|  |  |
| --- | --- |
| Internship Project Title | Automate detection of different emotions from paragraphs and predict overall emotion |
| Name of the Company | TCSion |
| Name of the Industry Mentor | Ms. Himdweep Walia |
| Name of the Institute | Amity Online University |
| Submitted By | Ravi Prakash Mishra |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Start Date | End Date | Total Effort (hrs.) | Project Environment | Tools used |
| 08/04/2024 | 07/07/2024 | 210 | Python, Jupyter Notebook | Pandas, NumPy, Scikit-learn, TensorFlow, Keras, Seaborn, Matplotlib |

**TABLE OF CONTENT**

* Acknowledgements
* Objective
* Introduction / Description of Internship
* Internship Activities
* Approach / Methodology
* Assumptions
* Exceptions / Exclusions
* Charts, Table, Diagrams
* Algorithms
* Challenges & Opportunities
* Risk Vs Reward
* Reflections on the Internship
* Recommendations
* Outcome / Conclusion
* Enhancement Scope
* Link to code and executable file
* Research questions and responses

**Special Mention:** Attached is the complete end to end python program to save the best model. Simply open the folder in visual studio code install requirements.txt and run main.py and keep on closing the chart or diagrams whenever visible, at the end of the program will save the best performed model. To avoid data leakage the original dataset has been divided into three parts train.csv, test.csv and val.csv.

**1. Acknowledgements**

I would like to express my gratitude to my industry mentor, Ms. Himdweep Walia, for their guidance and support throughout the internship. I also thank TCSion and Amity Online University for providing me with this opportunity. Special thanks to my colleagues and friends who provided valuable feedback and encouragement.

**2. Objective**

The primary objective of this internship project is to develop and evaluate machine learning and deep learning models for emotional text classification. The goal is to achieve high performance metrics, such as accuracy, precision, recall, and F1-score, to ensure the models' reliability and effectiveness in classifying text data.

**3. Introduction / Description of Internship**

This internship focuses on building and comparing various text classification models using both machine learning (Logistic Regression, Naive Bayes, SVM) and deep learning (LSTM) techniques. The project involves preprocessing textual data, performing feature extraction, applying different models, and evaluating their performance. The significance of text classification spans multiple domains including spam detection, sentiment analysis, and topic categorization.

**4. Internship Activities**

4.1 Data Collection

Data collection is a critical step in the project. The data was sourced from various repositories to ensure diversity and comprehensiveness. The datasets used include:

* Training Data: Consisting of labeled text samples.
* Testing Data: Used for evaluating model performance.
* Validation Data: Used during model training for hyperparameter tuning.

import pandas as pd

import re

from sklearn.preprocessing import LabelEncoder

from tensorflow.keras.utils import to\_categorical

def load\_data(file\_path):

    return pd.read\_csv(file\_path)

4.2 Data Preprocessing

Text data preprocessing involved:

* Lowercasing: Converting all text to lowercase to maintain uniformity.
* Punctuation Removal: Removing punctuation marks to clean the text.
* Tokenization: Splitting text into individual words or tokens.
* Stop Words Removal: Removing common words that do not contribute much to the meaning (e.g., 'the', 'and').

def clean\_text(text):

    text = text.lower()

    text = re.sub(r'[^\w\s]', '', text)

    return text

def preprocess\_data(data):

    data['text'] = data['text'].apply(clean\_text)

    return data

def encode\_labels(labels):

    encoder = LabelEncoder()

    labels\_encoded = encoder.fit\_transform(labels)

    labels\_one\_hot = to\_categorical(labels\_encoded, num\_classes=6)

    return labels\_one\_hot, encoder

def split\_data(data):

    X, y = data['text'], data['label']

    return X, y

4.3 Feature Extraction

Feature extraction transforms raw text into numerical features that machine learning algorithms can process. The Term Frequency-Inverse Document Frequency (TF-IDF) vectorizer was used to convert text data into numerical vectors.

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from sklearn.feature\_extraction.text import TfidfVectorizer

def tokenize\_and\_pad(X\_train, X\_test, X\_val, max\_len=100):

    tokenizer = Tokenizer(num\_words=5000)

    tokenizer.fit\_on\_texts(X\_train)

    X\_train\_seq = tokenizer.texts\_to\_sequences(X\_train)

    X\_test\_seq = tokenizer.texts\_to\_sequences(X\_test)

    X\_val\_seq = tokenizer.texts\_to\_sequences(X\_val)

    X\_train\_pad = pad\_sequences(X\_train\_seq, maxlen=max\_len)

    X\_test\_pad = pad\_sequences(X\_test\_seq, maxlen=max\_len)

    X\_val\_pad = pad\_sequences(X\_val\_seq, maxlen=max\_len)

    return X\_train\_pad, X\_test\_pad, X\_val\_pad, tokenizer.word\_index

def tfidf\_vectorize(X\_train, X\_test, X\_val):

    tfidf\_vectorizer = TfidfVectorizer(max\_features=5000)

    X\_train\_tfidf = tfidf\_vectorizer.fit\_transform(X\_train)

    X\_test\_tfidf = tfidf\_vectorizer.transform(X\_test)

    X\_val\_tfidf = tfidf\_vectorizer.transform(X\_val)

    return X\_train\_tfidf, X\_test\_tfidf, X\_val\_tfidf

import matplotlib.pyplot as plt

import seaborn as sns

def plot\_label\_distribution(data, save\_path=None):

    plt.figure(figsize=(10, 6))

    sns.countplot(x='label', data=data, palette='viridis')

    plt.title('Label Distribution')

    plt.xlabel('Labels')

    plt.ylabel('Count')

    if save\_path:

        plt.savefig(save\_path)

    plt.show()

def plot\_text\_length\_distribution(data, save\_path=None):

    data['text\_length'] = data['text'].apply(len)

    plt.figure(figsize=(10, 6))

    sns.histplot(data['text\_length'], bins=30, kde=True, color='purple')

    plt.title('Text Length Distribution')

    plt.xlabel('Text Length')

    plt.ylabel('Frequency')

    if save\_path:

        plt.savefig(save\_path)

    plt.show()

def plot\_word\_cloud(data, save\_path=None):

    from wordcloud import WordCloud

    text = ' '.join(data['text'].tolist())

    wordcloud = WordCloud(width=800, height=400, background\_color='white').generate(text)

    plt.figure(figsize=(10, 6))

    plt.imshow(wordcloud, interpolation='bilinear')

    plt.axis('off')

    plt.title('Word Cloud of Text Data')

    if save\_path:

        plt.savefig(save\_path)

    plt.show()

4.4 Model Development

Developed and trained various models:

-Logistic Regression: A simple yet effective linear model for binary and multiclass classification.

- Naive Bayes: A probabilistic classifier based on Bayes' theorem.

- Support Vector Machine (SVM): A powerful classifier that finds the hyperplane that best separates classes.

- LSTM (Long Short-Term Memory): A type of recurrent neural network (RNN) capable of learning long-term dependencies.

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, SpatialDropout1D, Bidirectional, LSTM, Conv1D, GlobalMaxPooling1D, Dense

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.svm import SVC

from sklearn.model\_selection import GridSearchCV

from sklearn.metrics import classification\_report, accuracy\_score, confusion\_matrix

import seaborn as sns

import matplotlib.pyplot as plt

import numpy as np

def build\_lstm\_model(input\_dim, input\_length, embedding\_dim=100):

    model = Sequential()

    model.add(Embedding(input\_dim=input\_dim, output\_dim=embedding\_dim, input\_length=input\_length))

    model.add(SpatialDropout1D(0.2))

    model.add(Bidirectional(LSTM(64, dropout=0.2, recurrent\_dropout=0.2)))

    model.add(Dense(6, activation='softmax'))

