Risk Evaluation of Tweets Content using BERT

Brezuleanu Mihai Alexandru

January 16, 2024

Abstract

This paper addresses the challenge of Risk Evaluation, focusing on a natural language processing classification task. The objective is to accurately assess the risk associated with textual content of tweets covering various topics. The implementation relies on BERT (Bidirectional Encoder Representations from Transformers), a Google-developed NLP model known for its context-aware understanding of language. The bidirectional approach of BERT proves effective in text classification, as demonstrated in both the Benchmark Performance and Ablation Study components.The article highlights the impact of these approaches, presenting public and private scores, execution times, and their correlation with model variants. Notably, the pre-trained BERT model outperforms, achieving higher accuracy and efficiency, making it the preferred choice for the competition's final ranking.

1 Introduction

This article aims to solve the problem of Risk Evaluation within Annual International Data Science & AI Competition 2023. This consists of a natural language processing (NLP) classification task, where the goal is to accurately assess the risk associated with each tweet from a test dataset. Training dataset consists of over 25,000 tweets, organized into "Tweets" column which contains the textual content of tweets covering various topics like product reviews and customer experiences, but also a second column called "Risk Analysis" which indicates the risk associated with each tweet. A "0" in this column means no risk, while a "1" indicates the presence of risk.

In our implementation we relied on BERT(Bidirectional Encoder Representations from Transformers), a natural language processing (NLP) model developed by Google which excels at understanding the context of words in a sentence by considering both the words that come before and after each target word. For our text classification task, this bidirectional approach proved to be a good choice, in terms of the results and metrics obtained both in the **Benchmark Performance** component and in the **Ablation Study**.

Regarding the two previously mentioned components, the approaches used were different. Thus, within the **Benchmark Performance** component, we used a model with weights whose values were randomly initialized. This resulted in a lower performance, but good relative to the final ranking, since this aspect is not specified. On the other hand, within the **Ablation Study** component we used a pre-trained model, the results compared to the previous approach, being significantly better.

Therefore in the first phase, using a model without pretrained weights, we obtained a public score equal to **0.90000** and a private score equal to **0.89725**. Regarding the execution time, **3535.5** seconds were recorded, using the GPU P100 accelerator in a kaggle notebook. In the Ablation Study component, where we used a pre-trained BERT model, we obtained a public score equal to **0.95652** and a private score equal to **0.95509**. The execution time obtained was **1651.5** seconds, in the same environment.
The differences are noticeable both in terms of execution time and final accuracy. Thus, the variant that ranked, obviously, on a better place in the final ranking, was the one with pre-trained weights.

2 Competition and dataset

The 4th International Competition in Data Science & Artificial Intelligence, known as the World Championship in Data Science & Artificial Intelligence, is being hosted by The International Society

of Data Scientists. It has been designed to serve as a dynamic arena where young Data Scientists can expand their knowledge base, gain valuable hands-on experience, and foster collaborative efforts. The competition addressed in this article is called Risk Evaluation.

The dataset on which the competition is based comprises information pertaining to CVS Health customers in the USA, spanning from February 1, 2022, to August 1, 2022. It encompasses over 25,000 tweets, organized into two columns. The first column is called "Tweets".This column encompasses the textual content of tweets related to CVS customers. Each record corresponds to an individual tweet posted on the platform. The text covers a range of topics, including product reviews, customer experiences, and general discussions about CVS.

The second column is called "Risk Analysis". It denotes the risk analysis associated with each tweet. The values in this column signify the risk status linked to each tweet, with "0" indicating no risk and "1" denoting the presence of risk. A risk status suggests occurrence of an issue, or the possibility of a negative impact on the company's competitiveness.

2.1 Rankings of the competition

Competition Risk Evaluation started on Oct 1, 2023 and ended on Nov 1, 2023, during this time period 54 submissions being registered by different participants . More precisely, 13 submissions registered a score lower than 85%, 17 submissions had a score between 85%-95% and 23 submissions had a score above 95%. It should be mentioned that this last category includes scores up to 0.96924, this being the best result. Considering the difference compared to the best score, we can see that the results obtained in this article were some very good ones.

