GROUP 5: ADVAITH ACHANTA
AMOGH JAGINI
B. SREE VANI
S.HARIPRIYA
JYOTI NAIN

IMDB MOVIE REVIEWS

BHAVAN'S VIVEKANADA COLLEGE,SAINIKPURI | BSC HONS. IN DATA SCIENCE

BACKGROUND

The IMDB dataset is a binary classification dataset consisting of movie reviews from the Internet Movie Database. The dataset contains 50,000 movie reviews.

OBJECTIVE

Use TFIDF vectors to convert text to vectors and word2vec to convert word to vectors then apply a machine learning algorithm.

PATH

We utilised the Machine Learning algorithm to make predictions and obtained a high degree of accuracy.

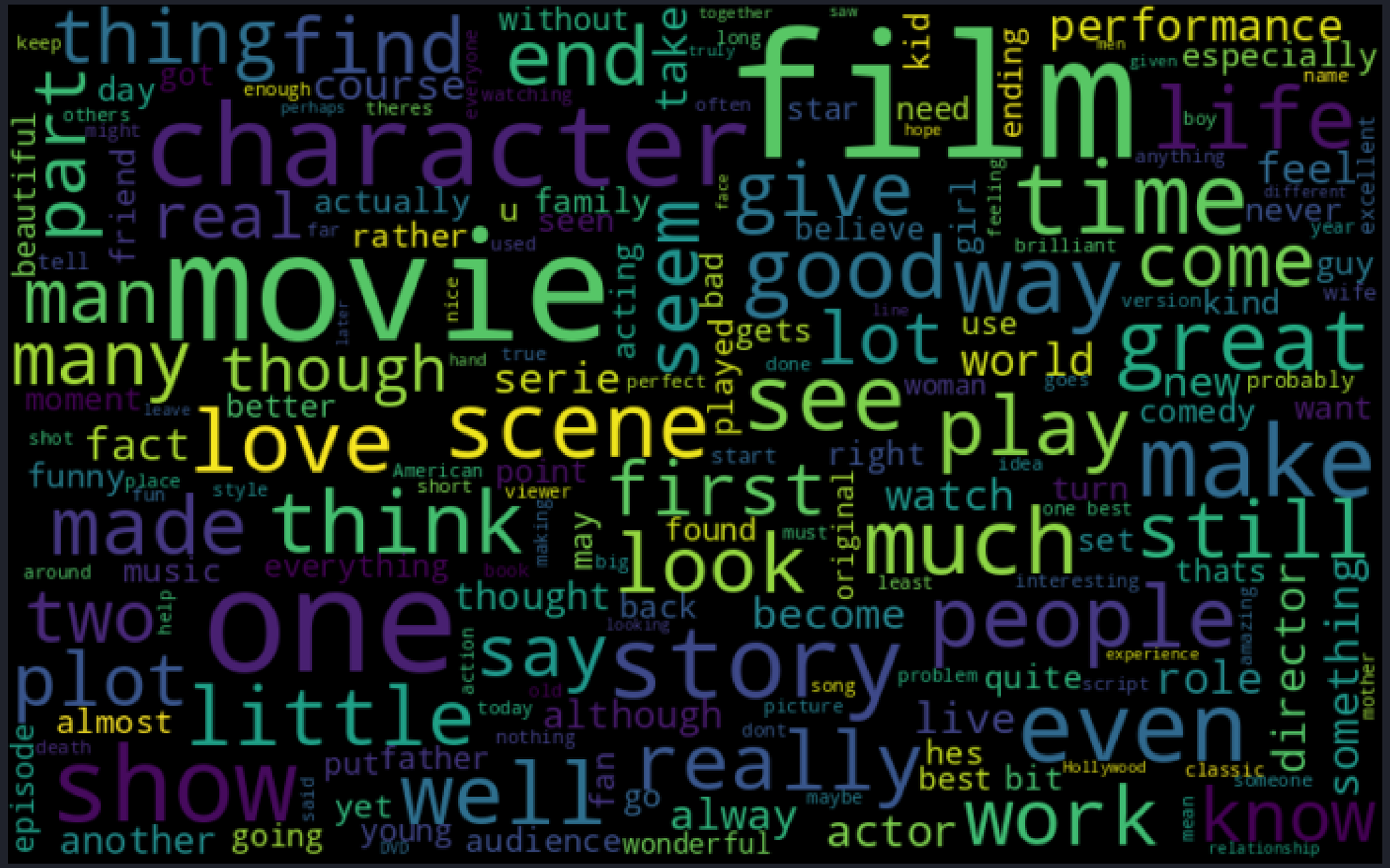
ABOUT THE DATASET

- The IMDB dataset is a binary classification dataset.
- It consists of two columns i.e., sentiment and review.
- The dataset consists of 50,000 movie reviews.
- Each review is labeled as either positive or negative.
- 1 indicates positive review.
- 0 indicates negative review.

