WEEK-3

Construction of an Inverted Index for a given document collection comprising of at least 50 documents with a total vocabulary size of at least 1000 words.

AIM:

To construct an inverted index for a collection of 50 documents and vocabulary of 1000 words.

DESCRIPTION:

An **inverted index** is a fundamental data structure in the field of information retrieval. It is designed to enable fast and efficient full-text searches over a large collection of documents. Instead of storing documents sequentially as they are, an inverted index reverses the natural relationship between documents and terms by mapping each **unique word (term)** in the corpus to a list of **documents** in which it appears. This allows for quick lookup of all documents containing a given word. In constructing an inverted index, the document collection must undergo several preprocessing steps to normalize and clean the data. These steps typically include

- converting all text to lowercase, removing punctuation and stopwords (common, non-informative words like “the,” “is,” “and”),
- tokenizing the text into words,
- applying stemming or lemmatization to reduce words to their root forms (e.g., “running” becomes “run”).

PYTHON CODE:

```
import os import nltk

from nltk.corpus import stopwords

from nltk.tokenize import word_tokenize

from nltk.stem import PorterStemmer

from collections import defaultdict import json

nltk.download('punkt')

nltk.download('stopwords')
```

```
def preprocess(text):
    text = text.lower()
    tokens = word_tokenize(text)
    stop_words = set(stopwords.words('english')) stemmer = PorterStemmer()
    words = [stemmer.stem(word) for word in tokens if word.isalnum() and word not in
    stop_words]
    return words documents = {}
for filename in os.listdir():
    if filename.endswith(".txt"):
        with open(filename, 'r', encoding='utf-8', errors='ignore') as f:
            text = f.read()
            documents[filename] = preprocess(text)
print(f"Total documents loaded: {len(documents)}")
inverted_index = defaultdict(set)
for doc_id, words in documents.items():
    for word in set(words): # avoid duplicates per document
        inverted_index[word].add(doc_id)
vocab_size = len(inverted_index)
print(f"\nVocabulary Size: {vocab_size} words")
print("\nSample inverted index terms:")
for term in list(inverted_index)[:10]:
    print(f"{term}: {sorted(inverted_index[term])}")
```

OUTPUT:

```
Total documents loaded: 11
defaultdict(<class 'set'>, {'today': {'HI.txt'}, 'work': {'HI.txt'}, 'warm': {'untitled4.txt', 'HI.txt'},
'hi': {'HI.txt'}, 'professor': {'HI.txt'}, 'aditya': {'HI.txt'}, 'sushuma': {'HI.txt'}, 'assist': {'HI.txt'},
```

'sunni': {'untitled4.txt'}})

Vocabulary Size: 9 words

Sample inverted index terms:

hi: ['HI.txt']

sushuma: ['HI.txt']

today: ['HI.txt']

aditya: ['HI.txt']

work: ['HI.txt']

professor: ['HI.txt']

assist: ['HI.txt']

warm: ['HI.txt', 'untitled4.txt']

sunni: ['untitled4.txt']

WEEK-4

Classification of a set of Text Documents into known classes (You may use any of the Classification algorithms like Naive Bayes, Max Entropy, Rochio‟s, Support Vector Machine). Standard Datasets will have to be used to show the results.

AIM:

To perform classification of text documents into known classes.

DESCRIPTION:

Introduction to Text Classification

Text classification is a fundamental task in Natural Language Processing (NLP) where a piece of text (e.g., a document, sentence, or paragraph) is assigned to one or more predefined categories. Applications include:

- Spam detection
- Sentiment analysis
- Topic labelling
- News categorization

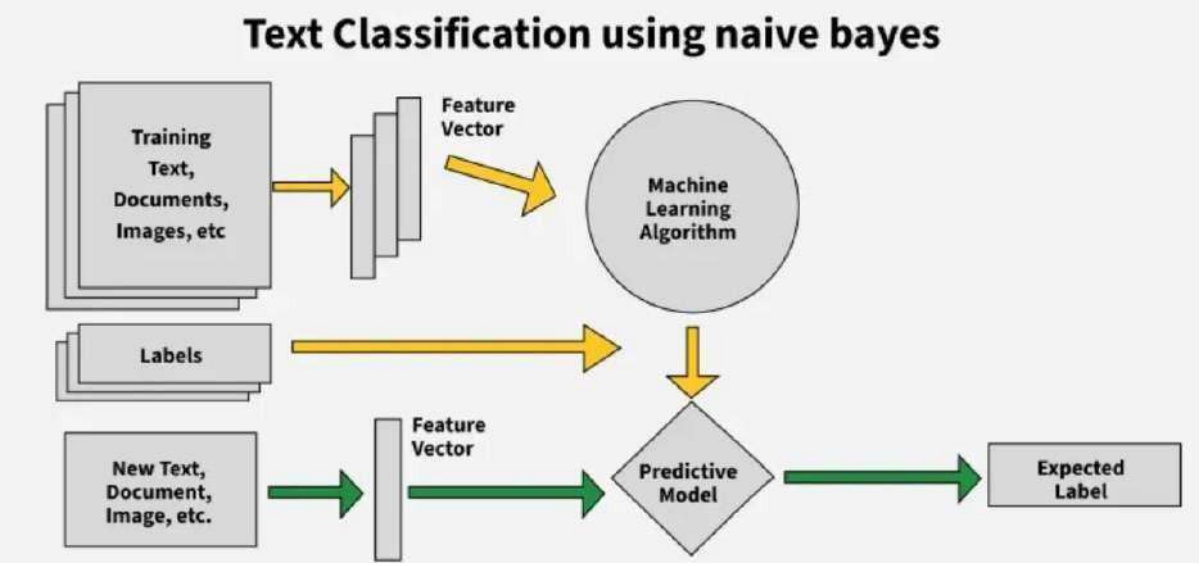
Naive Bayes

Naive Bayes is a classification algorithm that uses probability to predict which category a data point belongs to, assuming that all features are unrelated. It is named as "Naive" because it assumes the presence of one feature does not affect other features.

Why Naive Bayes?

The Naive Bayes classifier is a popular probabilistic machine learning algorithm based on Bayes' Theorem with an assumption of independence among features. It is particularly well- suited for text classification due to:

- **Efficiency** in handling high-dimensional data (like text)
- **Robustness** even with small training datasets
- **Scalability** and speed of training and prediction



PROGRAM:

```
from sklearn.datasets import fetch_20newsgroups

from sklearn.model_selection import train_test_split

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.naive_bayes import MultinomialNB

from sklearn.metrics import accuracy_score, classification_report

categories = ['sci.space', 'rec.sport.hockey', 'comp.graphics', 'alt.atheism']

newsgroups= fetch_20newsgroups(subset='all',categories=categories,shuffle=True, random_state=42)

print(f"Total documents: {len(newsgroups.data)}")

print(f"Target classes: {newsgroups.target_names}")

X_train, X_test, y_train, y_test = train_test_split (newsgroups.data, newsgroups.target, test_size=0.2,
random_state=42 )

vectorizer = TfidfVectorizer(stop_words='english')

X_train_tfidf = vectorizer.fit_transform(X_train)

X_test_tfidf = vectorizer.transform(X_test)

nb = MultinomialNB()
```

```
nb.fit(X_train_tfidf, y_train)
y_pred = nb.predict(X_test_tfidf)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n")
print(classification_report(y_test, y_pred, target_names=newsgroups.target_names))
for i in range(5):
    print("\nText:\n", X_test[i])
    print("Actual:", newsgroups.target_names[y_test[i]])
    print("Predicted:", newsgroups.target_names[y_pred[i]])
```

OUTPUT:

Accuracy: 0.9840425531914894

Classification Report:

	precision	recall	f1-score	support
alt.atheism	1.00	1.00	1.00	152
comp.graphics	0.96	0.99	0.97	196
rec.sport.hockey	0.99	1.00	1.00	194
sci.space	0.99	0.95	0.97	210
accuracy			0.98	752
macro avg	0.99	0.99	0.99	752
weighted avg	0.98	0.98	0.98	752

WEEK-6

Crawling/ Searching the Web to collect news stories on a specific topic (based on user input). The program should have an option to limit the crawling to certain selected websites only.

AIM:

To Crawl/ Search the Web to collect news stories on a specific topic.

DESCRIPTION:

Crawling

Crawling is the process by which a program (called a **crawler** or **spider**) automatically browses the web to collect and download information from websites.

Introduction

Web crawling is the automated process of systematically browsing and extracting information from web pages. In the context of news story collection, a crawler is programmed to search the internet for news articles related to a **specific topic** provided by the user. The goal is to collect relevant content from the web and present it in a usable format.