#	Δ	Team	Members	Score	Entries	Last	Solution
1	▲ 10	NTPhuonggggggggg		0.96924	15	3mo	
2	▼ 1	TheEyes 🕶		0.96900	53	3mo	
3	—	Davnchi		0.96843	38	3mo	
4	▲ 2	PogChamp		0.96547	13	3mo	
5	—	pogba paul		0.96426	12	3mo	
6	▲ 1	childhood wish		0.96412	32	3mo	
7	▼ 5	abo zekry		0.96408	50	3mo	
8	▼ 4	CTDTKS AI		0.96291	34	3mo	
9	—	Se7enista		0.96216	27	3mo	
10	▼ 2	Leo		0.96115	12	3mo	
11	▲ 20	JRPC		0.96095	6	3mo	
12	▼ 2	Rùa Non :))))))))))		0.96030	5	3mo	
13	▲ 6	Word Whisperers		0.95964	21	3mo	

Figure 1: Ranking of the competition

2.2 Statistical analysis

Performing statistical analysis on the provided datasets is a crucial step in the data science pipeline. This preliminary exploration helps in understanding of the characteristics of our data, identifying patterns, and making informed decisions about the subsequent steps in our analysis. For this, we extracted some particularities of the data set. Thus, for the column "Tweets" we processed "Length of a tweet", "Number of words inside each tweet", "Number of sentences inside a tweet" but also the "Mean length of words in a tweet".

Based on the exploratory visualizations(fig:2), it can be concluded that the average length of a tweet falls within the range of 0 to 300 characters. Although tweets exceeding 300 characters exist, they are relatively rare occurrences. Similarly, the distribution of the number of words in tweets indicates that most tweets contain between 0 to 60 words, with a noticeable decline in frequency beyond the 60-word threshold. This observation suggests that very lengthy tweets are less prevalent.

Another visualization(fig:3) provides insights into how the variable 'Length of a tweet' is distributed across various risk categories in the training data. Thus, analyzing this information, we can draw the conclusion that tweets with a longer length (more than 250 characters) tend to present a higher degree of risk, while the majority of short tweets is not a risk most of the time.

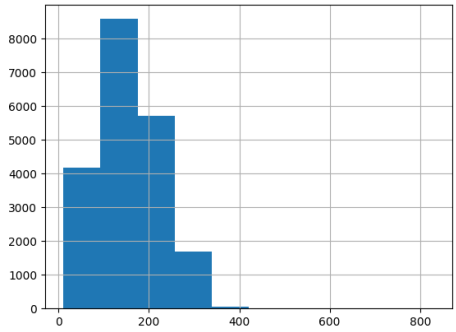

Figure 2: Length of tweets

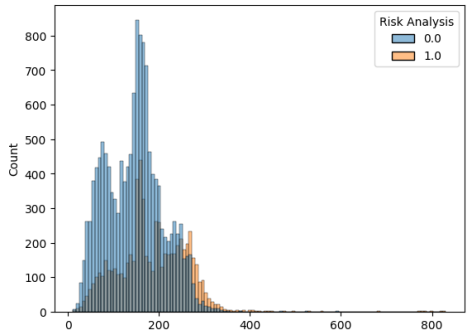

Figure 3: Risk Analysis based on lengths

3 Literature Review

For our article, we conduct a literature review, exploring similar papers that serve as sources of inspiration and ideas. This process allows us to build upon existing research and draw insights from related works in the field. The first analyzed article is entitled **BERT-based conformal predictor for sentiment analysis**[4]. It examines the effectiveness of both the original BERT model and the suggested ICP extension using the Large Movie Review dataset, which comprises 50,000 movie reviews. Their findings indicate that the strong performance exhibited by the base classifier seamlessly extends to the ICP without significant accuracy reduction. Furthermore, the generated prediction sets are sufficiently precise, making them practical for real-world applications. Another article we referred to is **Target-Dependent Sentiment Classification with BERT**[2]. It aims to validate whether BERTs context-aware representation can yield comparable performance enhancements in aspect-based sentiment analysis. Unexpectedly, integrating BERT with intricate neural networks that previously excelled with embedding representations does not consistently demonstrate significant value. Conversely, the inclusion of target information consistently exhibits improved accuracy, and the experiments reveal the most effective means of leveraging this information.

4 Implementation

Once the training and testing datasets received in csv format(train_data.csv and test_data_post.csv) have been read, we define a function called extract_features to process and extract additional features from the tweet data. Therefore we extract both for test and training data:

- tweet_length: represents the number of characters in each tweet.
- word_count: store the number of words in each tweet. It counts the number of space-separated words in each tweet.
- sentence_count: represents the number of sentences in each tweet. It counts the number of sentences separated by periods (".").

