# Homework 5

March 1, 2020

## 0.1 Homework 5

[1]:
```python
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from pylab import rcParams
import seaborn as sns
import pandas as pd
import sklearn
import warnings
import itertools
import sklearn.metrics
warnings.filterwarnings('ignore')
```

### 0.1.1 Q1

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?

Though in many cases, the choice of impurity measurement does not produce much difference on the splitting result, we are more likely to use Gini index or cross-entropy when growing trees. If we using classification error, supposing we have cut the dataset into a larger set A and a smaller set B, then for any potential next cut point (making subset C and D), if the proportion for A's observations in C and D are both larger than that of B's observations in C and D, the information gain would be zero, preventing further splitting. (See an example) But such situation would not happen to Gini index and cross entropy because they are more sensitive to the variation of splitted proportions in subsets. So when we grow trees, we are more likely to use Gini index or cross-entropy. Some people claim that cross_entropy is a bit more computationally intensive because of the logarithmic calculation, so they think the Gini index is the best.

When prunning the tree, the general criterion is to choose the measurement according to your goal. Generally, classification error is preferable if prediction accuracy of the final pruned tree is the goal, and it would not bring problems like overfitting after pruning.

### 0.1.2 Q2

Estimate the following models, predicting colrac using the training set (the training .csv) with 10-fold CV

[9]:
```python
from sklearn.model_selection import GridSearchCV

# load data
gss_train = pd.read_csv('data/gss_train.csv')
gss_test = pd.read_csv('data/gss_test.csv')
gss_train_features = gss_train.drop(columns='colrac')
gss_test_features = gss_test.drop(columns='colrac')
```

[19]:
```python
# logistic Regression with regularization
from sklearn.linear_model import LogisticRegression
LR_clf = LogisticRegression(n_jobs = -1)
LR_paras_grid = {'C':np.linspace(0,100,51), 'penalty':['elasticnet'],
'solver': ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'],
'tol':10**(np.linspace(-10,-4,7)), 'l1_ratio':np.linspace(0,1,11)}
LR_optimized = GridSearchCV(LR_clf,
LR_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[20]:
```python
# logistic Regression result
print('Best logistic Regression Training Accuracy:', LR_optimized.best_score_)
print('Best logistic Regression Training Parameters:', LR_optimized.
 ↪best_params_)
```

```
Best logistic Regression Training Accuracy: 0.7906646442360727
Best logistic Regression Training Parameters: {'C': 4.0, 'l1_ratio': 0.2,
'penalty': 'elasticnet', 'solver': 'saga', 'tol': 1e-05}
```

[55]:
```python
# Naive Bayes with smoothing
from sklearn.naive_bayes import MultinomialNB
NB_clf = MultinomialNB()
NB_paras_grid = {'alpha':np.linspace(0.0,100,51)}
NB_optimized = GridSearchCV(NB_clf,
NB_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[57]:
```python
# Naive Bayes result
print('Best Naive Bayes Training Accuracy:', NB_optimized.best_score_)
print('Best Naive Bayes Training Parameters:', NB_optimized.best_params_)
```

```
Best Naive Bayes Training Accuracy: 0.6944612980327266
Best Naive Bayes Training Parameters: {'alpha': 0.0}
```

[58]:
```python
# Elastic Net Regression
from sklearn.linear_model import ElasticNet
ENR_clf = ElasticNet(normalize = True)
```

```
ENR_paras_grid = {'alpha':np.linspace(0,100,51),
'l1_ratio':np.linspace(0,1,11),
'tol':10**(np.linspace(-10,-4,7))}
ENR_optimized = GridSearchCV(ENR_clf,
ENR_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[59]:
```
# Elastic Net Regression result
print('Best Elastic Net Training Accuracy:', ENR_optimized.best_score_)
print('Best Elastic Net Training Parameters:', ENR_optimized.best_params_)
```

```
Best Elastic Net Training Accuracy: 0.3985564825083226
Best Elastic Net Training Parameters: {'alpha': 0.0, 'l1_ratio': 0.0, 'tol':
1e-10}
```

[69]:
```
# Desicion Tree
from sklearn.tree import DecisionTreeClassifier
Tree_clf = DecisionTreeClassifier()

Tree_paras_grid = [{'criterion':["gini", "entropy"],
'max_depth':np.linspace(0,20,21),
'min_samples_split':np.linspace(2,52,51)},
{'criterion':["gini", "entropy"],
'max_depth':np.linspace(0,20,21),
'min_samples_leaf':np.linspace(0,1,11)},
{'criterion':["gini", "entropy"],
'max_depth':np.linspace(0,20,21),
'max_leaf_nodes':np.arange(2,len(gss_train),5)},
{'criterion':["gini", "entropy"],
'max_depth':np.linspace(0,20,21),
'ccp_alpha':np.linspace(0,0.5,11)}]

Tree_optimized = GridSearchCV(Tree_clf,
Tree_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[71]:
```
# Decision Tree result
print('Best Decision Tree Training Accuracy:', Tree_optimized.best_score_)
print('Best Decision Tree Training Parameters:', Tree_optimized.best_params_)
```

```
Best Decision Tree Training Accuracy: 0.7811270454127597
Best Decision Tree Training Parameters: {'criterion': 'gini', 'max_depth': 4.0,
'max_leaf_nodes': 12}
```

[75]:
```
# Bagging
from sklearn.ensemble import BaggingClassifier
Bag_clf = BaggingClassifier(n_jobs=-1)
Bag_paras_grid = {'n_estimators':[int(x) for x in np.linspace(2,20,19)],
'max_samples':np.linspace(0.1,1.0,10), 'oob_score':[True,False]}
```

```
Bag_optimized = GridSearchCV(Bag_clf,
Bag_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[76]:
```
# Bagging result
print('Best Bagging Training Accuracy:', Bag_optimized.best_score_)
print('Best Bagging Training Parameters:', Bag_optimized.best_params_)
```

```
Best Bagging Training Accuracy: 0.7980649016363301
Best Bagging Training Parameters: {'max_samples': 0.7000000000000001,
'n_estimators': 15, 'oob_score': True}
```

[87]:
```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
RF_clf = RandomForestClassifier(n_jobs=-1, oob_score=True)
RF_paras_grid = {'n_estimators':[int(x) for x in np.linspace(50,200,4)],
'max_features': np.linspace(0.1,1.0,10),
'max_depth':np.linspace(2,20,10)}
RF_optimized = GridSearchCV(RF_clf,
RF_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[88]:
```
# Random Forest result
print('Best Random Forest Training Accuracy:', RF_optimized.best_score_)
print('Best Random Forest Parameters:', RF_optimized.best_params_)
```

```
Best Random Forest Training Accuracy: 0.806218974076117
Best Random Forest Parameters: {'max_depth': 10.0, 'max_features': 0.2,
'n_estimators': 50}
```

[90]:
```
# Boosting
from sklearn.ensemble import GradientBoostingClassifier
Boost_clf = GradientBoostingClassifier()
Boost_paras_grid = {'n_estimators':[int(x) for x in np.linspace(50,200,5)],
'learning_rate':[0.001,0.01]+ list(np.linspace(0.1,1.0,10)),
'max_depth':np.linspace(2,10,9)}
Boost_optimized = GridSearchCV(Boost_clf,
Boost_paras_grid, n_jobs=-1, cv=10).fit(gss_train_features, gss_train['colrac'])
```

[91]:
```
# Boosting result
print('Best Boosting Training Accuracy:', Boost_optimized.best_score_)
print('Best Boosting Training Parameters:', Boost_optimized.best_params_)
```

```
Best Boosting Training Accuracy: 0.8021465342893913
Best Boosting Training Parameters: {'learning_rate': 0.30000000000000004,
'max_depth': 2.0, 'n_estimators': 50}
```

### 0.1.3 Q3

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

```
[104]: from sklearn.metrics import roc_auc_score
       from sklearn.metrics import mean_absolute_error

       performance_df = pd.DataFrame(columns = ['Classifer',
       'training CV error', 'training AUC-ROC'])
       # Logistic Regression
       performance_df.loc[len(performance_df)] = ['Logistic Regression',
       1-LR_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       LR_optimized.predict(gss_train_features))]
       # Naive Bayes
       performance_df.loc[len(performance_df)] = ['Naive Bayes',
       1-NB_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       NB_optimized.predict(gss_train_features))]
       # Elastic Net Regression
       performance_df.loc[len(performance_df)] = ['Elastic Net',
       mean_absolute_error(gss_train['colrac'],
       ENR_optimized.predict(gss_train_features)),
       roc_auc_score(gss_train['colrac'],
       ENR_optimized.predict(gss_train_features))]
       # Decision Tree
       performance_df.loc[len(performance_df)] = ['Decision Tree',
       1-Tree_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       Tree_optimized.predict(gss_train_features))]
       # Bagging
       performance_df.loc[len(performance_df)] = ['Bagging',
       1-Bag_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       Bag_optimized.predict(gss_train_features))]
       # Random Forest
       performance_df.loc[len(performance_df)] = ['Random Forest',
       1-RF_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       RF_optimized.predict(gss_train_features))]
       # Boosting
       performance_df.loc[len(performance_df)] = ['Boosting',
       1-Boost_optimized.best_score_, roc_auc_score(gss_train['colrac'],
       Boost_optimized.predict(gss_train_features))]

       performance_df
```

```
[104]:             Classifer  training CV error  training AUC-ROC
       0  Logistic Regression           0.209335          0.807885
       1          Naive Bayes           0.305539          0.699519
       2          Elastic Net           0.298442          0.893726
```

```
3        Decision Tree          0.218873          0.790311
4              Bagging          0.201935          0.975276
5        Random Forest          0.193781          0.985226
6             Boosting          0.197853          0.859193
```

