CS7641 – Machine Learning

# Assignment 1

Jingyao Zhu

jzhu398@gatech.edu

## 1 Overview & Classification Problems

In this report, five machine learning algorithms including Decision tree, K-nearest neighbors, support vector machines, boosting, and neural networks have been applied to two dataset to solve binary classification problem.

### i. Dataset Overview

#### 1. Twitch gamer dataset

**(https://snap.stanford.edu/data/twitch_gamers.html)**

The first dataset is coming from Stanford snap data in Spring 2018, which is about 168,114 twitch gamers information. It has two separate CSV files involving 'edges.csv' and 'features.csv'. In the 'edges.csv', 6,797,557 edges have been linked. In the 'features.csv', it contains 9 features such as 'views', 'mature', 'lifetime', etc. 7 of 9 features in features.csv will be used to train ML models. Then edges in edges.csv will be used to count mutual friends, which will be used in the ML models. Overall, 8 features will be used to predict and dead accounts will be used as a tag "churn".

#### 2. Bike sharing demand

**(https://www.kaggle.com/c/bike-sharing-demand/overview)**

The second dataset is about bike-sharing systems. As many circles mentioned, automated renting bicycles could make human life so different in daily life throughout a city. But this system could have many processes such as obtaining membership, rental, and bike return. Rent a bike from one location and return it to another location. There are 12 features including duration of travel, departure location, arrival location, etc. The dataset could be used to forecast the frequency of rental bikes (high or low) in the Capital Bikeshare program in Washington, D.C.

### ii. Binary classification problem and interesting points

#### 1. Twitch gamer dataset

Twitch provides an American video live streaming service focused on video games, esports, and music. In its ecosystem, it has two ends – creators and users. The creators are the primary customers for Twitch since it relies on creators to make attractive contents for users to watch. But eventually, the company will be benefit from the high volume of users. Analysis on users end will be as much important as the one on creators end.

The first dataset provides much information from the user end. The target - churn is a binary classification problem whether a twitch gamer account will be alive or dead in the future. Understanding which kind of user will be churned and what is the potential reason for churned users is kind of interesting for me since many competitors may take over Twitch No.1 place in the future. Many new fields as improvement plans could be added to the Twitch service.

#### 2. Bike sharing demand

For the bike-sharing demand dataset, the original problem is to use the current dataset corresponding with weather information to predict demand expected as a regression problem. But it is hard to predict a specific number or range in real life because of the high error rate. Then according to the available cutoff online, I decide to change the scope of the problem from predicting demands to predicting high or low demands. Then the problem changes to a binary classification problem.

It can help companies who offer sharing bikes to predict the impact of marketing in different locations according to reliable predictions. Also, the ML models can help bike-sharing companies focus on different market strategies to avoid malicious competition. Moreover, companies may be able to launch differentiable services based on different demands and lower the cost in order to maximize the profit margin.

## 2 Analysis on Twitch gamer dataset

### i. Decision Tree

#### 1. parameter tuning and pruning

Decision Tree contains different methods to separate subtrees but not all of them will work for my problem. I selected parameters - criterion, max_depth, min_samples_leaf, min_impurity_decrease, and class_weight. As shown in Figure 1, the criterion has two values in it, which are 'gini' and 'entropy'. Which one is a good fit for the current problem? Compared to accuracy results, 'gini' will win this game by applying a 70% training set. I think the reason is because min gini to max gini is kind of 0 to 0.5 but for min entropy to max entropy is kind of 0 to 1. In this case, gini somehow will generally be better than entropy since the difference among data is small. As shown in Figure 2, when max_depth is larger than 5, decision tree accuracy on the training set stays same. However, the testing set is going down. In other words, the model may be overfitting the training set.

Following the same logic, grid search is applied to help search min_samples_leaf and min_impurity_decrease. The max_depth, min_samples_leaf and min_impurity_decrease are parameters to help prune the tree. max_depth is to avoid the tree growing too deep. min_impurity_decrease and min_samples_leaf are to help avoid too many non-informatic subtrees. But they are different. One is based on the sample in the branch, the other is based on information gain. Since my dataset is super imbalanced, I used 'balanced' in 'class_weight' to balance the dataset. 5 folders cross-validation is applied here because cross-validation will fully use the remaining data points to evaluate model performance, which could help solve an issue that not many data points are available to evaluate the model.

Finally, I provided the decision tree with the best parameter searched by grid search. Figure3 shows that training accuracy will drop a little bit with more data points in the imbalanced dataset. Here are the reasons. Data lacks one class examples when the imbalanced training set is small and the misclassification portion will be small. When more and more example (may have edge cases) came to model, it starts to truly learn and make mistakes. Eventually, it remains at 79.2% accuracy. Compared the training and cross-validation accuracy, they overlapped overall but it has slightly different at the beginning because cross-validation used data points much more efficiently. It has less different from the overall true accuracy for this model.

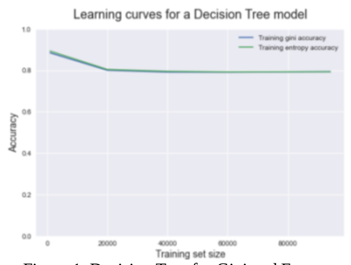

Figure 1: Decision Tree for Gini and Entropy

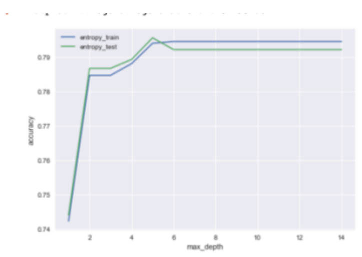

Figure 2: Search for max depth

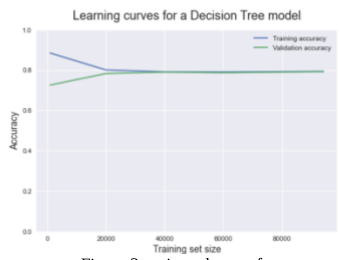

Figure 3: train and cv performance

**2. model explanation**

Decision tree is one of the induction learnings involving multiply knowledges such as statistics, data mining, machine learning. It is used tree mode to form the decisions based on the input and results. Generally speaking, a leaf in tree represents a step that could much faster move to clear decision and branch represents conjunctions of subset of whole dataset. In this dataset, I will use decision tree to solve binary classification problem.

**3. model performance and best parameters**

Since I balanced my dataset when training, model accuracy should be at least above 50%. The fewer misclassification errors, the better model will perform. The second method to say ‘success’ is that when the testing set is predicted by the trained decision tree model, it still could make even, or slightly above performance compared to training set model performance.

The best parameters are {‘class_weight’: ‘balanced’, ‘criterion’: ‘gini’, ‘max_depth’: 5, ‘min_impurity_decrease’: 0.001, ‘min_samples_leaf’: 50}. The training accuracy of the decision tree model is 79.2% and testing accuracy is 79.3%. Overall, decision tree wins the whole game compared with other algorithms.

## ii. Neural Networks

**1. parameter tuning and pruning**

MLP classifier contains different methods to separate its classes but not all of them will work for my problem. I selected ‘activation’, ‘learning_rate’, and ‘solver’ to help solve my problem. As shown in Figure 4, the solver has three values in it, which are ‘lbfgs’, ‘sgd’, ‘adam’. Compared to accuracy results, ‘lbfgs’ will win this game by applying a 70% training set. I think the reason is because ‘lbfgs’ will converge fast in small dataset with compatible performance. ‘lbfgs’ means an optimizer in the family of quasi-Newton methods, ‘adam’ means stochastic gradient-based optimizer and ‘sgd’ means stochastic gradient descent. In this case, ‘lbfgs’ somehow will generally be better since my dataset is small. As shown in Figure 5, when relu is applied, MLPClassifier accuracy on the training set and testing set provides the highest since relationship is quite close to rectified linear.

Following the same logic, grid search is applied to help search ‘learning_rate’. 5 folders cross-validation is applied here because cross-validation will fully use the remaining data points to evaluate model performance, which could help solve an issue that not many data points are available to evaluate the model.

Finally, I provided the MLPClassifier with the best parameter searched by grid search. Figure 6 shows that training will drop a little bit with more data points in the imbalanced dataset. The reason has explained in decision tree section. Eventually, it remains at 57.2% accuracy. Compared the training and cross-validation accuracy, they overlapped overall but it has slightly different at the beginning because cross-validation used data points much more efficiently. It has less different from the overall true accuracy for this model.