- mean_word_len: calculated by dividing the tweet_length by the word_count. This gives the mean length of a word in each tweet.

After features extraction, two subsets of our initial training data are created: one for actual training (train_set) and another one for validation (valid_set). The test_size parameter is set to 0.2, meaning that 20% of the data is allocated for validation, while the remaining 80% is assigned to the training set. The random_state parameter is again set to 42 for reproducibility.

4.1 Tokenization

Tokenization is a crucial step in text preprocessing, its purpose being to break down raw text into manageable units such as words or subwords(known as tokens), making it easier to process and analyze.

In our implementation we used the BertTokenizer from the Hugging Face transformers library to tokenize text data. This is commonly done when preparing text data for input to BERT-based models.

More exactly, we created an instance of the BERT tokenizer (bert_tokenizer) using the pre-trained model 'bert-base-uncased'. This pre-trained model is commonly used for tasks involving English text. Next we, define a function get_encodings that tokenizes the text data in the 'Tweets' column of the dataset using the BERT tokenizer and returns the tokenized representations with truncation, padding, and conversion to PyTorch tensors. This representations are suitable as input to BERT-based models for tasks like natural language processing and sentiment analysis.

4.2 Standardization

Standardization is a common preprocessing step in machine learning, especially when features in the dataset have different scales. Therefore, for the previous extracted features from our datasets("tweet_length", "word_count","mean_word_len", "sentence_count") we used StandardScaler class. Its main purpose is to transform the data so that it has a mean of 0 and a standard deviation of 1. This is achieved by applying the following formula to each feature: $z = \frac{x - \text{mean}}{\text{std deviation}}$, where x is the original value of the feature, *mean* is the mean of the feature values, and *std deviation* is the standard deviation of the feature values.

StandardScaler is applied to the selected features in the training, validation, and test sets to ensure that these features have similar scales for subsequent tasks.

4.3 Model

We are defining a custom PyTorch model named MyCustomBertModel which combines a BERT model with additional features and a custom classification head. In the __init__ method, we define the components of our model. Here, we included a BERT model for sequence classification (BertForSequenceClassification) and set the num_labels parameter to 256. Also, we create a custom classification head using a Sequential module. It consists of a linear layer with input size 256 + len(extracted_features) (combining BERT output and additional features), followed by a ReLU activation function, and another linear layer with output size 2, considering that the competition assumes a binary classification.

In the forward method, we define the forward pass of your model. Here, we pass the input BERT-related information (input_ids and attention_mask) from the input x to the BERT model and obtain the logits. After, we concatenate the BERT embeddings (bert_emb) with additional tabular data (x["tab_data"]) along the specified dimension (dim=1). Finally, the concatenated features are passed through the custom classification head (self.head), producing the final output of the model.

4.4 Training

Before moving on to the training loop of our model, we define data loaders for training, validation, and test datasets. Data loaders are essential for efficiently loading and batching data during the training process. They are based on BERT token encodings, standardized features, and labels. For the training and validation data loader we used a batch size of 32, while for testing the batch size is 128. Also, the DataLoader for training will randomly shuffle the samples at the beginning of each epoch.

As criterion for training the classification model, we used torch.nn.CrossEntropyLoss().. It combines the softmax activation function and the negative log-likelihood loss. It is commonly used for multi-class classification problems where each input can belong to one class out of many.

The chosen optimizer is AdamW, an extension of the Adam optimizer with weight decay. Weight decay is a regularization technique that penalizes large weights to prevent overfitting. The parameters of the model (my_bert_model.parameters()) are optimized using this optimizer. The learning rate(optim_lr) is a hyperparameter that determines the step size during optimization. It is a crucial parameter that influences the convergence and stability of the training process. Weight decay(optim_weight_decay) is a form of L2 regularization applied to the optimizer. It helps prevent overfitting by adding a penalty term to the loss function based on the magnitude of the model weights. In our implementation the learning rate was 2e-5 while the weight decay was set to 5e-5.

4.4.1 Epoch Training

An epoch refers to one complete pass through the entire training dataset. During an epoch, the model sees and learns from every example in the training dataset once. The training process is typically divided into epochs to ensure that the model has the opportunity to learn from the entire dataset multiple times.

