In [20]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

1. Conceptual: cost function for classification trees

When growing trees, gini index and cross-entropy are more frequently used. The Gini index can measure the total amount of variance across all K classes to avoid overfitting. The cross-entropy shrinks when observations are more purely classified. These two methods are more sensitive to node purity.

When pruning trees, our goal is to select a subtree that leads to the lowest test error rate. Intuitively, we can use classification error rate to do that. Though classification error rate is prefered, we could also use gini index and cross-entropy to prune.

2. estimate the models & 3. compare and present model's performance

In [2]:

```
df_train = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-5-master/data/gss
_train.csv')
df_test = pd.read_csv('/Users/lijiaxuan/Downloads/problem-set-5-master/data/gss_
test.csv')
```

In [3]:

```
X_train, Y_train = df_train.loc[:, df_train.columns != 'colrac'],df_train['colra
c']
X_test, Y_test = df_test.loc[:, df_test.columns != 'colrac'],df_test['colrac']
print(X_train.shape,Y_train.shape,X_test.shape,Y_test.shape)
```

```
(1476, 55) (1476,) (493, 55) (493,)
```

In [10]:

```
print('sklearn: %s' % sklearn.__version__)
```

```
sklearn: 0.21.3
```

2.1. Logistic regression

In [46]:

```
from sklearn.linear_model import LogisticRegression
logreg=LogisticRegression(solver='lbfgs',max_iter = 2000)
print('logistic regression error rate',1-cross_val_score(logreg, X_train, Y_trai
n, cv=10,scoring='accuracy').mean())
print('logistic regression roc/auc',cross_val_score(logreg, X_train, Y_train, cv
=10,scoring='roc_auc').mean())
```

```
logistic regression error rate 0.20255765284528482
logistic regression roc/auc 0.8710162573844666
```

2.2. Naive Bayes

In [47]:

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
print('Naive Bayes error rate',1-cross_val_score(gnb, X_train, Y_train, cv=10,sc
oring='accuracy').mean())
print('Naive Bayes roc/auc',cross_val_score(gnb, X_train, Y_train, cv=10,scoring
='roc_auc').mean())
```

```
Naive Bayes error rate 0.26555250977590383
Naive Bayes roc/auc 0.8080500250922787
```

2.3.Elastic net regression

In [15]:

```
from sklearn.linear_model import ElasticNetCV
elas = ElasticNetCV(cv= 10).fit(X_train,Y_train)
#elas_mse = mean_squared_error(Y_test,elas.predict(X_test))
print(elas.l1_ratio_)
print(elas.alpha_)
```

```
0.5
0.0038452641680228584
```

2.4. Decision tree

In [40]:

```
best_elas = ElasticNetCV(alphas = [0.0038452641680228584],l1_ratio = 0.5,cv = 10
)
print('ElasticNet mse',-cross_val_score(best_elas,X_train,Y_train,cv = 10,scorin
g = 'neg_mean_squared_error').mean())
print('ElasticNet roc/auc',cross_val_score(best_elas,X_train,Y_train,cv = 10,sco
ring = 'roc_auc').mean())
```

```
ElasticNet mse 0.1471453221731916
ElasticNet roc/auc 0.8740225489138439
```

In [56]:

```
from sklearn.tree import DecisionTreeClassifier
param_grid = {
    'max_features':  ['auto', 'sqrt']
}
dt_grid = GridSearchCV(DecisionTreeClassifier(random_state=0),param_grid = param
_grid,cv = 10)
dt_grid.fit(X_train,Y_train)
```

Out[56]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=DecisionTreeClassifier(class_weight=None,
                                              criterion='gini', max_
depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=
0.0,
                                              min_impurity_split=Non
e,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_le
af=0.0,
                                              presort=False, random_
state=0,
                                              splitter='best'),
             iid='warn', n_jobs=None,
             param_grid={'max_features': ['auto', 'sqrt']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=None, verbose=0)
```

In [58]:

```
dt_grid.best_estimator_
```

Out[58]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_dept
h=None,
                       max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split
=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [59]:

```
best_dt = DecisionTreeClassifier(random_state=0,max_features = 'auto')
```

In [60]:

```
print('DecisionTree error rate',1-cross_val_score(best_dt, X_train, Y_train, cv=
10,scoring = 'accuracy').mean())
print('DecisionTree roc/auc',cross_val_score(best_dt, X_train, Y_train, cv=10,sc
oring = 'roc_auc').mean())
```

```
DecisionTree error rate 0.29748349911341276
DecisionTree roc/auc 0.7018232002739045
```

2.5. Bagging

In [24]:

```
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier

param_grid = {
    'base_estimator__max_depth' : [1, 2, 3, 4, 5],
    'max_samples' : [0.05, 0.1, 0.2, 0.5]
}

clf = GridSearchCV(BaggingClassifier(DecisionTreeClassifier(),
                                     n_estimators = 100, max_features = 0.5),
                   param_grid, scoring = 'accuracy',cv = 10)
clf.fit(X_train, Y_train)
```

Out[24]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=BaggingClassifier(base_estimator=DecisionTree
Classifier(class_weight=None,

criterion='gini',

max_depth=None,

max_features=None,

max_leaf_nodes=None,

min_impurity_decrease=0.0,

min_impurity_split=None,

min_samples_leaf=1,

min_samples_split=2,

min_weight_fraction_leaf=0.0,

presort=False,

random_state=...
                                         bootstrap=True,
                                         bootstrap_features=False,
                                         max_features=0.5, max_sampl
es=1.0,
                                         n_estimators=100, n_jobs=No
ne,
                                         oob_score=False, random_sta
te=None,
                                         verbose=0, warm_start=Fals
e),
             iid='warn', n_jobs=None,
             param_grid={'base_estimator__max_depth': [1, 2, 3, 4,
5],
                         'max_samples': [0.05, 0.1, 0.2, 0.5]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring='accuracy', verbose=0)
```

In [25]:

```
clf.best_params_
```

Out[25]:

```
{'base_estimator__max_depth': 5, 'max_samples': 0.2}
```

In [62]:

```
best_bag = BaggingClassifier(n_estimators = 5)
print('bagging error rate',1-cross_val_score(best_bag,X_train,Y_train,scoring = 'accuracy',cv = 10).mean())
print('bagging roc/auc',cross_val_score(best_bag,X_train,Y_train,scoring = 'roc_auc',cv = 10).mean())
```

```
bagging error rate 0.2445376790295295
bagging roc/auc 0.8275304306692636
```

2.6. Random forest

In [32]:

```
from sklearn.ensemble import RandomForestClassifier
max_depth = [int(x) for x in np.linspace(10, 100, num = 10)]
max_depth.append(None)
param_grid = {
    'n_estimators': [int(x) for x in np.linspace(start = 10, stop = 100, num = 1
0)],
    'max_depth' : max_depth,
    'max_features':  ['auto', 'sqrt']
}
rf_grid = GridSearchCV(RandomForestClassifier(),param_grid = param_grid,cv = 10)
rf_grid.fit(X_train,Y_train)
```

Out[32]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_
weight=None,
                                              criterion='gini', max_
depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=
0.0,
                                              min_impurity_split=Non
e,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_le
af=0.0,
                                              n_estimators='warn', n
_jobs=None,
                                              oob_score=False,
                                              random_state=None, ver
bose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_depth': [10, 20, 30, 40, 50, 60, 70, 8
0, 90, 100,
                                       None],
                         'max_features': ['auto', 'sqrt'],
                         'n_estimators': [10, 20, 30, 40, 50, 60, 7
0, 80, 90,
                                          100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=None, verbose=0)
```

In [33]:

```
rf_grid.best_params_
```

Out[33]:

```
{'max_depth': 90, 'max_features': 'auto', 'n_estimators': 40}
```

In [63]:

```
best_rf = RandomForestClassifier(n_estimators = 40,max_depth = 90, max_features
= 'auto')
print('randomforest error rate',1-cross_val_score(best_rf,X_train,Y_train,scorin
g = 'accuracy',cv = 10).mean())
print('randomforest roc/auc',cross_val_score(best_rf,X_train,Y_train,scoring =
'roc_auc',cv = 10).mean())
```

```
randomforest error rate 0.20327910776137048
randomforest roc/auc 0.8771145772403319
```

