Predicting 30-Day Readmission Outcomes for Diabetes Patients: Early, Late, or No Readmission

Loyd Flores
Department of Computer Science
CUNY Queens College
New York, NY 11355

Abstract

abstract here

1 Literature Review

1.1 HuggingFace and the democritization of LLM's

The rapid advancements in Natural Language Processing (NLP) have introduced transformative capabilities to various sectors, ranging from healthcare to finance. However, these advancements have also exposed a critical gap in accessibility and usability, particularly for smaller organizations and individual developers. This gap is where Hugging Face has played a pivotal role by democratizing access to state-of-the-art large language models (LLMs). Hugging Face's platform provides an ecosystem of tools, datasets, and pre-trained models, fostering an inclusive environment for developers to fine-tune and deploy LLMs for specialized tasks.

One of Hugging Face's key contributions is its commitment to accessibility. The platform simplifies the traditionally complex process of implementing LLMs, enabling a broader range of users to engage with cutting-edge AI technologies (OpenDataScience, n.d.). By lowering the technical barriers, Hugging Face empowers developers and organizations to innovate without the need for extensive computational resources or expertise in deep learning. This democratization has a cascading effect, fueling innovation across diverse industries and ensuring that the benefits of AI are not limited to a privileged few.

2 Introduction

The field of Natural Language Processing (NLP) has undergone rapid advancements in recent years, driven by the development of large pre-trained language models. These models, such as those provided by Hugging Face, enable applications ranging from sentiment analysis to machine translation with high levels of accuracy and efficiency. However, to achieve optimal performance on domain-specific tasks, fine-tuning these pre-trained models is essential.

This report explores the process of fine-tuning NLP models using Hugging Face's ecosystem, which has emerged as a leading platform for model deployment, training, and experimentation. Leveraging its comprehensive suite of tools, we focus on adapting a pre-trained transformer model to a specialized dataset, addressing challenges such as data preprocessing, hyperparameter selection, and evaluation metrics. The goal is to demonstrate how fine-tuning can significantly enhance model performance for targeted applications.

3 Methodology

3.1 Understanding the data

The dataset used for this project is the TweetEval dataset, a comprehensive benchmark specifically designed for evaluating sentiment analysis and emotion classification tasks in natural language processing (NLP). This dataset is publicly available through Hugging Face's Datasets library, providing seamless integration with widely used NLP frameworks.

The TweetEval dataset was selected due to its relevance and structure tailored for multi-class classification tasks, such as emotion detection. With labeled data for emotions including anger, joy, optimism, and sadness, it offers a targeted and compact dataset that aligns with the project's goals of developing and fine-tuning an NLP model for emotion recognition in text. Additionally, the dataset's preprocessed and clean structure reduces initial overhead, enabling faster experimentation and iterations. The dataset's focus on Twitter data introduces real-world complexity, such as informal language and abbreviations, making it an ideal testbed for practical model applications.

The dataset is divided into three splits:

- **Training Set:** Contains 3,257 samples, comprising the majority of the dataset.
- **Validation Set:** Consists of 374 samples, used for model tuning.
- **Test Set:** Includes 1,421 samples for evaluating model performance.

Each data sample includes two fields:

- **Text:** A tweet containing the input text for analysis.
- **Label:** An integer representing the corresponding emotion class.

Example:

- **Text:** "Worry is a down payment on a problem you may never have. #motivation"
- **Label:** 2 (mapped to the emotion category "optimism")

Label: 2 (mapped to the emotion category "optimism")

3.2 Data Transformation

3.2.1 Filtering

Filtering involves removing sequences that exceed a predefined length to optimize training efficiency and resource usage.

Filtering ensures that sequences are manageable in size, preventing memory overflow and reducing computational complexity during training.

By discarding excessively long sequences, we maintain a consistent dataset while minimizing training time. However, this may result in the loss of some potentially informative samples.

The dataset was filtered using a custom Python function that compared the length of each sequence with the predefined maximum length of 36 words. The DatasetDict class's filter method was employed to create a new dataset that retained only sequences within this limit.

3.2.2 Padding

Padding adds extra tokens to shorter sequences to make them the same length as the longest sequence in the dataset.

Padding ensures uniform input size for batch processing and facilitates efficient computation on GPUs.

While padding simplifies model input, excessive padding can introduce unnecessary computation and may slightly degrade model performance.

Padding was integrated into the tokenization process using the tokenizer's built-in functionality. Sequences shorter than 36 words were padded with the [PAD] token, ensuring that all sequences conformed to the maximum length.

3.2.3 Tokenization

Tokenization is the process of converting textual data into numerical format that the model can process. Each word or subword is assigned a unique identifier, enabling the text to be represented as a sequence of tokens.

Tokenization ensures that text input is compatible with the model's architecture, as models like DistilBERT require numerical inputs.

Proper tokenization improves the model's ability to understand the context and meaning of the text. It also includes handling special tokens like [CLS] and [SEP], which help in sequence classification tasks.

I utilized the pre-trained tokenizer from the Hugging Face library. The tokenizer was instantiated using the AutoTokenizer class, which is specifically designed to work with DistilBERT. Padding and truncation were applied to ensure all sequences matched the length of the longest sequence in the dataset (36 words). This step standardized input dimensions for the model.

3.2.4 Attention Masks

Attention masks are binary arrays that indicate which tokens in a sequence should be attended to by the model.

Attention masks help the model differentiate between meaningful tokens and padding tokens, ensuring accurate context processing.

