Assignment 1

Problem Statement:

Write a program for pre-processing of a text document such as stop word removal, stemming.

Objective:

To understand the concepts of information retrieval and web mining

Theory:

Text data derived from natural language is unstructured and noisy. Text preprocessing involves transforming text into a clean and consistent format that can then be fed into a model for further analysis and learning.

Text preprocessing techniques may be general so that they are applicable to many types of applications, or they can be specialized for a specific task. For example, the methods for processing scientific documents with equations and other mathematical symbols can be quite different from those for dealing with user comments on social media.

However, some steps, such as sentence segmentation, tokenization, spelling corrections, and stemming, are common to both.

Here's what you need to know about text preprocessing to improve your natural language processing (NLP).

The NLP Preprocessing Pipeline

A natural language processing system for textual data reads, processes, analyzes, and interprets text. As a first step, the system preprocesses the text into a more structured format using several different stages. The output from one stage becomes an input for the next—hence the name "preprocessing pipeline."

An NLP pipeline for document classification might include steps such as sentence segmentation, word tokenization, lowercasing, stemming or lemmatization, stop word removal, and spelling correction. Some or all of these commonly used text preprocessing stages are used in typical NLP systems, although the order can vary depending on the application.

Segmentation

Segmentation involves breaking up text into corresponding sentences. While this may seem like a trivial task, it has a few challenges. For example, in the English language, a period normally indicates the end of a sentence, but many abbreviations, including "Inc.," "Calif.," "Mr.," and "Ms.," and all fractional numbers contain periods and introduce uncertainty unless the end-of-sentence rules accommodate those exceptions.

Tokenization

The tokenization stage involves converting a sentence into a stream of words, also called "tokens." Tokens are the basic building blocks upon which analysis and other methods are built.

Many NLP toolkits allow users to input multiple criteria based on which word boundaries are determined. For example, you can use a whitespace or punctuation to determine if one word has ended and the next one has started. Again, in some instances, these rules might fail. For example, *don't*, *it's*, etc. are words themselves that contain punctuation marks and have to be dealt with separately.

Change Case

Changing the case involves converting all text to lowercase or uppercase so that all word strings follow a consistent format. Lowercasing is the more frequent choice in NLP software.

Spell Correction

Many NLP applications include a step to correct the spelling of all words in the text.

Stop-Words Removal

"Stop words" are frequently occurring words used to construct sentences. In the English language, stop words include *is*, *the*, *are*, *of*, *in*, and *and*. For some NLP applications, such as document categorization, sentiment analysis, and spam filtering, these words are redundant, and so are removed at the preprocessing stage.

Stemming

The term *word stem* is borrowed from linguistics and used to refer to the base or root form of a word. For example, *learn* is a base word for its variants such as *learn, learns, learning,* and *learned.*

Stemming is the process of converting all words to their base form, or stem. Normally, a lookup table is used to find the word and its corresponding stem. Many search engines apply stemming for retrieving documents that match user queries. Stemming is also used at the preprocessing stage for applications such as emotion identification and text classification.

Lemmatization

Lemmatization is a more advanced form of stemming and involves converting all words to their corresponding root form, called "lemma." While stemming reduces all words to their stem via a lookup table, it does not employ any knowledge of the parts of speech or the context of the word. This means stemming can't distinguish which meaning of the word *right* is intended in the sentences "Please turn right at the next light" and "She is always right."

The stemmer would stem *right* to *right* in both sentences; the lemmatizer would treat *right* differently based upon its usage in the two phrases.

A lemmatizer also converts different word forms or inflections to a standard form. For example, it would convert *less*to*little, wrote* to *write, slept* to *sleep,* etc.

A lemmatizer works with more rules of the language and contextual information than does a stemmer. It also relies on a dictionary to look up matching words. Because of that, it requires more processing power and time than a stemmer to generate output. For these reasons, some NLP applications only use a stemmer and not a lemmatizer.

Text Normalization

Text normalization is the preprocessing stage that converts text to a canonical representation. A common application is the processing of social media posts, where input text is shortened or words are spelled in different ways. For example, *hello* might be written as *hellooo* or *something* might appear as *smth*, and different people might choose to write *real time*, *real-time*, or*realtime*. Text normalization cleans the text and ideally replaces all words with their corresponding canonical representation. In the last example, all three forms would be converted to *realtime*. Many text normalization stages also replace emojis in text with a corresponding word. For example, *:-)* is replaced by *happy face*.

Parts of Speech Tagging

One of the more advanced text preprocessing techniques is parts of speech (POS) tagging. This step augments the input text with additional information about the sentence's grammatical structure. Each word is, therefore, inserted into one of the predefined categories such as a noun, verb, adjective, etc. This step is also sometimes referred to as grammatical tagging.

Conclusion:

By using above steps, we have performed pre-processing of a text document such as stop word removal, stemming successfully.

Assignment 2

Problem Statement:
Implement a program for retrieval of documents using inverted files.

Objective:
1. Evaluate and analyse retrieved information
2. To study Indexing, Inverted Files and searching with the help of inverted file

Theory:
An inverted index is an index data structure storing a mapping from content, such as words or numbers, to its locations in a document or a set of documents. In simple words, it is a HashMap like data structure that directs you from a word to a document or a web page.

Creating Inverted Index
We will create a **Word level inverted index** that is it will return the list of lines in which the word is present. We will also create a dictionary in which key values represent the words present in the file and the value of a dictionary will be represented by the list containing line numbers in which t hey are present. To create a file in Jupiter notebook, use magic function:

```
%% write file file.txt
This is the first word.
This is the second text, Hello! How are you?
This is the third, this is it now.
```
This will create a file named file.txt will the following content.

```
# this will open the file
file=open('file.txt', encoding='utf8')
read =file.read()
file.seek(0)
read
 # to obtain the
# number of lines
# in file
line =1
forword inread:
    ifword =='\n':
        line +=1
print("Number of lines in file is: ", line)
 # create a list to
# store each line as
# an element of list
array =[]
fori                              inrange(line):
    array.append(file.readline()) array
```

```
Number of lines in file is: 3
['This is the first word.\n',
'This is the second text, Hello! How are you?\n',
'This is the third, this is it now.']
```

Functions used:

- **Open:** It is used to open the file.
- **read:** This function is used to read the content of the file.
- **seek(0):** It returns the cursor to the beginning of the file.

