Assignment 1: Supervised Learning

[Xinhuiyu Liu]
[xliu736@gatech.edu]

I. Introduction - Dataset Explanation

In this assignment, I choose two data sets to perform my experiments and analysis. The first (Dataset 1 hereafter) is from the Kaggle Austin Weather dataset. It contains Historical temperature, precipitation, humidity, and wind speed for Austin, Texas and the target variable is the weather events. The purpose of this dataset is to predict whether it is a clear day or rainy or has thunderstorms based on other meteorological variables. After some data cleaning and preprocessing, Dataset 1 is an imbalanced multi-classes dataset. It has three classes: clear, rain and thunderstorm and the class distributions are as follows: clear (65.1%), rain (14.6%) and thunderstorm (13.2%). It has 20 features and 1319 instances in total.

The second (Dataset 2 hereafter) is from OpenML Phishing-Websites dataset. It contains features like URL length, having subdomain or not, age of domain, page rank and google index etc. to predict whether the given website is a phishing website or not. So this is a dataset for binary classification. It is a generally balanced dataset and the class distributions are as follows: isPhishing (55.7%), notPhishing (44.3%). It has 31 features and 11055 instances in total.

I find these two classfication problems interesting because they are both real-world applications as it is helpful for people know whether it will be rain or not and whether a given a website is a phishing website or not if we can predict it accurately. These two datasets are also large-scale or high-dimensional Data and classification problems that involve large amounts of data or high-dimensional feature spaces can be interesting because they pose a challenge for developing efficient and effective models. Te different features of these two datesets also provide a way to explore how different algorithms perform on different datasets.

The remainder of this report is organized as follows: for each supervised machine learning algorithms and for each dataset, I first introduce how it works for classification problems, then I discuss the hyper-parameters I use to tune the model and the model complexity curve showing the model performance on both the training data (80% of the whole dataset) and test data (20% of the whole dataset) as a function of the degree of complexity of the model. Then I show the results for the best parameters of the model and use that to plot the learning curve which shows the model performance as a function of the training data size by splitting the dataset further into train and cross-validation datasets. In the final section, I compare different algorithms in terms of their performance using F1 score, accuracy, precision and recall, and model efficiency quantified by training time and prediction time.

II. Decision Trees

Decision tree is a supervised machine learning algorithm mainly used for Regression and Classification. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The splitting is based on a set of splitting rules based on classification features. Information gain is used in decision trees and random forest to decide the best split. The general rule is that we want nodes to be as pure as possible. Information gain is metric to calculate the difference between the entropy of the parent node and the children node. The final result is a tree with decision nodes and leaf nodes. Then, when predicting the output value of a set of features, it will predict the output based on the subset that the set of features falls into.

A. *Dataset 1*

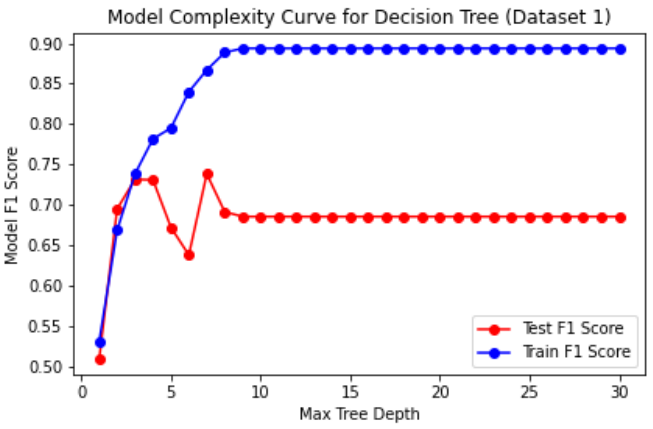

Fig. 1. Model F1 score (Decision Tree) as a function of max tree depth for Dataset 1.

Model Complexity Curve:

The hyper-parameters I experimented with to tune the decision tree model are maximum tree depth and the minimum number of samples required to be at a leaf node. The first plot (Fig. 1) is a model complexity curve for Dataset 1 which shows the model performance as a function of maximum tree depth. As the figure shows, the F1 score for the training data continues to increase as the model becomes more complex while the test F1 score fluctuates and then stabilizes. This suggests that the model underfits the data at the beginning, and as the maximum tree depth increases, the underfitting decreases but starts to overfit the data.

In order to find the optimal parameters for the model, I use GridSearchCV, which is a utility class in the scikit-learn

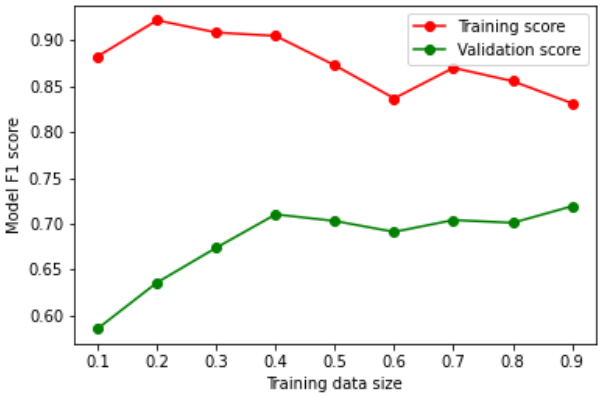

Fig. 2. Model F1 score (Decision Tree) as a function of training data size for Dataset 1.

library that performs an exhaustive search over a specified parameter grid for an estimator. In GridSearchCV, you specify a range of values for each hyperparameter, and the class will then perform a grid search by training and evaluating a model for every possible combination of hyperparameter values. At the end of the grid search, it will return the best combination of hyper-parameters that result in the best performance, as determined by a scoring metric such as accuracy, F1 score, or AUC. After this process, the best parameters are "max depth = 6, min sample leaf = 5.

