Homework 5: Tree-based Inference

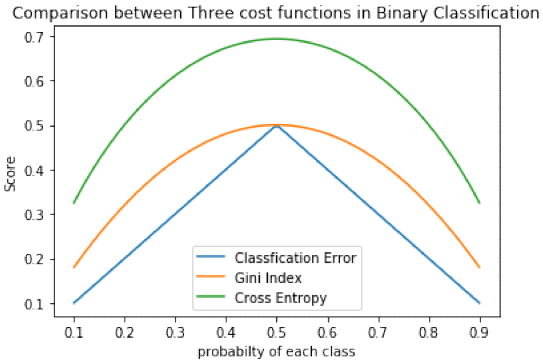
by Chu Zhuang

Conceptual: Cost functions for classification trees

In simple classification with two classes, conceptually these three methods for loss function are very close. **1) Cross-entropy** is the most accurate and sophisticated one, which calculates the cross-entropy of each class between predicted and true label, which accurately measures their difference of the whole distribution information contained; **2) Gini index** actually is a simpler form of cross-entropy that Gini index is the first order Taylor expansion of Cross-entropy at x=1; and in two classes classification settings they perform really close to each other (while Gini is more computationally friendly which does not calculate log); **3) Classification error** is a much simpler form and a coarse estimation of the loss that only calculate the discrepancy of the most different class, while in two class setting, this reduced accuracy in measuring loss is not that significant compared with the other two scores. **As shown in the figure below:**

```
#visualization of the difference between three cost functions, across different pmk
#since this is a binary classification, m=1 and m=0 should be identical
import numpy as np
import math
p=np.linspace(0.1,0.9,100)  #varying the pmk value across 0-1, when

#calculate cost functions
class_error=[1-max(p0,1-p0) for p0 in p]
Gini=[p0*(1-p0)+(1-p0)*(1-(1-p0)) for p0 in p]
Cross_entro=[-p0*math.log(p0)-(1-p0)*math.log((1-p0)) for p0 in p]
```

```
import matplotlib.pyplot as plt
plt.plot(p,class_error,label='Classfication Error');
plt.plot(p,Gini,label='Gini Index');
plt.plot(p,Cross_entro,label='Cross Entropy');
plt.xlabel('probabilty of each class');
plt.ylabel('Score');
plt.legend();
plt.title('Comparison between Three cost functions in Binary Classification');
```



All in all, Cross-entropy and Gini Index usually are prefered in calculating loss and when it is computationally heavy, Gini Index is more prefered.

As for growing and pruning a decision tree, the methods should not be differentiated that the evluation score for pruning process should be identical for growing process. There is no means of differentiation between these two process regarding loss functions.

Application: Predicting attitudes towards racist college professors

Load the data:

```
import pandas as pd
import numpy as np
df_gss_train=pd.read_csv('data/gss_train.csv')
df_gss_test=pd.read_csv('data/gss_test.csv')
df_gss_train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	age	attend	authoritarianism	black	born	childs	colath	colrac	colcom	colmil	...	partyid_3_Ind	pa
0	21	0	4	0	0	0	1	1	0	1	...	1	0
1	42	0	4	0	0	2	0	1	1	0	...	1	0
2	70	1	1	1	0	3	0	1	1	0	...	0	0
3	35	3	2	0	0	2	0	1	0	1	...	1	0
4	24	3	6	0	1	3	1	1	0	0	...	1	0

5 rows × 56 columns

Organize the data:

```
#organize the data to fit in model, for feature and predict value
#drop the predict value from the feature set
df_gss_train0=df_gss_train.drop('colrac',axis=1)
df_gss_test0=df_gss_test.drop('colrac',axis=1)

np_gss_train_feature=df_gss_train0.values
np_gss_train_y=df_gss_train['colrac'].values

np_gss_test_feature=df_gss_test0.values
np_gss_test_y=df_gss_test['colrac'].values
```

2. Estimate the Models

2.1 Logistic Regression

```
#build model to run Logistic Regression with 10-fold CV
from sklearn.linear_model import LogisticRegressionCV
lg_cv=LogisticRegressionCV(cv=10,solver='liblinear')
lg_cv.fit(np_gss_train_feature,np_gss_train_y)

#save cross validation scores (accuracy)
all_scores=np.mean(lg_cv.scores_[1],axis=1)

#predict Y based on train feature
lg_pY=lg_cv.predict(np_gss_train_feature)
```

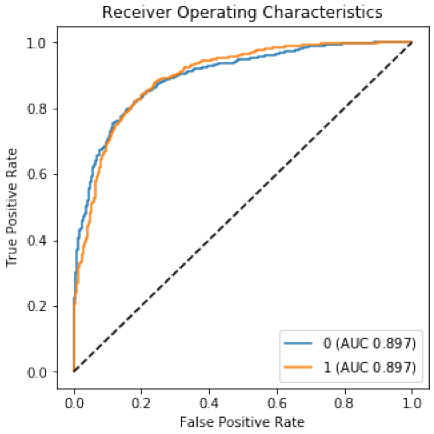
```
#calculate cross validation error rate, AUC score
import sklearn
cv_error_rate=1 - np.mean(all_scores)
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, lg_pY)
print('Cross Validated Error Rate of best Logistic Regression model:',round(cv_error_rate,4))
print('AUC score of best Logistic Regression model:',round(auc_score,4))
```

