Supervised Learning Assignment

Jingxiong Liu

CS7641-Assignment 1

Abstract

In this assignment, five techniques in supervised learning will be explored (Decision Tree, Neural Networks, Boosting, Support Vector Machines, and KNN) on two different classification datasets. The difference among those algorithms will be analyzed and will be illustrated both in graphs and words.

1 Dataset introduction

1.1. The Breast Cancer Dataset

Medical staff took digitized images of a patient's breast mass following fine-needle aspiration (FNA) and extracted features from these digital images that describe the appearance of nuclei in the images. Tumors can be divided into benign and malignant.

The data table includes a total of 32 characteristics, first we need to load the data source. In the preparation stage, you need to explore the loaded data source and view the sample features and feature values. You can also use data visualization in this process, which can help us to further understand the data and the relationship between the data. Then, the quality of the data is evaluated according to the "complete unity" criterion. If the data quality is not high, data cleaning is required. After data cleaning, you can do feature selection to facilitate subsequent model training.

1.2. The Wine Classification Dataset

The dataset was observed on OpenML.org after some data cleaning techniques. The datasets obtain around 13 features with 3 classes of wine, which were produced in the same region.

The three wines include 13 different properties: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, Proline. In the data file, each row represents a sample of a wine, with a total of 178 samples; a total of 14 columns, of which the first column is the class flag attribute, there are three categories, marked as "1", "2", "3"; The following 13 columns are the sample values of the corresponding attributes of each sample. Among them, there are 59 samples in the first class, 71 samples in the second class, and 48 samples in the third class.

Since, there are some difference between the two datasets, especially the wine data is not a binary classification problem, some additional techniques are applied to analyze the data.

2 Pre-Processing of Dataset

For both datasets, we started from the data exploration.

Preparation stage: We first need to explore the data of the training set and test set, analyze the data quality, clean the data, and then reduce the dimension of the data through feature selection to facilitate subsequent classification operations.

Classification stage: First, the decision tree classifier is obtained through the feature matrix of the training set and the classification result, and then the classifier is applied to the test set. We then analyze the accuracy of the decision tree classifier and visualizet the decision tree model.

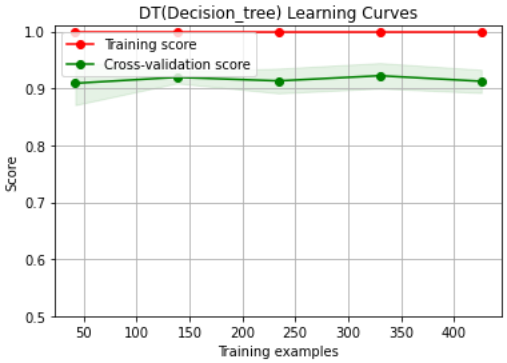
Although this part of data exploration has no substanial effect on the classifier, it cannot be ignored. Only when we know enough about the characteristics of these data can we help us do data cleaning and feature selection. So how to do data exploration? Here are some functions you need to know: use info() to understand the basic situation of the data table: number of

rows, columns, data type of each column, data integrity; use **describe()** to understand the statistics of the data table: total, average , standard deviation, minimum, maximum, etc.; use describe(include=['O']) to view the overall situation of string **types (non-numeric)**; use **head** to view the first few lines of data (the default is the first 5 lines); use **tail** looks at the last few lines of data (the default is the last 5 lines).

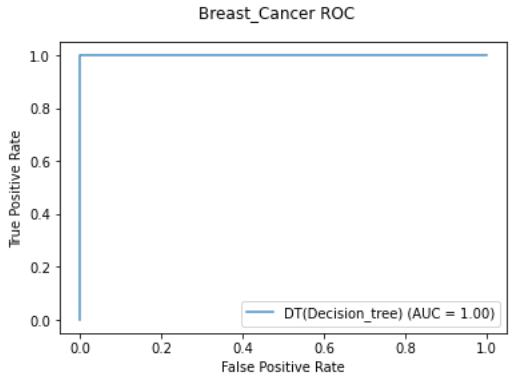
3 Decision Trees

3.1 The Breast Cancer Dataset

I first use the simple and default settings in the DecisionTreeClassifier (), the whole training process takes about 0.005699s and the learning curve illustrated below.



Also, the ROC curve is also printed out as follows.



This can show that for this cancer dataset, the Decision-Tree classifier will be a good classifier.

When I draw the learning curve, I realized that there is too much of bias and little variance comparing to the bias. This means the algorithm can't learn the data quite well.

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

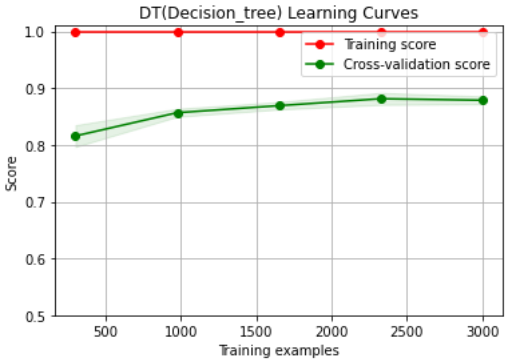
The final results illustrate that, Best Estimator: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None, 'splitter': 'random'}.

When compare with the first simple Decision-Tree classifier, we can find that there's slight improvement in the precision from 92.62% to 94.83%, and recall decrease from 95.67% to 93.22%, which can show that the second model after optimization performs better than the first simple one.

```
****************** DT(Decision_tree) ********************
DecisionTreeClassifier()
training took 0.005699s!
precision: 92.62%, recall: 95.76%
accuracy: 92.55%


training took 0.774900s!
DecisionTreeClassifier(min_samples_split=10, splitter='random')
precision: 94.83%, recall: 93.22%
accuracy: 92.55%
```

3.2 The Wine Dataset

Started with the simple Decision Tree model, the first accuracy score I get is 89.4% on the Wine Dataset.



```
****************** DT(Decision_tree) ********************
DecisionTreeClassifier()
training took 0.050836s!
precision: 89.43%, recall: 89.40%
accuracy: 89.40%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.88      0.92      0.90       315

   micro avg       0.88      0.92      0.90       315
   macro avg       0.44      0.46      0.45       315
weighted avg       0.88      0.92      0.90       315
```

The learning curve shows that the first decision tree gets a reasonable amount of bias and variance. To optimize, I made GridSearch's on both sides (more complex and less complex). After a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_impurity_split': None, 'min_samples_leaf': 11, 'min_samples_split': 3, 'min_weight_fraction_leaf': 0.0, 'presort': 'deprecated', 'random_state': None, 'splitter': 'best'}

Finally, we reach the DecisionTreeClassifier(min_samples_leaf=11, min_samples_split=3) model.

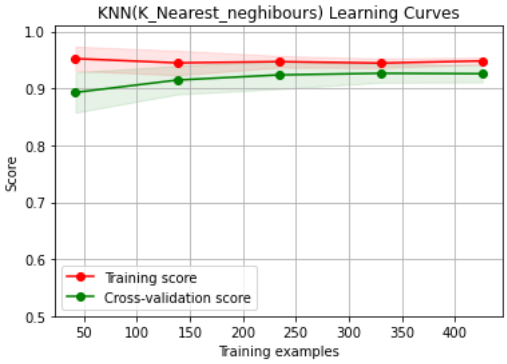
```
DecisionTreeClassifier(min_samples_leaf=11, min_samples_split=
precision: 90.62%, recall: 90.60%
accuracy: 90.60%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.90      0.92      0.91       315

   micro avg       0.90      0.92      0.91       315
   macro avg       0.45      0.46      0.45       315
weighted avg       0.90      0.92      0.91       315
```

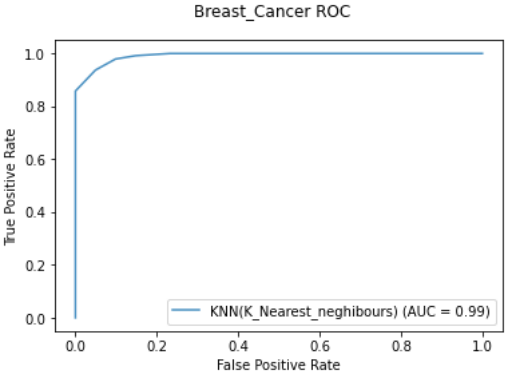
When compare with the first simple Decision-Tree classifier, we can find that the accuracy score is increasing from 89% to 90%, and there's slight decrease in the precision from 89.43% to 90.62%, and recall increase from 89.4% to 90.6%, which can show that the first model after optimization performs slightly better than the second simple one.