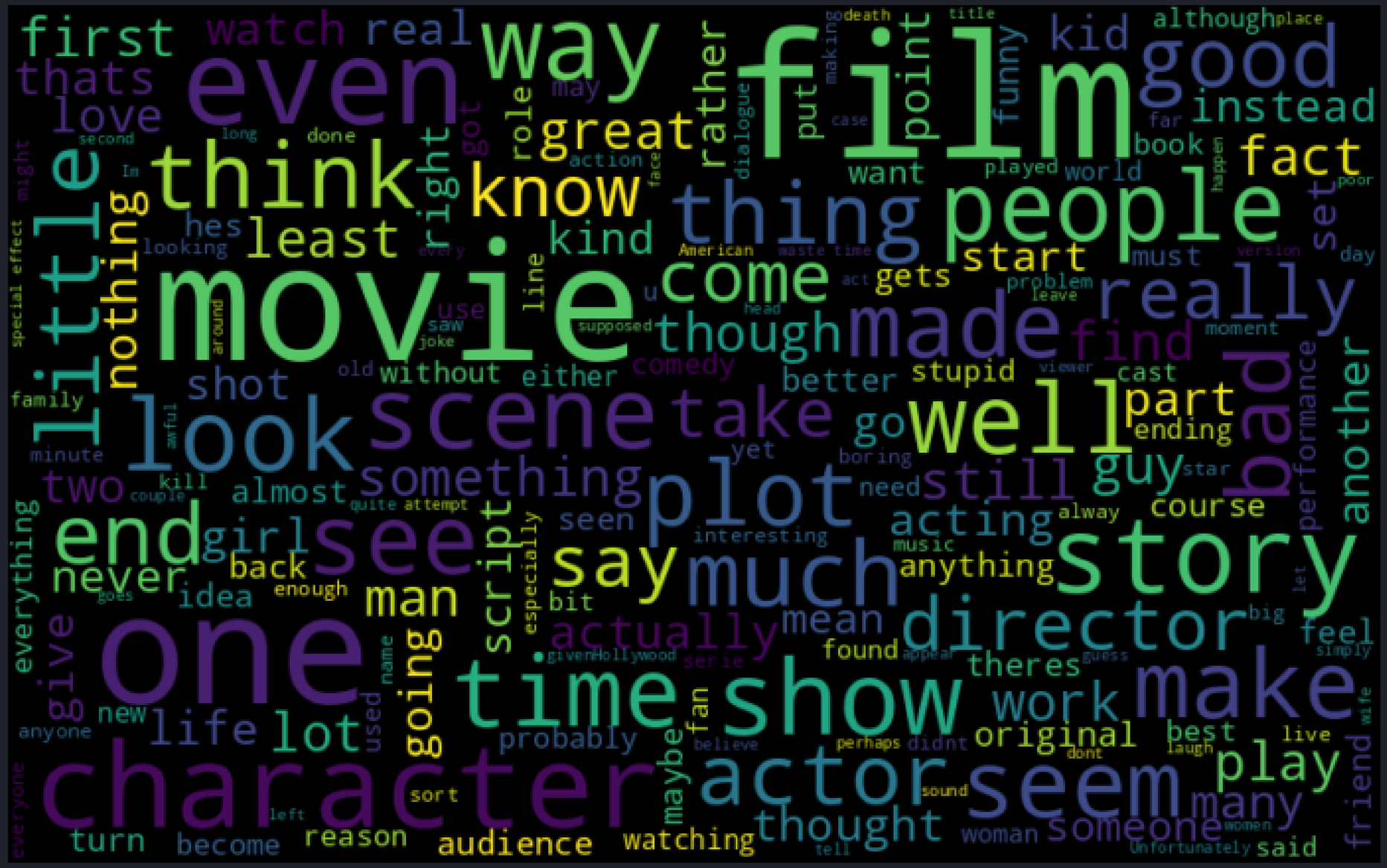
DATA PRE-PROCESSING

- Tokenization of words
- Removing HTML strips
- Removing square brackets
- Removing noisy data
- Removing special characters
- Lemmatization
- Removing stop words

