Project3-200574904_Satish_Kumar

October 8, 2024

[1]:
```
import pandas as pd

# Load the IMDB dataset
file_path = 'IMDB.csv'
df = pd.read_csv(file_path)

# Displaying first few rows of the dataset
df.head()
```

[1]:

	review	sentiment
0	I thought this was a wonderful way to spend ti…	positive
1	Probably my all-time favorite movie, a story o…	positive
2	I sure would like to see a resurrection of a u…	positive
3	This show was an amazing, fresh & innovative i…	negative
4	Encouraged by the positive comments about this…	negative

[2]:
```
df.shape
```

[2]:
```
(25000, 2)
```

0.1 Preprocess Text Data(Remove punctuation, Perform Tokenization, Remove stopwords and Lemmatize/Stem)

[3]:
```
import string
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer

# Download necessary resources for stopwords and lemmatization
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')

# Initialize stopwords and lemmatizer
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
# Text Preprocessing Functions
def preprocess_text(text):
    # Remove punctuation
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Tokenize text
    tokens = word_tokenize(text.lower())
    # Remove stopwords
    tokens = [word for word in tokens if word not in stop_words]
    # Lemmatize the tokens
    tokens = [lemmatizer.lemmatize(word) for word in tokens]
    return ' '.join(tokens)

# Apply preprocessing to the review column
df['cleaned_review'] = df['review'].apply(preprocess_text)

# Display the first few rows after preprocessing
df[['review', 'cleaned_review']].head()
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/satishmatani/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/satishmatani/nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/satishmatani/nltk_data…
```

[3]:

	review	\
0	I thought this was a wonderful way to spend ti…	
1	Probably my all-time favorite movie, a story o…	
2	I sure would like to see a resurrection of a u…	
3	This show was an amazing, fresh & innovative i…	
4	Encouraged by the positive comments about this…	

	cleaned_review
0	thought wonderful way spend time hot summer we…
1	probably alltime favorite movie story selfless…
2	sure would like see resurrection dated seahunt…
3	show amazing fresh innovative idea 70 first ai…
4	encouraged positive comment film looking forwa…

This step ensured that the model would now focus on the meaningful content of the reviews without distractions from common words or formatting issues as we've removed them from our dataset.

0.2 Perform TFIDF Vectorization

[5]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize TFIDF Vectorizer
tfidf_vect = TfidfVectorizer(max_features=5000)

# Apply TFIDF Vectorization on the cleaned reviews
X_tfidf = tfidf_vect.fit_transform(df['cleaned_review'])

# Display the shape of the resulting TFIDF matrix
X_tfidf.shape, tfidf_vect.get_feature_names_out()
```

[5]:
```
((25000, 5000),
 array(['10', '100', '1000', …, 'zero', 'zombie', 'zone'], dtype=object))
```

This method highlighted important words in the reviews based on their frequency across the dataset, which will help the models to capture the sentiment more effectively.

0.3 Exploring parameter settings using GridSearchCV on Random Forest & Gradient Boosting/Xgboost classifier

[9]:
```
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score

# Encode the sentiment labels
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(df['sentiment'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_tfidf, y, test_size=0.2,␣
 ↪random_state=42)

# Define parameter grids for both Random Forest and Gradient Boosting
rf_params = {'n_estimators': [50, 100], 'max_depth': [10, 20, None]}
gb_params = {'n_estimators': [50, 100], 'max_depth': [3, 5, 7], 'learning_rate':
 ↪ [0.1, 0.05]}

# Initialize Random Forest and Gradient Boosting models
rf_model = RandomForestClassifier()
gb_model = GradientBoostingClassifier()

# Initialize GridSearchCV for both models
rf_grid = GridSearchCV(rf_model, rf_params, cv=3, n_jobs=-1, verbose=1)
gb_grid = GridSearchCV(gb_model, gb_params, cv=3, n_jobs=-1, verbose=1)
```

```
# Fit both models using GridSearchCV
rf_grid.fit(X_train, y_train)
gb_grid.fit(X_train, y_train)

# Display best parameters for both models
print("Random Forest Best Params:", rf_grid.best_params_)
print("Gradient Boosting Best Params:", gb_grid.best_params_)
```

```
Fitting 3 folds for each of 6 candidates, totalling 18 fits
Fitting 3 folds for each of 12 candidates, totalling 36 fits
Random Forest Best Params: {'max_depth': None, 'n_estimators': 100}
Gradient Boosting Best Params: {'learning_rate': 0.1, 'max_depth': 7,
'n_estimators': 100}
```

Both models were trained on the training dataset after splitting it from the original data. Hyperparameters were optimized using GridSearchCV, which allowed me to systematically evaluate different combinations of parameters to find the best settings for each model.

0.4 Perform Final evaluation of models on the best parameter settings using the evaluation metrics

[15]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix

# Final evaluation of the models

# Evaluate Random Forest
rf_best = rf_grid.best_estimator_
y_pred_rf = rf_best.predict(X_test)

# Evaluate Gradient Boosting
gb_best = gb_grid.best_estimator_
y_pred_gb = gb_best.predict(X_test)

# Calculate and display evaluation metrics
print("Random Forest Confusion Matrix: \n\n",confusion_matrix(y_test,␣
 ↪y_pred_rf))
print("\nRandom Forest Classification Report:␣
 ↪\n\n",classification_report(y_test, y_pred_rf))

print("Gradient Boosting Confusion Matrix: \n\n",confusion_matrix(y_test,␣
 ↪y_pred_gb))
print("\nGradient Boosting Classification Report:␣
 ↪\n\n",classification_report(y_test, y_pred_gb))
```

```
Random Forest Confusion Matrix:
```

```
 [[2135  392]
 [ 379 2094]]

Random Forest Classification Report:

              precision    recall  f1-score   support

           0       0.85      0.84      0.85      2527
           1       0.84      0.85      0.84      2473

    accuracy                           0.85      5000
   macro avg       0.85      0.85      0.85      5000
weighted avg       0.85      0.85      0.85      5000

Gradient Boosting Confusion Matrix:

 [[2010  517]
 [ 330 2143]]

Gradient Boosting Classification Report:

              precision    recall  f1-score   support

           0       0.86      0.80      0.83      2527
           1       0.81      0.87      0.83      2473

    accuracy                           0.83      5000
   macro avg       0.83      0.83      0.83      5000
weighted avg       0.83      0.83      0.83      5000
```

0.5 Include a Conclusion section, compare the model performance, report the best performing model, and include key insights obtained from the analysis

0.6 Conclusion:

In this project, I analyzed IMDB movie reviews to predict sentiment using Random Forest and Gradient Boosting models using GridSearchCV. After preprocessing the data and applying TFIDF vectorization, I compared the performance of both models, whose details are as below:

0.7 Comparing Model Performance:

Random Forest achieved an accuracy of 85% with a balanced precision and recall for both classes (0.85 and 0.84).

Gradient Boosting had an accuracy of 83%, with slightly lower precision (0.86 for class 0, 0.81 for class 1) and recall (0.80 for class 0, 0.87 for class 1).

0.8 Best performing model:

The Random Forest model outperformed Gradient Boosting, making it the better choice for this sentiment analysis task.

0.9 Key insights:

Both models effectively classified movie sentiments, but Random Forest was more accurate.

The confusion matrices revealed some misclassifications, with Random Forest misclassifying 392 positive reviews as negative, while Gradient Boosting misclassified 517 negative reviews as positive.

Overall, this project shows that machine learning techniques like Random Forest can be effective for sentiment analysis.

[ ]: