PROJECT: - Poem Sentiment Classification using Transformers (DistilBERT)

Bachelor of Technology In Computer Science & Artificial Intelligence

By

Name:		Roll Number:
G. RITHIN GOUD	-	2203A51353
Annam Nishank Reddy	-	2203A51334

Under the Guidance of

KANDEEBAN.R

Submitted to

SCHOOL OF COMPUTER SCIENCE & ARTIFICIAL INTELLIGENCE
SR UNIVERSITY, ANANTHASAGAR, WARANGAL
March, 2025.

CERTIFICATE OF COMPLETION

This is to certify that this **Project Title: Poem Sentiment Classification using Transformers (DistilBERT),** is the bonafied work carried out by G. Rithin goud , **Nishank Reddy** students of CSE, 3rd Year-2nd Semester, Project for the partial fulfillment to award the degree BACHELOR OF TECHNOLOGY in School of Computer Science and Artificial Intelligence during the academic year 2024-2025 under our guidance and Supervision.

Dr. KANDEEBAN.
Associate Professor
SR University
Warangal

Dr. M. Sheshikala
Professor & HOD (CSE)
SR University
Warangal

ACKNOWLEDGMENT

We owe an enormous debt of gratitude to our project guide Dr. KANDEEBAN.R Professor &Dean as well as Head of the CSE Department Dr. M. Sheshikala , Professor for guiding us from the beginning through the end of the Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We wish to take this opportunity to express our sincere gratitude and deep sense of respect to our beloved Vice Chancellor, Prof. Deepak Garg, for his continuous support and guidance tocomplete this project in the institute.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

TABLE OF CONTENT

TOPIC :-

ABSTRACT

In the age of artificial intelligence, understanding human emotions through textual data has become a core task in natural language processing (NLP). While sentiment analysis in news, reviews, and social media has been extensively studied, poetry remains an area where emotional depth is both nuanced and challenging to capture computationally. This project presents an innovative approach to classify sentiments in English poetic lines using transformer-based deep learning models.

The core of this project is built around DistilBERT, a distilled version of BERT (Bidirectional Encoder Representations from Transformers) that retains 97% of BERT's language understanding capabilities while being significantly smaller and faster. The model is fine-tuned on a pre-labeled poetry sentiment dataset—poem_sentiment—which contains verses annotated with three sentiment categories: positive, neutral, and negative.

Through a robust pipeline consisting of text tokenization, embedding extraction, classification, and evaluation, the project demonstrates how transformer models can effectively grasp the abstract and metaphorical language used in poetry. This system has potential applications in literary research, automated tagging of digital poetry collections, emotion-aware recommendation systems, and personalized content curation.

OBJECTIVE

The primary objective of this project is to automatically identify the sentiment expressed in poetic lines by classifying them into one of three categories:

- Positive
- Neutral
- Negative

Key goals include:

- To explore the applicability of transformer models, particularly DistilBERT, for the classification of poetic sentiments.
- To build a complete NLP pipeline—from dataset loading, preprocessing, model training, evaluation, to inference.
- To evaluate the performance of the model on unseen poetry verses to verify its generalization capability.
- To provide a user-friendly method for predicting sentiment in poetic lines, facilitating research and educational tools.

REQUIREMENTS

1. Software Requirements :-

- **Python 3.8+**: Programming language for building the entire pipeline.
- **Jupyter Notebook or VSCode**: Development environment.
- **TensorFlow 2.x**: Deep learning library used to train and evaluate the model.
- **Transformers (Hugging Face)**: For accessing pretrained DistilBERT and tokenizers.
- **Datasets (Hugging Face)**: For loading and managing the poem sentiment dataset.
- **NumPy**: For data handling and output manipulation.
- **Matplotlib / Seaborn (Optional)**: For visualization of results or training curves.

2. Hardware Requirements :-

- **CPU**: Minimum Intel i5 or equivalent.
- **RAM**: 8 GB or more recommended.
- **GPU (Optional)**: NVIDIA GPU with CUDA support (for faster training).
- **Disk Space**: 2-3 GB for storing datasets, model weights, and output files.

3. Dataset:-

- **Name**: poem_sentiment
- **Source**: Hugging Face Datasets Library
- **Description**: Consists of English poetic verses labeled with sentiment categories:
    - 0 = Negative
    - 1 = Neutral
    - 2 = Positive

DESIGN

The system architecture is designed as a modular pipeline involving the following components:

1. Data Acquisition & Exploration

- Load the dataset using load_dataset('poem_sentiment').
- Explore training, validation, and test splits.
- Inspect sample verses and the corresponding labels.

2. Tokenization & Preprocessing

- Use DistilBertTokenizer to tokenize input text.
- Apply padding and truncation to standardize sequence lengths.
- Add attention masks for handling padded tokens.
- Encode the entire dataset using .map() for batch processing.

3. Model Architecture

- Load a pretrained TFAutoModelForSequenceClassification with:
  - Base model: distilbert-base-uncased
  - Output layer: Dense classification head with 3 neurons (for 3 classes)
- Optionally freeze base layers to reduce training time on limited datasets.
- Use dropout and regularization if needed to prevent overfitting.

4. Data Preparation for Training

- Convert the Hugging Face dataset into TensorFlow tf.data.Dataset format.
- Batch and shuffle training data.
- Maintain validation data for evaluation after every epoch.

5. Model Training

- Compile the model using:
  - **Optimizer**: Adam with learning rate 5e-5
  - **Loss**: SparseCategoricalCrossentropy (from logits)
  - **Metric**: SparseCategoricalAccuracy
- Train for multiple epochs (e.g., 5 epochs) using .fit().

6. Inference Module

- Accept custom poetic lines for prediction.
- Preprocess and tokenize inputs similarly as done during training.
- Use .predict() to obtain logits and apply argmax for final label.
- Compare predicted labels with ground truth for evaluation purposes.

7. Evaluation Metrics (Optional Enhancements)

- Confusion Matrix
- Precision, Recall, F1-Score
- Accuracy trends over epochs

8. Output

- Print predicted sentiment labels for each input verse.
- Display true sentiment (if available) for evaluation.
- Output can be extended to include confidence scores from logits.

IMPLEMENTATION

📦 Step 1: Importing the Transformers Library

python
CopyEdit
import transformers

# Set to avoid warning messages.
transformers.logging.set_verbosity_error()

- You import Hugging Face's transformers library.
- The second line suppresses warning messages, making your output cleaner.

📦 Step 2: Installing the Dataset Library

python
CopyEdit
!pip install datasets

- This command installs the Hugging Face datasets library, which provides easy access to a wide range of NLP datasets, including poem_sentiment.

📦 Step 3: Loading the Dataset

python
CopyEdit
from datasets import load_dataset

model_name = "distilbert-base-uncased"
dataset_name = "poem_sentiment"

poem_sentiments = load_dataset(dataset_name)

print(poem_sentiments)
print(poem_sentiments["test"][20:25])

print("\nSentiment Labels used", poem_sentiments["train"].features.get("label").names)

- load_dataset() loads the poem_sentiment dataset.
- The dataset is split into train, test, and validation.
- You print a few samples and label names (like negative, neutral, positive).

🔠 Step 4: Tokenizing the Data

python
CopyEdit
from transformers import DistilBertTokenizer

```
db_tokenizer = DistilBertTokenizer.from_pretrained(model_name)

def tokenize(batch):
    return db_tokenizer(batch["verse_text"], padding=True, truncation=True)

enc_poem_sentiment = poem_sentiments.map(tokenize, batched=True, batch_size=None)

print(enc_poem_sentiment["train"][0:5])
```

- Load the tokenizer for distilbert-base-uncased.
- Define a tokenize() function that turns poem text into model-readable format (input IDs, attention masks).
- Apply tokenization across all dataset splits using .map().

🧠 Step 5: Viewing Encoded Inputs

python
CopyEdit

```
print("Text :", enc_poem_sentiment["train"][1].get("verse_text"))
print("\nInput Map :", enc_poem_sentiment["train"][1].get("input_ids"))
print("\nAttention Mask :", enc_poem_sentiment["train"][1].get("attention_mask"))
```

- Prints tokenized input (IDs and masks).
- input_ids are numerical IDs for tokens.
- attention_mask marks which tokens should be attended to (1 = real token, 0 = padding).

🧪 Step 6: Splitting into Train/Validation

python
CopyEdit

```
training_dataset = enc_poem_sentiment["train"]
validation_dataset = enc_poem_sentiment["validation"]

labels = training_dataset.features.get("label")
num_labels = len(labels.names)
```

- Separate the dataset into training and validation sets.
- Retrieve the number of unique sentiment labels.

📐 Step 7: Loading Pretrained Model for Classification

python
CopyEdit

```
from transformers import TFAutoModelForSequenceClassification

sentiment_model = TFAutoModelForSequenceClassification.from_pretrained(model_name, num_labels=num_labels)
```

sentiment_model.get_config()

- Load the pretrained DistilBERT model with a classification head for 3 sentiment classes.