```
Cross Validated Error Rate of best Logistic Regression model: 0.2231
AUC score of best Logistic Regression model: 0.8165
```

```
#define variables to store all evaluations results
model_label=[]
error_rate_all=[]
auc_score_all=[]
```

```
#save the results
model_label.append('logistic_regression')
error_rate_all.append(cv_error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(lg_cv,np_gss_train_feature,np_gss_train_y,'logistic_regression')
```




```
#function to plot ROC
import matplotlib.pyplot as plt
def plot_ROC_curve(model,data_feature,true_label,method_des):
    clf=model
    classes = clf.classes_
    try:
        probs = clf.predict_proba(data_feature)
    except AttributeError:
        print("The {} classifier does not apear to support prediction probabilties, so an ROC curve can't be created. You can try adding
`probability = True` to the model specification or use a different model.".format(type(clf)))
        return
    predictions = clf.predict(data_feature)

    #setup axis for plotting
    fig, ax = plt.subplots(figsize = (5,5))

    #We can return the AUC values, in case they are useful
    aucVals = []
    for classIndex, className in enumerate(classes):        #Setup binary classes
        truths = [1 if c == className else 0 for c in true_label]
        predict = [1 if c == className else 0 for c in predictions]
        scores = probs[:, classIndex]

        #Get the ROC curve
        fpr, tpr, thresholds = sklearn.metrics.roc_curve(truths, scores)
        #fpr, tpr, thresholds = sklearn.metrics.roc_curve(truths, predictions)
        auc = sklearn.metrics.auc(fpr, tpr)
        aucVals.append(auc)

        #Plot the class's line
        ax.plot(fpr, tpr, label = "{} (AUC ${:.3f}$)".format(str(className).split(':')[0], auc))

    #Make the plot nice, then display it
    ax.set_title('Receiver Operating Characteristics')
    plt.plot([0,1], [0,1], color = 'k', linestyle='--')
    ax.set_xlabel('False Positive Rate')
    ax.set_ylabel('True Positive Rate')
    ax.legend(loc = 'lower right')
    plt.show()
    #plt.close()
```

2.2 Naive Bayes

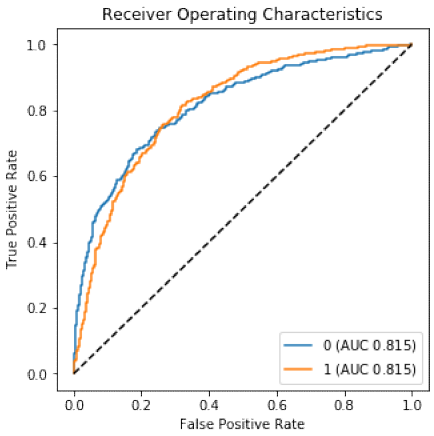
```
#build model to run Naive Bayes with 10-fold CV
from sklearn.naive_bayes import GaussianNB
model_nb=GaussianNB()  #use the GaussianNB

#10-fold cross validation, with no hyperparameters to tune
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=10)
er_nb=[]
auc_nb=[]
for train_index, test_index in skf.split(np_gss_train_feature,np_gss_train_y):
    X_train, X_test = np_gss_train_feature[train_index], np_gss_train_feature[test_index]
    y_train, y_test = np_gss_train_y[train_index], np_gss_train_y[test_index]
    model_nb.fit(X_train,y_train)

    #calculate accuracy based on test(validation) samples
    er_score=1-sklearn.metrics.accuracy_score(y_test, model_nb.predict(X_test))
    er_nb.append(er_score)
    auc_score=sklearn.metrics.roc_auc_score(y_test, model_nb.predict(X_test))
    auc_nb.append(auc_score)
```

```
#calculate the error rate and auc_score
error_rate=sum(er_nb)/len(er_nb)
auc_score=sum(auc_nb)/len(auc_nb)
print('Error Rate of Naive Bayes model:',round(error_rate,4))
print('AUC score of Naive Bayes model:',round(auc_score,4))
```

```
Error Rate of Naive Bayes model: 0.2656
AUC score of Naive Bayes model: 0.7351
```

```
#save the results
model_label.append('Naive Bayes')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC curve
plot_ROC_curve(model_nb,np_gss_train_feature,np_gss_train_y,'Naive Bayes')
```



2.3 Elastic net regression

Because this is a **classification problem**, I will build the **elastic net regression** with logistic regression model.

```
#standardized the data
from sklearn import preprocessing
np_gss_train_feature1=np_gss_train_feature
np_gss_test_feature1=np_gss_test_feature

scaler = preprocessing.StandardScaler().fit(np_gss_train_feature1)
scaler.transform(np_gss_train_feature1)
scaler.transform(np_gss_test_feature1)
```

```
array([[-1.4633296 , -0.52099312, -0.91644208, ..., -0.54812456,
        -0.76956677,  0.3363364 ],
       [ 0.04903065, -1.22070726,  0.94588263, ..., -0.54812456,
        -0.76956677,  0.3363364 ],
       [ 0.10504399, -1.22070726,  0.32510772, ..., -0.54812456,
         1.29943241,  0.3363364 ],
       ...,
       [ 0.7772041 ,  0.17872102, -0.91644208, ..., -0.54812456,
        -0.76956677,  0.3363364 ],
       [-1.12724955,  0.17872102,  0.94588263, ..., -0.54812456,
         1.29943241,  0.3363364 ],
       [ 1.44936422, -0.87085019, -0.29566718, ...,  1.82440285,
        -0.76956677,  0.3363364 ]])
```

```
#build model to run Logistic Regression with 10-fold CV
from sklearn.linear_model import LogisticRegressionCV
l1_ratio=np.linspace(0,1,11)    #set a range of l1_ratio, parameters to tune
lgen_cv=LogisticRegressionCV(cv=10,penalty='elasticnet',solver='saga',random_state=5,l1_ratios=l1_ratio)
lgen_cv.fit(np_gss_train_feature1,np_gss_train_y)
```

```
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the
coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
C:\Users\zhuangchu\Anaconda3\lib\site-packages\sklearn\linear_model\sag.py:337: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  "the coef_ did not converge", ConvergenceWarning)
```

```
LogisticRegressionCV(Cs=10, class_weight=None, cv=10, dual=False,
                     fit_intercept=True, intercept_scaling=1.0,
                     l1_ratios=array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
                     max_iter=100, multi_class='warn', n_jobs=None,
                     penalty='elasticnet', random_state=5, refit=True,
                     scoring=None, solver='saga', tol=0.0001, verbose=0)
```

```
#save the returned model and parameters
en_p_beta=lgen_cv.coef_
en_best_l1_ratio = lgen_cv.l1_ratio_
en_best_l1_ratio=en_best_l1_ratio.tolist()[0]
en_p_beta=en_p_beta.tolist()[0]
```

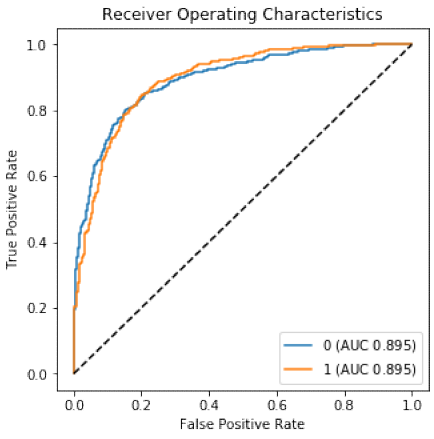
```
#save cross validation scores (accuracy)
all_scores=np.mean(lgen_cv.scores_[1],axis=1)

#find the non-zero coefficients in Lasso:
en_p_beta0=[(i,p) for i,p in enumerate(en_p_beta) if p>0]

#predict Y based on train feature
lgen_pY=lgen_cv.predict(np_gss_train_feature1)
```

```
#calculate error rate, AUC score
error_rate=1-np.mean(all_scores)
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, lg_pY)

#print the parameters of best model
print('Error Rate of best Elastic Net Regression model:',round(error_rate,4))
print('AUC score of best Elastic Net Regression model:',round(auc_score,4))
print('Elastic Net L1_ratio:',round(en_best_l1_ratio,4))
print('Elastic Net Num of Non-zero Coefficicents:',len(en_p_beta0))
print('Elastic Net Coefficicents:',en_p_beta0)
```

```
Error Rate of best Elastic Net Regression model: 0.2496
AUC score of best Elastic Net Regression model: 0.8165
Elastic Net L1_ratio: 0.2
Elastic Net Num of Non-zero Coefficicents: 27
Elastic Net Coefficicents: [(1, 0.04395280045805817), (3, 0.18096958989920955), (6, 1.2885664431649664), (8, 0.6377740811756197), (10,
1.6964389392962764), (11, 0.1387604121740369), (12, 0.003421298906047333), (15, 0.08503594361799081), (16, 0.03676421880738991), (17,
0.09520257857938343), (18, 0.034889168181790296), (19, 0.5842143608788467), (20, 0.3760309622211284), (24, 0.04648943093841409), (26,
0.15056828327216032), (27, 0.051527811563957956), (29, 8.059984625544057e-05), (31, 0.11205431172058442), (35, 0.08944159132801908), (39,
0.4574744980624184), (44, 0.0036647473603332982), (47, 0.17010718664405572), (48, 0.17751416051470115), (49, 0.29707529127686777), (50,
0.11916214623985218), (52, 0.2760428366660965), (54, 0.11169643713409692)]
```

```
#save the results
model_label.append('ElasticNet_Regression')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(lgen_cv,np_gss_train_feature1,np_gss_train_y,'ElasticNet_Regression')
```



2.4 Decision Tree (CART)

Grow a CART tree and tune the pruning parameters: tree depth & max leaf nodes with 10-fold cross validation.

