Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)
Department of Computer Science and Engineering (Data Science)

ACL PRACTICALS

Experiment 1

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: K – K1 / D11

Aim: - Implement a Spam classifier using Naïve Bayes classifier

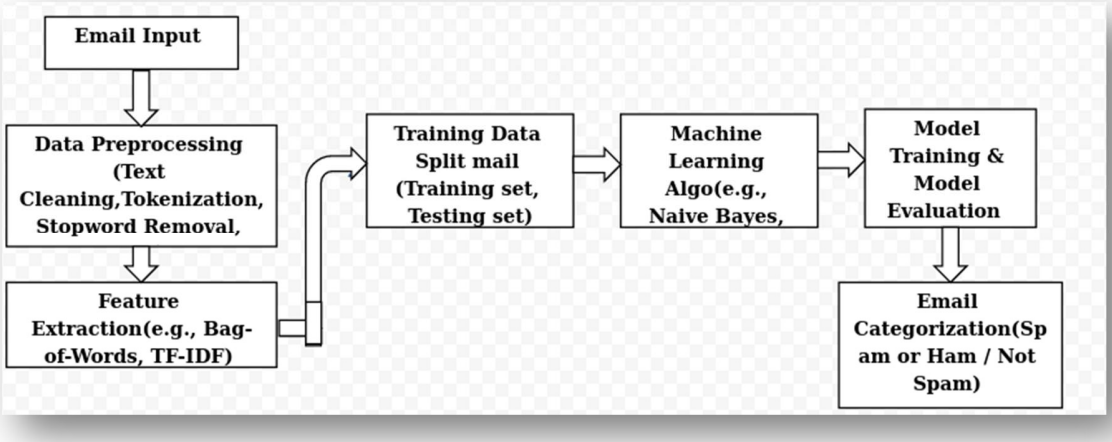
Theory:

Spam Classification

Spam classification, also known as email spam filtering or spam detection, is a classic problem in the field of natural language processing and machine learning. It involves automatically identifying and categorizing incoming emails or messages as either "spam" (unsolicited and often irrelevant or inappropriate) or "ham" (legitimate and desired).

The primary goal of spam classification is to build a model that can accurately distinguish between spam and non-spam messages, thereby reducing the amount of unwanted and potentially harmful content that reaches users' inboxes. This is especially important considering the sheer volume of spam emails generated daily, as well as the potential security risks associated with phishing attempts and malicious content that may be contained in spam messages.

To solve the spam classification problem, machine learning algorithms are employed. Common approaches include the use of traditional algorithms like Naive Bayes, support vector machines (SVMs), decision trees, and more modern techniques like deep learning using neural networks.



Naïve Bayes classifier

The Naive Bayes classifier is a simple, yet effective probabilistic machine learning algorithm commonly used for classification tasks, particularly in natural language processing, spam filtering, sentiment analysis, and more. Despite its simplicity, Naive Bayes often performs surprisingly well and can be computationally efficient, making it a popular choice for certain applications.

The "Naive" in Naive Bayes comes from the assumption that the features (or attributes) used for classification are conditionally independent, given the class label. In other words, the model assumes that each feature contributes independently to the probability of a specific class. This is a simplification, and in reality, features might be correlated, but the assumption makes the algorithm computationally tractable.

How Naive Bayes works:

1. Data and Labels: The Naive Bayes classifier is trained on a labeled dataset containing features and their corresponding class labels. For example, in email spam classification, the features could be the words in the email, and the labels would be "spam" or "ham" (not spam).
2. Prior Probability: The first step in Naive Bayes is to calculate the prior probabilities of each class. The prior probability of a class is the probability of that class occurring without considering any features. For example, if you have 70% spam emails and 30% non-spam emails in your dataset, the prior probabilities would be P(spam) = 0.7 and P(ham) = 0.3.

