PROJETO DE BLOCO

KDT E KDD

CURSO DE PÓS-GRADUAÇÃO EM BIG DATA

Projeto de avaliação das competências ensinadas durante os módulos Mecanismos de Busca e Mineração de Texto

Antonio Cavalcante de Paula Filho

Luciano Bonfim de Azevedo

Ronei Frota

Bruno Brandão

Professor: Fábio Soares

Coordenador: Eduardo Morelli

Rio de Janeiro, 2017

RESUMO

Esse trabalho tem por objetivo avaliar competências ensinadas durante os módulos Mecanismos de Busca e Mineração de Texto. É um trabalho onde são abordados assuntos referentes as fases de KDT e KDD, métodos e técnicas, aplicadas a uma base de reclamações do consumidor disponibilizada pelo consumidor.gov.br, onde o trabalho consiste em classificar se determinado texto de reclamação é ou não da área de Telecomunicações.

ABSTRACT

This paper aims to evaluate the skills taught during the modules of Search and Text Mining Mechanisms. It is a work where the results of the KDT and KDD phases, methods and techniques are approached, applied to a base of consumer complaints made available by the consumer.gov.br, where this work consists of classifying whether a particular claim is or not a text from the Telecommunications area.

ÍNDICE

1. FUNDAMENTAÇÃO

1.1. CLASSIFICAÇÃO

Segundo Bird et al. (2014), classificação é a tarefa de escolher um *label* de classe corretamente para uma determinada entrada.

Classificação é a tarefa de tornar genérica uma estrutura conhecida para aplicar a novos dados (VILLA et al., 2014).

Nas tarefas de classificação básica, cada entrada é considerada isoladamente de todas as outras entradas e o conjunto de *labels* é definido com antecedência.

Alguns exemplos de tarefas de classificação são:

- Decidir se um e-mail é spam ou não;
- Decidir qual é o tópico de um artigo de notícia, a partir de uma lista fixa de tópicos como "esportes", "tecnologia", "política", etc.;
- Decidir se uma determinada ocorrência da palavra manga está sendo usada para se referir a uma manga de camisa, a fruta, ou ainda a cúpula de um candeeiro.

A tarefa de classificação básica tem algumas variantes interessantes. Por exemplo, na classificação *multi-class*, a cada instância podem ser atribuídos vários rótulos; na classificação *open-class*, o conjunto de rótulos não é definido de antemão; e na *sequence classification*, uma lista de entradas é classificada em conjunto.

Um classificador é chamado supervisionado se ele é construído com base em corpora de treinamento contendo o *label* correto para cada entrada.

A primeira etapa na criação de um classificador é decidir quais recursos da entrada são relevantes e como codificar esses recursos.

A maioria dos métodos de classificação precisam que os recursos sejam codificados usando tipos de valores simples, como booleanos, números e cadeias de caracteres. Mas note que apenas porque um recurso tem um tipo simples, isso não significa necessariamente que o valor do recurso é simples de expressar ou computar. De fato, é até possível usar valores muito

complexos e informativos, como a saída de um segundo classificador supervisionado, como características.

Nesse processo os atributos são separados em dois tipos, os previsores e o atributo alvo, é esse atributo único que é classificado. Cada valor distinto do atributo alvo, normalmente corresponde a um rótulo categórico pertencente a um conjunto pré-definido, no exemplo do e-mail o conjunto seria legítimo ou spam.

A Classificação consiste em descobrir uma função para mapear um conjunto de registros em um conjunto de classes. Uma vez descoberta, essa função pode ser aplicada a novos registros de forma a prever a classe em que os registros se enquadram (GOLDSCHMIDT et al., 2016).

2. PROBLEMA ESPECÍFICO

O problema de classificação textual, portanto KDT, demonstrado nesse trabalho, consiste em encontrar um classificador que consiga prever – dadas entradas textuais de reclamação de um produto e/ou serviços em diversas áreas como: Turismo/Viagens, Produtos de Telefonia e Informática, Transportes, Telecomunicações, etc. - se a reclamação é da área de Telecomunicações ou não. É uma classificação binária.

A base utilizada para esse trabalho é a base de reclamações de consumidores de produtos e serviços em várias áreas, disponibilizada pelo consumidor.gov.br.

A base:
Dicionário de Dados Versão: 1.1
Base de Dados: consumidor.gov.br
Autor: Senacon/MJ
Contato: Suporte Consumidor.gov.br
Data: primeiro semestre de 2014 até 2016 final do segundo semestre de 2016

O Consumidor.gov.br foi lançado em 2014 com a intenção de promover o diálogo efetivo entre consumidores e empresas de diversos segmentos, é um serviço público para solução de conflitos de consumo, é monitorada pelos órgãos de defesa do consumidor e pela Secretaria Nacional do Consumidor do Ministério.

A base usada tem 509998 registros referentes aos anos de 2014, 2015 e 2016 e 19 campos: Região, UF, Cidade, Sexo, Faixa Etária, Data de Finalização, Tempo Resposta, Nome Fantasia, Segmento de Mercado, Área, Assunto, Grupo Problema, Problema, Como Comprou Contratou, Procurou Empresa, Respondida, Situação, Avaliação Reclamação, Nota do Consumidor.

Desses 19 campos, os campos considerados para esse trabalho de classificação foram os campos Área e Problema. Área refere-se a área do problema (Turismo/Viagens, Produtos de Telefonia e Informática, Transportes, Telecomunicações, etc.) e o campo Problema refere-se à descrição do problema, é o campo textual que será usado para o classificador.

Os 509.998 registros da base estão na seguinte distribuição por área:
Educação: 218
Demais Serviços: 7512
Demais Produtos: 22041
Água, Energia, Gás: 3612
Telecomunicações: 219786
Produtos de Telefonia e Informática: 63991
Alimentos: 1081
Transportes: 6856
Produtos Eletrodomésticos e Eletrônicos: 31155
Turismo/Viagens: 2173
Saúde: 3118
Serviços Financeiros: 148455

A figura 1 a seguir, foi retirada do boletim do consumidor.gov.br de 2016, ela mostra que a área de Telecomunicações, em azul, é a que mais tem reclamações. É justamente essa classe que o classificador elaborado nesse trabalho irá classificar.

