CS273A Machine Learning Project Report

Shijing Liu, Kai Ye, Lingxi Meng

December 15, 2017

1 Project Introduction

To train the data, we tried lots of models, including KNN, ensemble, decision trees, neural networks and boosting. KNN and decision tree turned out to be the best algorithms for this project. The best results on Kaggle was the latest submission (Kai Ye, $Y_submit_prob.txt$, 0.79875) which was implemented by gradient boosting decision tree, which will be introduced in the next section.

2 First method: Gradient Boosting Decision Tree

```
import xgboost as xgb
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import mltools as ml
from sklearn.metrics import accuracy_score
from sklearn import linear_model
from sklearn.metrics import roc_curve, auc
from sklearn import cross_validation, metrics  #Additional scklearn functions
from sklearn.grid_search import GridSearchCV   #Perforing grid search
from sklearn.feature_selection import SelectFromModel

def AUC_Model(Yva, Yte):
    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(2):
        fpr[i], tpr[i], _ = roc_curve(Yva, Yte)
        roc_auc[i] = auc(fpr[i], tpr[i])

    fpr["micro"], tpr["micro"], _ = roc_curve(Yva.ravel(), Yte.ravel())
    roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])

    print("AUC=",roc_auc[1])
    return roc_auc[1]

Xtr = np.loadtxt("data/Xtrain.txt") # 5 entities, each contains 10 features
Ytr = np.loadtxt("data/Ytrain.txt") # binary target
```

```
Xte = np.loadtxt("data/Xtest.txt")
print("data loaded!")
Xtr, Xva, Ytr, Yva = ml.splitData(Xtr, Ytr, train_fraction=0.75)
```

2.1 Try the feature selection

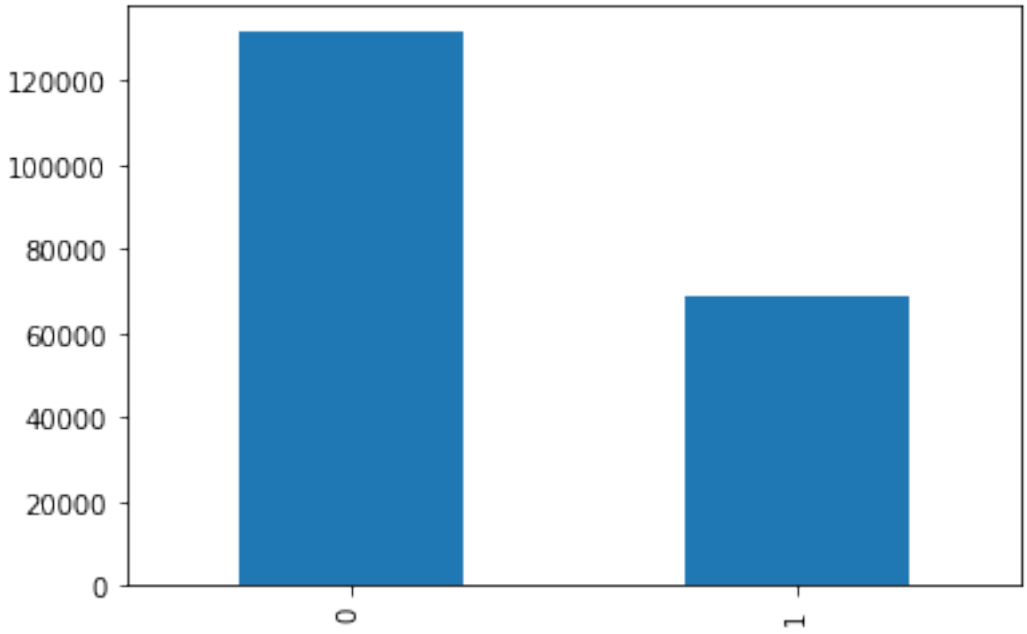
2.1.1 Original Features

```
import pandas as pd

data = pd.read_csv("data/Xtrain.txt", sep=" ", header=None)
data1 = pd.read_csv("data/Ytrain.txt", sep=" ", header=None)
# Xte = np.loadtxt("data/Xtest.txt")
print("data loaded!")

print(data.shape, data1.shape)
# data = pd.read_csv("data/X_train.txt", sep=" ", header=None)
# data1 = pd.read_csv("data/Y_train.txt", sep = " ", header = None)
train = pd.concat([data1,data],axis=1)
```

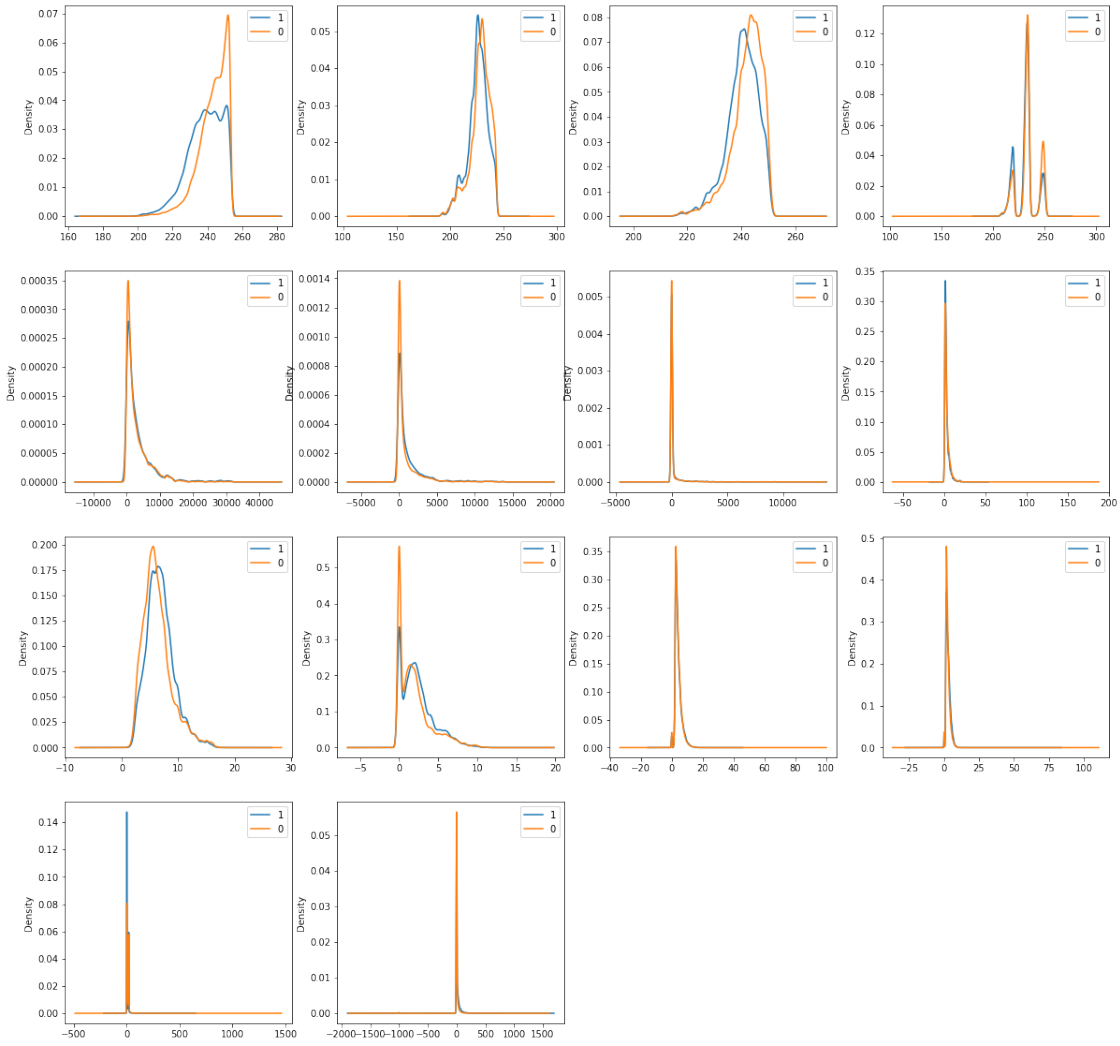
```
data loaded!
(200000, 14) (200000, 1)
```

From the printed shape above, we know we have 200000 data and 14 features at the beginning. data1 represents the label we give to the specific data point. As we are predicting whether it will rain, so 1 means it will rain, 0 means it will not. So we are curious about the distribution of Y_train.

```
import matplotlFrom the bar chart above, we know that about 7000 labels are 1, about 13

train.label.value_counts().plot(kind='bar')
plt.figure(figsize=(5,3))
plt.show()
```

