Fine-tuning the DistilBERT for sentiment analysis with Monetary Policy Speeches

Ozodbek Ozodov

Sunday 10th September, 2023

Abstract

This study is dedicated to the adaptation of the DistilBERT architecture to better interpret the sentiment embedded within monetary policy speeches. While the DistilBERT model has demonstrated its efficacy across various domains, the unique lexicon and subtleties present in central banking communications necessitate specialized tuning. This study involved the meticulous annotation of speech, report, and press release data from the European Central Bank (ECB) and subsequent model fine-tuning. Results underscore the model's adeptness at discerning sentiment in this specialized context, offering a valuable tool for researchers and analysts examining the intersection of sentiment and monetary policy.

1 Introduction

Sentiment analysis, a burgeoning field within Natural Language Processing (NLP), has seen growing interest in its application to the domain of economics, particularly in understanding the intricate narratives of monetary policy communications. The underlying sentiment of such communications, emanating from central banks and other pivotal financial institutions, holds significant sway over financial markets. Hence, the precision with which we discern this sentiment can greatly impact economic forecasts, policy responses, and investment strategies.

DistilBERT, a lighter version of the BERT model, has been proposed as a model that retains most of BERT's performance characteristics while being less resource-intensive. Developed by HuggingFace, it leverages the Transformer architecture for dealing with a myriad of NLP tasks, thus providing a promising foundation for fine-tuning in specialized domains.

Historically, fine-tuning pre-trained models for domain-specific tasks has proven efficacious in multiple contexts. Within the realm of economics and finance, models like FinBERT have sought to harness the power of BERT's architecture to interpret financial texts. However, preliminary experimentation with FinBERT for decoding monetary policy speeches yielded sub-optimal results, suggesting a need for a more tailored approach. It's pertinent to note that while FinBERT offered invaluable insights, the choice to diverge from it in this study stems from a pedagogic perspective; this work is as much about contributing to the academic dialogue as it is about the researcher's journey of exploration and learning.

Thus, this study pivots to the DistilBERT architecture, aiming to capture the nuances of monetary policy communications better than existing models. By focusing on a curated dataset comprising speeches, reports, and press releases from the European Central Bank (ECB), this research fine-tunes the DistilBERT model to better cater to the sentiment embedded in these specialized communications.

The implications of this study are manifold. A well-tuned model can be a significant asset for policymakers to gauge the implicit sentiment of their communications. Financial analysts and investment strategists can also utilize this model to anticipate market reactions, adjusting strategies in real-time. Moreover, researchers engaged in economic discourse analysis may find the fine-tuned DistilBERT a valuable tool for dissecting the rhetoric of central banking communications.

In essence, this research not only augments the arsenal of sentiment analysis tools available for economic texts but also charts a reflective academic journey, emphasizing the iterative nature of learning and model development in the rapidly evolving landscape of NLP.

2 Model development and fine-tuning

2.1 Baseline DistilBERT

2.1.1 An Overview

DistilBERT, stemming from the family of BERT (Bidirectional Encoder Representations from Transformers) models, represents a distilled and streamlined variant designed to retain the robustness of BERT while being computationally more efficient. Developed by HuggingFace, it emerges as a product of knowledge distillation where the distilled model (DistilBERT) is trained to emulate a teacher model (BERT) through the soft targets produced by the latter.

2.1.2 Architecture and sentiment analysis

At its core, DistilBERT adopts the Transformer architecture, which is predicated on self-attention mechanisms allowing it to weigh input tokens differently, thereby capturing contextual relationships in texts irrespective of their positional distance. While BERT leverages this architecture with multiple layers (typically 12 for BERT-base and 24 for BERT-large), DistilBERT reduces the number of these layers by 50 percent, resulting in faster computation without substantially compromising performance.

For sentiment analysis, DistilBERT, like BERT, operates by comprehending the contextualized embeddings of input text. In a baseline scenario, the process follows a set pattern:

1. Tokenization: The input text is tokenized, broken down into subwords or tokens suitable for processing. These tokens are then mapped to their corresponding embeddings in the pre-trained model.