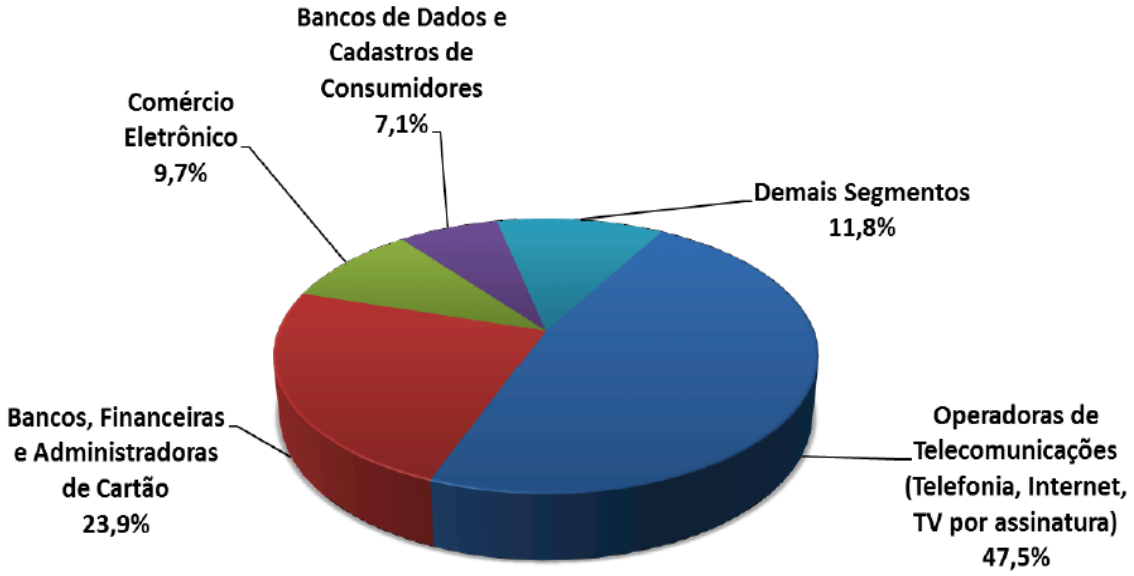

Figura 1: boletim do consumidor.go.br 2016

3. PRÉ-PROCESSAMENTO

Na fase de pré-processamento da base, foram reunidos todos os registros de cinco arquivos diferentes, referentes aos anos de 2014, 2015 e 2016 em um único arquivo CSV, para isso foi utilizado a ferramenta *excel*, esse foi o primeiro pré-processamento.

A segunda fase de pré-processamento foi a eliminação dos campos desnecessários, reduzindo a base que tinha 19 campos para apenas dois campos: Área e Problema.

Para isso foi preciso recorrer a programação em *python*, porque fazendo o processo de remoção das colunas pelo *excel*, tanto em Linux usando o open office, como no Windows utilizando o *excel* do *office* 2016, o separador ponto e vírgula do arquivo era eliminado e substituído por espaço em branco, de forma a impossibilitar a leitura desse arquivo programaticamente.

O código *python* a seguir foi utilizado para eliminar as colunas indesejadas da base total com todos os registros e campos, reduzindo a base a dois campos: Área e Problema.

```python
# encoding: utf-8
import csv

lista = []
#Ler o arquivo CSV com todos os 592895 com seus 19 campos e armazena em lista
with open('C:/path/do/arquivo/2014_2015_2016.csv') as csvfile:
    readCSV = csv.reader(csvfile)
    for row in readCSV:
        if len(row) > 1:
            lista.append(','.join(row).split(";"))
        else:
            lista.append(row[0].split(";"))

with open("C:/path/do/arquivo/2014_2016_dois_campos.csv", 'a') as outcsv:
    #configura a escrita para standard csv file
    writer = csv.writer(outcsv, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL, lineterminator='\n')
    for item in lista:
        #Escreve em 2014_2016_dois_campos.csv apenas os campos 9 (área) e 12 (Problema)
        writer.writerow([item[9], item[12]])

```

Figura 2: Código que elimina os campos desnecessários

Na terceira fase de pré-processamento foi criado um campo adicional binário chamado Classificação com valores zero ou um. Esse campo a mais foi criado através do código *python* mostrado a seguir:

```
# encoding: utf-8
import csv

lista = []
with open('C:/path/para/o/arquivo/2014_2016_dois_campos.csv') as csvfile:
    readCSV = csv.reader(csvfile)
    for row in readCSV:
        if len(row) > 1:
            lista.append(','.join(row).split(";"))
        else:
            lista.append(row[0].split(";"))

listaNova = []
#Adiciona o campo Classificação
lista[0].append('Classificação')
for item in lista[1:]:
    if item[0] == 'Telecomunicações':
        item.append(1)
    else:
        item.append(0)
#Escreve o arquivo 2014_2016_coluna_classificacao.csv adicionando a coluna Classificação
with open("C:/path/para/o/arquivo/2014_2016_coluna_classificacao.csv", 'a') as outcsv:
    #configure writer to write standard csv file
    writer = csv.writer(outcsv, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL, lineterminator='\n')
    for item in lista:
        #Write item to outcsv
        writer.writerow([item[0], item[1], item[2]])

```

Figura 3: Código que adiciona o campo Classificação

A base agora possui três campos: Área, Problema e Classificação, com todos os registros de 2014 até final de 2016.

4. BALANCEAMENTO DA BASE

O balanceamento da base é a quarta fase do pré-processamento, foram considerados apenas 40.000 registros, 20.000 mil de exemplos da classe que se quer classificar (Telecomunicações) e 20.000 das classes de contraexemplos distribuídas da forma mais equilibrada possível, como tem 20.000 registros de contraexemplos e 11 classes, o cálculo é: 20.000/11 que é 1818, valor esse arredondado para 2.000, portanto, cada classe de contraexemplo deve ter preferivelmente 2.000 registros. A base utilizada para classificação tem 39.299 registros.