    model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model

def build\_cnn\_model(input\_dim, input\_length, embedding\_dim=100):

    model = Sequential()

    model.add(Embedding(input\_dim=input\_dim, output\_dim=embedding\_dim, input\_length=input\_length))

    model.add(SpatialDropout1D(0.2))

    model.add(Conv1D(filters=100, kernel\_size=5, activation='relu'))

    model.add(GlobalMaxPooling1D())

    model.add(Dense(64, activation='relu'))

    model.add(Dense(6, activation='softmax'))

    model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

    return model

def train\_and\_evaluate\_model(model, X\_train, y\_train, X\_val, y\_val, X\_test, y\_test, model\_name, epochs=5, batch\_size=64):

    history = model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, validation\_data=(X\_val, y\_val), verbose=2)

    y\_pred\_prob = model.predict(X\_test)

    y\_pred = np.argmax(y\_pred\_prob, axis=1)

    y\_test\_int = np.argmax(y\_test, axis=1)

    print(f"{model\_name}:")

    print(classification\_report(y\_test\_int, y\_pred))

    print("Accuracy:", accuracy\_score(y\_test\_int, y\_pred))

    print("Confusion Matrix:")

    cm = confusion\_matrix(y\_test\_int, y\_pred)

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.title(f'Confusion Matrix: {model\_name}')

    plt.savefig(f'plots/confusion\_matrix - {model\_name}')

    return model, accuracy\_score(y\_test\_int, y\_pred)

4.5 Hyperparameter Tuning

Hyperparameter tuning was conducted using GridSearchCV to find the optimal parameters for each machine learning model.

def train\_ml\_model(model, param\_grid, X\_train, y\_train, X\_test, y\_test, model\_name):

    grid\_search = GridSearchCV(model, param\_grid, cv=5, scoring='accuracy')

    grid\_search.fit(X\_train, y\_train)

    best\_model = grid\_search.best\_estimator\_

    y\_pred = best\_model.predict(X\_test)

    print(f"{model\_name}:")

    print("Best Parameters:", grid\_search.best\_params\_)

    print(classification\_report(y\_test, y\_pred))

    print("Accuracy:", accuracy\_score(y\_test, y\_pred))

    print("Confusion Matrix:")

    cm = confusion\_matrix(y\_test, y\_pred)

    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

    plt.xlabel('Predicted')

    plt.ylabel('True')

    plt.title(f'Confusion Matrix: {model\_name}')

    plt.savefig(f'plots/confusion\_matrix - {model\_name}')

    return best\_model, accuracy\_score(y\_test, y\_pred)

4.6 Model Evaluation

Models were evaluated using various metrics:

- Accuracy: The proportion of correct predictions.

- Precision: The proportion of positive identifications that were actually correct.

- Recall: The proportion of actual positives that were identified correctly.

- F1 Score: The harmonic mean of precision and recall.

- Confusion Matrix: A table used to describe the performance of a classification model.

**5. Approach / Methodology**

The methodology employed in this project includes several key steps:

1. Data Collection: Gathering relevant text data from multiple sources.

2. Data Preprocessing: Cleaning and preparing data for modeling.

3. Feature Extraction: Using TF-IDF to convert text into numerical features.

4. Model Training: Training various machine learning and deep learning models.

5. Hyperparameter Tuning: Optimizing model parameters using GridSearchCV.

6. Model Evaluation: Evaluating model performance using standard metrics.

7. Visualization: Presenting results through charts and confusion matrices.

**6. Assumptions**

- The dataset is representative of the real-world scenario.

- Labels in the dataset are accurate and reliable.

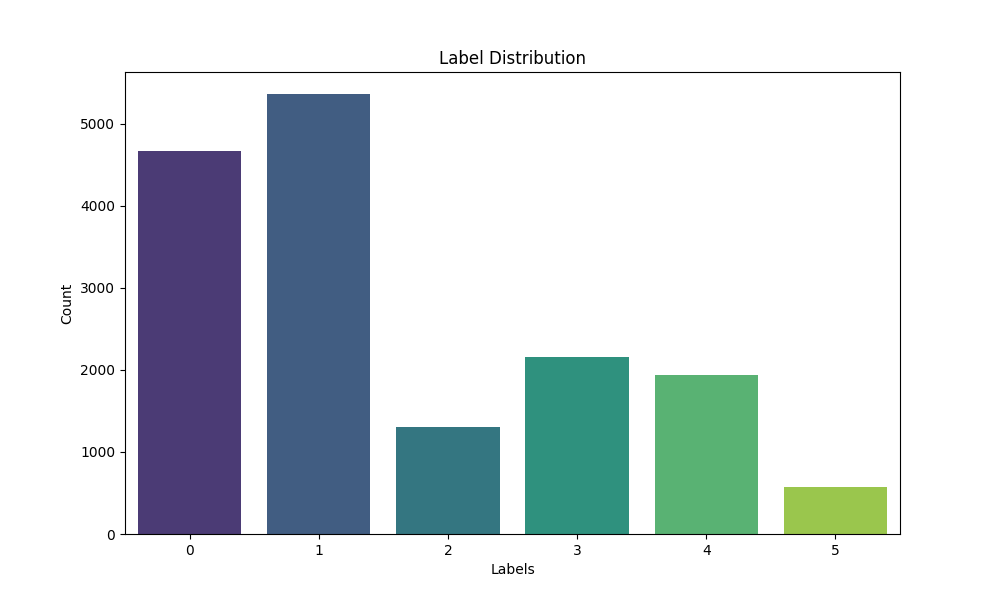
- Models chosen are appropriate for the text classification task.

