Titulo: Exercício 6 - Aprendizado de Máquina em textos
Autor: Juan Sebastián Beleño Díaz
Data: 15 de Novembro de 2016

Introdução

Neste trabalho é feito o pipeline completo para a classificação de textos: processamento dos textos(conversao de caracteres maiusculos para minusculos, remoçao de pontuaçao, remoçao de stop words, steming dos termos, etc.), processamento de *Bag of Words* e *Term-Frequency Matrix*, classificação nas matrizes esparsas usando Naïve Bayes e Regressão Logística, finalmente, classificação na *Term-Frequency Matrix* com PCA usando SVM e Random Forest.

Dados

Os arquivos utilizados neste trabalho são os textos e as classes dos textos. O arquivo de *textos* é um arquivo zip que contem um conjunto de pastas com o nome das categorias e dentro delas estão os arquivos de cada categoria. O arquivo *classes dos textos* contem duas colunas: a primerira delas é o nome do arquivo e a segunda é a categoria à qual pertence o arquivo.

Preparação dos dados

Antes de começar trablahar com os dados é preciso incluir as dependecias do projeto:

```
# Loading the libraries
import numpy as np
import pandas as pd
#import os
import zipfile
import urllib.request

from nltk.stem.snowball import PorterStemmer
from nltk.tokenize import word_tokenize
#from nltk.corpus import stopwords

from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

Existem muitas maneiras de abrir os arquivos e obter os dados, mas neste caso foi usado *pandas* para obter os dataframes diretamente desde a URL. Além disso, foi utilizado *urllib* para obter o arquivo zip e *zipfile* para descompactar as pastas do arquivo zip.

```
# URLs with data
url_zip_data =
'http://www.ic.unicamp.br/%7Ewainer/cursos/2s2016/ml/ex6/file-sk.zip'
```

```
url_categories =
'http://www.ic.unicamp.br/%7Ewainer/cursos/2s2016/ml/ex6/category.tab'

# Local path for different files and directories
filepath_zip = '../assets/file-sk.zip'
dirpath_zip = '../assets'
directories = {'Apps': '../assets/filesk/Apps/',
               'Enterprise': '../assets/filesk/Enterprise/',
               'Gadgets': '../assets/filesk/Gadgets/',
               'Social': '../assets/filesk/Social/',
               'Startups': '../assets/filesk/Startups/'}
# dirpath_zip = '../assets/file-sk'

# Upload the zip file
urllib.request.urlretrieve(url_zip_data, filepath_zip)

# Creating the directory where I'll put the uncompressed files
# if not os.path.exists(dirpath_zip):
#    os.makedirs(dirpath_zip)

# Uncompress the zip files in a directory
zip_ref = zipfile.ZipFile(filepath_zip, 'r')
zip_ref.extractall(dirpath_zip)
zip_ref.close()

# Extracting the categories for each file
df_categories = pd.read_csv(url_categories,
                            header= None,
                            names= ['file', 'category'],
                            skiprows = [0],
                            delimiter = " " )
```

Parte 1 - Processamento de textos

Para facilitar as tarefas de classificação foi preciso primeiro processar os dados para diminuir a dimensionalidade e escolher atributos relevantes usando: conversao de caracteres maiusculos para minusculos, remoçao de pontuaçao, remoçao de stop words, steming dos termos, etc.

```
punctuation = ['.', ',', ';', ':', " ' ", " 's ", '?',
               '"', '”', '‘', '’s', '—', '/'
               '(', ')', '[', ']', '1', '2', '3',
               '4', '5', '6', '7', '8', '9', '0',
               '$', '%', '-', '•', '~']

stop_words = [" a ", " about ", " above ", " above ", " across ", " after ", "
afterwards ",
              " again ", " against ", " all ", " almost ", " alone ", " along ", "
already ",
              " also ","although ","always ","am ","among ", " amongst ", "
amoungst ",
              " amount ",  " an ", " and ", " another ", " any ","anyhow ","anyone
",
              " anything ","anyway ", " anywhere ", " are ", " around ", " as ",  "
at ",
```

