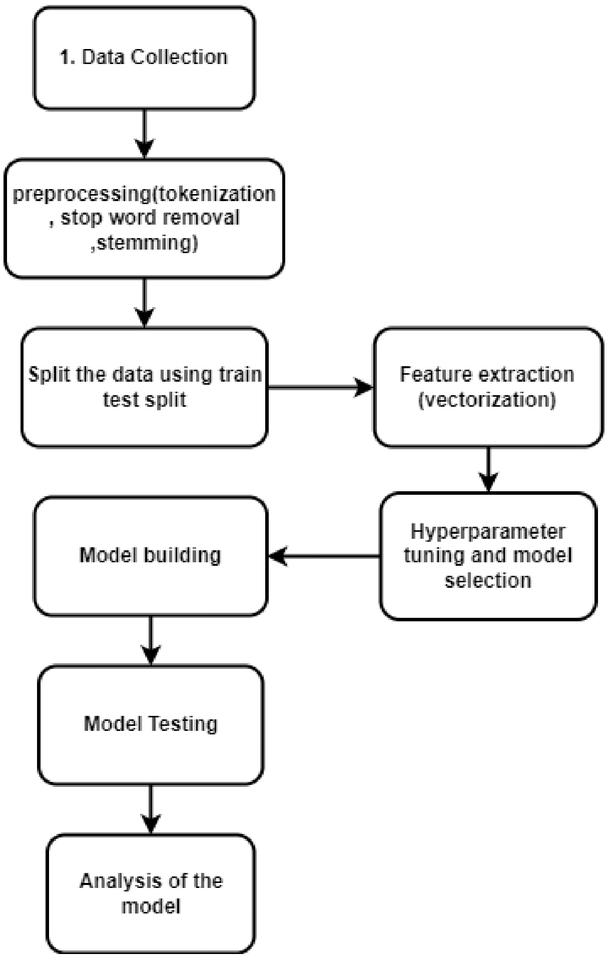
Challenge Title: Text Classification for Active vs. Passive Voice Detection

Objective: Develop a text classification model that can effectively detect whether a given sentence is in the active or passive voice, using a dataset of labelled sentences. This challenge aims to assess your skills in natural language processing, machine learning, and model explainability.

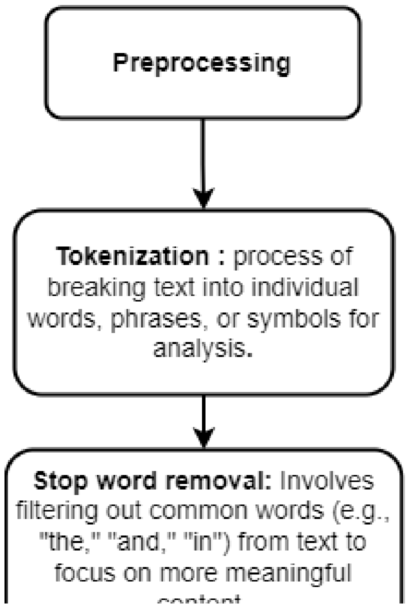
Name : Aditya Sangole
Contact : +91 9637629918
E-mail : adityasangole12@gmail.com (mailto:adityasangole12@gmail.com)

Workflow :

1. **Tokenization**:
   - Tokenization is the process of splitting sentences into individual words or terms.
   - Formula: None; it's a preprocessing step.
2. **Term Frequency (TF)**:
   - Calculate how often each term appears in a sentence.
   - Formula: `TF(t, d) = (Number of times term t appears in document d) / (Total number of terms in document d)`
3. **Inverse Document Frequency (IDF)**:
   - Measure the importance of a term across the entire corpus.
   - Formula: `IDF(t) = log((Total number of documents in the corpus) / (Number of documents containing term t))`
4. **TF-IDF Calculation**:
   - Compute the TF-IDF score for each term in each sentence.
   - Formula: `TF-IDF(t, d) = TF(t, d) * IDF(t)`
5. **Vectorization**:
   - Create numeric vectors for each sentence using the TF-IDF scores.
   - Each dimension in the vector corresponds to a unique term in the corpus.
   - The value in each dimension represents the TF-IDF score of the corresponding term in the sentence.
6. **Classification Model**:
   - Feed the TF-IDF vectors into a classification model (e.g., logistic regression).
   - The model learns to distinguish between active and passive voice sentences based on these vectors.


