Supervised Learning Algorithm Comparison and Analysis

Yi Zhao

yzhao644@gatech.edu

Introduction

This is the supervised learning analysis report for homework 1. I obtained two data sets from UCI Machine Learning Repository, and implemented Decision Tree, Neural Network, Boosting SVM and KNN algorithms to build a machine learning model to solve two classification problems.

The process begin with analyzing the input data and some easy feature engineering. Then use cross validation and grid search to find the best hyper parameters. Third step is to draw a learning curve with the best hyper parameters. Last step is to train the model with all the training data and get a score on the testing data to measure the model.

The report is divided by the data set. Some future work is also brought up to make the model more accurate or generalized for all the data.

Data Set 1: Income level classifier

1. Data Description

The data is obtained from UCI Adult Income Dataset. The data is extracted from 1994 Census database. The task is to predict whether a person will make more than 50K a year.

The features include many geographic information, like age, race, education, origin country etc.

What I like about the data set is there are decent amount of data points, about 30k data points. For feature wise, it has a mix of categorical data and numerical data. The data is clean, most of the data points have valid data points, except a few question marks, which I will explain how to impute later.

The problem itself is a non-linear problem, makes it suitable for decision tree, boosting, neural network and KNN models. SVM can use a non-linear kernel for better result.

1.1 Data details

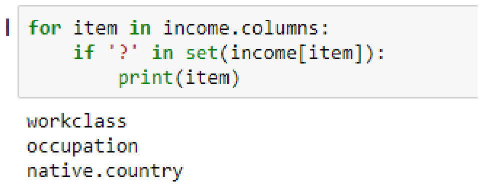
Data shape is 32561,15. There are 6 numerical data and 9 categorical data.

One feature called fnlwgt represents the weight of the row data. According to the document, it could be used for data selection. To make data simple, I decided to exclude this feature.

One education feature is duplicate to the education.num, which is a very good encoding for categorical data education level, since larger number means the person has a higher education.

1.2 Impute Method

There are no null values in the data set. But there are columns with question marks.



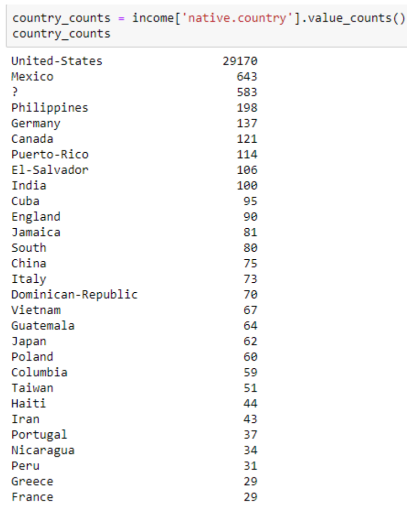
```
for item in income.columns:
    if '?' in set(income[item]):
        print(item)
```

```
workclass
occupation
native.country
```

For the first two columns, the I created another class as others for them. It might worth some analysis on whether they are highly correlated if we need to avoid multilinearity, like PCA method.

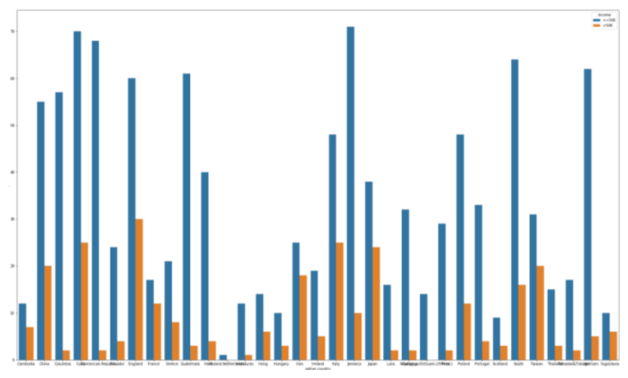
It turns out most of them are missing at the same time. There are seven rows left, where workclass is 'neverworked', I used same encoding for the occupation feature.

For the native.country feature, I checked distribution of the label to decide whether it's a good idea to combine the small portion countries and other class.



```
country_counts = income['native.country'].value_counts()
country_counts
```

```
United-States                 29170
Mexico                          643
?                               583
Philippines                     198
Germany                         137
Canada                          121
Puerto-Rico                     114
El-Salvador                     106
India                           100
Cuba                             95
England                          90
Jamaica                          81
South                            80
China                            75
Italy                            73
Dominican-Republic               70
Vietnam                          67
Guatemala                        64
Japan                            62
Poland                           60
Columbia                         59
Taiwan                           51
Haiti                            44
Iran                             43
Portugal                         37
Nicaragua                        34
Peru                             31
Greece                           29
France                           29
```



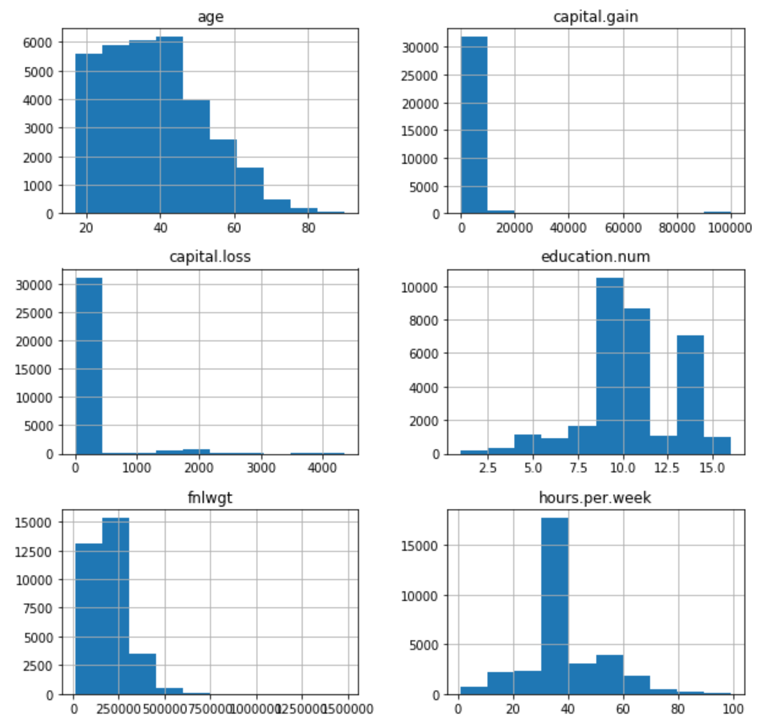
From the data, most of the data points are from states. i draw the below plot to check overall distribution of the labels.

The result shows that other class has about the same proportions of the positive labels and negative labels. So I decided to combine them as one large group of the other.

This will at the same time reduce the feature dimensions to avoid the curse of dimensionality.

1.3 Distribution Plot

For the numerical data, I plot the histogram to check the overall distribution of the features.



The distribution looks good in general. There are not much of outliers to work with.

One thing to notice is the scale of the variables, the capital numbers are in thousands level where ages and hours are tens.

I need to scale down the feature for the neural network to get good results. I will cover this in the later chapters.

1.4 One Hot Encoding and Train Test Split

The last step of data preparation is to change categorical data to numerical data for model training. Here I used one hot encoding method.

I also leave 20% of data as testing data and 80% for training.

2. Model Training details

For the model training session, I used scikit-learn library for all the algorithms, data preprocessing, data validation and plotting parts.

2.1 Data Pipeline

As mentioned earlier, I divide training data with 8:2 for training and testing. Testing data is set aside but never touched during training session. This is done by scikit-learn's train-test split method.

```
import numpy as np
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

I also created normalized data for future use. I tried min-max and standardized scaler and decided to go with standardized scaler for better training result.

```
from sklearn.preprocessing import StandardScaler  # doctest: +SKIP
scaler = StandardScaler()  # doctest: +SKIP
# Don't cheat - fit only on training data
scaler.fit(X_train)  # doctest: +SKIP
X_train_norm = scaler.transform(X_train)  # doctest: +SKIP
# apply same transformation to test data
X_test_norm = scaler.transform(X_test)  # doctest: +SKIP
```

Besides the algorithm part, I also use GridSearch, cross-validation, learning curve to compare results to find the best hyper-parameters and find the best model to predict the results.

2.2 Decision Tree

For decision tree, I used DecisionTreeClassifier from scikit-learn library with Gini Index as default parameter to measure the quality of a split.

```
estimator = DecisionTreeClassifier(random_state=0
                                   ,max_depth=20
                                      ,max_features=0.8,
                                     min_samples_leaf=20)
```

I started to use a model with default parameters to get a sense of the data.

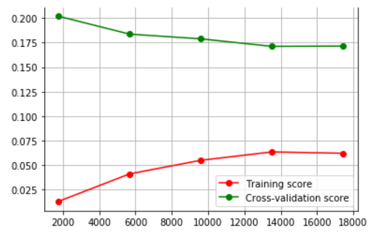
I used 10 folds validation for the measurement. All default parameters, which means the score is accuracy.

