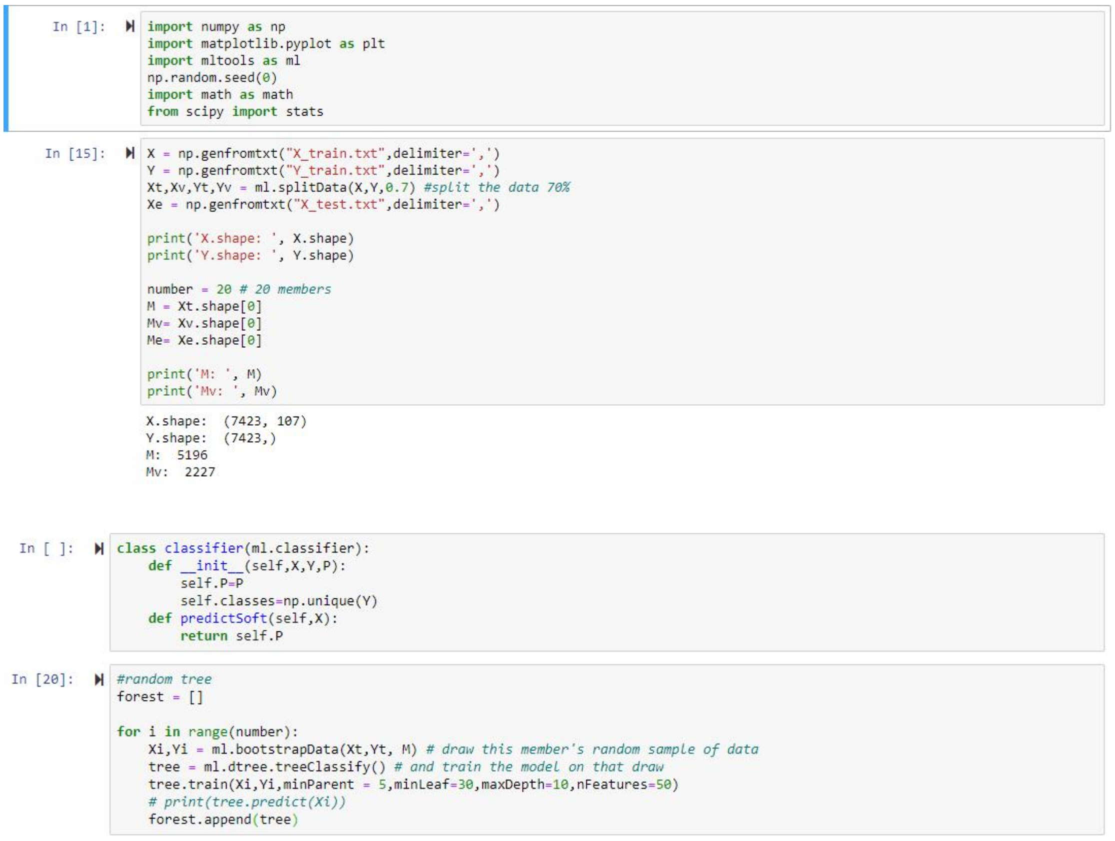


```
In [1]:  import numpy as np
         import matplotlib.pyplot as plt
         import mltools as ml
         np.random.seed(0)
         import math as math
         from scipy import stats
```