The function in question, iterates over batches provided by the training data loader, preparing inputs for the model, such as BERT-related inputs (input_ids, attention_mask), additional tabular data (tab_data), and ground truth labels (labels). Next, it clears the gradients before each backward pass and gets the model predictions by passing inputs through the model.

After computing the loss using the specified criterion (criterium) and ground truth labels, the training function performs backpropagation to compute gradients, and updates the model parameters using the optimizer. Finally, the average loss over all batches is returned.

4.4.2 Evaluation

The primary purpose of the evaluation function is to provide insights into how well the model generalizes to unseen data and to measure its performance. Similar to the training function, it iterates over batches provided by the validation data loader, preparing inputs for the model. Next, it disables gradient computation during the forward pass to save memory during evaluation and passes the input data through the model to get the outputs.

In order to get the predicted labels, it detaches the output tensor from the computation graph, moves it to the CPU, and converts it to a NumPy array. This is done to facilitate further processing. Further, the predicted probabilities for the positive class are extracted and the loss is calculated.

Finally, the average loss over all batches is returned along with Area Under the Curve (AUC) of the Receiver Operating Characteristic(ROC) curve as the evaluation metrics.

4.4.3 Prediction

The purpose of this function is for making predictions for a binary classification task. It behaves much like the evaluation function, except the fact that it applies softmax activation to the model outputs along the last dimension(class dimension). This this will effectively return the predicted probabilities for each class.

4.4.4 Training Loop

The training loop, iterating through each training epoch, calls the training process to compute and store the training loss. Subsequently, the model's performance is evaluated on the validation set using the *eval* function with the BERT model and the validation data loader. This evaluation process yields both the validation loss (loss_valid) and the area under the ROC curve (auc_roc).

A mechanism to save the best model is introduced to capture improvements in the AUC-ROC score on the validation set. If the current AUC-ROC surpasses the best AUC-ROC (best_auc), the model is deep-copied to optim_model, and the best_auc and best_epoch variables are updated.

The code further incorporates an early stopping condition, where training halts if there is no improvement in performance for a specified number of epochs (early_stop_patience). The early stopping mechanism is included to prevent overfitting and improve efficiency by stopping training when validation performance plateaus. It can also save computational resources by avoiding unnecessary training epochs.

4.5 Performance

Before computing the metrics, predicted probabilities and the true labels are defined on the validation set for later evaluation.

Next, a list of 100 evenly spaced thresholds between 0 and 1 are generated. For each threshold in this list, predicted probabilities(y_pred) are converted into binary predictions using the threshold and computes the F1 score by comparing these predictions with the true labels (y_true).

Then, we used a NumPy's convolution function to apply a moving average to the elements in the array of F1 scores. This operation computes the moving average of the F1 scores, where each element in the result is the average of itself and its neighboring elements according to the specified window. This smoothing operation is commonly used to reduce noise and highlight trends in a sequence of values, making it easier to identify patterns or variations in the data

The smoothed F1 score sequence is then used to determine the threshold that maximizes the F1 score (optim_threshold) and identifies the best F1 score (best_f1). This threshold optimization process is valuable for finding an optimal balance between precision and recall in binary classification tasks.

Finally, we applied the best BERT model to generate predictions on the test dataset. The instances are classified into high or low risk based on the previously determined optimal threshold.

5 BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers, is a transformer-based deep learning model introduced by Google in 2018. BERT has had a significant impact on natural language processing (NLP) and has become a foundational model for various NLP tasks.[5]

5.1 BERT tokenizer

Tokenizers play a dual role in natural language processing. First, they break down raw text into smaller units, referred to as tokens, operating at the word, subword, or character level. Secondly, these tokens are coverted into a numerical format that can be comprehended by machine learning models.

BERT utilizes a subword tokenization algorithm in its tokenizer. Subword tokenization strikes a favorable balance between vocabulary size and sequence length, offering improved handling of rare and out-of-vocabulary words without the need to treat them as unknown tokens.[1]

Specifically, BERT employs the WordPiece subword tokenization algorithm, which constructs a vocabulary of tokens up to a specified size. This process involves selecting and merging character pairs in the text dataset based on their score. The score is determined by the ratio of the frequency of the pair to the product of the frequencies of the individual elements: $\text{SCORE} = \frac{\text{pair\_freq}}{\text{first\_elem\_freq} \times \text{second\_elem\_freq}}$

The input text undergoes tokenization using the BERT tokenizer, resulting in a sequence of tokens. Each distinct token within the vocabulary is given a unique integer ID. This ID acts as a condensed representation for a one-hot encoded vector.