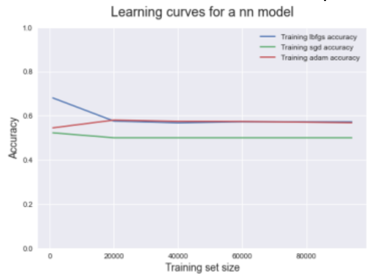

Figure 4: Different solver in NN

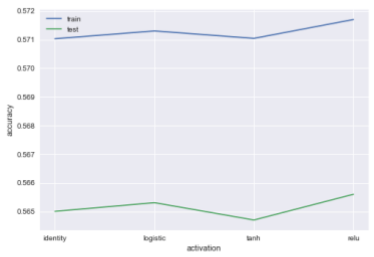

Figure 5: Search for activation

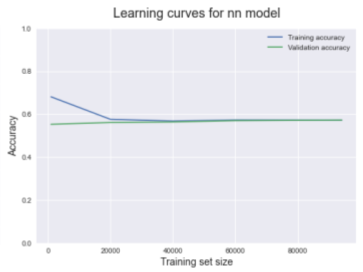

Figure 6: train and cv performance

**2. model explanation**

The neural network mimics the way the human brain does to learn and process the relationships of sets of data. But in this specific problem, MLP classifier is one of the artificial neural networks that have multiple layers of perceptrons. Generally speaking, it contains three layers including an input layer, a hidden layer and an output layer. It feeds the data with feedforward and corrects the mistakes by backpropagation.

**3. model performance and confusion matrix**

Since I balanced my dataset when training, model accuracy should be at least above 50%. The fewer misclassification errors, the better model will perform. The second method to say ‘success’ is that when the testing set is predicted by the trained neural networks model, it still could make even, or slightly above performance compared to training set model performance.

The best parameters are {‘activation’: ‘relu’, ‘learning_rate’: ‘constant’, ‘solver’: ‘lbfgs’}. The training accuracy of the neural networks model is 57.2% and testing accuracy is 56.6%.

## iii. Boosting

### 1. parameter tuning and pruning

Adaboost is based on the tree methods to provide good split points. the critic parameters are ‘algorithm’, ‘learning_rate’, and ‘n_estimators’. As shown in Figure 7, the ‘algorithm’ has two values – ‘SAMME’ and ‘SAMME.R’. Based on formula, SAMME.R algorithm provides all weak learners an equal weight, but SAMME algorithm uses weighted weak learners. And another difference is that SAMME.R used probability but SAMME provides discrete-valued algorithm such as 0 or 1. Technically, SAMME.R will converge faster than SAMME with compatible performance. The learning_rate is that weight will be updated every iterator based on the current learning rate. If the learning rate is too high, it may miss the local/global minimum. If learning rate is too low, weight will be updated too slow and without enough estimator, it won’t go local/global minimum either. Estimator is the max instance that will be provided before early stop. As shown in Figure 8, when ‘learning_rate’ starts from 0.25, Adaboost accuracy on the training set and testing set keeps going up. When it hits 1, it provides 57.5% accucracy in the training and 56.8% in the testing set. 5 folders cross-validation is applied for more accurate results. Figure 9 shows that training will drop a little bit with more data points in the imbalanced dataset. The reason has explained in decision tree section.

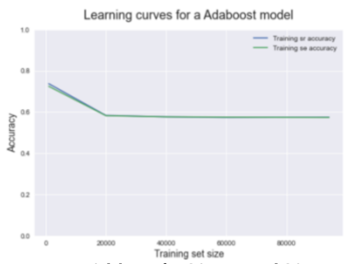

Figure 7: Adaboost for SAMME and SAMME.R

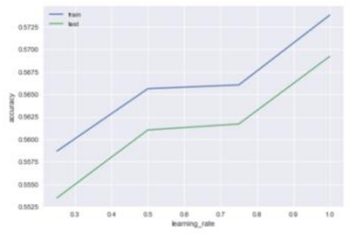

Figure 8: Search for learning rate

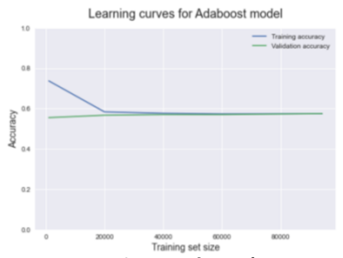

Figure 9: train and cv performance

### 2. model explanation

Boosting is a machine learning methodology that will highlight incorrect instances and try to correct them literally. Adaptive Boosting is an ensemble classification algorithm that combines with many weak learners to get more accurate results. Even though the individual learners may be weaker, overall group of weaker learners can be converted to a strong learner.

### 3. model performance and best parameter

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say ‘success’ is that when the testing set is predicted by the trained adaboost model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, Adaboost performs better if adaboost has much higher accuracy than others.

The best parameters are {'algorithm': 'SAMME.R', 'learning_rate': 1, 'n_estimators': 40}. The training accuracy of the decision tree model is 57.4% and testing accuracy is 56.8%.

## iv. Support Vector Machines

### 1. parameter tuning and pruning

SVM is based on the tree methods to provide good split points. the critic parameters are ‘decision_function_shape’ and ‘kernel’. As shown in Figure 10, the ‘kernel’ has four values – linear, poly, rbf, and sigmoid. Since the relationship between independent variable and dependent variable is linear, then kernel is linear. As shown in Figure 11, one-vs-one shapes (n_samples, n_classes * (n_classes - 1) / 2) and one-vs-rest shapes like (n_samples, n_classes) in the decision_function_shape. Based on the function, ovo closes to my dataset since my dataset is like (12, 40000+). 5 folders cross-validation is applied for more accurate results. Figure 12 shows that training remains the same all the time, but validation starts going up and then stays the same. The reason has explained in decision tree section.

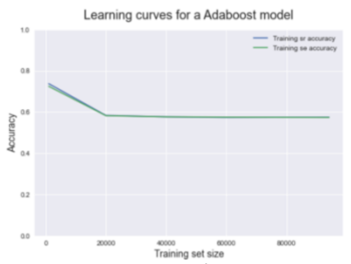

Figure 10: svm for kernels

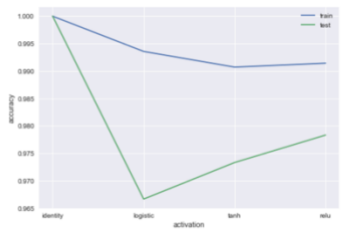

Figure 11: Search for decision_function_shape

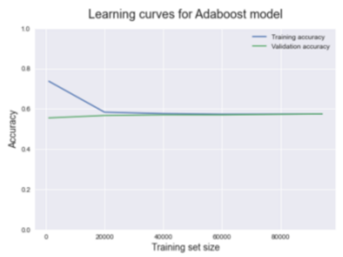

Figure 12: train and cv performance

### 2. model explanation

In machine learning, support vector machine is one of the supervised learning that maps a hyperplane to maximum the width of the gap between the two classes in solving classification analysis by analyzing training data. Whenever there are new examples, the algorithm will map them into that same space and provide the class based on where they fall.

#### 3. model performance and confusion matrix

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say 'success' is that it should be fast to compute. Then when the testing set is predicted by the trained svm model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, svm performs better if svm has much higher accuracy than others.

The best parameters are {'decision_function_shape': 'ovo', 'kernel': 'linear'}. The training accuracy of the decision tree model is 59.8% and testing accuracy is 56.8%.

### v. K-nearest neighbors

#### 1. parameter tuning and pruning

The knn contains different methods to estimate new points based on the current known data points. The critic parameters are 'n_neighbors' and 'weights'. As shown in Figure 13, the 'weights' has two values - 'uniform' and 'distance'. Compared to accuracy results, 'uniform' will win this game by applying a 70% training set. I think the reason is that I normalized the original features and all features are same important to predict target. 'uniform' means that weight all points equally but distance means that closer neighbors will have greater weight. As shown in Figure 14, when 'n_neighbors' hits 45, knn accuracy on the training set remains but knn accuracy on the testing set starts drop. In other words, the algorithm starts overfitting the problem. 5 folders cross-validation is applied for more accurate results.

Figure 15 shows that training and cross-validation accuracy goes up at the beginning but remains the same when data is large. Here are the reasons. When data set is small and n is not small, it will start learning some data points that should not be involve in this class. When data is large enough and n remains, the size of data will not affect so much since it only collects close enough data point.