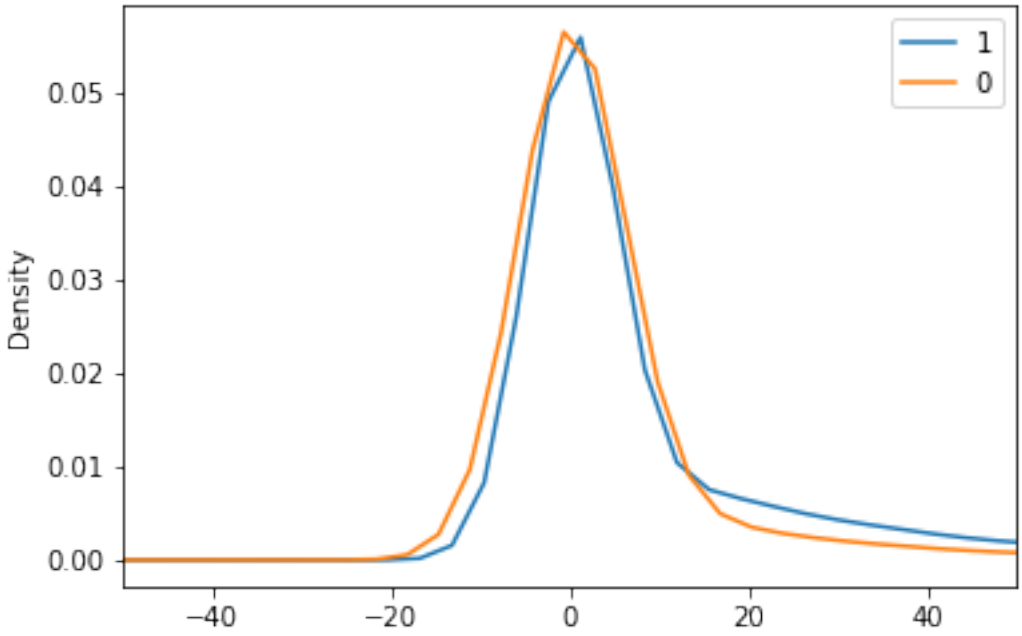


From the bar chart above, we know that about 7000 labels are 1, about 13000 labels are 0. This means the probability to rain is smaller, which is quite normal in our real life. After analyzing the labels, we want to know more about the features. Because features are important for our model and the prediction. However, we do not know the specific meaning of each feature, we only know that these features are related to the infrared satellite image information, so we use 14 subplots to draw the range and the density of each feature to see whether the feature is useful.



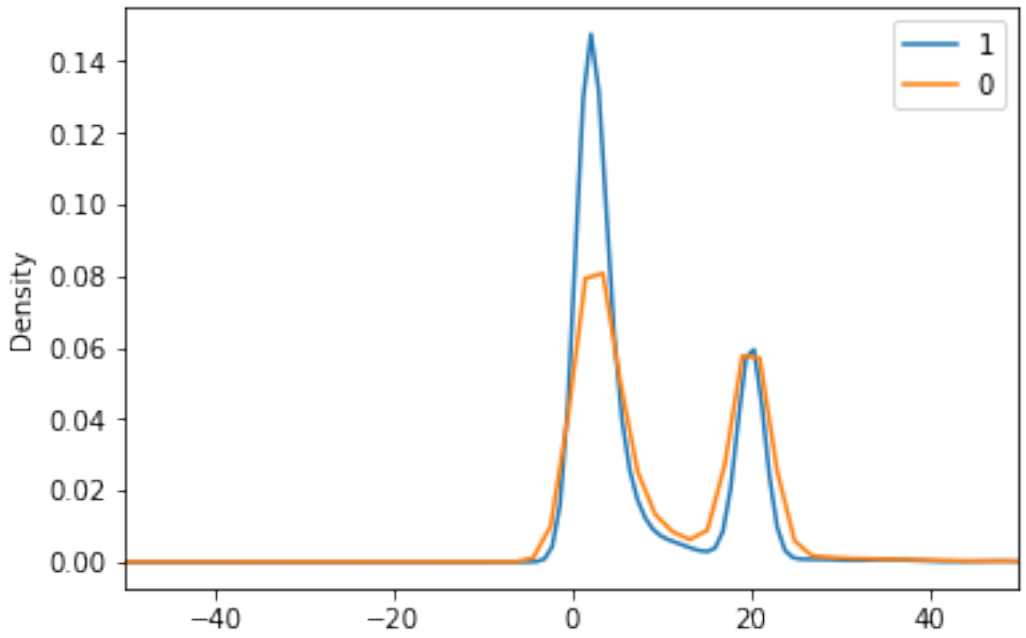
So from the subplots above we know that both the range and the density vary from feature to feature. For example, the range of feature 5 is -10000 to 40000 which is huge, while for feature 10, the range is -5 to 20. Also some feature only appears at a limited range such as feature 14, the value is around 0 only.

```
train.f14[train.label == 1].plot(kind='kde')
train.f14[train.label == 0].plot(kind='kde')
plt.legend(('1', '0'),loc='best')
plt.xlim(-50,50)
plt.show()
```



