# CSE 412: Machine Learning Lab
## Lab Activity 6: Naïve Bayes Classifier

Dr Muhammad Abul Hasan*

April 1, 2024

## 1 Objective

The Naïve Bayes Classifier brings the power to Machine Learning, building a very simple yet powerful classifier. In this lab activity, we will see an overview on how this classifier works, which suitable applications it has, and how to use it in Python and the Scikit-Learn library.

In this lab activity, you will learn:

- What Naïve Bayes classifier is.
- How to implement Naïve Bayes classifier in Python with scikit-learn
- How to analyze the outcome of the classifier

## 2 Theory Behind Bayes' Theorem

If you studied Artificial Intelligency theory, Statistics, or any other courses involving statistics, it is very likely that at some point you studied the following formula:

$$P(H \mid E) = \frac{(P(E \mid H) \times P(H))}{P(E)}$$

where

- $P(H \mid E)$ is the probability of hypothesis $H$ given the event $E$, a posterior probability.
- $P(E \mid H)$ is the probability of event $E$ given that the hypothesis $H$ is true.
- $P(H)$ is the probability of hypothesis $H$ being true (regardless of any related event), or prior probability of $H$.
- $P(E)$ is the probability of the event occurring (regardless of the hypothesis).

This is the Bayes Theorem. At first glance it might be hard to make sense out of it, but it is very intuitive if we explore it through an example:

Let's say that we are interested in knowing whether an e-mail that contains the word *discount* (event) is spam (hypothesis). If we go back to the theorem description, this problem can be formulated as:

*Associate Professor, Deptartment of CSE, GUB ⁘ ✉ muhammad.hasan@cse.green.edu.bd

$$P(class = SPAM \mid contains = \text{“}discount\text{”}) = \frac{(P(contains = \text{“}discount\text{”} \mid class = SPAM) \times P(class = SPAM))}{P(contains = \text{“}discount\text{”})}$$

which in plain English is: The probability of an e-mail containing the word *discount* being spam is equal to the proportion of SPAM emails that contain the word *discount* multiplied by the proportion of e-mails being spam and divided by the proportion of e-mails containing the word *discount*.

Let's dissect this piece by piece:

- $P(class = SPAM \mid contains = \text{“}discount\text{”})$ is the probability of an e-mail being SPAM given that this e-mail contains the word *discount*. This is what we are interested in predicting.
- $P(contains = \text{“}discount\text{”} \mid class = SPAM)$ is the probability of an e-mail containing the word *discount* given that this e-mail has been recognized as SPAM. This is our training data, which represents the correlation between an e-mail being considered SPAM and such e-mail containing the word *discount*.
- $P(class = SPAM)$ is the probability of an e-mail being SPAM (without any prior knowledge of the words it contains). This is simply the proportion of e-mails being SPAM in our entire training set. We multiply by this value because we are interested in knowing how significant is information concerning SPAM e-mails. If this value is low, the significance of any events related to SPAM e-mails will also be low.
- $P(contains = \text{“}discount\text{”})$ is the probability of an e-mail containing the word *discount*. This is simply the proportion of e-mails containing the word *discount* in our entire training set. We divide by this value because the more exclusive the word *discount* is, the more important is the context in which it appears. Thus, if this number is low (the word appears very rarely), it can be a great indicator that in the cases it does appear, it is a relevant feature to analyze.

In summary, the Bayes Theorem allows us to make reasoned deduction of events happening in the real world based on prior knowledge of observations that may imply it. To apply this theorem to any problem, we need to compute the two types of probabilities that appear in the formula.

## 3 Class Probabilities

In the theorem, $P(A)$ represents the probabilities of each event. In the Naïve Bayes Classifier, we can interpret these Class Probabilities as simply the frequency of each instance of the event divided by the total number of instances. For example, in the previous example of spam detection, $P(class = SPAM)$ represents the number of e-mails classified as spam divided by the sum of all instances (this is spam + not spam)

$$P(class = SPAM) = \frac{count(class = SPAM)}{(count(class = notSPAM) + count(class = SPAM))}$$

# 4 Conditional Probabilities

In the theorem, $P(A \mid B)$ represents the conditional probabilities of an event $A$ given another event $B$. In the Naïve Bayes Classifier, these encode the posterior probability of $A$ occurring when $B$ is true.

For the spam example, $P(class = SPAM | contains = \text{“}discount\text{”})$ represents the number of instances in which an e-mail is considered as spam and contains the word *discount*, divided by the total number of e-mails that contain the word *discount*:

$$P(class = SPAM \mid contains = \text{“}discount\text{”}) = \frac{count(class = SPAM, contains = discount)}{count(contains = discount)}$$

# 5 Applications

The application of the Naïve Bayes Classifier has been shown successful in different scenarios. A classical use case is document classification: determining whether a given document corresponds to certain categories. Nonetheless, this technique has its advantages and limitations.