A one-hot encoded vector has the length equals to the total number of tokens in the text dataset's vocabulary. All elements are 0 except, the element at the index corresponding to the represented token which has value 1. The purpose of encoded vectors and integer IDs is to select the corresponding row from a lookup table.

Once a sample of raw text is passed to the tokenizer, processed text(string tokens), along with three distinct integer encodings are produced. String tokens also consist of 3 special categories:

- Classification Token[CLS]: Positioned at the beginning of a sequence, the token serves as a representation for the entire sequence.
- Separation Token[SEP]: This token is appended to the end of each sentence within a sequence, in order to facilitate BERT in distinguishing both sentences and their boundaries.
- Padding Token[PAD]: This token is added at the ind of sequence in order to extend its length, ensuring uniform sequence lengths across all samples in the batch. This uniformity is crucial for the parallel processing of multiple samples in a batch.

In terms of integer encodings, after processing, the raw text produces the following categories:

- Input IDs: The BERT base model employs a vocabulary of 30,522 tokens. Each unique string token is mapped to a unique integer ID. These integer IDs represent the tokens in a given sequence.
- Attention Mask: Sequences shorter than the defined maximum length are padded with tokens serving as placeholders. To ensure these padding tokens don't affect the model output, an attention mask is applied. It assigns the value 1 at the indices of non-padding tokens and 0 to elements at the indices of padding tokens.
- Token Type IDs: Are closely related to the task of next-sentence prediction, where sequences are constructed as 2 sentences. These designate the belonging of tokens to either the first or second sentence. For sequence classification, all tokens are treated uniformly, assigned a token type ID of zero. In this context, token type IDs don't influence the model output during sequence classification.

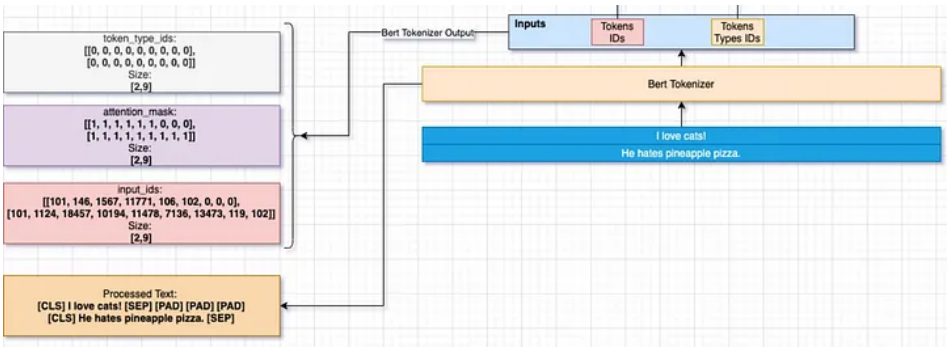

Figure 4: BERT tokenizer

5.1.1 Embedings

Embeddings serve as vector representations of tokens, effectively organizing them in a multidimensional space. Tokens with similar meanings are ideally situated closer together, while those with dissimilar meanings are positioned farther apart.[3]

The dimensionality of the embedding vector, a design choice, determines the space in which the embeddings reside. A larger space provides more room for organizing tokens. In practice, embedding lengths are often set relatively high to accommodate extensive vocabularies and capture the nuances of contextual meaning. For instance, BERT base utilizes an embedding length of 768.

Therefore, the encodings generated by the tokenizer are transferred to the Embedding module. Here, the input IDs and token type IDs are employed to retrieve their respective embeddings through a lookup operation.

Three types of embeddings are generated for each token and then aggregated:

- Word Embeddings: Is obtained by replacing each input ID with the corresponding row vector from the word embedding lookup table. The number of table rows will be equal to the number of unique tokens in the vocabulary (maximum 30522).
- Token Type Embeddings: It follows the same principle as word embeddings, but with only two rows in this lookup table. These two rows correspond to the two sentences in the sequence. However, it will not be relevant for our task.
- Positional Embeddings: Provide the model with a sense of token order in the sequence. Positional IDs retrieve the corresponding positional embeddings from the lookup table. The positional IDs are consistent for each sequence, forming a vector of consecutive integers from 0 to the maximum sequence length (512 is the limit).