2. Processing via the Transformer: The token embeddings traverse the Transformer layers, undergoing a series of transformations influenced by the self-attention mechanism. This process ensures that each token captures the essence of its surrounding context.

3. Classification Layer: For sentiment analysis, a classification layer is appended on top of the Transformer output. Specifically, the embedding corresponding to the '[CLS]' token (a special token placed at the beginning of every input text) is often used as a summarization of the entire input, and is thus passed through a dense layer to yield sentiment predictions.

4. Fine-tuning: While DistilBERT comes pre-trained on vast corpuses like the BookCorpus and English Wikipedia, for specific tasks like sentiment analysis, the model is typically fine-tuned on labeled data in the target domain. This involves backpropagating the loss from incorrect predictions and adjusting the model weights accordingly.

DistilBERT has been recognized for offering a balanced trade-off between computational efficiency and performance prowess. Its application in sentiment analysis serves as a testament to the model's capability to discern subtle emotional nuances, especially when fine-tuned to domain-specific datasets

2.2 Adjusting the model to specific needs

The development of our sentiment analysis model using DistilBERT can be broken down into stages of computational processing and data preprocessing, each underpinned by deliberate choices informed by the needs of the task and the constraints of available resources.

Given the sophisticated and intricate nature of DistilBERT, training such models necessitates substantial computational power. In this study, the model was trained utilizing a robust 20GB GPU provided by Google Colab. This cloud-based platform has increasingly become an integral part of deep learning research due to its ability to offer powerful GPUs, making it conducive to train complex models with expansive datasets.

However, a crucial point of consideration is the reproducibility and application of the trained model. Training on a high-capacity GPU means that attempting to replicate the training process or even run the model on conventional, less powerful machines can be computationally prohibitive. Recognizing this, the trained model was saved post-training. This approach ensures that researchers and analysts can efficiently utilize the model for inferencing tasks without the necessity of a high-performance GPU, thereby enhancing the accessibility and usability of the model across diverse platforms.

2.3 Data preprocessing

The integrity and effectiveness of a machine learning model are invariably tied to the quality of its input data. Recognizing this, a meticulous preprocessing pipeline was employed, informed by both academic literature and industry best practices.

Text Lowercasing: All words in the dataset were converted to lowercase. This is a conventional practice in NLP that serves to standardize the dataset, ensuring that the model does not treat variations in capitalization (e.g., "Economics" vs. "economics") as distinct entities. Such a procedure is pivotal for consistency and reducing the dimensionality of the input space.

Stopword Removal: Stopwords, typically the most common words in a language (e.g., "and", "the", "is"), were removed from the dataset. While these words are essential for human language comprehension, they often contribute minimal semantic value in model training. Their removal can help in reducing noise and computational load.

Removal of Non-recognizable Characters: Any characters that were non-recognizable were purged from the dataset. Such characters can arise from encoding errors, non-standard text inputs, or other anomalies. Removing them ensures the model is trained on clean and standardized data, thereby boosting its ability to generalize from the training data to unseen samples.

A noteworthy aspect of the data preprocessing was the manual intervention involved. While automation underpins much of modern NLP preprocessing, the nuances and intricacies of real-world data sometimes necessitate human judgment. In this study, some segments of the dataset were manually curated to address specific, non-systematic anomalies. While every individual alteration may not be exhaustively documented, this level of manual scrutiny underscores a commitment to data quality and integrity, even if it introduces an element of opacity in the preprocessing pipeline.

In summation, the model development phase was characterized by a synergy of computational rigor and meticulous data preparation. These choices, while informed by existing practices, were tailored to the specific demands of the project, ensuring the resultant model is both robust in performance and grounded in sound academic principles.

2.4 Model architecture

2.4.1 Token Encoding and Preparation for Model Input

In the described procedure, the raw sentences from the dataset undergo a systematic transformation to become suitable for ingestion by the DistilBERT model. This process involves several critical steps that ensure the sentences are correctly tokenized, encoded, and standardized for consistent model input:

Tokenization and Special Tokens: Each sentence is tokenized, which involves breaking down the text into individual tokens recognizable by the DistilBERT vocabulary. Along with this tokenization, two special tokens are added - [CLS] at the beginning and [SEP] at the end of each sentence. The [CLS] token, in particular, plays a pivotal role in classification tasks, as its corresponding output embedding often serves as a summary representation of the entire input sequence.

