# Shivam Wagh

DSBDA Practical No A-7: Text Analytics

1. Extract Sample document and apply following document preprocessing methods: Tokenization, POS Tagging, stop words removal, Stemming and Lemmatization.
2. Create representation of document by calculating Term Frequency and Inverse Document Frequency.

```
pip install nltk scikit-learn
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (3.9.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (1.6.1)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk) (4.67.1)
Requirement already satisfied: numpy>=1.19.5 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn) (3.6.0)
```

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk import pos_tag
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
import string
```

```
# Download necessary NLTK datasets
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
# Input text
text = "Machine learning is very very important because it allows computers to learn from data."
```

```
nltk.download('punkt_tab')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt_tab.zip.
True
```

```
# Step 1: Tokenization
tokens = word_tokenize(text)
tokens
```

```
['Machine',
 'learning',
 'is',
 'very',
 'very',
 'important',
 'because',
 'it',
 'allows',
 'computers',
 'to',
 'learn',
 'from',
 'data',
 '.']
```

```
nltk.download('averaged_perceptron_tagger_eng')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger_eng to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger_eng.zip.
True
```

```
# Step 2: POS Tagging
pos_tags = pos_tag(tokens)
pos_tags
```

```
[('Machine', 'NN'),
 ('learning', 'NN'),
 ('is', 'VBZ'),
 ('very', 'RB'),
 ('very', 'RB'),
 ('important', 'JJ'),
 ('because', 'IN'),
 ('it', 'PRP'),
 ('allows', 'VBZ'),
 ('computers', 'NNS'),
 ('to', 'TO'),
 ('learn', 'VB'),
 ('from', 'IN'),
 ('data', 'NNS'),
 ('.', '.')]
```

```
# Step 3: Stop Words Removal
stop_words = set(stopwords.words('english'))
print(stop_words)
```

```
{'didn', 'whom', 'they', 'my', 'y', 'each', 'll', 'that', 'weren', 'but', 'on', 'some', 'same', 'there', "he's", 'yourselves', "he'll",
```

```
filtered_tokens = [word for word in tokens if word.lower() not in stop_words and word not in string.punctuation]
filtered_tokens
```

```
['Machine', 'learning', 'important', 'allows', 'computers', 'learn', 'data']
```

```
# Step 4: Stemming
stemmer = PorterStemmer()
stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
stemmed_tokens
```

```
['machin', 'learn', 'import', 'allow', 'comput', 'learn', 'data']
```

```
# Step 5: Lemmatization
lemmatizer = WordNetLemmatizer()
lemmatized_tokens = [lemmatizer.lemmatize(word) for word in filtered_tokens]
lemmatized_tokens
```

```
['Machine', 'learning', 'important', 'allows', 'computer', 'learn', 'data']
```

```
# Step 6: Term Frequency (TF)
word_count = len(filtered_tokens)
tf = {word: filtered_tokens.count(word) / word_count for word in filtered_tokens}
tf
```

```
{'Machine': 0.14285714285714285,
 'learning': 0.14285714285714285,
 'important': 0.14285714285714285,
 'allows': 0.14285714285714285,
 'computers': 0.14285714285714285,
 'learn': 0.14285714285714285,
 'data': 0.14285714285714285}
```

```
# Step 7: Inverse Document Frequency
import math


def calculate_idf(filtered_tokens, total_documents):
    idf_values = {word: math.log(total_documents / 1) for word in filtered_tokens} # log(1) = 0
    return idf_values
```

```
# Given filtered tokens
filtered_tokens = ['Machine', 'learning', 'important', 'allows', 'computers', 'learn', 'data']


# Total number of documents (since we have only one document, N = 1)
total_documents = 1


# Calculate IDF
idf_results = calculate_idf(filtered_tokens, total_documents)


# Print IDF values
print("IDF values for the given words:")
for word, idf in idf_results.items():
  print(f"{word}: {idf}")
```

```
IDF values for the given words:
Machine: 0.0
learning: 0.0
important: 0.0
allows: 0.0
computers: 0.0
learn: 0.0
data: 0.0
```

```
# Step 8: TF-IDF=> Term Frequency-Inverse Document Frequency


def calculate_tf_idf(tf_values, idf_values):
  tf_idf_values = {word: tf_values[word] * idf_values[word] for word in tf_values}
  return tf_idf_values


tf_idf_results = calculate_tf_idf(tf, idf_results)


# Print TF-IDF values
print("TF-IDF values for the given words:")
for word, tf_idf in tf_idf_results.items():
  print(f"{word}: {tf_idf}")
```

```
TF-IDF values for the given words:
Machine: 0.0
learning: 0.0
important: 0.0
allows: 0.0
computers: 0.0
learn: 0.0
data: 0.0
```

```
##### End of Program #####


# Alternate Method for all above steps
# Use TfidfVectorizer() from scikit-learn
# Handles preprocessing automatically (removes stop words, normalizes words)


text_filtered = "Machine learning important allows computers learn data"


vectorizer = TfidfVectorizer()
tfidf_matrix = vectorizer.fit_transform([text_filtered])


# Extract feature names (words) from the vectorizer
tfidf_feature_names = vectorizer.get_feature_names_out()


# Display TF, IDF, and TF-IDF values
print("\nWord\tTF\tIDF\tTF-IDF")
for i, word in enumerate(tfidf_feature_names):
  idf = vectorizer._tfidf.idf_[i] # Access IDF value from the vectorizer
  tfidf_value = tfidf_matrix[0, i]
  print(f"{word}\t{tf.get(word, 0):.4f}\t{idf:.4f}\t{tfidf_value:.4f}")
```

```

Word    TF      IDF     TF-IDF
```

```
allows  0.1429  1.0000  0.3780
computers       0.1429  1.0000  0.3780
data    0.1429  1.0000  0.3780
important       0.1429  1.0000  0.3780
learn   0.1429  1.0000  0.3780
learning        0.1429  1.0000  0.3780
machine 0.0000  1.0000  0.3780
```

```
# TfidfVectorizer() gives different results than manual calculations because of
# default settings like sublinear TF scaling, smoothing, stop-word removal, L2 Normalization etc
```

Start coding or generate with AI.