Report on Multi-Label Text Classification with FastAPI and BERT

1. Overview

This project demonstrates the development of a multi-label text classification system using BERT (Bidirectional Encoder Representations from Transformers). The process involves generating synthetic data, preprocessing the text, training a multi-label classification model, and deploying the model as a REST API using FastAPI. The model predicts multiple categories for a given input text, which is useful in various real-world applications like customer feedback analysis, where each piece of text can be assigned multiple labels.

2. Project Structure

The project is structured into the following directories:

- **PROJECT_DIR**: Main directory for the project.
- **DATA_DIR**: Directory for storing the dataset.
- **MODELS_DIR**: Directory for saving trained models and associated files.

The directories are created if they do not already exist.

3. Synthetic Dataset Creation

A synthetic dataset is generated for training and testing the model. The dataset contains customer service-related text snippets and their corresponding labels. Each text sample is associated with one or more labels (e.g., "positive," "customer_support").

- **Example Texts**:
    - "The customer service was excellent, and the agent was very helpful."
    - "The product quality was disappointing and below expectations."
    - "I am extremely satisfied with the delivery speed."
- **Example Labels**:
    - "positive,customer_support"
    - "negative,product_quality"
    - "positive,delivery"

The dataset is stored in a CSV file (calls_dataset.csv).

4. Data Preprocessing

The preprocessing steps applied to the dataset include:

1. **Text Cleaning**: Using regular expressions, all non-alphanumeric characters are removed, and the text is converted to lowercase to standardize it and remove noise.
2. **Splitting Labels**: The labels column, which contains multiple labels per sample, is split into separate labels. This ensures that the model can handle multi-label classification, where each sample can have more than one label.

The preprocessed data is then split into training and testing sets (80% training, 20% testing) using train_test_split from scikit-learn.

5. Multi-Label Classification Setup

The multi-label classification setup uses the following steps:

1. **BERT Tokenizer**: The BERT tokenizer (bert-base-uncased) is used to tokenize the text into input IDs and attention masks. The text is truncated or padded to a maximum length of 128 tokens to ensure consistent input size.
2. **BERT Model**: The model used is BertForSequenceClassification from the Hugging Face transformers library, modified to handle multiple labels. The model outputs one value per label.
3. **Label Encoding**: Labels are converted into binary vectors. For each label, a value of 1 is assigned if it is present for a given sample, and 0 otherwise.

A label map is created to map each label to an index, which is stored in a JSON file (label_map.json) for later use.

6. Model Training

The BERT model is trained using the following configuration:

1. **Optimizer**: AdamW optimizer with a learning rate of 5e-5 is used for updating the model parameters during training.
2. **Training Process**: The model is trained for 3 epochs. For each epoch, the loss is calculated by comparing the model's predictions with the actual labels, and the optimizer updates the model weights accordingly.
3. **Loss Function**: The cross-entropy loss is computed for each sample, and backpropagation is performed to minimize the loss.

After training, the model and tokenizer are saved to the MODELS_DIR directory for future use.

7. FastAPI REST API

A REST API is created using FastAPI to serve the trained model. The API provides a POST endpoint (/predict/) that accepts a text input and returns the model's predictions for each label.

- **Prediction Endpoint**: The PredictionRequest class defines the input format, where the text is provided by the user. The text is tokenized, and the BERT model is used to predict the probabilities of each label. The probabilities are then passed through a sigmoid activation function to obtain values between 0 and 1, representing the likelihood of each label being applicable to the text.
- **Predictions**: The output is returned as a dictionary where the keys are the labels and the values are the predicted probabilities.

Example Response:

json

CopyEdit{

```
  "predictions": {
    "positive": 0.87,
    "customer_support": 0.92,
    "negative": 0.12,
    "product_quality": 0.85
  }
}
```


If an error occurs during the prediction process, a 500 HTTP error is raised with the exception details.

8. Key Findings and Observations

- **Synthetic Dataset**: The synthetic dataset is useful for demonstrating the workflow, but real-world datasets with more diverse and complex labels would lead to a more robust model.
- **Multi-Label Classification**: The model performs multi-label classification, predicting multiple categories for each input text. This approach is suitable for scenarios where one text can belong to several categories simultaneously (e.g., customer feedback with multiple aspects such as "service quality," "product quality," etc.).
- **Model Performance**: Given the synthetic nature of the dataset, the model's performance is likely to improve significantly with a larger and more diverse dataset. Future work could involve hyperparameter tuning, model fine-tuning, and the application of techniques like label smoothing or ensemble methods for better performance.

9. Conclusion

This project demonstrates the process of building and deploying a multi-label text classification model using BERT. By utilizing FastAPI, the model is made accessible via a simple REST API for real-time predictions. This approach can be extended to a wide range of text classification problems where multiple labels need to be predicted simultaneously, such as sentiment analysis, topic classification, and more.

---

10. Future Enhancements

1. **Model Optimization**: The training process can be further optimized by fine-tuning hyperparameters like learning rate, batch size, and number of epochs.
2. **Real-World Dataset**: Implement the system with a real-world dataset to evaluate and improve model performance.
3. **Advanced Techniques**: Incorporate advanced techniques such as transfer learning, data augmentation, or multi-task learning to improve the model’s accuracy and robustness.