Learning Curve:

Figure 2 shows the model performance as evaluated by F1 score as a function of the training data size. Training data size is shown in a fraction of the total training data. It may be counter-intuitive to see that the F1 score using training data generally decreases but the validation score increases as the training data size increases. But it is possible because Dataset 1 is an imbalanced dataset and the data distribution of the training and validation data could be different. So, the model may perform differently on each set. As the data size increases, the model may become better at fitting the validation data distribution and have a higher validation F1 score, even if the training F1 score decreases.

B. Dataset 2

Model Complexity Curve:

For the Dataset 2, model performances increases and then stabilizes as the max tree depth increases (Fig. 3). After performing GridSearchCV, the best parameters are "max depth = 9", "min sample leaf = 44".

Learning Curve:

As shown in Fig. 4, model performance improves as more training data are fed but starts to become poorer after the fraction of the training data reaches 0.9. When the amount of training data is limited, the model may not be able to learn the underlying patterns and relationships in the data, leading to underfitting, where the model is too simple to capture the complexity of the data. In this case, increasing the amount of training data can help the model better capture the complexity of the data, leading to improved performance.

However, as the amount of training data continues to increase, the model may start to overfit, where it becomes too complex and starts to memorize the training data instead of learning the underlying patterns. This can lead to poor performance on new, unseen data because the model has become too specific to the training data and is not able to generalize well to new situations.

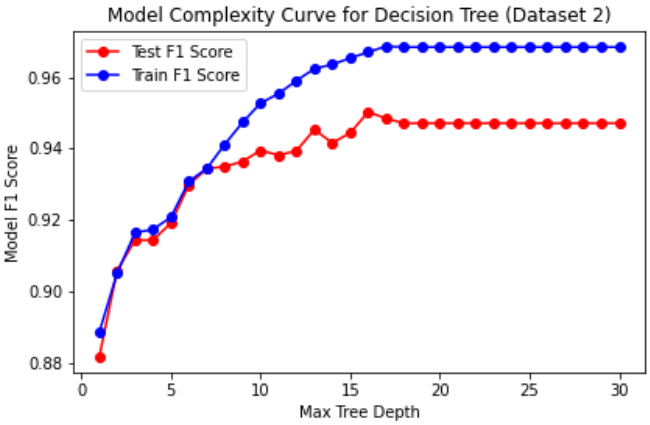

Fig. 3. Model F1 score (Decision Tree) as a function of max tree depth for Dataset 2.

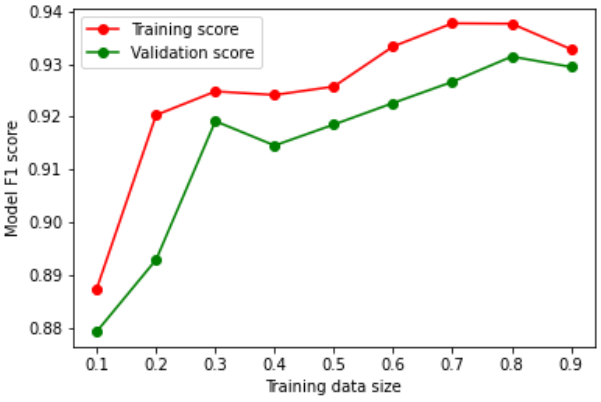

Fig. 4. Model F1 score (Decision Tree) as a function of training data size for Dataset 2.

III. KNN

K-Nearest Neighbors (KNN) is a simple and commonly used supervised learning algorithm for classification and regression problems. It is based on the idea that an instance can be classified by the majority vote of its neighbors in the feature space.

In KNN, the number of neighbors (K) to consider when making a prediction is specified by the user and is hyper-parameter to be tuned. For a given instance, the K nearest

neighbors are determined based on some distance metric, such as Euclidean distance or Manhattan distance. The prediction for the instance is then made based on the majority class of the K nearest neighbors.

KNN is a non-parametric method, meaning it does not make any assumptions about the underlying distribution of the data.

A. *Dataset 1*

Model Complexity Curve:

Figure 5 is the model complexity curve for the KNN model for Dataset 1. The hyper-parameter that needs to be tuned is K, which is the number of neighbors. We can see that train F1 score decreases as K increases and test F1 score first increases but then decreases as K increases. This is because when k is small, the model is more sensitive to outliers and may over-fit the training data. As k increases, the model becomes more robust to outliers, but also more likely to under-fit the data. This can lead to a decrease in the test F1 score as the model becomes less accurate in its predictions. Therefore, the best value of K would be around 20 as it gives the highest test F1 score.

