TITULO

Leandro Kümmel Tria Mendes

1 de julho de 2013

1 Introdução

Nesse projeto iremos clusterizar e classificar um conjunto de mensagens, pertencentes a 20 newsgroups[1]. Para isso, primeiro, consideramos a construção de um dicionário[3]. Segundo, a clusterização[4] das mensagens e, por último, a classificação[5] das mesmas.

2 Requisitos

Desenvolvemos o trabalho inteiramente em Python (versão 2.7.1) [2], em um ambiente Linux (distribuição FedoraCore15: Kernel_2.6.43.8-1.i686). Recomenda-se a instalação do **python-pip**[3] e do gcc++. As bibliotecas utilizadas foram:

- **PyEnchant** [4] **& Inflect**[5]: Bibliotecas para processamento textual. A primeira verifica se uma palavra é válida, inclusive se está escrita corretamente. Já a última converte plural para singular, também o contrário. Instalação: PyEnchant *sudo yum install python-enchant.i686* e Inflect *sudo easy_install inflect*

- **scikit-learn**[6]: Biblioteca que contém os algoritmos utilizados para classificação e clusterização das mensagens, tal com K-Means e SVM (Support Vector Machine).

3 Dicionário

As classes diretamente envolvidadas na construção do dicionário são: *Parser, Dictionary e Diretorio.*

- *parser.py*: Possui as duas classes de uso comum, *Parser e Diretorio.* A primeira, possui o atributo *separadores* o qual possui alguns caracteres espciais, os quais não são considerados como uma palavra válida.

```
#!/usr/bin/env python
# encoding: utf-8
from sets import Set #pacote de Set
import os #pacote para leitura do diretorio
import sys #pacote sys para exit
import enchant #verifica se uma palavra eh valida (instalado via yum)
import inflect #verifica plural

class Parser:
    def __init__(self):
        self.separadores = [" ","/",",","*","%","!",".","&","(",")","+","=","{
        self.diretorio = Diretorio()
        self.checkWord = enchant.Dict("en_US")
```

[1] http://revistausenet.com/o-que-e-um-newsgroup-2/
[2] http://www.python.org/
[3] https://pypi.python.org/pypi/pip
[4] http://pythonhosted.org/pyenchant/
[5] https://pypi.python.org/pypi/inflect
[6] http://scikit-learn.org

```
    def getArqs(self):
        return os.listdir(self.diretorio.messages)
class Diretorio:
    def __init__(self):
        self.raiz = "./"
        self.messages = self.raiz+"cluster-txt/messages/"
        self.common = self.raiz+"common/"
```

- *dictionary.py*: Classe que faz o parser da base de textos e seleciona as 100 palavras, de acordo com os critérios apresentados abaixo[**??**]. Alguns trechos relevantes desse arquivos, vale notar o check() presente em *inflect*.

```
###### SELECAO DE PALAVRAS ########
for c in texto:#lendo caracter por caracter no texto
                if c in self.parser.separadores:#o caracter eh um separador
                    if word: #palavra nao pode ser vazia
                        if not word.isdigit():#consideramos alfanumericos e
word = word.lower()#normalizando
                            word = word.decode('iso-8859-1').encode('utf8')
                            if len(word) > 3 and self.parser.checkWord.chec
                                self.totalPalavras += 1
                                if word in col_words:
                                    self.updatePre(word,f)
                                else:
                                    col_words.add(word) #colecao de palavra
                                    word = ""
                            else:
                                word = word + c
########## PROBABILIDADE DE UMA PALAVRA ESTAR EM UM ARQUIVO #########
            p_palavra = float(i[1]) / float(self.totalPalavras) #proporcao
            p_arq = float(i[2]) / float(len(self.arquivos)) #proporcao de
            ppa = float(p_palavra * p_arq) #Prob de selecionar uma Palavra
```

A seleção de palavras para o dicionário, de tamanho igual a 100, seguiu os seguintes critérios listados abaixo.

3.1 Processamento das mensagens

Consideramos que uma palavra é válida quando têm um tamanho maior do que 2 (ou 3[7]) caracteres e não possui alguns caracteres especiais, que estão em *parser.py*[3]. Após feita a essa seleção verificamos se a palavra está escrita corretamente, utilizando o *check() do inflect*.

3.2 Probabilidade da palavra

Cada palavra foi alocada em uma lista e junto a ela há o total de ocorrências dessa em todos os arquivos do newsgroup e o total de arquivos em que ela aparece.
Definimos a probabilidade de selecionarmos, aleatóriamente, uma palavra e esta

[7] considerando diferentes seeds para a clusterizacao com k-means

estar em uma determinada mensagem como sendo:

- *Probabilidade de selecionar uma palavra (pp)* : $\boxed{pp = \frac{\sum_{OcorrenciasdaPalavra}}{\sum_{i=1}^{N}}}$ onde N é o total de palavras, consideradas em [3.1]
- *Probabilidade de selecionar um arquivo (pa)*: $\boxed{pa = \frac{1}{\sum_{i=1}^{M}}}$ onde M é o total de arquivos, mensagens.
- *Probabilidade de selecionar uma palavra e estar em um arquivo (ppa)*: $\boxed{ppa = \frac{pp}{pa}}$