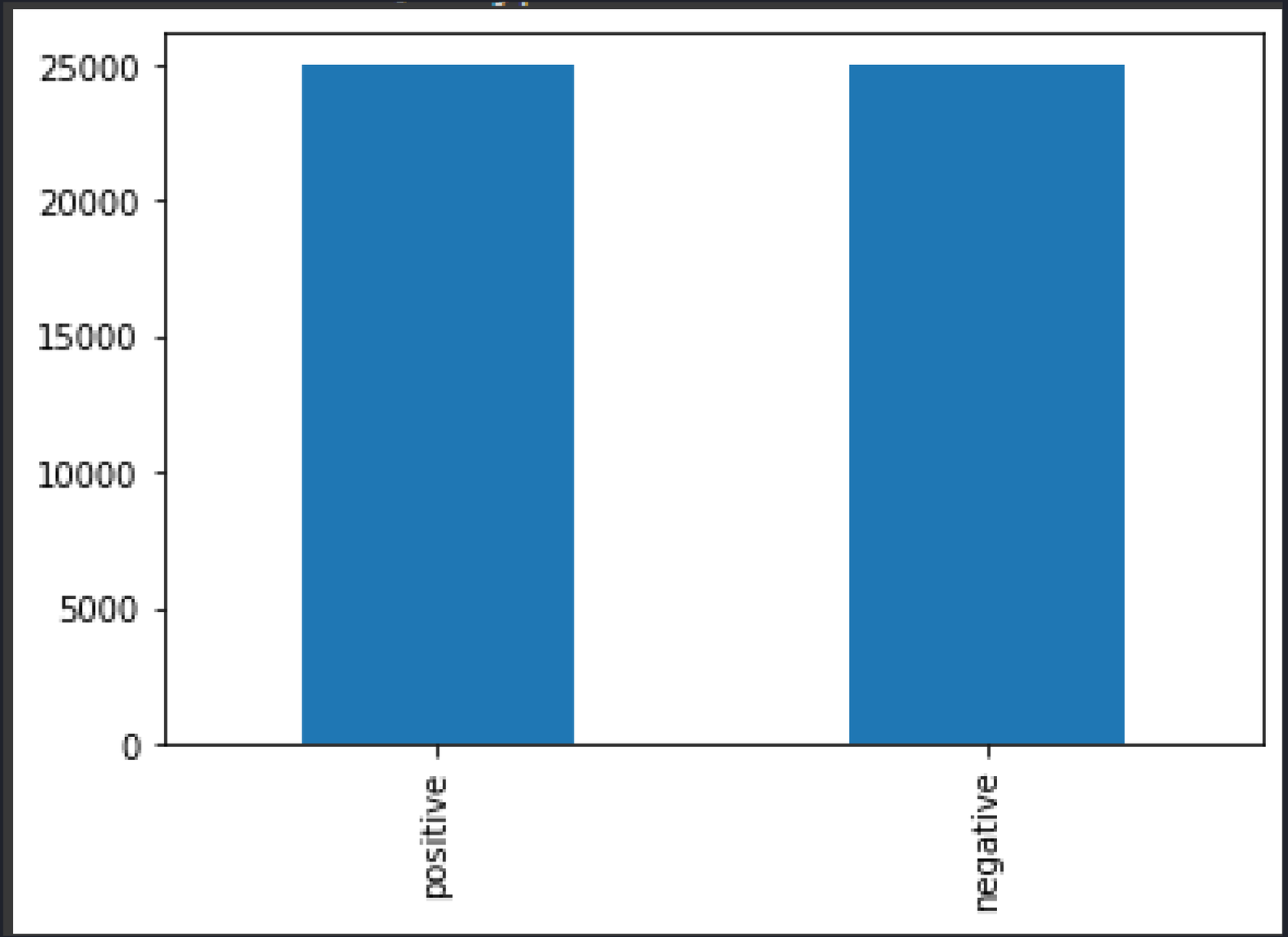
POSITIVE WORDS



NEGATIVE WORDS



VISUALIZATION OF THE TARGET VARIABLE



TFIDF VECTOR

Algorithm	Accuracy
Logistic Regression	64.56%
KNN	60.33%
SVM	64.72
Bagging Classifier	61.41%
Decision Tree	59.62%

Algorithm	Accuracy
Random Forest	62.40%
Ada Boost	64.22%
Gradient Boost	64.41
XG Boost	64.53%

80-20 train-test split

TFIDF VECTOR

Algorithm	Accuracy
Logistic Regression	64.28%
KNN	59.84%
SVM	64.30%
Bagging Classifier	60.86%
Decision Tree	59.41%

Algorithm	Accuracy
Random Forest	61.87%
Ada Boost	64.05%
Gradient Boost	64.16%
XG Boost	64.23%

70-30 train-test split

WORD2VEC VECTOR

Algorithm	Accuracy
Logistic Regression	86.10%
KNN	79.40%
SVM	86.37%
Bagging Classifier	79.92%
Decision Tree	72.09%

Algorithm	Accuracy
Random Forest	83.16%
Ada Boost	81.42%
Gradient Boost	83.67%
XG Boost	83.57%

80-20 train-test split

WORD2VEC VECTOR

Algorithm	Accuracy
Logistic Regression	85.87%
KNN	79.46%
SVM	86.32%
Bagging Classifier	79.89%
Decision Tree	72.66%

Algorithm	Accuracy
Random Forest	83.17%
Ada Boost	81.81%
Gradient Boost	83.54%
XG Boost	83.70%

70-30 train-test split

COMBINING TFIDF VECTOR WORD2VEC VECTOR

Algorithm	Accuracy