```
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(random_state=0)
cross_val_score(clf, X_train,y_train, cv=10)
```

```
array([0.82722273, 0.82218148, 0.80064161, 0.8304308 , 0.81989001,
       0.81843191, 0.81613939, 0.81889042, 0.82026593, 0.81522238])
```

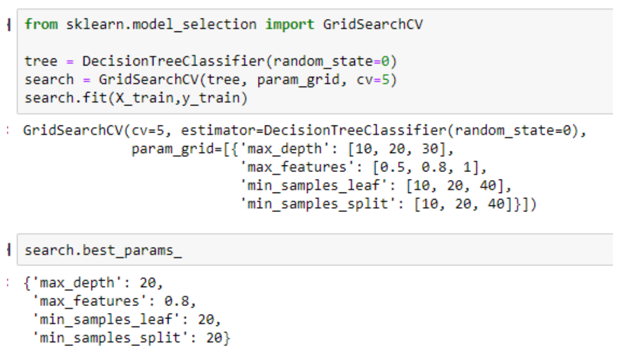
As we learnt from the previous knowledge, decision tree is prone to be overfitting with high variance.

I plot a learning curve to check it out.



The two curves are not converging at the end and validation error is higher than the training error.

Next, I am using pre-tuning to tune the decision tree. The hyper-parameters are tree depth, max feature proportion and min samples leaf size. I used grid search to determine the best parameters.



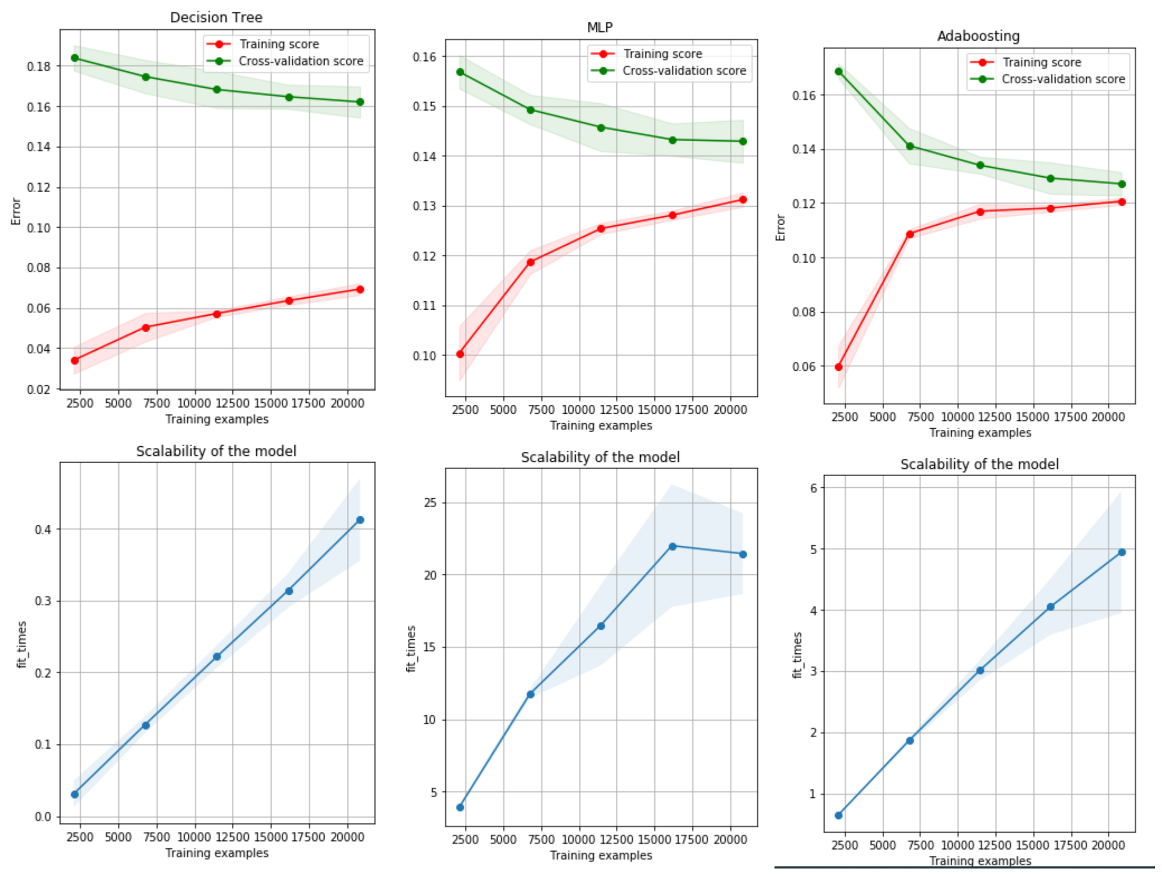
```
from sklearn.model_selection import GridSearchCV

tree = DecisionTreeClassifier(random_state=0)
search = GridSearchCV(tree, param_grid, cv=5)
search.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=0),
             param_grid=[{'max_depth': [10, 20, 30],
                          'max_features': [0.5, 0.8, 1],
                          'min_samples_leaf': [10, 20, 40],
                          'min_samples_split': [10, 20, 40]}])
```

```
search.best_params_
```

```
{'max_depth': 20,
 'max_features': 0.8,
 'min_samples_leaf': 20,
 'min_samples_split': 20}
```

The learning curve with the best param is as below.

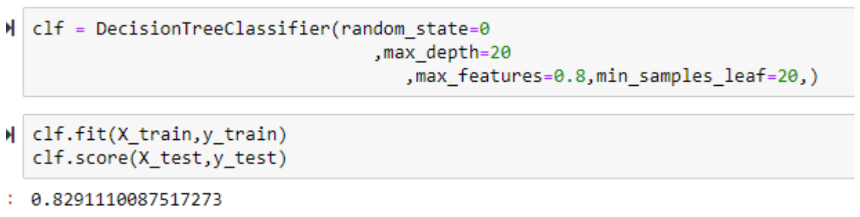


The learning curve is calculated with scikit-learns internal function learning_curve, which returns the accuracy from cross validation for training and testing data, together with training times. I transferred the accuracy to error by take 1-accuracy. I calculated the mean and standard deviation of the error to plot the learning curve. The shaded area is the one standard deviation with the mean value.

We can see from the plot that the error is improving from the initial default parameters (from 0.175 to 0.16). But still training and testing are not converging to the end, meaning either we need more training data or the model has high variance issue.

Another finding is that the fitting time is very fast and all below 1 second.

Final score of the model is as below.

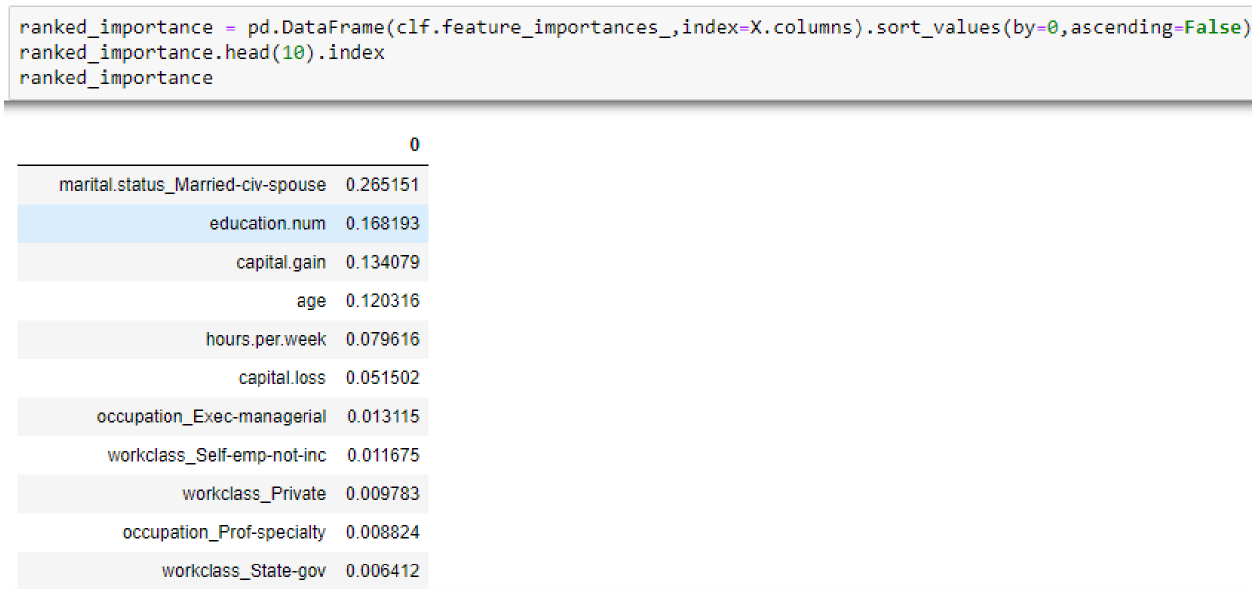


```
clf = DecisionTreeClassifier(random_state=0
                             ,max_depth=20
                                ,max_features=0.8,min_samples_leaf=20,)
```

```
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
```

```
: 0.8291110087517273
```

Another important information provided by decision tree is feature importance.



