NLP-Based Sentiment Analysis using Logistic Regression

In this project, I implemented a sentiment analysis model using Logistic Regression. Text data was cleaned, tokenized, and vectorized using TF-IDF. The Logistic Regression model was trained to classify sentiments (e.g., positive/negative). The model achieved good accuracy, serving as a strong baseline for future experimentation with more advanced algorithms.

Random Forest

Accuracy: ~56.7%

Strengths: Better handling of complex data. Improved sentiment prediction compared to SVM and LSTM.

Logistic Regression

Accuracy: ~82.67%

Best-performing model. Effectively classifies positive, negative, and neutral sentiments

LSTM

Accuracy: ~34%

Challenges: Struggled with neutral sentiments. Model requires tuning.

SVM

Accuracy: ~39.33%

Observations: Moderate performance. Difficulties with complex language structures.

Below is the code of 4 models.

```
import numpy as np

import pandas as pd

import re

import nltk
```

```
nltk.download('stopwords')
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
#printing the stopwords in english
print(stopwords.words('english'))
# Install the chardet module
!pip install chardet

# Import the chardet module
import chardet

# Detect the encoding of the CSV file
with open('projectML.csv', 'rb') as f:
    encoding = chardet.detect(f.read())['encoding']

# Read the CSV file with the detected encoding
df = pd.read_csv('projectML.csv', encoding=encoding)
```

```
# Verify that the data is loaded correctly

df.head()

#checking the number of rows and columns

df.shape

df

#counting the number of missing values in the dataset

df.isnull().sum()

df


#checking the distribution of target column where 0=Positive, 1=Negative,
2=Neutral

df['Sentiment'].value_counts()

# Import necessary modules

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt


# Create a copy of the df DataFrame

df1 = df.copy()


# Convert Timestamp to datetime

df1['Timestamp'] = pd.to_datetime(df1['Timestamp'])


# Plot 1: Sentiment Distribution
```

```
sns.countplot(x='Sentiment', data=df1)
plt.title('Sentiment Distributin')
plt.xlabel('Sentiment')
plt.ylabel('Count')
plt.show()
port_stem=PorterStemmer()
def stemming(content):
  stemmed_content= re.sub('[^a-zA-Z]',' ',content)
  stemmed_content=stemmed_content.lower()
  stemmed_content=stemmed_content.split()
  stemmed_content=[port_stem.stem(word) for word in stemmed_content if not word in stopwords.words('english')]
  stemmed_content=' '.join(stemmed_content)

  return stemmed_content
df['stemmed_content']=df['Text'].apply(stemming)
#Showing the stemmed content
df.head()
print(df['stemmed_content'])
print(df['Sentiment'])
#separating the data and label
X=df['stemmed_content'].values
Y=df['Sentiment'].values
X_train,X_test,Y_train,Y_test = train_test_split(X,Y, test_size=0.2, stratify=Y,random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
#Convertinng the textual data to numerical data

vectorizer= TfidfVectorizer()

X_train= vectorizer.fit_transform(X_train)
X_test= vectorizer.transform(X_test)
print(X_train)
print(X_test)
model=LogisticRegression(max_iter=1000)
model.fit(X_train,Y_train)
#accuracy score on the training data
X_train_prediction= model.predict(X_train)
training_data_accuracy=accuracy_score(Y_train,X_train_prediction)
print('Accuracy score on the training data:', training_data_accuracy)
#accuracy score on the test data
X_test_prediction= model.predict(X_test)
test_data_accuracy=accuracy_score(Y_test,X_test_prediction)
print('Accuracy score on the test data:', test_data_accuracy)
from sklearn.metrics import confusion_matrix

# Generate confusion matrix
conf_matrix = confusion_matrix(Y_test, X_test_prediction)
```

```
# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()

from sklearn.metrics import classification_report

# Generate classification report
class_report = classification_report(Y_test, X_test_prediction)

# Print classification report
print("Classification Report:\n", class_report)

# Import necessary modules
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize
import matplotlib.pyplot as plt

# Binarize the target variable
Y_test_binarized = label_binarize(Y_test, classes=['0', '1'])
```

```
# Convert X_test_prediction to float
X_test_prediction = X_test_prediction.astype(float)


# Compute ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(Y_test_binarized[:, 1], X_test_prediction)
roc_auc = auc(fpr, tpr)


# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.show()
sample_predictions = pd.DataFrame({ 'True_Sentiment': Y_test, 'Predicted_Sentiment': X_test_prediction})
print("\nSample Predictions:")
print(sample_predictions.head())
import pickle
filename= 'trained_model.sav'
```

```
pickle.dump(model,open(filename, 'wb'))

#Loading the saved model

loaded_model= pickle.load(open('/content/trained_model.sav', 'rb'))

X_new=X_test[10]

print(Y_test[10])


prediction=model.predict(X_new)

print(prediction)


if prediction[0] == '0':

  print("Positive")

elif prediction[0] == '1':

  print("Negative")

else:


  print("Neutral")
```

Built and compared three different machine learning models — LSTM, Support Vector Machine (SVM), and Random Forest — to classify text data based on sentiment or category. The project focused on preprocessing text data, converting it into suitable formats (sequences and TF-IDF vectors), and evaluating model performance through metrics like accuracy and classification reports.

