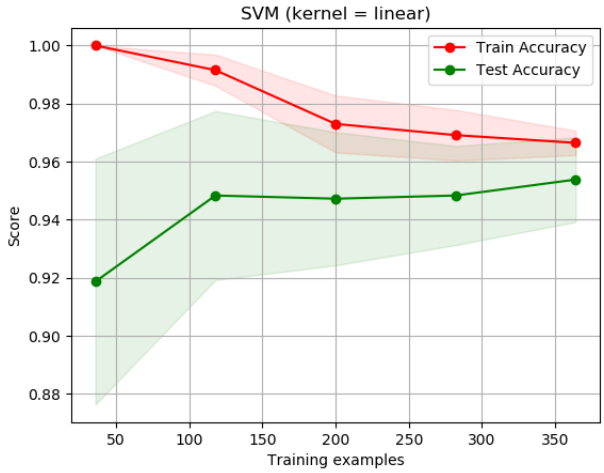

Fig 14: Learning curve of SVM (linear) on WDBC

As with in case of WDBC, the SVM algorithm performs the best using a linear kernel with a test accuracy of 92% as shown in Fig 15. SVM also performs reasonably well using the RBF kernel. However, unlike the WDBC dataset, RBF also performs well on the test set with an accuracy of 90%. Given that the linear kernel trains a lot faster than RBF in terms of wall clock time, we will choose a linear kernel to further analyze the SVM model. The sigmoid kernel, on the other hand, performs poorly on both the training set and test set with accuracies as low as 40%. The reasonably good performance of a linear kernel on both the datasets go on to reveal that the data is linearly separable. This is further reaffirmed by the poor performance using the sigmoid kernel on both datasets, as the sigmoid kernel generally performs well on data that is not linearly separable.

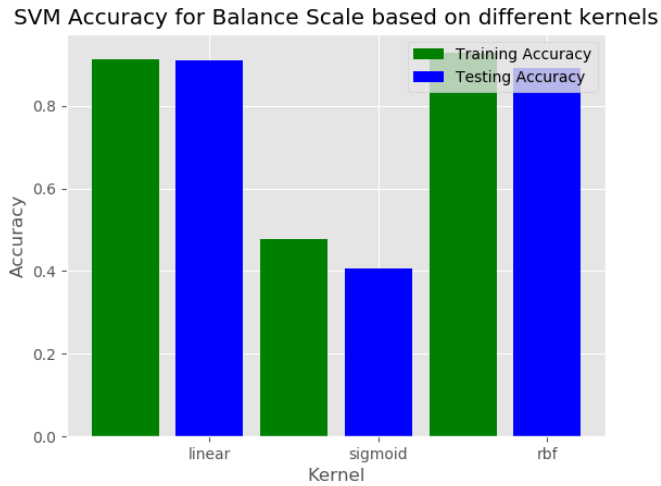

Fig 15: Accuracy of linear, sigmoid and RBF kernels on Balance Scale

As shown in the learning curve of the SVM on Balance Scale dataset using the optimal kernel (linear) based on Fig 16, we see observe that the model performs well with an increasing training set as the test accuracy rises from 82.5% on 10% of the training set

to about 90% on the full training set. While the training accuracy initially dips when the training set size is increased to 120 instances, it then becomes slight more accurately beyond this point. Unlike its performance on the WDBC dataset, the test accuracy benefits more from a larger training set size.

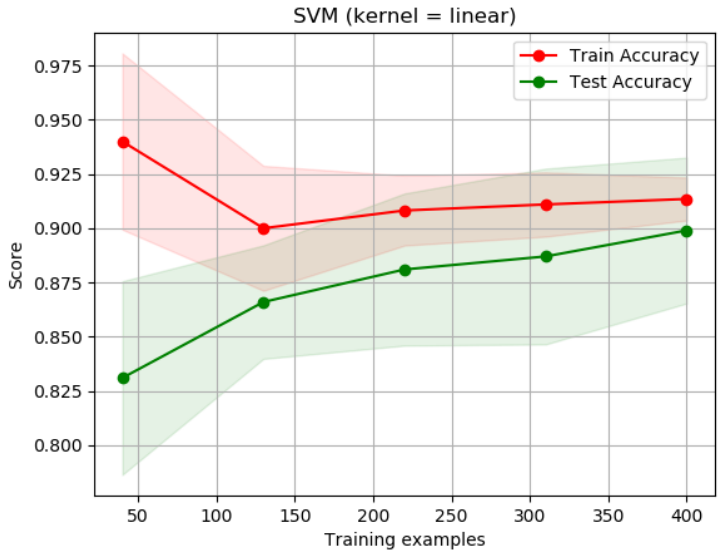

Fig 16: Learning Curve of SVM (linear) on Balance Scale

Neural Network

To analyze the performance of neural networks on my datasets, I used a multi-layer perceptron classifier. This model optimizes the log-loss function using the Adam optimizer, which is a type of stochastic gradient descent optimization technique. In Adam optimizer, we use both the gradients and the second moments of the gradient for optimization. The activation function used is the rectified linear unit function. In order to understand the performance of neural networks, I tuned the number of hidden layers on the network based on the 2 datasets obtain the optimal number of numbers. Each hidden layer adds a dimensionality in the dataset, thus allowing us to solve more complex problems that are not linearly separable. For both datasets, I chose to do 100 iterations over the entire dataset and used minibatches of 64 instances for training. I observed that the neural network performed much better when using a minibatch approach over the entire dataset. This is due to the nature of stochastic gradient descent optimization. By using a minibatch approach, we update the gradients more often during training; thus; introducing noise. This helps escape from saddle points or local minima and helps us arrive at a relatively speedy convergence.

