Project 1 Supervised Learning

Name: Luping Yang

GATech ID: lyang38

***Abstract*— It is important to understand a machine learning algorithm, such as why the algorithm works, how it behaves under a variety of circumstances, when it behaves well and when it behaves bad. As such, I implement some simple machine learning algorithms, and to compare their performance. These algorithms are: Decision trees (DTS), Artificial Neural networks (ANN), Boosting (BST), Support Vector Machines (SVM), k-nearest neighbors (KNN). These algorithms are applied to one simulated dataset (Planar dataset) and a real-life dataset (web phishing dataset). It is found that the models perform differently under different datasets.**

Keywords—Decision trees, Neural networks, Boosting, Support Vector Machines, k-nearest neighbors.

I. INTRODUCTION TO DATASETS

A. *Planar Dataset*

The planar dataset is a simulated dataset. The advantage of using simulated dataset is that I can construct the true function that we want to estimate. In this way, we know the answer that machine learning algorithms trying to estimate and we can expect how the algorithms should behave and see whether the estimated models meet our expectation.

The simulated data is called Planar data, which I got acquainted with when I took an on-line course from Andrew Ng on Coursera. The Planar dataset can be simulated using the function provided at Kaggle.

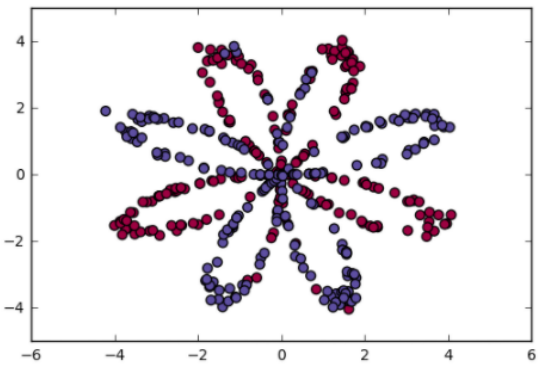

Fig. 1. Planar data visualization.

In Figure 1, the Planar dataset is visualized. The dataset consists of two features called, x1 and x2. This dataset has two classes which are marked by red and blue. From this plot, it is observed that the dataset has a complex pattern: it cannot be separated by a simple linear or curved boundary, especially in the center region, where a significant number of data points overlaps. This proposes a challenge to machine learning algorithm and it would be very interesting to see how the five machine learning algorithms can model this data set.

B. *Phishing Websits Dataset*

The Phishing Websites dataset is downloaded from OpenML PhishingWebsites. This dataset consists of 45 features and 11055 samples. Each sample is labeled 1 and -1, indicating if the website is a phishing website or not, with 1 being phishing website and -1 being not.

For the Phishing dataset, all the features and the label are categorical. Most of the features are binary values {-1, 1} and some features are {-1, 0, 1}. For the features with two values, one-hot encoding is used to convert these features to {0, 1}. For the features with three values, one-hot encoding is used to convert the features to {0, 1}. Thus, all features are {0, 1}.

II. METHODOLOGY

A. *Model Estimation Procedure*

When estimating a machine learning model, it is a methodological mistake to learn the parameters of a prediction function and test it on the same dataset. This is because a model could "remember" the labels of data samples that it has just seen. In this way, the estimator can achieve a perfect score on the dataset. But when we apply the estimator to predict the samples that the estimator has not seen, the estimator can easily fail. This situation is called overfitting.

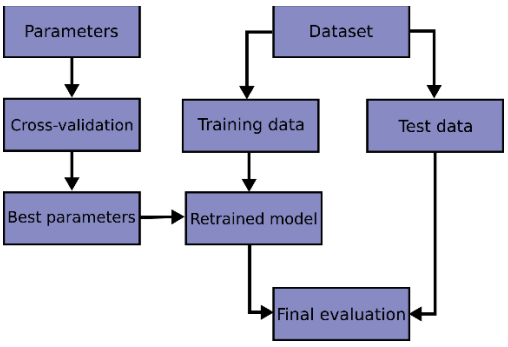

Fig. 2. Common procdures for supervised learning. (Source scikit-learn).

Thus, it is important to follow the common procedures established by machine learning professionals, when conducting machine learning experiments. The procedures are depicted in Figure 2. First, we need to split the dataset into training and testing dataset. The training dataset is what we used to estimate the parameters of our models. The testing dataset is used to see

how our estimated models predict the samples that they have not seen.

B. *Hyper Parameter Tuning*

For typical machine learning algorithms, we are faced with the task of hyperparameter estimation. When estimating the hyper-parameters of estimators, to avoid the leak of test set information to model estimation, we hold out another set of data from training dataset. This dataset is called "validation set".