Combining these three embeddings consolidates diverse information aspects into a unified representation. This composite embedding encompasses details regarding the **semantic significance** of

tokens in the sequence, the **positional relationships of tokens** in a sequence on an element-wise basis, and the **context of the sentence** in which the tokens reside (useful only in the context of next-sentence prediction).

5.1.2 Encoder

The BERT Encoder comprises 12 transformer layers, with the output from one layer sequentially influencing the subsequent one. Having more layers in the encoder generally enables the model to grasp more intricate patterns and relationships within the text.

In the lower layers, the model may focus on local and syntactic relationships, while the middle layers delve into sentence-level semantics. In contrast, the upper layers are more inclined to capture document-level relationships.

Each layer is structured with two primary sub-layers: multi-headed self-attention and the position-wise feed-forward network.

Multi-Headed Self-Attention

Multi-Headed Self-Attention mechanism allows the model to evaluate the significance of individual words within a sequence by considering their relationships with all other words. Through this process, the embedding vector associated with each token gains a richer understanding of its connections to the surrounding tokens.

Its necessity arises because tokens sharing the same ID possess identical embeddings, regardless of their context within the sequence. While variations based on position in the sequence (utilizing the position embedding component) exist, this alone is insufficient(ex: same word has completely different meanings).

Self-attention effectively addresses such nuanced relationships, including more complicated and higher-order connections. This is achieved by systematically comparing each token to every other token in the sequence. Consequently, the embedding vector for each token is enriched with pertinent relational and contextual information, allowing for a more nuanced representation that captures the diverse meanings a word may take on in different contexts.

Contextual enrichment involves passing an embedding vector through three distinct linear layers independently. Consequently, three new vectors(Query, Key, Value) are generated, each sharing the same shape as the input embedding vector.

This is achieved through matrix multiplication with their respective Weight matrices (W). Since there are multiple tokens in a sequence, we will have query, key, and value matrices (Q, K, V) where each row corresponds to a vector for a token.

Matrix multiplication facilitates the calculation of the dot product as a measure of similarity between vectors with all others simultaneously. Therefore, in the resulting QK.T matrix, each row vector corresponds to the same token in the query matrix (Q), and each element within a row vector tells us how similar that corresponding token in Q is to each token of the same sequence in K. Several other operations are applied to these scores, including setting padding token to negative infinity, and applying the softmax function. After applying the softmax function, the padding tokens will receive a weight of zero.

Finally, the QK.T weight matrix is multiplied with the value matrix (V), so that each element represents what degree a token is paying attention to the other tokens in the sequence.

Incorporating multiple attention heads enables the model to concentrate on diverse relationships and dependencies within a sequence, resulting in more enriched token representations.

Position-Wise Feed-Forward Network

The Position-Wise Feed-Forward Network is a crucial element within a transformer layer, contributing additional learnable parameters to enhance the model's capacity and supporting the ongoing refinement of token representations throughout the encoder's layers.

The term "position-wise" signifies that the transformation is individually and uniformly applied to each token, or in simpler terms, to every position in the sequence.

The sequence of operations in the position-wise feed-forward sub-layer involves: **Projection to a Higher Embedding Size**, **GeLU Activation Function**, **Projection back to the Original Embedding Size**, **Dropout**, **Skip Connection** and **Layer Normalization**.

5.1.3 Pooler

The encoder's output forms a matrix where each row represents an enhanced representation of the corresponding token in the input sequence. On the other hand, for sequence classification, a singular vector representation capturing the overall meaning of the entire input sequence is required, not just an individual token.

The chosen approach is to simply extract the representation of the first token in the input sequence, identified as the classification token. Originally crafted during the pretraining phase with a focus on next-sentence prediction, the [CLS] token is designed to encode the relationship between two sentences. It is also anticipated to contain a contextualized, high-level representation of the entire sequence, making it a suitable candidate for pooling in the context of sequence classification.

Classification Head

Lastly, the output from the pooler undergoes the classification head, a step involving the projection of the pooled embedding into a space with a dimensionality matching the number of distinct classes. Termed "head" means that this component is interchangeable to accommodate specific tasks. The predicted class is determined by selecting the maximum value from the resulting logits, where each class is associated with a single value.

