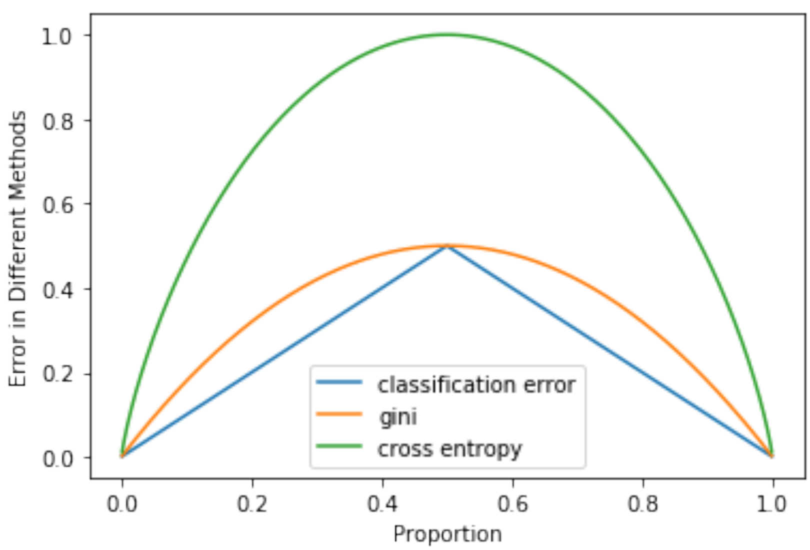
Conceptual: Cost functions for classification trees

1.(15 points) Consider the Gini index, classification error, and cross-entropy in simple classification settings with two classes. Of these three possible cost functions, which would be best to use when growing a decision tree? Which would be best to use when pruning a decision tree? Why?

In [31]:
```
p = np.linspace(0, 1, num=1000)[1:-1]
clf_err = [1 - max(i, 1-i) for i in p]
gini_err = [2*i*(1-i) for i in p]
ce_err = [-(i*np.log2(i)+(1-i)*np.log2(1-i)) for i in p]

plt.plot(p,clf_err,label='classification error')
plt.plot(p,gini_err,label='gini')
plt.plot(p,ce_err,label='cross entropy')
plt.ylabel('Error in Different Methods')
plt.xlabel('Proportion')
plt.legend()
plt.show()
```



When growing a decision tree, gini index and cross-entropy would be best, which are often used to measure the purity of the classification.These two methods are more sensitive to node purity. The Gini index can control the variance across all K classes to avoid the problem of overfitting. The cross-entropy is more sensitive to node impurity and allow us to split when observations are more purely classified.

When pruning a decision tree, classification error would be best, which is used to measure the accuracy.Because we want to maximize the prediction accuracy when doing pruning, the classification error is a better choice.

In [98]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression,ElasticNet, SGDCla
ssifier
from sklearn.naive_bayes import GaussianNB
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier
, GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_validate
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn.metrics import roc_auc_score, roc_curve, auc, accuracy_sc
ore
from sklearn.inspection import plot_partial_dependence
import warnings
warnings.filterwarnings('ignore')
```

Estimate the models

2.(35 points; 5 points/model) Estimate the following models, predicting colrac using the training set (the training .csv) with 10-fold CV:

In [20]:
```
train = pd.read_csv("gss_train.csv")
test = pd.read_csv("gss_test.csv")
x_train = train.drop('colrac', axis=1)
y_train = train.colrac
x_test = test.drop('colrac', axis=1)
y_test = test.colrac
```

In [67]:
```
#Logistic Regression Model
lr_estimate = LogisticRegression()
lr_estimate.fit(x_train, y_train)
```

Out[67]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_interce
pt=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, ver
bose=0,
                   warm_start=False)
```

In [42]:
```
lr_pred = cross_val_predict(LogisticRegression(solver='liblinear'),
                            x_train, y_train, cv=10)
lr_pred
```