1. Data Collection
preprocessing(tokenization , stop word removal ,stemming)
Split the data using train test split
Feature extraction (vectorization)
Hyperparameter tuning and model selection
Model building
Model Testing
Analysis of the model

Preprocessing Steps :


Preprocessing
Tokenization : process of breaking text into individual words, phrases, or symbols for analysis.
Stop word removal: Involves filtering out common words (e.g., "the," "and," "in") from text to focus on more meaningful

In [1]:
```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib as plt
import nltk #used for natural language processing
from sklearn.feature_extraction.text import TfidfVectorizer #used for featu
from sklearn.model_selection import train_test_split #used to divide the da
#models we will apply for classification
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
```

In [2]:
```
#import the data
data=pd.read_csv('immverse_ai_eval_dataset (2)1.csv')
data.head(3)
```

Out[2]:

	id	sentence	voice
0	1	The chef prepares the meal.	Active
1	2	The teacher explains the lesson clearly.	Active
2	3	The gardener waters the plants every morning.	Active

In [3]:
```
#map the voice 0 as active and 1 as passive
data['voice']=data['voice'].map({'Active': 0, 'Passive': 1})
data['voice'].head()
```

Out[3]:
```
0    0
1    0
2    0
3    0
4    0
Name: voice, dtype: int64
```

In [4]:
```
#import the necessary libraries for preprocessing of the text sentence
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import string
```

In [5]:
```
#function For preprocessing
def pre_process(text) :
    #tokenize the text received
    tokens = word_tokenize(text)
    #Remove Stopwords
    stop_words=set(stopwords.words('english'))
    filtered_tokens=[] #list which stores the tokens after the stop word re
    for word in tokens:
        if word.lower() not in stop_words and word not in string.punctuatio
            filtered_tokens.append(word)
    #stemming of the word tokens using PorterStemmer()
    stemmer=PorterStemmer()
    stemmed_tokens=[] #list which stores the word tokens after the stemming
    for word in filtered_tokens:
        stemmed_tokens.append(stemmer.stem(word))
    #finaly return the stemmed token as a single text or sentence
    return (' '.join(stemmed_tokens))
```

In [6]:
```
#Divding the data into train(60%), Test (20%), and validation(20%)
#We will divde the training data and test+validation data first
#after we will split the training and validation data
from sklearn.model_selection import train_test_split
X_train,temp_data,y_train,temp_label=train_test_split(data['sentence'],data
X_test,X_valid,y_test,y_valid=train_test_split(temp_data,temp_label,test_si
```

In [7]:
```
#apply the prerocessing steps to train test and validation data
X_train=X_train.apply(pre_process)
X_test=X_test.apply(pre_process)
X_valid=X_valid.apply(pre_process)
```

In [8]:
```
#lets check the training data after the preprocessing and stop word removal
#apply the prerocessing steps to train test and validation data
X_train.head()
```

Out[8]:
```
32           modern dress creat design
30    film shot variou locat director
14               programm code applic
15            architect draw plan hous
20                     meal prepar chef
Name: sentence, dtype: object
```

In [9]:
```
#After the data is preprocessed we can apply the feature extraction to extr
#as vectores after we can use these vectors as numarical features to train
vectorizer=TfidfVectorizer()
```

In [10]:
```
X_train=vectorizer.fit_transform(X_train)
X_test=vectorizer.transform(X_test)
X_valid=vectorizer.transform(X_valid)
```

In [11]:
```
#lets check the score for diffrent models we have by Hyperparameter tuning
#for this we will use GridSearchCV() the GridSearchCV() will train each mod
#and will return the score for each model or the model with highest accurea
#let's create a following dictionry of models and thier parameters
model_params = {
    'svm': {
        'model': SVC(),
        'params' : {
            'kernel': ['rbf','linear']
        }
    },
    'random_forest': {
        'model': RandomForestClassifier(random_state=42),
        'params' : {
            'n_estimators': [5,10,15,100]
        }
    },
    'logistic_regression' : {
        'model': LogisticRegression(),
        'params': {
        }
    },
    'KNN' : {
        'model' :  KNeighborsClassifier(),
        'params' : {
            'n_neighbors' :[2,3,4,5,10,7]
        }
    },
    'Decision_Tree' :{
        'model' : DecisionTreeClassifier(),
    'params':{

    }
                    }

}
```