6 Analyzed metrics

The performance of a machine learning model refers to its ability to accurately and effectively accomplish the task for which it was designed. Assessing and understanding the performance of a model is a fundamental aspect of the machine learning workflow. Metrics on the other hand, play a crucial role in assessing the performance of a machine learning model, providing a quantitative and objective measure of how well the model is accomplishing its intended task. Choosing appropriate metrics depends on the nature of the task. For this classification tasks we used F1 score, Receiver Operating Characteristic (ROC) curve and its Area Under the Curve (AUC).

F1 score

The F1 score is a metric commonly used in machine learning, especially in classification tasks, to assess the balance between precision and recall. Precision is the ratio of correctly predicted positive observations to the total predicted positives, indicating how many of the predicted positive instances are actually relevant. Recall defines the ratio of correctly predicted positive observations to the total actual positives. It measures the model's ability to capture all relevant instances of the positive class.

The F1 score is calculated as the harmonic mean of precision and recall and provides a more comprehensive evaluation than accuracy alone. It is particularly useful in situations where both false positives and false negatives are costly or have significant consequences. Also, it can be influenced by the choice of the classification threshold.

Receiver Operating Characteristic (ROC) curve

The Receiver Operating Characteristic (ROC) curve is a graphical representation of the performance of a binary classification model across different classification thresholds. It illustrates the trade-off between the true positive rate and the false positive rate at various threshold values. The ideal scenario is represented by a point in the top-left corner of the ROC space (0, 1), indicating a model with perfect sensitivity (captures all positive instances) and no false positives.

Area Under the Curve (AUC)

The Area Under the Curve (AUC) is a widely used metric in the context of Receiver Operating Characteristic (ROC) curve. It is a scalar value that quantifies the overall performance of a binary classification model. A higher AUC indicates better discriminative ability of the model across different threshold values.

7 Benchmark Performance

In this section of our article, we opted to use a variant of BERT (Bidirectional Encoder Representations from Transformers) model without pre-trained weights. This decision holds significance in the context of our study and contributes to the transparency and robustness of our benchmarking process. The decision not to use pre-trained weights means that we initialized the model with random weights instead of leveraging knowledge gained from a pre-training phase on a large corpus. This approach can be beneficial in scenarios where the specific nuances of our task might differ significantly from the pre-training data. By training the model from scratch, we tailor its parameters to the specifics of our sequence classification task.

However, training a model from scratch is computationally more expensive and time-consuming compared to fine-tuning a pre-trained model. As you can see the execution time(fig:5) is significantly higher than what we will get when we use pre-trained weights. This is due to the longer training time until overfitting occurs.

Run	Private Score	Public Score	Best Score
3535.5s - GPU P100	0.89725	0.90000	0.89725 V1

Figure 5: Benchmark Performance

Another disadvantage is the accuracy, as can be seen in the figure 6. Pre-trained models learn hierarchical and meaningful feature representations that capture syntactic and semantic structures. On the other hand, models without pre-training might lack these optimized feature representations, leading to suboptimal performance.

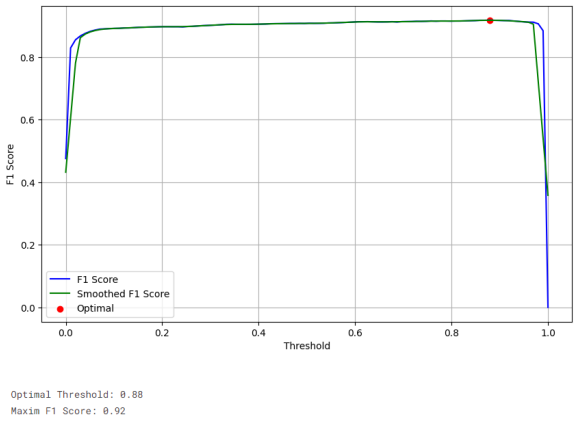

Figure 6: Benchmark Performance

Analyzing the results, our solution would rank 32nd in the final ranking(fig:7). However, this positioning is not very relevant because, in the final ranking, it is not specified whether or not pretraining was used. Most high-scoring solutions used this approach.