3. Likelihood Estimation: Next, Naive Bayes calculates the likelihood probabilities for each feature given the class labels. The likelihood probability is the probability of observing a particular feature given the class. For example, P(word='free' | spam) would represent the probability of the word "free" appearing in spam emails.
4. Posterior Probability: Using Bayes' theorem, the Naive Bayes classifier calculates the posterior probability of each class given the observed features. The posterior probability is the probability of a class given the evidence (features). The class with the highest posterior probability is the predicted class for the input.
5. Prediction: The Naive Bayes classifier then makes predictions based on the highest posterior probability. The class with the highest posterior probability for a given set of features is assigned as the predicted class.

Lab Experiments to be Performed in This Session: -

Data Set: - SMS Spam Collection

This dataset contains SMS messages labeled as spam or ham. It is commonly used for spam detection in text messages.

Exercise 1: - Perform Spam Classification With text preprocessing steps.

Step 1: Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Perform Data Preprocessing to convert it into a suitable format for training the Naive Bayes classifier. Common steps include (Removing punctuation and special characters, Tokenization, stop word Removal, converting to lowercase, Stemming or Lemmatization)

Step 3: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 4: Training the Naive Bayes Classifier

Step 5: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 6: Hyperparameter Tuning (Optional)

Step 7: Deployment. Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 2: - Perform Spam Classification Without Using Any text preprocessing steps.

Step 1: - Import Labeled with labels as "spam" or "not spam" (ham).

Step 2: Feature Extraction (bag-of-words model or term frequency-inverse document frequency (TF-IDF) representation.

Step 3: Training the Naive Bayes Classifier

Step 4: Evaluate the performance of the trained Naive Bayes classifier on the test dataset.

Step 5: Hyperparameter Tuning (Optional)

Step 6: Deployment. Once you are satisfied with the model's performance, you can deploy it to classify new, unseen emails or messages as spam or ham.

Exercise 3: - Conclusion

a. Compare The results of Exercise 1 and Exercise 2
b. Show the Comparison using different visualization techniques.

NAME: UMANG KIRIT LODAYA

SAP ID: 60009200032

BATCH: K - K1 / D11

AIM: NAIVE BAYES CLASSIFIER

Importing Libraries And Data

In [ ]:
```
!pip install nltk
```

In [2]:
```
import string
import numpy as np
import pandas as pd
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
```

In [4]:
```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

In [ ]:
```
import nltk
nltk.download('all')

from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
```

In [6]:
```
data = None
with open('/content/drive/MyDrive/Colab Notebooks/ACL/sms_spam_collection/SM
    data = f.readlines()
```

In [7]:
```
len(data)
```

Out[7]:
```
5574
```

In [8]:
```
table = str.maketrans('', '', string.punctuation)
stop_words = set(stopwords.words('english'))
porter = PorterStemmer()
```

In [9]:
```
clean_df = {'SMS': [], 'CLASS': []}
df = {'SMS': [], 'CLASS': []}

for i in range(len(data)):
    data[i] = data[i][:-1]
    c, s = data[i].split('\t')

    tokens = word_tokenize(s)
```

```
    tokens = [w.lower() for w in tokens]

    stripped = [w.translate(table) for w in tokens]

    words = [word for word in stripped if word.isalpha()]
    words = [w for w in words if not w in stop_words]

    words = [porter.stem(word) for word in words]

    clean_df['SMS'].append(' '.join(words))
    clean_df['CLASS'].append(c)

    df['SMS'].append(s)
    df['CLASS'].append(c)

clean_df = pd.DataFrame(clean_df)
df = pd.DataFrame(df)
df.head()
```

Out[9]:

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

In [10]:
```
df.shape
```

Out[10]:
```
(5574, 2)
```

In [11]:
```
C = list(set(list(df['CLASS'])))
```

In [12]:
```
df['C'] = df['CLASS'].apply(lambda x: C.index(x))
clean_df['C'] = clean_df['CLASS'].apply(lambda x: C.index(x))

df.head()
```

Out[12]:

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

Building Vocabulary

In [13]:
```
from nltk.corpus import words
```

In [14]:
```
set_words = set(words.words())
```

In [15]:
```
def build_vocab(DF):
    vocabulary = {}