Remove punctuation:

```
punc ='''!()-[]{};:'"\, <>./?@#$%^&*_~'''
forele inread:
    ifele inpunc:
        read =read.replace(ele, " ")
  read
 # to maintain uniformity
read=read.lower()
read
```

Output:

```
'this is the first word \n
this is the second text hello how are you \n
this is the third this is it now '
```

Tokenize the data as individual words:

Apply linguistic preprocessing by converting each words in the sentences into tokens. Tokenizing the sentences help with creating the terms for the upcoming indexing operation.

```
deftokenize_words(file_contents):
    """
    Tokenizes the file contents.
     Parameters
    -----------
    file_contents : list
        A list of strings containing the contents of the file.
         Returns
    --------
    list
        A list of strings containing the contents of the file tokenized.

    """
    result =[]
     fori inrange(len(file_contents)):
        tokenized =[]
         # print("The row is ", file_contents[i])
         # split the line by spaces
        tokenized =file_contents[i].split()
         result.append(tokenized)
    returnresult
```

Clean data by removing stopwords:

Stop words are those words that have no emotions associated with it and can safely be ignored without sacrificing the meaning of the sentence.

```
fromnltk.tokenize importword_tokenize
importnltk
fromnltk.corpus importstopwords
nltk.download('stopwords')
 fori inrange(1):
    # this will convert
    # the word into tokens
    text_tokens =word_tokenize(read)
 tokens_without_sw =[
    word forword intext_tokens ifnotword instopwords.words()]
 print(tokens_without_sw)
```

Output:

```
['first', 'word', 'second', 'text', 'hello', 'third']
```

Create an **inverted index:**

```
dict={}
 fori inrange(line):
    check =array[i].lower()
    foritem intokens_without_sw:
         ifitem incheck:
            ifitem notindict:
                dict[item] =[]
             ifitem indict:
                dict[item].append(i+1)
 dict
```

Output:

```
{'first': [1],

'word': [1],

'second': [2],

'text': [2],

'hello': [2],

'third': [3]}
```

Conclusion:

By this way, we can perform retrieval of documents using inverted files.

Assignment 3

Problem Statement:
Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using the standard Heart Disease Data Set (You can use Java/Python ML library classes/API.

Objective:
1. Evaluate and analyse retrieved information.
2. To study Bayesian network model.

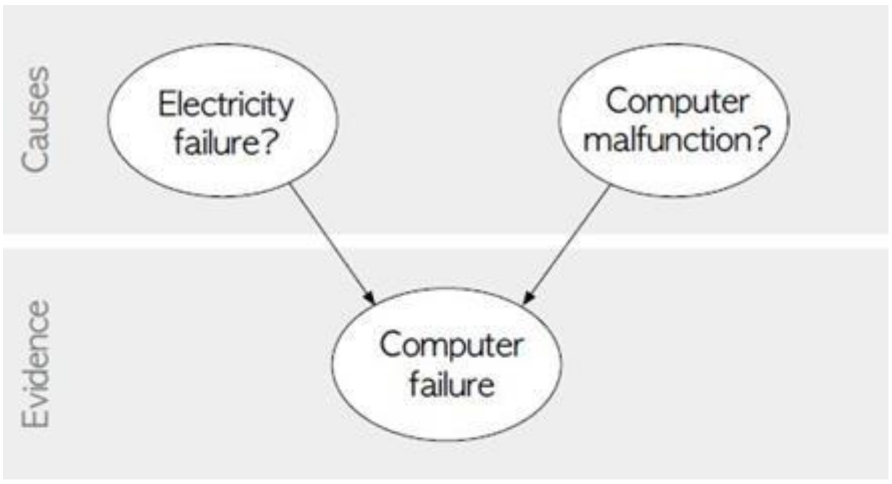
Theory:
A Bayesian network is a directed acyclic graph in which each edge corresponds to a conditional dependency, and each node corresponds to a unique random variable.

Bayesian network consists of two major parts: a directed acyclic graph and a set of conditional probability distributions

- The directed acyclic graph is a set of random variables represented by nodes.
- The conditional probability distribution of a node (random variable) is defined for every possible outcome of the preceding causal node(s).

For illustration, consider the following example. Suppose we attempt to turn on our computer, but the computer does not start (observation/evidence). We would like to know which of the possible causes of computer failure is more likely. In this simplified illustration, we assume only two possible causes of this misfortune: electricity failure and computer malfunction.

The corresponding directed acyclic graph is depicted in below figure.



The goal is to calculate the posterior conditional probability distribution of each of the possible unobserved causes given the observed evidence, i.e., P [Cause | Evidence].

Data Set:

<u>Title:</u> Heart Disease Databases
The Cleveland database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "Heartdisease" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Database:	0	1	2	3	4	Total
Cleveland:	164	55	36	35	13	303

<u>Attribute Information:</u>

1. age: age in years
2. sex: sex (1 = male; 0 = female)
3. cp: chest pain type
    1. Value 1: typical angina
    2. Value 2: atypical angina
    3. Value 3: non-anginal pain
    4. Value 4: asymptomatic
4. trestbps: resting blood pressure (in mm Hg on admission to the hospital)
5. chol: serum cholestoral in mg/dl
6. fbs: (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)
7. restecg: resting electrocardiographic results
    1. Value 0: normal
    2. Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV)
    3. Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
8. thalach: maximum heart rate achieved
9. exang: exercise induced angina (1 = yes; 0 = no)
10. oldpeak = ST depression induced by exercise relative to rest
11. slope: the slope of the peak exercise ST segment
    1. Value 1: upsloping
    2. Value 2: flat
    3. Value 3: downsloping
12. thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
13. Heartdisease: It is integer valued from 0 (no presence) to 4.

Some instance from the dataset:

age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	Heart disease
63	1	1	145	233	1	2	150	o	2.3	3	o	6	o
67	1	4	160	286	o	2	108	1	1.5	2	3	3	2

67	1	4	120	229	o	2	129	1	2.6	2	2	7	1
41	o	2	130	204	o	2	172	o	1.4	1	o	3	o
62	o	4	140	268	o	2	160	o	3.6	3	2	3	3
60	1	4	130	206	o	2	132	1	2.4	2	2	7	4

Python Program to Implement and Demonstrate Bayesian network using pgmpy Machine Learning

```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
#read Cleveland Heart Disease data
heartDisease = pd.read_csv('heart.csv')
heartDisease = heartDisease.replace('?',np.nan)
#display the data
print('Sample instances from the dataset are given below')
print(heartDisease.head())
print('\n Attributes and datatypes')
print(heartDisease.dtypes)
#Model Bayesian Network
model=BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisea
se'),('heartdisease','restecg'),('heartdisease','chol')])
#Learning CPDs using Maximum Likelihood Estimators
print('\nLearning CPD using Maximum likelihood estimators')
model.fit(heartDisease,estimator=MaximumLikelihoodEstimator)
# Inferencing with Bayesian Network
print('\n Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
#computing the Probability of HeartDisease given Age
print('\n 1. Probability of HeartDisease given evidence= restecg')
q1=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
#computing the Probability of HeartDisease given cholesterol
print('\n 2. Probability of HeartDisease given evidence= cp ')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

Output

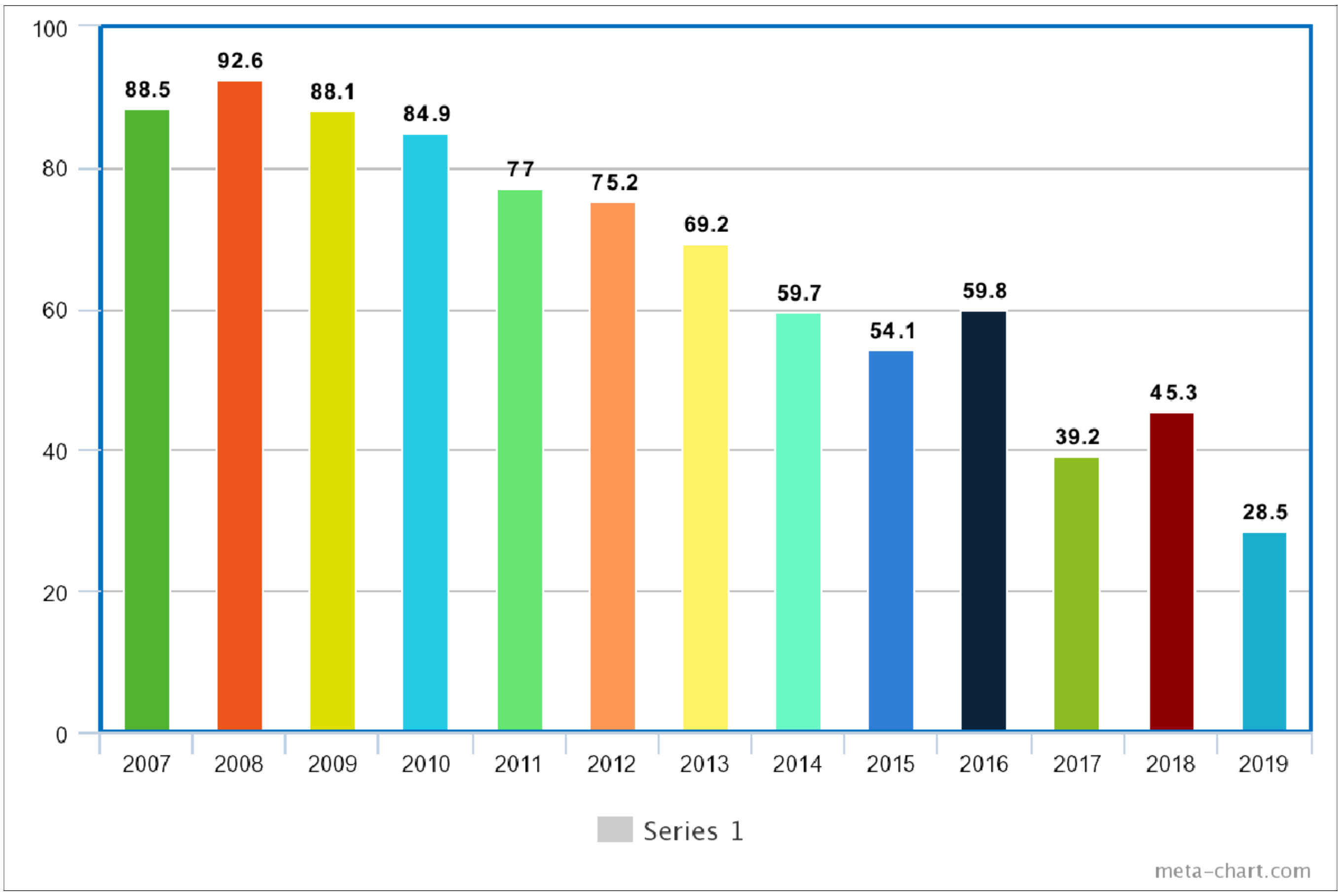
```
Learning CPD using Maximum likelihood estimators

 Inferencing with Bayesian Network:

 1. Probability of HeartDisease given evidence= restecg

   +-----------------+---------------------+
   | heartdisease    |   phi(heartdisease) |
   +=================+=====================+
   | heartdisease(0) |              0.1012 |
   +-----------------+---------------------+
   | heartdisease(1) |              0.0000 |
   +-----------------+---------------------+
   | heartdisease(2) |              0.2392 |
   +-----------------+---------------------+
   | heartdisease(3) |              0.2015 |
   +-----------------+---------------------+
   | heartdisease(4) |              0.4581 |
   +-----------------+---------------------+
2. Probability of HeartDisease given evidence= cp

   +-----------------+---------------------+
   | heartdisease    |   phi(heartdisease) |
   +=================+=====================+
   | heartdisease(0) |              0.3610 |
   +-----------------+---------------------+
   | heartdisease(1) |              0.2159 |
   +-----------------+---------------------+
   | heartdisease(2) |              0.1373 |
   +-----------------+---------------------+
   | heartdisease(3) |              0.1537 |
   +-----------------+---------------------+
   | heartdisease(4) |              0.1321 |
   +-----------------+---------------------+
```

Assignment 4

Problem Statement:
Implement e-mail spam filtering using text classification algorithm with appropriate dataset.

Objective:
1. Basic concepts of Spam Filtering.
2. To study KNN algorithm.

Theory:
In the new era of technical advancement, electronic mail (emails) has gathered significant users for professional, commercial, and personal communications. In 2019, on average, every person **received 130 emails each day,** and overall, **296 Billion** emails were sent that year.
Because of the high demand and huge user base, there is an upsurge in unwanted emails, also known as spam emails. At times, more than **50% of the total emails were spam.** Even today, people lose millions of dollars to fraud every day.
But, in the figure shown below, it can be observed that the quantity of such emails has decreased significantly after 2016 because of the evolution of the software that can detect these spam emails and can filter them out.

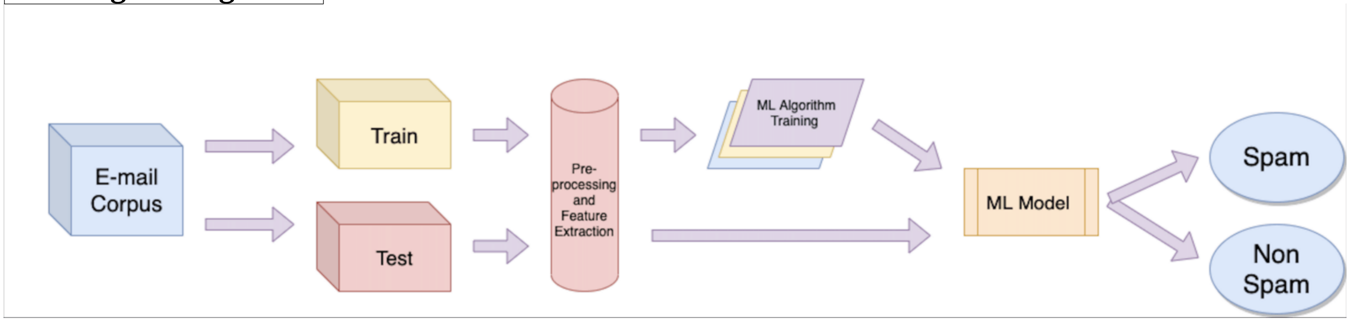


Percentage of emails marked as Spam (Source: **Statista)**
Steps to implement a Spam-classifier using the k-NN algorithm:
Many several techniques are present in the market to detect spam emails. If we want to classify broadly, there are five different techniques based on which algorithms decide whether any mail is spam.

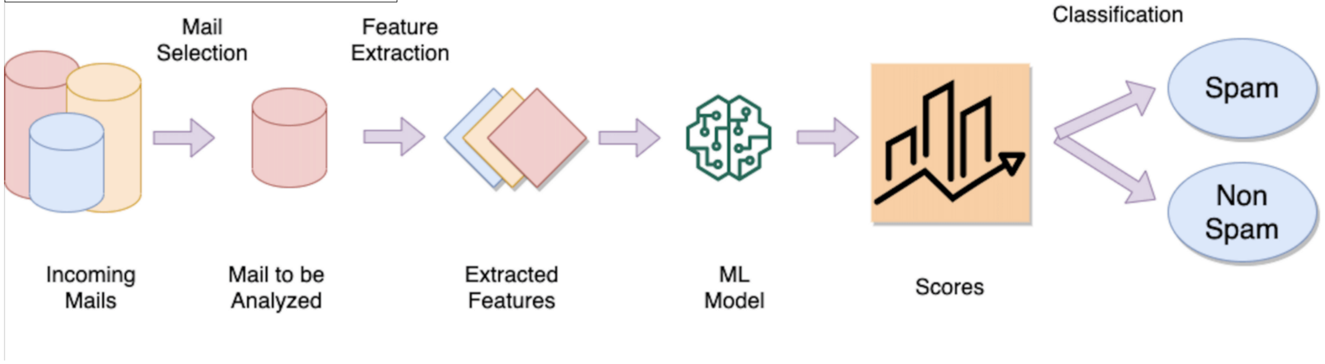
Content-Based Filtering Technique
Algorithms analyze words, the occurrence of words, and the distribution of words and phrases inside the content of emails and segregate them into spam and non-spam categories.
Case Base Spam Filtering Method
Algorithms trained on well-annotated spam/non-spam marked emails try to classify incoming emails into two categories.
Heuristic or Rule-Based Spam Filtering Technique
Algorithms use pre-defined rules as regular expressions to give a score to the messages in the emails. They segregate emails into spam and non-spam categories based on the scores generated.
The Previous Likeness Based Spam Filtering Technique
Algorithms extract the incoming mails' features and create a multi-dimensional space vector and draw points for every new instance. Based on the KNN algorithm, these new points get assigned to the closest class of spam and non-spam.
Adaptive Spam Filtering Technique
Algorithms classify the incoming emails into various groups and, based on the comparison scores of every group with the defined set of groups, spam and non-spam emails got segregated.

K-NN based algorithms:
K-NN based algorithms are widely used for clustering tasks. Let's quickly know the entire architecture of this implementation first and then explore every step. Executing these **five steps,** one after the other, will help us implement our spam classifier smoothly.

Training Testing Phase



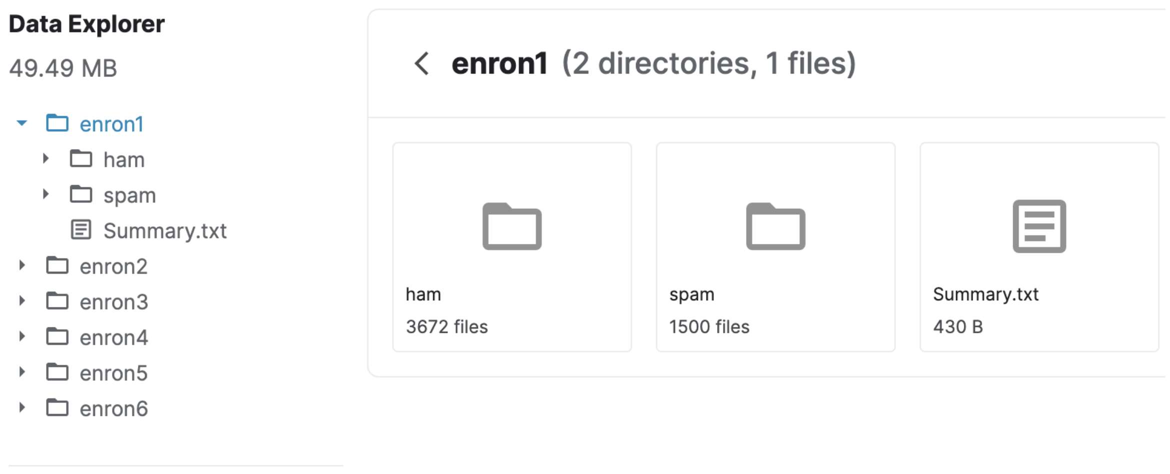
New Email Classification



Step 1: E-mail Data Collection
The dataset contained in a corpus plays a crucial role in assessing the performance of any spam filter. Many open-source datasets are freely available in the public domain. The two datasets below are widely popular as they contain many emails.
