**Laboratório Virtual - Atividade 2**

|  |  |
| --- | --- |
| **Disciplina** | **RDP – Reconhecimento de Padrões** |

**Objetivos**

A segunda atividade prática em laboratório virtual possui como objetivos principais:

* Realizar um problema de classificação real;
* Criar um classificador de spams;
* Avaliar o desempenho do classificador;

Os dados de entrada são um conjunto de e-mails previamente classificadas como "spam" ou “não spam”[[1]](#footnote-1). O objetivo do exercício é construir um modelo baseado no algoritmo Naïve Bayes para, com base no conteúdo do e-mail, identificar à qual classe uma determinada mensagem pertence. Ao final, faremos a avaliação do desempenho do preditor usando a matriz de confusão e a métrica de acurácia.

**Abrindo um novo Notebook no Jupyter**

Os arquivos do nosso projeto estão estruturados dentro das pastas da seguinte forma:

diretorio-projeto

└ src

└ SpamClassifier.ipynb

└ data

└ spam

└ ham

Nosso primeiro passo é abrir o arquivo SpamClassifier para iniciar o desenvolvimento do nosso projeto. Vamos começar?

**Fazendo a leitura do Dataset**

Os dados disponíveis se encontram no formato original e cada arquivo dentro de cada uma das pastas é um e-mail diferente classificado, e verificado, previamente como spam ou não. Os 2500 e-mails que se encontram dentro da pasta “ham” identificam e-mails que foram identificados previamente como NÃO SPAM. Já os 500 e-mails que se encontram dentro da pasta “spam” foram identificados, previamente, como spams.

Nosso primeiro passo é extrair, de cada e-mail, sua mensagem. Para isso, faremos uma função que lerá cada arquivo e identificará onde se localiza a mensagem para, então, extraí-la.

|  |
| --- |
| import os *# biblioteca para acessar funções do S.O.*  import io *# biblioteca para manipulação de arquivos*  *#função para a extração da mensagem de cada arquivo de e-mail*  def readFiles(path):  for root, dirnames, filenames in os.walk(path):  for filename in filenames:  *# encontra o arquivo*  path = os.path.join(root, filename)  inBody = False  lines = []  *# abre o arquivo*  f = io.open(path, 'r', encoding='latin1')  *# le o arquivo até encontra o corpo da mensagem*  for line in f:  if inBody:  lines.append(line)  elif line == '\n':  inBody = True  f.close()  message = '\n'.join(lines)  *# retorna os valores*  yield path, message |

Agora que já conseguimos recuperar os dados das mensagens de cada arquivo temos que lê-los e, para facilitar a manipulação dados, vamos coloca-los em uma estrutura na memória: o DataFrame. Para utilizar o DataFrame precisamos da biblioteca pandas disponível no python. Faremos então uma função que lê cada arquivo dentro de um diretório e retorna, ao final, um DataFrame com os dados de todos.

|  |
| --- |
| from pandas import DataFrame  def dataFrameFromDirectory(path, classification):  rows = []  index = []  for filename, message in readFiles(path):  rows.append({'message': message, 'class': classification})  index.append(filename)  return DataFrame(rows, index=index) |

Com isso podemos criar nossos DataFrames. Teremos uma estrutura com todos os spams e não spams

|  |
| --- |
| *# coloque aqui o caminho do diretório onde se encontram os e-mails na sua máquina*  path = "C:/Users/.../igti-rdp/igti-rdp-pratica2"  *# cria o DataFrame com os spams 'spam' e dos não spam 'ham'*  data = DataFrame({'message': [], 'class': []})  data = data.append(dataFrameFromDirectory(path+'/emails/spam', 'spam'))  data = data.append(dataFrameFromDirectory(path+'/emails/ham', 'ham'))  *# se quisermos visualizar o conteúdo do DataFrame podemos invocar a função head*  data.head() |

**Realizando o Treinamento do Modelo**

O scikit-learn possui uma quantidade enorme de possibilidade de operações sobre os dados. Uma das estratégias mais comuns para a identificação de spams é a contagem da frequência das palavras de cada mensagem. Para o nosso problema utilizaremos:

* a estrutura **CountVectorizer** permite criar um processador de texto para transformar a mensagem em uma matriz com as palavras e a frequência com que cada palavra aparece.
* a estrutura **MultinomialNB** é um classificador Naive Bayes implementado em python. Para utilizá-lo precisamos importar as estruturas da biblioteca sklearn.

Como o nosso DataFrame data possui as duas colunas (‘message’ e ‘class’) utilizamos o CountVectorizer to contar a frequência de cada palavra do texto da mensagem e treinamos o classificador Naive Bayes com essa contagem (counts) e suas saídas esperadas (targets) através da função de treinamento *classifier.fit()*

|  |
| --- |
| from sklearn.feature\_extraction.text import CountVectorizer  from sklearn.naive\_bayes import MultinomialNB  *# cria o vetorizador de palavras da mensagem*  vectorizer = CountVectorizer()  *# conta a frequencia de cada palavra em cada mensagem*  counts = vectorizer.fit\_transform(data['message'].values)  *# cria o classificador Naive Bayes*  classifier = MultinomialNB()  *# define qual a saida correta para cada mensagem*  targets = data['class'].values  *# realiza o treinamento do classificador*  classifier.fit(counts, targets) |

**Exemplo de Classificação**

Agora que temos nosso classificador treinado podemos testá-lo. Para exemplificar como o classificador criamos dois e-mails: um que se parece com um spam e outro não.

|  |
| --- |
| *#mensagens de exemplo*  spam = ['Free Viagra now!!!', "Hi Bob, how about a game of golf tomorrow?"]  notspam = ['Job Interview', "Hi Bob, are you free 10am tomorrow? We have an open hour and your test was rescheduled. Please dont be late"] |

Talvez sejam essas as tarefas mais simples. Tudo o que precisamos é invocar a função **predict** para gerarmos as previsões sobre a base de treinamento.

|  |
| --- |
| *# conta a frequencia das palavras da mensagem*  example\_counts = vectorizer.transform(spam)  *# realiza a predição*  predictions = classifier.predict(example\_counts)  *# mostra a primeira classe*  predictions[0] |

**Tarefa: Validação**

Por fim, devemos realizar a validação das predições realizadas. Para isso, temos que testar o desempenho do classificador sobre a própria base. Nossa tarefa então é:

1. Separar o DataFrame em dois: um DataFrame de treinamento (datatraining) que corresponde a 70% do tamanho total da base de spams e não spams (cuidado para manter a proporção de spams e não spams) e um DataFrame de validação (datatest) com os 30% restantes.
2. Utilizar datatraining para treinar o classificador Naïve Bayes e realize a classificação de todos os e-mails do datatest
3. Calcule a Acurácia do preditor comparando a classificação de cada e-mail do datatest com o rótulo atribuido inicialmente

**Conclusão**

Nessa segunda atividade aprendemos como criar um classificador. Entendemos como transformar os dados e estrutura-los em campos distintos. Aprendemos como criar um Pipeline de execução e avaliar os resultados da execução do algoritmo.

1. Solução adaptada do exemplo disponibilizada por Sundog Education (<http://sundog-education.com/>) [↑](#footnote-ref-1)