The density of 0 and 1 almost overlapped, which means in the range of around [-20,20], this feature almost make no contribution to the 0-1 decision. As for other range, it stays 0, no contribution at all.

```
train.f13[train.label == 1].plot(kind='kde')
train.f13[train.label == 0].plot(kind='kde')
plt.legend(('1', '0'),loc='best')
plt.xlim(-50,50)
plt.show()
```




```
model = xgb.XGBClassifier(max_depth=15, max_features=8)
model.fit(Xtr, Ytr)
Yte = model.predict(Xva)
AUC_Model(Yva, Yte)
```

```
AUC= 0.674919355976
```

2.1.2 Selected Features

```
model = xgb.XGBClassifier(max_depth=15, max_features=8)
model.fit(Xtr, Ytr)
print(model.feature_importances_)
# Yte = np.array(clf.predict_proba(Xte));
trmodel = SelectFromModel(model, prefit=True)
Xtr_new = trmodel.transform(Xtr)
Xva_new = trmodel.transform(Xva)
model = xgb.XGBClassifier(max_depth=15)
model.fit(Xtr_new, Ytr)
print(model.feature_importances_)
Yte = model.predict(Xva_new)
AUC_Model(Yva, Yte)
```

```
[ 0.15621923  0.0459085   0.08954261  0.05429314  0.09022153  0.04151589
  0.01321176  0.09277426  0.07746465  0.04698798  0.08456614  0.08693556
  0.05883511  0.06152363]
[ 0.19497782  0.14802194  0.14508258  0.13386738  0.13057677  0.12374908
```

```
  0.12372442]
AUC= 0.675815429908
```

```
Erange = range(100,301,40)
AUC=[]
for e in Erange:
    model = xgb.XGBClassifier(learning_rate=0.1, n_estimators = e, max_features='sqrt',
                              subsample=0.8, random_state=10)
    model.fit(Xtr, Ytr)
    Yte = model.predict(Xva);
    print("n_estimator: ", e)
#     print(model.score)
    AUC.append(AUC_Model(Yva, Yte));
```

```
n_estimator:  100       AUC= 0.606025924301
n_estimator:  140       AUC= 0.60964249555
n_estimator:  180       AUC= 0.612103580412
n_estimator:  220       AUC= 0.614213302058
n_estimator:  260       AUC= 0.616606680113
n_estimator:  300       AUC= 0.619271567724
```

The auc result keeps increasing as n_estimator increase.

2.2 Tuning for max_depth & min_child_weight

Considering the test efficiency, when comes to the max_depth and min_child_weight, the n_estimator has been set as 80.

For min_child_weight, A smaller value is chosen because it is a highly imbalanced class problem and leaf nodes can have smaller size groups.

2.2.1 max_depth 5-20 & min_child_weight 1-5

```
D = range(5,25,5) # Or something else
W = range(1,6,2) # Or something else
# Tr_auc1 = np.zeros((len(D),len(F)))
Va_auc1 = np.zeros((len(D),len(W)))

for i,d in enumerate(D):
    for j,w in enumerate(W):
        learner = xgb.XGBClassifier(learning_rate=0.1,
                                    n_estimators=100, max_depth=d, min_child_weight=w, subsampl
        learner.fit(Xtr, Ytr)
        Yte = learner.predict(Xva)
#         Tr_auc1[i][j] = learner.auc(XtS, Yt)
        print("max_depth: ", d, 'min_child_weight=', w)
        Va_auc1[i][j] = AUC_Model(Yva, Yte)
```

```
max_depth:  5       min_child_weight= 1      AUC= 0.619414508265
max_depth:  5       min_child_weight= 3      AUC= 0.619946844684
max_depth:  5       min_child_weight= 5      AUC= 0.619452036288
max_depth:  10      min_child_weight= 1      AUC= 0.656778871473
max_depth:  10      min_child_weight= 3      AUC= 0.656880040767
max_depth:  10      min_child_weight= 5      AUC= 0.654458175531
max_depth:  15      min_child_weight= 1      AUC= 0.682348269657
max_depth:  15      min_child_weight= 3      AUC= 0.677579495723
max_depth:  15      min_child_weight= 5      AUC= 0.675817704334
max_depth:  20      min_child_weight= 1      AUC= 0.68589766006
max_depth:  20      min_child_weight= 3      AUC= 0.683231159028
max_depth:  20      min_child_weight= 5      AUC= 0.683944241206
```