Learning Curve:

As shown in Fig.6, the training F1 score and validation F1 score fluctuates but generally improves as there are more training data.

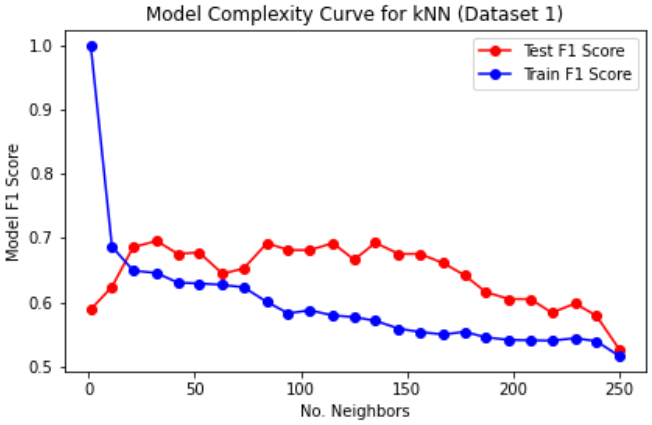

Fig. 5. Model F1 score (KNN) as a function of the number of hidden units for Dataset 1.

B. *Dataset 2*

Model Complexity Curve:

Model complexity curve for Dataset 2 (Fig. 7) shows that model performance better when K is smaller. This is a very interesting observation because normally we would think the model would overfit the data if the K value is really small. However, it is still possible for the model to perform well on the test data when we choose a very small value of K in KNN. This is probably because the data is well separated. If the data is well separated and there are clear boundaries between the different classes, a small value of K may be able to capture these boundaries and make accurate predictions.

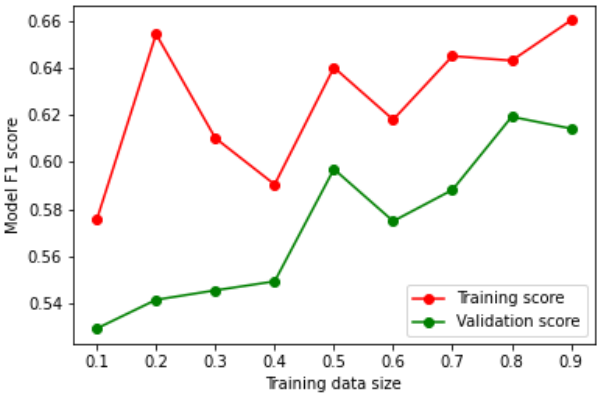

Fig. 6. Model F1 score (KNN) as a function of training data size for Dataset 1.

Learning Curve:

The learning curve (Fig. 8) is plotted with K = 1. It can be seen that the model generally performs better when there are more training data fed in since the validation score continues to improve.

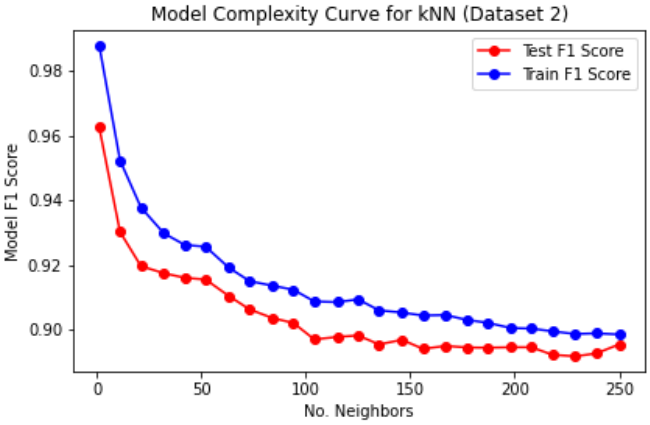

Fig. 7. Model F1 score (KNN) as a function of k for Dataset 2.

IV. Boosting

For the boosting method, I use Gradient Boosting algorithms and use decision tree as weak learners. It is an iterative ensemble learning method, which means that builds a series of weak models, each designed to correct the mistakes made by the previous model in the sequence. The predictions of all the models are then combined to make the final prediction. The number of weak learners (aka estimators), max tree depth, learning rate and min samples leaf are all hyper-parameters to tune for the boosted decision tree.

A. *Dataset 1*

Model Complexity Curve:

As shown in Fig. 9, both train F1 score and test F1 score improves as the number of estimators increase at the

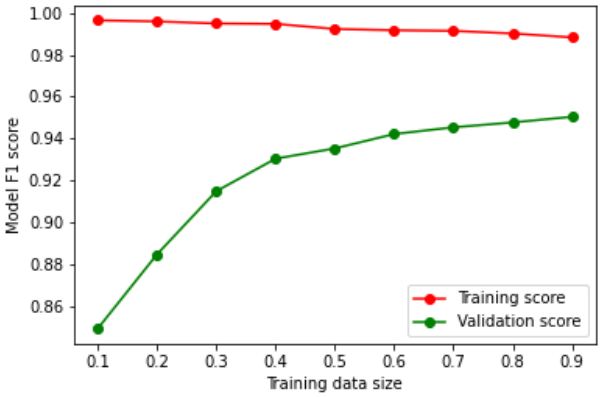

Fig. 8. Model F1 score (KNN) as a function of training data size for Dataset 2.

