SRH University Heidelberg

Professional Technical Report

TASK – 2

SMS Spam Detection Based on Character N-gram Model

Author:	Abhijith Pavithran
Matriculation Number:	11018118
Email Address:	11018118@stud.hochschule-heidelberg.de
Supervisor:	Prof. Dr. Milan Gnjatovic
Project Submission Date:	16-11-2022

Table of Contents

Table of Figures

ABSTRACT

In this project, program for the SMS Spam Detection based on character n-gram models approach is explained. In order to implement the program, jupyter notebook is used. For training and testing the model the program used UCI Machine Learning Repository SMS Spam Collection dataset. In order to use the dataset for n-gram model pre-processing is performed to clean the dataset. The dataset is then divided into training and test sets by taking 80% of dataset as training dataset and remaining as test dataset. The training procedure is performed by calculating the probability of all bigrams in the training dataset. The testing procedure then determines the probability of all SMS messages in the test dataset. The program is found to produce good results for bigram models.

INTRODUCTION

A Machine Learning system learns from historical data, builds the prediction models, and whenever it receives new data, predicts the output for it**.** The accuracy of predicted output depends upon the amount of data, as the huge amount of data helps to build a better model which predicts the output more accurately.

In this project, the goal is to apply bigram model to SMS spam classification problem. We use a database of 5574 text messages from UCI Machine Learning repository. It is a large dataset consisting of a collection of both spam and normal (ham) messages. In order to use the dataset for n-gram model pre-processing is performed to clean the dataset. The dataset is then divided into training and test sets by taking 80% of dataset as training dataset and remaining as test dataset. The training procedure is performed by calculating the probability of all bigrams in the training dataset. The testing procedure then determines the probability of all SMS messages in the test dataset.

N-GRAM MODEL

An N-gram model is a probabilistic language model which predicts the occurrence of a word in a sequence based on the occurrence of its N - 1 previous words. N-gram models express the linguistic intuition that the order of

words in a sequence is not random. They are largely used in applications such as natural language processing, speech recognition etc. The N-gram model not only gives the probability of occurrence of a word preceding a sequence of words, but also gives the probability of a given set of words.

For the character bigram, the model is determined by finding conditional probabilities of each character bigram present in the training dataset for both ham and spam classes. The probability of occurrence of character c_m immediately after character c_{m-1} is given by the Equation 1, where $C(c_{m-1}c_m)$ is the number of occurrences of character bigram $c_{m-1}c_m$ and $C(c_{m-1}\sqcup)$ is the number of occurrences of bigram in which c_{m-1} is the first word, and second word is arbitrary.

$$P(c_m|c_{m-1}) = \frac{C(c_{m-1}c_m)}{C(c_{m-1}\sqcup)} \tag{1}$$

From the Equation 1 it is clear that the model needs to find the occurrence of each character of the character vocabulary in the training set for both ham and spam. From Equation 1 it can be also noted that for the bigram model the character bigram count needs to be also determined.

An important factor in determining bigram count is that some of the bigrams might never appear in the training dataset. In such situations it is required to do smoothing of the calculated probability. In this project Add-k smoothing is used, where 0 < k < 1. Hence, for bigram model the probability calculation can be rewritten as shown in Equation 2, where V is the character vocabulary. For this project the k value used is 0.5.

$$P(c_m|c_{m-1}) = \frac{C(c_{m-1}c_m) + k}{C(c_{m-1}) + k|V|} \tag{2}$$

Testing the model created during the training procedure is carried out using the test dataset identified during the data preprocessing section. The test dataset consists of around 1114 SMS messages. Like the training dataset the test dataset contains data [0] column the class of the SMS message (ham or spam) and column data [1] contains the actual SMS message. For testing the model, each

test image data excluding the first column value is read row by row and passed to the predict function.

The probability of a character sequence c1, c2, ..., cm is given by the chain rule of probability as given in Equation 4, where < s > and < /s > are special symbols appended at the beginning and end of the sequence .

$$\begin{aligned} P(< s > w_1 w_2 ... w_m < /s >) = & P(w_1 | < s >) \\ & P(w_2 | < s > w_1) \\ & P(w_3 | < s > w_1 w_2) \\ & ... \\ & P(w_m | < s > w_1 w_2 ... w_{m1}) \\ & P(< /s > | < s > w_1 w_2 ... w_{m1} w_m) \end{aligned} \quad (4)$$

By applying the Markovian assumption for bigram, the above equation can be written as shown in Equation 5.

$$P(< s > w_1 w_2 ... w_m < /s >) \approx P(w_1 | < s >) P(w_2 | w_1) P(w_3 | w_2) ... P(w_m | w_{m-1}) P(< /s > | w_m) \quad (5)$$

Given the bigram probabilities estimated during the training procedure for ham and spam, it should be possible to use Equation 5 to determine probability of a test SMS message. A test SMS message is assigned to one of the classes (ham or spam) if the calculated probability is higher compared to other class.

A problem with multiplication of a sequence of probabilities is that it can lead to arithmetic underflow. In order to eliminate this issue, the log probability can be used. Equation 4 shows the log probability representation of the above equation.

$$\begin{aligned} log\ P(< s > w_1 w_2 ... w_m < /s >) \approx & log\ P(w_1 | < s >) + log\ P(w_2 | w_1) + log\ P(w_3 | w_2) + ... \\ & ... + log\ P(w_m | w_{m-1}) + log\ P(< /s > | w_m) \end{aligned} \quad (6)$$

FLOW CHART

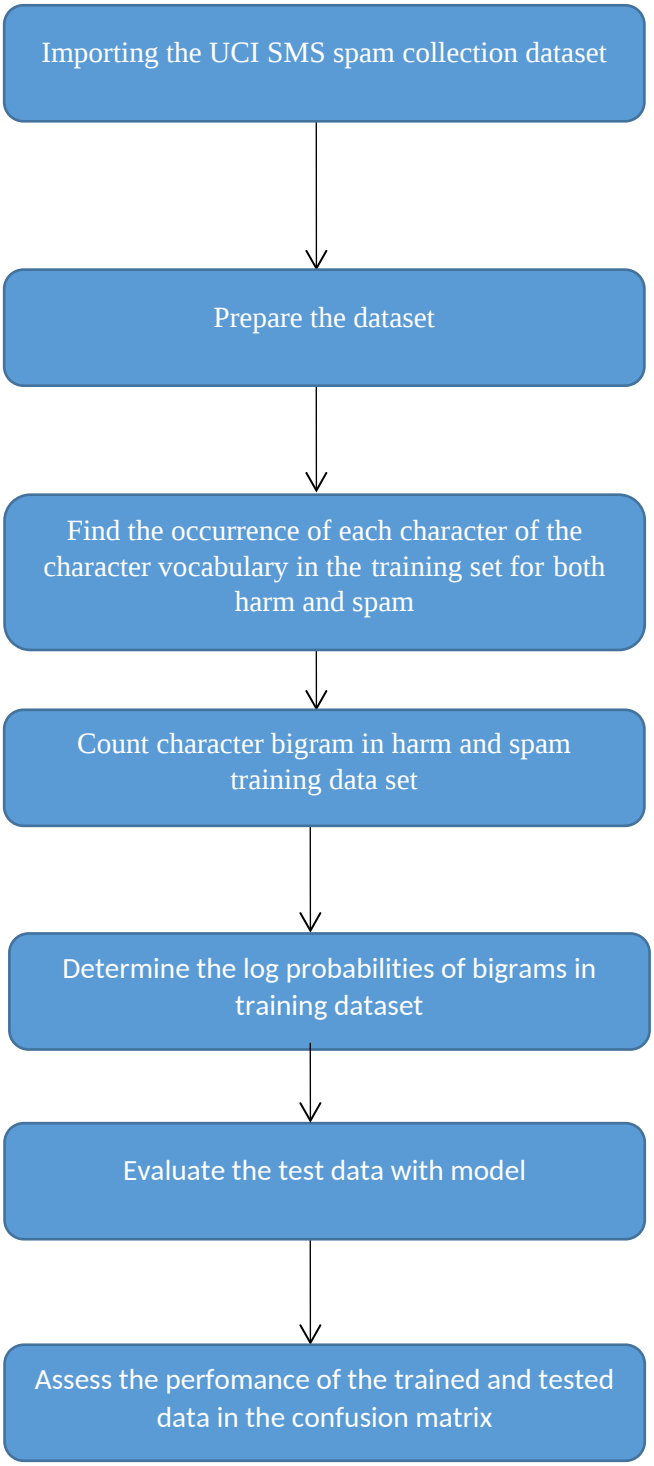

Figure 1: Flow chart

RESULT

In [23]:
```
data = pd.read_csv("./SMSSpamCollection.csv", sep='\t', header=None)
data
```

Out[23]:

	0	1
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...
...	...	...
5567	spam	This is the 2nd time we have tried 2 contact u...
5568	ham	Will ü b going to esplanade fr home?
5569	ham	Pity, * was in mood for that. So...any other s...
5570	ham	The guy did some bitching but I acted like i'd...
5571	ham	Rofl. Its true to its name

5572 rows × 2 columns

In [24]:
```
data.describe()
```

Out[24]:

	0	1
count	5572	5572
unique	2	5169

Figure 2 : Importing UCI SMS spam collection dataset using Pandas library.

In [25]:
```
Vocabulary_character = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', '
                        'v', 'w', 'x', 'y', 'z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', ' ', '/', '?', '
```

In [26]:
```
def data_preprocessing(sms):
    # convert to lowercase.
    sms = sms.lower()
    # replace all unknown character to 'γ'
    sms = "".join([ c if c in Vocabulary_character else 'γ' for c in sms ])
    # append start ('α') and end ('β') symbols
    sms = 'α' + sms + 'β'
    return sms
```

In [28]:
```
total_num_sms = len(data[0])
```

In [29]:
```
for i in range(total_num_sms):
    data[1][i] = data_preprocessing(data[1][i])

# determine min and max indices of train data.
train_min_index = 0
train_max_index = int((total_num_sms * 80) / 100)

# determine min and max indices of test data.
test_min_index = train_max_index + 1
test_max_index = total_num_sms
```

Figure 3 : Preparing dataset for the bigram model.

In [31]:
```
alphabet_count_spam = dict()
alphabet_count_ham = dict()
```

In [38]:
```
# count Vocabulary characters in ham and spam training dataset.
def count_trainingset_alphabet_spam_ham():
    # first set items in charecter vocabulary to 0.
    for char in Vocabulary_character:
        alphabet_count_spam[char] = 0
        alphabet_count_ham[char] = 0

    # count each item of charecter vocabulary in training data.
    for i in range(train_min_index, train_max_index):
        sms = data[1][i]
        if data[0][i] == "spam":
            for c in sms:
                alphabet_count_spam[c] = alphabet_count_spam[c] + 1
        else:
            for c in sms:
                alphabet_count_ham[c] = alphabet_count_ham[c] + 1
```

Figure 4 : Counting the number of alphabet characters in the training dataset.

In [44]:
```
bigram_count_spam = dict()
bigram_count_ham = dict()
```

In [48]:
```
# count character bigram in ham and spam training dataset.
def count_trainingset_bigram_spam_ham():
    # first set bigram count to 0.
    for first_char in Vocabulary_character:
        for second_char in Vocabulary_character:
            bigram = first_char + second_char
            bigram_count_spam[bigram] = 0
            bigram_count_ham[bigram] = 0

    # count each bigram in training data.
    for i in range(train_min_index, train_max_index):
        sms = data[1][i]
        if data[0][i] == "spam":
            for char in range(len(sms) - 1):
                bigram = sms[char] + sms[char + 1]
                bigram_count_spam[bigram] = bigram_count_spam[bigram] + 1
        else:
            for char in range(len(sms) - 1):
                bigram = sms[char] + sms[char + 1]
                bigram_count_ham[bigram] = bigram_count_ham[bigram] + 1
```

In [49]:
```
count_trainingset_bigram_spam_ham()
```

Figure 5 : Calculating the number of character bigram in the training dataset.

In [55]:
```
# predict the test SMS message for ham or spam.
def predict(x):
    spam_predict = log_probability(x, "spam")
    ham_predict = log_probability(x, "ham")

    if spam_predict >= ham_predict:
        return "spam"
    else:
        return "ham"
```

In [56]:
```
# predict test SMS message in each row in the test dataset.
test_data_count = test_max_index - test_min_index
predicted_classes = np.empty((test_data_count), dtype="<U10")
for n in range(test_min_index, test_max_index):
    index = n - test_min_index
    predicted_classes[index] = predict(data[1][n])

print(test_data_count)
print("predicted classes of test SMS messages")
print(predicted_classes)
predicted_classes[1]
```

```
1114
predicted classes of test SMS messages
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

Figure 6 : Determine the probabilities of each bigram.

In [83]:
```
actual_classes = np.array(data[0])[test_min_index:test_max_index]
print("actual classes of test SMS messages")
print(actual_classes)
actual_classes[0]
```

```
actual classes of test SMS messages
['ham' 'ham' 'spam' ... 'ham' 'ham' 'ham']
```

Out[83]:
```
'ham'
```

In [84]:
```
# determine the confusion matrix.
def confusion_matrix(predicted_classes, actual_classes):
    cm = np.zeros((num_classes, num_classes), dtype=np.int32)

    for i in range(len(predicted_classes)):
        if predicted_classes[i] == "spam":
            predicted_class = 0
        else:
            predicted_class = 1

        if actual_classes[i] == "spam":
            actual_class = 0
        else:
            actual_class = 1

        cm[predicted_class][actual_class] = cm[predicted_class][actual_class] + 1

    return cm
```

Figure 7 : Predicting test SMS message for ham or spam

```
In [85]: #making confusion matrix using seaborn library
         plt.figure(figsize=(10,4))
         Confusion_matrix = confusion_matrix(predicted_classes, actual_classes)
         ax = sns.heatmap(Confusion_matrix, annot=True, fmt = "d")
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
         print(confusion_matrix)

         <function confusion_matrix at 0x7f7537189d30>
```

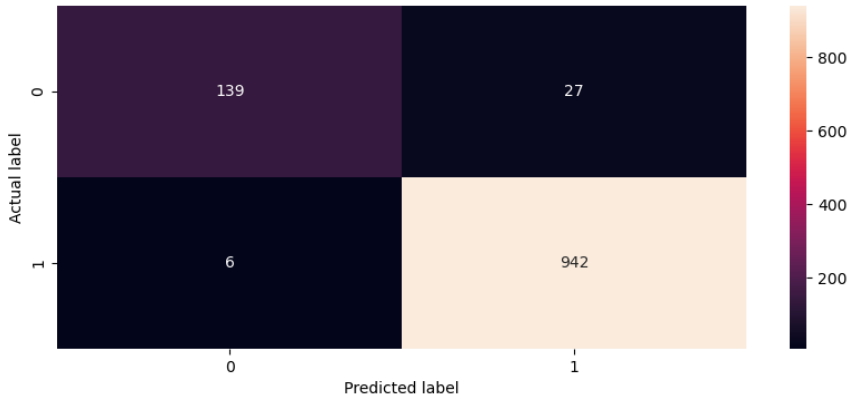

Figure 8 : Formation of confusion matrix.

```
In [17]: True_positive = 942
         True_negative = 139
         False_positive = 27
         False_negative = 6

In [18]: recall = True_positive/(True_positive + False_negative)
         recall

Out[18]: 0.9936708860759493

In [19]: Precision = True_positive/(True_positive + False_positive)
         Precision

Out[19]: 0.9721362229102167

In [20]: F1_score = (Precision * recall / (Precision + recall)) * 2
         F1_score

Out[20]: 0.9827856025039122
```

Figure 9 : Calculation of recall, precision, and F1 score from the confusion matrix.

```
In [89]: accuracy = np.diag(Confusion_matrix).sum()/Confusion_matrix.sum().sum()
         accuracy
         # overoll accuracy from confusion matrix

Out[89]: 0.9703770197486535

In [95]: # predict spam or ham for following sms:
         spam_msg = "This is the 2nd time we have tried 2 contact u "
         ham_msg = "Will ü b going to esplanade fr home?"

In [96]:
         # predict if spam message is predicted to be spam
         pre_processd_sms = data_preprocessing(spam_msg)
         prediction = predict(pre_processd_sms)
         print("message: \"", spam_msg, "\" is : ", prediction)

         message: " This is the 2nd time we have tried 2 contact u  " is :  ham

In [97]: # predict if ham message is predicted to be ham
         pre_processd_sms = data_preprocessing(ham_msg)
         prediction = predict(pre_processd_sms)
         print("message: \"", ham_msg, "\" is : ", prediction)

         message: " Will ü b going to esplanade fr home? " is :  ham

In [ ]:
```

Figure 10 : Calculation of accuracy and the model is tested against a randomly taken spam and a ham message.

CONCLUSION

In this project, a program for the SMS Spam Detection based on character n-gram models approach is successfully realized. The program is written using jupyter notebook. The project contains program to evaluate against bigram. For a given alphabet, the programs achieved 97.03% accuracy for bigram model. The program calculated Precision as 0.9721362229102167 and Recall as 0.9936708860759493. Finally, the F1-score, which is the harmonic mean of the above two values are calculated as 0.9827856025039122. The model is able to predict a randomly taken SMS message as spam or ham successfully.

REFERENCE

- n-gram - Wikipedia
- UCI Machine Learning Repository: SMS Spam Collection Data Set
- N-gram Models (gnjatovic.info)
- https://www.youtube.com/watch?v=9w4HJ0VUy2g
- An Introduction to N-grams: What Are They and Why Do We Need Them? - XRDSXRDS (acm.org)
- https://www.youtube.com/watch?v=YU0m0r9u2sA