Mapping Tokens to IDs: Post tokenization, each token is mapped to a unique identifier known as a token ID. These IDs correspond to entries in the model's vocabulary, allowing the model to process the input text in a numerical format.

Padding and Truncation: Given the varying lengths of sentences, it becomes imperative to standardize the input length for the model. This is achieved by setting a max_length, to which all tokenized sequences are either padded or truncated. In the described process, a maximum length of 500 tokens has been chosen. If a sentence has fewer tokens than this limit, it gets padded with special [PAD] tokens to reach the set length. Conversely, if a sentence exceeds this token count, it is truncated to fit the defined boundary.

Attention Masks: Alongside token IDs, attention masks are constructed. These masks play a crucial role in the transformer's self-attention mechanism. Essentially, they differentiate the actual tokens from the padding tokens. An attention mask has the same length as the input token sequence and carries a binary value for each token — typically "1" for real tokens and "0" for padding tokens. This ensures that the model does not assign importance to the padding tokens during the attention computation.

Tensor Conversion: The token IDs, attention masks, and labels are converted into PyTorch tensors. Tensors are multi-dimensional arrays, and they are the standard input format for neural models in the PyTorch framework.

Overall, this entire procedure encapsulates the essential pre-processing tasks needed to convert raw text data into a structured format that the DistilBERT model can efficiently process and understand.

2.4.2 Training and validation sets

In the context of machine learning, especially deep learning, it's essential to partition the dataset into distinct subsets, primarily to ensure that the model can generalize well to new, unseen data. These partitions typically come in the form of a training set and a validation set.

Role of Training and Validation Sets:

Training Set: This is the primary dataset used to train the model. The model learns patterns, relationships, and structures in this data. Validation Set: After training, the model's performance is evaluated on the validation set. This set is not used during training and provides a way to gauge how well the model is likely to perform on unseen data. Creating the TensorDataset: First, the token IDs, attention masks, and labels, which were previously converted into PyTorch tensors, are combined into a single TensorDataset. This structure makes it convenient to manage and access the corresponding elements of these three tensors in tandem.

Deciding the Split Ratio: The dataset is split into a 80-20 ratio. This means that 80 percent of the data is allocated to the training set, and the remaining 20 percent is reserved for the validation set. While this is a commonly used ratio, the exact proportions can be adjusted based on the size of the dataset and the specific needs of the task.

To ensure that both the training and validation sets are representative of the overall data distribution, samples are randomly selected when creating these sets. This random splitting helps in ensuring that the model does not overfit to specific subsets of the data and can generalize well across the entire dataset.

To ensure that the random splitting process can be consistently replicated (which is crucial for scientific reproducibility and consistent experimentation), a manual seed is set for the random number generator (torch.manual_seed(100)). By setting this seed, every time the split is executed, the same samples will be allocated to the training and validation sets, ensuring consistent results across multiple runs.

In summary, splitting the dataset into training and validation subsets is a foundational step in the machine learning workflow. It provides a mechanism to train the model and subsequently validate its performance, ensuring that it's capable of making accurate predictions on data it hasn't encountered during the training phase.

2.4.3 Hyperparameter tuning

In the realm of machine learning and deep learning, hyperparameter tuning plays an instrumental role in optimizing model performance. Hyperparameters can be perceived as the external knobs of an algorithm that, when adjusted, can directly influence the learning process. Unlike model parameters which are learned during training, hyperparameters are predefined by practitioners. The chosen values can have significant impacts on both the model's training dynamics and its final performance. Here, we elucidate the selected hyperparameters for the DistilBERT model and expound on their significance:

2.4.4 Batch size

The batch size pertains to the number of training examples utilized in one iteration of model update. A batch size of 32 means that in each training step, 32 samples from the training dataset are used to compute the gradient and update the model's weights. A moderate batch size, like 32, strikes a balance between computational efficiency and the gradient's variance. It allows the model to generalize better compared to using the entire dataset (batch gradient descent) or a single instance (stochastic gradient descent).