Out[42]:
```
array([1, 0, 0, ..., 1, 0, 1])
```

In [68]:
```
lr=LogisticRegression()
print('logistic regression error rate',1-cross_val_score(lr, x_train,
y_train, cv=10,scoring='accuracy').mean())
print('logistic regression roc/auc',cross_val_score(lr, x_train, y_tra
in, cv =10,scoring='roc_auc').mean())
```

```
logistic regression error rate 0.20731955760718945
logistic regression roc/auc 0.8703107556427476
```

In [44]:
```
#Naive Bayes
nb = GaussianNB()
nb.fit(x_train, y_train)
```

Out[44]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [45]:
```
nb_pred = cross_val_predict(GaussianNB(), x_train, y_train, cv = 10)
nb_pred
```

Out[45]:
```
array([1, 0, 1, ..., 1, 0, 1])
```

In [69]:
```
nb = GaussianNB()
print('Naive Bayes error rate',1-cross_val_score(nb, x_train, y_train,
cv=10,scoring='accuracy').mean())
print('Naive Bayes roc/auc',cross_val_score(nb, x_train, y_train, cv=1
0,scoring ='roc_auc').mean())
```

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

In [65]:
```
#Elastic Net Logistic Regression
elastic = ElasticNetCV(cv=10, random_state=0).fit(x_train, y_train)
print('best alpha is ', elastic.alpha_)
print('best l1_ratio is ', elastic.l1_ratio_)
```

```
best alpha is  0.0038452641680228584
best l1_ratio is  0.5
```

In [78]:
```
#Decision Tree
parameters = {
    "min_samples_split": [0.1, 0.2, 0.3, 0.4, 0.5],
    "min_samples_leaf": [0.1, 0.2, 0.3, 0.4, 0.5],
    "max_depth":range(1,20),
    }

dt = GridSearchCV(DecisionTreeClassifier(), parameters, cv = 10).fit(x
_train, y_train)
best_cart_model = dt.best_estimator_
print(dt.best_score_, dt.best_params_)
dt_error = 1-np.mean(cross_val_score(dt, x_train, y_train, cv=10, scor
ing='accuracy'))
print("Decision Tree 10-fold cross-validated classification error:", d
t_error)
```

```
0.7655826558265583 {'max_depth': 4, 'min_samples_leaf': 0.1, 'min_sa
mples_split': 0.1}
Decision Tree 10-fold cross-validated classification error: 0.245994
13137284654
```

In [79]:
```
#Bagging
parameters = {
    "base_estimator": [DecisionTreeClassifier()],
    "n_estimators": range(10,50,5)
    }
bag = GridSearchCV(BaggingClassifier(), parameters, cv = 10).fit(x_tra
in, y_train)
best_bag_model = bag.best_estimator_
print(bag.best_score_, bag.best_params_)
```

```
0.7899728997289973 {'base_estimator': DecisionTreeClassifier(class_w
eight=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=None, splitter='best'), 'n_estim
ators': 25}
```

In [80]:
```
#Random Forest
parameters = {
    'n_estimators': range(10,50,5),
    "max_depth":range(1,20),
    }
rf = GridSearchCV(RandomForestClassifier(), parameters, cv = 10).fit(x
_train, y_train)
best_rf_model = rf.best_estimator_
print(rf.best_score_, rf.best_params_)
```

```
0.8055555555555556 {'max_depth': 17, 'n_estimators': 30}
```

In [81]:
```
#Boosting
parameters = {
    'n_estimators': [20, 30, 40, 50],
    "max_depth": [11, 13, 15],
    "learning_rate": [0.025, 0.2]
    }

boost = GridSearchCV(GradientBoostingClassifier(), parameters, cv = 10
).fit(x_train, y_train)
best_boost_model = boost.best_estimator_
print(boost.best_score_, boost.best_params_)
```

```
0.7540650406504065 {'learning_rate': 0.2, 'max_depth': 11, 'n_estima
tors': 40}
```