Enron corpus datasets (Created in 2006 and having 55% spam emails)
Trec 2007 dataset (Created in 2007 and having 67% spam emails)

Train/Test Split: Split the dataset into train and test datasets but make sure that both sets balance the numbers of ham and spam emails (ham is a fancy name for non-spam emails).

Enron Corpus Dataset on Kaggle



Step 2: Pre-processing of E-mail content

As the dataset is in text format, we will need the pre-processing of text data. At this step, we mainly perform tokenization of mail. Tokenization is a process where we break the content of an email into words and transform big messages into a sequence of representative symbols termed tokens. These tokens are extracted from the email body, header, subject, and image.

These days, senders have options to attach inline images to the mail. These emails can be categorized as spam, not based on their mail content but on the images' content. This was not an easy task until google came up with the open-source library Tesseract. This library extracts the words from images automatically with certain accuracy. But still, Times New Roman and Captcha words are challenging to read automatically.

Step 3: Feature Extraction and Selection

After pre-processing, we can have a large number of words. Here we can maintain a database that contains the frequency of the different words represented in each column. These attributes can be categorized on another basis, like:

Essential attributes: Frequency of repeated words, Number of semantic discrepancies, an Adult content bag of words, etc.

Additional Attributes: Sender account features like Sender country, IP address, email, age of Sender, Number of replies, Number of recipients, and website address. Note: These web addresses are converted in the word format only. For example, https://www.google.com/ can be converted to "HTTP google." Sometimes these processes are called Normalization.

Less important attributes: Geographical distance between sender and receiver, Sender's date of birth, Account lifespan, Sex of Sender, and Age of the recipient.

You must be clear that the more the number of attributes, → more the time complexity of the model.

These attributes can be tremendous, so techniques like Stemming, noise removal, and stop-word removal can be used. One of the famous stemming algorithms is the Porter-Stemmer Algorithm.

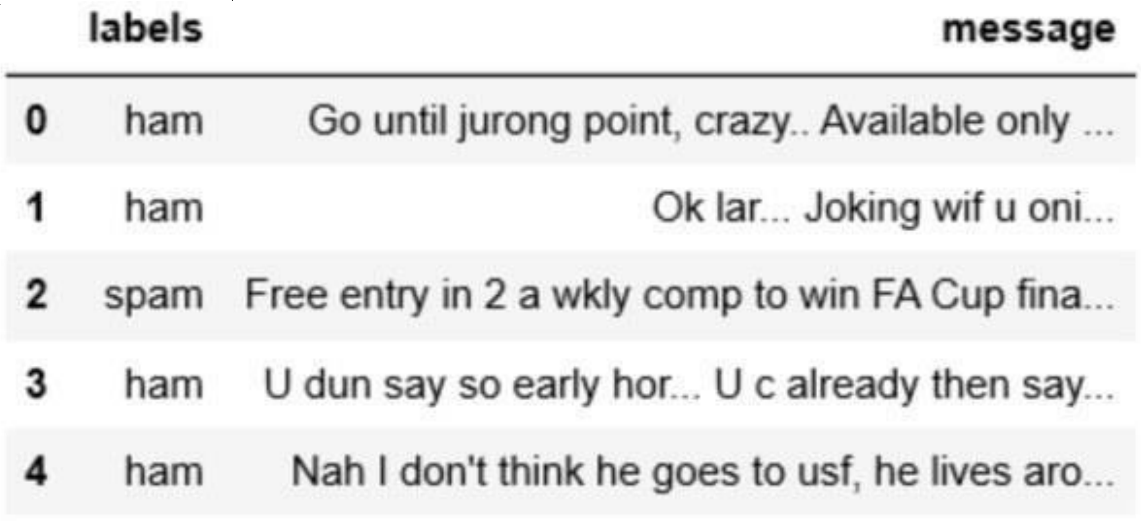
Some general things that we do in stemming are:

Removing suffixes (-ed, -ing, -full, -ness, etc.)

Removing prefixes (Un-, Re-, Pre-, etc.)

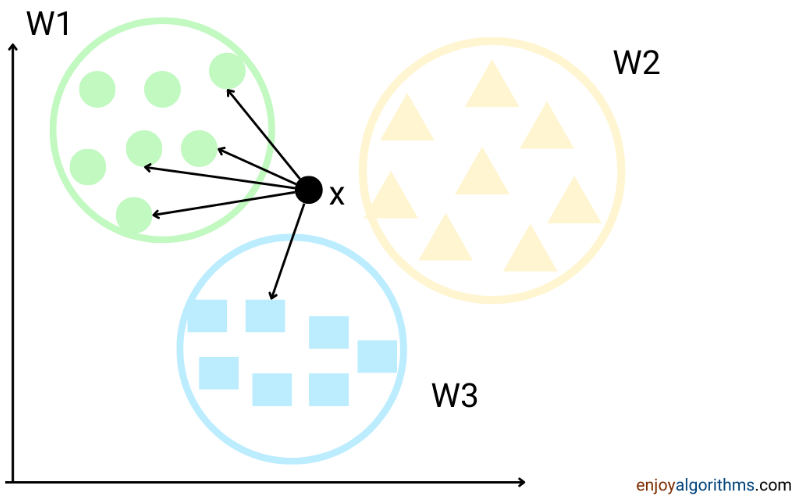
List of stop words ["i", "me", "my","him", "his", "himself", "she", “her", “hers", "herself",

Example dataset format



	labels	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...

Step 4: KNN (K-Nearest Neighbour) Implementation
Similar to the Nearest Neighbour algorithm, the K-Nearest Neighbour algorithm serves the purpose of clustering. Still, instead of giving just one nearest instance, it looks at the closest K instances to the new incoming instance. K-NN classifies the new instances based on the frequency of those K instances. The value of K is considered to be a hyper parameter that needs tuning. To tune this, one can take one of the famous Hit and Trial approaches, where we try some K values and then check the model's performance.



To find the nearest instance, one can use the Euclidean distance. One can use the Scikit-learn library to implement the K-NN algorithm for this task.

$$Euclidean = \sqrt{\Sigma_i^k (X_i - Y_i)^2}$$