    for i in range(DF.shape[0]):
        sms = DF.iloc[i, 0].split()
        indx = len(vocabulary)

        for word in sms:
            if not vocabulary.get(word.lower(), None) and word.lower() not :
                vocabulary[word] = indx
                indx += 1

    return vocabulary
```

In [16]:
```
vocabulary = build_vocab(df)
clean_vocabulary = build_vocab(clean_df)
```

In [17]:
```
def build_XY(DF, vocab):
    X = np.zeros((DF.shape[0], len(vocabulary)))
    Y = np.zeros((DF.shape[0]))

    for i in range(DF.shape[0]):
        sms = DF.iloc[i, 0].split()
        for word in sms:
            if word.lower() in vocab:
                X[i, vocab[word.lower()]] += 1
                Y[i] = DF.iloc[i, 2]

    return X, Y
```

In [18]:
```
X, Y = build_XY(df, vocabulary)
clean_X, clean_Y = build_XY(clean_df, clean_vocabulary)
```

In [19]:
```
print(X.shape)
print(Y.shape)
```

```
(5574, 10699)
(5574,)
```

In [20]:
```
print(clean_X.shape)
print(clean_Y.shape)
```

```
(5574, 10699)
(5574,)
```

In [21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
clean_X_train, clean_X_test, clean_Y_train, clean_Y_test = train_test_split
```

Performing Naive Bayes

Without Text Cleaning

In [22]:
```
NB = GaussianNB()
NB = NB.fit(X_train, Y_train)
```

In [23]:
```
Y_pred = NB.predict(X_train)
acc = round(100 * sum(Y_pred == Y_train) / len(Y_pred), 2)
print('TRAIN ACCURACY:', acc)
```

```
TRAIN ACCURACY: 94.8
```

In [24]:
```
Y_pred = NB.predict(X_test)
acc = round(100 * sum(Y_pred == Y_test) / len(Y_pred), 2)
print('TEST ACCURACY:', acc)
```

```
TEST ACCURACY: 80.87
```

With Text Cleaning

In [25]:
```
NB = GaussianNB()
NB = NB.fit(clean_X_train, clean_Y_train)
```

In [26]:
```
Y_pred = NB.predict(clean_X_train)
acc = round(100 * sum(Y_pred == clean_Y_train) / len(Y_pred), 2)
print('TRAIN ACCURACY:', acc)
```

```
TRAIN ACCURACY: 90.23
```

In [27]:
```
Y_pred = NB.predict(clean_X_test)
acc = round(100 * sum(Y_pred == clean_Y_test) / len(Y_pred), 2)
print('TEST ACCURACY:', acc)
```

```
TEST ACCURACY: 80.45
```

ACL PRACTICAL 1 – TASK

Report on Research Papers

Name: Umang Kirit Lodaya **SAP ID: 60009200032**

Batch: K – 1 / D11

"Effects of Preprocessing Methods on Text Classification of Restaurant Reviews"

1. Introduction

The article under consideration, titled "Effects of Preprocessing Methods on Text Classification of Restaurant Reviews," provides a comprehensive analysis of the impact of various preprocessing methods on the precision of text classification, with a particular focus on restaurant reviews. The purpose of this study is to evaluate the classification performance effects of preprocessing techniques such as tokenization, emoticon handling, abbreviation expansion, word correction, stop words elimination, stemming, lemmatization, lowercasing, removal of common words, and n-gram analysis. This report offers a comprehensive evaluation of the paper's content and findings.

2. Objective of the Study

The primary objective of this study is to ascertain how various preprocessing methods affect the classification accuracy of machine learning classifiers when applied to restaurant reviews. The purpose of the study is to determine which preprocessing techniques are most effective at enhancing accuracy and the optimal order for employing these techniques for optimal results.

3. Methodology

The paper employs a well-structured methodology that includes the following essential elements:

- The researchers select a set of preprocessing techniques, including tokenization, emoticon handling, abbreviation expansion, word correction, stop words elimination, stemming, lemmatization, lowercasing, removal of prevalent words, and n-gram analysis.

