Building a Spam Filter with Naive Bayes

June 12, 2020

Simon Piper

1 Overview

The aim of this project is to build a spam filter for SMS messages by employing a multinomial Naive Bayes algorithmn, in order to be able to correctly classify SMS messages as either 'spam' or 'ham' (non-spam).

This analysis uses a data set of 5572 SMS messages from the UCI Machine Learning Repository.

[1]:
```
import pandas as pd
import numpy as np

link = "https://raw.githubusercontent.com/PiperS52/
↪Building-a-Spam-Filter-with-Naive-Bayes/master/SMSSpamCollection"
smsspam = pd.read_csv(link, sep='\t', header=None, names=['Label', 'SMS'])
```

[2]:
```
print(smsspam.columns)
print(smsspam.shape)
smsspam.head(5)
```

```
Index(['Label', 'SMS'], dtype='object')
(5572, 2)
```

[2]:
```
  Label                                                SMS
0   ham  Go until jurong point, crazy.. Available only …
1   ham                      Ok lar… Joking wif u oni…
2  spam  Free entry in 2 a wkly comp to win FA Cup fina…
3   ham  U dun say so early hor… U c already then say…
4   ham  Nah I don't think he goes to usf, he lives aro…
```

[3]:
```
smsspam['Label'].value_counts(normalize=True) * 100
```

[3]:
```
ham     86.593683
spam    13.406317
Name: Label, dtype: float64
```

Approximately 86.6% of the data set is classified as 'ham' and 13.4% as 'spam'.

2 Creating the Training and Test Sets

The data set is now split with 80% used as a training set and 20% as a test set.

[4]:
```python
#randomise the data set
random = smsspam.sample(frac=1, random_state=1)

#create an index for the split
train_test_index = round(len(random) * 0.8)

#train/test split
train_set = random[:train_test_index].reset_index(drop=True)
test_set = random[train_test_index:].reset_index(drop=True)

print(train_set.shape)
print(test_set.shape)
```

```
(4458, 2)
(1114, 2)
```

[5]:
```python
train_set['Label'].value_counts(normalize=True) * 100
```

[5]:
```
ham     86.54105
spam    13.45895
Name: Label, dtype: float64
```

[6]:
```python
test_set['Label'].value_counts(normalize=True) * 100
```

[6]:
```
ham     86.804309
spam    13.195691
Name: Label, dtype: float64
```

There is the same proportion of 'ham' and 'spam' messages in both the training and test sets.

3 Data Cleaning

The data is then formatted, removing any punctuation and bringing to lower case for analysis, whereby the data set will then be transformed in order that each unique word in the vocabulary represents a different column:

[7]:
```python
train_set['SMS'] = train_set['SMS'].str.replace('\W', ' ')
train_set['SMS'] = train_set['SMS'].str.lower()
```

[8]:
```python
train_set.head(10)
```

[8]:
```
  Label                                                   SMS
0   ham                           yep  by the pretty sculpture
1   ham       yes  princess  are you going to make me moan
```

```
2   ham                          welp apparently he retired
3   ham                                              havent
4   ham  i forgot 2 ask ü all smth   there s a card on …
5   ham  ok i thk i got it  then u wan me 2 come now or…
6   ham  i want kfc its tuesday  only buy 2 meals only …
7   ham                          no dear i was sleeping   p
8   ham                           ok pa  nothing problem
9   ham                     ill be there on   lt   gt   ok
```

[9]:

```
train_set['SMS'] = train_set['SMS'].str.split()
train_set.head(5)
```

[9]:

```
  Label                                            SMS
0   ham                 [yep, by, the, pretty, sculpture]
1   ham  [yes, princess, are, you, going, to, make, me,…
2   ham                   [welp, apparently, he, retired]
3   ham                                          [havent]
4   ham  [i, forgot, 2, ask, ü, all, smth, there, s, a,…
```

A vocabulary list is then created containing all the unique words in the training set:

[10]:

```
vocab = []

for sms in train_set['SMS']:
    for word in sms:
        vocab.append(word)

vocab = set(vocab)
vocab = list(vocab)
print(len(vocab))
```

```
7783
```

There are 7783 unique words in the vocabulary.

[11]:

```
word_counts_per_sms = {unique_word: [0] * len(train_set['SMS']) for unique_word␣
 ↪in vocab}

for index, sms in enumerate(train_set['SMS']):
    for word in sms:
        word_counts_per_sms[word][index] += 1
```

[12]:

```
word_counts = pd.DataFrame(word_counts_per_sms)
```

[13]:

```
train_clean = pd.concat([train_set, word_counts], axis='columns')
train_clean.head(5)
```

[13]:
```
  Label                                                SMS  stock  \
0   ham                  [yep, by, the, pretty, sculpture]      0
1   ham  [yes, princess, are, you, going, to, make, me,…      0
2   ham                     [welp, apparently, he, retired]      0
3   ham                                           [havent]      0
4   ham  [i, forgot, 2, ask, ü, all, smth, there, s, a,…      0

   help08714742804  throwing  entry  box139  fav  pan  fifth  …  tmorrow  \
0                0         0      0       0    0    0      0  …        0
1                0         0      0       0    0    0      0  …        0
2                0         0      0       0    0    0      0  …        0
3                0         0      0       0    0    0      0  …        0
4                0         0      0       0    0    0      0  …        0

   batch  nutter  62220cncl  jess  eyes  doubletxt  dept  costs  ts
0      0       0          0     0     0          0     0      0   0
1      0       0          0     0     0          0     0      0   0
2      0       0          0     0     0          0     0      0   0
3      0       0          0     0     0          0     0      0   0
4      0       0          0     0     0          0     0      0   0

[5 rows x 7785 columns]
```

4 Calculating Constants

The Naive Bayes algorithmn will classify any new message as 'spam' or 'ham' according to the result of the following two equations:

$$P(Spam|w_1, w_2..., w_n) \; \alpha \; P(Spam) \cdot \prod_{i=1}^{n} P(w_i|Spam) \tag{1}$$

$$P(Ham|w_1, w_2..., w_n) \; \alpha \; P(Ham) \cdot \prod_{i=1}^{n} P(w_i|Ham) \tag{2}$$

where $P(w_i|Spam)$ and $P(w_i|Ham)$ can be defined as:

$$P(w_i|Spam) = \frac{N_{w_i|Spam} + \alpha}{N_{Spam} + \alpha \cdot N_{Vocabulary}} \tag{3}$$

$$P(w_i|Ham) = \frac{N_{w_i|Ham} + \alpha}{N_{Ham} + \alpha \cdot N_{Vocabulary}} \tag{4}$$

To begin with, the following can be calculated:

- $P(Spam)$ and $P(Ham)$
- N_{Spam}, N_{Ham}, $N_{Vocabulary}$

Where Laplace smoothing is used with $\alpha = 1$

[14]:
```
#P(Spam)
p_spam = (train_clean['Label'] == 'spam').sum() / len(train_clean)
#P(Ham)
p_ham = (train_clean['Label'] == 'ham').sum() / len(train_clean)
```

[15]:
```
print(p_spam)
print(p_ham)
```

```
0.13458950201884254
0.8654104979811574
```

[16]:
```
n_words_per_spam_msg = train_clean[train_clean['Label'] == 'spam']['SMS'].
 ↪apply(len)
#N(Spam)
n_spam = n_words_per_spam_msg.sum()

n_words_per_ham_msg = train_clean[train_clean['Label'] == 'ham']['SMS'].
 ↪apply(len)
#N(Ham)
n_ham = n_words_per_ham_msg.sum()

#N(Vocab)
n_vocab = len(vocab)

alpha = 1
```

5 Calculating Parameters

The parameters $P(w_i|Spam)$ and $P(w_i|Ham)$ can then be calculated, with each being a conditional probability for each word in the vocabulary:

[17]:
```
#isolating spam and ham messages
spam_messages = train_clean[train_clean['Label'] == 'spam']
ham_messages = train_clean[train_clean['Label'] == 'ham']

#Initiate parameters
parameters_spam = {unique_word:0 for unique_word in vocab}
parameters_ham = {unique_word:0 for unique_word in vocab}

#Calculate parameters
for word in vocab:
    n_word_given_spam = spam_messages[word].sum()
    p_word_given_spam = (n_word_given_spam + alpha) / (n_spam + alpha*n_vocab)
    parameters_spam[word] = p_word_given_spam
```

```python
    n_word_given_ham = ham_messages[word].sum()
    p_word_given_ham = (n_word_given_ham + alpha) / (n_ham + alpha*n_vocab)
    parameters_ham[word] = p_word_given_ham
```

6 Classifying a New Message

The spam filter can now be created which can be thought of as a function which:

- Takes in as input a new message $(w_1, w_2 ..., w_n)$
- Calculates $P(Spam|w_1, w_2 ..., w_n)$ and $P(Ham|w_1, w_2 ..., w_n)$
- Compares the values of $P(Spam|w_1, w_2 ..., w_n)$ and $P(Ham|w_1, w_2 ..., w_n)$, and:
  - If $P(Ham|w_1, w_2 ..., w_n) > P(Spam|w_1, w_2 ..., w_n)$ then the message is classified as 'ham'
  - If $P(Ham|w_1, w_2 ..., w_n) < P(Spam|w_1, w_2 ..., w_n)$ then the message is classified as 'spam'
  - If $P(Ham|w_1, w_2 ..., w_n) = P(Spam|w_1, w_2 ..., w_n)$ then the algorithmn may require human help

[18]:
```python
import re

def classify(message):

    message = re.sub('\W', ' ', message)
    message = message.lower()
    message = message.split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)

    if p_ham_given_message > p_spam_given_message:
        print('Label: Ham')
    elif p_ham_given_message < p_spam_given_message:
        print('Label: Spam')
    else:
        print('Equal proabilities, have a human classify this!')
```

[19]:
```python
classify('WINNER!! This is the secret code to unlock the money: C3421.')
```

```
P(Spam|message): 1.3481290211300841e-25
```

```
P(Ham|message): 1.9368049028589875e-27
Label: Spam
```

[20]:
```
classify('Sounds good, Tom, then see u there')
```

```
P(Spam|message): 2.4372375665888117e-25
P(Ham|message): 3.687530435009238e-21
Label: Ham
```

7 Measuring the Accuracy of the Spam Filter

A function is defined which returns classification labels, in order that the spam filter can be assessed on the test set:

[21]:
```
def classify_test_set(message):

    message = re.sub('\W', ' ', message)
    message = message.lower()
    message = message.split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'


test_set['predicted'] = test_set['SMS'].apply(classify_test_set)
test_set.head()
```

[21]:

	Label	SMS	predicted
0	ham	Later i guess. I needa do mcat study too.	ham
1	ham	But i haf enuff space got like 4 mb…	ham
2	spam	Had your mobile 10 mths? Update to latest Oran…	spam
3	ham	All sounds good. Fingers . Makes it difficult …	ham
4	ham	All done, all handed in. Don't know if mega sh…	ham

The number of correct predictions against the test set and the accuracy of the spam filter can be measured:

[22]:
```
correct = 0
total = len(test_set)

for row in test_set.iterrows():
    row = row[1]
    if row['Label'] == row['predicted']:
        correct += 1
```

[23]:
```
print('Correct', correct)
print('Total', total)
print('Accuracy', correct/total)
```

```
Correct 1100
Total 1114
Accuracy 0.9874326750448833
```

8 Conclusion

The spam filter has an impressive 98.74% accuracy at detecting spam SMS messages, after evaluating 1114 messages (1100 correctly) in the test set using a Naive Bayes algorithmn.