📦 Step 8: Model Configuration

python
CopyEdit

```
sentiment_model.layers[0].trainable = True
print(sentiment_model.summary())
```

- Ensures base DistilBERT layers are trainable (you can freeze them if needed).
- Prints the model architecture summary.

🛠️ Step 9: Preparing Data for Training

python
CopyEdit

```
import tensorflow as tf

batch_size = 64
tokenizer_columns = db_tokenizer.model_input_names

train_dataset = training_dataset.to_tf_dataset(
    columns=tokenizer_columns, label_cols=["label"], shuffle=True,
    batch_size=batch_size)

val_dataset = validation_dataset.to_tf_dataset(
    columns=tokenizer_columns, label_cols=["label"], shuffle=False,
    batch_size=batch_size)

sentiment_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=5e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=tf.metrics.SparseCategoricalAccuracy())

sentiment_model.fit(train_dataset, validation_data=val_dataset, epochs=5)
```

- Converts datasets into TensorFlow format for training.
- Compiles the model with optimizer, loss function, and accuracy metric.
- Trains the model over 5 epochs.

🧪 Step 10: Inference

python
CopyEdit

```
from datasets import Dataset, DatasetDict
```

```
infer_data = {
    'id': [0, 1],
    'verse_text': [
        'and be glad in the summer morning when the kindred ride on their way',
        'how hearts were answering to his own'],
    'label': [1, 0]
}

infer_dataset = Dataset.from_dict(infer_data)
ds_dict = DatasetDict()
ds_dict["infer"] = infer_dataset

enc_dataset = ds_dict.map(tokenize, batched=True, batch_size=None)

infer_final_dataset = enc_dataset["infer"].to_tf_dataset(
    columns=tokenizer_columns, shuffle=True,
    batch_size=batch_size)

predictions = sentiment_model.predict(infer_final_dataset)
```

- You create a small inference dataset with poem verses.
- Tokenize it just like training data.
- Run predictions using the trained model.

📜 Step 11: View Prediction Logits

python
CopyEdit

```
predictions.logits
```

- Raw output scores for each sentiment class (before applying softmax).

🏷️ Step 12: Get Predicted Labels

python
CopyEdit

```
import numpy as np

pred_label_ids = np.argmax(predictions.logits, axis=1)

for i in range(len(pred_label_ids)):
    print("Poem =", infer_data["verse_text"][i],
          " Predicted=", labels.names[pred_label_ids[i]],
          " True-Label=", labels.names[infer_data["label"][i]])
```

- Use argmax to convert logits into actual predicted labels.
- Print each poem with its predicted and true sentiment.

DEVELOPMENT AND MAINTAINANCE

🔧 Development Phase :-

The development of the *Poem Sentiment Classification* system was conducted through a series of well-defined stages, following modular and iterative practices to ensure robustness and flexibility:

1. Requirement Analysis and Dataset Selection

- The initial phase involved identifying the problem statement: classifying poem sentiments.
- The poem_sentiment dataset from Hugging Face was chosen for its labeled poetic verses, making it suitable for training a sentiment classifier.

2. Model Selection and Preprocessing

- A lightweight, efficient transformer model, **DistilBERT**, was chosen due to its balance between performance and computational efficiency.
- Tokenization was handled using DistilBertTokenizer, which converted poetic text into numerical input IDs and attention masks needed for model training.

3. Model Building

- A classification head was added to the base DistilBERT model using TFAutoModelForSequenceClassification, adapting it to output three sentiment classes: Positive, Neutral, and Negative.
- Fine-tuning of the pretrained model was done on the poem sentiment dataset to help the model adapt to poetic language nuances.

4. Training and Validation

- The training loop used TensorFlow datasets (to_tf_dataset) to optimize memory usage and performance.
- Accuracy and loss were monitored to prevent overfitting, and hyperparameters like learning rate and batch size were tuned experimentally.

5. Inference and Testing

- A test module was created to predict sentiments for new/unseen poetic lines using the trained model.
- The inference system allows input of any poem line and returns a predicted sentiment.

🔄 Maintenance Phase :-

Maintaining a machine learning system like this involves regular updates, monitoring, and adaptability. Key practices include:

1. Model Updating

- As more labeled poetic data becomes available, the model should be periodically retrained to improve its understanding of poetic structures and emerging language styles.

2. Performance Monitoring

- Performance metrics such as accuracy, F1-score, and confusion matrices should be regularly evaluated.

- A system to log incorrect predictions can help identify where the model fails and provide data for retraining.