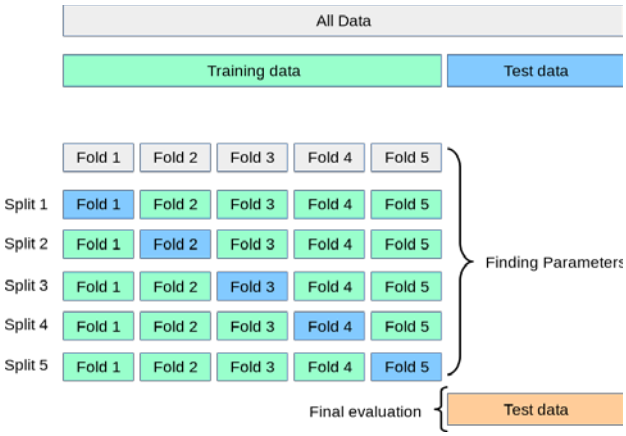

Fig. 3. Illustration of 5-fold cross validation (Source scikit-learn).

For hyperparameter tuning, cross-validation is used. As shown in Figure 3, A test set should still be held out for final evaluation, but the validation set is no longer needed when doing cross validation. In the basic approach, called k-fold CV, the training set is split into k smaller sets. The following procedure is followed for each of the k "folds":

- A model is trained using of the folds as training data;
- The resulting model is validated on the remaining data.

The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop. This approach can be computationally expensive, but it does not waste too much data, which is a major advantage in problems where the number of samples is very small (according to scikit-learn). For hyper-parameter tuning, I use Scikit-Learn GridSearchCV.

III. Model Estimation for Planar Data

In this section, I document the model estimation process for Planar dataset.

A. *K-Nearest Neighbors (KNN)*

For KNN model estimation, the hyper-parameters I tuned are n_neighbors (the number of neighbors used for classifying, values are [3, 5, 7, 9, 11, 13, 15, 25]) and weights ('uniform' or 'distance'). By GridSearchCV, the best model is using 7 neighbors and distance-weighting.

In Figure 4, the upper plot shows how the KNN model performs as the model complexity increases from 3 to 25. The scores used to judge the performance are f1 score and accuracy. It is observed the KNN performance does not improved significantly as the number of neighbors increases.

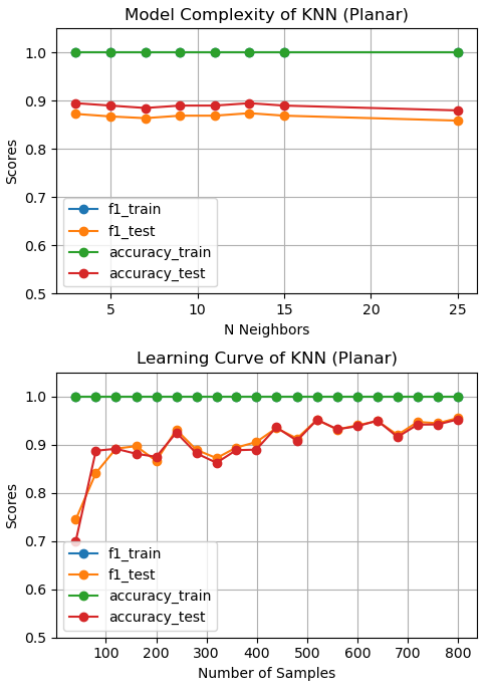

Fig. 4. KNN estimation (upper: model complexity; lower: learning curve).

In Figure 4, the lower plot shows the learning curve of KNN. The model performance increases significantly when sample size is less than 100. When have sample size larger than 100, the model performance still increases as number of samples, but at a much slower pace.

B. *Decision Trees (DTS)*

For DTS model estimation, the hyper-parameters I tuned are max_depth and min_samples_leaf. The default values for the two parameters will lead to fully grown and unpruned trees which can be very large on some data sets. To reduce memory consumption, the complexity and size of the trees should be controlled by setting those parameters, i.e. min_samples_leaf, and max_depth. By GridSearchCV, the best model is max_depth=9 and min_samples_leaf=4.

In Figure 5, the upper plot shows how the DTS model performs as the model complexity increases from 3 to 28. The scores used to judge the performance are f1 score and accuracy. It is observed the DTS performance increase significantly when max_depth is less than 5. When max_depth is larger than 10, the training scores still improve as max_depth increases, but the testing scores even decrease slightly, indicating overfitting when max_depth is larger than 10.

In Figure 5, the lower plot shows the learning curve of DTS. The model performance increases significantly when sample size is less than 100. When sample size is larger than 100, the model performance still increases as number of samples, but at a much slower pace.