## 5.1 Advantages

Naïve Bayes is a simple and easy to implement algorithm. Because of this, it might outperform more complex models when the amount of data is limited. Naïve Bayes works well with numerical and categorical data. It can also be used to perform regression by using Gaussian Naïve Bayes.

## 5.2 Limitations

Given the construction of the theorem, it does not work well when you are missing certain combination of values in your training data. In other words, if you have no occurrences of a class label and a certain attribute value together (e.g. class=SPAM, contains=”$$$”) then the frequency-based probability estimate will be zero. Given Naïve-Bayes’ conditional independence assumption, when all the probabilities are multiplied you will get zero.

Naïve Bayes works well as long as the categories are kept simple. For instance, it works well for problems involving keywords as features (e.g. spam detection), but it does not work when the relationship between words is important (e.g. sentiment analysis).

# 6 Developing Fundamental Concepts for Preprocessing Data

We need to apply a few preprocessing steps before applying a model to predict a class lebel for an input message. Let’s assume that we have a dataset having three sentences: 1) The canal is open to shipping, 2) This is a ship shipping ship, shipping shipping ships, and 3) The ship locked into the new canal. We would like to seperate each word and cound their frequencies in each sentence. Formally we call it creating a *Bag of Words*. In the following, we create a list containing the sentences.

```
1 documents = ['The canal is open to shipping.',
2               'This is a ship shipping ship, shipping shipping ships.',
3               'The ship locked into the new canal.']
4
```

```
print(documents)
```

**Output:**

```
['The canal is open to shipping.', 'This is a ship shipping ship, shipping
shipping ships.', 'The ship locked into the new canal.']
```

Next, we convert the sentences to all lower case letters.

```
lower_case_documents = []

for i in documents:
    lower_case_documents.append(i.lower())

print (lower_case_documents)
```

**Output:**

```
['the canal is open to shipping.', 'this is a ship shipping ship, shipping
shipping ships.', 'the ship locked into the new canal.']
```

Punction marks do not add any value in predicting class label. So, we remove punctuation from each senence.

```
import string

punctuation_removed_documents = []

for i in lower_case_documents:
    punctuation_removed_documents = ["".join( j for j in i
    if j not in string.punctuation) for i in  lower_case_documents]

print (punctuation_removed_documents)
```

**Output:**

```
['the canal is open to shipping', 'this is a ship shipping ship shipping
 shipping ships', 'the ship locked into the new canal']
```

Next, we break each sentence into words.

```
preprocessed_documents = []

for i in punctuation_removed_documents:
    preprocessed_documents.append(i.split(' ')) #split on space

print (preprocessed_documents)
```

**Output:**

```
[['the', 'canal', 'is', 'open', 'to', 'shipping'], ['this', 'is', 'a',
 'ship', 'shipping', 'ship', 'shipping', 'shipping', 'ships'], ['the',
 'ship', 'locked', 'into', 'the', 'new', 'canal']]
```

Finally, word frequency of each word is counted using `counter` function.

```
from collections import Counter

frequency_list = []

for i in preprocessed_documents:
    frequency_counts = Counter(i)
    frequency_list.append(frequency_counts)

print(frequency_list)
```

**Output:**

```
[Counter({'the': 1, 'canal': 1, 'is': 1, 'open': 1, 'to': 1,'shipping': 1}),
 Counter({'shipping': 3, 'ship': 2,'this': 1, 'is': 1, 'a': 1, 'ships': 1}),
 Counter({'the': 2,'ship': 1, 'locked': 1,'into': 1, 'new': 1, 'canal': 1})]
```

The above output shows the constituting words of each sentence and their frequencies. The above steps are a bit lengthy. Instead of using them, we can do the same using `CountVectorizer` from `sklearn.feature_extraction.text`.

```
from sklearn.feature_extraction.text import CountVectorizer

count_vector = CountVectorizer() #set the variable

count_vector.fit(documents) #fit the function
count_vector.get_feature_names() #get the outputs
```

**Output:**

```
['canal',
 'into',
 'is',
 'locked',
 'new',
 'open',
 'ship',
 'shipping',
```

```
'ships',
'the',
'this',
'to']
```

The above output shows the set of word occuring in the dataset. Now we would like to output the frequencies of each word using the following piece of code.

```
doc_array = count_vector.transform(documents).toarray()
doc_array
```

**Output:**

```
array([[1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1],
       [0, 0, 1, 0, 0, 0, 2, 3, 1, 0, 1, 0],
       [1, 1, 0, 1, 1, 0, 1, 0, 0, 2, 0, 0]])
```