Logistic Regression	86.14%
KNN	78.70%
SVM	86.34%
Bagging Classifier	79.26%
Decision Tree	72.56%

Algorithm	Accuracy
Random Forest	83.16%
Ada Boost	81.67%
Gradient Boost	84.18%
XG Boost	83.90%

80-20 train-test split

COMBINING TFIDF VECTOR WORD2VEC VECTOR

Algorithm	Accuracy
Logistic Regression	85.93%
KNN	78.91%
SVM	86.51%
Bagging Classifier	79.42%
Decision Tree	72.72%

Algorithm	Accuracy
Random Forest	83.28%
Ada Boost	82.11%
Gradient Boost	83.80%
XG Boost	83.66%

70-30 train-test split

COMPARISION OF SVM ALGORITHM

	Train-Test split	Accuracy	Train-Test split	Accuracy
tfidf	80-20	64.72	70-30	64.30%
word2vec	80-20	86.37%	70-30	86.32%
Combining tfidf & word2vec	80-20	86.34%	70-30	86.51%

Conclusion

- The best classifier for the following dataset is *Support Vector Machine Classifier*.

- Among all the combinations of *tfidf* and *word2vec* *SVM* yielded accuracy of *86.51%* under *70:30* train-test split.

Real World Applications

1. Customer feedback analysis.
2. Analyze customer reviews and feedback on their products or services.
3. Market and competitor research
4. Social media monitoring
5. Brand monitoring and reputation management

Thank You

APPENDIX....