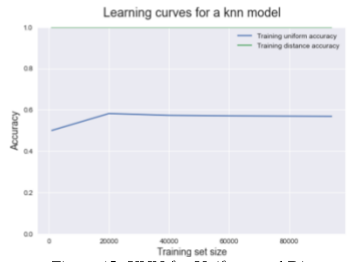

Figure 13: KNN for Uniform and Distance

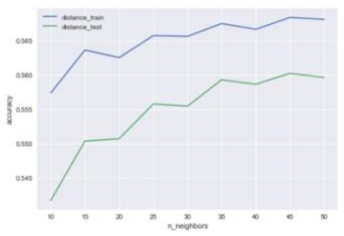

Figure 14: Search for n neighbors

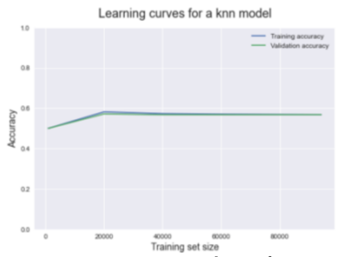

Figure 15: train and cv performance

#### 2. model explanation

K-Nearest Neighbors is a type of instance-based learning (lazy learning) and a non-parametric classification method. An observation is classified by the number of k nearest neighbors using k times of voting. The output is a group of classes. Since this algorithm relies on approximate estimation if the features represent the same units of the training data, it can improve its accuracy.

#### 3. model performance and best parameters

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say 'success' is that when the testing set is predicted by the trained decision tree model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, knn performs better if knn has much higher accuracy than others.

The best parameters are {'n_neighbors': 45, 'weights': 'uniform'}. The training accuracy of the decision tree model is 56.7% and testing accuracy is 56.1%.

## 3 Analysis on Bike sharing demand dataset

### i. Decision Tree

#### 1. parameter tuning and pruning

Decision Tree contains different methods to separate subtrees but not all of them will work for my problem. I selected parameters - criterion, max_depth, min_samples_leaf, min_impurity_decrease, and class_weight. As shown in Figure 1, the criterion has two values in it, which are 'gini' and 'entropy'. Compared to accuracy results, 'entropy' will win this game by applying a 70% training set. I think the reason is because min gini to max gini is kind of 0 to 0.5 but for min entropy to max entropy is kind of 0 to 1. In this case, entropy somehow will generally be better than gini because the difference among data is big. As shown in Figure 2, when max_depth is larger than 3, decision tree accuracy on the training set stays the same but decision tree accuracy on the testing set starts drop. In other words, the model may be overfitting the training set.

Following the same logic, grid search is applied to help search min_samples_leaf and min_impurity_decrease. The max_depth, min_samples_leaf and min_impurity_decrease are parameters to help prune the tree. max_depth is to avoid the tree growing too deep. min_impurity_decrease and min_samples_leaf are to help avoid too many non-informatic subtrees. But they are different. One is based on the sample in the branch, the other is based on information gain. Since my dataset is super imbalanced, I used 'balanced' in 'class_weight' to balance the dataset. 5 folders cross-validation is applied here because cross-validation will fully use the remaining data points to evaluate model performance, which could help solve an issue that not many data points are available to evaluate the model.

Finally, I provided the decision tree with the best parameter searched by grid search. Figure 3 shows that training and validation accuracy will go up a little bit with more data points coming in. Here is the reason. Data is not enough to train the model at the beginning and the misclassification portion will be quite big. When more and more example (may have edge cases) came to model, it starts to learn and correct mistakes. Eventually, it remains at 95.7% accuracy. Compared the training and cross-validation accuracy, they overlapped overall but it has slightly different at the beginning because cross-validation used data points much more efficiently. It has less different from the overall true accuracy for this model.

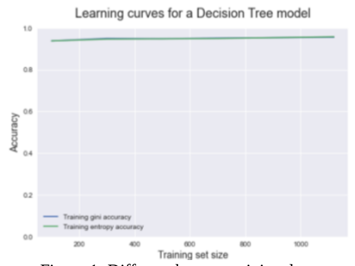

Figure 1: Different between gini and entropy

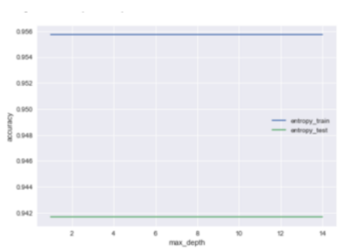

Figure 2: Search for max depth

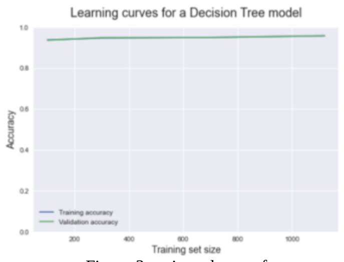

Figure 3: train and cv performance

### 2. model explanation

Decision tree is one of the induction learnings involving multiply knowledges such as statistics, data mining, machine learning. It is used tree mode to form the decisions based on the input and results. Generally speaking, a leaf in tree represents a step that could much faster move to clear decision and branch represents conjunctions of subset of whole dataset. In this dataset, I will use decision tree to solve binary classification problem.

