MO444 – Exercício 6

Aluno: Antonio Carlos Guimarães Junior RA 134985

Importação das Bibliotecas e Leitura dos Dados

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer,
TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import TruncatedSVD, PCA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from nltk.stem.porter import PorterStemmer
import string
import numpy as np


files = load_files("filesk")
target = files.target
```

Parte 1: Processamento do Texto

```
stemmer = PorterStemmer()
analyzer = CountVectorizer().build_analyzer()

def stemmed_words(doc):
    return (stemmer.stem(w) for w in analyzer(doc))

def prePros(x):
    x = x.translate(None, bytes(string.punctuation, "utf-8"))
    return x

files = map(prePros, files.data)
countVet = CountVectorizer(analyzer=stemmed_words, stop_words='english',
lowercase=True, min_df=2, strip_accents='unicode')
X_trainCounts = countVet.fit_transform(files)
tf_transformer = TfidfTransformer(use_idf=False).fit(X_trainCounts)
X_traintf = tf_transformer.transform(X_trainCounts)
print(X_trainCounts.shape)
print(X_traintf.shape)
```

No código acima a variável X_trainCounts representa a matriz binária e a variável X_traintf a matriz de frequências.

Para verificar o tamanho da matriz, seus shapes foram impressos:

```
(5000, 21557)
(5000, 21557)
```

Parte 2 - classificador multiclasse na matriz termo-documento original

Código:

```
Xtreino, Xtest, Ytreino, Ytest = train_test_split(X_trainCounts, target,
train_size=4000)
XtreinoTF, XtestTF, YtreinoTF, YtestTF = train_test_split(X_traintf,
target, train_size=4000)

clf = MultinomialNB().fit(Xtreino, Ytreino)
acuracia = clf.score(Xtest, Ytest)
print("Acuracia Naive Bayes:", acuracia)

clf = LogisticRegression(C=10000).fit(Xtreino, Ytreino)
acuracia = clf.score(Xtest, Ytest)
print("Acuracia LR Binaria:", acuracia)

clf = LogisticRegression(C=10000).fit(XtreinoTF, YtreinoTF)
acuracia = clf.score(XtestTF, YtestTF)
print("Acuracia LR Freq:", acuracia)
```

Saída:

```
Acuracia Naive Bayes: 0.82
Acuracia LR Binaria: 0.833
Acuracia LR Freq: 0.843
```

Parte 3 - classificador multiclasse na matriz termo-documento reduzida

Foram testados o SVM com RBF e o Random Forest. Foi utilizado código similar ao do lab 3 para encontrar os melhores hiperparametros. Em todos os casos, foram utilizados os dados com PCA de 0.99.

Código:

```
def gridSearch(metodo, hiperparametros, dados):
    print(type(metodo).__name__)
    gridS = GridSearchCV(metodo, hiperparametros, n_jobs=-1)
    gridS.fit(dados[0], dados[1])
    acuracia = gridS.score(dados[2], dados[3])
    print("Acuracia:", acuracia)
    print("Hiperparametros:", gridS.best_params_)
    return acuracia

# PCA
pca = PCA(n_components=0.99)
XtreinoTFPCA = pca.fit_transform(XtreinoTF.toarray(), YtreinoTF)
XtestTFPCA = pca.transform(XtestTF.toarray(), YtestTF)
print(XtreinoTFPCA.shape)
print(XtestTFPCA.shape)
dados = [XtreinoTFPCA, YtreinoTF, XtestTFPCA, YtestTF]

# RBF SVM
SVMAcuracia = gridSearch(SVC(), {'C': [2**(-5), 2**(0), 2**(5), 2**(10)],
'gamma': [2**(-15), 2**(-10), 2**(-5), 2**(0), 2**(5)]}, dados)

# Random Forest
RFoAcuracia = gridSearch(RandomForestClassifier(), {'max_features': [10,
15, 20, 25], 'n_estimators': [100, 200, 300, 400]}, dados)
```

Saída:

```
(4000, 2501)
(1000, 2501)
SVC
Acuracia: 0.856
Hiperparametros: {'C': 32, 'gamma': 1}
RandomForestClassifier
Acuracia: 0.587
Hiperparametros: {'max_features': 25, 'n_estimators': 100}
```

Assim, podemos ver que os melhores resultados foram para o SVM com RBF com os Hiperparametros C = 32 e gamma = 1. Tendo Acurácia de 85.6%.