28	Duy Nguyens		0.95556	5	3mo
29	Javohir Toshqurgonov		0.95495	1	3mo
30	test_re		0.95226	3	3mo
31	JRPC		0.95048	6	3mo
32	Ricardo Haus		0.89050	25	3mo
33	Guilherme Gobbo		0.88948	3	3mo
34	bangnashih		0.87890	4	3mo

Figure 7: Benchmark Rankings

8 Ablation Study

In this section of our article, we conducted experiments using BERT (Bidirectional Encoder Representations from Transformers) with pretrained weights. This choice carries several implications and advantages. One consists in the fact that, being a pre-trained model, BERT has already learned rich contextualized representations of language from vast and diverse corpora. Leveraging these pretrained weights provides our model with a strong foundation in understanding general language patterns and semantic relationships.

The pretrained model, having already learned valuable features from extensive datasets, significantly reduce the training time required for our specific task. Also, leveraging the wealth of information embedded in pretrained weights allows our model to generalize well to the nuances of our specific prediction task, leading to more robust and accurate outcomes, as can be seen in the figures 8 and 9.

Run	Private Score	Public Score	Best Score
1651.5s - GPU P100	0.95509	0.95652	0.95509 V1

Figure 8: Ablation Study

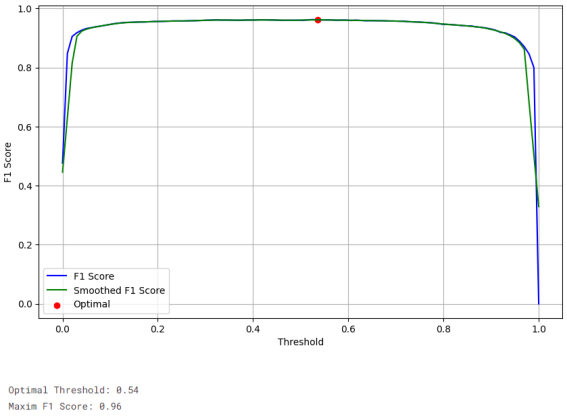

Figure 9: Ablation Study Metrics

Analyzing the obtained results we can notice a significant improvement in accuracy compared to the previous solution that does not use pretraining. Regarding the ranking of the competition, this approach would rank 20th(fig: 10), being a very good position considering the distance from the best result which is 0.96924.

16	▼ 2	Manav Trivedi		0.95559	2	3mo
17	▲ 4	l9m595		0.95548	6	3mo
18	▼ 2	Thomas		0.95543	6	3mo
19	▼ 7	FortunesCookies		0.95531	4	3mo
20	▲ 5	samye		0.95457	2	3mo
21	▼ 4	Abhi Patel		0.95345	2	3mo
22	▼ 2	Rahmads		0.95316	5	3mo
23	▼ 5	[Deleted] 0ad64ce3-75e5-4f65-b76c-bdd27b497abb		0.95226	5	3mo

Figure 10: Ablation Study Rankings

9 Conclusion

In conclusion, our investigation into the Risk Evaluation problem at the Annual International Data Science & AI Competition 2023 highlights the efficacy of employing the BERT model. A comparison of two approaches—randomly initialized weights versus pre-trained BERT—reveals the latter's superior performance in terms of accuracy, as indicated by higher public and private scores. Notably, the pre-trained BERT model also demonstrates efficiency gains, significantly reducing execution times. This emphasizes the importance of pre-training in enhancing NLP model performance, establishing it as the preferred choice for tweet risk assessment tasks and similar challenges in the field of data science and AI.

References

[1] Shivaji Alaparthi and Manit Mishra. Bert: A sentiment analysis odyssey. *Journal of Marketing Analytics*, 9(2):118–126, 2021.

[2] Zhengjie Gao, Ao Feng, Xinyu Song, and Xi Wu. Target-dependent sentiment classification with bert. *Ieee Access*, 7:154290–154299, 2019.

[3] MV Koroteev. Bert: a review of applications in natural language processing and understanding. *arXiv preprint arXiv:2103.11943*, 2021.

[4] Lysimachos Maltoudoglou, Andreas Paisios, and Harris Papadopoulos. Bert-based conformal predictor for sentiment analysis. In *Conformal and Probabilistic Prediction and Applications*, pages 269–284. PMLR, 2020.

[5] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline. *arXiv preprint arXiv:1905.05950*, 2019.