Step 5: Performance Analysis
Our algorithm is ready, so we must check the model's performance. Even a single missed important message may cause a user to reconsider the value of spam filtering. So we must be sure that our algorithm will be as close to 100% accurate. But some researchers feel that more than accuracy as the evaluation parameter for spam classification is needed.

According to the below table (also known as the confusion matrix), we must evaluate our spam classification model based on four different parameters.

Accuracy : (TP + TN)/(TP + FP + FN + TN)
Precision: TP / (TP + FP)
Sensitivity: TP / (TP + FN)
Specificity: TN / (TN + FP)

More advanced algorithms are available for this classification, but you can easily achieve more than 90% accuracy using k-NN-based implementation.

Gmail, Yahoo, and Outlook Case Study

Gmail

Google data centers use thousands of rules to filter spam emails. They provide the weightage to different parameters; based on that, they filter the emails. Google's spam classifier is said to be astate of an art technique that uses various techniques like Optical character recognition, linear regression, and a combination of multiple neural networks.

Yahoo

Yahoo mail is the world's first free webmail service provider, with more than 320 million active users. They have their filtering techniques to categorize emails. Yahoo's basic methods are URL filtering, email content, and user spam complaints. Unlike Gmail, Yahoo filter emails messages by domain and not the IP address. Yahoo also provides users with custom-filtering options to send mail directly to junk folders.

Outlook

Microsoft-owned mailing platform widely used among professionals. In 2013, Microsoft renamed Hotmail and Windows Live Mail to Outlook. At present, outlook has more than 400 Million active users. Outlook has its distinctive feature based on which it filters every incoming mail. Based on their official website, they have provided the list of spam filters they use to send any mail in the junk folder, which includes:
Safe Senders list
Safe Recipients list
Blocked Senders list
Blocked Top-Level Domains list
Blocked Encodings list

Oral Questions

What is Porter Stemmer's Algorithm?
Why did you use the k-NN algorithm for this problem?
Is this supervised learning or unsupervised learning?
What are the different algorithms that can replace k-NN here?
What steps can be taken to improve accuracy further?

Conclusion

In terms of the Number of spam emails sent daily and the Number of money people lose every day because of these spam scams, Spam-filtering becomes the primary need for all email-providing companies. So, in this way we have discussed the complete process of spam email filtering using advanced machine learning technologies. We also have closed one possible way of implementing our spam classifier using one of the most famous algorithms, k-NN. We also discussed the case studies of well-known companies like Gmail, Outlook, and Yahoo to review how they use ML and AI techniques to filter such spammers.

Assignment 5

Problem Statement:

Implement Agglomerative hierarchical clustering algorithm using appropriate dataset.

Objective:

- Evaluate and analyze the working of Agglomerative hierarchical clustering algorithm.
- To study working of dendrograms.

Theory:

Prerequisites: Agglomerative Clustering Algorithm,
Dataset: Credit Card Dataset.

Assumption: The clustering technique assumes that each data point is similar enough to the other data points that the data at the starting can be assumed to be clustered in 1 cluster.

Step 1: Importing the required libraries