IMPORTING STATEMENTS

IMPORTING LIBRARIES

```
import pandas as pd
import numpy as np
import re,string
from sklearn.feature_extraction.text import TfidfVectorizer
from gensim.models import Word2Vec
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from nltk.tokenize.toktok import ToktokTokenizer
from nltk.stem import LancasterStemmer,WordNetLemmatizer
from nltk.corpus import stopwords
import gensim.downloader as api
from nltk.tokenize import word_tokenize,sent_tokenize
import matplotlib.pyplot as plt
from wordcloud import WordCloud
from bs4 import BeautifulSoup
```

DATA LOADING

```
data=pd.read_csv("/content/drive/MyDrive/IMDB/IMDB Dataset.csv",header=0)
data
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive
...	...	...
49995	I thought this movie did a down right good job...	positive
49996	Bad plot, bad dialogue, bad acting, idiotic di...	negative
49997	I am a Catholic taught in parochial elementary...	negative
49998	I'm going to have to disagree with the previou...	negative
49999	No one expects the Star Trek movies to be high...	negative

50000 rows × 2 columns

PRE - PROCESSING

TOKENIZATION

```
#Tokenization of text
tokenizer=ToktokTokenizer()
#Setting English stopwords
stopword_list=nltk.corpus.stopwords.words('english')
```

REMOVING THE HTML STRIPS

```
#Removing the html strips
def strip_html(text):
    soup = BeautifulSoup(text, "html.parser")
    return soup.get_text()
#Removing the square brackets
def remove_between_square_brackets(text):
    return re.sub('\[[^]]*\]', '', text)
#Removing the noisy text
def denoise_text(text):
    text = strip_html(text)
    text = remove_between_square_brackets(text)
    return text
#Apply function on review column
data['review']=data['review'].apply(denoise_text)
```

REMOVING SPECIAL CHARACTERS

```
#Define function for removing special characters
def remove_special_characters(text, remove_digits=True):
    pattern=r'[^a-zA-z0-9\s]'
    text=re.sub(pattern,'',text)
    return text
#Apply function on review column
data['review']=data['review'].apply(remove_special_characters)
```

LEMMATIZER

```
#Lemmatizer example

def lemmatize_all(sentence):
    wnl = WordNetLemmatizer()
    for word, tag in pos_tag(word_tokenize(sentence)):
        if tag.startswith("NN"):
            yield wnl.lemmatize(word, pos='n')
        elif tag.startswith('VB'):
            yield wnl.lemmatize(word, pos='v')
        elif tag.startswith('JJ'):
            yield wnl.lemmatize(word, pos='a')
        else:
            yield word

def lemmatize_text(text):
    return ' '.join(lemmatize_all(text))
```

STOPWORDS

```
#set stopwords to english
stop=set(stopwords.words('english'))
print(stop)

#removing the stopwords
def remove_stopwords(text, is_lower_case=False):
    tokens = tokenizer.tokenize(text)
    tokens = [token.strip() for token in tokens]
    if is_lower_case:
        filtered_tokens = [token for token in tokens if token not in stopword_list]
    else:
        filtered_tokens = [token for token in tokens if token.lower() not in stopword_list]
    filtered_text = ' '.join(filtered_tokens)
    return filtered_text
#Apply function on review column
data['review']=data['review'].apply(remove_stopwords)
```

TFIDF VECTORIZATION

```
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
import spacy

nlp = spacy.load("en_core_web_sm")
```

```
[ ]  X= data['review']
     y=data["sentiment"]
     X.head
```

```
print("\n\nWith TFIDFVectorizer")
vectorizer = TfidfVectorizer(max_features=10)
X = vectorizer.fit_transform(data.review)
print(vectorizer.get_feature_names_out())
print(X.toarray())
print("\n")
#print(cosine_similarity(X))
```

```
print("\n\nWith TFIDFVectorizer and removing stop words")
vectorizer = TfidfVectorizer(stop_words=list(nlp.Defaults.stop_words),max_features=10)
X = vectorizer.fit_transform(data.review)
print(vectorizer.get_feature_names_out())
print(X.toarray())
print("\n")
#print(cosine_similarity(X))
```

LOGISTIC REGRESSION