2.4.5 Dropout

Dropout is a regularization technique conceived to mitigate the problem of overfitting in neural networks, a phenomenon where a model becomes excessively tailored to the training data and loses its generalization capability on unseen data. The central idea behind dropout is deceptively simple but profoundly impactful: during training, randomly selected neurons (or, in the case of transformers, attention weights) are temporarily "dropped out" or deactivated at each iteration, meaning they do not contribute to the forward pass and do not partake in backpropagation. This ensures that no single neuron or group of neurons becomes overly specialized or dependent on its neighboring neurons, promoting a more distributed and redundant representation.

The rate at which neurons are dropped, commonly referred to as the dropout rate, is a crucial hyperparameter. For instance, a dropout rate of 0.4 means that 40% of the neurons in the dropout layer are turned off during a particular training pass. Importantly, during model evaluation or inference, dropout is not applied, and all neurons are used. Instead, the outputs are typically scaled based on the dropout rate to ensure that the expected sum remains consistent between training and testing. By introducing this element of randomness and preventing intricate co-adaptations, dropout drives individual neurons to be more robust and to seek out salient features, resulting in a more resilient and generalizable model.

2.4.6 Optimizer, learning rate and epsilon

At the core of any deep learning model's training process lies an optimizer. It is an algorithm that adjusts the model's internal parameters, such as weights and biases, in a manner that minimizes the loss function. The loss function essentially quantifies how far off the model's predictions are from the actual data. By using the optimizer, we iteratively refine the model, steering it towards better performance by adjusting its parameters in response to the gradient of the loss function. The gradient provides information on the direction and magnitude of changes needed for each parameter to minimize the error. Thus, the optimizer's role is crucial—it dictates the manner and speed at which a model learns from data.

AdamW is an iteration on the widely-used Adam optimizer. Adam, which stands for "Adaptive Moment Estimation," combines the benefits of two gradient descent methodologies: AdaGrad and RMSProp. It maintains a running average of both the gradients (first moment) and the squared gradients (second moment). By computing adaptive learning rates for each parameter, Adam often converges faster and requires less fine-tuning of the learning rate compared to other optimizers.

AdamW introduces a modification in the way weight decay (a form of regularization) is applied in Adam. Traditional weight decay is applied directly to the weights, which can conflict with the adaptive nature of the learning rates in Adam. AdamW decouples the weight decay from the optimization steps, making the weight decay more "correct" and consistent, especially in scenarios where learning rates are dynamic.

For tasks like binary classification, it's crucial that the optimizer is sensitive to nuances in the data and can rapidly converge to an optimal solution. AdamW, with its adaptive learning rates, efficiently handles sparse gradients, which are common in large-scale and high-dimensional datasets, such as text data used for sentiment analysis or other binary classification problems. Furthermore, the weight decay component of AdamW serves as a regularization mechanism, preventing overfitting, especially in scenarios where the dataset might be imbalanced or when the distinction between the two classes is subtle. In essence, AdamW's combination of rapid convergence and effective regularization makes it a prime choice for binary classification tasks.

The learning rate is one of the most pivotal hyperparameters in neural network training. Conceptually, it controls the step size taken during the optimization process as the model traverses the loss landscape. The loss landscape can be imagined as a multidimensional space where each point represents a specific configuration of the model parameters, and the height of that point corresponds to the loss value. The goal is to find the lowest point in this landscape, analogous to a ball settling in the deepest valley.

In each iteration of training, the model computes the gradient of the loss with respect to its parameters. This gradient points in the direction of the steepest increase in the loss. To reduce the loss, the model updates its parameters by moving in the opposite direction of the gradient. Here, the learning rate dictates how far this move should be.

A high learning rate may cause the model to take large steps, possibly overshooting the optimal point and oscillating around it. This can lead to unstable training or even divergence, where the loss grows without bound.