```
#build model to run Decision Tree (CART) with 10-fold CV
from sklearn.tree import DecisionTreeClassifier
cart_cv=DecisionTreeClassifier()

#tuning parameters to prune the tree
max_depth=np.linspace(10,40,11)         #max depth
max_depth=max_depth.astype(np.int16)
max_leaf_nodes=np.linspace(5,50,11)     #max leaf nodes
max_leaf_nodes=max_leaf_nodes.astype(np.int16)

#search for best parameters based on 10-fold cross validation
from sklearn.model_selection import GridSearchCV
param_grid = [{'max_depth':max_depth,'max_leaf_nodes':max_leaf_nodes}]
grid_search = GridSearchCV(cart_cv, param_grid, cv=10, scoring='accuracy')
grid_search.fit(np_gss_train_feature,np_gss_train_y)

#return the best fitting results (model and parameter)
best_params=grid_search.best_params_
max_depth=best_params['max_depth']
```

```
max_leaf_nodes=best_params['max_leaf_nodes']
cart_best=grid_search.best_estimator_

#return the cross validated scores ('accuracy')
all_results=grid_search.cv_results_
all_params=all_results['params']
mean_test_score=all_results['mean_test_score']

#predict Y based on train feature
cart_pY=cart_best.predict(np_gss_train_feature)
```

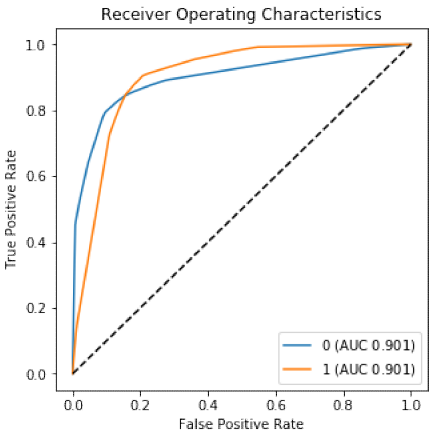
```
#find the best model and corresponding cross-validated error rate
for i,item in enumerate(all_params):
    if item['max_depth']==max_depth and item['max_leaf_nodes']==max_leaf_nodes:
        index_best_fit=i
error_rate=1-mean_test_score[index_best_fit]
```

```
#calculate error rate, AUC score
error_rate=1-mean_test_score[index_best_fit]
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, cart_pY)
print('Error Rate of best Decision Tree model CV:',round(error_rate,4))
print('AUC score of best Decision Tree model:',round(auc_score,4))
print('Depth of the Tree:',max_depth)
print('Max Num of leaf nodes:',max_leaf_nodes)
```

```
Error Rate of best Decision Tree model CV: 0.2134
AUC score of best Decision Tree model: 0.8488
Depth of the Tree: 28
Max Num of leaf nodes: 27
```

```
#save the results
model_label.append('Decision Tree-CART')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(cart_best,np_gss_train_feature,np_gss_train_y,'Decision Tree-CART')
```



2.5 Bagging

Tuning the hyperparameters in Bagging model: 1) num of trees for bagging; 2) max depth & 3) max leaf nodes to pruning trees based on 10-fold CV.

```
#build model to run Bagging with 10-fold CV
from sklearn.ensemble import RandomForestClassifier
bagg_cv=RandomForestClassifier()

#tuning parameters
#tuning parameters: considering the really slow process of training the model,
#I manually set several possible values of parameteres
#after several quick iterations
n_estimators=[5,20,35,50,75,100]        #numbers of trees for bagging
max_depth=[30,45,60,90]                 #max_depth of tree
max_leaf_nodes=[30,50,70,90,120]        #max leaf nodes

#search for best parameters based on 10-fold cross validation
from sklearn.model_selection import GridSearchCV
param_grid = [{'n_estimators':n_estimators,'max_depth':max_depth,'max_leaf_nodes':max_leaf_nodes}]
grid_search = GridSearchCV(bagg_cv, param_grid, cv=10, scoring='accuracy')
grid_search.fit(np_gss_train_feature,np_gss_train_y)

#return the best fitting results: model and parameters
bagg_best=grid_search.best_estimator_
```

```
best_params=grid_search.best_params_
max_depth=best_params['max_depth']
max_leaf_nodes=best_params['max_leaf_nodes']
n_estimator=best_params['n_estimators']

#return the score ('accuracy')
all_results=grid_search.cv_results_
all_params=all_results['params']
mean_test_score=all_results['mean_test_score']

#predict Y based on train feature
bagg_pY=bagg_best.predict(np_gss_train_feature)
```

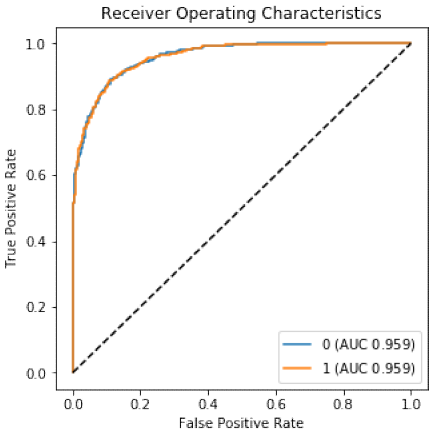
```
#find the best model and corresponding cross-validated error rate
for i,item in enumerate(all_params):
    if item['max_depth']==max_depth and item['max_leaf_nodes']==max_leaf_nodes and item['n_estimators']==n_estimator:
        index_best_fit=i
error_rate=1-mean_test_score[index_best_fit]
```

```
#calculate error rate, AUC score
#error_rate=1 -  sklearn.metrics.accuracy_score(np_gss_train_y, bagg_pY)
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, bagg_pY)
print('Error Rate of best Bagging Decision Tree model:',round(error_rate,4))
print('AUC score of best Bagging Decision Tree model:',round(auc_score,4))
print('Depth of the Tree:',max_depth)
print('Max Num of leaf nodes:',max_leaf_nodes)
print('Num of Trees:',n_estimator)
```

```
Error Rate of best Bagging Decision Tree model: 0.1972
AUC score of best Bagging Decision Tree model: 0.8773
Depth of the Tree: 30
Max Num of leaf nodes: 50
Num of Trees: 75
```

```
#save the results
model_label.append('Decision Tree-Bagging')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(bagg_best,np_gss_train_feature,np_gss_train_y,'Decision Tree-Bagging')
```



2.6 Random Forest

To build a Random Forest model, first of all, I manually calculate the 'default m-feature value': sqrt(all features)--which close to 7~8. Thus in the following tuning process, I will tune max_features parameter around this value.