### 3. model performance and confusion matrix

Since I balanced my dataset when training, model accuracy should be at least above 50%. The fewer misclassification errors, the better model will perform. The second method to say 'success' is that when the testing set is predicted by the trained decision tree model, it still could make even, or slightly above performance compared to training set model performance.

The best parameters are {'class_weight': 'balanced', 'criterion': 'entropy', 'max_depth': 3, 'min_impurity_decrease': 0.001, 'min_samples_leaf': 50}. The training accuracy of the decision tree model is 95.7% and testing accuracy is 96.5%. Overall, decision tree wins the whole game compared with other algorithms.

## ii. Neural Networks

### 1. parameter tuning and pruning

MLP classifier contains different methods to separate its classes but not all of them will work for my problem. I selected 'activation', 'learning_rate', and 'solver' to help solve my problem. As shown in Figure 4, the solver has three values in it, which are 'lbfgs', 'sgd', 'adam'. Compared to accuracy results, 'lbfgs' will win this game by applying a 70% training set. I think the reason is because 'lbfgs' will converge fast in small dataset with compatible performance. 'lbfgs' means an optimizer in the family of quasi-Newton methods, 'adam' means stochastic gradient-based optimizer and 'sgd' means stochastic gradient descent. In this case, 'lbfgs' somehow will generally be better since my dataset is small. As shown in Figure 5, when identity is applied, MLPClassifier accuracy on the training set and testing set provides the highest since it is a linear relationship overall.

Following the same logic, grid search is applied to help search 'learning_rate'. 5 folders cross-validation is applied here because cross-validation will fully use the remaining data points to evaluate model performance, which could help solve an issue that not many data points are available to evaluate the model.

Finally, I provided the MLPClassifier with the best parameter searched by grid search. Figure 6 shows that training stay 99.5% all the time but the validation starts lower and eventually overlaps with training. Compared the training and cross-validation accuracy, they overlapped overall but it has slightly different at the beginning because cross-validation used data points much more efficiently. It has less different from the overall true accuracy for this model.

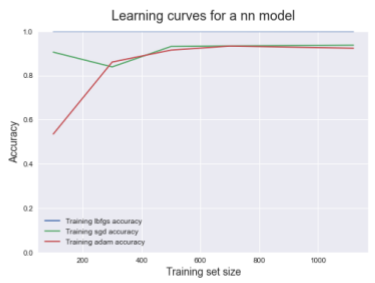

Figure 4: Different solver in NN

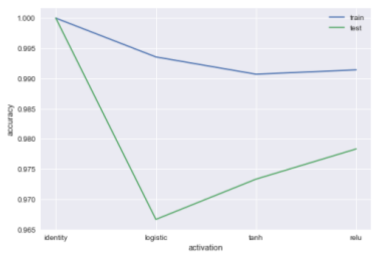

Figure 5: Search for activation

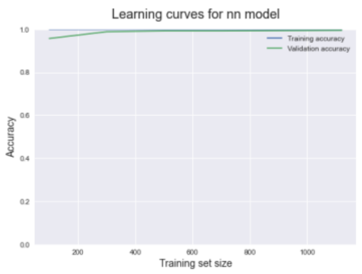

Figure 6: train and cv performance

**2. model explanation**

The neural network mimics the way the human brain does to learn and process the relationships of sets of data. But in this specific problem, MLP classifier is one of the artificial neural networks that have multiple layers of perceptrons. Generally speaking, it contains three layers including an input layer, a hidden layer and an output layer. It feeds the data with feedforward and corrects the mistakes by backpropagation.

**3. model performance and confusion matrix**

Since I balanced my dataset when training, model accuracy should be at least above 50%. The fewer misclassification errors, the better model will perform. The second method to say 'success' is that when the testing set is predicted by the trained neural networks model, it still could make even, or slightly above performance compared to training set model performance.

