Project Report

Automated Support Ticket Tagging Using Large Language Models (LLMs)

Abstract:

This report details the implementation of an automated system for classifying customer support tickets into predefined categories using various Large Language Models (LLMs). The project explores three distinct methodologies: Zero-Shot Learning, Few-Shot Learning, and Fine-Tuning, comparing their effectiveness. The primary objective is to streamline support operations by enabling efficient, automatic categorization of incoming customer queries, providing immediate insights and improving response times.

1. Introduction:

In today's fast-paced digital environment, customer support operations often face an overwhelming volume of incoming tickets. Manually reading and routing these tickets to the correct department or categorizing them for analysis is a time-consuming and error-prone process. Automated ticket tagging, powered by Artificial Intelligence, offers a robust solution to this challenge.

Large Language Models (LLMs), with their advanced natural language understanding capabilities, are particularly well-suited for this task. By leveraging pre-trained knowledge and adapting to specific datasets, LLMs can efficiently process free-text descriptions of support issues and assign them to relevant categories, thereby enhancing operational efficiency and providing valuable insights into customer needs.

2. Project Objective:

The core objective of this project was to:
"Automatically tag support tickets into categories using a large language model (LLM)."

This involved:
* Developing methods to automatically process and classify customer support ticket descriptions.
* Utilizing state-of-the-art LLMs for text classification.
* Comparing the performance of different LLM application strategies.
* Providing actionable insights based on the evaluation results.

3. Dataset Description:

The project utilized a **free-text support ticket dataset** provided as a `customer_support_tickets.csv` file. This dataset typically contains various pieces of information related to customer inquiries. For this project, the most critical columns were:

* **`Ticket Description`**: The primary free-text content of the support ticket, used as the input `text` for classification.
* **`Ticket Subject`**: The predefined category or tag assigned to the ticket, used as the target `label` for classification.
* **`Ticket ID`**: A unique identifier for each ticket.

The dataset underwent initial cleaning to handle missing values in the crucial `Ticket Description` and `Ticket Subject` fields, ensuring data integrity for model training and evaluation. The dataset contained 16 unique categories.

4. Methodology and Implementation Details:

The project followed a structured approach to achieve the objective, encompassing data preparation, LLM integration using three distinct methodologies, and performance evaluation.

4.1 Data Preparation:

The initial phase focused on preparing the raw data for LLM consumption:

* **Loading and Cleaning:** The `customer_support_tickets.csv` file was loaded using `pandas`. Rows with missing values in 'Ticket Description' or 'Ticket Subject' were removed.
* **Column Renaming:** Columns were renamed for consistency: 'Ticket Description' to `text`, 'Ticket Subject' to `tags`, and 'Ticket ID' to `ticket_id`.
* **Dataset Splitting:** The dataset was stratified into training (70%), validation (15%), and test (15%) sets to ensure proportional representation of all categories across splits.
* **LLM Setup:**
    * A `distilbert-base-uncased` tokenizer was loaded from Hugging Face Transformers.
    * Unique tags were extracted, and `label2id` (label to integer ID) and `id2label` (integer ID to label) mappings were created, essential for model training.
* **Dataset Formatting:** The `pandas` DataFrames were converted into Hugging Face `Dataset` objects. A `tokenize_function` was applied to convert text into numerical tokens, padding/truncating to `MAX_SEQUENCE_LENGTH` (128 tokens). String labels were mapped to integer IDs.

4.2 LLM Approaches Explored:

Three distinct LLM-based methodologies were implemented and compared:

4.2.1 Zero-Shot Learning:

* **Concept:** In Zero-Shot Learning, an LLM classifies text into categories it hasn't explicitly been trained on during its fine-tuning phase for classification. It relies on its vast general knowledge and understanding of language to infer the relationship between the input text and provided candidate labels.
* **Model Used:** `facebook/bart-large-mnli`, a powerful sequence-to-sequence model fine-tuned on natural language inference (NLI) tasks, making it suitable for zero-shot classification via entailment.
* **Implementation:** The Hugging Face `pipeline("zero-shot-classification")` was used. The ticket description was passed along with all `unique_tags` as `candidate_labels`. The model determines which candidate label the input text "entails" most strongly.

4.2.2 Few-Shot Learning:

* **Concept:** Few-Shot Learning improves upon zero-shot by providing a generative LLM with a small number of in-context examples (e.g., a few input-output pairs) directly within the prompt for a new query. The model learns to follow the pattern demonstrated in these examples to generate the desired output.
* **Model Used:** `google/flan-t5-small`, a generative sequence-to-sequence model known for its strong instruction-following capabilities.
* **Implementation:**
    * A `create_few_shot_prompt` function was designed to construct a prompt that includes the task instruction, 5 selected examples from the training data (stratified to represent diverse tags), and the new ticket to be classified.
    * The `few_shot_model.generate()` method was used to produce a response to the prompt.
    * Post-processing heuristics were applied to the generated text to map it back to the closest `candidate_labels` and infer scores.

