Nome: Nathana Facion

Exercício 6
Aprendizado de Máquina

Para melhor entendimento o código foi divididos em partes. Para início do programa comece pela sessão chamada Main, que corresponde ao número 6. O que foi pedido na parte 1,2,3 do enunciado do problema se encontram separado para melhor entendimento.

1. Bibliotecas:

```
import numpy as np
from sklearn.decomposition import PCA
from numpy import genfromtxt
from sklearn.datasets import load_files
import string
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
from sklearn import linear_model
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer
from copy import deepcopy
```

2. Funções usadas:

```
def accuracy(matrix):
  return round(float(matrix[0][0] + matrix[1][1]) / float(matrix.sum()), 4)

# remocao de stop words
def removeStopWords(data):
  print 'removeStopWords..'
  all_stopwords = set(stopwords.words('english'))
  list_data = []
  for s in data:
     if s not in all_stopwords:
        list_data.append(s)
  return list_data

# conversao de caracteres maiusculos para minusculos
def lowerWord(data):
```

```
  print 'lowerWord..'
  list_data = []
  for s in data:
     list_data.append(unicode(s.lower(),'utf-8'))
  return list_data

# remocao de pontuacao
def removePunctuation(data):
  print 'removePunctuation..'
  list_data = []
  punctuations = list(string.punctuation)
  for s in data:
     if s not in punctuations:
        list_data.append(s)
  return list_data

# steming dos termos
def stremingWord(data):
  print 'stremingWord..'
  stemmer = PorterStemmer()
  list_data = []
  for s in data:
     list_data.append(stemmer.stem(s))
  return list_data

# remocao dos termos que aparecem em um so documento
def removeWord(data):
  print 'removeWord..'
  dataUpdate = []
  s = " "
  for i, d1 in enumerate(data):
     # pega os elementos que estao nesse d1
     diffSet = set(d1.split())
     for j, d2 in enumerate(data):
        if i != j:
           # Retira do conjunto o que esta so nele
           diffSet =  (diffSet - set(d2.split()))
     delWord =  [word for word in d1.split() if word not in list(diffSet)]
     dataUpdate.append(s.join(delWord))
  return dataUpdate


# bag de freq term
def bagWords(data):
  print 'bagWords..'
  count_vect = CountVectorizer()
  bag = count_vect.fit_transform(data)
  print bag.toarray()
  return bag.toarray()

# bag binary
def bagWordsBinary(data):
  print 'bagWords..'
  count_vect = CountVectorizer(binary = True)
  bag = count_vect.fit_transform(data)
  print bag.toarray()
  return bag.toarray()
```

3. Parte 1 - processamento de texto

```
# Parte 1 - processamento de texto
def preProcessingData(data):
  data = lowerWord(data)
  data = removePunctuation(data)
  data = removeStopWords(data)
  data = stremingWord(data)
  data = removeWord(data)
  data = bagWords(data)
  return data

# Parte 1 - processamento de texto
def preProcessingDataBinary(data):
  data = lowerWord(data)
  data = removePunctuation(data)
  data = removeStopWords(data)
  data = stremingWord(data)
  data = removeWord(data)
  data = bagWordsBinary(data)
  return data
```

4. Parte 2 - classificador multiclasse na matriz termo-documento original

```
# Parte 2 - classificador multiclasse na matriz termo-documento original
def documentOriginal(data, dataClass):
  X_train = np.array([d for d in data])[1:4000]
  Y_train = np.array([d for d in dataClass])[1:4000]
  X_test = np.array([d for d in data])[4001:5000]
  Y_test = np.array([d for d in dataClass])[4001:5000]

  # Rode o naive bayes na matrix binaria.
  clf = GaussianNB()
  clf.fit(X_train, Y_train)
  predict = clf.predict(X_test)
  c_matrix = confusion_matrix(Y_test, predict)
  ac_GB = accuracy(c_matrix)
  print ('Naive Bayes :', ac_GB)

  # Em Python use C=10000 em sklearn.linear_model.LogisticRegression para evitar que haja regularizacao.
  logistic = linear_model.LogisticRegression(C=10000)
  logistic.fit(X_train, Y_train)
  predict = logistic.predict(X_test)
  c_matrix = confusion_matrix(Y_test, predict)
  ac_logistic = accuracy(c_matrix)
  print ('Logistic Regression :', ac_logistic)
```

5. Parte 3 - classificador multiclasse na matriz termo-documento reduzida

```
# Parte 3 - classificador multiclasse na matriz termo-documento reduzida
def documentReduced(data, dataClass):

  pca = PCA(n_components=0.99)
  pca.fit_transform(data)
  X_train = np.array([d for d in data])[1:4000]
  Y_train = np.array([d for d in dataClass])[1:4000]
  X_test = np.array([d for d in data])[4001:5000]
  Y_test = np.array([d for d in dataClass])[4001:5000]

  # Rode pelo menos 2 algoritmos
  # SVM com RBF,
  svm = SVC(kernel='rbf')
  svm.fit(X_train, Y_train)
  predict = svm.predict(X_test)
  c_matrix = confusion_matrix(Y_test, predict)
  ac_svm = accuracy(c_matrix)
  print ('SVM :', ac_svm)

  # gradient boosting
  gb = GradientBoostingClassifier()
  gb.fit(X_train, Y_train)
  predict = gb.predict(X_test)
  c_matrix = confusion_matrix(Y_test, predict)
  ac_gb = accuracy(c_matrix)
  print ('Gradient Boosting :', ac_gb)
```

6. Main

```
def main():
 # leitura dos dados
load = load_files('//home//nathana//AM//filesk//')
# leitura das classe
fileClass = "//home//nathana//AM//category.tab.txt"
data_class = genfromtxt(fileClass, delimiter='\t',dtype="string")[1:]
dataFreq = preProcessingData(load.data)
dataBinary = preProcessingDataBinary(load.data)

dataClass = []
for d in data_class:
  data_split = d.split(" ")
  dataClass.append(data_split[1].replace('"',""))

documentOriginal(dataFreq, dataClass)
documentReduced(dataFreq, dataClass)

documentOriginal(dataBinary, dataClass)
documentReduced(dataBinary, dataClass)

if __name__ == '__main__':
```

```
main()
```

7. Saídas

Bag Term Frequency:
('Naive Bayes :', 0.3323)
('Logistic Regression :', 0.3033)
('SVM :', 0.4545)
('Gradient Boosting :', 0.4274)
Bag Binary:
('Naive Bayes :', 0.3493)
('Logistic Regression :', 0.2963)
('SVM :', 0.4545)
Gradient Boosting :', 0.4204)

8. Respostas

Rode o naive bayes na matrix binaria. Qual a acuracia?
Term Frequency:
('Naive Bayes :', 0.3323)
Binary:
('Naive Bayes :', 0.3493)
Rode o logistic regression na matrix binaria e de term frequency. Quais as acurácias.
Term Frequency:
('Logistic Regression :', 0.3033)
Binary:
('Logistic Regression :', 0.2963)
Rode pelo menos 2 algoritmos dentre SVM com RBF, gradient boosting e random forest na matrix com o numero de dimensões reduzidas. Quais as acurácias?
Term Frequency:
('SVM :', 0.4545)
('Gradient Boosting :', 0.4274)
Binary:
('SVM :', 0.4545)
Gradient Boosting :', 0.4204)