```
#tuning parameters: max features
import math
math.sqrt(55)   #sqrt of total numbers of features
```

```
7.416198487095663
```

```
#build model to run Random Forest with 10-fold CV
import numpy as np
from sklearn.ensemble import RandomForestClassifier
```

```
RF_cv=RandomForestClassifier()
#tuning parameters: considering the really slow process of training the model,
#I manually set several possible values of parameteres
#after several quick iterations
max_features=[3,8,15]               #number of features for each split
n_estimators=[5,10,50,100,200]      #numbers of trees for bagging
max_leaf_nodes=[30,50,100,125]      #max leaf nodes

#search for best parameters based on 10-fold cross validation
from sklearn.model_selection import GridSearchCV
#param_grid = [{'n_estimators':n_estimators,'max_features':max_features,'max_depth':max_depth,'max_leaf_nodes':max_leaf_nodes}]
param_grid = [{'n_estimators':n_estimators,'max_features':max_features,'max_leaf_nodes':max_leaf_nodes}]
grid_search = GridSearchCV(RF_cv, param_grid, cv=10, scoring='accuracy')
grid_search.fit(np_gss_train_feature,np_gss_train_y)

#return the best fitting results: model and parameters
RF_best=grid_search.best_estimator_
best_params=grid_search.best_params_
max_leaf_nodes=best_params['max_leaf_nodes']
n_estimator=best_params['n_estimators']
max_feature=best_params['max_features']

#return the score ('accuracy')
all_results=grid_search.cv_results_
all_params=all_results['params']
mean_test_score=all_results['mean_test_score']

#predict Y based on train feature
RF_pY=RF_best.predict(np_gss_train_feature)
```

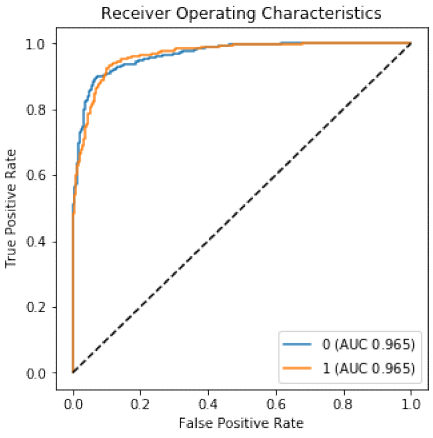
```
#find the best model and corresponding cross-validated error rate
for i,item in enumerate(all_params):
    if item['max_leaf_nodes']==max_leaf_nodes and item['n_estimators']==n_estimator and item['max_features']==max_feature:
        index_best_fit=i
error_rate=1-mean_test_score[index_best_fit]
```

```
#calculate error rate, AUC score
#error_rate=1 -  sklearn.metrics.accuracy_score(np_gss_train_y, RF_pY)
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, RF_pY)
print('Error Rate of best Random Forest model:',round(error_rate,4))
print('AUC score of best Random Foreste model:',round(auc_score,4))
print('Max Num of leaf nodes:',max_leaf_nodes)
print('Num of Trees:',n_estimator)
print('Num of Features for each Split:',max_feature)
```

```
Error Rate of best Random Forest model: 0.0928
AUC score of best Random Foreste model: 0.9053
Max Num of leaf nodes: 50
Num of Trees: 50
Num of Features for each Split: 15
```

```
#save the results
model_label.append('Randomn_Forest')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(RF_best,np_gss_train_feature,np_gss_train_y,'Randomn_Forest')
```



2.7 Boosting

Build Boosting model and tune hyperparameters based on 10-fold cross validation: 1) learning_rate; 2) number of trees for boostrapping; 3) max_depth of each tree:

```
#build model to run Bagging with 10-fold CV
from sklearn.ensemble import GradientBoostingClassifier
Boost_cv=GradientBoostingClassifier(subsample=0.8)
#tuning parameters
learning_rate=[0.01,0.05,0.1,0.3,0.5]  #learning_rate
n_estimators=[100,300,500,800]        #numbers of bagging
max_depth=[2,3,4]                      #max_depth of tree

#search for best parameters based on 10-fold cross validation
from sklearn.model_selection import GridSearchCV
param_grid = [{'learning_rate':learning_rate,'n_estimators':n_estimators,'max_depth':max_depth}]
grid_search = GridSearchCV(Boost_cv, param_grid, cv=10, scoring='accuracy')
grid_search.fit(np_gss_train_feature,np_gss_train_y)