2.2.2 max_depth 25-30 & min_child_weight 1-7

```
D = range(25,31,5) # Or something else
W = range(1,9,2) # Or something else
# Tr_auc1 = np.zeros((len(D),len(F)))
Va_auc2 = np.zeros((len(D),len(W)))

for i,d in enumerate(D):
    for j,w in enumerate(W):
        learner = xgb.XGBClassifier(learning_rate=0.1,
                                    n_estimators=100, max_depth=d, min_child_weight=w, subsampl
        learner.fit(Xtr, Ytr)
        Yte = learner.predict(Xva)
#         Tr_auc1[i][j] = learner.auc(XtS, Yt)
        print("max_depth: ", d, 'min_child_weight=', w)
        Va_auc2[i][j] = AUC_Model(Yva, Yte)
```

```
max_depth:  25 min_child_weight= 1      AUC= 0.684525968197
max_depth:  25 min_child_weight= 3      AUC= 0.687594978591
max_depth:  25 min_child_weight= 5      AUC= 0.686254581404
max_depth:  25 min_child_weight= 7      AUC= 0.685597414558
max_depth:  30 min_child_weight= 1      AUC= 0.685590967983
max_depth:  30 min_child_weight= 3      AUC= 0.685553290701
max_depth:  30 min_child_weight= 5      AUC= 0.688459409577
max_depth:  30 min_child_weight= 7      AUC= 0.687069159824
```

The max_depth should be 30 and the min_child_weight should be 5.

2.3 Tuning for gamma

```
grange = [i/10.0 for i in range(0,5)]
AUC1 = []
for g in grange:
    learner = xgb.XGBClassifier(learning_rate =0.1, n_estimators=100, max_depth=30,
```

```
                        min_child_weight=5, gamma=g, subsample=0.8, colsample_bytree=0.8,
                        objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)
            learner.fit(Xtr, Ytr)
            Yte = learner.predict(Xva)
        #         Tr_auc1[i][j] = learner.auc(XtS, Yt)
            print("max_depth: ", d, 'gamma', g)
            AUC1.append(AUC_Model(Yva, Yte))
```

```
max_depth:  30 gamma 0.0      AUC= 0.687083957805
max_depth:  30 gamma 0.1      AUC= 0.690391669166
max_depth:  30 gamma 0.2      AUC= 0.688677960554
max_depth:  30 gamma 0.3      AUC= 0.688868479236
max_depth:  30 gamma 0.4      AUC= 0.689939818983
```

The gamma should be 0.1.

```
f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(Va_auc1, interpolation='nearest')
f.colorbar(cax)
# ystick should be K since np.zeros((len(K),len(A)))
ax.set_xticklabels(['']+list(F))
ax.set_yticklabels(['']+list(D))
ax.set_title('Validation AUC')
plt.show()
```

2.4 Tuning for subsample & colsample_bytree

```
S = [i/10.0 for i in range(6,10)]
C = [i/10.0 for i in range(6,10)]
# Tr_auc1 = np.zeros((len(D),len(F)))
Va_auc3 = np.zeros((len(S),len(C)))

for i,s in enumerate(S):
    for j,c in enumerate(C):
        learner = xgb.XGBClassifier(learning_rate =0.1, n_estimators=100, max_depth=30,
        min_child_weight=5, gamma=0.1, subsample=s, colsample_bytree=c,
        objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)
        learner.fit(Xtr, Ytr)
        Yte = learner.predict(Xva)
        print("supsample= ", s, 'colsample_bytree=', c)
        Va_auc3[i][j] = AUC_Model(Yva, Yte)
```

```
supsample=  0.6     colsample_bytree= 0.6     AUC= 0.692106564743
supsample=  0.6     colsample_bytree= 0.7     AUC= 0.691567056776
supsample=  0.6     colsample_bytree= 0.8     AUC= 0.687463808203
supsample=  0.6     colsample_bytree= 0.9     AUC= 0.687522474168
supsample=  0.7     colsample_bytree= 0.6     AUC= 0.690442985893
supsample=  0.7     colsample_bytree= 0.7     AUC= 0.691407676402
```

```
supsample=  0.7      colsample_bytree= 0.8      AUC= 0.689230020507
supsample=  0.7      colsample_bytree= 0.9      AUC= 0.687436962873
supsample=  0.8      colsample_bytree= 0.6      AUC= 0.691662340993
supsample=  0.8      colsample_bytree= 0.7      AUC= 0.690386921302
supsample=  0.8      colsample_bytree= 0.8      AUC= 0.690391669166
supsample=  0.8      colsample_bytree= 0.9      AUC= 0.689106753748
supsample=  0.9      colsample_bytree= 0.6      AUC= 0.690942769595
supsample=  0.9      colsample_bytree= 0.7      AUC= 0.689649545416
supsample=  0.9      colsample_bytree= 0.8      AUC= 0.688064583509
supsample=  0.9      colsample_bytree= 0.9      AUC= 0.68731958119
```

The subsample should be 0.6 and colsample_bytree should be 0.6 as well.

