# Problem Set 5

Chia-Yun Chang

#### Conceptual: Cost functions for classification trees

For growing the tree, we want the tree to react to node impurity while not overfitting. Since classification error rate simply represent the fraction of the training data who does not share the most common value for a given attribute, i.e. misplaced obervation, it optimizes accuracy and does not classify according to node purity. Trying to maximize classification accuracy at each step may not end up selecting the accuracy-maximizing classifier overall. It might react to noise too much. Classification error rate, thus, would not be ideal for gorwing a tree. Gini index and cross entropy are scores that could reflect meaningful statistical learning results. Meaning that they classify base on the better odds. Gini index and cross entropy would be preferred for growing the tree, with cross entropy performing a little better (more sensitive to impurity), but this would also depend on what the data is. For the above mentioned reason, classification error rate will be ideal for pruning the tree, since it reflects accuracy.

#### Application: Predicting attitudes towards racist college professors

In [40]:
```
import pandas as pd
import numpy as np
from sklearn.linear_model import LogisticRegression, ElasticNetCV, Elast
icNet
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier,
GradientBoostingClassifier
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve, roc_auc_score, au
c
import matplotlib.pyplot as plt
from tabulate import tabulate
```

In [54]:
```
train = pd.read_csv("gss_train.csv")
test = pd.read_csv("gss_test.csv")
x = train.drop(['colrac'], axis=1)
#x_test = test.drop(['colrac'], axis=1)
x_test = test.loc[:, train.columns != 'colrac']
y = train['colrac']
y_test = test['colrac']
```

In [11]:
```
#LR
ac_score = []
lr = LogisticRegression(solver='liblinear')
grid = GridSearchCV(lr, scoring='accuracy', cv=10, param_grid = {})
fitlr = grid.fit(x, y)
ac_score = []
ac_score.append(fitlr.best_score_)
bestlr = fitlr.best_estimator_
```

In [14]:
```
#NB
nb = GaussianNB().fit(x, y)
grid = GridSearchCV(nb, scoring='accuracy', cv=10, param_grid={},)
fitnb = grid.fit(x, y)
ac_score.append(fitnb.best_score_)
bestnb = fitnb.best_estimator_
```

In [18]:
```
# Elastic net
en = ElasticNet()
parametersGrid = {"alpha": np.linspace(0.001, 0.01, 11),"l1_ratio": np.a
range(0.0, 1.0, 0.1)}
grid = GridSearchCV(en, parametersGrid, cv=10, refit=True)
fiten = grid.fit(x, y)
ac_score.append(fiten.best_score_)
best_en = fiten.best_estimator_
```

```
gap: 90.63678217269387, tolerance: 0.033067168674698846
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 93.11663058571901, tolerance: 0.03313667168674699
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 90.26610008865741, tolerance: 0.033117996987951814
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 91.73808531379969, tolerance: 0.033145105421686735
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 91.50665332681459, tolerance: 0.03317607223476299
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 91.71985872491838, tolerance: 0.033069224981188805
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 91.20214891725172, tolerance: 0.033135440180586895
  positive)
/Users/daphne/opt/anaconda3/lib/python3.7/site-packages/sklearn/linear_
model/_coordinate_descent.py:476: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality
gap: 92.54940260025582, tolerance: 0.03310158013544014
  positive)
```

In [23]:

```
#cart
ca = tree.DecisionTreeClassifier().fit(x, y)
param_grid = {'criterion': ['gini', 'entropy']}
grid = GridSearchCV(ca, param_grid, scoring = 'accuracy', cv=10)
fitca = grid.fit(x, y)
ac_score.append(fitca.best_score_)
best_ca = fitca.best_estimator_
```

In [24]:

```
#bagging
bag = BaggingClassifier()
param_grid = {'n_estimators': np.arange(10, 50, 10)}
grid = GridSearchCV(bag, param_grid, scoring = 'accuracy', cv=10)
fitbag = grid.fit(x, y)
ac_score.append(fitbag.best_score_)
best_bag = fitbag.best_estimator_
```

In [25]:
```
#random forest
rf = RandomForestClassifier()
param_grid = {'n_estimators': np.arange(100, 200, 10),'criterion': ['gin
i', 'entropy']}
grid = GridSearchCV(rf, param_grid, scoring = 'accuracy', cv=10)
fitrf = grid.fit(x, y)
ac_score.append(fitrf.best_score_)
best_rf = fitrf.best_estimator_
```

In [28]:
```
# boosting
bt = GradientBoostingClassifier()
param_grid = {'loss': ['deviance', 'exponential'],'learning_rate': np.ar
ange(0.1, 1, 0.1)}
grid = GridSearchCV(bt, param_grid, scoring = 'accuracy', cv=10)
fitbt = grid.fit(x, y)
ac_score.append(fitbt.best_score_)
best_bt = fitbt.best_estimator_
```

### Evaluate models

In [73]:
```
err_rate = [1- ac for ac in ac_score]
models = ['LR', 'NB', 'Elastic Net', 'CART', 'Bagginh', 'Random Forest',
'Boosting']
pr = []
for i in range(len(models)):
    pr.append([models[i],err_rate[i]])
pr[2][1] = -np.mean(cross_val_score(best_en, x, y,scoring='neg_mean_squa
red_error'))
print(tabulate(pr, headers = ['Model','Error rate']) )
```

```
Model            Error rate
-------------  ------------
LR                 0.207322
NB                 0.265577
Elastic Net        0.149207
CART               0.595151
Bagginh            0.595151
Random Forest      0.235463
Boosting           0.208002
```

In [74]:
```
pr = []
bests = [bestlr,bestnb, best_en, best_ca, best_bag, best_rf, best_bt]
for i in range(len(models)):
    ra = np.mean(cross_val_score(bests[i], x, y,scoring='roc_auc'))
    pr.append([models[i], ra])

print(tabulate(pr, headers = ['Model','Roc-Auc']) )
```

```
Model            Roc-Auc
-------------  ---------
LR              0.865978
NB              0.807522
Elastic Net     0.870687
CART            0.721708
Bagginh         0.865736
Random Forest   0.881668
Boosting        0.875281
```

### Best model

In terms of error rate, elastic net performs the best with the lowest error rate. In terms of roc-auc, random forest performs the best because it renders the largest roc-auc. If we judge collectively without giving weight to error rate or roc-auc, random forest is the best model overall.

In [77]:
```
best_rf.fit(x, y)
y_pred = best_rf.predict(x_test)
accuracy = accuracy_score(list(y_test),list( y_pred))
roc = roc_auc_score(y_test, y_pred)

print('Random Forest prediction accuracy:',accuracy)
print('Random Forest roc-auc scor:',roc)
```

```
Random Forest prediction accuracy: 0.8012170385395537
Random Forest roc-auc scor: 0.7930486593843098
```

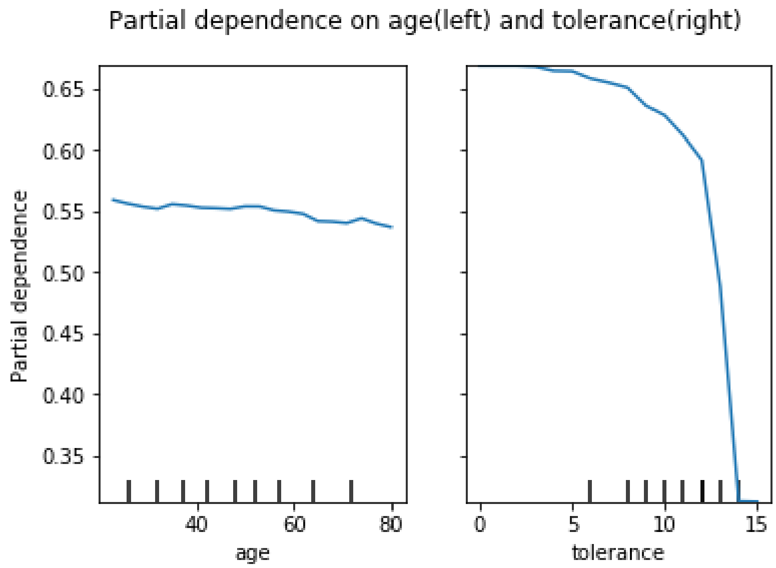
The prediction of random forest has an 0.8 accuracy, and 0.793 roc-auc score. This is pretty similar with the training data. We can then be rest asscured that the model could generalize pretty well and does not have the problem of overfit or being too complex.

### Bonus

In [107]:
```
from sklearn.inspection import partial_dependence
from sklearn.inspection import plot_partial_dependence

features = ['age', 'tolerance']

plot_partial_dependence(best_rf, x_test, features, n_jobs=3, grid_resolu
tion=20)
fig = plt.gcf()
fig.suptitle('Partial dependence on age(left) and tolerance(right)')
fig.subplots_adjust(hspace=0.3)
```




We can see from above that age does not have a strong influence on the predictions while tolerance has much more ostensible effect. The probability of being predicted as not allowed dropped while tolerance increase. This shows that racism could be more of a product of tolerance than of age.