4 K-Nearest Neighbors

4.1 The Breast Cancer Dataset

Starting with the sklearn's KneighborsClassifier's original parameters, I got 92.02% accuracy score on cross-validation, which seems lower than all other algorithms. As I draw the learning curve for this classifier:






```
****************** KNN(K_Nearest_neghibours) *****************
KNeighborsClassifier()
training took 0.003030s!
precision: 91.20%, recall: 96.61%
accuracy: 92.02%

classification_report:
              precision    recall  f1-score   support

           0       0.94      0.84      0.89        70
           1       0.91      0.97      0.94       118

    accuracy                           0.92       188
   macro avg       0.92      0.90      0.91       188
weighted avg       0.92      0.92      0.92       188
```

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 7, 'p': 2, 'weights': 'distance'}

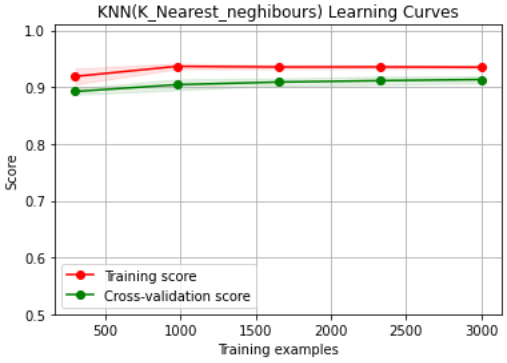
Finally, we reach the KNeighborsClassifier(n_neighbors=7, weights='distance') model.

When compare with the first simple K-Nearest Neighbors classifier, we can find that there's slight decrease in the precision from 91% to 90%, and recall increase from 96% to 97%, which can show that the first model after optimization performs slightly better than the second simple one.

```
precision: 90.55%, recall: 97.46%
accuracy: 92.02%

classification_report:
              precision    recall  f1-score   supp

           0       0.95      0.83      0.89
           1       0.91      0.97      0.94

    accuracy                           0.92
   macro avg       0.93      0.90      0.91
weighted avg       0.92      0.92      0.92
```

4.2 The Wine Dataset

Starting with the sklearn's KneighborsClassifier's original parameters, I got 92% accuracy score on cross-validation, which is much better comparing the UFC dataset. As I draw the learning curve for this classifier:




```
****************** KNN(K_Nearest_neghibours) ********************
KNeighborsClassifier()
training took 0.009449s!
precision: 92.04%, recall: 92.00%
accuracy: 92.00%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.91      0.94      0.92       315

   micro avg       0.91      0.94      0.92       315
   macro avg       0.45      0.47      0.46       315
weighted avg       0.91      0.94      0.92       315
```

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params': None, 'n_jobs': None, 'n_neighbors': 5, 'p': 2, 'weights': 'distance'}.

Finally, we reach the KNeighborsClassifier(weights='distance') model.

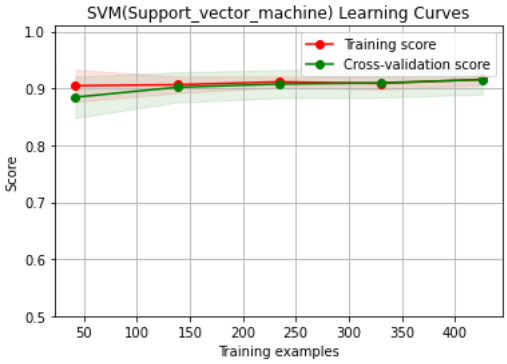
When compare with the first simple K-Nearest Neighbors classifier, we can find that the accuracy score increase from 92% to 92.1%, and there's slight increase in the precision from 90.55% to 92.04%, and recall increase from 97.46% to 92%, which can show that the second model after optimization performs slightly better than the first simple one.

