# AdaBoost

December 22, 2022

[104]:
```
import sys
import pandas as pd
import matplotlib.pyplot as plt
import sklearn.ensemble as ske
import numpy as np
import sklearn.metrics as skm
import sklearn.model_selection as skms
import sklearn.tree as skt
import sklearn.linear_model as sklm
import sklearn.discriminant_analysis as skda
from sklearn.model_selection import train_test_split

sys.path.append("..")
from utils.loading_data  import load_to_df_from_csv
df = load_to_df_from_csv("../data/train.csv")

for i in range(len(df)): #Loop to increment the lead words into female words or␣
 ↪male words

    if df.loc[i,"Lead"]=="Female":

        df.loc[i,"Number words female"]=df.loc[i,"Number words female"]+df.
 ↪loc[i,"Number of words lead"]
    else:
        df.loc[i,"Number words male"]=df.loc[i,"Number words male"]+df.
 ↪loc[i,"Number of words lead"]
df["difference_words_m_f"]=df.iloc[:,7]-df.iloc[:,0] #Create new feature, the␣
 ↪difference in words spoken between males and females.
```

[105]:
```
X=df.iloc[:,0:13]
y=df.iloc[:,13]


X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=2)␣
 ↪#Random split data 75,25

X_train.reset_index(drop=True,inplace=True) #Reset index
```

```
y_train.reset_index(drop=True,inplace=True) #Reset index

# Function to produce an array of all feature combinations for lenghts larger␣
 ↪than or equal to 8
def get_all_feature_combinations(data_columns):
    from itertools import chain, combinations
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    feature_combinations = list(chain.from_iterable(combinations(data_columns,␣
 ↪r) for r in range(len(data_columns)+1)))

    feature_combinations_set = []
    for feature_combination in feature_combinations:
        feature_combination_set = []
        for feature in feature_combination:
            feature_combination_set.append(feature)

        if len(feature_combination_set)>=8:
            feature_combinations_set.append(feature_combination_set)

    return feature_combinations_set

feature_combinations = get_all_feature_combinations(X.columns)
```

[106]:
```
#This cell implements Adaboost with base estimators logistic regression,␣
 ↪decision tree and random forest, on train data using stratified K-folds
#for all combinations of features of lenght larger than or equal to 8


logistic_object=sklm.LogisticRegression(max_iter=10000) #A logistic model, with␣
 ↪default solver

decisiontree_object=skt.
 ↪DecisionTreeClassifier(max_depth=10,criterion="entropy") #A decision tree␣
 ↪modele

randomforest_object=ske.
 ↪RandomForestClassifier(n_estimators=50,criterion="entropy") #Random forest

objects=[logistic_object,decisiontree_object,randomforest_object]

X=X_train

y=y_train
```

```
k_folds_object=skms.StratifiedKFold(n_splits=10,shuffle=True,random_state=2) #A
 ↪stratified K-fold object is created, to evaluate each model on 70% of the
#original data
Best=[0,"method","variable combination"] #Temporary list for the best accuracy,
 ↪with the best method, using the optimal variable combination

for variables in feature_combinations:

    X=X_train.loc[:,variables]

    for x in objects: #Iterate through each object
        boosting_object=ske.AdaBoostClassifier(base_estimator=x,n_estimators=50)
        acc_score = []
        k_folds_object=skms.
 ↪StratifiedKFold(n_splits=5,shuffle=True,random_state=2)

        for train_index , test_index in k_folds_object.split(X,y): #Stratified
 ↪K-fold
            X_train1 , X_test1 = X.iloc[train_index,:],X.iloc[test_index,:]
            y_train1 , y_test1 = y[train_index] , y[test_index]

            boosting_object.fit(X_train1,y_train1)

            pred_values = boosting_object.predict(X_test1)

            acc = skm.accuracy_score(pred_values , y_test1)
            acc_score.append(acc)
        avg_acc_score = np.mean(acc_score)
        if avg_acc_score>Best[0]:
            Best[0]=avg_acc_score
            Best[1]=str(x)
            Best[2]=variables
Best
```

[106]:
```
[0.9294292803970224,
 'LogisticRegression(max_iter=10000)',
 ['Number words female',
  'Number of words lead',
  'Number of male actors',
  'Number of female actors',
  'Number words male',
  'Mean Age Female',
  'Age Lead',
  'Age Co-Lead']]
```

One can see, based on the above exhaustive search over all feature combinations >= 8, base classifiers Logistic Regression, Decision tree and Random Forest, using a stratified K-folds accuracy

metric as method of comparison, that the AdaBoost model with base clasifier Logistic Regression, using the features " ['Number words female', 'Number of words lead', 'Number of male actors', 'Number of female actors', 'Number words male', 'Mean Age Female', 'Age Lead', 'Age Co-Lead']] ", provides the best accuracy.

Next, we find the accuracy on unseen data:

[107]:
```
#Cell to evaluate the optimal boosted LR model from above on unseen test data

X_train=X_train.loc[:,Best[2]]
X_test=X_test.loc[:,Best[2]]

boosted_logistic_object=ske.
 ↪AdaBoostClassifier(base_estimator=logistic_object,n_estimators=10)
boosted_logistic_object.fit(X_train,y_train)

#Train accuracy
pred=boosted_logistic_object.predict(X_train)
print(skm.accuracy_score(pred,y_train))

#Test accuracy
pred=boosted_logistic_object.predict(X_test)
print(skm.accuracy_score(pred,y_test))
```

```
0.9242618741976893
0.8807692307692307
```