2.7. Boosting

In [37]:

```
from sklearn.ensemble import GradientBoostingClassifier
param_grid = {
    'n_estimators': [int(x) for x in np.linspace(start = 10, stop = 100, num = 1
0)],
    'max_features':  ['auto', 'sqrt']
}

bst = GridSearchCV(GradientBoostingClassifier(),param_grid = param_grid,cv=10)
bst.fit(X_train,Y_train)
```

Out[37]:

```
GridSearchCV(cv=10, error_score='raise-deprecating',
             estimator=GradientBoostingClassifier(criterion='friedma
n_mse',
                                                  init=None, learnin
g_rate=0.1,
                                                  loss='deviance', m
ax_depth=3,
                                                  max_features=None,
                                                  max_leaf_nodes=Non
e,
                                                  min_impurity_decre
ase=0.0,
                                                  min_impurity_split
=None,
                                                  min_samples_leaf=
1,
                                                  min_samples_split=
2,
                                                  min_weight_fractio
n_leaf=0.0,
                                                  n_estimators=100,
                                                  n_iter_no_change=N
one,
                                                  presort='auto',
                                                  random_state=None,
                                                  subsample=1.0, tol
=0.0001,
                                                  validation_fractio
n=0.1,
                                                  verbose=0, warm_st
art=False),
             iid='warn', n_jobs=None,
             param_grid={'max_features': ['auto', 'sqrt'],
                         'n_estimators': [10, 20, 30, 40, 50, 60, 7
0, 80, 90,
                                          100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score
=False,
             scoring=None, verbose=0)
```

In [38]:

```
bst.best_params_
```

Out[38]:

```
{'max_features': 'auto', 'n_estimators': 70}
```

In [64]:

```
best_bst = GradientBoostingClassifier(max_features = 'auto',n_estimators = 70)
print('boosting error rate',1-cross_val_score(best_bst,X_train,Y_train,scoring =
'accuracy',cv = 10).mean())
print('boosting roc/auc',cross_val_score(best_bst,X_train,Y_train,scoring = 'roc
_auc',cv = 10).mean())
```

```
boosting error rate 0.19174659524611593
boosting roc/auc 0.8811281374561053
```

4. Which is the best model? Defend your choice.

The best model is the boosting classifier because this model has the smallest error rate and the highest roc/auc score.

5. Evaluate the best model

The error rate increases a little in the test set compared with the training set, and the roc/auc score is lower in the test set than in the training set. However, generally speakingly, this boosting model generalize very well in the test set because the drops of accuracy and roc score are not too much.

In [42]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
best_bst.fit(X_train,Y_train)
print('error rate', 1 - accuracy_score(Y_test,best_bst.predict(X_test)))
print('roc/auc', roc_auc_score(Y_test,best_bst.predict(X_test)))
```

```
error rate 0.20081135902636915
roc/auc 0.7923866269447203
```

6. Bonus: PDPs/ICE

In [68]:

```
print(X_train.columns)
```

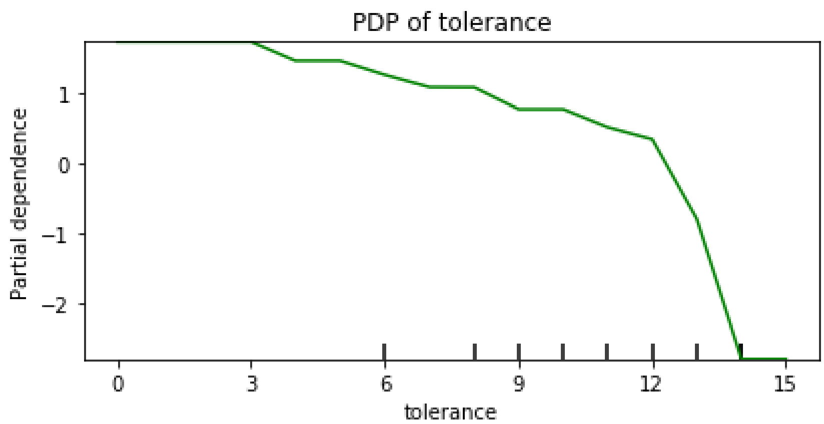
```
Index(['age', 'attend', 'authoritarianism', 'black', 'born', 'child
s',
       'colath', 'colcom', 'colmil', 'colhomo', 'colmslm', 'con_gov
t',
       'egalit_scale', 'evangelical', 'grass', 'happy', 'hispanic_
2',
       'homosex', 'income06', 'mode', 'owngun', 'polviews', 'pornlaw
2', 'pray',
       'pres08', 'reborn_r', 'science_quiz', 'sex', 'sibs', 'social_
connect',
       'south', 'teensex', 'tolerance', 'tvhours', 'vetyears', 'word
sum',
       'degree_Bachelor.deg', 'degree_other', 'marital_Divorced',
       'marital_Never.married', 'marital_other', 'news_FEW.TIMES.A.W
EEK',
       'news_LESS.THAN.ONCE.WK', 'news_NEVER', 'news_other', 'partyi
d_3_Ind',
       'partyid_3_Rep', 'relig_CATHOLIC', 'relig_NONE', 'relig_othe
r',
       'social_cons3_Mod', 'social_cons3_Conserv', 'spend3_Mod',
       'spend3_Liberal', 'zodiac_other'],
      dtype='object')
```

In [74]:

```
from sklearn.inspection import plot_partial_dependence
plot_partial_dependence(best_bst.fit(X_train,Y_train),X_train,[32])
plt.title('PDP of tolerance')
plt.xlabel('tolerance')
```

Out[74]:

```
Text(0.5, 0, 'tolerance')
```

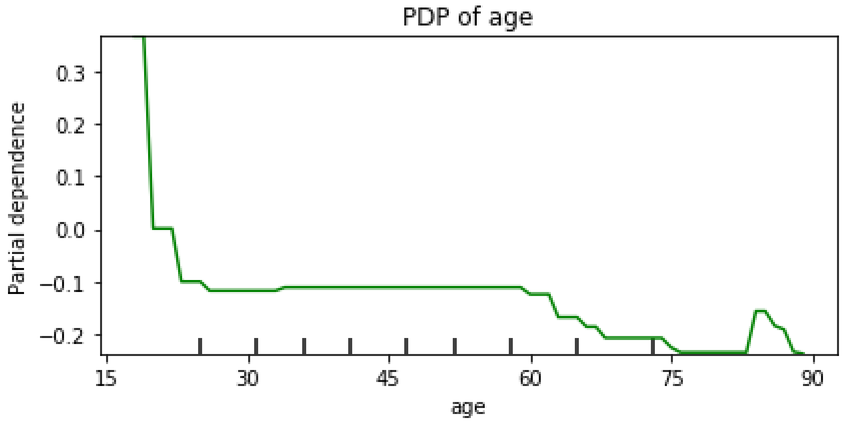


In [77]:

```
plot_partial_dependence(best_bst.fit(X_train,Y_train),X_train,[0])
plt.title('PDP of age')
plt.xlabel('age')
```

Out[77]:

```
Text(0.5, 0, 'age')
```



Tolerance: It seems that the impact of tolerance shifting from positive to negative as tolerance level increase. When tolerance level is less than 13, the tolerance has a positive effect on colrac; when tolerance level reaches above 13, the tolerance has a negative effect on colrac. Age: The impact of age on colrac also shifts from positive to negative as age increases. When age is less than 20, the positive correlation decreases quickly from 0.3 to 0, and it remains steady at around -0.1 between 20 to 60. After 60 years old, the age is becoming more negatively correlated with colrac as age increase, with a slight fluctuation at the 80 years old. Overall, the age has a smaller effect on colrac as compared to tolerance.