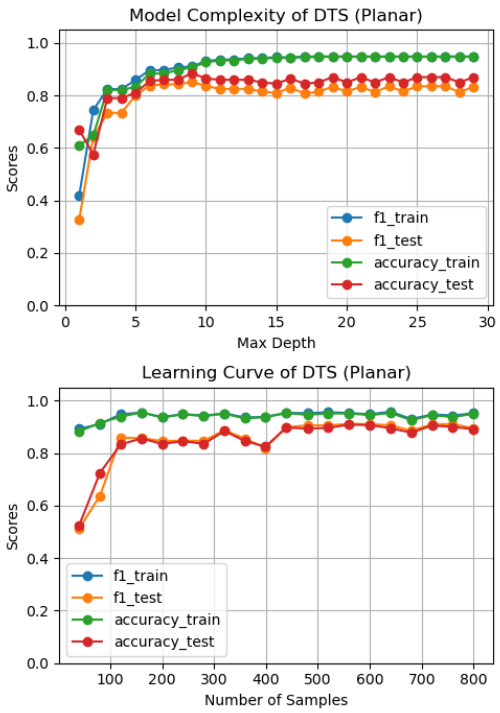

Fig. 5. DTS estimation (upper: model complexity; lower: learning curve).

C. *Boosting (BST)*

For BST model estimation, the hyper-parameters I tuned are min_samples_leaf, max_depth, learning_rate, n_estimators. By GridSearchCV, the best model is min_samples_leaf=20, max_depth=4.

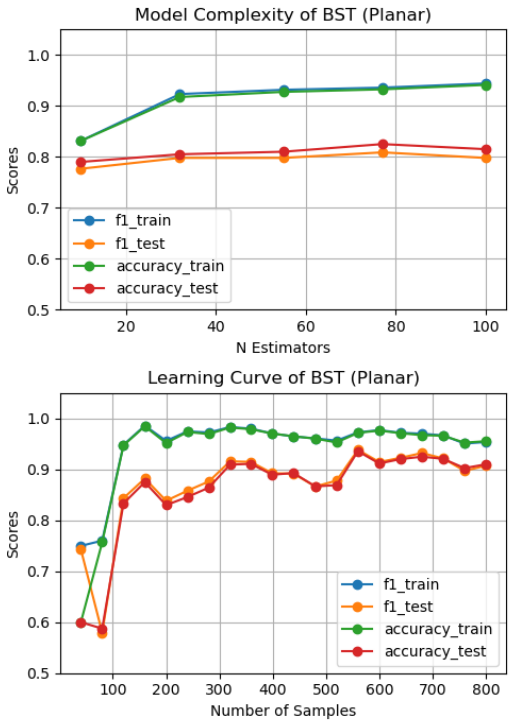

Fig. 6. BST estimation (upper: model complexity; lower: learning curve).

In Figure 6, the upper plot shows how the BST model performs as the model complexity increases from 5 to 100. It is observed the BST performance increase significantly when n_estimators is less than 35. When n_estimators is larger than 35, the training scores still improve as n_estimators increases, but the testing scores evening decrease slightly, indicating overfitting when n_estimators is larger than 35.

In Figure 6, the lower plot shows the learning curve of BST. The model performance increases significantly when sample size is less than 100. When sample size is larger than 100, the model performance still increases as number of samples, but at a much slower pace.

D. *Support Vector Machines (SVM)*

For SVM model estimation, the hyper-parameters I tuned are C (regularization parameter) and kernel (which are 'linear', 'poly', 'rbf', 'sigmoid'). By GridSearchCV, the best model is C=10, kernel='rbf'.

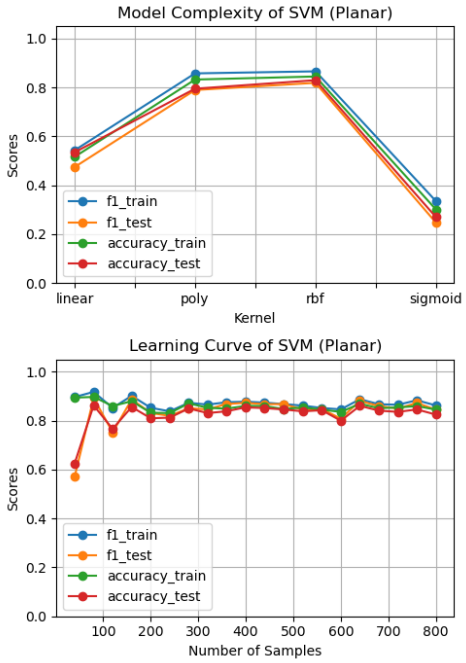

Fig. 7. SVM estimation (upper: model complexity; lower: learning curve).

In Figure 7, the upper plot shows how the SVM model performs as the model kennel varies. From this plot, it is observed that the poly and rbf kernel are best for the Planar dataset. The training and testing scores are very close, indicating insignificant overfitting.

In Figure 7, the lower plot shows the learning curve of SVM. The plots shows that SVM can achieved good performance with relatively small amount of data.

E. *Artificial Neural networks (ANN)*

For BST model estimation, the hyper-parameters I tuned are hidden_layer_sizes, activation, and learning_rate_init. By GridSearchCV, the best model is activation='logistic', hidden_layer_sizes=10, learning_rate_init=0.05.