3. Tokenizer and Library Updates

- The Hugging Face library and its tokenizers are frequently updated. Ensuring compatibility with newer versions is essential for long-term viability.
- Migrating to newer transformer models (like RoBERTa, BERTweet) could be explored in future upgrades.

4. Scalability Enhancements

- For wider deployment (e.g., integration with a web app), the model can be converted to **TensorFlow Lite** or served via **TensorFlow Serving** or **ONNX** for faster inference.

5. Bug Fixes and Code Refactoring

- Codebase should be modularized and cleaned periodically.
- Unit tests should be written for tokenization, inference, and data loading functions to ensure robustness.

CODE

```
[ ] import transformers

    #Set to avoid warning messages.
    transformers.logging.set_verbosity_error()
```

```
!pip install datasets
```

```
from datasets import load_dataset

#Use pretrained model checkpoint from Huggingface
model_name = "distilbert-base-uncased"
#Use pre-labeled dataset from huggingface
dataset_name= "poem_sentiment"

poem_sentiments = load_dataset(dataset_name)

#Apache Arrow format
print(poem_sentiments)
print(poem_sentiments["test"][20:25])

print("\nSentiment Labels used",
      poem_sentiments["train"].features.get("label").names)
```

```
#Encoding text

from transformers import DistilBertTokenizer

db_tokenizer = DistilBertTokenizer.from_pretrained(model_name)

def tokenize(batch):
    return db_tokenizer(batch["verse_text"],
                        padding=True,
                        truncation=True)

enc_poem_sentiment = poem_sentiments.map(
                        tokenize,
                        batched=True,
                        batch_size=None)

print(enc_poem_sentiment["train"][0:5])
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 1.63kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 1.50MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.35MB/s]
config.json: 100% 483/483 [00:00<00:00, 12.4kB/s]
Map: 100% 892/892 [00:00<00:00, 3462.08 examples/s]
Map: 100% 105/105 [00:00<00:00, 1736.65 examples/s]
Map: 100% 104/104 [00:00<00:00, 1566.20 examples/s]
{'id': [0, 1, 2, 3, 4], 'verse_text': ['with pale blue berries. in these peaceful shades--', 'it flows so long as falls the rain,
```

```
#Explore input IDs and Attention Mask

print("Text :",
      enc_poem_sentiment["train"][1].get("verse_text"))
print("\nInput Map :",
      enc_poem_sentiment["train"][1].get("input_ids"))
print("\nAttention Mask :",
      enc_poem_sentiment["train"][1].get("attention_mask"))

print("\nTotal tokens: ",
      len(enc_poem_sentiment["train"][1].get("input_ids")))
print("Non Zero tokens: ",
      len(list(filter(
        lambda x :x > 0,
          enc_poem_sentiment["train"][1].get("input_ids")))))
print("Attention = 1: ",
      len(list(filter(
        lambda x :x > 0,
          enc_poem_sentiment["train"][1].get("attention_mask")))))
```

```
Text : it flows so long as falls the rain,

Input Map : [101, 2009, 6223, 2061, 2146, 2004, 4212, 1996, 4542, 1010, 102, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Attention Mask : [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Total tokens:  28
Non Zero tokens:  11
Attention = 1:  11
```

```
#Separate training and validation sets
training_dataset = enc_poem_sentiment["train"]
validation_dataset=enc_poem_sentiment["validation"]

print("\nColumn Names : ",training_dataset.column_names)
print("\nFeatures : ",training_dataset.features)

labels = training_dataset.features.get("label")
num_labels=len(labels.names)
```

```
Column Names :  ['id', 'verse_text', 'label', 'input_ids', 'attention_mask']

Features :  {'id': Value(dtype='int32', id=None), 'verse_text': Value(dtype='string', id=None), 'label': ClassLabel(names=['negative', 'positive', 'no_impact', 'mixed'], id=None)
```

```
from transformers import TFAutoModelForSequenceClassification

#Load transformer checkpoint from huggingface
sentiment_model = (TFAutoModelForSequenceClassification
            .from_pretrained(model_name, num_labels=num_labels))

sentiment_model.get_config()
```

```
#Freeze the first layer if needed
sentiment_model.layers[0].trainable = True

#Add/remove layers if needed.
#sentiment_model.layers [append()/insert()/remove()]

print(sentiment_model.summary())
```

```
Model: "tf_distil_bert_for_sequence_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 distilbert (TFDistilBertMa  multiple                  66362880
 inLayer)

 pre_classifier (Dense)      multiple                  590592

 classifier (Dense)          multiple                  3076

 dropout_19 (Dropout)        multiple                  0 (unused)

=================================================================
Total params: 66956548 (255.42 MB)
Trainable params: 66956548 (255.42 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```python
#Using features from a pretrained model

import tensorflow as tf

batch_size=64
tokenizer_columns = db_tokenizer.model_input_names

# Convert to TF tensors
train_dataset = training_dataset.to_tf_dataset(
    columns=tokenizer_columns, label_cols=["label"], shuffle=True,
    batch_size=batch_size)
val_dataset = validation_dataset.to_tf_dataset(
    columns=tokenizer_columns, label_cols=["label"], shuffle=False,
    batch_size=batch_size)

sentiment_model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=5e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=tf.metrics.SparseCategoricalAccuracy())

sentiment_model.fit(train_dataset,
                    validation_data=val_dataset,
                    epochs=5)
```

```python
from datasets import Dataset,DatasetDict

#Input data for interference to predict sentiment
# the "label" array is not needed for inference, but added to provide true labels for comparison
infer_data = {'id':[0,1],
              'verse_text':['and be glad in the summer morning when the kindred ride on their way',
                            'how hearts were answering to his own'],
              'label':[1,0]}

infer_dataset = Dataset.from_dict(infer_data)

ds_dict=DatasetDict()
ds_dict["infer"] = infer_dataset

print(ds_dict)

#Encode the dataset, similar to training
enc_dataset=ds_dict.map(tokenize, batched=True, batch_size=None)

#Convert to Tensors
infer_final_dataset = enc_dataset["infer"].to_tf_dataset(
    columns=tokenizer_columns,  shuffle=True,
    batch_size=batch_size)

print(infer_final_dataset)

#Predict with the model
predictions=sentiment_model.predict(infer_final_dataset)
```

```
DatasetDict({
    infer: Dataset({
        features: ['id', 'verse_text', 'label'],
        num_rows: 2
    })
})
Map: 100% 2/2 [00:00<00:00, 62.83 examples/s]
<_PrefetchDataset element_spec={'input_ids': TensorSpec(shape=(None, 17), dtype=tf.int64, name=None), 'attention_mask': TensorSpec(shape=(None, 17), dtype=tf.int64, name=None)}>
1/1 [==============================] - 3s 3s/step
```

```python
predictions.logits
```

```
array([[-2.00713   ,  2.0774171 ,  0.09186057, -0.8323442 ],
       [-2.0107472 ,  0.68232715,  2.102883  , -1.3172748 ]],
      dtype=float32)
```