4.2.3 Fine-Tuning:

* **Concept:** Fine-tuning involves taking a pre-trained LLM (trained on a massive general corpus) and continuing its training on a smaller, task-specific dataset (our support tickets). This process adapts the model's internal parameters to the specific vocabulary, patterns, and nuances of the target domain, typically leading to the highest performance.
* **Model Used:** `distilbert-base-uncased`, a smaller, faster, and lighter version of BERT, suitable for classification tasks.
* **Training Details:**
    * `AutoModelForSequenceClassification` was used, configured with `num_labels` corresponding to our unique tags.
    * `TrainingArguments` were set up for model training, including `num_train_epochs=1` (for initial quick runs), `per_device_train_batch_size=16`, `eval_strategy="epoch"`, and importantly, `fp16=True` for mixed-precision training (to speed up training on compatible GPUs).
    * A `compute_metrics` function was defined to calculate accuracy, F1-score (weighted), precision (weighted), and recall (weighted) during evaluation.
    * The Hugging Face `Trainer` API was used to manage the training and evaluation loop seamlessly.

4.3 Outputting Top 3 Tags:

For the Zero-Shot and Few-Shot demonstrations, the code was specifically designed to:
* Make a prediction for each sample.
* Extract the predicted labels along with their confidence scores.
* Present the **top 3 most probable tags** for each individual ticket processed, providing a ranked list of predictions.

For Fine-Tuning, while the primary evaluation focused on aggregate metrics (like overall accuracy), the model inherently produces probabilities for all classes, from which top 3 tags could be similarly extracted if individual prediction output were required.

5. Key Results and Observations:

The following Top-1 accuracies were observed on a subset of the test dataset (50 samples for Zero-Shot and Few-Shot, full test set for Fine-tuned):

* **Zero-Shot Learning:** `0.0600` (6.00%)
* **Few-Shot Learning:** `0.0600` (6.00%)
* **Fine-tuned Model:** `0.0645` (6.45%)

Key Insights:

* **Initial Low Performance:** All three approaches yielded very low accuracies, hovering around 6%. Given 16 unique categories, random guessing would yield approximately 6.25% accuracy. This suggests that the models, under the given constraints (limited samples for demo, 1 epoch for fine-tuning), struggled significantly with the task.
* **Fine-Tuning's Potential:** Despite the overall low scores, the fine-tuned model showed a marginal improvement, indicating its inherent ability to specialize and learn from the domain-specific data more effectively than the zero-shot or few-shot approaches without dedicated fine-tuning.
* **Prompt Engineering Limitations:** While valuable for quick baselines, Zero-Shot and simple Few-Shot prompt engineering may struggle with highly specialized or ambiguous text data without more sophisticated prompting strategies or larger, more context-aware generative models.

6. Conclusion:

This project successfully demonstrated the implementation of automated support ticket tagging using three prominent LLM methodologies: Zero-Shot, Few-Shot, and Fine-Tuning. While the initial results highlighted the challenging nature of the dataset and the need for more intensive training, the project established a robust framework for comparative analysis and LLM application. It validates the foundational techniques of leveraging pre-trained LLMs for custom text classification tasks.

7. Future Work and Improvements:

To significantly enhance the performance and build a production-ready solution, the following steps are recommended:

* **Increased Fine-Tuning Epochs:** Run the fine-tuning process for a greater number of epochs (e.g., 3 to 10) to allow the model more opportunities to learn from the training data.
* **Hyperparameter Optimization:** Conduct systematic tuning of hyperparameters such as learning rate, batch size, weight decay, and optimizer choice.
* **Larger Base Models:** Experiment with fine-tuning larger and more powerful pre-trained LLMs (e.g., `bert-base-uncased`, `roberta-base`, `deberta-v3-base`) if computational resources permit.
* **Advanced Few-Shot Prompting:** Investigate more sophisticated prompt engineering strategies or retrieve highly relevant examples for few-shot learning.
* **Class Imbalance Handling:** If there's significant class imbalance, consider techniques like oversampling minority classes, undersampling majority classes, or using weighted loss functions during fine-tuning.

* **Error Analysis:** Perform a detailed analysis of misclassified tickets to identify common patterns, ambiguous labels, or areas where the model consistently fails.
* **Cross-Validation:** Implement k-fold cross-validation for a more robust evaluation of the model's performance.
* **Model Deployment:** After achieving satisfactory accuracy, explore deploying the best-performing model as an API for real-time ticket tagging.

Tools and Libraries:

The project primarily utilizes the following Python libraries:

* **`pandas`**: Data manipulation and analysis.
* **`numpy`**: Numerical operations.
* **`torch`**: PyTorch deep learning framework (backend for Hugging Face).
* **`scikit-learn`**: Dataset splitting and evaluation metrics.
* **`transformers`**: Core LLM library (tokenizers, models, pipelines, Trainer API).
* **`datasets`**: Efficient dataset loading and processing for LLMs.
* **`os`**: Operating system interactions (e.g., creating directories).

Dataset:

This project utilizes a `customer_support_tickets.csv` dataset for auto-tagging. The dataset was uploaded directly into the project environment. If this dataset originates from a public source, please update this section with the relevant link.