```
ranked_importance = pd.DataFrame(clf.feature_importances_,index=X.columns).sort_values(by=0,ascending=False)
ranked_importance.head(10).index
ranked_importance
```

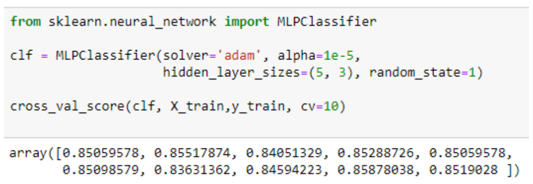
	0
marital.status_Married-civ-spouse	0.265151
education.num	0.168193
capital.gain	0.134079
age	0.120316
hours.per.week	0.079616
capital.loss	0.051502
occupation_Exec-managerial	0.013115
workclass_Self-emp-not-inc	0.011675
workclass_Private	0.009783
occupation_Prof-specialty	0.008824
workclass_State-gov	0.006412

An important finding here is that numerical features rank very high here, we will bring this back when I talk about KNN model later.

2.3 Neural Networks

For neural networks, I used scikit-learn's MLPClassifier. For neural networks, they usually work great with lot's of data. Here we have about 30k data points. It's not as many in terms of deep learning. So we might want to limit the parameter size. The activation function is RELU.

I also used default parameters and cross validation for the check.

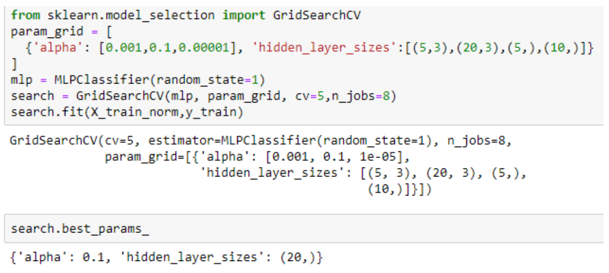


```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(solver='adam', alpha=1e-5,
                    hidden_layer_sizes=(5, 3), random_state=1)

cross_val_score(clf, X_train,y_train, cv=10)
```

```
array([0.85059578, 0.85517874, 0.84051329, 0.85288726, 0.85059578,
       0.85098579, 0.83631362, 0.84594223, 0.85878038, 0.8519028 ])
```

As we can see the default parameters perform pretty good off the bat.

I also used grid search for the best parameters. Another thing to notice is that when I normalize the data, the learning curve is smoother than original data.



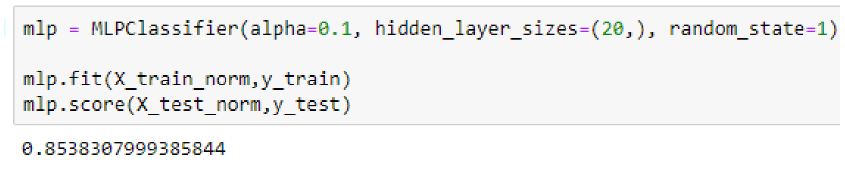
```
from sklearn.model_selection import GridSearchCV
param_grid = [
  {'alpha': [0.001,0.1,0.00001], 'hidden_layer_sizes':[(5,3),(20,3),(5,),(10,)]}
]
mlp = MLPClassifier(random_state=1)
search = GridSearchCV(mlp, param_grid, cv=5,n_jobs=8)
search.fit(X_train_norm,y_train)
```

```
GridSearchCV(cv=5, estimator=MLPClassifier(random_state=1), n_jobs=8,
             param_grid=[{'alpha': [0.001, 0.1, 1e-05],
                          'hidden_layer_sizes': [(5, 3), (20, 3), (5,),
                                                 (10,)]}])
```

```
search.best_params_
```

```
{'alpha': 0.1, 'hidden_layer_sizes': (20,)}
```

Final result is as below.



```
mlp = MLPClassifier(alpha=0.1, hidden_layer_sizes=(20,), random_state=1)

mlp.fit(X_train_norm,y_train)
mlp.score(X_test_norm,y_test)
```

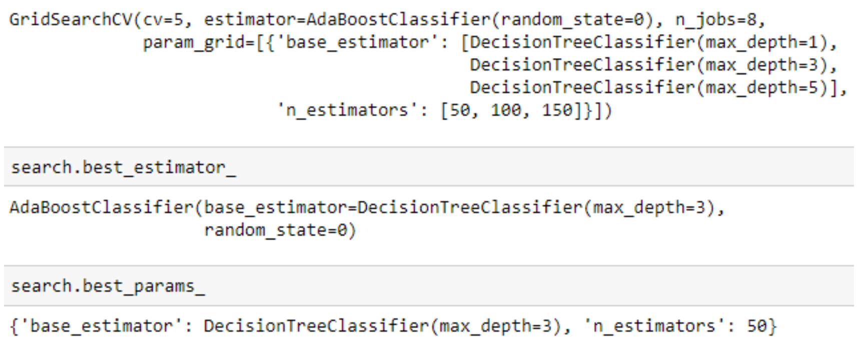
```
0.8538307999385844
```

2.4 Boosting

For boosting models, I tried both Gradient Boosting and Ada-Boosting algorithms. They both achieve similar results. I'll focus on the Ada-Boosting.

For hyper-parameter here, I tried different base estimators (trees with different max depth) and number of estimators. As suggested by the question, I went aggressive with pruning by keeping the depth up to 3.

Here's my grid search result.



