Exemplo: Naive Bayes para Filtros de Spam

Linguagem de Programação AplicadaSemana 2 / Parte 1

Prof. Alex Kutzke

17 de abril 2021

Introdução ao Naive Bayes

Introdução

- **Naive Bayes** é o nome que se dá à técnica de classificação baseada, entre outras coisas, no *Teorema de Bayes*;
- Veremos a seguir uma pequena aplicação dessa técnica para a classificação de emails entre Spam e Não Spam;

Teorema de Bayes

- Antes, para relembrar, segue o teorema de Bayes:

‘$P(E|F) = \frac{P(F|E)P(E)}{P(F)}$‘

Ou

‘$P(E|F) = \frac{P(F|E)P(E)}{[P(F|E)P(E)+P(F|\neg E)P(\neg E)]}$‘

- Em palavras simples: o teorema nos informa sobre a probabilidade de um evento E condicionado a F sendo que só temos informações iniciais sobre o evento F condicionado a E.
- Por exemplo, se sabemos a probabilidade de um email ser SPAM e conter a palavra X, gostaríamos de saber qual a probabilidade de um email ter a palavra X e ser SPAM.

Um Filtro de Spam Muito Simples

Cenário

- Considere os seguintes eventos:
  - ‘S‘: “a mensagem é spam”;
  - ‘B‘: “a mensagem contém a palavra *bitcoin*”;

- Segundo o Teorema de Bayes, temos:

‘$P(S|B) = \frac{P(B|S)P(S)}{[P(B|S)P(S)+P(B|\neg S)P(\neg S)]}$‘

- Numerador: probabilidade de a mensagem ser spam e conter “bitcoin”;
- Denominador: probabilidade de a mensagem conter “bitcoin”;

Aplicando o modelo

- Se temos uma grande coleção de mensagens classificadas como spam e não-spam:
  - Podemos calcular ‘$P(B|S)$‘ e ‘$P(B|\neg S)$‘.
- Se considerarmos que qualquer mensagem é igualmente provável de ser spam ou não-spam:
  - Temos que ‘$P(S) = P(\neg S) = 0.5$‘

‘$P(S|B) = \frac{P(B|S)}{[P(B|S)+P(B|\neg S)]}$‘

Exemplo

- Por exemplo, se:
  - ‘50%‘ das mensagens spam possuem a palavra “bitcoin”;
  - ‘1%‘ das mensagens não-spam possuem a palavra “bitcoin”;

‘$P(S|B) = \frac{P(B|S)}{[P(B|S)+P(B|\neg S)]}$‘

‘$P(S|B) = \frac{0.5}{(0.5+0.01)} = 98\%$‘

A probabilidade de qualquer email que contenha “bitcoin” seja spam é: ‘98%‘

Um Filtro de Spam Mais Sofisticado

Ampliando o Contexto

- Suponha um vocabulário de palavras ‘$w_1, \ldots, w_n$‘;
- ‘X_i‘ é o evento “mensagem contém palavra ‘w_i‘;
- ‘$P(X_i|S)$‘ é a probabilidade de uma **mensagem spam** conter a palavra ‘w_i‘;
- ‘$P(X_i|\neg S)$‘ é a probabilidade de uma **mensagem não-spam** conter a palavra ‘w_i‘;

A Suposição de Naive Bayes

- A técnica *Naive Bayes* se baseia em uma suposição um tanto inocente:
  - A presenças (ou ausências) de cada palavra são independentes uma das outras;
  - Obviamente isso é uma grande simplificação;
  - Entretanto, mesmo assim, Naive Bayes apresenta bons resultados.

- Isso significa, em outras palavras, por exemplo, que saber que uma mensagem contém ou não "bitcoin" não nos informa em nada sobre se ela contém ou não a palavra "carro".

‘$P(X_1 = x_1, \ldots, X_n = x_n|S) = P(X_1 = x_1|S) \times \ldots \times P(X_n = x_n|S)$‘

Explicando

- Se todo nosso vocabulário fosse composto por apenas "bitcoin" e "carro";
- Se metade das **mensagens de spam** contém "bitcoin" e a outra metade contém "carro";
- Assim, a técnica Naive Bayes nos diz que a probabilidade de uma mensagem spam conter ambas as palavras é:

‘$P(X_{bitcoin} = 1, X_{carro} = 1|S) = P(X_{bitcoin} = 1|S)P(X_{carro} = 1|S) =$ ‘

‘$= .5 \times .5 = .25$‘

Juntando tudo

- Segundo o nosso primeiro filtro, sabemos que podemos utilizar a seguinte equação para calcular a probabilidade de uma mensagem ser spam dado que contém uma palavra X:

‘$P(S|X = x) = \frac{P(X=x|S)}{[P(X=x|S)+P(X=x|\neg S)]}$‘

- Considerando a suposição de Naive Bayes:
  - Podemos calcular as probabilidades da direita multiplicando as probabilidades associadas a cada uma das palavras independentemente;

Detalhes antes da implementação (1)

- Multiplicação de muitas probabilidades pode causar *underflow* (números excessivamente pequenos);
- Como **todos lembram**:
  - ‘$\log(ab) = \log(a) + \log(b)$‘
  - ‘$exp(\log(x)) = x$‘
- Assim, podemos substituir a multiplicação ‘$p_1 \times \ldots \times p_n$‘ por:
  - ‘$exp(\log(p_1) + \ldots + \log(p_n))$‘

Detalhes antes da implementação (2)

- Ao calcular ‘$P(X_i|S)$‘ e ‘$P(X_i|\neg S)$‘:
  - Se a palavra "dado", por exemplo, ocorre **apenas** em mensagens não-spam, então ‘$P(dado|S) = 0$‘;
- Nosso classificador irá atribuir probabilidade 0 de spam para qualquer mensagem que contenha "dado" (por quê?);
  - Mesmo uma que contenha "dado que o bitcoin . . . ";

- Portanto, é necessário suavizar as probabilidades, para fugirmos de valores extremos (1 e 0);
- Para isso, utilizaremos um coeficiente 'k' e a seguinte equação adaptada:

'$P(X_i|S) = \frac{(k+número\_mensagens\_contendo\_w_i)}{2k+número\_de\_spams}$'

- O mesmo para '$P(X_i|\neg S)$'

Implementação

Divisão em palavras

```
from typing import Set
import re

def tokenize(text: str) -> Set[str]:
    text = text.lower()                            # Convert to lowercase,
    all_words = re.findall("[a-z0-9']+", text)  # extract the words, and
    return set(all_words)                          # remove duplicates.

assert tokenize("Data Science is science") == {"data", "science", "is"}
```

Tipo para Mensagem

```
from typing import NamedTuple

class Message(NamedTuple):
    text: str
    is_spam: bool
```

Nosso classificador

```
from typing import List, Tuple, Dict, Iterable
import math
from collections import defaultdict

class NaiveBayesClassifier:
    def __init__(self, k: float = 0.5) -> None:
        self.k = k  # smoothing factor

        self.tokens: Set[str] = set()
        self.token_spam_counts: Dict[str, int] = defaultdict(int)
        self.token_ham_counts: Dict[str, int] = defaultdict(int)
        self.spam_messages = self.ham_messages = 0
```

Treinando o modelo

```
# Dentro da classe NaiveBayesClassifier

    def train(self, messages: Iterable[Message]) -> None:
        for message in messages:
            # Increment message counts
            if message.is_spam:
                self.spam_messages += 1
            else:
                self.ham_messages += 1

            # Increment word counts
            for token in tokenize(message.text):
                self.tokens.add(token)
                if message.is_spam:
                    self.token_spam_counts[token] += 1
                else:
                    self.token_ham_counts[token] += 1
```

Cálculo das Probabilidades

```
# Dentro da classe NaiveBayesClassifier

    def probabilities(self, token: str) -> Tuple[float, float]:
        """returns P(token | spam) and P(token | ham)"""
        spam = self.token_spam_counts[token]
        ham = self.token_ham_counts[token]

        p_token_spam = (spam + self.k) / (self.spam_messages + 2 * self.k)
        p_token_ham = (ham + self.k) / (self.ham_messages + 2 * self.k)

        return p_token_spam, p_token_ham
```