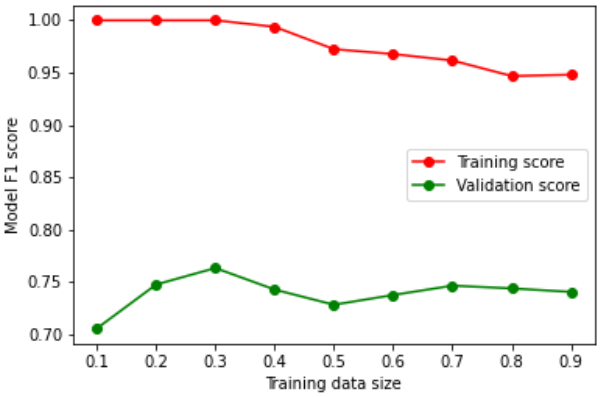

Fig. 10. Model F1 score (Boosting) as a function of training data size for Dataset 1.

beginning and then stabilize. After perform GridSearchCV for hyperparameter tuning, best parameters are: learning rate = 0.1, max depth = 3, min samples leaf = 5, n estimators = 55.

Learning Curve:

As suggested in Fig. 10, adding more training data helps to generalize the model at the beginning but after the fraction of the training data reaches 0.2, it doesn't help and validation score even decreases. It is because the model overfits the training data too much as the training score is much higher than the validation score, so adding more training data may not help to improve the validation score as the model will continue to fit the noise in the data rather than the underlying pattern.

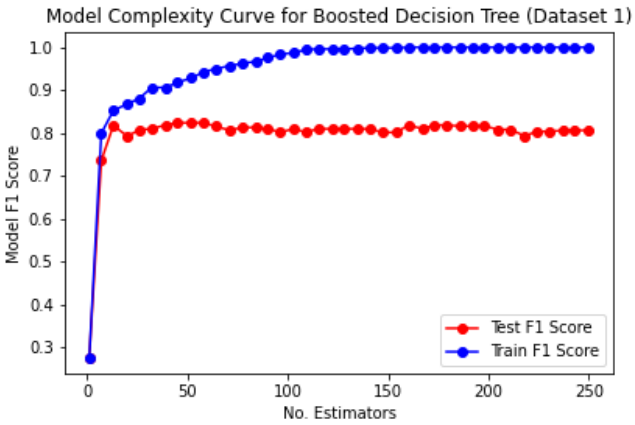

Fig. 9. Model F1 score (Boosting) as a function of the number of estimators for Dataset 1.

B. Dataset 2

Model Complexity Curve:

As shown in Fig. 11, it seems that the boosted decision tree model fits the Dataset 2 extremely well and both train F1 score and test F1 score improves as the number of estimators increase. And there is no overfitting issue happening. After perform GridSearchCV for hyperparameter tuning, best parameters are: learning rate = 0.1, max depth = 3, min samples leaf = 44, n estimators = 100.

Learning Curve:

Learning curve (Fig. 12) also shows that more training data help to generalize and improve the model performances as the validation score continues to increases.

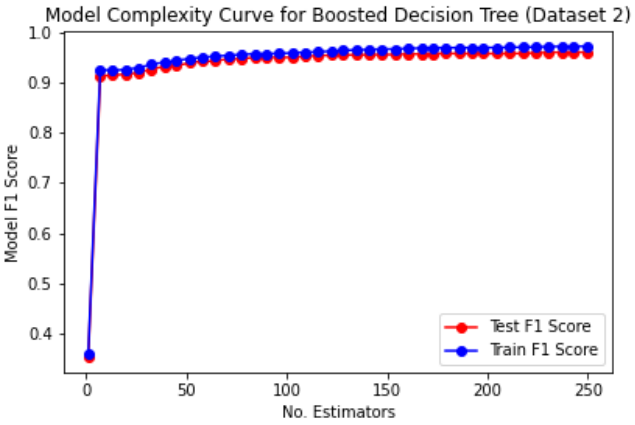

Fig. 11. Model F1 score (Boosting) as a function of the number of estimators for Dataset 2.

V. Neural Networks

For building a neural networks for classification problems, I use MLPClassifier in the scikit-learn library for Python that implements a multi-layer perceptron (MLP) algorithm. An MLP is a type of artificial neural network that consists of multiple layers of interconnected nodes, where each node in a layer receives inputs from the previous layer, processes these inputs, and passes the result to the next layer. The last layer of the network is usually the output layer that provides the final prediction for the classification task. The hyper-parameters I use to tune the MLPClassifier are the number of hidden layers and the learning rate.

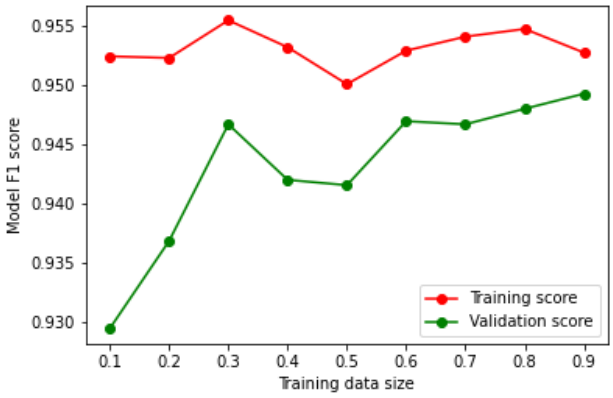

Fig. 12. Model F1 score (Boosting) as a function of training data size for Dataset 2.