By focusing on relevant tokens, the model avoids learning from padding, thereby improving its overall performance.

Attention masks were automatically generated during tokenization. Tokens corresponding to actual words received a value of 1, while padding tokens were assigned a value of 0. These masks were stored as an additional feature in the dataset.

3.2.5 Dataset Preparation

Splitting the dataset into training, validation, and test sets ensures that the model is trained, tuned, and evaluated on separate subsets of data. The train-test split helps to assess the model's generalization capabilities by evaluating its performance on unseen data during testing. Ensuring a proper split ratio minimizes the risk of overfitting and underfitting while providing sufficient data for model evaluation. For this project, the dataset was already pre-split into training, validation, and test sets, simplifying this process.

After tokenization and during train-test split, converting data into TensorFlow tensors is essential for feeding it into the model during training. TensorFlow tensors provide a structured and efficient way to handle data for computation, ensuring compatibility with the TensorFlow framework.

This conversion standardizes the data format and allows seamless integration with TensorFlow's data pipeline, enabling batch processing and efficient GPU utilization. Features such as input_ids and attention_mask were extracted from the tokenized dataset and converted into TensorFlow tensors. Labels were one-hot encoded using TensorFlow's to_categorical function, ensuring they were in the correct format for multi-class classification. These tensors were then grouped into TensorFlow datasets using the tf.data.Dataset.from_tensor_slices method, facilitating batch processing during model training.

3.3 Pretrained model Integration and Fine Tuning

3.3.1 Model Setup

Setting up the pre-trained model involves initializing the architecture and preparing it for transfer learning. The pre-trained model provides a foundation built on large amounts of data, allowing fine-tuning on a specific task without requiring full retraining. Using a pre-trained model reduces training time and computational costs while maintaining high performance for domain-specific tasks. The **AutoModelForSequenceClassification** class from HuggingFace was used to load the pre-trained DistilBERT model. The model's output layers were configured for multi-class classification by specifying the num_labels parameter as 4 (corresponding to anger, joy, optimism, and sadness).

3.4 Model Architecture

The fine-tuned model includes additional layers designed to optimize classification performance. These layers adapt the pre-trained DistilBERT model to the emotion classification task by focusing on task-specific features. Adding layers enhances the model's ability to learn distinctions in the data while maintaining the pre-trained weights' generalization capabilities.

- **Pre-classifier Dense Layer:** A dense layer added after the DistilBERT transformer outputs to prepare features for classification.
- **Classifier Dense Layer:** The final layer, which maps features to the 4 emotion classes.
- **Dropout Layer:** Incorporated to reduce overfitting by randomly deactivating neurons during training.

3.5 Freezing Weights

Freezing specific layers prevents updates to their weights during training. This step ensures that the pre-trained knowledge embedded in DistilBERT remains intact, while only the added layers are trained. Freezing reduces computational requirements and prevents overfitting, especially when working with smaller datasets.

3.6 Hyperparameter Tuning

Optimizing hyperparameters tailors the training process to the dataset's characteristics. Effective hyperparameter tuning enhances model accuracy while mitigating overfitting. Adjusting parameters such as learning rate impacts training efficiency and final model performance.

Implementation:

- **Learning Rate Scheduler:** A schedule was implemented to gradually reduce the learning rate as training progressed, using TensorFlow's LearningRateScheduler class.
- **Performance Metrics:** The model was compiled with categorical cross-entropy as the loss function and categorical accuracy as the evaluation metric.

3.7 Selecting Metrics and Compilation Details

Selecting appropriate performance metrics ensures effective evaluation of the model's progress and final performance. Metrics such as accuracy and loss provide insights into how well the model is learning and generalizing to unseen data.

The model was compiled using the following classes:

- **tf.keras.optimizers.Adam:** Optimizer for gradient-based updates.
- **tf.keras.losses.CategoricalCrossentropy:** Loss function for multi-class classification tasks.
- **tf.keras.metrics.CategoricalAccuracy:** Evaluation metric to assess accuracy.

4 Experimentation

4.1 Performance Measurement

4.1.1 Accuracy

Accuracy is a performance metric that measures the proportion of correct predictions made by a model out of the total predictions. It is a straightforward and commonly used metric for evaluating classification models. Accuracy calculates how often the model's predictions match the actual labels. It works by comparing the predicted labels to the true labels and counting the number of correct predictions.

$$\text{Accuracy} = \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}}$$

4.1.2 Log Loss

Log loss, also known as logarithmic loss or cross-entropy loss, is a performance metric commonly used in classification problems. It measures the difference between the predicted probabilities and the actual labels.

4.2 Results

4.2.1 Training Performance

The training categorical accuracy increased steadily from 60.88% in the first epoch to 76.54% by the last epoch. The training log loss decreased significantly, demonstrating effective learning by the model.

4.3 Validation Performance

Validation categorical accuracy improved from 61.76% in the first epoch to a peak of 68.98% in epoch 12. Validation loss decreased to 0.7870 by epoch 13 but slightly plateaued afterward, indicating some overfitting in the later epochs.

4.4 Testing Performance

The final evaluation on the test set yielded:

- **Log Loss:** 0.7024
- **Categorical Accuracy:** 72.77%

5 References

[1] OpenDataScience (n.d.) The Evolution of Hugging Face and its Role in Democratizing AI. Retrieved from https://opendatascience.com/the-evolution-of-hugging-face-and-its-role-in-democratizing-ai.