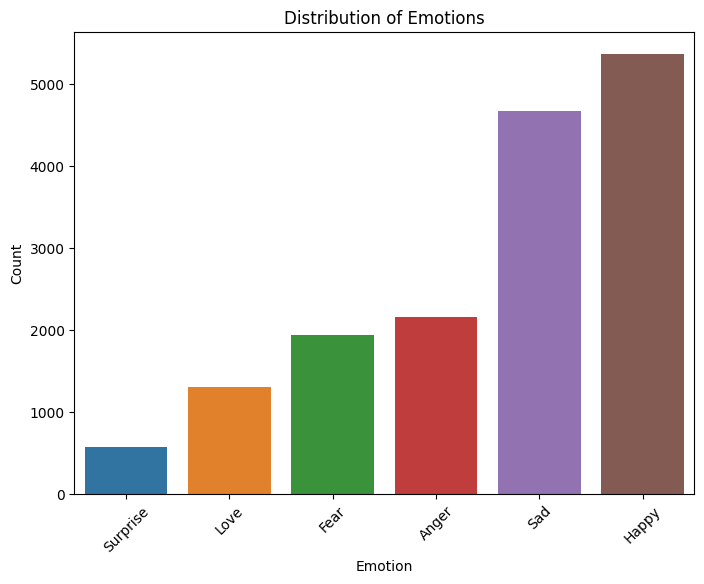
**7. Exceptions / Exclusions**

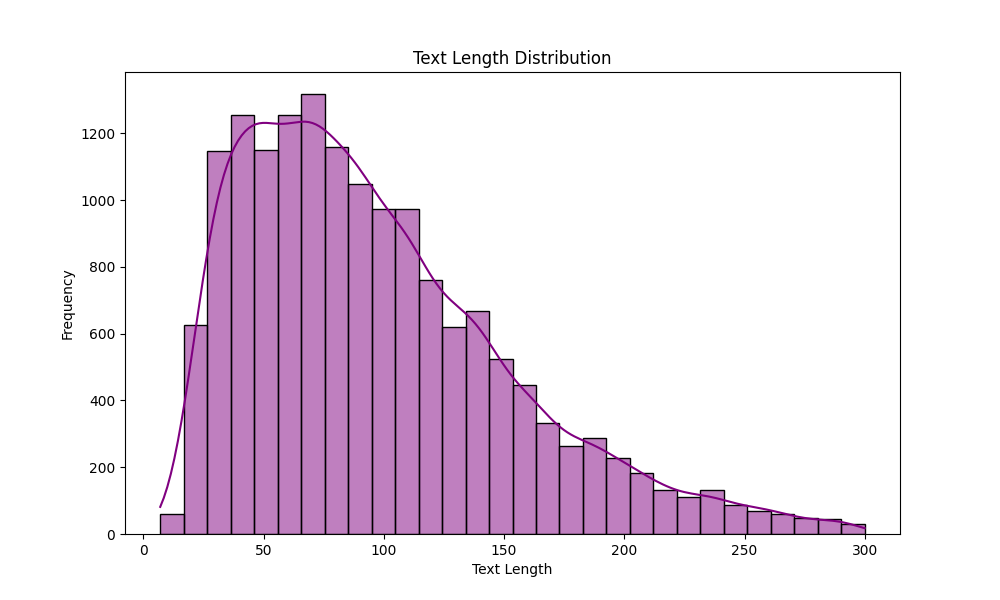
- Advanced models like transformers (e.g., BERT) were not used due to computational constraints.

- No data augmentation techniques were applied.

**8. Charts, Tables, Diagrams**

8.1 Exploratory Data Analysis







8.2 Confusion Matrices

Confusion matrices for each model highlight the number of true positives, false positives, true negatives, and false negatives, providing insights into model performance.

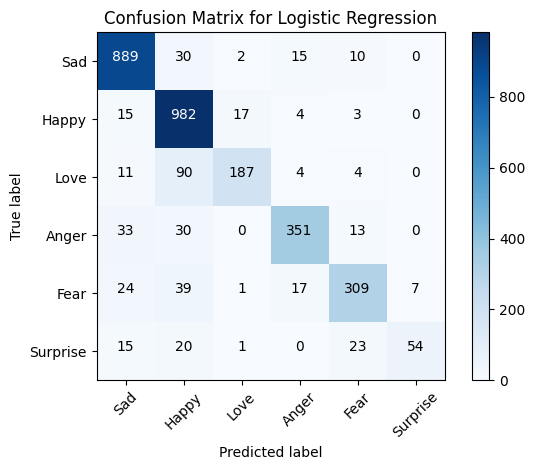
- Logistic Regression:

Accuracy: 0.86625

Precision: 0.869451247978607

Recall: 0.86625

F1 Score: 0.8612996326427544



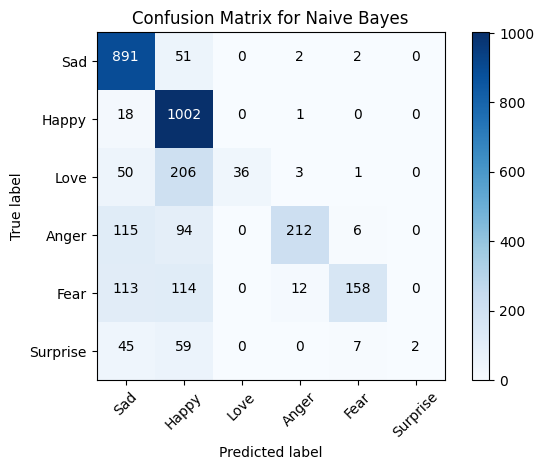
- Naive Bayes:

Accuracy: 0.7190625

Precision: 0.7867641179461465

Recall: 0.7190625

F1 Score: 0.6689774046680794



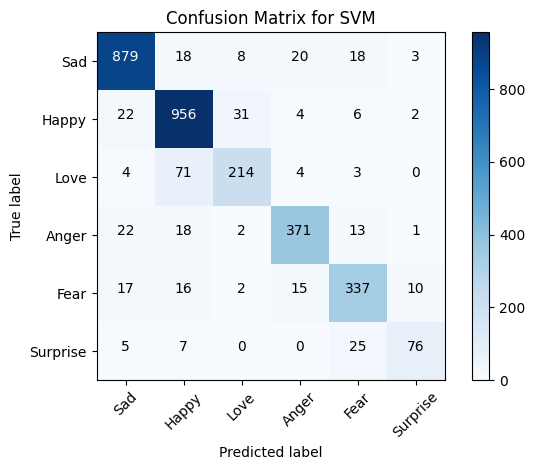
- SVM:

Accuracy: 0.8853125

Precision: 0.8844634009790472

Recall: 0.8853125

F1 Score: 0.8839436930517072



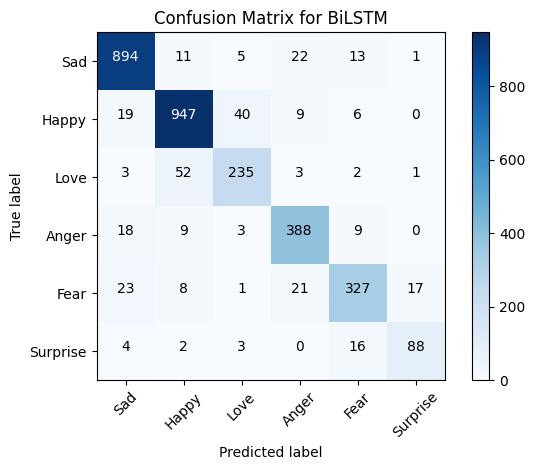
- BiLSTM:

Accuracy: 0.9218125

Precision: 0.9045532485637063

Recall: 0.9218125

F1 Score: 0.9032015817835848



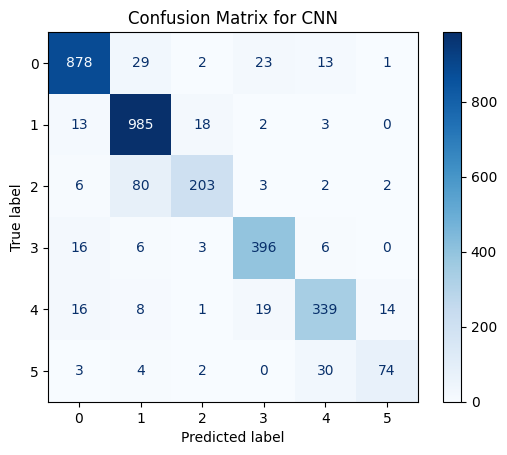
- CNN:

Accuracy: 0.9121875

Precision: 0.9139316306685835

Recall: 0.9121875

F1 Score: 0.9123694343538419



8.3 Performance Metrics Tables

Tables summarizing accuracy, precision, recall, and F1-score for each model.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1 Score |
| Logistic Regression | 0.86625 | 0.869451248 | 0.86625 | 0.861299633 |
| Naive Bayes | 0.7190625 | 0.786764118 | 0.7190625 | 0.668977405 |
| SVM | 0.8853125 | 0.884463401 | 0.8853125 | 0.883943693 |
| BiLSTM | 0.9218125 | 0.904553249 | 0.9218125 | 0.903201582 |
| CNN | 0.9121875 | 0.913931631 | 0.9121875 | 0.912369434 |

**9. Algorithms**

9.1 Logistic Regression

A linear model used for binary and multiclass classification. It predicts the probability of a class based on input features.

# Train and evaluate Logistic Regression Model

    log\_reg\_params = {

        'C': [0.01, 0.1, 1, 10, 100],

        'solver': ['newton-cg', 'lbfgs', 'liblinear']

    }

    log\_reg\_model, log\_reg\_accuracy = train\_ml\_model(LogisticRegression(), log\_reg\_params, X\_train\_tfidf, y\_train\_int, X\_test\_tfidf, y\_test\_int, "Logistic Regression")

9.2 Naive Bayes

A probabilistic classifier based on Bayes' theorem, assuming independence between features.