```
GridSearchCV(cv=5, estimator=AdaBoostClassifier(random_state=0), n_jobs=8,
             param_grid=[{'base_estimator': [DecisionTreeClassifier(max_depth=1),
                                             DecisionTreeClassifier(max_depth=3),
                                             DecisionTreeClassifier(max_depth=5)],
                          'n_estimators': [50, 100, 150]}])
```

```
search.best_estimator_
```

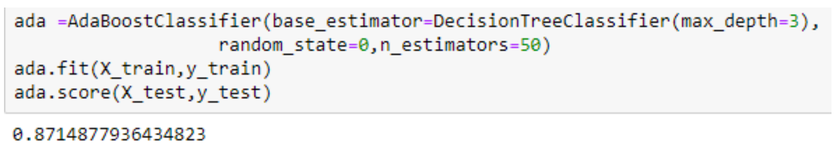
```
AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                   random_state=0)
```

```
search.best_params_
```

```
{'base_estimator': DecisionTreeClassifier(max_depth=3), 'n_estimators': 50}
```

Learning curve also shows better convergence than decision tree and neural networks.

The final test result also by far the best one.



```
ada =AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                        random_state=0,n_estimators=50)
ada.fit(X_train,y_train)
ada.score(X_test,y_test)
```

```
0.8714877936434823
```

2.5 SVM

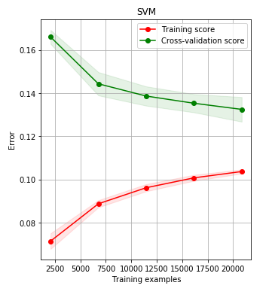
For SVM, my machine has a very slow training time, so instead of using grid search method, I use cross validation to compare linear and rbf kernels, parameter C and gamma ('auto' and fix number).

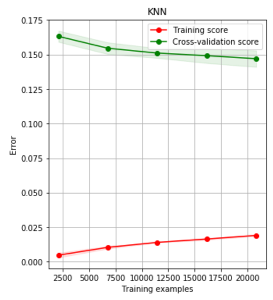
Details could be found in the notebook.

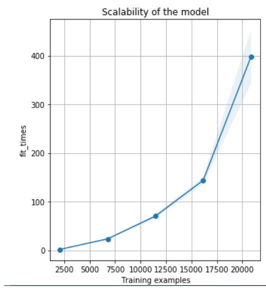
My final decision is as below.

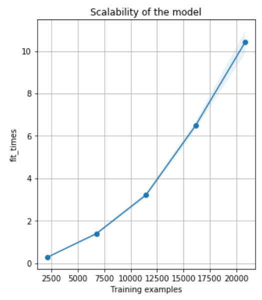
sv = svm.SVC(gamma='auto',C=10,kernel = ‘rbf’)

my learning curve is as below. The training time goes up to 400 seconds.









The result shows better than MLP, Decision tree. But the training time is way higher than above two.

```
sv.fit(X_train,y_train)
sv.score(X_test,y_test)
```

```
0.8651926915399969
```

2.6 KNN

KNN is an instance-based learning. There are no parameters needed to be trained to predict the result. There are hyper-parameters like K and weight method to choose from. It also usually requires a lot of training data for better result.

I also used grid search to find my best hyper-parameters. The learning curve is shown as above. It’s not converging well as other and the final validation result is also not as good as other algorithms.

```
knn = KNeighborsClassifier(n_neighbors=50,weights='distance')
knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

```
0.8481498541378781
```

3. Result and Reasoning

For test scores, Ada-Boosting has the best score with 0.87. It also converges really well between training and testing error. Decision tree has the lowest training result due to overfitting.

For the training time, decision tree has the best training time while svm is the slowest. Other algorithms range between 5-20 seconds.

I think it makes sense to me for Ada-Boosting to perform well, since it's an ensemble method that could take advantage of decision trees and also avoid overfitting.

Neural network is also very promising. It might perform better if proving more data.

