Information Retrieval – Assignment 15

# Code

# IR16A.py CS5154/6054 cheng 2022  
# Comparing classifiers on documents as binary, count, or tfidf vector  
# on three random segments of bible.txt from the first, third, and last fifths  
# 100 test documents are at the center of 1000 training documents  
# Only the four in IIR Chapters 13 and 14 are implemented  
# you need to add the ten others imported like you did for IR15A.py  
# Usage: python IR16A.py  
  
import numpy as np  
import random  
from sklearn.feature\_extraction.text import CountVectorizer  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.naive\_bayes import BernoulliNB  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.neighbors import NearestCentroid  
from sklearn.linear\_model import LogisticRegression  
from sklearn.svm import LinearSVC  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.tree import ExtraTreeClassifier  
from sklearn.ensemble import ExtraTreesClassifier  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.linear\_model import Perceptron  
from sklearn.neural\_network import MLPClassifier  
from sklearn.metrics import accuracy\_score  
  
f = open("bible.txt", "r")  
docs = f.readlines()  
f.close()  
N =len(docs)  
N1 = N // 5 - 1100  
  
doc\_as\_list = ['binary', 'count', 'tf-idf']  
  
for iteration in range(0, 5):  
  
 accuracy\_scores = dict()  
  
 c0 = random.randrange(N1)  
 c1 = 2 \* N // 5 + random.randrange(N1)  
 c2 = N - 1100 - random.randrange(N1)  
 print('Random segments at -', c0, c1, c2)  
  
 trainX = np.concatenate([docs[c0:c0+500], docs[c0+600:c0+1100],  
 docs[c1:c1+500], docs[c1+600:c1+1100], docs[c2:c2+500], docs[c2+600:c2+1100]])  
 y = np.concatenate([np.zeros(1000, dtype=np.int16), np.ones(1000, dtype=np.int16), np.full(1000, 2, dtype=np.int16)])  
 testX = np.concatenate([docs[c0+500:c0+600], docs[c1+500:c1+600], docs[c2+500:c2+600]])  
 testY = np.concatenate([np.zeros(100, dtype=np.int16), np.ones(100, dtype=np.int16), np.full(100, 2, dtype=np.int16)])  
  
 # documents as binary vectors  
 cv = CountVectorizer(binary=True, max\_df=0.4, min\_df=4)  
 X0 = cv.fit\_transform(trainX).toarray()  
 T0 = cv.transform(testX).toarray()  
  
 # documents as count vectors  
 cv = CountVectorizer(max\_df=0.4, min\_df=4)  
 X1 = cv.fit\_transform(trainX).toarray()  
 T1 = cv.transform(testX).toarray()  
  
 # documents as tfidf vectors  
 cv = TfidfVectorizer(max\_df=0.4, min\_df=4)  
 X2 = cv.fit\_transform(trainX).toarray()  
 T2 = cv.transform(testX).toarray()  
  
 model = BernoulliNB()  
 model.fit(X0, y)  
 pred = model.predict(T0)  
 A0 = accuracy\_score(testY, pred)  
 print ('BernoulliNB -', A0)  
 accuracy\_scores['BernoulliNB'] = [A0]  
  
 model = MultinomialNB()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 print ('MultinomialNB -', A0, A1)  
 accuracy\_scores['MultinomialNB'] = [A0, A1]  
  
 model = KNeighborsClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('KNN -', A0, A1, A2)  
 accuracy\_scores['KNN'] = [A0, A1, A2]  
  
 model = NearestCentroid()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Rocchio -', A0, A1, A2)  
 accuracy\_scores['Rocchio'] = [A0, A1, A2]  
  
 model = LogisticRegression()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Logistic Regression -', A0, A1, A2)  
 accuracy\_scores['Rocchio'] = [A0, A1, A2]  
  
 model = LinearSVC()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('LinearSVC -', A0, A1, A2)  
 accuracy\_scores['LinearSVC'] = [A0, A1, A2]  
  
 model = SVC()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('SVC -', A0, A1, A2)  
 accuracy\_scores['SVC'] = [A0, A1, A2]  
  
 model = DecisionTreeClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Decision Tree Classifier -', A0, A1, A2)  
 accuracy\_scores['Decision Tree Classifier'] = [A0, A1, A2]  
  
 model = ExtraTreeClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Extra Tree Classifier -', A0, A1, A2)  
 accuracy\_scores['Extra Tree Classifier'] = [A0, A1, A2]  
  
 model = ExtraTreesClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Extra Trees Classifier -', A0, A1, A2)  
 accuracy\_scores['Extra Trees Classifier'] = [A0, A1, A2]  
  
 model = AdaBoostClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Ada Boost Classifier -', A0, A1, A2)  
 accuracy\_scores['Ada Boost Classifier'] = [A0, A1, A2]  
  
 model = RandomForestClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Random Forest Classifier -', A0, A1, A2)  
 accuracy\_scores['Random Forest Classifier'] = [A0, A1, A2]  
  
 model = Perceptron()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('Perceptron -', A0, A1, A2)  
 accuracy\_scores['Perceptron'] = [A0, A1, A2]  
  
 model = MLPClassifier()  
 model.fit(X0, y)  
 A0 = accuracy\_score(testY, model.predict(T0))  
 model.fit(X1, y)  
 A1 = accuracy\_score(testY, model.predict(T1))  
 model.fit(X2, y)  
 A2 = accuracy\_score(testY, model.predict(T2))  
 print ('MLP Classifier -', A0, A1, A2)  
 accuracy\_scores['MLP Classifier'] = [A0, A1, A2]  
  
 doc\_as = ""  
 classifier = ""  
 max\_accuracy = 0  
  
 for key, value in accuracy\_scores.items():  
 if max(value) > max\_accuracy:  
 doc\_as = doc\_as\_list[value.index(max(value))]  
 classifier = key  
 max\_accuracy = max(value)  
 elif max(value) == max\_accuracy:  
 doc\_as = doc\_as + f'/{doc\_as}'  
 classifier = classifier + f'/{key}'  
  
 print(f'Winning Combination = {classifier} with document as {doc\_as} vectors')  
 print('----------------------------------------------------------------------')

# Results

## Iteration 1

Text

Description automatically generated

## Iteration 2

Text

Description automatically generated

## Iteration 3

Text

Description automatically generated

## Iteration 4

Text

Description automatically generated

## Iteration 5

Text

Description automatically generated