### 0.1.4 Q4

Which is the best model? Defend your choice.

[105]:
```
performance_df.sort_values(by=['training CV error'])
```

[105]:
```
             Classifer  training CV error  training AUC-ROC
5        Random Forest          0.193781          0.985226
6             Boosting          0.197853          0.859193
4              Bagging          0.201935          0.975276
0  Logistic Regression          0.209335          0.807885
3        Decision Tree          0.218873          0.790311
2          Elastic Net          0.298442          0.893726
1          Naive Bayes          0.305539          0.699519
```

[106]:
```
performance_df.sort_values(by=['training AUC-ROC'], ascending = False)
```

[106]:
```
             Classifer  training CV error  training AUC-ROC
5        Random Forest          0.193781          0.985226
4              Bagging          0.201935          0.975276
2          Elastic Net          0.298442          0.893726
6             Boosting          0.197853          0.859193
0  Logistic Regression          0.209335          0.807885
3        Decision Tree          0.218873          0.790311
1          Naive Bayes          0.305539          0.699519
```

Based on the performance of the average cross_validation error and the AUC-ROC, we can see that the random forest classifer is the best one after tunning. The random forest classifier achieves the lowest cross validation error as well as the highest AUC-ROC, which reveals the effectiveness of the addition of picking random attributes and bootstraping. Also, we can see that ensemble models like bagging and boosting models also yields better performance than others. Interestingly, the elastic net model also has a high AUC-ROC but the average error for it is not low. Though we can not directly compare this with other classifiers (because regressors and classifiers use different metrics), which suggest that the classifier might be good at capture postive class but can hardly distinguish the negative class.

### 0.1.5 Q5

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?

[101]:
```
from sklearn.metrics import accuracy_score
best_model = RF_optimized.best_estimator_
# testing on the test set
print('best model test error rate:',
1-accuracy_score(gss_test['colrac'],
best_model.predict(gss_test_features)))

print('best model AUC-ROC:',
roc_auc_score(gss_test['colrac'],
best_model.predict(gss_test_features)))
```

```
best model test error rate: 0.2068965517241379
best model AUC-ROC: 0.7845829195630586
```

[107]:
```
# Logistic Regression
performance_df.loc[0,'test error']=1-accuracy_score(gss_test['colrac'],
LR_optimized.predict(gss_test_features))
performance_df.loc[0,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
LR_optimized.predict(gss_test_features))
# Naive Bayes
performance_df.loc[1,'test error']=1-accuracy_score(gss_test['colrac'],
NB_optimized.predict(gss_test_features))
performance_df.loc[1,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
NB_optimized.predict(gss_test_features))
# Elastic Net Regression
performance_df.loc[2,'test error']=mean_absolute_error(gss_test['colrac'],
ENR_optimized.predict(gss_test_features))
performance_df.loc[2,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
ENR_optimized.predict(gss_test_features))
# Decision Tree
performance_df.loc[3,'test error']=1-accuracy_score(gss_test['colrac'],
Tree_optimized.predict(gss_test_features))
performance_df.loc[3,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
Tree_optimized.predict(gss_test_features))
# Bagging
performance_df.loc[4,'test error']=1-accuracy_score(gss_test['colrac'],
Bag_optimized.predict(gss_test_features))
performance_df.loc[4,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
Bag_optimized.predict(gss_test_features))
# Random Forest
performance_df.loc[5,'test error']=1-accuracy_score(gss_test['colrac'],
RF_optimized.predict(gss_test_features))
performance_df.loc[5,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
RF_optimized.predict(gss_test_features))
# Boosting
```

```
performance_df.loc[6,'test error']=1-accuracy_score(gss_test['colrac'],
Boost_optimized.predict(gss_test_features))
performance_df.loc[6,'test AUC-ROC']=roc_auc_score(gss_test['colrac'],
Boost_optimized.predict(gss_test_features))

performance_df
```