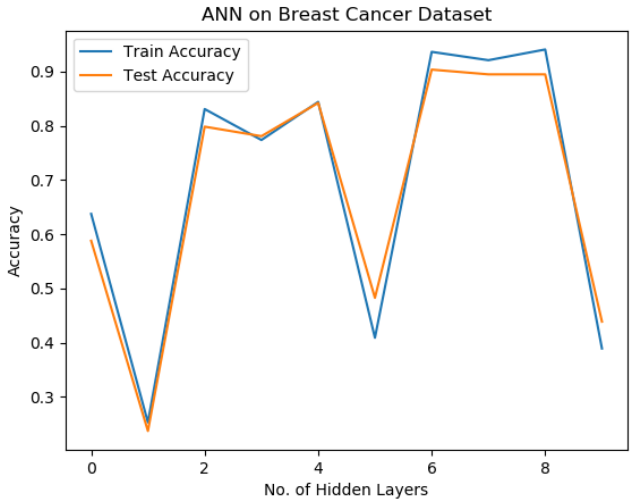

Fig 17: Accuracy vs. Number of Hidden Layers on WDBC

As shown in Fig 17, the optimal number of hidden layers is 6 with 16 neurons in each layer, yielding 90% test accuracy. It is important, however, to realize that we face the risk of overfitting the training set. In order to prevent overfitting, I have used L2 regularlization in the neural network, which reduces limits the magnitude of model weights by adding a penalty for weights to the error function. In scikit-learn, the default L2 regularization is set to 0.0001 but can be tuned according to the behavior of the dataset. In Fig 18, I am showing the comparison in performance of the neural network on the WDBC dataset when the L2 regularization term is set to 0.0001 (left) and 0.001 (right). We observe that increasing the L2 regularization value to 0.001 for the WDBC dataset solved the problem of having sharp dips in accuracy like when the regularization value is set to 0.0001. Adding a bigger penalty to the weights is a powerful tool. This is especially useful in this experiment as both datasets contain a relatively small number of instances, which makes it a suspect for overfitting.

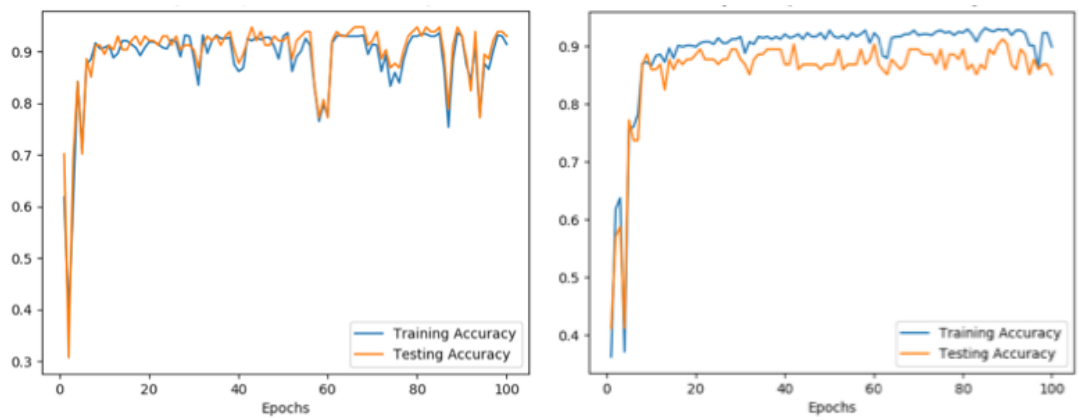

Fig 18: Accuracy vs No. of Epochs when L2 regularization set to 0.0001 (left) and 0.001(right)

As shown in Fig 19, the test accuracy using 6 hidden layers on the WDBC dataset is close to 90% in less than 15 iterations (epochs). The test accuracy sharply increases until 15 epochs and fluctuates between 85%-90% in subsequent iterations.