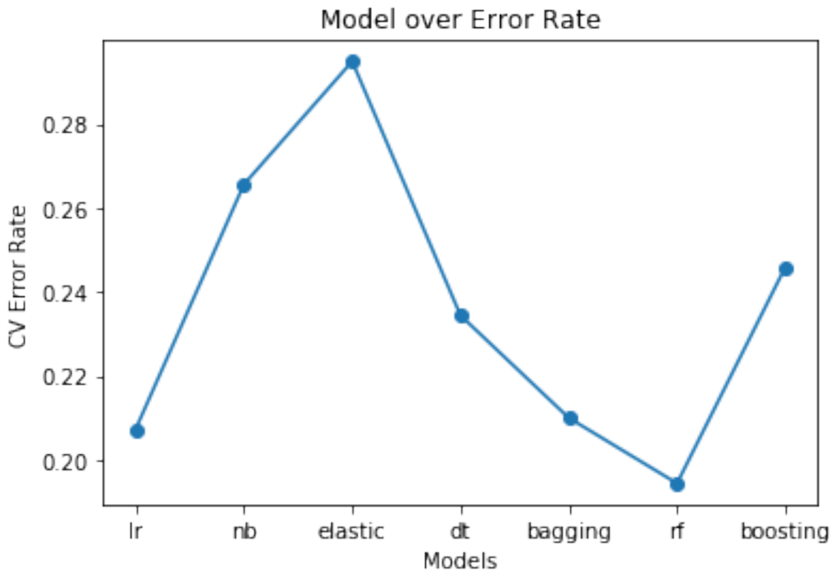
Evaluate the models

3.(20 points) Compare and present each model's (training) performance based on Cross-validated error rate and ROC/AUC

In [89]:
```
best_lr = LogisticRegression(solver='liblinear').fit(x_train, y_train)
best_nb = GaussianNB().fit(x_train, y_train)
best_elastic = SGDClassifier(alpha = elastic.alpha_,
                        l1_ratio = elastic.l1_ratio_).fit(x_train, y_t
rain)
err_lr = 1 - cross_val_score(LogisticRegression(solver='liblinear'),
                          x_train, y_train, cv = 10).mean()
err_nb = 1 - cross_val_score(GaussianNB(), x_train, y_train, cv = 10).
mean()
err_elastic = 1 - cross_val_score(best_elastic, x_train, y_train, cv =
10).mean()
print('lr', round(err_lr, 4))
print('Naive Bayes', round(err_nb, 4))
print('Elastic Net', round(err_elastic, 4))
print('Cart', round(1 - dt.best_score_, 4))
print('Bagging', round(1 - bag.best_score_, 4))
print('Random Forest', round(1 - rf.best_score_, 4))
print('Boosting', round(1 - boost.best_score_, 4))
```

```
lr 0.2073
Naive Bayes 0.2656
Elastic Net 0.2949
Cart 0.2344
Bagging 0.21
Random Forest 0.1944
Boosting 0.2459
```

