

🔑 master ▾

lab-ead / src / Lab 12 - Model Bayesa.md

Go to file

...



phosphide Dodane rozszerzeni...

Latest commit f06ef92 3 minutes ago

🕒 History

👤 1 contributor

144 lines (110 sloc) | 5.75 KB

Raw

Blame



Lab 12 - Operacje na tekście i naiwny Algorytm Bayesa (naive Bayes)

Dataset

Dataset zawiera zestaw krótkich wiadomości tekstowych (sms), który został poetykietowany jako spam i ham. Zestaw pochodzi z [bazy UCI](#) i można go pobrać z [SMSSpamCollection](#)

```
import pandas as pd
import numpy as np
from bs4 import BeautifulSoup
import re
import nltk
```

```
sms_data = pd.read_csv('SMSSpamCollection', header=None, sep='\t')
```

W bazie są 2 kolumny oznaczane jako Label i SMS

Czyszczenie tekstu

Ponieważ podejście do klasyfikacji oparte o model Bayesa polega na wyznaczeniu prawdopodobieństwa przynależności danej wiadomości do spamu, pod warunkiem, że znane są prawdopodobieństwa warunkowe poszczególnych wyrazów. W związku z tym wyrazy powinny zostać zunifikowane (zredukowane do korpusu, zamienione na małe litery), ponadto powinny zostać usunięte znaki interpunkcyjne i inne wartości, które nie są wyrazami.

```
def prep(string):  
  
    # Remove HTML tags.  
    string = BeautifulSoup(string, 'html.parser').get_text()  
  
    # Remove non-letters  
    string = re.sub("[^a-zA-Z]", " ", string)  
  
    # Lower case  
    string = string.lower()  
  
    # Tokenize to each word.  
    token = nltk.word_tokenize(string)  
  
    # Stemming  
    string = [nltk.stem.SnowballStemmer('english').stem(w) for w  
    # Join the words back into one string separated by space, ar  
    return string
```

Zastosowanie powyższej funkcji do jednego wiersza może mieć postać:

```
sms_data['SMS'].iloc[:1].apply(prepare).iloc[0]
```

Stwórz nową kolumnę i zapisz do niej przetransformowaną zawartość wiadomości tekstowych:

```
sms_data['clean_sms'] = ....
```

Algorytm Bayesa

Na początku dokonajmy podziału na część która zostanie wykorzystana do stworzenia modelu (zbiór uczący) i zbiór testowy:

```
train_data = sms_data.sample(frac=0.8, random_state=1).reset_index()
test_data = sms_data.drop(train_data.index).reset_index(drop=True)
train_data = train_data.reset_index(drop=True)
```

Prawdopodobieństwo przynależności wiadomości do zbioru spamu i hamu, oraz liczbę wyrazów odpowiednio w zbiorze spamu i hamu można wyznaczyć jako:

```
Pspam = train_data['Label'].value_counts()['spam'] / train_data.
Pham = train_data['Label'].value_counts()['ham'] / train_data.s
Nvoc = len(train_data.columns) - 3 #całkowita liczba unikalnych
Nspam = train_data.loc[train_data['Label'] == 'spam', 'clean_sms
Nham = train_data.loc[train_data['Label'] == 'ham', 'clean_sms']
```

Stworzenie słownika unikalnych wyrazów, można zrealizować za pomocą drzewa binarnego (set):

```
vocabulary = list(set(train_data['clean_sms'].sum()))
```

zmienna vocabulary opisuje przestrzeń wektora cech, gdzie każdą współrzędną jest osobny wyraz. W takiej przestrzeni dla każdego wpisu wyznaczany jest wektor cech word_count_per_sms , a następnie scalany z zbiorem danych:

```
word_counts_per_sms = pd.DataFrame([
    [row[1].count(word) for word in vocabulary]
    for _, row in train_data.iterrows()], columns=vocabulary)
train_data = pd.concat([train_data.reset_index(), word_counts_per
```

Zakładając że train_data jest zbiorem obserwacji definiującym model Bayesa prawdopodobieństwo warunkowe przynależności danego elementu do zbioru spamu opisuje funkcja:

$$P(w_i \mid \text{Spam}) = \frac{N_{w_i \mid \text{Spam}} + \alpha}{N_{\text{Spam}} + \alpha \cdot N_{\text{Vocabulary}}}$$

gdzie $N_{w|spam}$ oznacza liczbę wystąpień danego wyrazu we wiadomościach typu spam, N_{spam} oznacza całkowitą liczbę wyrazów we wiadomościach typu spam. Alpha jest współczynnikiem, który ma znaczenie gdy dany wyraz nie występuje w modelu, wtedy, przyjmowane jest, że prawdopodobieństwo wystąpienia dla spamu i hamu są równe i wynoszą $1/N_{vocabulary}$

Implementacja funkcji może mieć formę:

```
def p_w_spam(word, alpha=1):
    if word in train_data.columns[4:]:
        return (train_data.loc[train_data['Label'] == 'spam', wc
    else:
        return 1
```

Napisz funkcję `p_w_ham` wyznaczającą prawdopodobieństwo wystąpienia wyrazu pod warunkiem, że sms należy do zbioru ham.

```
def p_w_ham(word):
    # do zdefiniowania
```

prawdopodobieństwo warunkowe tego że dana wiadomość jest spamem (Posterior probability) opisane jest twierdzeniem Bayesa:

$$P(\text{Spam} \mid w_1, w_2, \dots, w_n) \propto P(\text{Spam}) \cdot \prod_{i=1}^n P(w_i \mid \text{Spam})$$

Bazując na wzorze oraz na szkielecie napisz funkcję, która dokonuje klasyfikacji wiadomości:

```
def classify(message):
    p_spam_given_message = Pspam
    p_ham_given_message = Pham
    for word in message:
        p_spam_given_message = ... # Do zaimplementowania
        p_ham_given_message = ... # Do zaimplementowania
    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_ham_given_message < p_spam_given_message:
        return 'spam'
    else:
        return 'unknown'
```

Dokonaj klasyfikacji wiadomości:

```
test_data['predicted'] = test_data['clean_sms'].apply(classify)
```

Skuteczność klasyfikacji możesz sprawdzić zliczając ilość prawidłowych klasyfikacji.

```
correct = (test_data['predicted'] == test_data['Label']).sum() /
```

Pytania:

1. Jaka jest dokładność klasyfikatora
2. Które wyrazy mają (top 10) najwyższe prawdopodobieństwo tego że wchodzi w skład wiadomości typu spam
3. Które wiadomości należące do spamu (top 3) mają najwyższe prawdopodobieństwo że są spamem?

Autorzy: *Piotr Kaczmarek i Jakub Tomczyński*