Fig 19: Accuracy vs Number of Epochs using ANN on WDBC with 2 hidden Layers

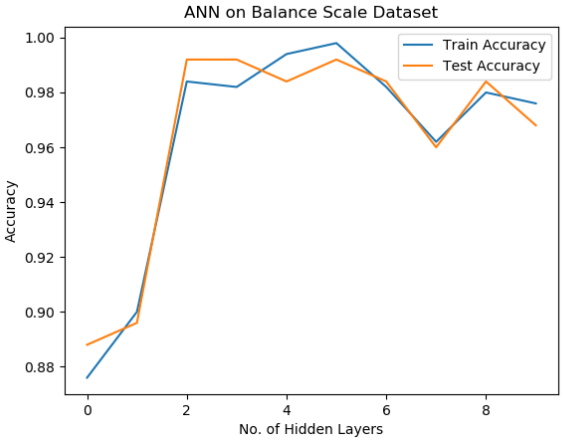

Fig 19: Accuracy vs Number of Hidden Layers on Balance Scale dataset

Fig 20 shows that the optimal number of hidden layers for the Balance Scale dataset is 2, achieving about 95% test accuracy. Beyond this point, the accuracy suffers. The learning curve for the ANN on the Balance Scale dataset is shown below in Fig 20. Unlike in the case of WDBC, the test accuracy doesn't have much noise in it. This goes on to show that there is less overfitting on this dataset. Furthermore, the test accuracy benefits from the larger number of iterations and reaches over 90% accuracy at the 100th iteration. It is also evident that the training accuracy is very close to the test accuracy.

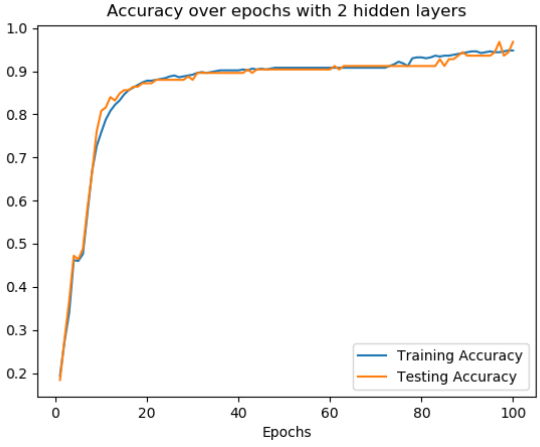

Fig 20: Accuracy vs Number of Epochs using ANN on Balance Scale with 2 hidden Layers

Neil Barooah

Summary

Table 1 summarizes the performance across all the algorithms analyzed in this experiment. From the experiments conducted, I have had the chance to observe the strengths of different supervised learning algorithms. In case of the WDBC dataset, both the SVM (linear kernel) and KNN performed well on the test set with 93.86% and 93.85% accuracy on the test set. SVM (linear kernel) also performed well on the Balance Scale dataset with 92% accuracy on the test set. However, the neural network performed even better on the Balance Scale dataset with 96% accuracy on the test set. The relatively good performance of neural networks is a surprising revelation as the number of instances in the Balance Scale dataset was quite small (625 instances). However, the cross validation score obtained using the neural network on the WDBC was extremely low at 36.92%. The reason for this is the relatively low number of instances along with the high number of features that the dataset had.

	Wisconsin Diagnostic Breast Cancer (WDBC)			**Balance Scale**		
Algorithm	Train Accuracy	CV Accuracy	Test Accuracy	Train Accuracy	CV Accuracy	Test Accuracy
Boosting (Adaboost)	99.78%	94.95%	92.11%	96.6%	91.37%	90.4%
Decision Tree	100%	92.32%	90.31%	100%	76.22%	86.4%
Neural Netowrk	92.01%	36.92%	88.59%	99.8%	95.8%	96%
KNN	94.06%	93.84%	93.85%	90.2%	88.99%	84.8%
Support Vector Machine	96.26%	95.59%	93.86%	91.6%	90.59%	92%

Table 1: Performance of algorithms analyzed

For the boosting algorithm, I chose to vary the number of estimators using a cross validation technique to obtain the optimal number. Similarly, for the decision tree algorithm, I used cross validation to determine the maximum depth of the tree as a pruning mechanism. For both datasets on decision trees, the training accuracy was converging to 100% very quickly. This is evident of overfitting in the model. In hindsight, I would have limited my experiment to a smaller range of estimators and maximum depth for both of these algorithms. This would have then helped considering large values for both parameters that would have led to overfitting. Adaboost for balance scale, however, did not have as much of an overfitting issue.

The cross validation to obtain the optimal k for the KNN algorithm was a success. It led to relatively good test accuracies, with the learning curve showing that there was little to no overfitting as seen by the training set accuracy. In terms of the SVM, both datasets performed well with a linear kernel. An added advantage of using a linear kernel was the advantage of quick training times.

Table 2 summarizes the wall clock time for all algorithms analyzed in this experiment. Not surprisingly, boosting took the longest amount of time for both datasets. The

second slowest was the support vector machine for WDBC and neural networks for the balance scale. The fasted algorithm was the KNN, which took less than a second for both datasets. Surprisingly, the SVM performed faster than expected. This is because we chose a linear kernel based on the results of the cross validation.

	Wisconsin Diagnostic Breast Cancer (WDBC)	**Balance Scale**
Algorithm	Training Time (sec)	Training Time
Boosting (Adaboost)	90.95	235.71
Decision Tree	2.62	1.72
KNN	1.15	0.89
Neural Network	4.64	9.86
Support Vector Machine	19.32	2.04

Table 2: Wall clock time of algorithms analyzed

All things considered, I believe that the SVM was a good algorithm for this experiment. This is because of the relatively high cross validation and testing accuracies across both datasets.