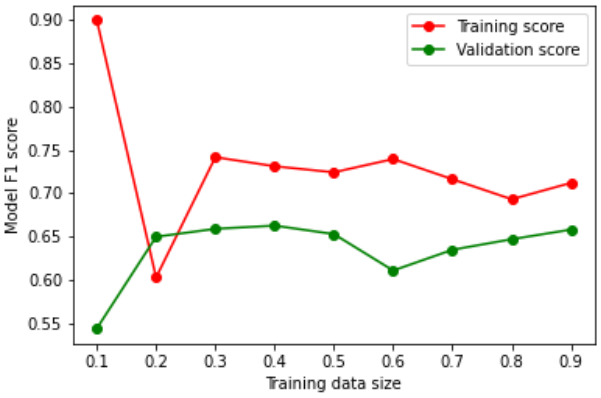

Fig. 14. Model F1 score (Neural Networks) as a function of training data size for Dataset 1.

A. *Dataset 1*

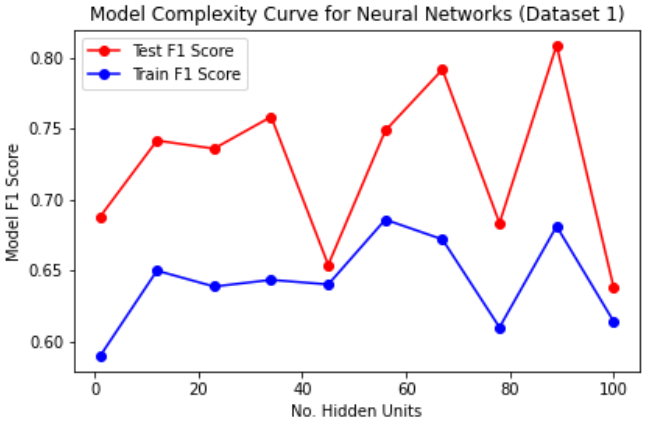

Fig. 13. Model F1 score (Neural Networks) as a function of the number of hidden units for Dataset 1.

Model Complexity Curve:

Model F1 score as a function of the number of hidden layers are shown in Fig. 13. It may look surprising that test F1 score is higher than the train F1 score and it indicates that the model is generalizing well to new, unseen data and is not overfitting the training data. It could be also by chance and the neural network could not fit the training data well because of the complex nature of the underlying pattern of the training data but it happens to predict the test data better. And the number of the hidden layers do not seem to impact the model performance as the model F1 scores fluctuates as the number of hidden layers increase. After performing GridSearchCV to find the optimal parameters, the best parameters are: number of hidden layers = 30, learning rate = 0.01.

Learning Curve:

Learning curve (Fig. 14) shows that training F1 score is very high but validation score is lowest when the fraction of training data is 0.1. This indicates the overfitting issue when there are not enough training data. And both the training and validation score improves later as there are more training data feeding into the neural networks.

B. *Dataset 2*

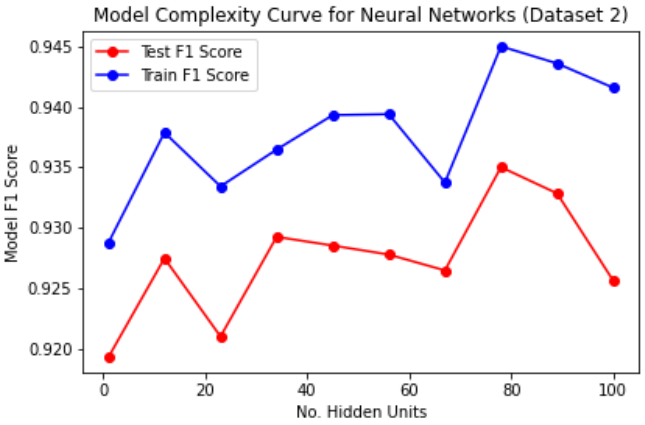

Fig. 15. Model F1 score (Neural Networks) as a function of the number of hidden units for Dataset 2.

Model Complexity Curve:

Compared to Dataset 1, Dataset 2 seems easier to be fitted and the train F1 score is consistently higher than the test F1 score as the number of hidden layers increase. And there is no significant trend of model performance improving as the number of hidden layers increase (Fig. 15). It is noted that Fig. 22 is plotted when learning rate is set to 0.1 in order to make the neural networks converge faster. Per hyper-parameter tuning, the best parameters are: number of hidden layers = 30, learning rate = 0.01.

Learning Curve:

Learning curve (Fig. 16) shows that the model performances improve as the training data size increases.