The best parameters are {'activation': 'identity', 'learning_rate': 'constant', 'solver': 'lbfgs'}. The training accuracy of the neural networks model is 99.5% and testing accuracy is 99.3%.

## iii. Boosting

**1. parameter tuning and pruning**

Adaboost is based on the tree methods to provide good split points. the critic parameters are 'algorithm', 'learning_rate', and 'n_estimators'. As shown in Figure 7, the 'algorithm' has two values – 'SAMME' and 'SAMME.R'. Based on formula, SAMME.R algorithm provides all weak learners an equal weight, but SAMME algorithm uses weighted weak learners. And another difference is that SAMME.R used probability but SAMME provides discrete-valued algorithm such as 0 or 1. In this case, SAMME will be better since my problem is a binary classification problem. The learning_rate is that weight will be updated every iterator based on the current learning rate. If the learning rate is too high, it may miss the local/global minimum. If learning rate is too low, weight will be updated too slow and without enough estimator, it won't go local/global minimum either. Estimator is the max instance that will be provided before early stop. As shown in Figure 8, when 'learning_rate' starts from 0.25, Adaboost accuracy on the training set and testing set keeps going up. When it hits 0.5, it provides 100% accuracy in the training and 99.9% in the testing set. After 0.5, training accuracy remains the same but testing accuracy starts drop. 5 folders cross-validation is applied for more accurate results. Figure 9 shows that training remains the same all the time but validation starts going up and then stays the same. The reason has explained in decision tree section.

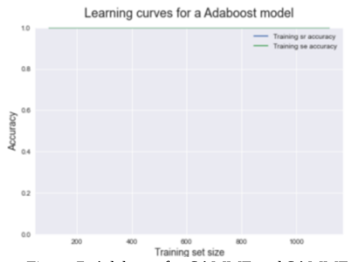

Figure 7: Adaboost for SAMME and SAMME.R

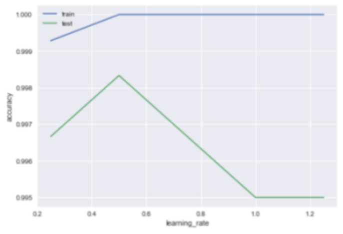

Figure 8: Search for learning rate

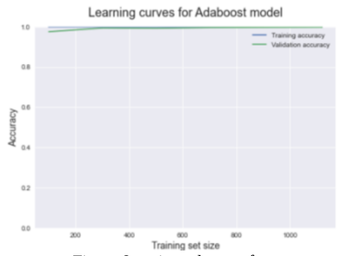

Figure 9: train and cv performance

**2. model explanation**

Boosting is a machine learning methodology that will highlight incorrect instances and try to correct them literally. Adaptive Boosting is an ensemble classification algorithm that combines with many weak learners to get more accurate results. Even though the individual learners may be weaker, overall group of weaker learners can be converted to a strong learner.

**3. model performance and confusion matrix**

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say 'success' is that when the testing set is predicted by the trained adaboost model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, Adaboost performs better if adaboost has much higher accuracy than others.

The best parameters are {'algorithm': 'SAMME', 'learning_rate': 0.5, 'n_estimators': 60}. The training accuracy of the decision tree model is 100% and testing accuracy is 99.9%. Overall, Adaboost wins the game.

## iv. Support Vector Machines

### 1. parameter tuning and pruning

SVM is based on the tree methods to provide good split points. the critic parameters are ‘decision_function_shape’ and ‘kernel’. As shown in Figure 10, the ‘kernel’ has four values – linear, poly, rbf, and sigmoid. Since the relationship between independent variable and dependent variable is linear, then kernel is linear. As shown in Figure 11, one-vs-one shapes (n_samples, n_classes * (n_classes - 1) / 2) and one-vs-rest shapes like (n_samples, n_classes) in the decision_function_shape. Based on the function, ovo closes to my dataset since my dataset is like (9, 2000). 5 folders cross-validation is applied for more accurate results. Figure 12 shows that training remains the same all the time, but validation starts going up and then stays the same. The reason has explained in decision tree section.