2.5 Testing Result

Make the n_estimators bigger, set to 300.

Apply the rest of parameters.

```
learner = xgb.XGBClassifier(learning_rate =0.1, n_estimators=300, max_depth=30,
                min_child_weight=5, gamma=0.1, subsample=0.6, colsample_bytree=0.6,
                objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)
learner.fit(Xtr, Ytr)
Yte = learner.predict(Xva)
AUC_Model(Yva,Yte)
```

```
AUC= 0.695510768609
```

```
learner = xgb.XGBClassifier(learning_rate =0.5, n_estimators=300, max_depth=30,
                min_child_weight=5, gamma=0.1, subsample=0.6, colsample_bytree=0.6,
               objective= 'binary:logistic', nthread=4, scale_pos_weight=1,seed=27)
learner.fit(Xtr, Ytr)
Yte = learner.predict(Xva)
AUC_Model(Yva,Yte)
```

```
AUC= 0.681247063859
```

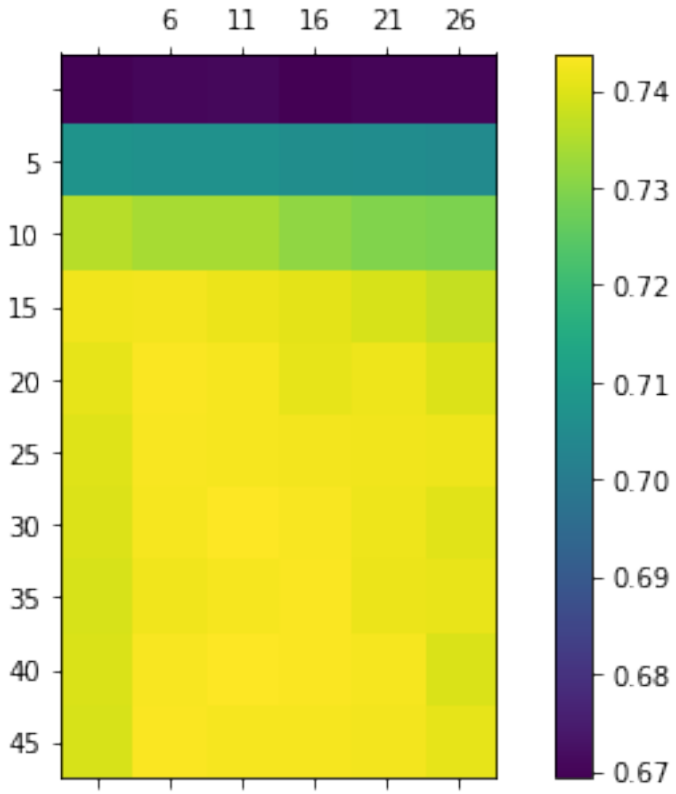
3 Second Method: Random Forest (Bagged Decision Tree)

There are four main properties in random forest, that are max depth,min samples split, n estimators, and min samples leaf. In this part, we will try to find the best value for each property.

3.1 min samples split

In this part, I draw the 2D graph for different values of max depth and min sample split. From this graph, we can see that min sample split produce the best result when it equals to 6. But we cannot see which max depth is the best clearly