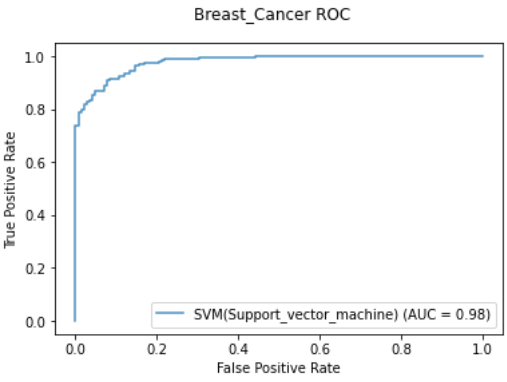
```
KNeighborsClassifier(weights='distance')
precision: 92.11%, recall: 92.10%
accuracy: 92.10%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.92      0.94      0.93       315

   micro avg       0.92      0.94      0.93       315
   macro avg       0.46      0.47      0.46       315
weighted avg       0.92      0.94      0.93       315
```

5 Support Vector Machines

5.1 The Breast Cancer Dataset

Starting with the sklearn's basic SVC with 'rbf' kernel and C=1.0, I have added class_weight: balanced parameter and got 90.96% f1_macro score on cross-validation. As I draw the learning curve for this classifier:





```
****************** SVM(Support_vector_machine) ***********
SVC(probability=True)
training took 0.012866s!
precision: 87.97%, recall: 99.15%
accuracy: 90.96%

classification_report:
              precision    recall  f1-score   support

           0       0.98      0.77      0.86        70
           1       0.88      0.99      0.93       118

    accuracy                           0.91       188
   macro avg       0.93      0.88      0.90       188
weighted avg       0.92      0.91      0.91       188
```

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'C': 10, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 0.0001, 'kernel': 'rbf', 'max_iter': -1, 'probability': True, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}

After we do the optimization, we find that the training time become longer from 0.01s to 0.14s.

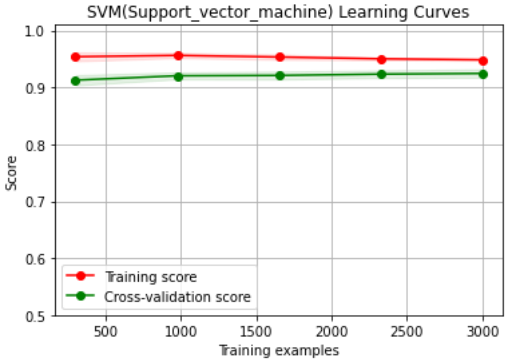
Finally, we reach the SVC (C=10, gamma=0.0001, probability=True) model.

When compare with the first simple Decision-Tree classifier, we can find that the accuracy increase from 90.96% to 92.02%, and there's slight increase in the precision from 87% to 91%, and recall decrease from 99% to 95%, which can show that the second model after optimization performs slightly better than the first simple one.

```
precision: 91.87%, recall: 95.76%
accuracy: 92.02%

classification_report:
              precision    recall  f1-score   su

           0       0.92      0.86      0.89
           1       0.92      0.96      0.94

    accuracy                           0.92
   macro avg       0.92      0.91      0.91
weighted avg       0.92      0.92      0.92
```

5.2 The Wine Dataset

Started with the sklearn's basic SVC, I got 0.92 f1_weighted score on cross-validation. As I draw the learning curve for this classifier:




```
****************** SVM(Support_vector_machine) ********************
SVC(probability=True)
training took 0.728184s!
precision: 93.12%, recall: 93.10%
accuracy: 93.10%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.92      0.94      0.93       315

   micro avg       0.92      0.94      0.93       315
   macro avg       0.46      0.47      0.46       315
weighted avg       0.92      0.94      0.93       315
```

Even though I get a good f1_macro score, the learning curve suggests that the SVM under-fits the dataset because there is no difference between the training and the validation scores. So, I needed to increase the complexity of the SVM algorithm.

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'C': 1, 'break_ties': False, 'cache_size': 200, 'class_weight': None, 'coef0': 0.0, 'decision_function_shape': 'ovr', 'degree': 3, 'gamma': 'scale', 'kernel': 'linear', 'max_iter': -1, 'probability': True, 'random_state': None, 'shrinking': True, 'tol': 0.001, 'verbose': False}.

Finally, we reach the SVC (C=1, kernel='linear', probability=True) model.

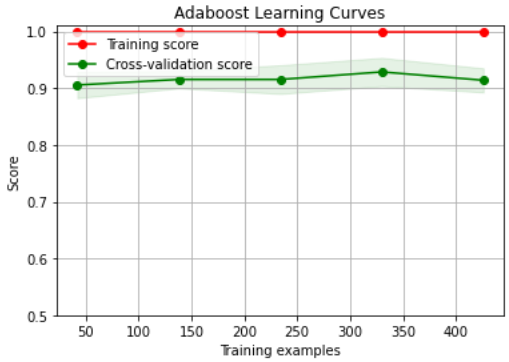
When compare with the first simple Support Vector Machine classifier, we can find that the accuracy score slightly not change, and there's slight decrease in the precision from 93.12% to 93.02%, and recall increase from 93.1% to 93.01%, which can show that the SVM model in the grid search not improve much in the GridSearch.