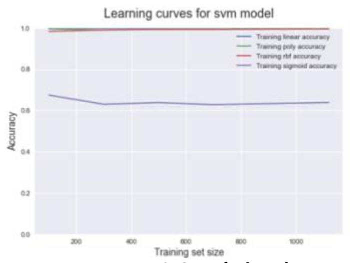

Figure 10: SVM for kernel

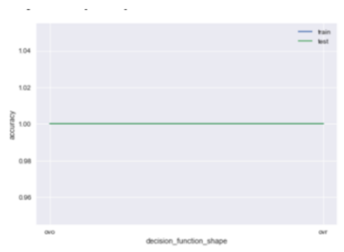

Figure 11: Search for decision_function_shape

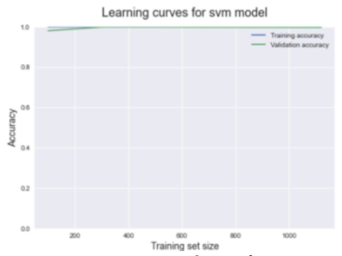

Figure 12: train and cv performance

### 2. model explanation

In machine learning, support vector machine is one of the supervised learning that maps a hyperplane to maximum the width of the gap between the two classes in solving classification analysis by analyzing training data. Whenever there are new examples, the algorithm will map them into that same space and provide the class based on where they fall.

### 3. model performance and confusion matrix

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say ‘success’ is that it should be fast to compute. Then when the testing set is predicted by the trained svm model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, svm performs better if svm has much higher accuracy than others.

The best parameters are {‘decision_function_shape’: ‘ovo’, ‘kernel’: ‘linear’}. The training accuracy of the decision tree model is 99.7% and testing accuracy is 99.7%.

## v. K-nearest neighbors

### 1. parameter tuning and pruning

The knn contains different methods to estimate new points based on the current known data points. The critic parameters are ‘n_neighbors’ and ‘weights’. As shown in Figure 13, the ‘weights’ has two values - ‘uniform’ and ‘distance’. Compared to accuracy results, ‘uniform’ will win this game by applying a 70% training set. I think the reason is that I normalized the original features and all features are same important to predict target. ‘uniform’ means that weight all points equally but distance means that closer neighbors will have greater weight. As shown in Figure 14, when ‘n_neighbors’ hits 10, knn accuracy on the training goes up but knn accuracy on the testing set starts dropping. In other words, the algorithm starts overfitting the problem. 5 folders cross-validation is applied for more accurate results.

Figure 15 shows that training and cross-validation accuracy goes up at the beginning but remains the same when data is large. Here are the reasons. When data set is small and n is not small, it will start learning some data points that should not be involve in this class. When data is large enough and n remains, the size of data will not affect so much since it only collects close enough data point.

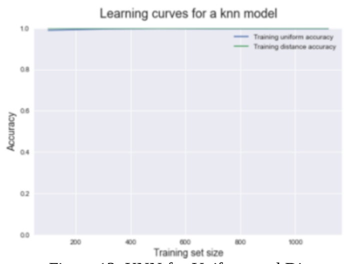

Figure 13: KNN for Uniform and Distance

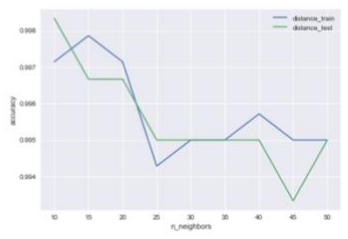

Figure 14: Search for n neighbors

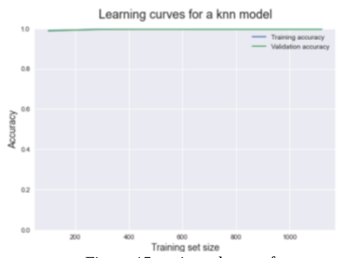

Figure 15: train and cv performance

### 2. model explanation

K-Nearest Neighbors is a type of instance-based learning (lazy learning) and a non-parametric classification method. An observation is classified by the number of k nearest neighbors using k times of voting. The output is a group of classes. Since this algorithm relies on approximate estimation if the features represent the same units of the training data, it can improve its accuracy.

### 3. model performance and confusion matrix

First sanity checklist is that right algorithm solves the right problem. Whenever the algorithm scope is right, then the fewer misclassification errors, the better model will perform. The first method to say ‘success’ is that when the testing set is predicted by the trained decision tree model, it still could make even, or slightly above performance compared to training set model performance. Compared with other algorithms, knn performs better if knn has much higher accuracy than others.

The best parameters are {‘n_neighbors’: 10, ‘weights’: ‘uniform’}. The training accuracy of the decision tree model is 99.7% and testing accuracy is 99.8%.