[107]:
```
             Classifer  training CV error  training AUC-ROC  test error  \
0  Logistic Regression           0.209335          0.807885    0.210953
1          Naive Bayes           0.305539          0.699519    0.292089
2          Elastic Net           0.298442          0.893726    0.314324
3        Decision Tree           0.218873          0.790311    0.235294
4              Bagging           0.201935          0.975276    0.233266
5        Random Forest           0.193781          0.985226    0.206897
6             Boosting           0.197853          0.859193    0.202840

   test AUC-ROC
0      0.785402
1      0.707175
2      0.856902
3      0.752963
4      0.760667
5      0.784583
6      0.790500
```

[111]:
```
performance_df.sort_values(by=['test error'])[:-1]
```

[111]:
```
             Classifer  training CV error  training AUC-ROC  test error  \
6             Boosting           0.197853          0.859193    0.202840
5        Random Forest           0.193781          0.985226    0.206897
0  Logistic Regression           0.209335          0.807885    0.210953
4              Bagging           0.201935          0.975276    0.233266
3        Decision Tree           0.218873          0.790311    0.235294
1          Naive Bayes           0.305539          0.699519    0.292089

   test AUC-ROC
6      0.790500
5      0.784583
0      0.785402
4      0.760667
3      0.752963
1      0.707175
```

[110]:
```
performance_df.sort_values(by=['test AUC-ROC'], ascending = False)[1:]
```

[110]:
```
             Classifer  training CV error  training AUC-ROC  test error  \
6             Boosting           0.197853          0.859193    0.202840
```

```
0  Logistic Regression            0.209335            0.807885     0.210953
5        Random Forest            0.193781            0.985226     0.206897
4              Bagging            0.201935            0.975276     0.233266
3        Decision Tree            0.218873            0.790311     0.235294
1          Naive Bayes            0.305539            0.699519     0.292089

   test AUC-ROC
6      0.790500
0      0.785402
5      0.784583
4      0.760667
3      0.752963
1      0.707175
```

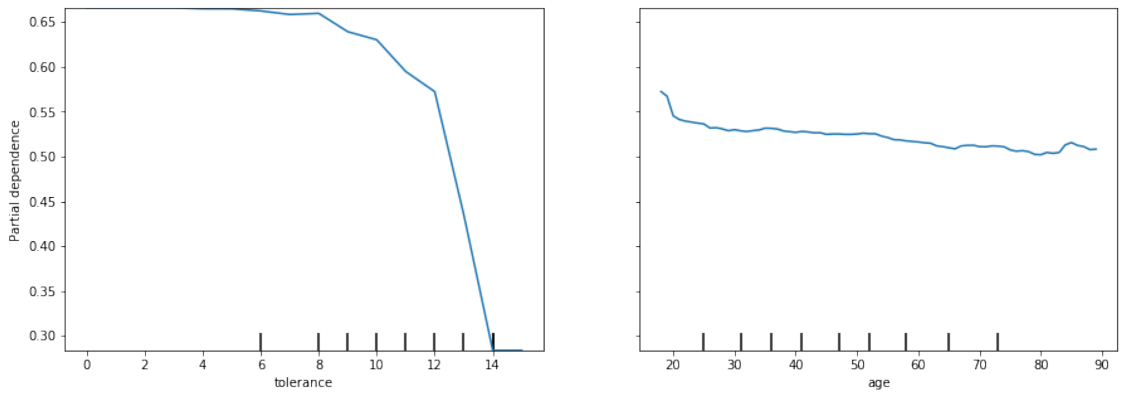
When we test the classifiers (I exclude the elastic net regressor here because it is not a classifier) on the test set, the result is similar. The best model for training set-random forest classifier also reveals disirable performance on the test set, but not the best. From the sorted tables above, we see that the Boosting classifier becomes the champion for both the evaluation of test error and AUC-ROC. Also, the ensemble models are generally better than single classifiers as on the training set, which show their robustness. However, one exception is the logistic regression classifier, it is the best among those single classifers and sometimes outperforms ensemble models. This might be related to the regularization used on the initial regressors.

### 0.1.6 Q6

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).

[116]:
```python
from sklearn.inspection import plot_partial_dependence as pdp

plt.rcParams['figure.figsize'] = [15,5]
pdp(best_model, gss_train_features, ['tolerance','age'], n_jobs = -1)
plt.show()
```



According to the partial dependence we can see that tolerance has a bigger impact on the classification result than age. We can see that at the tolerance level of 8 to 12, the curve starts to drop intensively, which indicates a potential big difference on the response variable (allow to not allow). While when we the curve of the age dependence, it does not change much as the age value varies, which means that our response variable might not be sensitive to it.