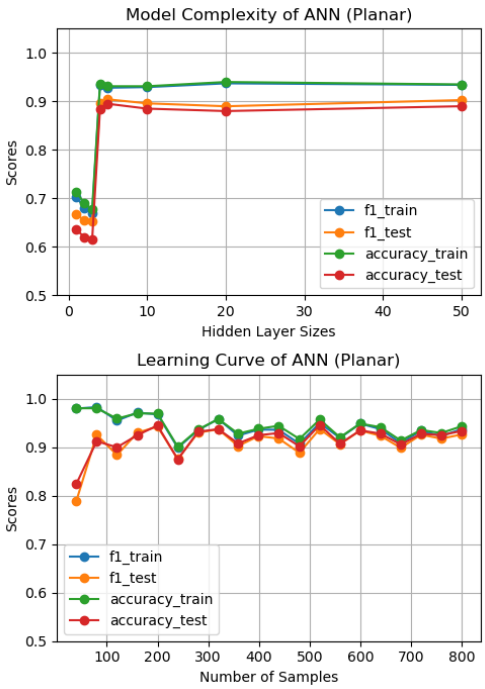

Fig. 8. ANN estimation (upper: model complexity; lower: learning curve).

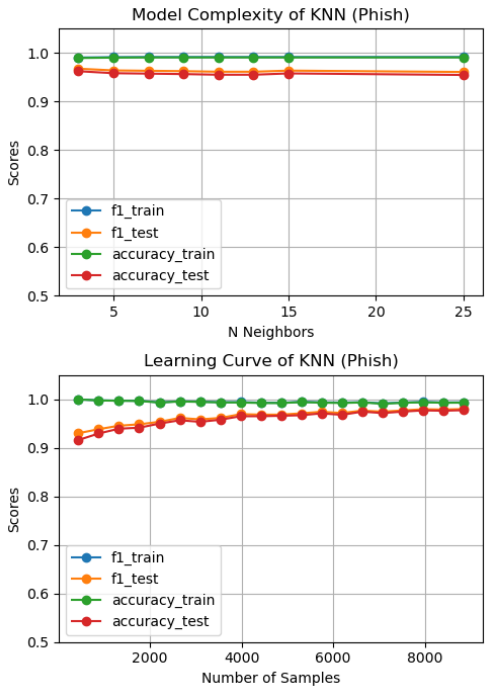

Fig. 9. KNN estimation (upper: model complexity; lower: learning curve).

In Figure 8, the upper plot shows how the ANN model performs as the hidden layer size increases from 2 to 50. It is observed the ANN performance increase significantly when hidden layer size is less than 5. When hidden layer size is larger than 5, the training and test scores do not improve.

In Figure 6, the lower plot shows the learning curve of ANN. The model performance increases significantly when sample size is less than 100. When sample size is larger than 100, the model training and testing scores converges, which indicates that the model is neither underfitting nor overfitting.

IV. Model Estimation for Phishing Data

In this section, I document the model estimation process for Phishing Websites dataset.

A. *K-Nearest Neighbors (KNN)*

For KNN model estimation, the hyper-parameters I tuned are n_neighbors (the number of neighbors used for classifying, values are [3, 5, 7, 9, 11, 13, 15, 25]) and weights (uniform or distance). By GridSearchCV, the best model is using 5 neighbors and distance-weighting.

In Figure 9, the upper plot shows how the KNN model performs as the model complexity increases from 3 to 25. The scores used to judge the performance are f1 score and accuracy. It is observed the KNN performance does not improved significantly as the number of neighbors increases.

In Figure 9, the lower plot shows the learning curve of KNN. The model performance increases gradually as the increase of samples, but the increase of scores is marginal and slower for larger number of samples.

B. *Decision Trees (DTS)*

For DTS model estimation, the hyper-parameters I tuned are max_depth and min_samples_leaf. By GridSearchCV, the best model is max_depth=7 and min_samples_leaf=44.

In Figure 10, the upper plot shows how the DTS model performs as the model complexity increases from 3 to 28. It is observed the DTS performance increase significantly when max_depth is less than 5. When max_depth is larger than 5, the training scores do not improve significantly and the training and testing scores are very close to each other, indicating the model is neither underfitting nor overfitting.

In Figure 10, the lower plot shows the learning curve of DTS. The learning curve of DTS becomes flat when number observation reaches 2000, indicating that 2000 samples is roughly sufficient to train DTS for Phishing Websites dataset. A good sign is that the training and testing scores almost overlaps with each other.

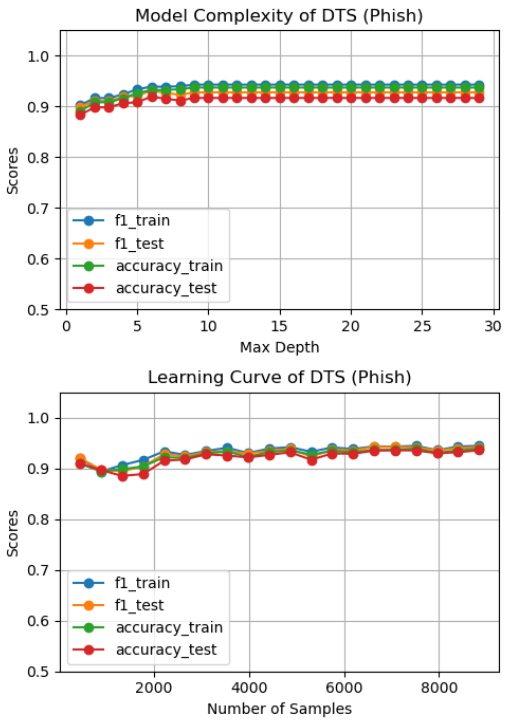

Fig. 10. DTS estimation (upper: model complexity; lower: learning curve).