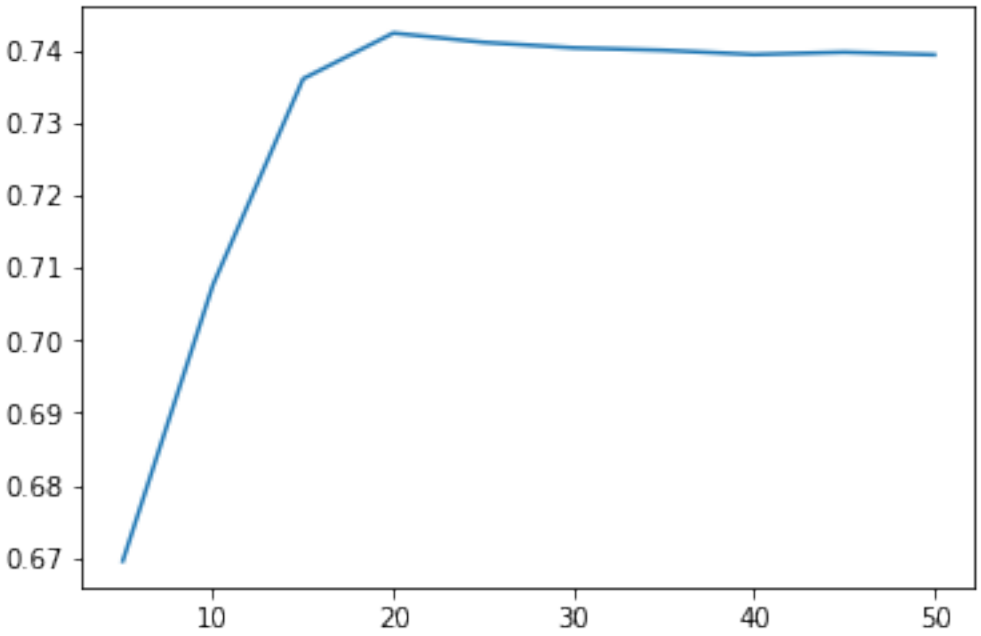
```
max_depth_arr = [5,10,15,20,25,30,35,40,45,50]
min_sample_split=[6,11,16,21,26,31]
tr_auc = np.zeros((len(max_depth_arr),len(min_sample_split)))
va_auc = np.zeros((len(max_depth_arr),len(min_sample_split)))
for i,d in enumerate(max_depth_arr):
    for j,s in enumerate(min_sample_split):
        print(i,j)
        clf = RandomForestClassifier(n_estimators=100, n_jobs=4,
                                     max_depth=d, min_samples_split=s)
        clf.fit(Xtr, Ytr)
        yr=clf.predict(Xva)
        Yresult = clf.predict_proba(Xva)
        fpr, tpr, thresholds = metrics.roc_curve(Yva, Yresult[:,1])
        va_auc[i][j]=metrics.auc(fpr, tpr)

f, ax = plt.subplots(1, 1, figsize=(8, 5))
cax = ax.matshow(va_auc, interpolation='nearest')
f.colorbar(cax)

ax.set_yticklabels(['']+list(max_depth_arr))
ax.set_xticklabels(['']+list(min_sample_split))
ax.set_yticks(np.arange(len(max_depth_arr)))
ax.set_xticks(np.arange(len(min_sample_split)))
```



```
plt.plot(max_depth_arr, va_auc[:,0])
plt.show()
```

3.2 Max Depth

In this part, we plot the auc value for max depth from 1 to 50, getting that it produce the best auc when max depth in 20.

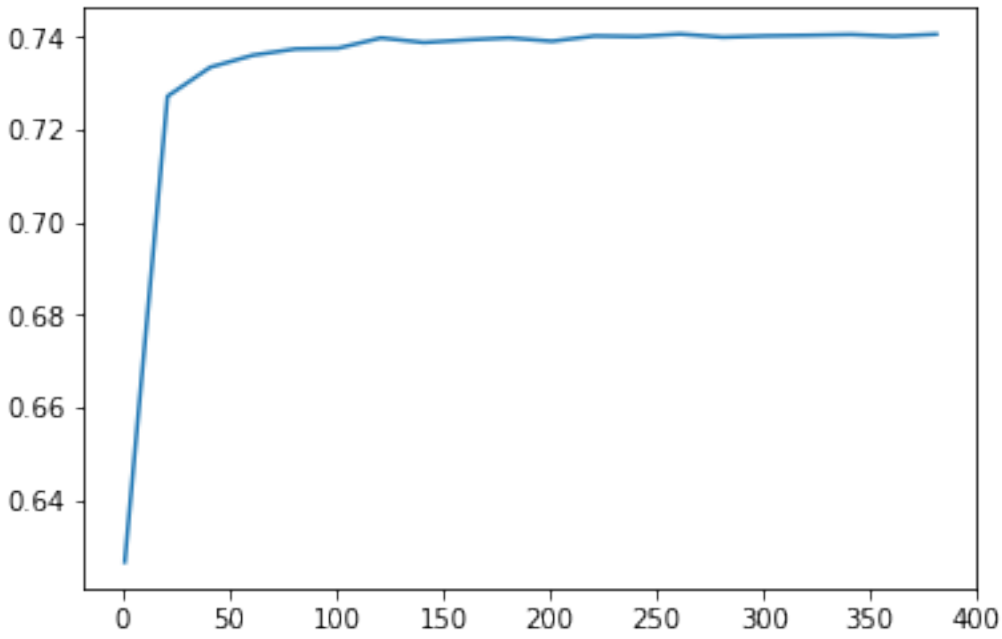



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
import numpy as np
import matplotlib.pyplot as plt
import mltools as ml
from sklearn import metrics


Xtr = np.loadtxt("X_train.txt", delimiter=None)
Ytr = np.loadtxt("Y_train.txt", delimiter=None)
Xte = np.loadtxt("X_test.txt", delimiter=None)
print("data loaded!")