```
            " back ","be ","became ", " because ","become ","becomes ", "
becoming ",
            " been ", " before ", " beforehand ", " behind ", " being ", " below
",
            " beside ", " besides ", " between ", " beyond ", " bill ", " both ",
            " bottom ","but ", " by ", " call ", " can ", " cannot ", " cant ", "
co ",
            " con ", " could ", " couldnt ", " cry ", " de ", " describe ", "
detail ",
            " do ", " done ", " down ", " due ", " during ", " each ", " eg ", "
eight ",
            " either ", " eleven ","else ", " elsewhere ", " empty ", " enough ",
" etc ",
            " even ", " ever ", " every ", " everyone ", " everything ", "
everywhere ",
            " except ", " few ", " fifteen ", " fify ", " fill ", " find ", "
fire ",
            " first ", " five ", " for ", " former ", " formerly ", " forty ", "
found ",
            " four ", " from ", " front ", " full ", " further ", " get ", " give
", " go ",
            " had ", " has ", " hasnt ", " have ", " he ", " hence ", " her ", "
here ",
            " hereafter ", " hereby ", " herein ", " hereupon ", " hers ", "
herself ",
            " him ", " himself ", " his ", " how ", " however ", " hundred ", "
ie ", " if ",
            " in ", " inc ", " indeed ", " interest ", " into ", " is ", " it ",
" its ",
            " itself ", " keep ", " last ", " latter ", " latterly ", " least ",
" less ",
            " ltd ", " made ", " many ", " may ", " me ", " meanwhile ", " might
", " mill ",
            " mine ", " more ", " moreover ", " most ", " mostly ", " move ", "
much ",
            " must ", " my ", " myself ", " name ", " namely ", " neither ", "
never ",
            " nevertheless ", " next ", " nine ", " no ", " nobody ", " none ", "
noone ",
            " nor ", " not ", " nothing ", " now ", " nowhere ", " of ", " off ",
" often ",
            " on ", " once ", " one ", " only ", " onto ", " or ", " other ", "
others ",
            " otherwise ", " our ", " ours ", " ourselves ", " out ", " over ", "
own ",
            " part ", " per ", " perhaps ", " please ", " put ", " rather ", " re
", " same ",
            " see ", " seem ", " seemed ", " seeming ", " seems ", " serious ", "
several ",
            " she ", " should ", " show ", " side ", " since ", " sincere ", "
six ", " sixty ",
            " so ", " some ", " somehow ", " someone ", " something ", " sometime
",
            " sometimes ", " somewhere ", " still ", " such ", " system ", " take
", " ten ",
            " than ", " that ", " the ", " their ", " them ", " themselves ", "
then ",
```

```
               " thence ", " there ", " thereafter ", " thereby ", " therefore ", "
therein ",
               " thereupon ", " these ", " they ", " thickv ", " thin ", " third ",
" this ",
               " those ", " though ", " three ", " through ", " throughout ", " thru
",
               " thus ", " to ", " together ", " too ", " top ", " toward ", "
towards ",
               " twelve ", " twenty ", " two ", " un ", " under ", " until ", " up
", " upon ",
               " us ", " very ", " via ", " was ", " we ", " well ", " were ", "
what ",
               " whatever ", " when ", " whence ", " whenever ", " where ", "
whereafter ",
               " whereas ", " whereby ", " wherein ", " whereupon ", " wherever ", "
whether ",
               " which ", " while ", " whither ", " who ", " whoever ", " whole ", "
whom ",
               " whose ", " why ", " will ", " with ", " within ", " without ", "
would ",
               " yet ", " you ", " your ", " yours ", " yourself ", " yourselves ",
" the "]

# Iterate over the directories to clean the dataset
for directory in directories:
    # for filename in os.listdir(os.getcwd()):
    for filename in np.ravel(df_categories['file'].loc[df_categories['category'] ==
directory]):