Figure 17 shows the loss curve for both the datasets and it suggests that neural networks definitely has a better perfor-

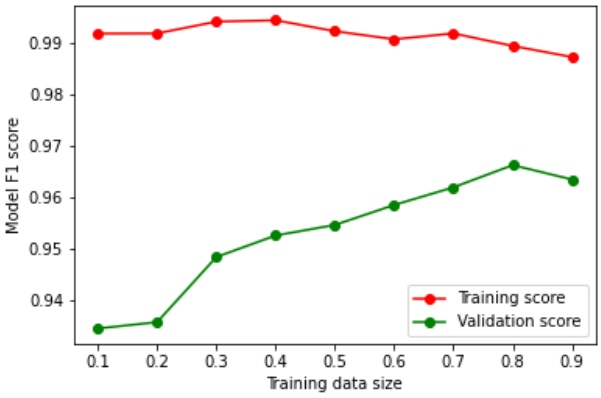

Fig. 16. Model F1 score (Neural Networks) as a function of training data size for Dataset 2.

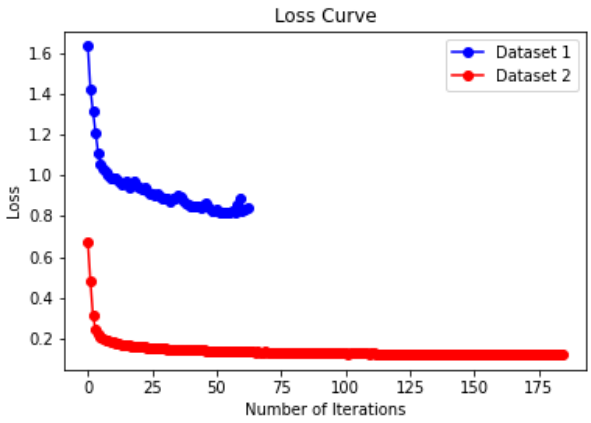

Fig. 17. Loss curve of Neural Networks for the two datasets.

mance on Dataset 2 and the loss can be converged to really low compared to the Dataset 1. However, it takes less iterations to converge for Dataset 1.

VI. SVM

Support Vector Machines (SVM) find a hyperplane that separates the data into different classes by maximizing the margin between the classes. There are three hyper-parameters that are considered here:

C: C is a regularization parameter that controls the trade-off between achieving a low training error and a low testing error. A smaller value of C will result in a wider margin, but a larger value of C will result in a smaller margin.

Kernel: The kernel is a function that maps the input data into a higher-dimensional space where it can be separated by a hyperplane. Common kernels include linear, polynomial, radial basis function (RBF), and sigmoid.

Gamma: Gamma is a hyperparameter in the radial basis function (RBF) kernel that determines the shape of the radial basis function. A larger value of gamma will result in a more complex decision boundary.

A. *Dataset 1*

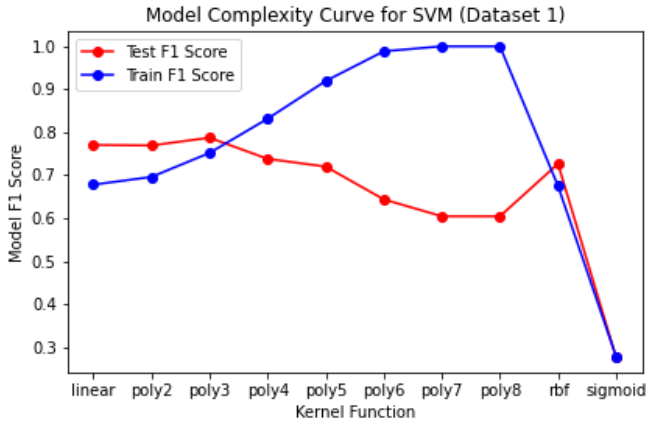

Fig. 18. Model F1 score (Neural Networks) as a function of the number of hidden units for Dataset 1.

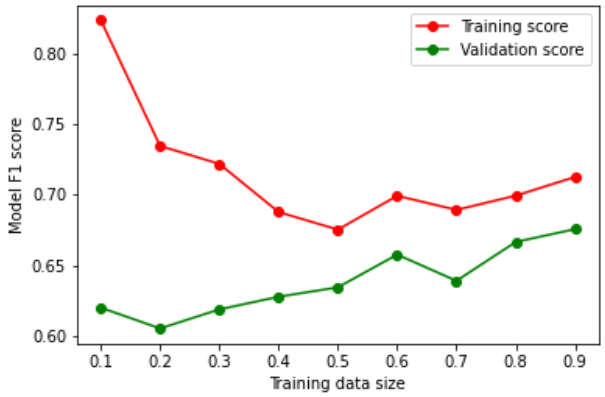

Fig. 19. Model F1 score (SVM) as a function of training data size for Dataset 1.

Model Complexity Curve:

Figure 18 shows the model performance as a function of different choices of the kernel function for Dataset 1. It can be clearly seen that the model overfits the data for high-order polynomial kernel functions. Per hyper-parameter tuning, the best parameters are: kernal = linear, C = 10.0, gamma = 1.

Learning Curve:

The learning curve for SVM on Dataset 1 looks similar to that of decision tree (Fig. 19). As the training data size increases, training F1 score decreases first but then increases and the validation score continues to improve. This suggests that model performances best when using all of the training data.

B. *Dataset 2*

Model Complexity Curve:

Figure 20 shows the model performance as a function of different choices of the kernel function for Dataset 2. And as

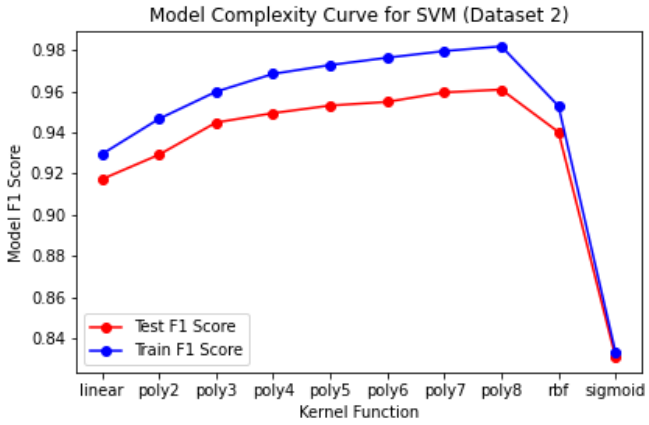

Fig. 20. Model F1 score (Neural Networks) as a function of the number of hidden units for Dataset 2.

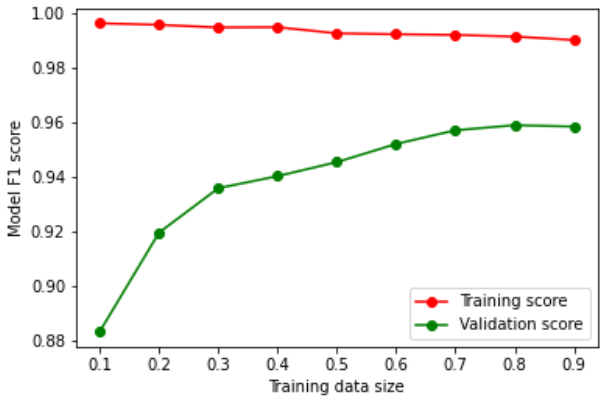

Fig. 21. Model F1 score (SVM) as a function of training data size for Dataset 2.

the order of polynomial kernel function increases, SVM seems to have a better performance. Per hyper-parameter tuning, the best parameters are: kernal = poly, C = 10.0, gamma = 1.

Learning Curve:

For Dataset 2, it is not surprising that model training score are constantly high and validation score continues to improve as there are more training data feeding into the model (Fig. 21).

VII. Model Comparison

The 5 classifiers are compared in terms of model performance, training time and prediction time for both the datasets. Model performances diff for each dataset but for both the datasets, training time generally increases as more training data are given. The prediction time is roughly same because the test data size doesn't change.

A. *Dataset 1*

Models with performance from the best to the worst are: boosted decision tree, neural networks, SVM, KNN, decision tree (Fig. 22). It is interesting boosted decision tree greatly improve the model performance compared to the single decision tree, indicating the power of the ensemble method. Neural network is also a very powerful supervised learning model and it is also takes long to train. It is not surprising that the two models with best performance (boosted tree and neural networks) also are the two that takes the longest to train (Fig. 23). The training time for the KNN model is much quicker than the prediction time because during the training phase, the algorithm simply stores the training data and there is no complex calculation or fitting of parameters involved. This makes the training phase relatively fast, especially when the size of the training data is small. On the other hand, during the prediction phase, the algorithm needs to compare the test data point with all the stored training data points, which can be computationally expensive, especially when the size of the training data is large. The prediction is based on the distance between the test data point and the training data points. The calculation of distances can be time-consuming, especially when there are many features in Dataset 1 (Fig. 24).

It is also worth noting that accuracy score and precision is really not a good indicator for evaluating model performances on imbalanced datasets. Figure 22 shows that accuracy score are all way higher than the F1 score for all kinds of algorithms.

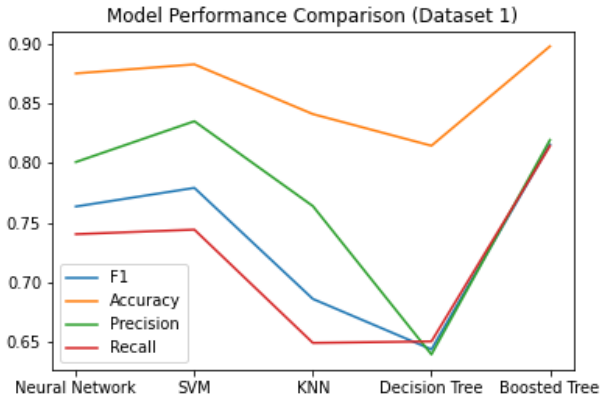

Fig. 22. Model performance comparison for Dataset 1.

B. *Dataset 2*

Models with performance from the best to the worst are: neural networks, SVM, KNN, boosted tree, decision tree (Fig. 25). Even though KNN is a simple model, it performs well on Dataset 2. However, the complicated boosted tree performs not as good as other models. It may be because the underlying pattern of the Dataset 2 is relatively simple so that simpler models can actually achieve better results than complex models. Neural Networks and SVM take longest time train and the training time of KNN and decision tree are negligible (Fig. 26). Training time for the boosted tree lies between. KNN and SVM take the longest time to predict while the prediction time for other models are negligible (Fig. 27).

