Caique Garutti Moreira 145584

Exercício 6 - MC886

Para a parte de stemming do texto utilizei o PorterStemmer da biblioteca NLTK. Todo o restante do preprocessamento foi feito utilizando o CountVectorizers, passando as flags específicas:

```
vectorizer = CountVectorizer(data, strip_accents='ascii',
 stop_words='english', min_df=2, lowercase=True)
```

O CountVectorizer foi utilizado para criar a Matriz Binária, e o TfidfTransformer para criar a matriz de frequencias.

Após aplicar PCA na matriz de frequencias, utilizei o StratifiedKFold com 5 folds para dividir aleatoriamente em conjunto com 4000 linhas de treino e 1000 linhas de teste.

Por fim, apliquei Naive Bayes, Logistic Regression, SVM e RF nas matrizes para verificar a acuracia em adivinhar a classe do conjunto de teste

Acurácia do Naive Bayes rodando na Matriz Binaria:
0.712861415753

Acurácia do LogisticRegression na Matriz Binaria: **0.822532402792**

Acurácia do Logistic Regression na Matriz de Frequências:

0.851445663011

Acurácia do SVM na Matriz de Frequencias: **0.473579262213**

Acurácia do RF na Matriz de Frequencias: **0.613160518445**

Código

```
from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectoriz
er
from sklearn.feature_extraction.text import TfidfTransfor
mer
from nltk.stem.porter import *
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier

def lab6():

    # preprocessing
    texts = load_files('filesk')
    texts.data = stemData(texts.data)
```

```
    binMatrix = createBinMatrix(texts.data)
    freqMatrix = createFreqMatrix(texts.data)

    freqMatrixPCA = aplicarPCA(freqMatrix)

    (treinoBin, testeBin, treinoFreq, testeFreq, treinoFr
eqPCA, testeFreqPCA, yTreino, yTeste) = particionarTreino
Teste(texts.target, binMatrix, freqMatrix, freqMatrixPCA)

    scoreBayes = aplicarNaiveBayes(treinoBin, testeBin, y
Treino, yTeste)
    print("Score do Nayve Bayes eh " + str(scoreBayes))
    scoreLogRegBinMatrix = aplicarLogisticRegression(trei
noBin, testeBin, yTreino, yTeste)
    print("Score da Logistic Regression na matriz binaria
 eh " + str(scoreLogRegBinMatrix))
    scoreLogRegFreqMatrix = aplicarLogisticRegression(tre
inoFreq, testeFreq, yTreino, yTeste)
    print("Score da Logistic Regression na matriz de freq
uencias eh " + str(scoreLogRegFreqMatrix))

    scoreSVM = aplicarSVM(treinoFreqPCA, testeFreqPCA, yT
reino, yTeste)
    print("Score do SVM eh " + str(scoreSVM))
    scoreRF =  aplicarRF(treinoFreqPCA, testeFreqPCA, yTr
eino, yTeste)
    print("Score do RF eh " + str(scoreRF))
```

```
# Aplicando Classificador Random Forest
def aplicarRF(X_treino, X_test, y_treino, y_test):
    print("Aplicando Random Forest")
    rf = RandomForestClassifier()
    rf.fit(X_treino, y_treino)
    score = rf.score(X_test, y_test)

    return score

# Aplicando Classificador SVM
def aplicarSVM(X_treino, X_test, y_treino, y_test):
    print("Aplicando SVM")
    svc = SVC()
    svc.fit(X_treino, y_treino)
    score = svc.score(X_test, y_test)

    return score

# Aplicando PCA com variancia 99%
def aplicarPCA(X):
    print("Aplicando PCA")
    pca = PCA(n_components=0.99)
    pca.fit(X.toarray())
    return pca.transform(X.toarray())

# Aplicando Naive bayes
def aplicarNaiveBayes(X_treino, X_test, y_treino, y_test)
:
```

```
    print("Aplicando Naive Bayes")
    gnb = GaussianNB()
    gnb.fit(X_treino.toarray(), y_treino)
    score = gnb.score(X_test.toarray(), y_test)

    return score

# Aplicando Logistic Regression
def aplicarLogisticRegression(X_treino, X_test, y_treino,
 y_test):
    print("Aplicando Logistic Regression")
    logreg = LogisticRegression(C=10000)
    logreg.fit(X_treino, y_treino)
    score = logreg.score(X_test, y_test)

    return score

# Particionando entre treino e teste, usando StratifiedKF
old
def particionarTreinoTeste(targets, binMatrix, freqMatrix
, freqMatrixPCA):
    print("Particionando entre treino e teste")
    fold_5 = StratifiedKFold(n_splits=5)
    splits = fold_5.split(binMatrix, targets)
    for indices_treino, indices_test in splits:
        indTreino = indices_treino
        indTeste = indices_test
        break
```

```
    return (binMatrix[indTreino], binMatrix[indTeste], fr
eqMatrix[indTreino], freqMatrix[indTeste], freqMatrixPCA[
indTreino], freqMatrixPCA[indTeste], targets[indTreino],
targets[indTeste])

# Fazendo stemming dos textos
def stemData(data):
    print("Iniciando processo de stemming")
    stemmer = PorterStemmer()

    stemmedTextData = []

    # stemming
    for text in data:
        tokens = text.decode("utf8").split(' ')
        stemmedText = ' '.join([stemmer.stem(token) for t
oken in tokens])
        stemmedTextData += [stemmedText.encode("utf8")]

    return stemmedTextData

# Criando matriz binaria
def createBinMatrix(data):
    print("Criando matriz esparsa binaria")
    vectorizer = CountVectorizer(data, strip_accents='asc
ii', stop_words='english', min_df=2, lowercase=True, bina
ry=True)
```

```
    vectorizer.fit(data)
    binMatrix = vectorizer.transform(data)

    return binMatrix

# Criando matriz de frequencias
def createFreqMatrix(data):
    print("Criando matriz com frequencias")
    vectorizer = CountVectorizer(data, strip_accents='asc
ii', stop_words='english', min_df=2, lowercase=True)
    vectorizer.fit(data)
    sparseMatrix = vectorizer.transform(data)

    transformer = TfidfTransformer()

    transformer.fit(sparseMatrix)

    return transformer.transform(sparseMatrix)
```