```
[ ]  # Create a logistic regression object
     lr1 = LogisticRegression()
     lr1.fit(X_train, y_train)
     y_pred = lr1.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

K NEAREST NEIGHBOR

```
[ ]  # Create a KNN
     from sklearn.neighbors import KNeighborsClassifier
     model1=KNeighborsClassifier(n_neighbors=5)
     model1.fit(X_train, y_train)
     y_pred = model1.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

SUPPORT VECTOR MACHINE

```
# Create a SVM
from sklearn import svm
cls1=svm.SVC(kernel='rbf')
cls1.fit(X_train, y_train)
y_pred = cls1.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

BAGGING

```
# Create a Bagging classifier
from sklearn.ensemble import BaggingClassifier
clf1 = BaggingClassifier()
clf1.fit(X_train, y_train)
y_pred = clf1.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

DECISION TREE

```
# Create a Decision Tree
from sklearn.tree import DecisionTreeClassifier
tree3 = DecisionTreeClassifier()
tree3.fit(X_train, y_train)
y_pred = tree3.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

RANDOM FOREST

```
[ ] # Create a Random Forest
from sklearn.ensemble import RandomForestClassifier
rf3 = RandomForestClassifier()
rf3.fit(X_train, y_train)
y_pred = rf3.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

ADABOOST

```
[ ]  # Create a Ada Boost
     from sklearn.ensemble import AdaBoostClassifier
     adaboost3 = AdaBoostClassifier()
     adaboost3.fit(X_train, y_train)
     y_pred = adaboost3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

GRADIENT BOOST

```
[ ]  # Create a Gradient Boost
     from sklearn.ensemble import GradientBoostingClassifier
     grad_boost3 = GradientBoostingClassifier()
     grad_boost3.fit(X_train, y_train)
     y_pred = grad_boost3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

XG BOOST

```
# Create a XG Boost
import xgboost as xgb
from xgboost import XGBClassifier
xgb_boost1 = XGBClassifier()
xgb_boost1.fit(X_train, y_train)
y_pred = xgb_boost1.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

WORD2VEC

```
# Train and download Word2Vec vectors
sentences = [review.split() for review in data['review']]
model = Word2Vec(sentences, size=100, window=5, min_count=1, workers=4)
model.save('word2vec.model')
```

```
[ ] # Convert each word to vector and represent the sentence in vector form using the word embeddings
def sentence_vector(sentence,model):
    words = sentence.split()
    word_vectors = [model.wv[word] for word in words if word in model.wv.vocab]
    if len(word_vectors) == 0:
        return np.zeros((100,))
    return np.mean(word_vectors, axis=0)
```

```
[ ] # Convert each sentence in the dataset to a vector using the Word2Vec model
word2vec_train = np.array([sentence_vector(sentence, model) for sentence in data['review']])
word2vec_test = np.array(data['sentiment'])
```

```
[ ] X_train,X_test,y_train,y_test=train_test_split(word2vec_train,word2vec_test,test_size=0.2,random_state=42)
```

Logistic Regression

```
[ ]  # Create a logistic regression object
     lr2 = LogisticRegression()
     lr2.fit(X_train, y_train)
     y_pred = lr2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

KNN

```
[ ]  # Create a KNN
     from sklearn.neighbors import KNeighborsClassifier
     model2=KNeighborsClassifier(n_neighbors=5)
     model2.fit(X_train, y_train)
     y_pred = model2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

SVM

```
# Create a SVM
from sklearn import svm
cls2=svm.SVC(kernel='rbf')
cls2.fit(X_train, y_train)
y_pred = cls2.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

```
Test accuracy: 86.37
```

Bagging Classifier

```
[ ]  # Create a Bagging classifier
     from sklearn.ensemble import BaggingClassifier
     clf2 = BaggingClassifier()
     clf2.fit(X_train, y_train)
     y_pred = clf2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Decision Tree

```
[ ]  # Create a Decision Tree
     from sklearn.tree import DecisionTreeClassifier
     tree2 = DecisionTreeClassifier()
     tree2.fit(X_train, y_train)
     y_pred = tree2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Random Forest