C. *Boosting (BST)*

For BST model estimation, the hyper-parameters I tuned are min_samples_leaf, max_depth, learning_rate, n_estimators. By GridSearchCV, the best model is min_samples_leaf=44, max_depth=3.

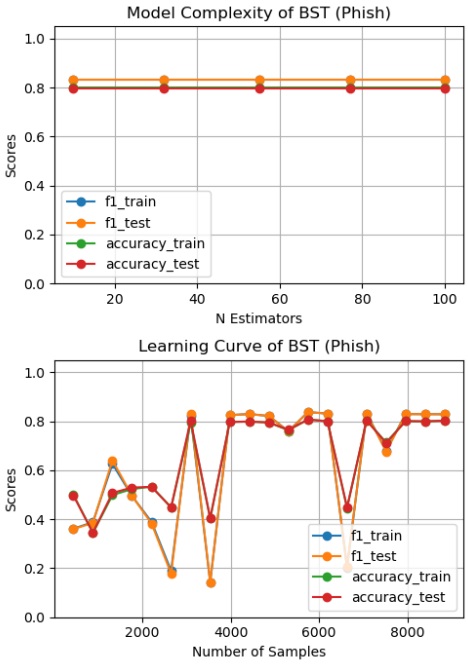

Fig. 11. BST estimation (upper: model complexity; lower: learning curve).

In Figure 11, the upper plot shows how the BST model performs as the number of estimators increases from 5 to 100. It is observed the BST performance does not vary much with number of estimators. This might be due to BST is not sensitive to the number of estimator parameter for the Phishing Websites dataset.

In Figure 11, the lower plot shows the learning curve of BST. The model performance increases significantly when sample size is less than 4000. It is observed, however, the scores of BST is very low even when the number of samples is larger than 6000. This might be due to some parameters are not well-tuned.

D. *Support Vector Machines (SVM)*

For SVM model estimation, the hyper-parameters I tuned are C (regularization parameter) and kernel (which are 'linear', 'poly', 'rbf', 'sigmoid'). By GridSearchCV, the best model is C=10, kernel='rbf'.

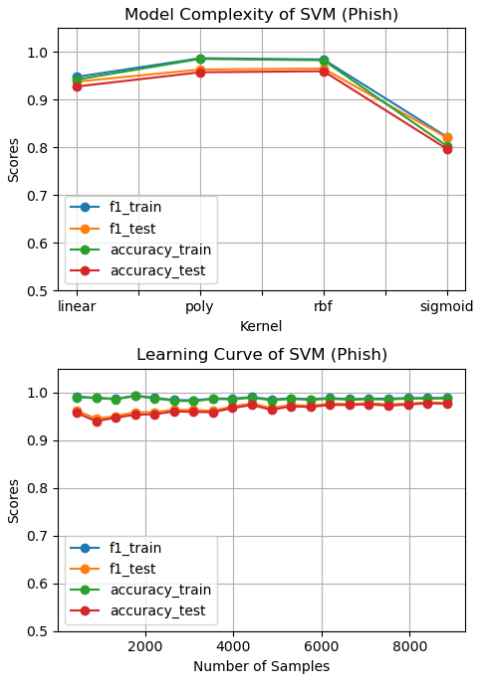

Fig. 12. SVM estimation (upper: model complexity; lower: learning curve).

In Figure 12, the upper plot shows how the SVM model performs as the model kennel varies. From this plot, it is observed that the poly and rbf kernel are best for the Phishing Websites dataset. The training and testing scores are very close, indicating insignificant overfitting.

In Figure 12, the lower plot shows the learning curve of SVM. The plots shows that SVM can achieved good performance with relative less amount of data. The test scores do increase when samples size is larger than 4000, but much slower.

E. *Artificial Neural networks (ANN)*

For BST model estimation, the hyper-parameters I tuned are hidden_layer_sizes, activation, and learning_rate_init. By GridSearchCV, the best model is activation='logistic', hidden_layer_sizes=5, learning_rate_init=0.05.

