1 Identificando Spam

Nesta prática vamos usar o modelo para classificar alguns e-mails como sendo ou não spam. Todo o texto foi baseado no livro Data Science do Zero de Joel Grus com algumas modificações.

1.1 Download

Um conjunto de dados bem popular (mas um pouco antigo) que pode ser usado para treinar e testar o modelo é o corpus público do SpamAssassin (`https://spamassassin.apache.org/old/publiccorpus/`). Nesta prática vamos iniciar usando os arquivos de prefixo 20021010.

O primeiro bloco de código é o responsável por fazer o download e descompactar os arquivos com e-mails para treinamento e teste (listagem 1).

```
from io import BytesIO  # Agora podemos tratar bytes como um arquivo
import requests         # Para baixar os arquivos, que
import tarfile          # estão no formato .tar.bz.

BASE_URL = "https://spamassassin.apache.org/old/publiccorpus"
FILES = [       "20021010_easy_ham.tar.bz2",
                "20021010_hard_ham.tar.bz2",
                "20021010_spam.tar.bz2"]

# Os dados ficarão aqui,
# nos subdiretórios /spam, /easy_ham e /hard_ham
# Altere para o diretório escolhido.
OUTPUT_DIR = 'spam_data'

for filename in FILES:
        # Use solicitações para obter o conteúdo dos arquivos em cada
        ↪  URL.
        content = requests.get(f"{BASE_URL}/{filename}").content

        # Encapsule os bytes na memóriapara usálos como um "arquivo"
        fin = BytesIO(content)

        # E extraia todos os arquivos para o diretório de saída
        ↪  especificado.
        with tarfile.open(fileobj=fin, mode='r:bz2') as tf:
                tf.extractall(OUTPUT_DIR)
```

Exemplo 1: Script *download.py* para download dos dados de treinamento e teste.

1.2 Identificando o assunto

Depois de baixar os dados existirão três pastas: *spam*, *easy_ham* e *hard_ham*. Cada pasta contém vários e-mails, e cada e-mail fica em um arquivo. Para simplificar bastante, no exemplo de código é examinado apenas a linha de assunto de cada e-mail. Para identificar a linha de assunto procura-se o termo "Subject:" (listagem 2).

```
from naivebayesclassifier import *
from fileinput import filename
import glob
from typing import List

# Vamos usar a linha de assunto na identificação do spam

# modifique o caminho para indicar o local dos arquivos
path = 'spam_data/*/*'

data: List[Message] = []

# glob.glob retorna todos os nomes de arquivos que correspondem ao
↪  caminho com curinga
for filename in glob.glob(path):
        is_spam = "ham" not in filename

        # Existem alguns caracteres de lixo nos e-mails;
        # o errors='ignore' os ignora em vez de gerar uma exceção.
        with open(filename, errors='ignore') as email_file:
                for line in email_file:
                        if line.startswith("Subject:"):
                                subject = line.lstrip("Subject: ")
                                data.append(Message(subject, is_spam))
                                break # arquivo finalizado
```

Exemplo 2: Código do script *run.py* para encontrar as linhas com "Subject:".

1.3 Separando os dados

O próximo passo é dividir o conjunto de dados como dados de treinamento e dados de teste para, em seguida, criar o classificador (listagem 3. Note que é preciso adicionar o arquivo `https://raw.githubusercontent.com/joelgrus/data-science-from-scratch/master/scratch/machine_learning.py` na pasta *scratch*.

```
# Dividimos o conjunto de dados em dados de treinamento
# e de teste para usar o classificador

import random
from scratch.machine_learning import split_data

random.seed(0)  # Para que você chegue aos mesmos resultados que o autor
↪  do livro
train_messages, test_messages = split_data(data, 0.75)

model = NaiveBayesClassifier()
model.train(train_messages)
```

Exemplo 3: Código do script *run.py* para separar o conjunto de dados.

1.4 Executando o classificador

O bloco de código da listagem 4 realiza algumas classificações sobre os dados. Note que para cada linha da *confusion_matrix* a primeira coluna da tupla é o estado verdadeiro e a segunda é a previsão do classificador.

```
# Classificando algumas mensagens como spam para verificar o
↪  funcionamento do modelo
from collections import Counter

predictions = [(message, model.predict(message.text))
               for message in test_messages]

# Presuma que spam_probability > 0.5 corresponde à previsão de spam
# e conte as combinações de (real is_spam, previsto is_spam)
confusion_matrix = Counter((message.is_spam, spam_probability > 0.5)
               for message, spam_probability in predictions)

print(confusion_matrix)
```

Exemplo 4: Código do script *run.py* para classificar os e-mails.

1.5 Inspecionando o modelo

Por fim, o último bloco de código na listagem 5 serve para inspecionar o interior do modelo e exibir as palavras mais e menos indicativas de spam:

```python
# Inspecionando o interior do modelo para determinar as palavras menos e
# mais indicativas de spam:

def p_spam_given_token(token: str, model: NaiveBayesClassifier) ->
↪  float:
        # Aqui, não recomento chamar métodos privados, mas é por uma boa
        ↪  causa.
        prob_if_spam, prob_if_ham = model._probabilities(token)

        return prob_if_spam / (prob_if_spam + prob_if_ham)

words = sorted(model.tokens, key=lambda t: p_spam_given_token(t, model))

print("spammiest_words", words[-10:])
print("hammiest_words", words[:10])
```

Exemplo 5: Código do script *run.py* para encontrar as palavras mais e menos indicativas de spam.

2 Atividade

Adicione os seguintes melhoramentos no código anterior.

1. Adicione suporte a todos os arquivos da série 20030228.
2. Analise o conteúdo da mensagem e não somente o assunto. Obviamente você deve excluir o cabeçalho da sua análise.
3. O classificador analisa todas as palavras do conjunto de treinamento, até as que só aparecem uma vez. Modifique o classificador para aceitar um limite opcional min_count e ignorar os tokens que não aparecerem, pelo menos, esse número de vezes.
4. Escreva um pequeno texto explicando quais modificações você fez.