    # Train and evaluate Naive Bayes Model

    nb\_params = {

        'alpha': [0.5, 1.0, 1.5]

    }

    nb\_model, nb\_accuracy = train\_ml\_model(MultinomialNB(), nb\_params, X\_train\_tfidf, y\_train\_int, X\_test\_tfidf, y\_test\_int, "Naive Bayes")

9.3 Support Vector Machine (SVM)

A classifier that finds the hyperplane that best separates classes in a high-dimensional space.

    # Train and evaluate SVM Model

    svm\_params = {

        'C': [0.01, 0.1, 1, 10, 100],

        'kernel': ['linear', 'rbf']

    }

    svm\_model, svm\_accuracy = train\_ml\_model(SVC(), svm\_params, X\_train\_tfidf, y\_train\_int, X\_test\_tfidf, y\_test\_int, "Support Vector Machine")

9.4 LSTM (Long Short-Term Memory)

A type of recurrent neural network capable of learning long-term dependencies, particularly useful for sequential data like text.

    # Train and evaluate deep learning models

    lstm\_model = build\_lstm\_model(input\_dim, max\_len)

    lstm\_model, lstm\_accuracy = train\_and\_evaluate\_model(lstm\_model, X\_train\_pad, y\_train\_one\_hot, X\_val\_pad, y\_val\_one\_hot, X\_test\_pad, y\_test\_one\_hot, "Bidirectional LSTM", epochs=5, batch\_size=64)

    cnn\_model = build\_cnn\_model(input\_dim, max\_len)

    cnn\_model, cnn\_accuracy = train\_and\_evaluate\_model(cnn\_model, X\_train\_pad, y\_train\_one\_hot, X\_val\_pad, y\_val\_one\_hot, X\_test\_pad, y\_test\_one\_hot, "CNN", epochs=5, batch\_size=64)

**10. Challenges & Opportunities**

10.1 Challenges

- Imbalanced Data: Handling datasets where some classes are underrepresented.

- Computational Constraints: Limited resources for training deep learning models.

- Data Quality: Ensuring the text data is clean and accurately labeled.

10.2 Opportunities

- Model Improvement: Exploring advanced models and techniques for better performance.

- Real-world Application: Applying the models to practical problems like spam detection and sentiment analysis.

- Further Research: Investigating additional features and embeddings to enhance model accuracy.

**11. Risk Vs Reward**

11.1 Risks

- Overfitting: Models may perform well on training data but poorly on unseen data.

- Bias: Training data may introduce bias, affecting model fairness.

- Computational Cost: Training complex models requires significant computational resources.

11.2 Rewards

- High Accuracy Models: Development of robust models with high accuracy and reliability.

- Practical Experience: Gaining hands-on experience with machine learning and deep learning techniques.

- Innovation: Potential for innovative applications in various domains.

**12. Reflections on the Internship**

This internship has been a highly enriching experience, providing deep insights into text classification and machine learning. The process of developing and tuning models has enhanced my technical skills and understanding of NLP tasks. I have learned the importance of data preprocessing and feature extraction in achieving high model performance. Collaborating with my mentor and peers has also been a valuable aspect of the internship.

**13. Recommendations**

- Use Larger Datasets: For better generalization, larger and more diverse datasets should be used.

- Advanced Models: Consider implementing transformer-based models like BERT for improved performance.

- Data Augmentation: Apply data augmentation techniques to address class imbalances and improve model robustness.

**14. Outcome / Conclusion**

The project successfully developed and evaluated multiple text classification models. The LSTM model, while computationally intensive, showed promising results. Machine learning models like Logistic Regression and SVM also performed well, highlighting the effectiveness of traditional techniques in text classification. Overall, the project demonstrated the importance of thorough preprocessing, feature extraction, and model tuning in achieving high accuracy.

The best model is Bidirectional LSTM with an accuracy of 0.9210, The best model Bidirectional LSTM has been saved successfully in model folder of the project.

Some of the other detailed findings are as follows.