Os 39.299 registros utilizados para a análise estão na seguinte distribuição por área:

Educação: 218

Demais Serviços: 2000

Demais Produtos: 2000

Água, Energia, Gás: 2000

Telecomunicações: 20000

Produtos de Telefonia e Informática: 2000

Alimentos: 1081

Transportes: 2000

Produtos Eletrodomésticos e Eletrônicos: 2000

Turismo/Viagens: 2000

Saúde: 2000

Serviços Financeiros: 2000

As classes de contraexemplos Educação e Alimentos não alcançaram os 2000 registros, por isso a base ter 39.299 e não 40.000, que seria o mais lógico. No mais, a base está balanceada.

O código a seguir foi usado para gerar a base balanceada:

```python
# encoding: utf-8
import csv
import random

lista = []

with open('C:/path/para/o/arquivo/2014_2016_coluna_classificacao.csv') as csvfile:
    readCSV = csv.reader(csvfile)
    for row in readCSV:
        if len(row) > 1:
            lista.append(','.join(row).split(";"))
        else:
            lista.append(row[0].split(";"))

areas = {'Telecomunicações': [], 'Produtos de Telefonia e Informática': [], 'Educação': [], 'Demais Serviços ': [],
         'Demais Produtos': [], 'Água, Energia, Gás': [],
         'Alimentos': [], 'Transportes': [], 'Produtos Eletrodomésticos e Eletrônicos': [],
         'Turismo/Viagens': [], 'Saúde': [], 'Serviços Financeiros': []}

for item in lista[1:]:
    if item[0] == 'Habitação':
        areas['Demais Produtos'].append(item)
    elif item[2] == '0' and item[0] != 'Área' and len(areas[item[0]]) < 2000:
        areas[item[0]].append(item)
    elif len(areas['Telecomunicações']) < 20000:
        areas['Telecomunicações'].append(item)

listaBalanceada = []
for valor in areas.values():
    for item in valor:
        listaBalanceada.append(item)
random.shuffle(listaBalanceada)

with open("C:/path/para/o/arquivo/2014_2016_balanceada.csv", 'a') as outcsv:
    #configure writer to write standard csv file
    writer = csv.writer(outcsv, delimiter=';', quotechar='|', quoting=csv.QUOTE_MINIMAL, lineterminator='\n')
    writer.writerow(('Área', 'Descrição do problema', 'Classificação'))
    for each_list in listaBalanceada:
        writer.writerow(each_list)
```

Figura 4: Código que gera a base balanceada

5. MINERAÇÃO

Para a mineração o algoritmo usado foi o *Naive-Bayes*, esse algoritmo trabalha com a suposição de independência entre os preditores. Ele assume que a presença de um atributo particular em uma classe não está relacionada com a presença de qualquer outro recurso (Ray, 2016). Por exemplo, um e-mail pode ser considerado spam se contiver a palavra “Viagra”, um artigo é sobre economia se tiver a *string* “PIB”.

Mesmo que esses recursos dependam uns dos outros ou da existência de outras características, todas estas propriedades contribuem de forma independente para a probabilidade de que este e-mail seja spam ou o texto ser de economia.

É um algoritmo bastante útil para grandes conjuntos de dados. Ele é conhecido por gerar métodos de classificação muito sofisticados (Ray, 2016).

5.1. PRÓS E CONTRAS DO *NAIVE-BAYES* (Ray, 2016)

Prós:

- Fácil e rápido de prever o conjunto de dados da classe de teste. Também tem um bom desempenho na previsão de múltiplas classes.
- Quando a suposição de independência prevalece, um classificador *Naive-Bayes* tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento.
- Quando a suposição de independência prevalece, um classificador *Naive-Bayes* tem melhor desempenho em comparação com outros modelos como regressão logística, e você precisa de menos dados de treinamento.