        # Setting up the relative path for the file
        filename = directories[directory] + str(filename) + '.txt'

        # Open the file and read the content
        with open(filename, "r") as inputFile:
            content = inputFile.read()

        # Open the file in writing mode
        with open(filename, "w") as outputFile:

            # Transform the content to lowercase
            content = content.lower()

            # Remove punctuation
            for char in punctuation:
                content = content.replace(char, '')

            # special puntuation
            content = content.replace('’re', ' are')
            content = content.replace('n’t', ' not')
            content = content.replace('s’', 's')
            content = content.replace('-', ' ')

            # Remove stop words
            # content = [w for w in content if not w in stopwords.words("english")]
            for stop_word in stop_words:
                content = content.replace(stop_word, ' ')
```

```
            # Steming
            stemmer = PorterStemmer()
            words = word_tokenize(content)
            stem_words = [stemmer.stem(w) for w in words]
            content = " ".join(stem_words)

            # write the preprocessed content
            outputFile.write(content)
```

É preciso conhecer as dimensões das matrizes *Bag of Words* e *Term-Frequency Matrix* antes de trabalhar com elas para saber se a memória RAM é suficente.

```
# The words in all documents
word_list = []

# Iterate over the directories to find the words in all the documents
for directory in directories:
    for filename in np.ravel(df_categories['file'].loc[df_categories['category'] ==
directory]):

        # Setting up the relative path for the file
        filename = directories[directory] + str(filename) + '.txt'

        # Open the file and read the content
        with open(filename, "r") as inputFile:
            content = inputFile.read()

            # Getting the word of the text in array format
            words = content.split()

            word_list.extend(words)
            word_list = list(set(word_list))

print("Número de palavras: ", len(word_list)) # 30,940
print("Número de documentos: ", df_categories.shape[0]) # 5,000
print("----------------------------------------")

# 2 arrays, each one with 30,940 columns and 5,000 rows
# Storing data in int32 => 30,940 * 5,000 * 2 * 4 bytes ~ 1.2376 GB RAM
```

```
Número de palavras:  30940
Número de documentos:  5000
----------------------------------------
```

Consequentemente, calculamos *Bag of Words* e *Term-Frequency Matrix*.

```
# Bags of words and term frequency matrix
bag_of_words = np.zeros(shape=(df_categories.shape[0], len(word_list)))
tf_matrix = np.zeros(shape=(df_categories.shape[0], len(word_list)))

# Iterate over the directories to do bag of words and TF matrix
for directory in directories:
```

```
    for filename in np.ravel(df_categories['file'].loc[df_categories['category'] ==
directory]):

        # Setting up the relative path for the file
        file_name = directories[directory] + str(filename) + '.txt'

        # Open the file and read the content
        with open(file_name, "r") as inputFile:
            content = inputFile.read()

            # Getting the word of the text in array format
            words = content.split()

            for w in words:
                word_index = word_list.index(w)
                bag_of_words[filename - 1][word_index] = 1
                tf_matrix[filename - 1][word_index] = tf_matrix[filename -
1][word_index] + 1
```

Parte 2 - classificador multiclasse na matriz termo-documento original

É separado aleatoriamente os conjunto de treino(4000 textos) e de teste(1000 textos) para cada matriz.

```
# Classes dataframe
df_classes = np.ravel(df_categories['category'])

# Split the dataset 4000 for training and 1000 for testing randomically
# We need at least 1.2376 GB RAM more to split the data

# Split Bag of Words in test and train data
BW_train, BW_test, BW_categories_train, BW_categories_test = train_test_split(
    bag_of_words, df_classes, test_size=0.2, random_state=1992)

# Split Term Frequency Matrix in test and train data
TF_train, TF_test, TF_categories_train, TF_categories_test = train_test_split(
    tf_matrix, df_classes, test_size=0.2, random_state=1992)
```

Executamos o classificador Naïve Bayes nas matrizes de *Bags of words* and *term frequency*.