Xtr, Ytr= ml.shuffleData(Xtr, Ytr);
Xtr, Xva, Ytr, Yva = ml.splitData(Xtr,Ytr,train_fraction=0.25)
```

3.3 n estimators

In this part, we draw the graph between auc and min n estimators. From the result, we can see that auc value does not improve significantly after n estimator becomes 120. Therefore, we pick 120 as the best value for n estimator

```
results=np.zeros(20)
print(results.shape)
x=range(0,20)
for i in range(0,20):
    clf = RandomForestClassifier(n_estimators=20*i+1, n_jobs=4, max_depth=20,
                                 min_samples_split=6, min_samples_leaf=1)
    clf.fit(Xtr, Ytr)
    yr=clf.predict(Xva)
    Yresult = clf.predict_proba(Xva)
    fpr, tpr, thresholds = metrics.roc_curve(Yva, Yresult[:,1])
    results[i]=metrics.auc(fpr, tpr)
plt.plot(x,results)
plt.show()
```

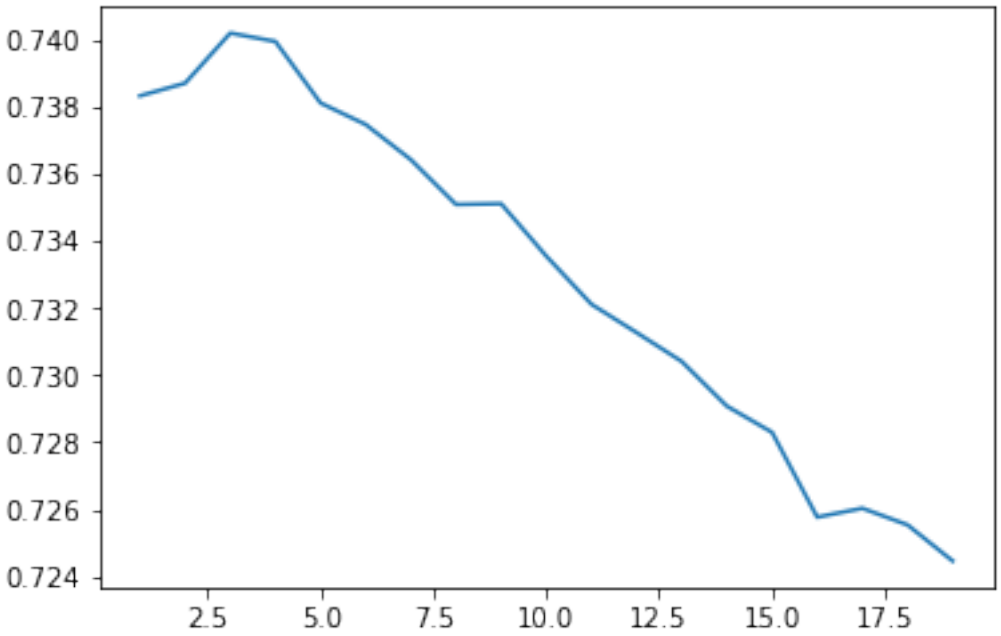


3.4 min sample leaf

In this part, we draw the graph between auc and min sample leaf. From the result, we can see that 3 is the best value for min sample leaf

```
results=np.zeros(19)
print(results.shape)
```

```
x=range(1,20)
for i in range(0,19):
    clf = RandomForestClassifier(n_estimators=120, n_jobs=4, max_depth=20,
                                 min_samples_split=6, min_samples_leaf=x[i])
    clf.fit(Xtr, Ytr)
    yr=clf.predict(Xva)
    Yresult = clf.predict_proba(Xva)
    fpr, tpr, thresholds = metrics.roc_curve(Yva, Yresult[:,1])
    results[i]=metrics.auc(fpr, tpr)
plt.plot(x,results)
plt.show()
```



3.5 Result in Kaggle

According to the previous tests, we get the best value for all the four main parameters. We use the results to train our model with all the data in xTrain and yTrain, and then predict y value for xTest. In this method, the score we got from Kaggle is 0.7406