#return the best fitting results: model and parameters
Boost_best=grid_search.best_estimator_
best_params=grid_search.best_params_
max_depth=best_params['max_depth']
n_estimator=best_params['n_estimators']
learning_rate=best_params['learning_rate']

#return the score ('accuracy')
all_results=grid_search.cv_results_
all_params=all_results['params']
mean_test_score=all_results['mean_test_score']

#predict Y based on train feature
Boost_pY=Boost_best.predict(np_gss_train_feature)
```

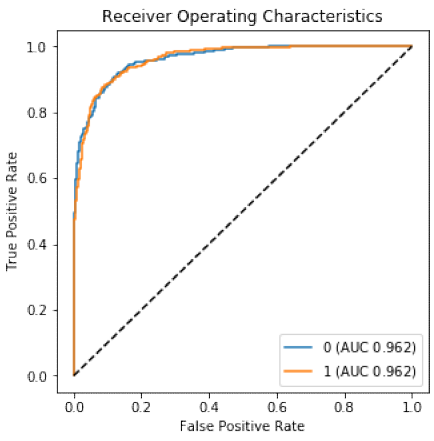
```
#find the best model and corresponding cross-validated error rate
for i,item in enumerate(all_params):
    if item['max_depth']==max_depth and item['n_estimators']==n_estimator and item['learning_rate']==learning_rate:
        index_best_fit=i
```

```
#calculate error rate, AUC score
error_rate=1-mean_test_score[index_best_fit]
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, Boost_pY)
print('Error Rate of best Boosting model:',round(error_rate,4))
print('AUC score of best Boosting model:',round(auc_score,4))
print('Depth of the Tree:',max_depth)
print('Num of Trees:',n_estimator)
print('Learning Rate:',learning_rate)
```

```
Error Rate of best Boosting model: 0.1958
AUC score of best Boosting model: 0.8879
Depth of the Tree: 3
Num of Trees: 800
Learning Rate: 0.01
```

```
#save the results
model_label.append('Boosting')
error_rate_all.append(error_rate)
auc_score_all.append(auc_score)
```

```
#plot the ROC
plot_ROC_curve(Boost_best,np_gss_train_feature,np_gss_train_y,'Boosting')
```



3/4. Evaluate the Model

Aggregate training performance of each model after parameters tuning:

```
df_ER=pd.DataFrame({'Error Rate':error_rate_all,'AUC Score':auc_score_all},index=model_label)
df_ER
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Error Rate	AUC Score
logistic_regression	0.223100	0.816476
Naive Bayes	0.265553	0.735113
ElasticNet_Regression	0.249598	0.816476
Decision Tree-CART	0.213415	0.848834
Decision Tree-Bagging	0.197154	0.877261
Randomn_Forest	0.092818	0.905279
Boosting	0.195799	0.887924

Best Model: Random Forest

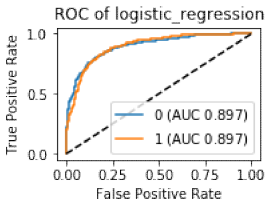
As we could see from both the Error Rate, AUC score and ROC, the best model is the `Random Forest` (with at most 15 features into consideration for each split and max depth 50), which has the lowest cross validated Error Rate and highest AUC score.

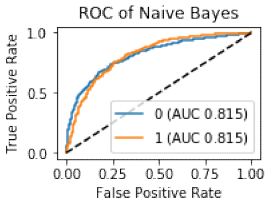
Naive Bayes perform worst among all models which might due to the violation of variable independence. Logistic/Elastic net performs very close and in between of naive bayes and tree models, which means the relationship between the predictors and response 'Colrac' is not purely linear.

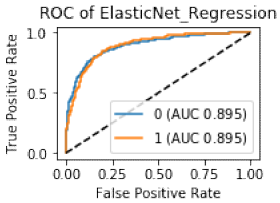
Tree models perform best in this problem, with Random Forest the best, which takes both 'bagging' and 'feature similarity' into account by boostrapping with varied set of features each time. Boosting Tree also performs very well in this problem, but does not outperform Random Forest, which might due to the large number of predictors in this dataset that Random Forest could better take advantage of within smaller numbers of iteration, or it could also because of the incomplete parameters tuning process in this homework analysis due to computation limitation.

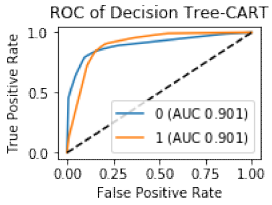
Plot the ROC of all models:

```
#plt.figure(figsize=(20,20))
plot_ROC_curve_sub(lg_cv,np_gss_train_feature,np_gss_train_y,'logistic_regression',221)
plot_ROC_curve_sub(model_nb,np_gss_train_feature,np_gss_train_y,'Naive Bayes',222)
plot_ROC_curve_sub(lgen_cv,np_gss_train_feature1,np_gss_train_y,'ElasticNet_Regression',223)
plot_ROC_curve_sub(cart_best,np_gss_train_feature1,np_gss_train_y,'Decision Tree-CART',224)
```