```
[ ]  # Create a Random Forest
     from sklearn.ensemble import RandomForestClassifier
     rf2 = RandomForestClassifier()
     rf2.fit(X_train, y_train)
     y_pred = rf2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Ada Boost

```
[ ]  # Create a Ada Boost
     from sklearn.ensemble import AdaBoostClassifier
     adaboost2 = AdaBoostClassifier()
     adaboost2.fit(X_train, y_train)
     y_pred = adaboost2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Gradient Boost

```
     # Create a Gradient Boost
     from sklearn.ensemble import GradientBoostingClassifier
     grad_boost2 = GradientBoostingClassifier()
     grad_boost2.fit(X_train, y_train)
     y_pred = grad_boost2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

XG Boost

```
[ ]  # Create a XG Boost
     import xgboost as xgb
     from xgboost import XGBClassifier
     xgb_boost2 = xgb.XGBClassifier()
     xgb_boost2.fit(X_train, y_train)
     y_pred = xgb_boost2.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Combining tfidf and word2vec

```
def combine_vectors(doc):
    tfidf_vec = vectorizer.transform([doc])
    w2v_vec = sentence_vector(doc, model)
    combined_vec = np.concatenate([np.squeeze(tfidf_vec.toarray()), w2v_vec])
    return combined_vec
```

```
[ ] combined_train = np.array([combine_vectors(doc) for doc in data['review']])
    combined_test = np.array(data['sentiment'])
```

```
[ ] # Split the dataset into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(combined_train, combined_test, test_size=0.2, random_state=42)
```

Logistic Regression

```
[ ]  # Create a logistic regression object
     lr3 = LogisticRegression()
     lr3.fit(X_train, y_train)
     y_pred = lr3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

KNN

```
# Create a KNN
from sklearn.neighbors import KNeighborsClassifier
model3=KNeighborsClassifier(n_neighbors=5)
model3.fit(X_train, y_train)
y_pred = model3.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

SVM

```
[ ]  # Create a SVM
     from sklearn import svm
     cls3=svm.SVC(kernel='rbf')
     cls3.fit(X_train, y_train)
     y_pred = cls3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Bagging Classifier

```
[ ]  # Create a Bagging classifier
     from sklearn.ensemble import BaggingClassifier
     clf3 = BaggingClassifier()
     clf3.fit(X_train, y_train)
     y_pred = clf3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Decision Tree

```
[ ]  # Create a Decision Tree
     from sklearn.tree import DecisionTreeClassifier
     tree3 = DecisionTreeClassifier()
     tree3.fit(X_train, y_train)
     y_pred = tree3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Random Forest

```
[ ]  # Create a Random Forest
     from sklearn.ensemble import RandomForestClassifier
     rf3 = RandomForestClassifier()
     rf3.fit(X_train, y_train)
     y_pred = rf3.predict(X_test)
     y_pred
     # Calculate the accuracy of the model
     acc = accuracy_score(y_test, y_pred)
     print('Test accuracy:', acc*100)
```

Ada Boost

```
# Create a Ada Boost
from sklearn.ensemble import AdaBoostClassifier
adaboost3 = AdaBoostClassifier()
adaboost3.fit(X_train, y_train)
y_pred = adaboost3.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```

Gradient Boost

```
[ ] # Create a Gradient Boost
    from sklearn.ensemble import GradientBoostingClassifier
    grad_boost3 = GradientBoostingClassifier()
    grad_boost3.fit(X_train, y_train)
    y_pred = grad_boost3.predict(X_test)
    y_pred
    # Calculate the accuracy of the model
    acc = accuracy_score(y_test, y_pred)
    print('Test accuracy:', acc*100)
```

XG Boost

```
# Create a XG Boost
import xgboost as xgb
from xgboost import XGBClassifier
xgb_boost3 = xgb.XGBClassifier()
xgb_boost3.fit(X_train, y_train)
y_pred = xgb_boost3.predict(X_test)
y_pred
# Calculate the accuracy of the model
acc = accuracy_score(y_test, y_pred)
print('Test accuracy:', acc*100)
```