-Best Parameter of ML model

Logistic Regression: {'C': 10, 'solver': 'liblinear'}

Naive Bayes: {'alpha': 0.5}

Support Vector Machine: {'C': 1, 'kernel': 'linear'}

-Label wise precision, recall, and f1-score

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Logistic Regression | | | | |
| Label | precision | recall | f1-score | support |
| 0 | 0.92 | 0.94 | 0.93 | 581 |
| 1 | 0.9 | 0.93 | 0.91 | 695 |
| 2 | 0.8 | 0.75 | 0.77 | 159 |
| 3 | 0.89 | 0.87 | 0.88 | 275 |
| 4 | 0.88 | 0.84 | 0.86 | 224 |
| 5 | 0.75 | 0.64 | 0.69 | 66 |
|  |  |  |  |  |
| Naïve Bayes | | | | |
| Label | precision | recall | f1-score | support |
| 0 | 0.74 | 0.92 | 0.82 | 581 |
| 1 | 0.71 | 0.98 | 0.82 | 695 |
| 2 | 0.97 | 0.25 | 0.39 | 159 |
| 3 | 0.94 | 0.53 | 0.68 | 275 |
| 4 | 0.89 | 0.5 | 0.64 | 224 |
| 5 | 0 | 0 | 0 | 66 |
|  |  |  |  |  |
| Support Vector Machine | | | | |
| Label | precision | recall | f1-score | support |
| 0 | 0.93 | 0.92 | 0.93 | 581 |
| 1 | 0.88 | 0.95 | 0.91 | 695 |
| 2 | 0.82 | 0.69 | 0.75 | 159 |
| 3 | 0.89 | 0.88 | 0.88 | 275 |
| 4 | 0.85 | 0.86 | 0.86 | 224 |
| 5 | 0.74 | 0.56 | 0.64 | 66 |
|  |  |  |  |  |
| Bidirectional LSTM | | | | |
| Label | precision | recall | f1-score | support |
| 0 | 0.96 | 0.96 | 0.96 | 581 |
| 1 | 0.95 | 0.93 | 0.94 | 695 |
| 2 | 0.79 | 0.87 | 0.83 | 159 |
| 3 | 0.92 | 0.95 | 0.93 | 275 |
| 4 | 0.91 | 0.86 | 0.88 | 224 |
| 5 | 0.67 | 0.67 | 0.67 | 66 |
|  |  |  |  |  |
| CNN | | | | |
| Label | precision | recall | f1-score | support |
| 0 | 0.97 | 0.95 | 0.96 | 581 |
| 1 | 0.94 | 0.95 | 0.94 | 695 |
| 2 | 0.83 | 0.79 | 0.81 | 159 |
| 3 | 0.91 | 0.92 | 0.91 | 275 |
| 4 | 0.86 | 0.91 | 0.88 | 224 |
| 5 | 0.73 | 0.77 | 0.75 | 66 |

**15. Enhancement Scope**

- Incorporate Transformers: Implement models like BERT or GPT for state-of-the-art performance.

- Explore Different Embeddings: Use word embeddings like GloVe or Word2Vec for better feature representation.

- Real-time Applications: Extend the project to include real-time text classification systems.

**16. Link to Code and Executable File**

Attached is the data and all the executable files.

**17. Research Questions and Responses**

17.1 What preprocessing steps are necessary for text data?

Preprocessing steps include lowercasing, punctuation removal, tokenization, and stop words removal. These steps ensure that the text data is clean and uniform, making it suitable for modeling.

17.2 How do different machine learning models compare in text classification?

Machine learning models like Logistic Regression, Naive Bayes, and SVM have different strengths. Logistic Regression and SVM generally perform well with high accuracy, while Naive Bayes is computationally efficient and works well with smaller datasets.

17.3 What are the benefits of using deep learning models like LSTM for text classification?

LSTM models can capture sequential dependencies in text data, making them effective for tasks where context is important. They are particularly useful for handling long-term dependencies, which traditional models may struggle with.