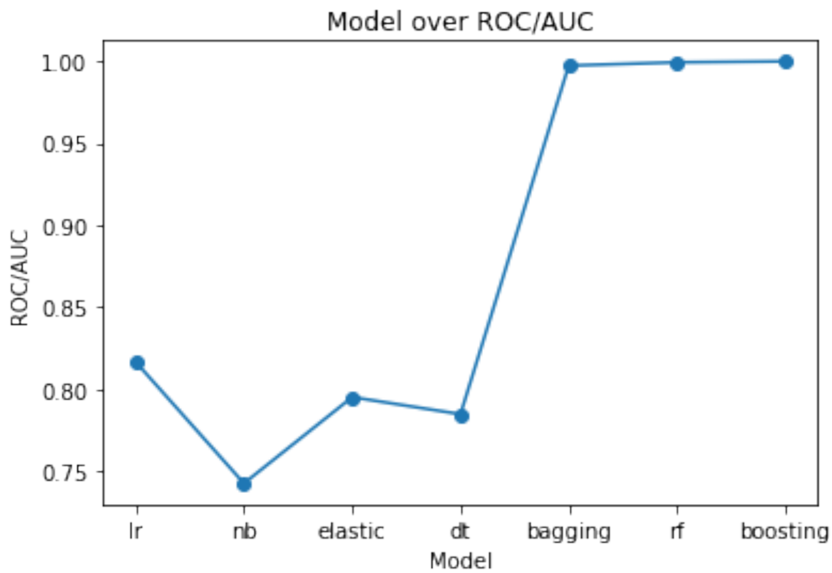
In [90]:
```
error_rate = [0.2073,0.2656,0.2949,0.2344,0.21,0.1944,0.2459]
model = ['lr', 'nb', 'elastic', 'dt', 'bagging', 'rf', 'boosting']
plt.plot(model, error_rate, marker='o')
plt.xlabel('Models')
plt.ylabel('CV Error Rate')
plt.title('Model over Error Rate');
```



In [91]:
```
pred_lr = LogisticRegression(solver='liblinear').fit(x_train, y_train)
.predict(x_train)
pred_nb = GaussianNB().fit(x_train, y_train).predict(x_train)
pred_elastic = best_elastic.fit(x_train, y_train).predict(x_train)
roc = roc_auc_score(pred_lr, y_train)
print('lr', round(roc, 4))
roc = roc_auc_score(pred_nb, y_train)
print('Naive Bayes', round(roc, 4))
roc = roc_auc_score(pred_elastic.astype(int), y_train)
print('Elastic Net', round(roc, 4))
roc = roc_auc_score(dt.predict(x_train), y_train)
print('Cart', round(roc, 4))
roc = roc_auc_score(bag.predict(x_train), y_train)
print('Bagging', round(roc, 4))
roc = roc_auc_score(rf.predict(x_train), y_train)
print('Random Forest', round(roc, 4))
roc = roc_auc_score(boost.predict(x_train), y_train)
print('Boosting', round(roc, 4))
```

```
lr 0.8166
Naive Bayes 0.7425
Elastic Net 0.7951
Cart 0.7847
Bagging 0.9974
Random Forest 0.9994
Boosting 1.0
```

In [93]:
```
roc_auc = [0.8166,0.7425,0.7951,0.7847,0.9974,0.9994,1.0]
model = ['lr', 'nb', 'elastic', 'dt', 'bagging',
         'rf', 'boosting']
plt.plot(model, roc_auc, marker='o')
plt.xlabel('Model')
plt.ylabel('ROC/AUC')
plt.title('Model over ROC/AUC');
```



4.(15 points) Which is the best model? Defend your choice. Random Forest is the best model. Because it has the lowest cross-validated error rate and is also one of the models that get the highest ROC/AUC score.Boosting and Elastic Net are also good models.

Evaluate the best model

5.(15 points) Evaluate the final, best model's (selected in 4) performance on the test set (the test .csv) by calculating and presenting the classification error rate and AUC. Compared to the fit evaluated on the training set in questions 3-4, does the "best" model generalize well? Why or why not? How do you know?

In [96]:
```
#The best model is Random Forest
pred = rf.predict(x_train)
error = sum(pred != y_train)/ len(y_train)
error
pred = rf.predict(x_test)
error = sum(pred != y_test)/ len(y_test)
roc = roc_auc_score(pred, y_test)
print('classification error', error)
print('ROC/AUC', roc)
```

```
classification error 0.20486815415821502
ROC/AUC 0.8047385047385048
```