A low learning rate ensures that the model takes small, cautious steps. This can lead to more stable convergence, but it may also mean that training is slower, or in some cases, the model might get stuck in a local minimum, which isn't the best possible solution.

Finding the right balance for the learning rate is crucial. Too high, and the model may never converge; too low, and it might take an impractically long time to train or fail to capture the best model.

In the context of certain optimization algorithms, especially those that involve division by the accumulated gradient, epsilon is a small, positive constant added to prevent potential division by zero or numerical instability. This is crucial for maintaining the robustness of the algorithm, ensuring that it doesn't produce undefined or extremely large values during optimization.

For optimizers like Adam and AdamW, the denominator when updating parameters is an accumulation of past squared gradients. There's a rare chance, especially early in training or with sparse data, that this accumulated value becomes very small. Dividing by an extremely small number can lead to very large weight updates, destabilizing the training process. By adding epsilon to the denominator, we ensure the updates remain within a reasonable range, safeguarding the training dynamics.

2.4.7 Model building

The algorithm initiates by setting specific configurations, hyperparameters, and ensuring reproducibility of results. This reproducibility ensures that the consistency in experiments is maintained across different runs with the same initial conditions.

Training proceeds in epochs, with each epoch representing a comprehensive forward and backward pass of all training examples. The iterative nature of this structure enables the model to incrementally learn and adjust its parameters using the entire dataset multiple times. Within each epoch, the model processes the training dataset in smaller segments termed as batches. For every batch, the model predicts outcomes based on its current parameter state.

Subsequent to these predictions, the deviation between the model's predictions and the actual outcomes is quantified using a loss function. This loss serves as a metric, indicating the extent to which the model's predictions align with the true labels. To enhance the model and minimize this deviation, the gradients of this loss concerning each model parameter are computed using the backpropagation algorithm. These gradients essentially represent the direction and magnitude of changes required for each parameter to reduce the loss.

An optimization algorithm then leverages these gradients to adjust and refine the model's parameters, thereby ensuring that with each subsequent epoch, the model's predictions come increasingly closer to the actual outcomes.

Following the training phase in each epoch, the model's performance is gauged against a validation dataset. This set, distinct from the training data, offers an unbiased evaluation metric. The model predicts outputs for this dataset, and several accuracy metrics, including but not limited to loss, precision, recall, and F1 score, are computed. Each of these metrics provides a unique perspective into the model's performance, ranging from its overall accuracy in predictions to its capability in identifying and detecting positive samples correctly.

Upon the conclusion of all epochs, a comprehensive summary of the model's evolution and performance across epochs is provided. This trajectory provides insights into the model's learning capability and its adeptness at making accurate predictions on novel, previously unseen data.

In an endeavor to derive insights from monetary policy speeches, the DistilBERT model was subjected to a series of fine-tuning procedures. Observing the metrics across ten epochs provides insight into the model's learning trajectory and its ability to generalize.

Consistently decreasing training loss, which moves from 0.68 to 0.60 over the epochs, indicates a robust learning curve. The decline in validation loss, albeit marginal, demonstrates the model's competency in making predictions on unseen data. However, it's worth noting that as the epochs increase, the descent in validation loss plateaus, hinting at the model reaching its optimal state with the given data and hyperparameters. Prolonging the training further might render the model susceptible to overfitting, thereby compromising its predictive capacity on new, unseen data.

Diving deeper into the model's performance metrics, the validation accuracy plateaued at 0.65. In the context of classification tasks, precision and recall offer a granular perspective. The attained peak precision of 0.81 during the sixth epoch suggests that out of all positive predictions, 81 percent were true positives. Complementarily, the corresponding recall value underscores that out of all actual positive instances, 81% were correctly predicted by the model. Such closely aligned values of precision and recall often allude to a model that doesn't disproportionately err on the side of false positives or false negatives. The F1 Score's peak value at the sixth epoch further reaffirms this balance, indicating a harmonious trade-off between precision and recall.

Integrating dropout at a rate of 0.4 served as an invaluable tool in this study. By periodically deactivating 40 percent of the neurons during training, the model was nudged to develop a more holistic understanding of the data, rather than becoming overly reliant on specific neurons or patterns. This inherently reduced the model's vulnerability to overfitting, ensuring a more generalizable performance.