4. Imbalanced Labels

For this data set, we have data imbalance issue. We have about 3 time the positive data than the negative data.

I decided not to up sampling or down sampling the data because the imbalance with not very bad.

There are also other metrics to measure imbalanced data set, for example confusion matrix or AUC-ROC score. I decided to use the accuracy for simplicity with scikit learn model and easy to understand. But I would explore other options in the future.

Data Set 2: UCI wine quality data set

1. Data Description

This data set also comes from the UCI data set. These data are the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines.

The goal is to predict the quality of the wine by providing a score ranging from 3 to 8.

The features include chemical levels of ingredients from the wine, like alkalinity of ash and alcohol level.

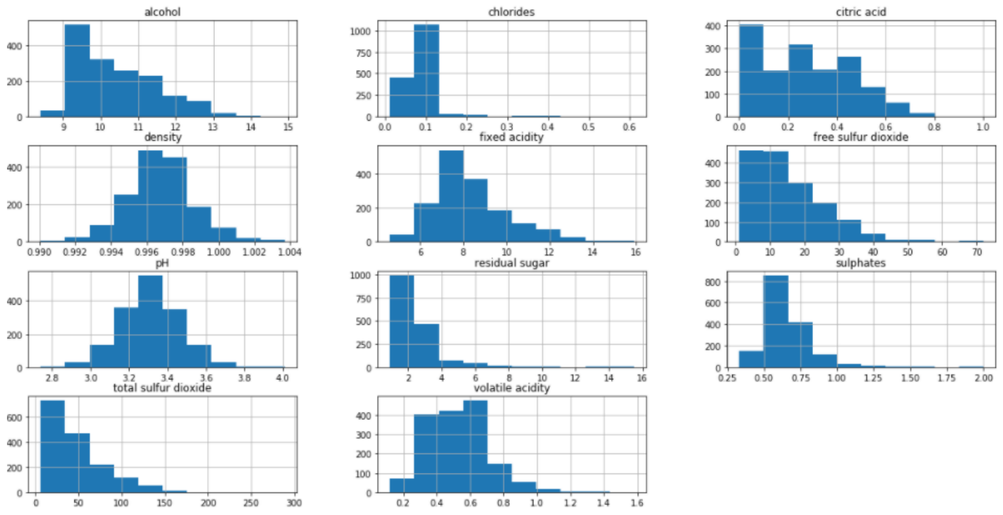
What I liked about this data set is that it's not very complicated. All the features are numerical data. And the data size is not as large as income data set, so I can do more cross validation, hyper parameter tuning without waiting too long.

Another difference from the income data set is that it's actually a multiple class instead of binary class. I hope it might bring in difference in model performance.

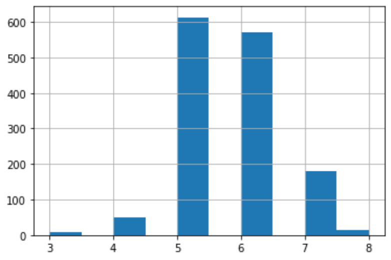
1.1 Exploratory data analysis

As mentioned above, all the features are numerical data and there are not null values to take care of.

Here are the hist plot for the features.



There are two challenges with the data set. First is the labels are very unbalanced.



Second is there are not many data points available for training. I solved these problems by up sampling the minority class with scikit-learn's resample method. As a result, all the class has same size of 638.

I also use scaler to normalize both training and test data. The scaler was fit only by training data to avoid the data leakage.

2. Model Training and Comparison

I used the similar approach as first data set. Use cross validation and grid search to find the best hyper parameters for the model and plot the learning curve. Use the training data to train the model and validate with the testing data that was set aside in the beginning, which was not up-sampled.

I will skip the training and hyper-parameter trainings for data set 2.

The accuracy of different models for testing data of the wine data is as follows.