Calculo das probabilidades de uma mensagem

```
# Dentro da classe NaiveBayesClassifier
    def predict(self, text: str) -> float:
        text_tokens = tokenize(text)
        log_prob_if_spam = log_prob_if_ham = 0.0

        # Iterate through each word in our vocabulary
        for token in self.tokens:
            prob_if_spam, prob_if_ham = self.probabilities(token)

            # If *token* appears in the message,
```

```
            # add the log probability of seeing it
            if token in text_tokens:
                log_prob_if_spam += math.log(prob_if_spam)
                log_prob_if_ham += math.log(prob_if_ham)

            # Otherwise add the log probability of _not_ seeing it,
            # which is log(1 - probability of seeing it)
            else:
                log_prob_if_spam += math.log(1.0 - prob_if_spam)
                log_prob_if_ham += math.log(1.0 - prob_if_ham)

        prob_if_spam = math.exp(log_prob_if_spam)
        prob_if_ham = math.exp(log_prob_if_ham)
        return prob_if_spam / (prob_if_spam + prob_if_ham)
```

Testando o modelo

```
messages = [Message("spam rules", is_spam=True),
            Message("ham rules", is_spam=False),
            Message("hello ham", is_spam=False)]

model = NaiveBayesClassifier(k=0.5)
model.train(messages)

# Verificar contagens
assert model.tokens == {"spam", "ham", "rules", "hello"}
assert model.spam_messages == 1
assert model.ham_messages == 2
assert model.token_spam_counts == {"spam": 1, "rules": 1}
assert model.token_ham_counts == {"ham": 2, "rules": 1, "hello": 1}
```

Testando o modelo - parte matemática

```
text = "hello spam"

probs_if_spam = [
    (1 + 0.5) / (1 + 2 * 0.5),      # "spam"  (present)
    1 - (0 + 0.5) / (1 + 2 * 0.5),  # "ham"   (not present)
    1 - (1 + 0.5) / (1 + 2 * 0.5),  # "rules" (not present)
    (0 + 0.5) / (1 + 2 * 0.5)       # "hello" (present)
]

probs_if_ham = [
    (0 + 0.5) / (2 + 2 * 0.5),      # "spam"  (present)
    1 - (2 + 0.5) / (2 + 2 * 0.5),  # "ham"   (not present)
    1 - (1 + 0.5) / (2 + 2 * 0.5),  # "rules" (not present)
```

```
    (1 + 0.5) / (2 + 2 * 0.5),      # "hello" (present)
]

p_if_spam = math.exp(sum(math.log(p) for p in probs_if_spam))
p_if_ham = math.exp(sum(math.log(p) for p in probs_if_ham))

# Should be about 0.83
assert model.predict(text) == p_if_spam / (p_if_spam + p_if_ham)
```

Lendo mensagens reais

- Dados de https://spamassassin.apache.org/old/publiccorpus

```
import glob, re

# modify the path to wherever you've put the files
path = 'spam_data/*/*'

data: List[Message] = []

# glob.glob returns every filename that matches the wildcarded path
for filename in glob.glob(path):
    is_spam = "ham" not in filename

    # There are some garbage characters in the emails; the errors='ignore'
    # skips them instead of raising an exception.
    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  # done with this file
```

Treino X Teste

```
import random
from scratch.machine_learning import split_data

random.seed(0)      # just so you get the same answers as me
train_messages, test_messages = split_data(data, 0.75)

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

Testando

```
from collections import Counter

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

# Assume that spam_probability > 0.5 corresponds to spam prediction
# and count the combinations of (actual is_spam, predicted is_spam)
confusion_matrix = Counter((message.is_spam, spam_probability > 0.5)
                           for message, spam_probability in predictions)

print(confusion_matrix)
```

Investigando efeito das palavras

```
def p_spam_given_token(token: str, model: NaiveBayesClassifier) -> float:
    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])
```

Possibilidades para melhorar o modelo

- Analisar o conteúdo da mensagem e não apenas o Assunto;
- Considerar apenas palavras que aparecem um número mínimo de vezes (`min_count`);
- Utilizar apenas radicais das palavras (pesquise por "Porter Stemmer");
- Considerar não apenas presença de palavras, mas outras características:
  - Por exemplo, se a mensagem possuí números:
    - A função `tokenizer` pode retornar *tokens* especiais para isso (por exemplo: `contains:number`).

Referências

- GRUS, Joel - Data Science do Zero: Primeiras Regras com Python, Editora Alta Books, 1a Edição, 2016;