```
# Naive Bayes on the Bag of Words
clf_naive_bayes = MultinomialNB()
clf_naive_bayes.fit(BW_train, BW_categories_train)
score_nb_bw = clf_naive_bayes.score(BW_test, BW_categories_test)

# Naive Bayes on the Term Frequency Matrix
# Reusing the classifier to optimize memory
clf_naive_bayes = MultinomialNB()
clf_naive_bayes.fit(TF_train, TF_categories_train)
score_nb_tf = clf_naive_bayes.score(TF_test, TF_categories_test)
```

Executamos o classificador de Regressão Logística nas matrizes de *Bags of words* and *term frequency*. Usando um valor de C=100000 para evitar que haja regularização e usamos paralelização para melhorar os tempos de execução.

```
# Changing C value in Logistic Regression to prevent regularization
param_C=10000

# Improving the performance using parallelization
n_jobs = 3

# Logistic Regression on the Bag of Words
clf_lr = LogisticRegression(C = param_C, n_jobs = n_jobs)
clf_lr.fit(BW_train, BW_categories_train)
score_lr_bw = clf_lr.score(BW_test, BW_categories_test)

# Logistic Regression on the Term Frequency Matrix
# Reusing the classifier to optimize memory
clf_lr = LogisticRegression(C = param_C, n_jobs = n_jobs)
clf_lr.fit(TF_train, TF_categories_train)
score_lr_tf = clf_lr.score(TF_test, TF_categories_test)
```

Finalmente, são apresentados os resultados destes dois classificadores.

```
# Results
print('Acurácia de Naive Bayes em Bag of Words: ', score_nb_bw)
print('Acurácia de Naive Bayes em Term Frequency Matrix: ', score_nb_tf)
print('Acurácia de Logistic Regression em Bag of Words: ', score_lr_bw)
print('Acurácia de Logistic Regression em Term Frequency Matrix: ', score_lr_tf)
print('-----------------------------------------------------------------------')
```

```
Acurácia de Naive Bayes em Bag of Words:  0.79
Acurácia de Naive Bayes em Term Frequency Matrix:  0.834
Acurácia de Logistic Regression em Bag of Words:  0.841
Acurácia de Logistic Regression em Term Frequency Matrix:  0.855
-----------------------------------------------------------------------
```

Parte 3 - classificador multiclasse na matriz termo-documento reduzida

Fizemos uma redução de dimensionalidade usando PCA sobre a matriz de *term frequency*, mantendo o 99% da variância.

```
variance_percentage_pca = 0.99

# PCA in Term Frequency matrix
pca = PCA(n_components = variance_percentage_pca)
pca.fit(TF_train)
```

```
params_reduced_train = pca.transform(TF_train)
params_reduced_test = pca.transform(TF_test)
```

Executamos o classificador SVM sobre a matriz reduzida.

```
# SVM Classifier with RBF kernel
clf_svm = SVC()
clf_svm.fit(params_reduced_train, TF_categories_train)
score_svm = clf_svm.score(params_reduced_test, TF_categories_test)
```

Executamos o classificador Random Forest sobre a matriz reducidad.

```
# Random Forest Classifier
clf_rf = RandomForestClassifier()
clf_rf.fit(params_reduced_train, TF_categories_train)
score_rf = clf_rf.score(params_reduced_test, TF_categories_test)
```

Finalmente, são aprensentados os resultados.

```
print('Classificação em TF Matrix com dados de dimensionalidade reduzida por PCA')
print('Acurácia SVM: ', score_svm)
print('Acurácia Random Forest: ', score_rf)
print('-------------------------------------------------------------------------')
```

```
Classificação em TF Matrix com dados de dimensionalidade reduzida por PCA
Acurácia SVM:  0.82
Acurácia Random Forest:  0.593
-------------------------------------------------------------------------
```