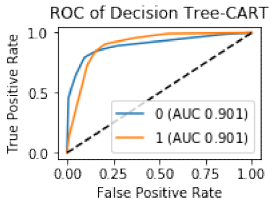


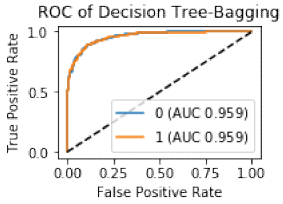


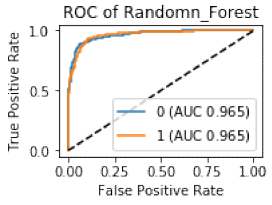


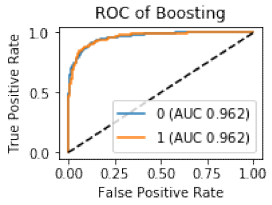



```
#plt.figure(figsize=(10,10))
plot_ROC_curve_sub(cart_best,np_gss_train_feature1,np_gss_train_y,'Decision Tree-CART',221)
plot_ROC_curve_sub(bagg_best,np_gss_train_feature,np_gss_train_y,'Decision Tree-Bagging',222)
plot_ROC_curve_sub(RF_best,np_gss_train_feature,np_gss_train_y,'Randomn_Forest',223)
plot_ROC_curve_sub(Boost_best,np_gss_train_feature,np_gss_train_y,'Boosting',224)
```










```
#function to plot ROC of subplots
def plot_ROC_curve_sub(model,data_feature,data_category,method_des,loc):
    clf=model
    classes = clf.classes_
    try:
        probs = clf.predict_proba(data_feature)
    except AttributeError:
        print("The {} classifier does not apear to support prediction probabilties, so an ROC curve can't be created. You can try adding
`probability = True` to the model specification or use a different model.".format(type(clf)))
        return
    predictions = clf.predict(data_feature)

    #setup axis for plotting
    #fig, ax = plt.subplot(221)
    plt.subplot(loc)

    #We can return the AUC values, in case they are useful
    aucVals = []
    for classIndex, className in enumerate(classes):        #Setup binary classes
        truths = [1 if c == className else 0 for c in data_category]
        predict = [1 if c == className else 0 for c in predictions]
        scores = probs[:, classIndex]
```

```
        #Get the ROC curve
        fpr, tpr, thresholds = sklearn.metrics.roc_curve(truths, scores)
        auc = sklearn.metrics.auc(fpr, tpr)
        aucVals.append(auc)

        #Plot the class's line
        plt.plot(fpr, tpr, label = "{} (AUC ${:.3f}$)".format(str(className).split(':')[0], auc))

    #Make the plot nice, then display it
    plt.title('ROC of '+method_des)
    plt.plot([0,1], [0,1], color = 'k', linestyle='--')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.legend(loc = 'lower right')
    plt.show()
    #plt.close()
```

5. Evaluate the Best model on Test

Finally, evaluate the best model on the test dataset:

```
#predict Y based on test feature
RF_pY_test=RF_best.predict(np_gss_test_feature)
```

```
#calculate the error rate and AUC score for Test
error_rate=1 - sklearn.metrics.accuracy_score(np_gss_test_y, RF_pY_test)
auc_score=sklearn.metrics.roc_auc_score(np_gss_test_y, RF_pY_test)
print('For Testing:')
print('Error Rate of the best Random Forest model:',round(error_rate,4))
print('AUC score of the best Random Forest model:',round(auc_score,4))
```

```
For Testing:
Error Rate of the best Random Forest model: 0.2049
AUC score of the best Random Forest model: 0.7877
```

As we could see from the performance results (above: testing; below: training): **the Random Forest model does not generalize very well**, the performance of test dataset is not good as the training dataset, even not reach the lowest performance model 'Naive Bayes' in training. The reasons might come from the difference between the training and testing dataset, since Tree models are extremely sensitive to different data and have generally **High variance** in predicting. 'Overfitting' might also be another reason. however, considering the well-controlled pruning process, limiting both tree depth and num of leafs, we have tried best to avoid the 'overfitting' problem when growing the tree. The high-variance of tree model in generalization is inherent and non-negligible.

```
#compare the result with training
error_rate=1 -  sklearn.metrics.accuracy_score(np_gss_train_y, RF_pY)
auc_score=sklearn.metrics.roc_auc_score(np_gss_train_y, RF_pY)
print('For Training:')
print('Error Rate of the best Random Forest model:',round(error_rate,4))
print('AUC score of the best Random Forest model:',round(auc_score,4))
```

```
For Training:
Error Rate of the best Random Forest model: 0.0928
AUC score of the best Random Forest model: 0.9053
```

Bonus:

PDP for `Tolerance`: first of all, find the range of the raw score

```
min(df_gss_train0['tolerance'])
```

```
0
```

```
max(df_gss_train0['tolerance'])
```

```
15
```

Cut the raw score into different bins:

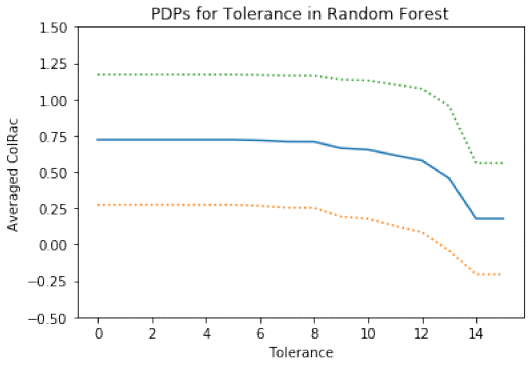
```
tolerance_grid=np.linspace(0,15,16)
tolerance_grid=tolerance_grid.astype((np.int16))
tolerance_grid
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15],
      dtype=int16)
```