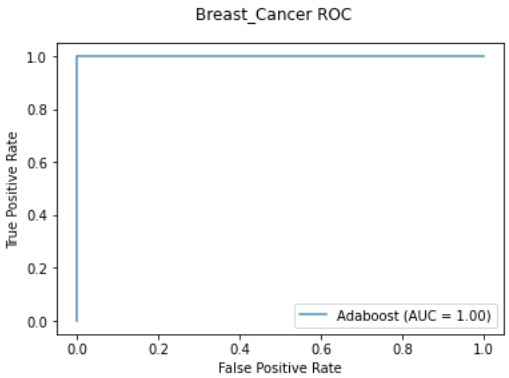
```
SVC(C=1, kernel='linear', probability=True)
precision: 93.02%, recall: 93.00%
accuracy: 93.00%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.92      0.95      0.93       315

   micro avg       0.92      0.95      0.93       315
   macro avg       0.46      0.47      0.47       315
weighted avg       0.92      0.95      0.93       315
```

6 BOOSTING

6.1 The Breast Cancer Dataset

In this section, I have used AdaBoost in sklearn library. Started with the sklearn's basic AdaBoostClassifier, boosting got 94.15% accuracy score on cross- validation. As I draw the learning curve for this classifier:






```
****************** Adaboost ******************
AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifie
          learning_rate=0.4, n_estimators=7)
training took 0.009270s!
precision: 95.73%, recall: 94.92%
accuracy: 94.15%

classification_report:
              precision    recall  f1-score   support

           0       0.92      0.93      0.92        70
           1       0.96      0.95      0.95       118

    accuracy                           0.94       188
   macro avg       0.94      0.94      0.94       188
weighted avg       0.94      0.94      0.94       188
```

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'algorithm': 'SAMME', 'base_estimator__ccp_alpha': 0.0, 'base_estimator__class_weight': None, 'base_estimator__criterion': 'gini', 'base_estimator__max_depth': None, 'base_estimator__max_features': None, 'base_estimator__max_leaf_nodes': None, 'base_estimator__min_impurity_decrease': 0.0, 'base_estimator__min_impurity_split': None, 'base_estimator__min_samples_leaf': 1, 'base_estimator__min_samples_split': 2, 'base_estimator__min_weight_fraction_leaf': 0.0, 'base_estimator__presort': 'deprecated', 'base_estimator__random_state': None, 'base_estimator__splitter': 'best', 'base_estimator': DecisionTreeClassifier(), 'learning_rate': 0.30000000000000004, 'n_estimators': 5, 'random_state': None}.

After we do the optimization, we find that the training time become longer from 0.009s to 0.39s.

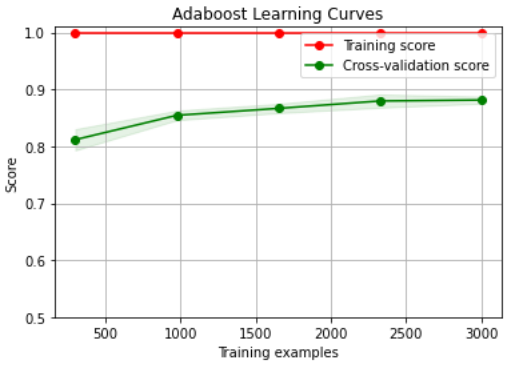
Finally, we reach the AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(), learning_rate=0.30000000000000004, n_estimators=5) model.