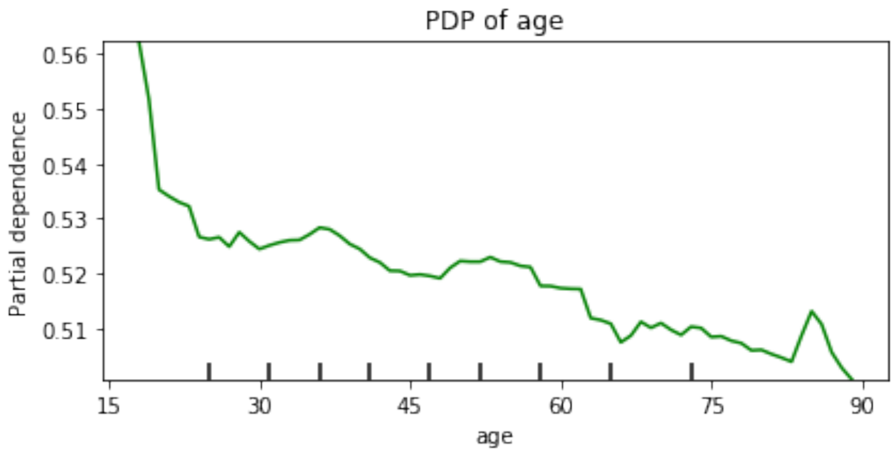
Compared to the fit evaluated on the training set, the classification error of testing set is 0.2 (while the figure for the train data is 0). ROC/AUC decreases is 0.8(while the figure for the train data is 1.0). Therefore, the generalization is not perfect, because the model may be overfitting.

Bonus: PDPs/ICE

6.(Up to 5 extra points) Present and substantively interpret the "best" model (selected in question 4) using PDPs/ICE curves over the range of: tolerance and age. Note, interpretation must be more than simple presentation of plots/curves. You must sufficiently describe the changes in probability estimates over the range of these two features. You may earn up to 5 extra points, where partial credit is possible if the solution is insufficient along some dimension (e.g., technically/code, interpretation, visual presentation, etc.).

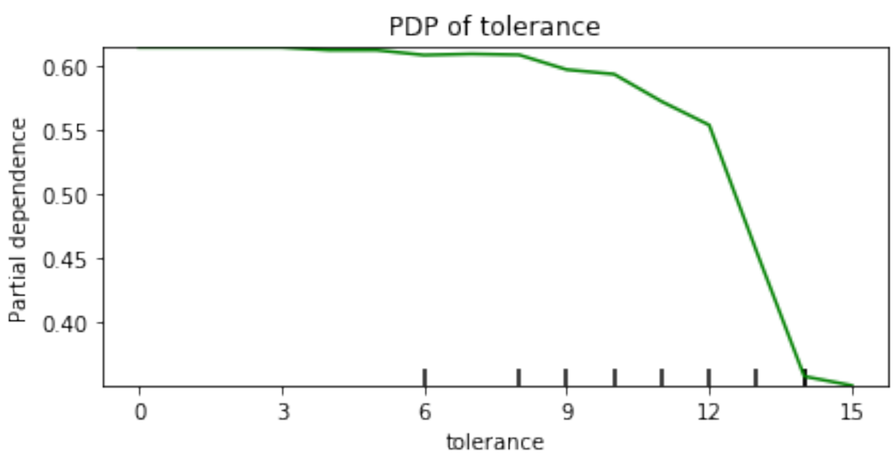
In [102]:
```
plot_partial_dependence(rf.fit(x_train,y_train),x_train,[0])
plt.title('PDP of age')
plt.xlabel('age')
```

Out[102]: Text(0.5, 0, 'age')



In [103]:
```
plot_partial_dependence(rf.fit(x_train,y_train),x_train,[32])
plt.title('PDP of tolerance')
plt.xlabel('tolerance')
```

Out[103]: Text(0.5, 0, 'tolerance')



According to the above graph,as age goes up, its partial dependence goes down;as tolerance goes up, its partial dependence goes down more quickly compared to age. Therefore, we could say tolerance is a relatively better predictor for colrac.