Copy the training dataset and substitute the original value of 'tolerance' with values of each bin, and predict the `Colrac` score by the best model `Random Forest` on each substituted training dataset:

```
#calculate predicted value for Tolerance
tol_num=len(tolerance_grid)
tol_pY_all,tol_pY0_all,tol_pY1_all=[],[],[]
model=RF_best
for i in range(tol_num):
    #copy the training dataset and substitue orginal value
    df_gss_train1=df_gss_train0
    df_gss_train1['tolerance']=tolerance_grid[i]
    np_gss_train_feature0=df_gss_train1.values

    tol_pY=model.predict(np_gss_train_feature0)
    tol_pY_avg=np.mean(tol_pY)
    tol_pY_sd=np.std(tol_pY,ddof = 1)
    tol_pY_avg_sd1=tol_pY_avg+tol_pY_sd
    tol_pY_avg_ssd1=tol_pY_avg-tol_pY_sd

    tol_pY_all.append(tol_pY_avg)
    tol_pY0_all.append(tol_pY_avg_ssd1)
    tol_pY1_all.append(tol_pY_avg_sd1)
```

```
import matplotlib.pyplot as plt
plt.plot(tolerance_grid,tol_pY_all);
plt.plot(tolerance_grid,tol_pY0_all,linestyle=':');
plt.plot(tolerance_grid,tol_pY1_all,linestyle=':');
plt.ylim(-0.5,1.5);
plt.xlabel('Tolerance');
plt.ylabel('Averaged ColRac');
plt.title('PDPs for Tolerance in Random Forest');
plt.show();
```



From the PDP graph above for `Tolerance`, we could see that with the increasing of level of `Tolerance`, the predicted `Colrac` probability drops; especially when the level of `Tolerance` is extremely high (>12), it significantly negatively influences the `Colrac` probability; which when the level of `Tolerance` is low, this negative influence is relatively small.

All in all, from the PDP graph we could infer that `Tolerance` has an negative impact on attitudes towards racist teacher and this negative relationship is not linear and is biggest at the upper extreme-- Extreme higher tolerance score indicates extreme lower `Colrac` score-not allowing a racist teacher. (Very sadly, I did not find an explanation of the variable `Tolerance` in the GSS website and could not interpret this varaible in a meaningful way)

PDP for `Age`: implement the same process as for `Tolerance`.

First of all find the range of `Age` and divde it into bins:

```
min(df_gss_train0['age'])
```

```
18
```

```
max(df_gss_train0['age'])
```

```
89
```

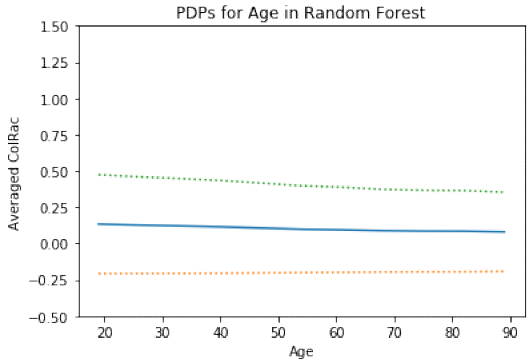
```
age_grid=np.linspace(19,89,11)
age_grid=age_grid.astype((np.int16))
age_grid
```

```
array([19, 26, 33, 40, 47, 54, 61, 68, 75, 82, 89], dtype=int16)
```

Copy the training dataset and substitute the original value of 'age' with values of each bin, and predict the `Colrac` score by the best model `Random Forest` on each substituted training dataset:

```
#calculate predicted value for Age
age_num=len(age_grid)
age_pY_all,age_pY0_all,age_pY1_all=[],[],[]
model=RF_best
for i in range(age_num):
    df_gss_train1=df_gss_train0
    df_gss_train1['age']=age_grid[i]
    np_gss_train_feature0=df_gss_train1.values

    age_pY=lg_cv.predict(np_gss_train_feature0)
    age_pY_avg=np.mean(age_pY)
    age_pY_sd=np.std(age_pY,ddof = 1)
    age_pY_avg_sd1=age_pY_avg+age_pY_sd
    age_pY_avg_ssd1=age_pY_avg-age_pY_sd

    age_pY_all.append(age_pY_avg)
    age_pY0_all.append(age_pY_avg_ssd1)
    age_pY1_all.append(age_pY_avg_sd1)
```

```
import matplotlib.pyplot as plt
plt.plot(age_grid,age_pY_all);
plt.plot(age_grid,age_pY0_all,linestyle=':');
plt.plot(age_grid,age_pY1_all,linestyle=':');
plt.ylim(-0.5,1.5);
plt.xlabel('Age');
plt.ylabel('Averaged ColRac');
plt.title('PDPs for Age in Random Forest');
plt.show();
```



From the PDP graph above for `Age`, we could see that in Random Forest model, `Age` is not a contributing predictor and does not contribute to predict the attitudes towards racist professor. No matter in which age category, the averaged probability of `Colrac` is almost the same, which means age could not distinguish different categories of `Colrac` and is irrelant with this response variable in the Random Forest model.

```
```