When compare with the first simple AdaBoosting classifier, we can find that the accuracy increase from 90.15% to 94.68%, and there's slight no change in the precision from 87% to 91%, and recall increase from 94.42% to 95.76%, which can show that the second model after optimization performs slightly better than the first simple one.

```
precision: 95.76%, recall: 95.76%
accuracy: 94.68%

classification_report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        70
           1       0.96      0.96      0.96       118

    accuracy                           0.95       188
   macro avg       0.94      0.94      0.94       188
weighted avg       0.95      0.95      0.95       188
```

6.2 The Wine Dataset

Started with the sklearn's basic AdaBoostClassifier, boosting got 89.6% f1_weighted score on cross- validation. As I draw the learning curve for this classifier:




```
****************** Adaboost ********************
AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionT
                   learning_rate=0.4, n_estimators=7)
training took 0.053189s!
precision: 89.67%, recall: 89.60%
accuracy: 89.60%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.89      0.92      0.91       315

   micro avg       0.89      0.92      0.91       315
   macro avg       0.44      0.46      0.45       315
weighted avg       0.89      0.92      0.91       315
```

It can be seen that there is really little variance in this order. So, I have gone into GridSearch to find ways to reduce the bias by choosing the best hyper- parameters in max_depth, splitter, n_estimator, and criterion parameters. When none of these parameters help me to reduce the bias, I have made a grid search with more n_estimators and lower learning_rate.

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'algorithm': 'SAMME', 'base_estimator__ccp_alpha': 0.0, 'base_estimator__class_weight': None, 'base_estimator__criterion': 'gini', 'base_estimator__max_depth': None, 'base_estimator__max_features': None, 'base_estimator__max_leaf_nodes': None, 'base_estimator__min_impurity_decrease': 0.0, 'base_estimator__min_impurity_split': None, 'base_estimator__min_samples_leaf': 1, 'base_estimator__min_samples_split': 2, 'base_estimator__min_weight_fraction_leaf': 0.0, 'base_estimator__presort': 'deprecated', 'base_estimator__random_state': None, 'base_estimator__splitter': 'best', 'base_estimator': DecisionTreeClassifier(), 'learning_rate': 0.5, 'n_estimators': 8, 'random_state': None}After we do the optimization, we find that the training time become longer from 0.05s to 0.8s.

Finally, we reach the MLPClassifier(hidden_layer_sizes=(30, 50), learning_rate='invscaling') model.

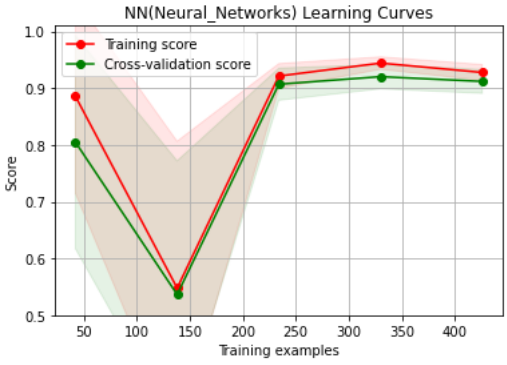
When compare with the first simple AdaBoosting classifier, we can find that the accuracy, Precision, and the Recall score perform slightly the same. So, there may be more space to improve in the parameter search.

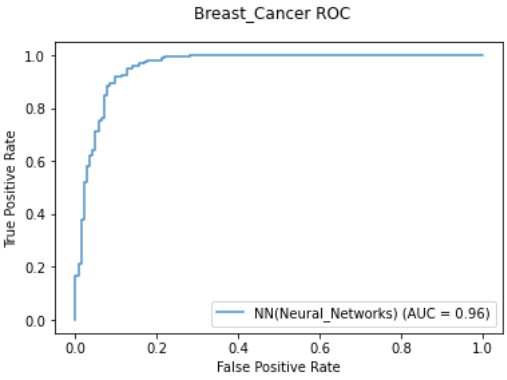
```
AdaBoostClassifier(algorithm='SAMME', base_estimator=DecisionTreeClassifier(),
                   learning_rate=0.5, n_estimators=8)
precision: 89.11%, recall: 89.00%
accuracy: 89.00%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.88      0.92      0.90       315

   micro avg       0.88      0.92      0.90       315
   macro avg       0.44      0.46      0.45       315
weighted avg       0.88      0.92      0.90       315
```

7 Neural Networks

7.1 The Breast Cancer Dataset

Started with the sklearn's basic MLPClassifier with one hidden layer of 100 nodes. It got a promising 90.96% accuracy score on cross-validation.





```
****************** NN(Neural_Networks) ******************
MLPClassifier()
training took 0.339857s!
precision: 91.74%, recall: 94.07%
accuracy: 90.96%

classification_report:
              precision    recall  f1-score   support

           0       0.90      0.86      0.88        70
           1       0.92      0.94      0.93       118

    accuracy                           0.91       188
   macro avg       0.91      0.90      0.90       188
weighted avg       0.91      0.91      0.91       188
```

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'activation': 'relu', 'alpha': 0.001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': (200, 400), 'learning_rate': 'constant', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}.