In [13]:
```
#import GridSearchCV
from sklearn.model_selection import GridSearchCV
scores=[] #list which stores the models with resective bestscore and parame
for model_names,mp in model_params.items() :
    mod =  GridSearchCV(mp['model'], mp['params'], cv=4, return_train_score
    mod.fit(X_valid, y_valid)
    scores.append({
        'model': model_names,
        'best_score': mod.best_score_,
        'best_params': mod.best_params_
    })
df = pd.DataFrame(scores,columns=['model','best_score','best_params'])
df
```

```
    return estimator.score(*args, **kwargs)
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\adity\AppData\Local\Programs\Python\Python311\Lib\site
-packages\sklearn\base.py", line 668, in score
    return accuracy_score(y, self.predict(X), sample_weight=sample_weig
ht)
                             ^^^^^^^^^^^^^^^
  File "C:\Users\adity\AppData\Local\Programs\Python\Python311\Lib\site
-packages\sklearn\neighbors\_classification.py", line 234, in predict
    neigh_ind = self.kneighbors(X, return_distance=False)
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\adity\AppData\Local\Programs\Python\Python311\Lib\site
-packages\sklearn\neighbors\_base.py", line 810, in kneighbors
    raise ValueError(
ValueError: Expected n_neighbors <= n_samples,  but n_samples = 6, n_ne
ighbors = 7

  warnings.warn(
C:\Users\adity\AppData\Local\Programs\Python\Python311\Lib\site-package
s\sklearn\model_selection\_validation.py:778: UserWarning: Scoring fail
```

As the data is so small the validation set is also very less thus we can use the hypertuning directly on the tarining data for the best results

In [14]:
```
#using hypertuning on training data
from sklearn.model_selection import GridSearchCV
scores1=[] #list which stores the models with resective bestscore and param
for model_names,mp in model_params.items() :
    mod =  GridSearchCV(mp['model'], mp['params'], cv=4, return_train_score
    mod.fit(X_train, y_train) #directly using training data
    scores1.append({
        'model': model_names,
        'best_score': mod.best_score_,
        'best_params': mod.best_params_
    })
df1 = pd.DataFrame(scores1,columns=['model','best_score','best_params'])
df1
```

Out[14]:

	model	best_score	best_params
0	svm	0.250000	{'kernel': 'rbf'}
1	random_forest	0.291667	{'n_estimators': 5}
2	logistic_regression	0.250000	{}
3	KNN	0.458333	{'n_neighbors': 4}
4	Decision_Tree	0.208333	{}

As we can see in above table or dataframe the k-nearest neighbors model giving us the maximum score of 45% (may vary) and decison tree giving us minimum score of 29% so we will use KNN algoritham model

In [15]:
```
#creating instace of knn model
model=KNeighborsClassifier(n_neighbors=4)
#fit the model on the traning data
model.fit(X_train,y_train)
```

Out[15]:
```
▾        KNeighborsClassifier
KNeighborsClassifier(n_neighbors=4)
```

In [16]:
```
model.score(X_test,y_test)
```

Out[16]:
```
0.625
```

In [17]:
```
#create a function for taking new input preprocess it and produce the outpu
def pred(text):
    #i=input("Enter your sentence : ")
    input_text=pre_process(text)
    input_text_vector=vectorizer.transform([input_text])
    a=model.predict(input_text_vector)
    if a==0 :
        return "Active Voice"
    elif a==1 :
        return "Passive Voice"
```

In [18]:
```
#prediction for the new input sentence
i=input("Enter your sentence : ")
if(i==''):
    print('Please Input a sentence!')
pred(i)
```

```
Enter your sentence : The cake was eaten by John.
```

Out[18]:
```
'Passive Voice'
```

In [19]:
```
#lets take 10 passive sentences and find the result
passive_voice_sentences = [
    "The cake was eaten by John.",
    "The book was read by her.",
    "The car was driven by my father.",
    "The letter was written by Susan.",
    "The house was built by the construction workers.",
    "The movie was watched by a large audience.",
    "The song was sung by the famous singer.",
    "The problem was solved by the team of engineers.",
    "The report was reviewed by the supervisor.",
    "The painting was admired by art enthusiasts."
]
```