```
importpandas as pd
importnumpy as np
importmatplotlib.pyplot as plt
fromsklearn.decompositionimportPCA
fromsklearn.clusterimportAgglomerativeClustering
fromsklearn.preprocessingimportStandardScaler, normalize
fromsklearn.metricsimportsilhouette_score
importscipy.cluster.hierarchy as shc
```

Step 2: Loading and Cleaning the data

```
# Changing the working location to the location of the file
cd C:\Users\Dev\Desktop\Kaggle\Credit_Card
 X =pd.read_csv('CC_GENERAL.csv')
 # Dropping the CUST_ID column from the data
X =X.drop('CUST_ID', axis =1)
 # Handling the missing values
X.fillna(method ='ffill', inplace=True)
```

Step 3: Preprocessing the data

```
# Scaling the data so that all the features become comparable
scaler =StandardScaler()
X_scaled=scaler.fit_transform(X)
 # Normalizing the data so that the data approximately
# follows a Gaussian distribution
X_normalized=normalize(X_scaled)
 # Converting the numpy array into a pandas DataFrame
X_normalized=pd.DataFrame(X_normalized)
```

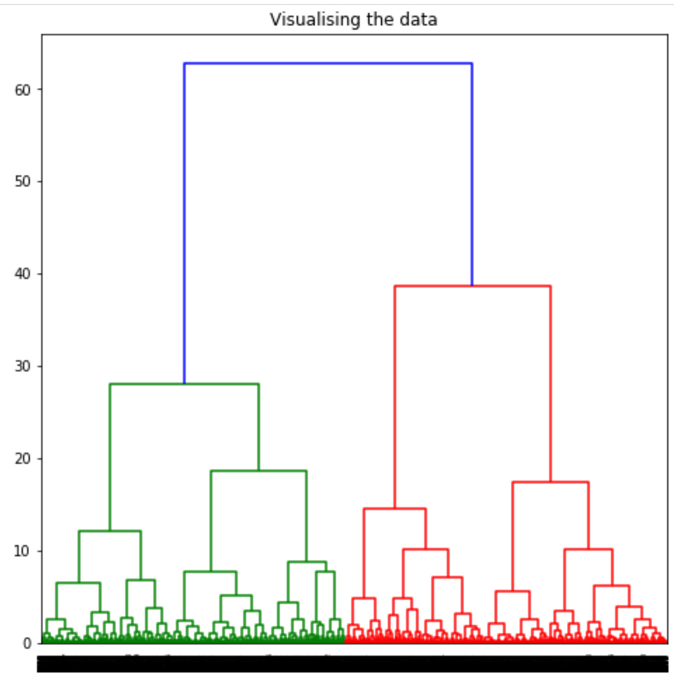
Step 4: Reducing the dimensionality of the Data

```
pca=PCA(n_components=2)
X_principal=pca.fit_transform(X_normalized)
X_principal=pd.DataFrame(X_principal)
X_principal.columns=['P1', 'P2']
```

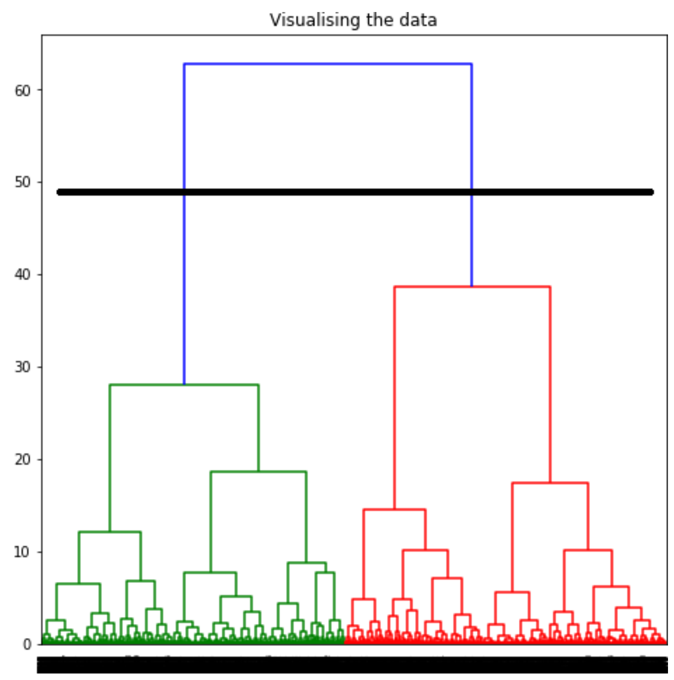
Dendrograms are used to divide a given cluster into many different clusters.

Step 5: Visualizing the working of the Dendrograms

```
plt.figure(figsize=(8, 8))
plt.title('Visualising the data')
Dendrogram =shc.dendrogram((shc.linkage(X_principal, method ='ward')))
```



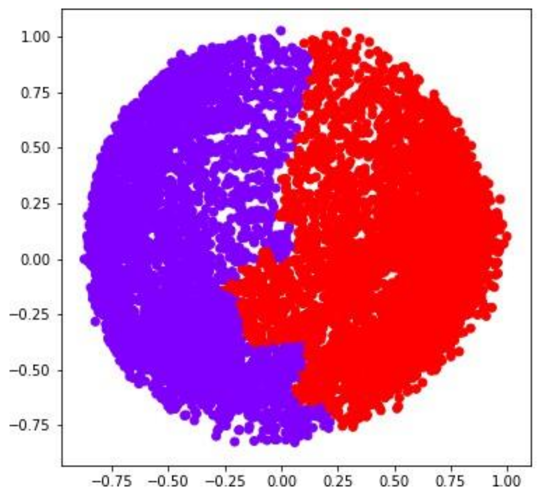
To determine the optimal number of clusters by visualizing the data, imagine all the horizontal lines as being completely horizontal and then after calculating the maximum distance between any two horizontal lines, draw a horizontal line in the maximum distance calculated.



The above image shows that the optimal number of clusters should be 2 for the given data.

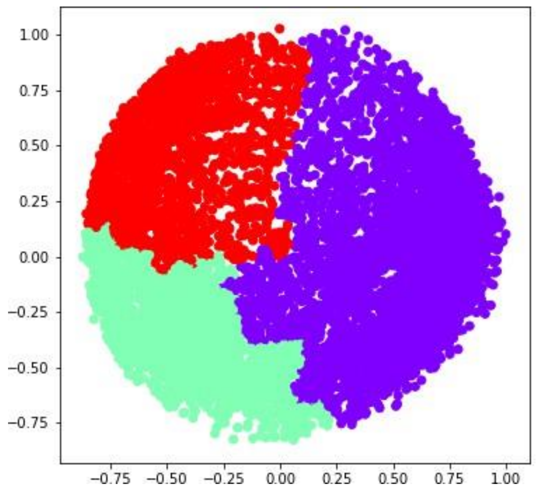
Step 6: Building and Visualizing the different clustering models for different values of k a) **k = 2**