Decision Tree: 0.8875

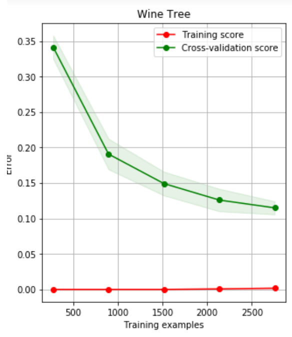
Neural Network: 0.90

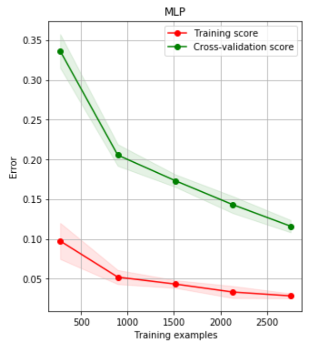
SVM: 0.9033

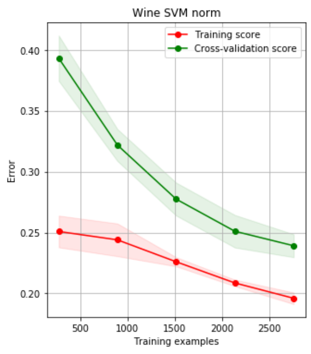
Boosting: 0.3603

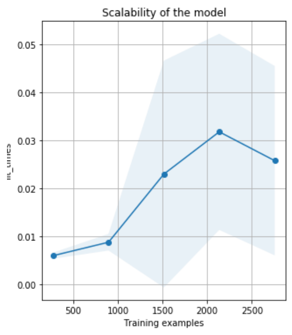
KNN: 0.8695

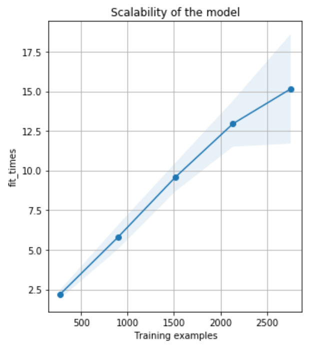
Below are the learning curves for comparison.

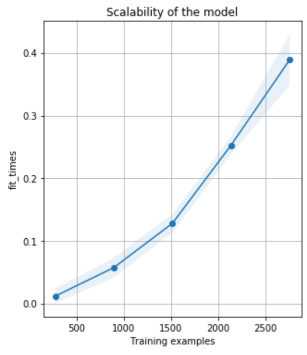


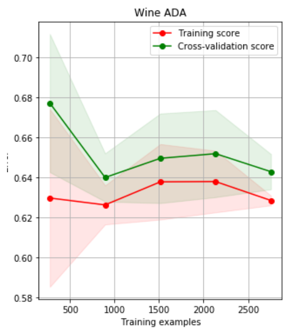


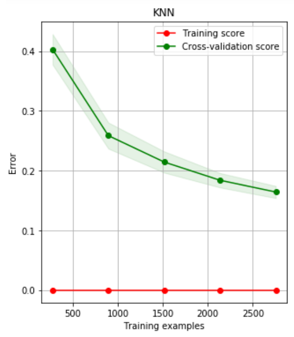


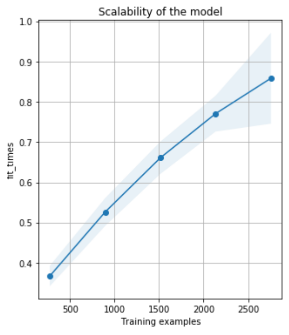


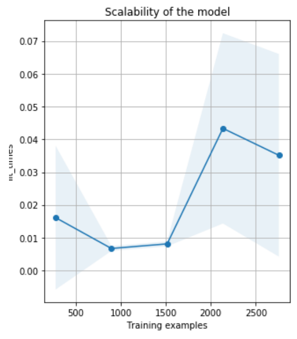












One of the biggest differences between data set 1 is that the Ada-Boosting is no longer perform well. It is actually the worst in terms of accuracy. This is because in the up sampling process, we only duplicate the data points. It doesn't help Ada-Boosting to emphasize the errors.

Another finding is that decision tree is performing considerably well on wine data set. My interpretation of this is that wine data set has much less features than income data set, which makes it harder for the tree to grow and easier for the algorithm to generalize the data.

SVM and Neural network are performing really well as previous findings.

KNN has some change in training time when example increase. One reason could be there are more duplicate points when the sample size increase and those decrease the total training time when calculating distance between target and training samples.

Another big difference is except KNN and Decision tree, all the training errors are still decreasing. This could due to we don't have a lot of training data for the model to learn well.

Summary

Decision trees are easy to overfit. Pruning can help with the overfitting but will not avoid it with limit data. And the accuracy will not be as high compared to other examples. One exception is that if the features are limited, then the tree won't be as deep so that overfit doesn't happen as much compared to deeper trees.

Ada-Boosting works great in terms of balance of variance and bias and generally doesn't overfit. But it works really bad when training data are duplicated for upsampling.

SVM and Neural Networks usually works well with proper hyper-parameter training. But training time increases a lot compared to Decision Tree and Boosting, especially SVM.

KNN usually comes with high variance problem but the accuracy is not bad with testing data.

Reference

1. UCI adult income data set http://archive.ics.uci.edu/ml/datasets/Adult

2. UCI wine quality data set https://archive.ics.uci.edu/ml/datasets/wine

3. Scikit-learn library https://scikit-learn.org/stable/