Contras:

- Se uma variável categórica tem uma categoria (nos dados de teste) que não foi verificada no conjunto de dados de treinamento, então o modelo irá atribuir probabilidade 0 (zero) e não sendo capaz de fazer uma previsão. Para resolver isso, pode-se usar a técnica de alisamento. Uma das técnicas mais simples de alisamento é a chamada estimativa de Laplace.
- O *Naive-Bayes* é também conhecido como um mau estimador, por isso, as probabilidades calculadas não devem ser levadas muito a sério.
- Outra limitação do *Naive-Bayes* é a suposição de preditores independentes. Na vida real, é quase impossível ter um conjunto de indicadores que sejam completamente independentes.

O *Naive-Bayes* é muito eficiente para classificação em tempo real, também funciona muito bem para previsão multi-classes.

Ele é muito utilizado principalmente em classificação de textos (devido a um melhor resultado em problemas de classes múltiplas e regra de independência) têm maior taxa de sucesso em comparação com outros algoritmos (Ray, 2016).

O código a seguir mostra a aplicação do *Naive-Bayes* na base do projeto.

```
# encoding: utf-8
import csv
import string
import random
import operator
import re
from functools import reduce
import nltk
from nltk import word_tokenize

lista = []
with open('C:/Users/toti.cavalcanti/Downloads/Compressed/2014_2016_balanceada.csv') as csvfile:
    readCSV = csv.reader(csvfile)
    for row in readCSV:
        if len(row) > 1:
            lista.append(','.join(row).split(";"))
        else:
            lista.append(row[0].split(";"))

#lista de stopwords em português
stopwords = nltk.corpus.stopwords.words('portuguese')
stopwords.append('/')
stopwords.append("-")
stopwords.append(',')
stopwords.append("(")
stopwords.append(")")

lista_de_tuplas = []
todas_palavras = []
for item in lista[1:]:
    lista_aux = []
    item[1] =  word_tokenize(item[1].lower())
    item[1] = [word for word in item[1] if not re.fullmatch('[' + string.punctuation + ']+', word)]
    item[1] = [word for word in item[1] if word not in stopwords]
    #alguns registros tem palavras com a '/' colada na última letra
    #impossibilitando que a '/' não seja retirada pelo processo de stopword
    #por isso o 'for' abaixo
    for p in item[1]:
        if p[-1] == '/':
            item[1][item[1].index(p)] = p.replace('/', ' ')
    lista_aux.append(item[1])
```

```
42      lista_aux.append(item[2])
43      lista_de_tuplas.append(tuple(lista_aux))
44      todas_palavras.append(item[1])
45
46  #reduce faz um flat da lista de lista
47  todas_palavras = reduce(operator.add, todas_palavras)
48
49  #FreqDist({'palavra': frequência})
50  distribuicao_frequencia_todas_palavras = nltk.FreqDist(todas_palavras)
51  word_features = list(distribuicao_frequencia_todas_palavras)[:2000]
52
53  def document_features(document):
54      document_words = set(document)
55      features = {}
56      for word in word_features:
57          features['contains({})'.format(word)] = (word in document_words)
58      return features
59
60  featuresets = [(document_features(d), c) for (d,c) in lista_de_tuplas]
61  train_set, test_set = featuresets[11789:], featuresets[:11789]
62  classifier = nltk.NaiveBayesClassifier.train(train_set)
63  nltk.classify.accuracy(classifier, train_set)
64
65  classifier.show_most_informative_features(5)
```

Figura 5: Código do teste da base com *Naive-Bayes*

Para a mineração, a base foi separada em dois conjuntos, 70% para treinamento e 30% para teste.

70% da base 39299 * 0.7 = 27510

30% da base 39299 * 0.3 = 11789

As entradas para o *Naive-Bayes* foram as ocorrências ou não de palavras no corpo de cada texto da descrição do problema e a importância de cada palavra no corpus total, isto é, todas as descrições de problemas usando distribuição de frequências das palavras.

Com isso, o algoritmo prevê a probabilidade de um documento contendo determinada palavra, ser ou não da área de Telecomunicações.