```
ac2 =AgglomerativeClustering(n_clusters=2)
# Visualizing the clustering
plt.figure(figsize=(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
           c =ac2.fit_predict(X_principal), cmap='rainbow')
plt.show()
```



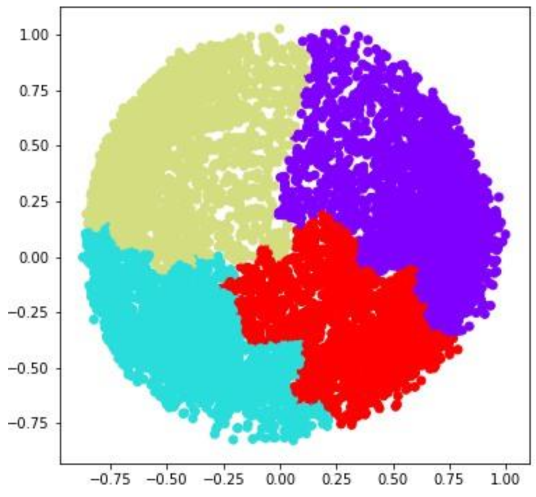
b) **k = 3**

```
ac3 =AgglomerativeClustering(n_clusters=3)

plt.figure(figsize=(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
        c =ac3.fit_predict(X_principal), cmap='rainbow')
plt.show()
```



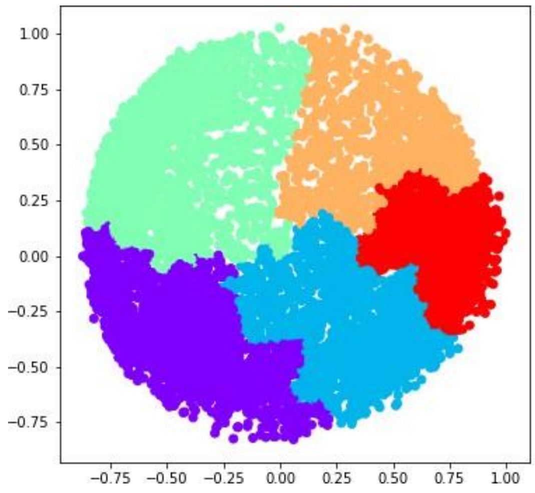
c) **k = 4**

```
ac4 =AgglomerativeClustering(n_clusters=4)

plt.figure(figsize=(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
         c =ac4.fit_predict(X_principal), cmap='rainbow')
plt.show()
```



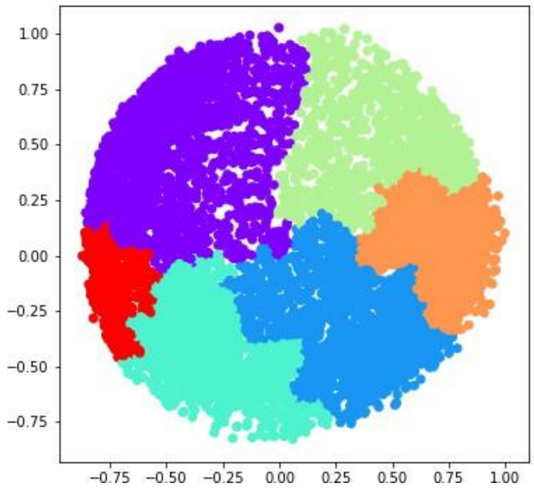
d) **k = 5**

```
ac5 =AgglomerativeClustering(n_clusters=5)

plt.figure(figsize=(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
         c =ac5.fit_predict(X_principal), cmap='rainbow')
plt.show()
```



e) **k = 6**

```
ac6 =AgglomerativeClustering(n_clusters=6)

plt.figure(figsize=(6, 6))
plt.scatter(X_principal['P1'], X_principal['P2'],
            c =ac6.fit_predict(X_principal), cmap='rainbow')
plt.show()
```

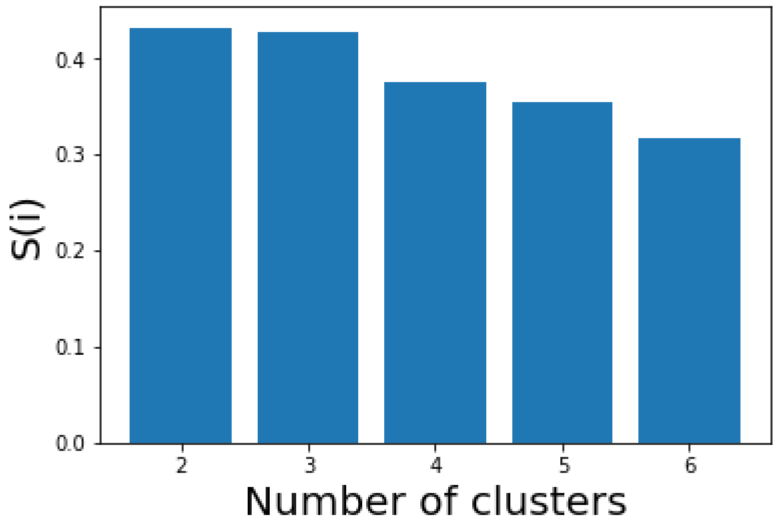


We now determine the optimal number of clusters using a mathematical technique. Here, we will use the **Silhouette Scores** for the purpose.

Step 7: Evaluating the different models and Visualizing the results.

```
k =[2, 3, 4, 5, 6]

# Appending the silhouette scores of the different models to the list
silhouette_scores=[]
silhouette_scores.append(
        silhouette_score(X_principal, ac2.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal,  ac3.fit_predict(X_principal)))
```

```
silhouette_scores.append(
        silhouette_score(X_principal, ac4.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal, ac5.fit_predict(X_principal)))
silhouette_scores.append(
        silhouette_score(X_principal,  ac6.fit_predict(X_principal)))

# Plotting a bar graph to compare the results
plt.bar(k, silhouette_scores)
plt.xlabel('Number of clusters', fontsize=20)
plt.ylabel('S(i)', fontsize=20)
plt.show()
```



Thus, with the help of the silhouette scores, it is concluded that the optimal number of clusters for the given data and clustering technique is 2.

Conclusion:

In this way, we have studied Agglomerative hierarchical clustering algorithm using appropriate dataset.