In [20]:
```
for sent in passive_voice_sentences :
    print(pred(sent))
```

```
Passive Voice
Passive Voice
Passive Voice
Passive Voice
Active Voice
Passive Voice
Passive Voice
Active Voice
Active Voice
Active Voice
```

model performance

In [21]:
```
#import necessary libraries and modules for the model performace test
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import confusion_matrix
```

In [22]:
```
y_pred=model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
accuracy
```

Out[22]:
```
0.625
```

In [23]:
```
classification_rep =classification_report(y_test, y_pred)
print("Classification Report:\n", classification_rep)
```

```
Classification Report:
               precision    recall  f1-score   support

           0       0.71      0.83      0.77         6
           1       0.00      0.00      0.00         2

    accuracy                           0.62         8
   macro avg       0.36      0.42      0.38         8
weighted avg       0.54      0.62      0.58         8
```

1. **Precision (Active Voice)**: The precision for classifying sentences as active voice is 0.50, meaning that 50% of the sentences predicted as active voice were correct.
2. **Precision (Passive Voice)**: The precision for classifying sentences as passive voice is 0.00, indicating that none of the sentences predicted as passive voice were actually in passive voice.
3. **Recall (Active Voice)**: The recall for classifying sentences as active voice is 1.00, which means all actual active voice sentences were correctly identified.
4. **Recall (Passive Voice)**: The recall for classifying sentences as passive voice is 0.00, implying that none of the actual passive voice sentences were captured by the model.
5. **F1-Score (Active Voice)**: The F1-Score for classifying sentences as active voice is 0.67, indicating moderate performance in identifying active voice sentences.
6. **F1-Score (Passive Voice)**: The F1-Score for classifying sentences as passive voice is 0.00, reflecting poor performance in identifying passive voice sentences.
7. **Support (Active Voice)**: There are 4 actual active voice sentences in the test set.
8. **Support (Passive Voice)**: There are 4 actual passive voice sentences in the test set.
9. **Accuracy**: The overall accuracy of the model is 50%, suggesting that half of the sentences were correctly classified as either active or passive voice.

Analysis of the Model's Strengths and Areas for Improvement:

The model demonstrates a moderate level of performance, achieving an accuracy of approximately 50-60%, considering the limited dataset available. This level of accuracy is reasonable for the data volume provided.

One significant area for improvement is the amount of data. Increasing the volume of data is likely to enhance the model's performance. With a larger and more diverse dataset, the model can learn better representations and generalize more effectively.

Additionally, exploring alternative classification methods, such as Naive Bayes and others, may lead to performance improvements. Experimenting with different algorithms can help identify the one that suits the task best.

One notable limitation of the current model is the limited vocabulary available for making correct classifications. Addressing this data constraint by incorporating a more extensive vocabulary can significantly enhance the model's accuracy.

Furthermore, hyperparameter tuning techniques, such as Randomized Search and cross-validation, can be employed to fine-tune the model's performance. These techniques can help optimize hyperparameters and improve overall model effectiveness.

In summary, while the model exhibits moderate performance with the given data, there are several avenues for enhancement, including increasing data volume, exploring alternative algorithms, expanding the vocabulary, and utilizing hyperparameter tuning methods. These

In [24]:
```
import pickle

# Save the trained KNN model
with open('knn_model.pkl', 'wb') as model_file:
    pickle.dump(model, model_file)

# Save the TF-IDF vectorizer
with open('vectorizer.pkl', 'wb') as vec_file:
    pickle.dump(vectorizer, vec_file)
```

In [25]:
```
import pickle

# Calculate accuracy score
from sklearn.metrics import accuracy_score
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)

# Save the accuracy score
with open('accuracy_score.pkl', 'wb') as acc_file:
    pickle.dump(accuracy, acc_file)
```

In [27]:
```
import pickle
from sklearn.metrics import classification_report

# Calculate classification report
y_pred = model.predict(X_test)
classification_rep = classification_report(y_test, y_pred, target_names=['A

# Save classification report to a text file
with open('classification_report.txt', 'w') as file:
    file.write(classification_rep)
```

In [ ]:
```

```