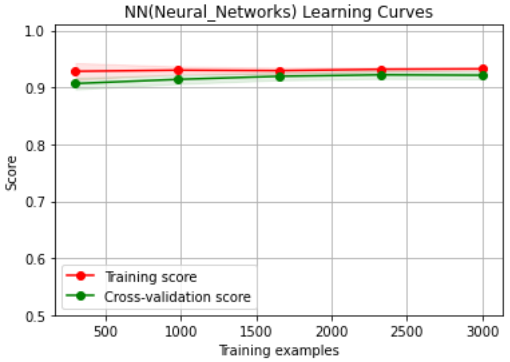
Finally, we reach the MLPClassifier(alpha=0.001, hidden_layer_sizes=(200, 400)) model.

When compare with the first simple Neural Networks classifier, we can find that the accuracy decrease from 90% to 89%, and there's slight increase in the precision from 91% to 87%, and recall increase from 94% to 97%, which can show that the second model after optimization performs slightly better than the first simple one.

```
precision: 87.79%, recall: 97.46%
accuracy: 89.89%

classification_report:
              precision    recall  f1-score

           0       0.95      0.77      0.85
           1       0.88      0.97      0.92

    accuracy                           0.90
   macro avg       0.91      0.87      0.89
weighted avg       0.90      0.90      0.90
```

7.2 The Wine Dataset

Started with the sklearn's basic MLPClassifier with one hidden layer of 100 nodes. It got a promising 93.2% promising score on cross-validation.




```
****************** NN(Neural_Networks) ******************
MLPClassifier()
training took 12.087928s!
precision: 93.24%, recall: 93.20%
accuracy: 93.20%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.91      0.94      0.93       315

   micro avg       0.91      0.94      0.93       315
   macro avg       0.46      0.47      0.46       315
weighted avg       0.91      0.94      0.93       315
```

The learning curve shows that the first neural network gets a reasonable amount of bias and variance. To optimize, I made GridSearch's on both sides (more complex and less complex). So, I have used 2-layer network, unlike Breast Cancer Dataset case. I have aimed to reduce the bias more as well as variance by setting other parameters such as *alpha, learning_rate*.

When I try to optimize the algorithms, I made a couple GridSearch's where I draw the learning curve in the end.

The final results illustrate that, Best Estimator: {'activation': 'relu', 'alpha': 0.0001, 'batch_size': 'auto', 'beta_1': 0.9, 'beta_2': 0.999, 'early_stopping': False, 'epsilon': 1e-08, 'hidden_layer_sizes': (30, 50), 'learning_rate': 'invscaling', 'learning_rate_init': 0.001, 'max_fun': 15000, 'max_iter': 200, 'momentum': 0.9, 'n_iter_no_change': 10, 'nesterovs_momentum': True, 'power_t': 0.5, 'random_state': None, 'shuffle': True, 'solver': 'adam', 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': False, 'warm_start': False}Finally, we reach the MLPClassifier(hidden_layer_sizes=(30, 50), learning_rate='invscaling') model.

When compare with the first simple Neural Networks classifier, we can find that the accuracy increases from 93.4% to 93.2%, and there's slight increase in the precision from 93.24% to 93.4%, and recall increase from 93.2% to 93.4%, which can show that the second model after optimization performs slightly better than the first simple one.

```
MLPClassifier(hidden_layer_sizes=(30, 50), learning_rate='invscaling')
precision: 93.41%, recall: 93.40%
accuracy: 93.40%

classification_report:
              precision    recall  f1-score   support

           0       0.00      0.00      0.00         0
           1       0.92      0.94      0.93       315

   micro avg       0.92      0.94      0.93       315
   macro avg       0.46      0.47      0.47       315
weighted avg       0.92      0.94      0.93       315
```

8 Running Time

As the clock time, the neural network and SVM classifiers took the most time during the training process and the KNN algorithm took the most time during predictions. One interesting result was this SVM takes a lot longer time on dataset with more features, whereas Neural Networks takes longer time at both because it has a lot of parameters and SVM's parameters depend on the features. Decision Trees took the least time because it can reach the result in log(m) time at most.

9 Appendix

The visualized Decision Trees is illustrated in another file named Vis_DT.