- Classifier selection: Several machine-learning classifiers, including KNN, DT, RF, LR, SGD, NB, and SVM, are used to evaluate the impact of preprocessing techniques on classification accuracy.

- Dataset: This study employs a dataset of 10,000 restaurant reviews in an effort to reliably predict the star ratings (1 to 5) of these reviews.

4. Preprocessing Methods and Their Effects

The paper meticulously evaluates each preprocessing method, discussing its impact on classification accuracy. Key findings include:

Method	Discussion
Tokenization	Different tokenization techniques yield similar results, indicating no significant difference in classifier performance.
Emoticons and Punctuation	Replacing emoticons and removing punctuation had minimal impact on accuracy, suggesting their insignificance in this context.
Abbreviations and Acronyms	Expanding abbreviations and acronyms did not improve classification accuracy significantly.
Word Correction	The autocorrection of misspelled words was ineffective and did not enhance accuracy
Stopword Elimination	Removing stopwords significantly improved accuracy, particularly with a modified stopwords list.
Stemming and Lemmatization	Minimal impact on accuracy, suggesting they are not critical for this classification task.
Lowercasing	Lowercasing significantly improved accuracy, emphasizing the importance of text case consistency.
Removing Common Words	Improved accuracy by reducing confusion caused by these words.
N-gram Analysis	Combining n-grams (Uni, Bi, Tri) provided the best accuracy compared to individual n-grams.
Preprocessing Order	The order of applying preprocessing methods affected accuracy by up to 10%, highlighting its importance.

5. Conclusion and Future Work

The conclusion of the research paper summarises the main findings and highlights the importance of preprocessing techniques in text classification. It emphasises the significance of selecting preprocessing methods and their order with care to optimise accuracy. In addition, the study outlines future work, such as investigating additional preprocessing strategies and comparing different algorithms to improve the accuracy of predicting fine-grained review ratings.

"A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing"

1. Introduction

The research paper titled "A Survey on Text Pre-Processing & Feature Extraction Techniques in Natural Language Processing" by Ayisha Tabassum and Dr. Rajendra R. Patil explores the fundamental aspects of Natural Language Processing (NLP). The authors delve into preprocessing and feature extraction techniques that play a critical role in preparing textual data for machine interpretation. The paper emphasizes the significance of these techniques in various NLP tasks like text classification, sentiment analysis, and information retrieval.

2. Objective of the Study

The primary objective of the research is to investigate the essential preprocessing techniques and feature extraction methods in NLP. The study aims to shed light on the importance of effective text preprocessing for converting raw text into a format that is meaningful for computers. Furthermore, the paper addresses various commonly used techniques and their applications in NLP.

3. Preprocessing Techniques

The paper outlines key preprocessing techniques essential for preparing raw text for analysis:

Method	Discussion
Sentence Segmentation	Divides text into sentences, aiding in further processing.
Change to Lowercase	Converts text to lowercase to ensure consistency and avoid word duplication.
Tokenization	Splits text into words, aiding in analysis by filtering unwanted words.
Parts-of-Speech Tagging	Identifies parts of speech (noun, verb, etc.) in the text.
Stopword Elimination	Removing stopwords significantly improved accuracy, particularly with a modified stopwords list.
Removal of Punctuations	Eliminates punctuation marks from text
Stemming and Lemmatization	Minimal impact on accuracy, suggesting they are not critical for this classification task.

4. Feature Extraction Techniques

The paper discusses significant feature extraction methods:

Method	Discussion
Named Entity Recognition (NER)	Identifies named entities like person names, organizations, and locations.
Bag-of-Words Model (BoW)	Represents features based on word occurrences, disregarding word order.
TF-IDF	Balances word frequency by considering term importance in a document.

5. Conclusion

In conclusion, the research paper underscores the pivotal role of preprocessing and feature extraction techniques in the field of Natural Language Processing. It emphasizes that effective preprocessing significantly impacts the accuracy of machine learning algorithms applied to textual data. The order and selection of preprocessing steps are crucial, ensuring that the subsequent NLP pipeline operates optimally.