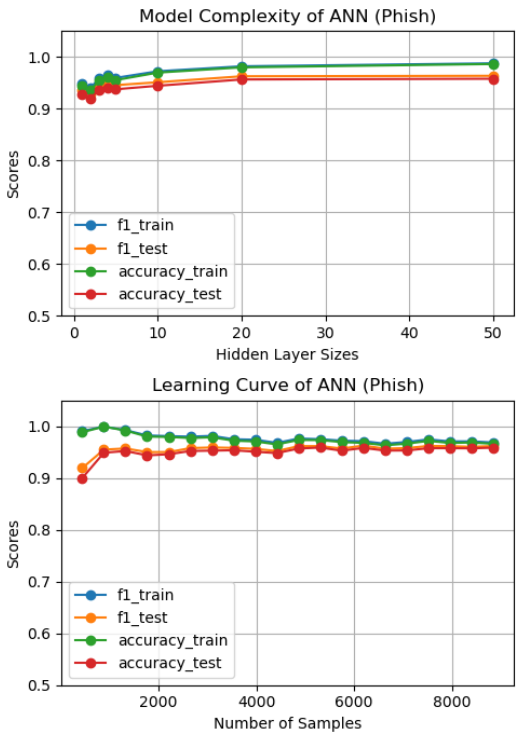

Fig. 13. ANN estimation (upper: model complexity; lower: learning curve).

In Figure 13, the upper plot shows how the ANN model performs as the hidden layer size increases from 2 to 50. It is observed the ANN performance increase significantly when hidden layer size is less than 5. When hidden layer size is larger than 5, the training and test scores do not improve significantly.

In Figure 13, the lower plot shows the learning curve of ANN. The model can achieve relatively good scores using roughly 1000 samples. When having more samples larger than 1000, the model training and testing scores converges, which indicates that the model is neither underfitting nor overfitting.

V. Decision Boundaries of Models

This section presents the decision boundaries of the selected algorithms for Planar dataset. Since we simulated the Planar dataset with two features and one class label, it is possible to plot the decision boundaries. The function to plot the decision boundary is adapted for Andrew Ng's deep learning class, which I finished two years ago.

A. *Decision Boundary of ANN*

First, the decision boundaries of ANN with different number of hidden units are presented. In Figure 14, six subplots of decision boundaries are plotted with the number of units varies from [1, 2, 4, 5, 20, 100].

Subplot 1 of Figure 14 shows the decision boundary of ANN with only one unit in hidden layer. The decision boundary is a linear plane, which tries hard to separates the data points into their correct label. But since this ANN has only one unit, the best it can do is to draw a line to separate the red and blue point as much as possible. With this line, this one-unit ANN can achieve roughly 75% accuracy, which is decent.

Subplot 2 of Figure 14 shows the decision boundary of ANN with two units in the hidden layer. With two units and logistic activation, the decision boundary is a curved plane slightly twisted at the center. With two units, the ANN model tries to separate the points but still do not have enough freedom to do so.

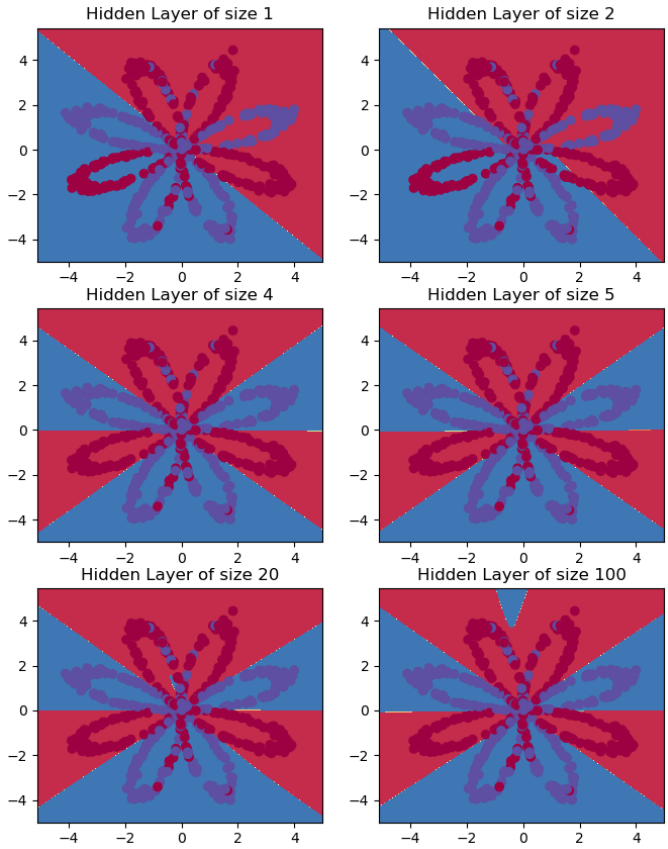

Fig. 14. Decision boundary of ANN with varying number of units in hidden layer.

In subplot 3 of Figure 14, the ANN has 4 units. With the 4 units in hidden layer, the ANN can correctly label each region to match the color of the majority number of points in their region. It is amazing and desired to see ANN have this capability to learn the pattern of Planar dataset.

In subplot 4 and 5, the number of units increase to 5 and 20, respectively. With the increased number of units, it is observed that the model performance does not increase. But we neither observe underfitting nor overfitting.