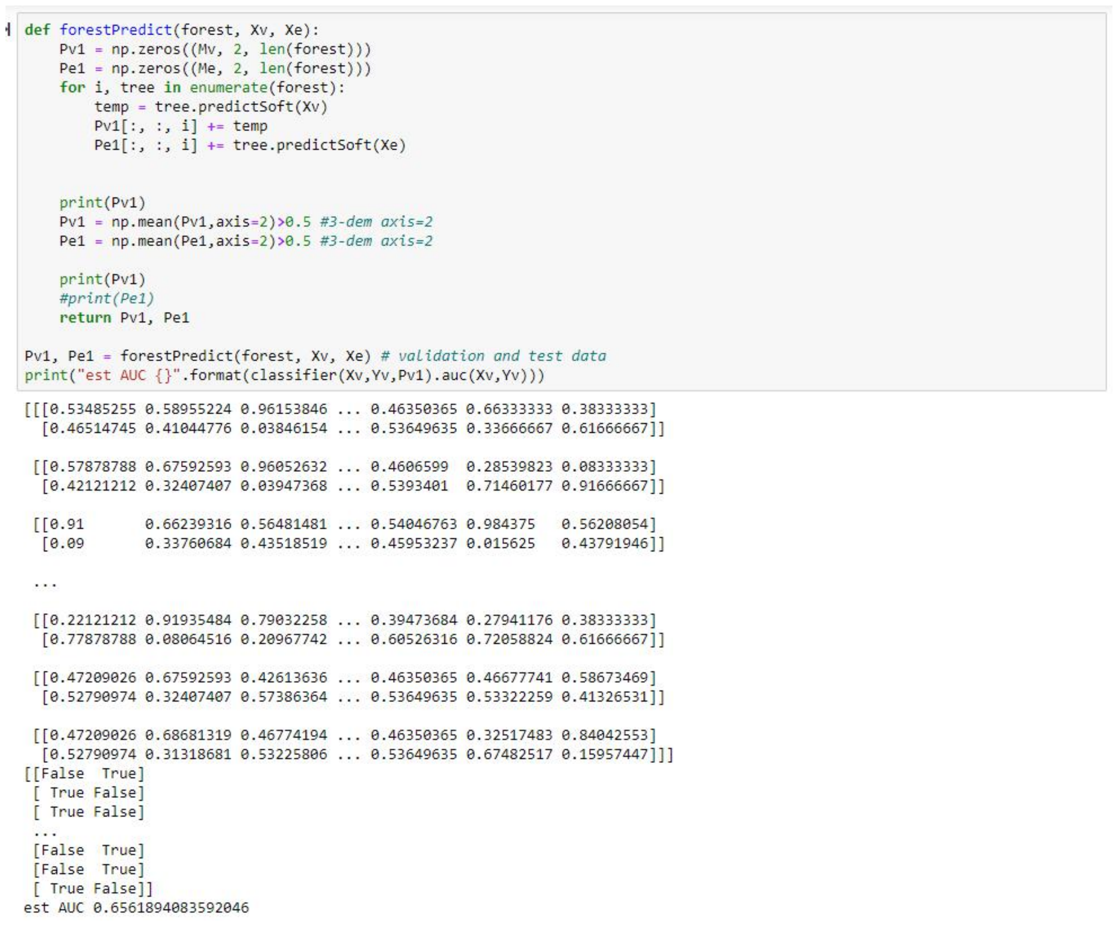
```
In [15]: X = np.genfromtxt("X_train.txt",delimiter=',')
         Y = np.genfromtxt("Y_train.txt",delimiter=',')
         Xt,Xv,Yt,Yv = ml.splitData(X,Y,0.7) #split the data 70%
         Xe = np.genfromtxt("X_test.txt",delimiter=',')

         print('X.shape: ', X.shape)
         print('Y.shape: ', Y.shape)

         number = 20 # 20 members
         M = Xt.shape[0]
         Mv= Xv.shape[0]
         Me= Xe.shape[0]

         print('M: ', M)
         print('Mv: ', Mv)
```

```
X.shape:  (7423, 107)
Y.shape:  (7423,)
M:  5196
Mv:  2227
```

```
In [ ]:  class classifier(ml.classifier):
             def __init__(self,X,Y,P):
                 self.P=P
                 self.classes=np.unique(Y)
             def predictSoft(self,X):
                 return self.P
```

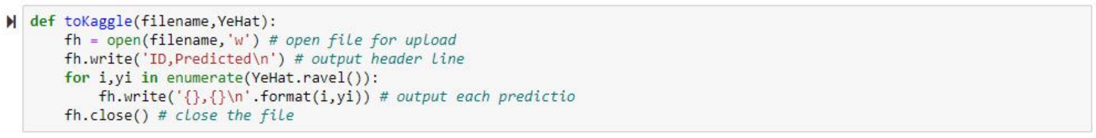
```
In [20]: #random tree
         forest = []

         for i in range(number):
             Xi,Yi = ml.bootstrapData(Xt,Yt, M) # draw this member's random sample of data
             tree = ml.dtree.treeClassify() # and train the model on that draw
             tree.train(Xi,Yi,minParent = 5,minLeaf=30,maxDepth=10,nFeatures=50)
             # print(tree.predict(Xi))
             forest.append(tree)
```

This model is a random forest model of bagged trees. The algorithms are constructed on Python and using the base functions provided by class like mltools. The main packages imported are matplotlib and numpy. The data is split by 70%. Here I use 20 members for this algorithm. Firstly, a forest(list of trees) is built. For each member, the function draws the member's random sample of data and then train the model on the draw. It is trained by assigned hyperparameters of minParent=5, minLeaf=30, maxDepth=10, and nFeatures=50). Then put the forest into prediction algorithm with X_validation and X_test. Using the soft prediction provided by class to generate predicted results. The estimated AUC here is around 0.65. At last, the predicted results are loaded into a new file.