OUTPUT

The output of this project is a trained sentiment analysis model specifically tuned for English poetic verses. The key results and system outputs include:

Model Performance

- The DistilBERT-based model was trained using the poem_sentiment dataset and achieved good accuracy in classifying poetic lines into three sentiment categories: **Positive**, **Neutral**, and **Negative**.
- The model successfully processed tokenized inputs and generated **logits** (raw prediction scores) which were then transformed into label predictions using argmax.

Sample Inference Output

Given a few sample poetic verses, the system provided the following predictions:

```
import numpy as np
pred_label_ids=np.argmax(predictions.logits, axis=1)

for i in range(len(pred_label_ids)):
    print("Poem =", infer_data["verse_text"][i],
          " Predicted=",labels.names[pred_label_ids[i]],
          " True-Label=",labels.names[infer_data["label"][i]])
```

```
Poem = and be glad in the summer morning when the kindred ride on their way  Predicted= positive  True-Label= positive
Poem = how hearts were answering to his own  Predicted= no_impact  True-Label= negative
```

✅ CONCLUSION

The **Poem Sentiment Classification using Transformers (DistilBERT)** project successfully demonstrates the application of deep learning techniques to the domain of **literary text analysis**.
Using the DistilBERT transformer, we were able to train a sentiment classification model that:

- Understands poetic structure and language well enough to classify emotion with reasonable accuracy.
- Can be used as a tool for **automated literary analysis**, **emotional tone detection**, or even **recommendation systems** for poetry platforms.

🧠 Key Learnings

- Transformers like DistilBERT can effectively generalize to domain-specific text like poetry with appropriate fine-tuning.
- Preprocessing and tokenization play a vital role in preparing text for transformer-based models.
- Attention mechanisms and transfer learning allow models to adapt to subtle and metaphorical language.

🔄 Future Enhancements

- Incorporate more advanced models (e.g., BERT-large, RoBERTa) for improved accuracy.
- Introduce attention visualizations to show which parts of the verse influence sentiment classification.
- Expand to multilingual poetry sentiment analysis.
- Combine with other tasks like emotion tagging or topic detection.