In subplot 6 of Figure 14, the number units increase to 100. We observer a curved small blue region at the top which should be marked red. This is a sign of overfitting. In this situation, the overfitted ANN is very tricky and sensitive. This model demonstrates instability.

B. *Decision Boundary of All Models*

In this section, I plot the decision boundaries of all models: KNN, DTS, BST, SVM, ANN. The parameters of these models are obtained by tuning these models to Planar dataset, which has been described in Section III.

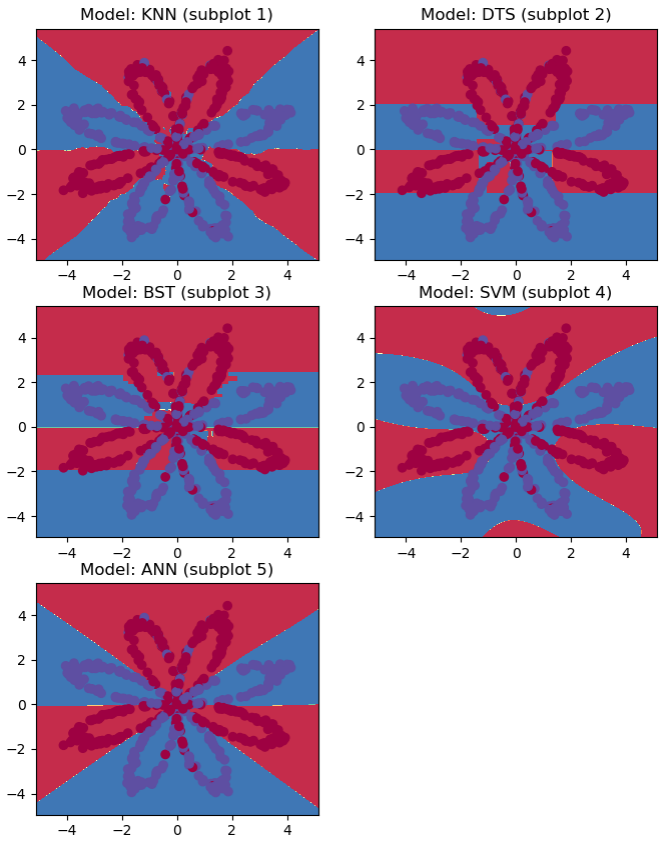

Fig. 15. Decision boundary of KNN, DTS, BST, SVM, ANN.

In subplot 1 of Figure 15, the decision boundary of KNN is plotted. In this subplot, it is observed that the KNN do a good job by labeling the regions correctly. The decision boundaries are curved but smooth, which is due to the weighting used in KNN.

In subplot 2 of Figure 15, the decision boundary of Decision Trees (DTS) is plotted. It is observed the decision boundary of DTS is not smooth curves but rectangles, which reflects the fact that the DTS use a particular value of a feature to separate the sample points.

In subplot 3 of Figure 15, the decision boundary of BOOSTING (BST) is plotted. The decision boundary of BST has similar pattern as DTS, since BST use DTSs as estimator and then improves the DTSs and aggregate the DTSs.

In subplot 4 of Figure 15, the decision boundary of SVM is plotted. With the 'rbf' kernel, the SVM can generate curved decision boundary, which can successfully separate most sample points.

In subplot 5 of Figure 15, the decision boundary of ANN is plotted. I have described the decision boundary of ANN in detail.

VI. Model Comparision

In this section, I compare the performance of the five models from the precision and efficiency perspective. The precision of the models are measured by five metrics: accuracy score, ROC AUC score, recall, precision, f1 score. The ROC AUC computes area under the Receiver Operating Characteristic Curve (ROC AUC) from prediction scores. F1 score combines precision and recall as the harmonic mean of precision and recall.

A. *Model Accuracy*

In Figure 16, we compare the model accuracy for Planar data. It is observed that KNN has top accuracy, roc_auc and f1 score, but KNN has lower precision but higher recall, indicating that this model is less balanced. The precision of SVM is relatively lower compared to KNN, DTS and ANN. More importantly, SVM also has lower precision but higher recall. DTS is among the models have top precision. The precision of BST is lowest among the five models. ANN has best precision among the five models. ANN also strikes a balance between recall and precision.

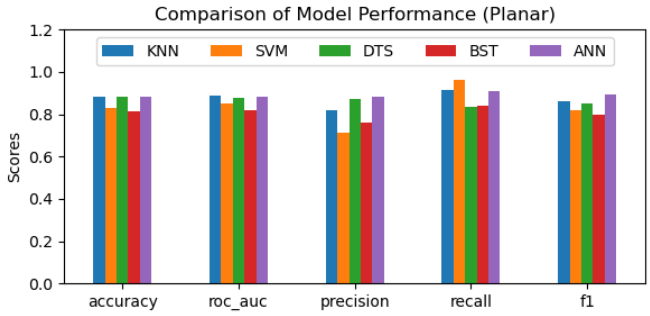

Fig. 16. Comparison of model accuracy for Planar data.

Figure 17 compares the precision of the five models for Phishing Websites data. It is observed the scores are close to 1, with 1 being indicator of good precision. It is observed that KNN, SVM and BST are the top performers and DTS and ANN being the low performers. All these models strike a good balance between recall and precision.

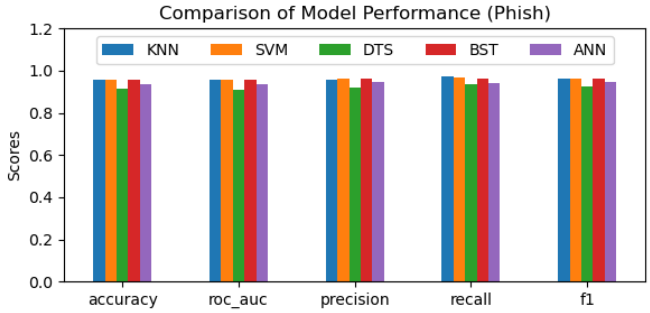

Fig. 17. Comparison of model accuracy for Phishing Websites data.

B. *Training Time*

In this section, the efficiency of the five models is compared. Since the prediction time is much smaller than training time, I will focus on comparing the training time.

Figure 18 presents the time used to train the five models for Planar data. I used 800 samples to train the five models. In Figure 18, it is observed that DTS, SVM and KNN used the smallest time to train, which is reasonable, since the three models are very simple. BST uses more time than the three simple models. The reason BST model uses more time is

because BST need to repeatedly improve for the samples that are mistakenly labeled. ANN uses most time because ANN uses gradient descent to modify the unit weights time and time again until the objective function converges.

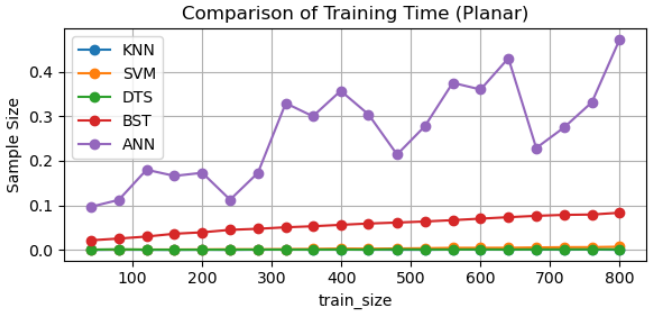

Fig. 18. Comparison of training time for Planar data.

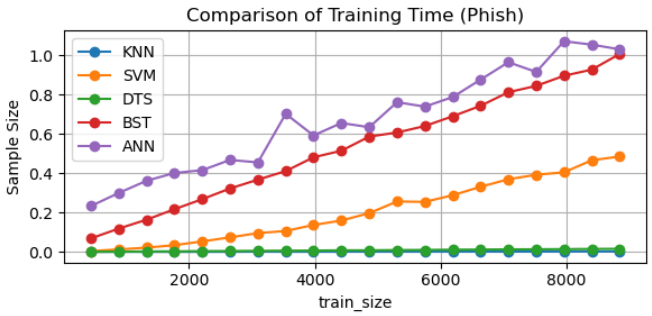

Fig. 19. Comparison of training time for Phishing Websites data.

Figure 19 compares the training time of the five models for Phishing Websites data. It is observed that DTS and KNN use the smallest amount of time and the training time barely increase with sample size. SVM uses more time than KNN and DTS, the time used for training increases linearly with the size of sample. ANN and BST use most time among the five models. The training time of ANN and BST increase linearly with sample size.

Conclusion and Future Work

In this report, I documented the training process of Decision trees (DTS), Artificial Neural networks (ANN), Boosting (BST), Support Vector Machines (SVM), k-nearest neighbors (KNN) for Planar and Phishing dataset. Then the decision boundaries of the five models are analyzed to gain insights of the five models. It is observed the shaped of the decision boundaries are determined by the model itself.

The model precision of the five models is compared. ANN performs best for Planar dataset, while KNN, SVM and BST perform best for Phishing Websites dataset, which is consistent with the fact that no single model is universally better than other models.

References

[1] https://en.wikipedia.org/wiki/Precision_and_recall

[2] https://www.kaggle.com/andresjejen/planar-utils0-py

[3] https://scikit-learn.org/stable/modules/cross_validation.html#cross-validation

[4] Mohammad, Rami, Thabtah, Fadi Abdeljaber and McCluskey, T.L. (2014) Predicting phishing websites based on self-structuring neural network. Neural Computing and Applications, 25 (2). pp. 443-458. ISSN 0941-0643

[5] Mohammad, Rami, McCluskey, T.L. and Thabtah, Fadi Abdeljaber (2014) Intelligent Rule based Phishing Websites Classification. IET Information Security, 8 (3). pp. 153-160. ISSN 1751-8709