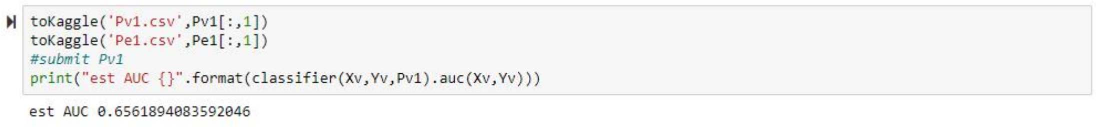


```
def forestPredict(forest, Xv, Xe):
    Pv1 = np.zeros((Mv, 2, len(forest)))
    Pe1 = np.zeros((Me, 2, len(forest)))
    for i, tree in enumerate(forest):
        temp = tree.predictSoft(Xv)
        Pv1[:, :, i] += temp
        Pe1[:, :, i] += tree.predictSoft(Xe)


    print(Pv1)
    Pv1 = np.mean(Pv1,axis=2)>0.5 #3-dem axis=2
    Pe1 = np.mean(Pe1,axis=2)>0.5 #3-dem axis=2

    print(Pv1)
    #print(Pe1)
    return Pv1, Pe1

Pv1, Pe1 = forestPredict(forest, Xv, Xe) # validation and test data
print("est AUC {}".format(classifier(Xv,Yv,Pv1).auc(Xv,Yv)))
```

```
[[[0.53485255 0.58955224 0.96153846 ... 0.46350365 0.66333333 0.38333333]
  [0.46514745 0.41044776 0.03846154 ... 0.53649635 0.33666667 0.61666667]]

 [[0.57878788 0.67592593 0.96052632 ... 0.4606599  0.28539823 0.08333333]
  [0.42121212 0.32407407 0.03947368 ... 0.5393401  0.71460177 0.91666667]]

 [[0.91       0.66239316 0.56481481 ... 0.54046763 0.984375   0.56208054]
  [0.09       0.33760684 0.43518519 ... 0.45953237 0.015625   0.43791946]]

 ...

 [[0.22121212 0.91935484 0.79032258 ... 0.39473684 0.27941176 0.38333333]
  [0.77878788 0.08064516 0.20967742 ... 0.60526316 0.72058824 0.61666667]]

 [[0.47209026 0.67592593 0.42613636 ... 0.46350365 0.46677741 0.58673469]
  [0.52790974 0.32407407 0.57386364 ... 0.53649635 0.53322259 0.41326531]]

 [[0.47209026 0.68681319 0.46774194 ... 0.46350365 0.32517483 0.84042553]
  [0.52790974 0.31318681 0.53225806 ... 0.53649635 0.67482517 0.15957447]]]
[[False  True]
 [ True False]
 [ True False]
 ...
 [False  True]
 [False  True]
 [ True False]]
est AUC 0.6561894083592046
```



```
def toKaggle(filename,YeHat):
    fh = open(filename,'w') # open file for upload
    fh.write('ID,Predicted\n') # output header line
    for i,yi in enumerate(YeHat.ravel()):
        fh.write('{},{}\n'.format(i,yi)) # output each predictio
    fh.close() # close the file
```

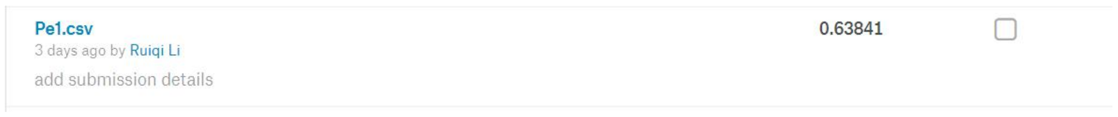


```
toKaggle('Pv1.csv',Pv1[:,1])
toKaggle('Pe1.csv',Pe1[:,1])
#submit Pv1
print("est AUC {}".format(classifier(Xv,Yv,Pv1).auc(Xv,Yv)))
```

```
est AUC 0.6561894083592046
```

The trees are appended into a forest list and learns every one. After learning and averaging, they will be combined. The random forest is picked because it can better handle some data with more dimensions and can learn the whole data's accuracy from the provided sample data which can be efficient.

The algorithm here is not that effective considered the reported AUC score which is less than 0.7. We have several hypotheses to this disadvantage. First, we may have a small number of entry members (20 may not be enough). Second, the hyperparameters are set poorly and result in overfitting. Thirdly, the prediction algorithm is done with 3-dimensional array which may increase the complexity.


Pe1.csv
3 days ago by Ruiqi Li
add submission details
0.63841

score of the poorest model on Kaggle: 0.63841 (use it? ROC? LOL)

(no particular why about how good is it, how to combine models??? ummm)

（我只能帮到这了，应该 not perfect 呜呜呜呜）