NLP Project Final Report – Part A: IMDb Movie Review Sentiment Analysis

---

1. Data Loading and Initial Exploration

1.1. Importing Libraries

We started by importing the necessary libraries:

- pandas & numpy: For working with structured data and performing numerical computations.
- matplotlib & seaborn: For visualizing the data through charts and plots.
- re & string: Useful for cleaning text, like removing punctuation.
- nltk: A toolkit for text processing, used for:
    - Removing stopwords (e.g., "the", "is")
    - Splitting sentences into words using tokenization
    - Lemmatizing words (e.g., "running" becomes "run")
- scikit-learn (sklearn): Used for:
    - Converting text into numbers (TF-IDF)
    - Splitting the dataset for training and testing
    - Evaluating model performance with metrics like accuracy and ROC-AUC
    - Implementing models such as Logistic Regression, Naïve Bayes, SVM, and Random Forest

1.2. Downloading NLTK Resources

We downloaded essential components like:

- Stopwords: Common words filtered out during preprocessing.
- Punkt: Helps in breaking text into tokens.
- WordNet: Needed for lemmatization.

1.3. Exploring the Dataset

Before diving into processing, we did a quick check using:

- df.head() – to see the first few rows of the dataset.
- df.info() – to get details about the number of rows, columns, and any missing data.
- df['sentiment'].value_counts() – to count how many reviews are labeled as positive or negative.

---

2. Data Preprocessing

2.1. Preparing for Text Cleaning

We set up:

- A list of English stopwords to remove unimportant words.
- A lemmatizer to reduce words to their base forms.

Copy code

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

2.2. Text Preprocessing Function

We defined a function to clean and prepare text data:

Copy code

```
def preprocess_text(text):
    text = text.lower()
    text = text.translate(str.maketrans('', '', string.punctuation))
    tokens = word_tokenize(text)
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]
    return " ".join(tokens)
```

This function:

- Converts all text to lowercase
- Removes punctuation
- Breaks sentences into words (tokenization)
- Removes stopwords and lemmatizes each word

2.3. Applying the Preprocessing

We applied the function to the review column and saved the result in a new column called cleaned_text:

Copy code

```
df['cleaned_text'] = df['review'].apply(preprocess_text)
```

---

3. Feature Engineering

3.1. Converting Text into Numerical Form

Using TF-IDF Vectorizer, we turned text into numbers:

Copy code

```
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(df['cleaned_text'])
```

- It captures the importance of each word in a review.
- We limit it to the top 5,000 most important words to keep things efficient.

3.2. Encoding Target Labels

We converted sentiment labels into numeric form:

Copy code

```
y = df['sentiment'].map({'positive': 1, 'negative': 0})
```

This helps the models treat the task as a binary classification problem.

3.3. Splitting the Data

We split the dataset for training and testing:

Copy code

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

- 80% of the data is used for training
- 20% is reserved for testing
- random_state=42 ensures consistent results every time

---

4. Model Training and Evaluation

4.1. Creating a Training & Evaluation Function

We defined a function to train the model and assess its performance:

Copy code

```
def train_and_evaluate_model(model, model_name):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    y_pred_prob = model.predict_proba(X_test)[:, 1]
```

- Trains the model on the training set
- Predicts outcomes for the test set
- Computes probability scores for ROC-AUC

4.2. Evaluating Performance

We used the following to assess model accuracy:

- Accuracy: Overall correct predictions
- Classification Report: Includes precision, recall, and F1-score
- ROC Curve: Helps visualize the balance between true positives and false positives

- Confusion Matrix: Displayed as a heatmap for easier understanding

---

5. Models Used and Their Results

1. Logistic Regression

- Accuracy: 0.8871
- Performance Summary:

Sentiment	Precision	Recall	F1-Score	Support
Negative	0.90	0.87	0.88	4961
Positive	0.88	0.90	0.89	5039

---

2. Multinomial Naïve Bayes

- Accuracy: 0.8546
- Performance Summary:

Sentiment	Precision	Recall	F1-Score	Support
Negative	0.86	0.85	0.85	4961
Positive	0.85	0.86	0.86	5039

---

3. Random Forest Classifier

- Accuracy: 0.8513
- Settings:
    - n_estimators=100: Uses 100 decision trees.
    - random_state=42: For consistent results.
- Performance Summary:

Sentiment	Precision	Recall	F1-Score	Support
Negative	0.84	0.86	0.85	4961

Sentiment	Precision	Recall	F1-Score	Support
Positive	0.86	0.84	0.85	5039

Conclusion

- Logistic Regression gave the best overall results.
- Naïve Bayes performed well and was computationally efficient.
- Random Forest had slightly lower accuracy but balanced performance across classes.

VIDEO EXPLANATION: -

https://drive.google.com/file/d/1HL3tqjvSJ78s00pTuHM30WRuwxqgcQgA/view?usp=sharing