One notable observation from the experiment was the pronounced sensitivity to the learning rate. While the model's responsiveness to other parameters remained relatively stable, the learning rate's impact was pronounced. This accentuates the intrinsic challenge in deep learning models: identifying an optimal learning rate that neither impedes convergence nor causes the model to overshoot the minimum.

The labeling strategy, although simplified to 'negative' and 'positive', was a double-edged sword. While it facilitated a more streamlined experimental environment, it simultaneously introduced potential ambiguities. Expanding the label taxonomy could potentially pave the way for richer insights and finer-grained predictions.

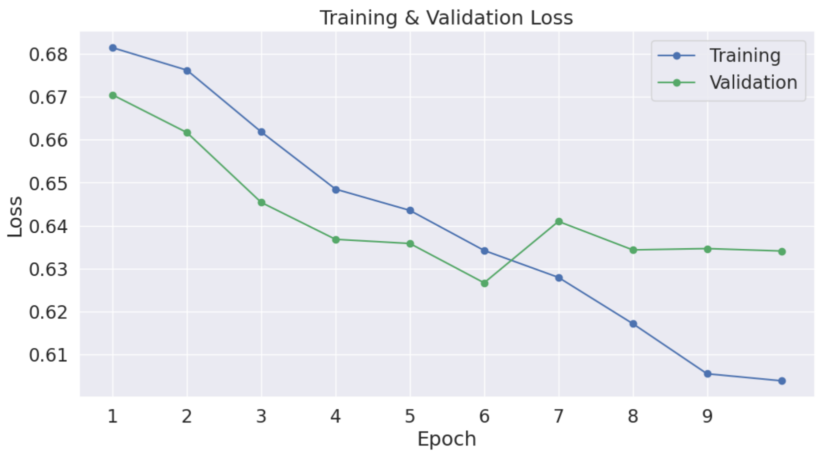
To encapsulate, while the DistilBERT model demonstrated commendable performance on monetary policy speeches with the provided dataset and configuration, several avenues for enhancement have been elucidated. The findings emphasize the intricate interplay of hyperparameters, data quality, and model architecture in achieving optimal results.

3 Results and Discussion

The endeavor to fine-tune the DistilBERT model, tailored specifically for monetary policy speeches, highlights the potential of modern neural networks in the domain of financial linguistics. This experiment is reflective of the broader shift in the field towards leveraging deep learning models to understand and predict policy impacts through the nuanced language of speeches.

To optimize the model, standard practices were rigorously followed. Tokenization and padding, essential in maintaining input consistency and in ensuring that each input sequence into the model has a uniform length, were seamlessly incorporated. The pivotal step, however, was the precise setting of hyperparameters. The choice of the AdamW optimizer, renowned for its capability in efficiently handling sparse gradients and non-stationary objectives, became foundational for the subsequent training process. A learning rate of 0.00001 and an epsilon of 1e-8 were chosen, striking a balance between rapid convergence and model stability.

The batch size and maximum token length were set at 32 and 500, respectively. These choices are often a result of trade-offs — between computational efficiency and the granularity of data representation. The table detailing training results over ten epochs provides a comprehensive overview of the model's progression. An evident downward trend in both training and validation losses suggests the model's growing adeptness in capturing patterns within the data. The validation loss showed a gentle decline from 0.67 in the initial epoch to 0.63 by the 10th epoch. This indicates the model's improved generalization capabilities over successive iterations.



But mere loss values, though indicative, aren't exhaustive in understanding a model's performance. The validation accuracy, which commenced at 0.59, showed a modest improvement, reaching 0.64 by the end of training. The precision, recall, and F1 score metrics provide a more granulated perspective. A notable leap in these metrics, particularly between the 2nd and 3rd epochs, underscores the model's sharpened ability to differentiate between the 'negative' and 'positive' classes.

The introduction of a dropout rate, set at 0.4, was a strategic move to curb overfitting. In neural network training, overfitting can often masquerade as improvement, misleadingly suggesting that the model is performing well by merely memorizing the training data. Dropout, by periodically "turning off" 40% of the neurons during training, ensures a more generalized learning process, reducing the model's propensity to over-rely on specific training data features.

Several potential factors, however, could pose challenges to the outcomes. The learning rate's sensitivity warrants an intricate and possibly adaptive tuning approach. Although other parameters demonstrated stability, one must approach their settings with a dose of cautious optimism, keeping in mind the idiosyncrasies of the data in question.

The dataset, split into 1,709 training and 570 validation entries, was labeled under the binary 'negative' and 'positive' categories. While simplifying the labeling process, this dichotomy might not capture the multifaceted sentiments embedded in monetary speeches. This suggests an avenue for refinement, urging the expansion of label categories to better encapsulate the diverse sentiments these speeches might harbor.

In encapsulation, this experiment has underscored the aptitude of the DistilBERT model in processing and understanding monetary policy speeches. The metrics, their trends, and the observed outcomes all point towards promising avenues for future research. With refinements in hyperparameter tuning, dataset enhancement, and label sophistication, there lies a promising horizon for more nuanced insights and richer interpretations in subsequent studies.

4 Conclusion and Heuristics

The study embarked on the journey of utilizing DistilBERT by fine-tuning it with monetary policy speech data. Through the course of the research, a range of tools and techniques from classroom teachings were integrated, making extensive use of platforms like HuggingFace and leveraging the computational power offered by Google Colab for GPU support. Yet, despite these efforts, the resulting model exhibits some clear limitations.

A primary constraint faced during the project was in the realm of data acquisition and labeling. Given the limited resources and time constraints, the majority of the data was collected and labeled manually. Minimal automation was introduced in the data gathering phase. This approach, while pragmatic given the constraints, has underscored the need for more robust data collection and labeling methodologies. It's believed that with a more rigorous data sourcing mechanism, the model's accuracy and efficiency could be further enhanced.

In terms of computational resources, while Google Colab played a pivotal role in facilitating GPU support, there remains a vast potential for more advanced models when backed by superior computational infrastructure. The model's design allows it to be scalable, meaning that as more data becomes available or as computational resources are upgraded, the model can be adapted accordingly. Consequently, it is imperative to perceive this model as a foundational effort that sets the stage for more advanced iterations in the future.

Starting with a basic understanding of Natural Language Processing (NLP), the study has significantly expanded the knowledge frontier in this domain. An attempt was made to simulate how a full-fledged NLP model might operate in an authentic production environment. This exercise provided insights into the versatility of NLP, the multifaceted problems it can address, the nuances of project planning, and the strategic utilization of accessible tools and resources.

Furthermore, the study underscored the significance of model training. Emphasis was placed on optimizing the training process, making effective use of computational resources, and exploring the vast landscape of hyperparameter tuning. By employing various evaluation metrics, the model's performance was critically assessed, thereby shedding light on areas of improvement.

In sum, this project served as a valuable learning experience, underscoring the intricacies of machine learning and NLP. For the enthusiasts of machine learning, the findings and experiences from this study offer a comprehensive introduction and broad skillset that can potentially be useful for future.

5 Table

	Training Loss	Valid. Loss	Valid. Accur.	Precision(V)	Recall(V)	F1 Score(V)	Training Time
1	0.68	0.67	0.59	0.48	0.69	0.57	0:00:26
2	0.68	0.66	0.60	0.48	0.69	0.57	0:00:23
3	0.66	0.65	0.62	0.76	0.77	0.76	0:00:23
4	0.65	0.64	0.63	0.71	0.73	0.71	0:00:23
5	0.64	0.64	0.64	0.80	0.81	0.80	0:00:23
6	0.63	0.63	0.65	0.81	0.81	0.79	0:00:23
7	0.63	0.64	0.63	0.72	0.69	0.70	0:00:23
8	0.62	0.63	0.64	0.77	0.77	0.77	0:00:23
9	0.61	0.63	0.64	0.77	0.77	0.77	0:00:23
10	0.60	0.63	0.64	0.77	0.77	0.77	0:00:23