XGBoost

November 13, 2019

[2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import StratifiedKFold, GridSearchCV
from sklearn.ensemble import RandomForestClassifier,␣
 ↪GradientBoostingClassifier, AdaBoostClassifier, VotingClassifier,␣
 ↪BaggingClassifier
from sklearn.metrics import roc_auc_score, roc_curve, auc,␣
 ↪precision_recall_curve
from sklearn.metrics import classification_report, confusion_matrix
from xgboost import XGBClassifier
from mlxtend.plotting import plot_learning_curves
from yellowbrick.model_selection import LearningCurve
import matplotlib.gridspec as gridspec
import itertools
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.naive_bayes import MultinomialNB
from sklearn.utils import shuffle
from sklearn.preprocessing import LabelEncoder, OrdinalEncoder
from sklearn.metrics import recall_score
```

[3]:
```
def timer(start_time=None):
    if not start_time:
        start_time = datetime.now()
        return start_time
    elif start_time:
        tmin, tsec = divmod((datetime.now() - start_time).total_seconds(), 60)
        print('\n Tempo Necessário: %i minutos and %s segundos.' % (tmin,␣
 ↪round(tsec, 2)))
```

[4]:
```
train = pd.read_csv('trainLR.csv')
X_train = train.iloc[:,1:87]
Y_train = train.loc[:, train.columns == 'Y']
test = pd.read_csv('validationLR.csv')
X_test = test.iloc[:,1:87]
Y_test = test.loc[:, test.columns == 'Y']
```

Criar um XGBClassifier para usar como referência

[5]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

[6]:
```
kfold = KFold(n_splits=3, random_state=7)
```

[22]:
```
xgb_clf = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=96, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=0,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)

xgb_clf.fit(X_train, Y_train.values.ravel())
scores = cross_val_score(xgb_clf, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
```

[23]:
```
print(scores.mean())
```

```
0.6016276570380427
```

Importar Gridsearch e eleger os parâmetros otimizados para o XGBClassifier

[11]:
```
from sklearn.model_selection import GridSearchCV
```

[13]:
```
#Parâmetros XGBC
learning_rate=[0.1,0.2,0.3,0.4,0.5]
max_depth=[3,4,5,6,7,8,9,10]
objective=['binary:logitraw','binary:logistic']
n_estimators=[200,300,400,500]
param_grid = dict(learning_rate=learning_rate, max_depth=max_depth,␣
 ↪objective=objective, n_estimators=n_estimators)
```

[15]:
```
import time
xgb_clf = XGBClassifier(penalty='l2')
grid = GridSearchCV(estimator=xgb_clf, param_grid=param_grid, cv =3, n_jobs=4)

start_time = time.time()
grid_result = grid.fit(X_train, Y_train.values.ravel())
```

```
# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
/home/nakayama/anaconda3/envs/ML_Final/lib/python3.7/site-
packages/joblib/externals/loky/process_executor.py:706: UserWarning: A worker
stopped while some jobs were given to the executor. This can be caused by a too
short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning

Best: 0.628343 using {'learning_rate': 0.2, 'max_depth': 3, 'n_estimators': 200,
'objective': 'binary:logistic'}
Execution time: 11259.44020819664 ms
```

Criar um Random Forest para usar como referência

[16]:
```
kfold = KFold(n_splits=3, random_state=7)
rf = RandomForestClassifier(n_estimators=200,
                                max_depth=2, criterion='entropy',
                                n_jobs=4)
scores = cross_val_score(rf, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5483137752175815
```

Criar uma Árvore de decisão para usar como referência

[24]:
```
kfold = KFold(n_splits=3, random_state=7)
dt = DecisionTreeClassifier(class_weight=None, criterion='entropy',␣
 ↪max_depth=None, max_features=None, max_leaf_nodes=None)
scores = cross_val_score(dt, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5602895955891993
```

Criar um AdaBoost para usar como referência

[25]:
```
kfold = KFold(n_splits=3, random_state=7)
ada = AdaBoostClassifier(n_estimators=200, random_state=1, learning_rate=0.5)
scores = cross_val_score(ada, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5935179168967303
```

Importar Gridsearch e eleger os parâmetros otimizados para o Adaboost

[33]:
```
#Parâmetros Adaboost
learning_rate=[0.1,0.2,0.3,0.4,0.5,0.6]
n_estimators=[200,300,400,500,600,700,800,1000]
param_grid = dict(learning_rate=learning_rate,  n_estimators=n_estimators)
```

[34]:
```
import time
ada = AdaBoostClassifier()
grid = GridSearchCV(estimator=ada, param_grid = param_grid, cv =3, n_jobs=4)

start_time = time.time()
grid_result = grid.fit(X_train, Y_train.values.ravel())
# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
Best: 0.614828 using {'learning_rate': 0.4, 'n_estimators': 600}
Execution time: 1229.4651312828064 ms
```

Criar um Naive Bayes para usar como referência

[39]:
```
kfold = KFold(n_splits=3, random_state=7)
nb = GaussianNB()
scores = cross_val_score(nb, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5489127425413282
```

Criar uma SVM para usar como referência

[8]:
```
from sklearn.svm import SVC
svc = SVC(probability=False, gamma='auto')
scores = cross_val_score(svc, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5804914870300352
```

Criar uma LDA para usar como referência

[10]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
kfold = KFold(n_splits=3, random_state=7)
lda = LDA(n_components=1)
scores = cross_val_score(lda, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
/home/nakayama/.local/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are
collinear.
```

```
  warnings.warn("Variables are collinear.")
/home/nakayama/.local/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are
collinear.
  warnings.warn("Variables are collinear.")
```

```
0.5653290656607292
```

```
/home/nakayama/.local/lib/python3.7/site-
packages/sklearn/discriminant_analysis.py:388: UserWarning: Variables are
collinear.
  warnings.warn("Variables are collinear.")
```

Criar um KNN para usar como referência

[11]:
```
from sklearn.neighbors import KNeighborsClassifier
kfold = KFold(n_splits=3, random_state=7)
knn = KNeighborsClassifier()
scores = cross_val_score(knn, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.5652540714793778
```

Criar uma Random Forest para usar como referência

[13]:
```
kfold = KFold(n_splits=3, random_state=7)
rf = RandomForestClassifier(n_estimators=200,
                            max_depth=2, criterion='entropy',
                            n_jobs=4)
scores = cross_val_score(rf, X_train, Y_train.values.ravel(), cv=kfold,␣
 ↪scoring='precision')
print(scores.mean())
```

```
0.548952159082064
```

Importar Gridsearch e eleger os parâmetros otimizados para a Random Forest

[14]:
```
#Parâmetros RF
max_depth=[2,3,4,5,6,7,8]
criterion=['entropy','gini']
n_estimators=[100, 200,300,400,500]
param_grid = dict(max_depth=max_depth, criterion=criterion,␣
 ↪n_estimators=n_estimators)
```

[15]:
```
import time
rf = RandomForestClassifier()
grid = GridSearchCV(estimator=rf, param_grid = param_grid, cv =3, n_jobs=4)

start_time = time.time()
```

```
grid_result = grid.fit(X_train, Y_train.values.ravel())
# Summarize results
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
print("Execution time: " + str((time.time() - start_time)) + ' ms')
```

```
Best: 0.616492 using {'criterion': 'gini', 'max_depth': 8, 'n_estimators': 300}
Execution time: 311.5600333213806 ms
```

[ ]: