IC - UNICAMP

MC886 - Aprendizado de Máquina

Igor Gustavo Hitzschky Lema, 155758

Exercício 6

Inicialmente, os dados são lidos e as tarefas de processamento de texto usuais são aplicadas:

- Conversão de Maiúsculos para Minúsculos
- Remoção de Pontuação
- Remoção de Stop Words
- Steming de Termos
- Remoção dos Termos que aparecem em apenas 1 documento

Por fim, os textos são convertidos em uma matriz bag of words e uma matriz TF. também, os documentos são divididos em 1000 documentos teste e 4000 treino:

```
categories = ['Apps', 'Enterprise', 'Gadgets', 'Social', 'Startups']
docs = load_files('filesk/', categories=categories, shuffle=True, random_state=42) # Leitura

count_vect = CountVectorizer(analyzer='word', tokenizer=None, stop_words='english',
strip_accents='unicode', lowercase=True, min_df=1) # Tarefas de Processamento de Texto

X_train_counts = count_vect.fit_transform(docs.data) # Matriz Binaria
X_train_tf = TfidfTransformer(use_idf=False).fit_transform(X_train_counts) # Matriz TF

X_train_counts, X_test_counts, y_train_counts, y_test_counts = train_test_split(X_train_counts,
docs.target, test_size=1000, random_state=38)

X_train_tf, X_test_tf, y_train_tf, y_test_tf = train_test_split(X_train_tf, docs.target,
test_size=1000, random_state=38)
```

Foram testados dois técnicas de classificação: O Naive Bayes e o Logistic Regression. Foi executado o Naive Bayes na matriz binária e o Logistic Regression na matriz binária e na matriz TF.

```
# Algoritmo Naive Bayes
def naive_bayes(X_train, X_test, y_train, y_test):
        clf = MultinomialNB().fit(X_train, y_train)
        return clf.score(X_test, y_test)

# Algoritmo Logistic Regression com C = 10000 para evitar regularização
def log_reg(X_train, X_test, y_train, y_test):
        clf = LogisticRegression(C=10000).fit(X_train, y_train)
        return clf.score(X_test, y_test)

print naive_bayes(X_train_counts, X_test_counts, y_train_counts, y_test_counts) # NB na binária
print log_reg(X_train_counts, X_test_counts, y_train_counts, y_test_counts) # logReg na binária
print log_reg(X_train_tf, X_test_tf, y_train_tf, y_test_tf) # logReg na TF
```

As acurácias encontradas foram para o Naive Bayes: **82.9%** na matriz binária. Para o Logistic Regression: **83.7%** para a matriz binária e **85.3%** para a matriz TF.

Como a matriz TF é uma matriz esparsa, para ser possível executar o PCA através da biblioteca do Sklearn, o algoritmo utilizado é o TruncatedSVD que tem o mesmo objetivo, porém é necessário determinar o número exato de componentes. Para descobrir o número de componentes foi executado várias vezes o seguinte código a fim de encontrar um número de componentes aproximado cuja soma da variância é de 99%. Por fim, reduzindo o número de atributos de 42307 para 3000.

```
pca = TruncatedSVD(n_components = 3000).fit(X_train_tf)
print pca.explained_variance_ratio_.sum()
```

A seguir, essa matriz reduzida TF, foi executado o algoritmo SVM com RBF e Random Forest:

```
# Algoritmo Random Forest
def rf(X_train, X_test, y_train, y_test):
        params = {
                'n_estimators':[100, 200, 300, 400],
                'max_features':[10, 15, 20, 25]
        }

        clf = GridSearchCV(RandomForestClassifier(), params)
        clf.fit(X_train, y_train)
        return clf.score(X_test, y_test)

# Algoritmo SVM com RBF
def svm_rbf(X_train, X_test, y_train, y_test):
        params =  {
                'C':[2**-5, 2**0, 2**5, 2**10],
                'gamma':[2**-15, 2**-10,2**-5, 2**0, 2**5]
        }

        clf = GridSearchCV(SVC(), params)
        clf.fit(X_train, y_train)
        return clf.score(X_test, y_test)

print svm_rbf(X_train_tf, X_test_tf, y_train_tf, y_test_tf) # SVM com RBF na matriz TF com PCA
print rf(X_train_tf, X_test_tf, y_train_tf, y_test_tf) # RF na matriz TF com PCA
```

Os resultados dessa etapa foram para o SVM com RBF: **82.8%**. Para o Random Forest: **54.8%**.

A seguir código completo usado para obter os resultados já descritos.

```
# -*- coding: UTF-8 -*-
import numpy as np

from sklearn.datasets import load_files
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.decomposition import TruncatedSVD
from sklearn.model_selection import GridSearchCV
```

```
def naive_bayes(X_train, X_test, y_train, y_test):
        clf = MultinomialNB().fit(X_train, y_train)
        return clf.score(X_test, y_test)

def log_reg(X_train, X_test, y_train, y_test):
        clf = LogisticRegression(C=10000).fit(X_train, y_train)
        return clf.score(X_test, y_test)

def rf(X_train, X_test, y_train, y_test):
        params = {
                'n_estimators':[100, 200, 300, 400],
                'max_features':[10, 15, 20, 25]
        }

        clf = GridSearchCV(RandomForestClassifier(), params)
        clf.fit(X_train, y_train)
        return clf.score(X_test, y_test)

def svm_rbf(X_train, X_test, y_train, y_test):
        params =  {
                'C':[2**-5, 2**0, 2**5, 2**10],
                'gamma':[2**-15, 2**-10,2**-5, 2**0, 2**5]
        }

        clf = GridSearchCV(SVC(), params)
        clf.fit(X_train, y_train)
        return clf.score(X_test, y_test)

categories = ['Apps', 'Enterprise', 'Gadgets', 'Social', 'Startups']
docs = load_files('filesk/', categories=categories, shuffle=True, random_state=42)

count_vect = CountVectorizer(analyzer='word', tokenizer=None, stop_words='english',
strip_accents='unicode', lowercase=True, min_df=1)

X_train_counts = count_vect.fit_transform(docs.data)
X_train_tf = TfidfTransformer(use_idf=False).fit_transform(X_train_counts)

X_train_counts, X_test_counts, y_train_counts, y_test_counts = train_test_split(X_train_counts,
docs.target, test_size=1000, random_state=38)

X_train_tf, X_test_tf, y_train_tf, y_test_tf = train_test_split(X_train_tf, docs.target,
test_size=1000, random_state=38)

print naive_bayes(X_train_counts, X_test_counts, y_train_counts, y_test_counts)
print log_reg(X_train_counts, X_test_counts, y_train_counts, y_test_counts)
print log_reg(X_train_tf, X_test_tf, y_train_tf, y_test_tf)

pca = TruncatedSVD(n_components = 3000).fit(X_train_tf)
X_train_tf = pca.transform(X_train_tf)
X_test_tf = pca.transform(X_test_tf)
print pca.explained_variance_ratio_.sum()

print svm_rbf(X_train_tf, X_test_tf, y_train_tf, y_test_tf)
print rf(X_train_tf, X_test_tf, y_train_tf, y_test_tf)
```