How It Works

1. **User Input**
    - The user specifies a **topic** (e.g., "Climate Change", "Artificial Intelligence", "Olympics 2024").
    - Optionally, the user can **limit the search to specific websites** (e.g., *bbc.com*, *nytimes.com*).
2. **URL Queue Initialization**
    - The crawler starts with a set of **seed URLs** (either from the user or pre-defined news portals).
3. **Fetching Pages**
    - The crawler downloads the HTML content of these web pages.

4. **Parsing & Extraction**
    - The HTML is analyzed to extract:
        - Headlines
        - Publication dates
        - Article content
        - Author names
    - Relevant **links to other news pages** are also extracted.
5. **Filtering by Topic**
    - Using **keyword matching** or **Natural Language Processing (NLP)**, the crawler selects only those articles related to the given topic.
6. **Storing Results**
    - The collected news stories are saved in a database or file for further reading, analysis, or summarization.

Install package: pip install serpapi beautifulsoup4 requests About package:

- **serpapi**: A Python wrapper for SerpAPI, which lets you retrieve Google/Bing/News results through their API (requires a SerpAPI key).
- **beautifulsoup4**: A Python library for parsing HTML and XML documents; commonly used to extract data from web pages.
- **requests**: A popular library to send HTTP requests easily in Python.

PROGRAM:

```
pip install serpapi beautifulsoup4 requests

import requests

from bs4 import BeautifulSoup

topic = input("Enter the news topic to search for: ")
```

```
websites = input("Enter comma-separated websites to limit crawling (e.g., bbc.com,cnn.com):
        ").split(',')
SERP_API_KEY = '8d6bc2b3eef2e66c277a5a34be29b70d490834e929934539b15ae91c71dd569c'
search_url = 'https://serpapi.com/search.json'
def search_news(topic, websites):
    all_results = []
    for site in websites:
        params = {
            "engine": "google",
            "q": f"{topic} site:{site.strip()}",
            "api_key": SERP_API_KEY
        }
        response = requests.get(search_url, params=params)
        data = response.json()
        if "organic_results" in data:
            for result in data["organic_results"]:
                title = result.get("title")
                link = result.get("link")
                snippet = result.get("snippet", "")
                all_results.append((title, link, snippet))
    return all_results
def display_results(results):
    for idx, (title, link, snippet) in enumerate(results, start=1):
        print(f"\nNews {idx}:")
        print(f"Title  : {title}")
```

```
            print(f"URL     : {link}")
            print(f"Summary : {snippet}")
results = search_news(topic, websites)
if results:
    display_results(results)
else:
    print("No results found.")
```

OUTPUT:

Enter the news topic to search for: AI in healthcare

Enter comma-separated websites to limit crawling (e.g., bbc.com,cnn.com): bbc.com,cnn.com

News 1:

Title : AI in healthcare: what are the risks for the NHS?

URL : https://www.bbc.com/news/articles/c6233x9k4dlo

Summary : Generative AI will be transformative for NHS patient outcomes, a senior government advisor says.

News 2:

Title : How AI can spot diseases that doctors aren't looking for

URL : https://www.bbc.com/news/articles/c9q7zqy1xlpo

Summary : AI can take a second look at medical scans and flag up potential problems that doctors might not see.

News 3:

Title : How AI Has Transformed Healthcare's Future

URL : https://www.bbc.com/storyworks/hpe-greenlake/how-ai-has-transformed-healthcares-future

Summary : AI can link seemingly unrelated information to reveal new research pathways that yield better results. For example, AI models have identified potential ...

News 4:

Title : Hospitals will use AI to speed up patient care

URL : https://www.bbc.com/news/articles/cye0yywdegdo

Summary : Hospitals across the region are to use artificial intelligence (AI) technology to reduce unnecessary admissions and lengthy stays, ...

News 5:

Title : Can AI help modernise Ireland's healthcare system?

URL : https://www.bbc.com/news/articles/cly7yxm3py5o

Summary : Ireland is investing billions of euros to revamp its healthcare service - will AI help?

News 6:

Title : How artificial intelligence is matching drugs to patients

URL : https://www.bbc.com/news/business-65260592

Summary : Health-tech firms around the world are increasingly using AI to help tailor drugs for patients.