Treino da base com NaiveBayesClassifier:

comando: nltk.classify.accuracy(classifier, train_set)

Resultado: 0.8108687749909124

Rodando o comando abaixo e pegando algumas saídas:

:

classifier.show_most_informative_features(200)

Most Informative Features

contains(desligamento) = True 1 : 0 = 353.8 : 1.0

contains(indevido) = True 1 : 0 = 353.8 : 1.0

contains(garantia) = True 0 : 1 = 260.5 : 1.0

contains(funcionamento) = True 1 : 0 = 227.1 : 1.0

contains(inadequado) = True 1 : 0 = 226.1 : 1.0

Teste da base com NaiveBayesClassifier

comando: nltk.classify.accuracy(classifier, test_set)

resultado: 0.8055814742556621

Rodando o comando abaixo e pegando algumas saídas:

classifier.show_most_informative_features(200)

Resultado:

Most Informative Features

Os termos abaixo em verde têm alta probabilidade de acerto.

contains(desligamento) = True 1 : 0 = 353.8 : 1.0

contains(indevido) = True 1 : 0 = 353.8 : 1.0

contains(garantia) = True 0 : 1 = 260.5 : 1.0

contains(funcionamento) = True 1 : 0 = 227.1 : 1.0

contains(inadequado) = True 1 : 0 = 226.1 : 1.0

contains(indevido) = True 1 : 0 = 353.8 : 1.0

contains(desligamento) = True 1 : 0 = 353.8 : 1.0

Os termos abaixo em vermelho têm baixa probabilidade de acerto.

contains(solicitação) = False 0 : 1 = 1.0 : 1.0

contains(alteração) = False 0 : 1 = 1.0 : 1.0

Interpretação dos resultados:

A palavra 'desligamento', aparece 353.8 vezes mais em textos da área de Telecomunicações do que de outra área.

A palavra ‘garantia’, aparece 260,5 vezes mais em textos de outra área do que da área de Telecomunicações.

A palavra ‘indevido’, aparece 353.8 vezes mais em textos da área de Telecomunicações do que de outra área.

E assim por diante.

É possível também concluir que os termos "solicitação" e "alteração", aparecem em reclamações de todas as áreas, por isso, possuem baixa ou zero probabilidade de serem identificadas corretamente.

6. CONCLUSÃO

O que foi percebido foi que o algoritmo de mineração de dados supervisionado *Naive-Bayes* funcionou bem para a classificação proposta, classificar a descrição de um problema como sendo da área de telecomunicações ou não.

O algoritmo trabalhou com textos pequenos, conseguiu um nível bem razoável de acurácia, tanto com o conjunto de dados de treino como com o de testes, com uma variação um pouco abaixo no conjunto de teste, mas sem disparate entre os resultados das duas bases, conseguiu um resultado bem satisfatório.

REFERÊNCIAS

BIRD, STEVEN; KLEIN, EWAN; LOPER EDWARD, 2014, **NLTK, NATURAL LANGUAGE TOOL KIT, PROCESSING WITH PYTHON.** Disponível em: < http://www.nltk.org/book/ >. Acesso em: 29 de março de 2017, às 14:17 horas.

GOLDSCHMIDT, RONALDO; BEZERRA, EDUARDO, 27 de Junho de 2016 - 08h35, **EXEMPLOS DE APLICAÇÕES DE DATA MINING NO MERCADO BRASILEIRO**. Disponível em: <http://computerworld.com.br/exemplos-de-aplicacoes-de-data-mining-no-mercado-brasileiro>. Acesso em: 13 de março de 2017, às 13:49 horas.

VILLA, ALLAN CARLOS CLAUDINO; CANDIDO JUNIOR, ELI, 2014, **UTILIZAÇÃO DE INFORMAÇÕES PARA ADQUIRIR CONHECIMENTO ATRAVÉS DA MINERAÇÃO DE DADOS**. Centro universitário Toledo Prudente, ETIC, encontro de iniciação científica. ISSN 21-76-8498.

RAY, SUNIL, 21 de abril de 2016, **6 EASY STEPS TO LEARN NAIVE BAYES ALGORITHM (WITH CODE IN PYTHON)**. Disponível em: <https://www.vooo.pro/insights/6-passos-faceis-para-aprender-o-algoritmo-naive-bayes-com-o-codigo-em-python/>. Acesso em: 03 de abril de 2017, às 20:17 horas.