3.3 Descritor

Após a etapa de construção do dicionário [3] implementamos um descritor para cada mensagem. Esse é simplesmente um matriz onde cada linha corresponde a um arquivo e cada coluna corresponde a uma palavra do dicionário. Logo, o valor da matriz na linha L e coluna C é o total de ocorrências da palavra C no arquivo L.
Essa estrutura, a qual denominaremos matriz descritora servirá de entrada para os métodos de clusterização e classificação das mensagens. Ela é gerada pelo arquivo *descriptor.py*, o qual contém a classe *Descriptor*, abaixo citamos os trechos relevantes, observa-se que utiza-se o mesmo método descrito em *processamento das mensagens 3.1*:

```
#### CONTAGEM DAS OCORRENCIAS DAS PALAVRAS POR ARQUIVO ####
for c in texto:#lendo caracter por caracter no texto
                if c in self.parser.separadores:
                    if word:
                        if not word.isdigit():#o caracter eh um separador de pa
                            word = word.lower()#normalizando
                            word = word.decode('iso-8859-1').encode('utf8')#alg
                            if len(word) > 3 and self.parser.checkWord.check(wo
                                iw = self.getIndiceWord(word)
#                                 print str(i)+" "+str(iw)
                                if iw >= 0:
                                    self.descriptor[i][iw] += 1
                    word = ""
                else:
                    word = word + c
```

4 Clusterização

Para a clusterização, ou aglomeração, dos dados utilizamos um algoritmo presente em *scikit-learn [2]* denominado K-Means, o qual objetiva particionar as observações, em K aglomerados[8], onde cada observação pertence ao cluster mais

[8] Nesse caso utilizamos K=20

próximo da média, resultando em um *Diagrama de Voronoi*[9]. O código esta presente na classe *Cluster*, localizada no arquivo *cluster.py*. Como resultado mostramos a matriz de pertinência e fizemos uma análise de sensibilidade para dois seeds diferentes.

```
#!/usr/bin/env python
# encoding: utf-8
# clusterizacao das mensagens utilizando k-means
from pylab import plot,show #plot em grafico
from numpy import vstack,array#array
from numpy.random import rand#randomico
from scipy.cluster.vq import kmeans,vq,whiten #kmeans
from sklearn import metrics
from sklearn import cluster, datasets
#clusterizacao utilizando k-means
class Cluster:
    def __init__(self,s,k):#s sao as seeds para o algoritmo, k o num de cluster
        self.centro = list([])
        self.sens = 0.0#sensibilidade
        self.seeds = s
        self.resp = list(list([]))
        self.matrix = list(list([]))
        self.k=k
    def perform(self):
        print "Start KMeans"
        data = whiten(self.seeds)#normalizando os dados
        self.centro,self.sens = kmeans(data,self.k)
        self.matrix,_ = vq(data,self.centro)
        self.resp = self.centro[self.matrix]
        print "Sensibilidade: "+str(self.sens)
```

5 Classificação

Para a classificação das mensagens utilizamos, também, o *scikit-learn [2]*, porém o algoritmo utilizado foi o *SVM*, ou, *Support Vector Machine*. O SVM utilizado tem como entrada um conjunto descritor e prediz, para cada entrada dada, qual as possíveis classes a entrada faz parte, o que faz do SVM um classificado linear não probabilístico. O código esta presente na classe *Classifier*, localizada no arquivo *classifier.py*. Como resultado mostramos a matriz de confusão e a média de acerto.

```
#!/usr/bin/env python
# encoding: utf-8
# classificacao das mensagens utilizando svm (support vector machines)
from sklearn import svm,metrics #suport vector machine
from sklearn.feature_extraction.text import HashingVectorizer
class Classifier:
    def __init__(self,d,tr,te):
```

[9] http://www.sgsi.com/MIUserGroup/Tech_FindNearest_SGSI.htm

```
        self.training = tr
        self.test = te
        self.data = d
        self.prediction = list([])
    def perform(self):
        s = svm.SVC()#support vector classifier
        s.fit(self.data,self.training)
        self.prediction = s.predict(self.data)
        print self.prediction
        print "Matriz de confusao"
        mc = metrics.confusion_matrix(self.test, self.prediction)
        print mc
        desc = str(s).split('(')[0]
```

6 Resultados

Os resultados estão presentes na pasta *./resultados/*, sendo que, *dictionary* é o dicionário com as 100 palavras, *mpert* é a matriz de pertinência, *mconf* a matriz de confusão e *media* a média de acertos.

7 Conclusão

Observamos que para a construção do dicionário, foi necessário levar em conta o que é realmente uma palavra válida, ou seja, números e caracteres especias não foram considerados. O algoritmo de clusterização, ou k-means, obteve um baixa diferença de sensibilidade para um conjunto diferente de sementes (seeds), para o primeiro obtivemos uma sensibilidade de 5.8438, já para o segundo de 5.1649.