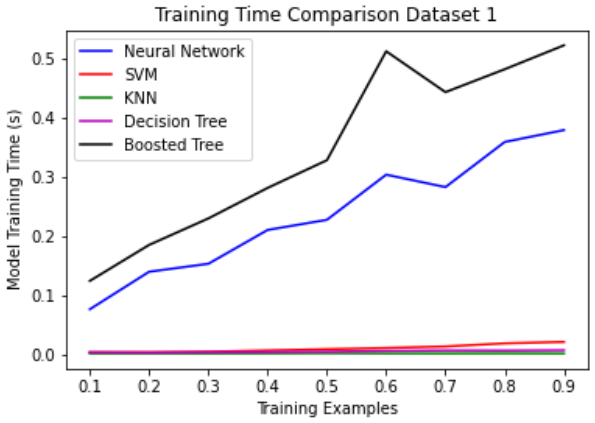

Fig. 23. Model comparison for training time as a function of training data size for Dataset 1.

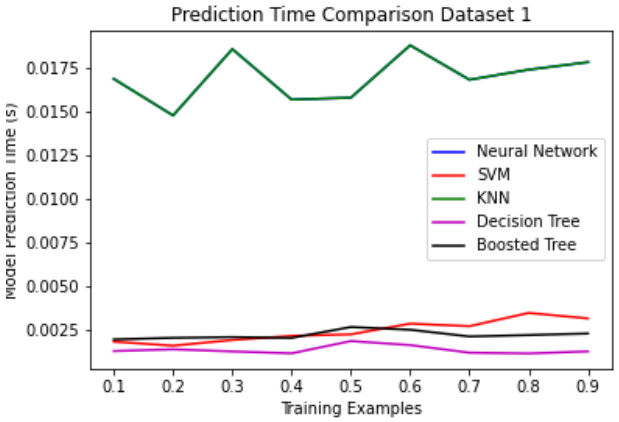

Fig. 24. Model comparison for prediction time as a function of training data size for Dataset 1.

Another thing to note is that since Dataset 1 is a balanced dataset, all metrics use to evaluate the model performance are similar.

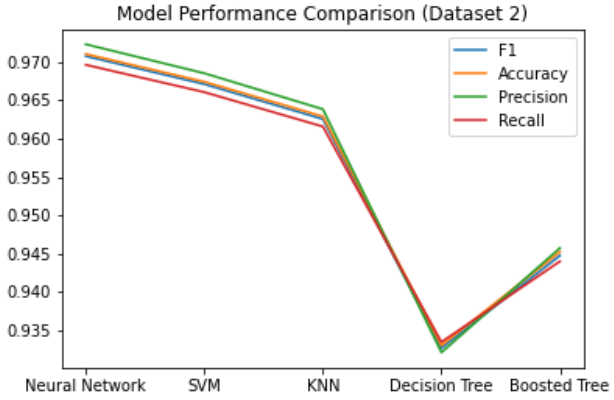

Fig. 25. Model performance comparison for Dataset 2.

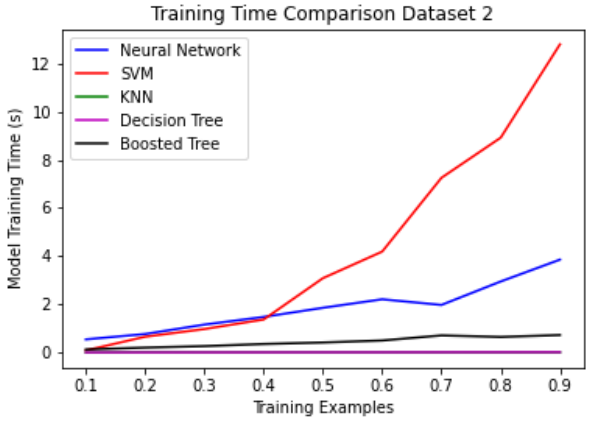

Fig. 26. Model comparison for training time as a function of training data size for Dataset 1.

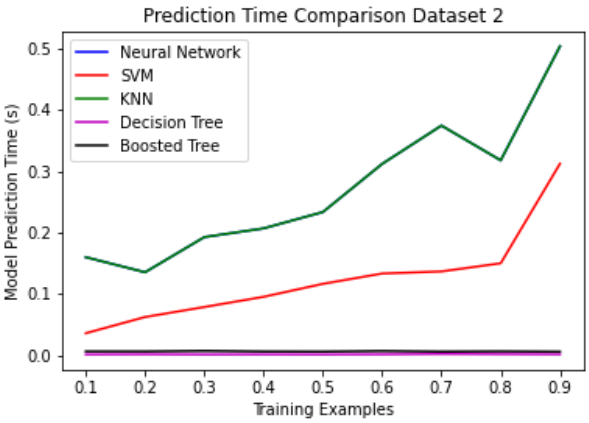

Fig. 27. Model comparison for prediction time as a function of training data size for Dataset 2.