Pandas is a great tool for visualization. We represent the above outputs using Pandas DataFrame which displays the output in a much meaningful way.

```
import pandas as pd
frequency_matrix = pd.DataFrame(doc_array,columns =
count_vector.get_feature_names())

frequency_matrix
```

**Output:**

| | canal | into | is | locked | new | open | ship | shipping | ships | the | this | to |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 0 | 2 | 3 | 1 | 0 | 1 | 0 |
| 2 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 2 | 0 | 0 |

# 7 Implement Naïve Bayes Classifier in Scikit-Learn

We use Python 3 together with Scikit-Learn to build a very simple SPAM detector for SMS messages. We use two libraries which make our coding much easier: `scikit-learn`, `numpy`, and `pandas`.

## 7.1 Loading the Data

The SMS Spam Collection v.1 is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according being ham (legitimate) or spam. The distribution is a total of 4,827 SMS legitimate messages (86.6%) and a total of 747 (13.4%) spam messages.

To load the data, we can use Pandas' Dataframe read_table method. This allows us to define a separator (in this case, a tab) and rename the columns accordingly:

```
import pandas as pd
# Dataset from
# https://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection
df = pd.read_table('SMSSpamCollection',
                   sep='\t',
                   header=None,
                   names=['label', 'message'])

df.head()
```

**Output:**

| | label | message |
|---|---|---|
| 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... |

Once we have our data ready, it is time to do some preprocessing. We will focus on removing useless variance for our task at hand. First, we have to convert the labels from strings to binary values for our classifier. Map applies a function to all the items in an input list or df column.

```
df['label'] = df.label.map({'ham': 0, 'spam': 1})

df.head()
```

**Output:**

| | label | message |
|---|---|---|
| 0 | 0 | Go until jurong point, crazy.. Available only ... |
| 1 | 0 | Ok lar... Joking wif u oni... |
| 2 | 1 | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | 0 | U dun say so early hor... U c already then say... |
| 4 | 0 | Nah I don't think he goes to usf, he lives aro... |

Now that we have performed feature extraction from our data, it is time to build our model. We will start by splitting our data into training and test sets:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df['message'],
                                                    df['label'], random_state=1)

print ("Original dataset contains", df.shape[0], "messages")
print ("Training set contains", X_train.shape[0], "messages")
print ("Testing set contains", X_test.shape[0], "messages")
```

**Output:**

```
Original dataset contains 5572 messages
Training set contains 4179 messages
Testing set contains 1393 messages
```

Next, we transform the data into occurrences, which will be the features that we will feed into our model:

```
from sklearn.feature_extraction.text import CountVectorizer

count_vector = CountVectorizer() #set the variable

train = count_vector.fit_transform(X_train)
test = count_vector.transform(X_test)
```

Our spam classifier will use multinomial Naïve Bayes method from `sklearn.nive_bayes`. This method is well-suited for discrete inputs (like word counts) whereas the Gaussian Naïve Bayes classifier performs better on continuous inputs.

```
from sklearn.naive_bayes import MultinomialNB

#call the method
naive_bayes = MultinomialNB()

#train the classifier on the training set
naive_bayes.fit(train, y_train)
```

**Output:**

```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Once we have put together our classifier, we can evaluate its performance in the testing set:

```
import numpy as np

#predic using the model on the testing set
predictions = naive_bayes.predict(test)

print(np.mean(predictions == y_test))
```

**Output:**

```
0.9885139985642498
```

Our simple Naïve Bayes Classifier has 98.2% accuracy with this specific test set! But it is not enough by just providing the accuracy, since our dataset is imbalanced when it comes to

the labels (86.6% legitimate in contrast to 13.4% spam). It could happen that our classifier is over-fitting the legitimate class while ignoring the spam class. To solve this uncertainty, let's have a look at the confusion matrix:

```
1 from sklearn.metrics import confusion_matrix
2
3 print(confusion_matrix(y_test, predictions))
```

**Output:**

```
[[1203    5]
 [  11  174]]
```

As we can see, the amount of errors is pretty balanced between legitimate and spam, with 5 legitimate messages classified as spam and 11 spam messages classified as legitimate. Overall, these are very good results for our simple classifier.

# 8 Conclusion

In this lab activity, we have learned both theory and practice of the Naïve Bayes Classifier. We have put together a simple Multimodal Naïve Bayes Classifier that achieves 98.2% accuracy on spam detection for SMS messages.

# 9 Lab Exercises

Please code yourself and write a report based on your findings:

1. Create a dataset containing two different types of news: sports and ploitics. (Collect 100 news, 50 from each class.)
2. Perform Naïve Bayes Classifier on your dataset and report classification results.
3. [Bonus] How would you extend your dataset and code for more than two classes?

My Notes Date:_____