- Compared models using classification reports and accuracy scores.
- Observed trade-offs between training time (LSTM slower but more accurate) and model complexity.

```
# DataFrame
```

```
import pandas as pd

# Matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

from matplotlib.ticker import MaxNLocator
import matplotlib.gridspec as gridspec
import matplotlib.patches as mpatches

# Scikit-learn
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report,
accuracy_score
from sklearn.manifold import TSNE
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.decomposition import LatentDirichletAllocation, NMF
from sklearn.metrics import f1_score, accuracy_score

# Keras
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
```

```
from keras.layers import Activation, Dense, Dropout, Embedding, Flatten, Conv1D, MaxPooling1D, LSTM

from keras import utils

from keras.callbacks import ReduceLROnPlateau, EarlyStopping


# NLTK

import nltk

from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer

from nltk.tokenize import word_tokenize

from nltk.probability import FreqDist

from nltk.corpus import wordnet

from nltk.stem import WordNetLemmatizer

nltk.download('stopwords')

nltk.download('punkt')

nltk.download('wordnet')


# Word2Vec

import gensim

from gensim.models import Word2Vec


# Utility

import string

import re

import numpy as np
```

```
import os
from collections import Counter
import logging
import time
import pickle
import itertools
import random
import datetime

# WordCloud
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
from collections import Counter, defaultdict

# Warnings
import warnings
warnings.filterwarnings('ignore')

# Set log
logging.basicConfig(format='%(asctime)s : %(levelname)s : %(message)s',
level=logging.INFO)

!pip install chardet

# Import the chardet module
```

```
import chardet

with open('projectML.csv', 'rb') as f:
    encoding = chardet.detect(f.read())['encoding']
# Load the dataset with the detected encoding
df = pd.read_csv('projectML.csv', encoding=encoding)

# View the first few rows
df.head()

# Check for null values
df.isnull().sum()

# Drop rows with null values
df.dropna(inplace=True)

# Check value counts for each category
df['category'].value_counts()

# Text Preprocessing
stop_words = set(stopwords.words("english"))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
```

```
    text = text.lower()
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words and len(word) > 2]
    return " ".join(tokens)

df['cleaned_text'] = df['text'].apply(preprocess_text)

# Tokenization and Padding
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(df['cleaned_text'])

X = tokenizer.texts_to_sequences(df['cleaned_text'])
X = pad_sequences(X)

# Label Encoding
le = LabelEncoder()
y = le.fit_transform(df['category'])

# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# LSTM Model
model_lstm = Sequential()
```

```
model_lstm.add(Embedding(input_dim=5000, output_dim=128,
input_length=X.shape[1]))
model_lstm.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model_lstm.add(Dense(128, activation='relu'))
model_lstm.add(Dropout(0.5))
model_lstm.add(Dense(len(set(y)), activation='softmax'))

model_lstm.compile(loss='sparse_categorical_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])

# Callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=3,
restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2)

# Train the model
history = model_lstm.fit(X_train, y_train,
                         validation_split=0.1,
                         epochs=10,
                         batch_size=128,
                         callbacks=[early_stopping, reduce_lr])

# Evaluate the model
loss, accuracy = model_lstm.evaluate(X_test, y_test)
```

```
print(f"LSTM Accuracy: {accuracy}")


# Predict
y_pred = model_lstm.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)


# Classification Report
print(classification_report(y_test, y_pred_classes))
```

```
# SVM Model
from sklearn.pipeline import Pipeline
from sklearn.svm import SVC


# TF-IDF vectorization
tfidf = TfidfVectorizer(max_features=5000)


# Build SVM pipeline
svm_pipeline = Pipeline([
    ('tfidf', tfidf),
    ('svm', SVC(kernel='linear'))
])


# Split again for raw text input
X_train_raw, X_test_raw, y_train_svm, y_test_svm =
train_test_split(df['cleaned_text'], y, test_size=0.2, random_state=42)
```

```python
# Train SVM
svm_pipeline.fit(X_train_raw, y_train_svm)

# Predict and evaluate
y_pred_svm = svm_pipeline.predict(X_test_raw)
print("SVM Classification Report:")
print(classification_report(y_test_svm, y_pred_svm))
```

```python
# Random Forest
from sklearn.ensemble import RandomForestClassifier

# TF-IDF for Random Forest
tfidf_rf = TfidfVectorizer(max_features=5000)
X_tfidf_rf = tfidf_rf.fit_transform(df['cleaned_text'])

# Train-test split
X_train_rf, X_test_rf, y_train_rf, y_test_rf = train_test_split(X_tfidf_rf, y,
test_size=0.2, random_state=42)

# Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train_rf, y_train_rf)

# Predict and evaluate
y_pred_rf = rf_model.predict(X_test_rf)
print("Random Forest Classification Report:")
```

```
print(classification_report(y_test_rf, y_pred_rf))
```