MSc Natural Language Processing – 2022 - 2023
UE 705 – Supervised Project

Evaluating the effectiveness of sBert and miniLM on analogy classification with FrameNet

Students:	**Supervisors:**
Omar ABEDELKADER	Miguel COUCEIRO
Ekaterina KOZACHENKO	Esteban MARQUER
Juba AIT ABDELMALEK	Pierre MONNIN

Université de Lorraine

June 22, 2023

Contents

Introduction

Analogies play a fundamental role in human cognition and reasoning, enabling us to understand relationships between concepts and make inferences based on similarities. With the advent of natural language processing (NLP) and semantic representation frameworks, the exploration of analogies in textual data has garnered significant attention. In this study, we delve into the task of identifying valid and invalid analogies within the context of the FrameNet dataset, utilizing the power of state-of-the-art models, namely sentence-BERT (SBERT) and a fine-tuned MiniLM model.

The FrameNet dataset serves as an invaluable resource for capturing the semantics of lexical units within specific frames. By associating words with their syntactic and semantic roles, FrameNet enables us to discern the contextual relationships and meanings expressed in various scenarios. Leveraging this rich lexical database, we aim to distinguish between valid and invalid analogies, advancing our understanding of linguistic structures and their underlying conceptual associations.

To approach this task, we employ the sentence-BERT (SBERT) model as a powerful sentence embedding technique. SBERT encodes sentences into continuous vector representations, preserving semantic similarities among sentences. By leveraging this embedding, we can effectively compare and measure the semantic relatedness between the components of an analogy, discerning valid correspondences from those that lack conceptual coherence.

Furthermore, we explore the fine-tuning of a MiniLM model, leveraging the annotated analogies within the FrameNet dataset. The MiniLM model, trained on this annotated data, aims to refine the classification task by learning from the labeled examples and capturing the intricate patterns that distinguish valid and invalid analogies. Through this fine-tuning process, we seek to enhance the model's accuracy and its ability to generalize across different analogy instances.

Preliminary experiments have shown promising results, with the fine-tuned MiniLM model achieving an impressive 99% accuracy in differentiating between valid and invalid analogies within the FrameNet dataset. This high level of accuracy suggests the effectiveness of our approach and motivates further investigation into the nuances of analogy comprehension and representation learning.

In this study, we provided a comprehensive analysis and comparison of the performance of the SBERT and fine-tuned MiniLM models, exploring their strengths and limitations in handling the task of analogical reasoning. Additionally, we examined the impact of dataset size, model architecture, and other factors on the overall performance and generalizability of the models.

In conclusion, our aim is to contribute to the field of NLP and semantic understanding by tackling the task of distinguishing valid and invalid analogies using the rich knowledge encoded within the FrameNet dataset. By harnessing the power of advanced models such as SBERT and fine-tuned MiniLM, we strive to deepen our understanding of analogical reasoning and pave the way for improved applications in various domains, including question answering systems, semantic search, and intelligent dialogue agents.

Chapter 1

Definition of the problem

Machine learning is a popular topic which is constantly evolving and as such in modern times using analogical proportions has begun to be used in order to benefit this field. Semantic role labeling has also begun to be a popular area of research, with many applications from text generation to translation.

Human intelligence relies on humans to consider relationships between things, rather than focus on individual entities (Lim et al., 2021). As neural networks were first created by Frank Rosenblatt, specifically the perceptron, to be modeled after the human brain it is not a far fetched thought to believe that they could learn in a similar manner to humans. When comparing two sentences, finding similarities between them can rely on more than just an understanding of the individual components of the sentence, but also rely on a more holistic understanding of it. Cosine similarity is generally used to determine the similarity of sentences, but by using analogies a more sturdy understanding of these sentences could be formed. By combining these two aspects of machine learning and computation, this paper hopes to achieve the following task:

- Create a model which uses analogies to determine if sentences belong to certain frames from the FrameNet knowledge graph.

A CNN based model will be used in order to complete this task, and compared against each other. Two different embedding types will be used. The embedding being used will be SBERT and MiniLM which will be compared against each other. Also for both embedding techniques the model will be tested un-tuned and fine-tuned. In total there will be 4 different models to compare at the end of the experiment. The input in a CNN model passes through several convolution layers followed by pooling layers until the desired depth is obtained. This is then input into fully connected dense layers then through the activation function which will be a sigmoid function in this case.

SBERT will be used in order to convert the sentences into dense vectors which will be passed into the model. Compared to other embedding models which produce sparse vectors, SBERT's dense vectors carry more of the meaning of a sentence as a whole, by carrying the contextual meaning of words and their use in the sentence. It operates by adding an additional layer of embedding on the traditional BERT method, which converts the vectors into a sentence vector with a modified pooling method. It creates a vector which holds the semantic meaning of the sentence which was generated.

MiniLM tokenizes the words in the sentences, then passes the token through a pre-trained neural network in order to create fixed length vectors for each sentence, capturing semantic meaning in these generated vectors.

1.1 Task to address

In order to complete the task of creating a model which can determine the frame to which a sentence belongs through analogy we must first formally define what a valid analogy is. An analogy exists as a:b::c:d such that the reaction between a and b is identical to the relationship between c and d. In this case the analogies which will be looked at will be in the form $\mathrm{S_A{:}S_B{::}S_C{:}S_D}$.

In order to classify the analogies into the two classes of valid or invalid, where the valid analogy must pass certain propositions (Prade and Richard, 2021).

$$f(S_\mathrm{A}, S_\mathrm{B}, S_\mathrm{C}, S_\mathrm{D}) = 1 \quad \text{where} \quad (S_\mathrm{A}, S_\mathrm{B} \in F_\mathrm{AB} \quad \text{and} \quad S_\mathrm{C}, S_\mathrm{D} \in F_\mathrm{CD})$$

A model will need to be created to complete this task, which can then be compared to state of the art models for effectiveness. A binary classifier can be used to determine whether an analogy is valid or not, outputting 1 for a valid analogy and 0 for an invalid analogy. Different parameters and types of neural networks will be tested in this model to determine which is the most effective for the FrameNet knowledge graph.

1.1.1 Classification

For this the data will be passed into the bi-lstm as a tuple consisting of ($\mathrm{S_A{:}S_B{::}S_C{:}S_D}$) which will then be tested for the validity of the analogy. This will return a binary output which will determine whether or not the analogy is considered valid by the model.

$$f(S_\mathrm{A}, S_\mathrm{B}, S_\mathrm{C}, S') = 0 \quad \text{where} \quad (S_\mathrm{A}, S_\mathrm{B} \in F_\mathrm{AB}, S_\mathrm{C} \in F_\mathrm{CD} \quad \text{and} \quad S' \in F')$$

Incorrect analogies will be generated from the frames being used, and these will be compared against the correct analogies. A random frame will be chosen from the list of frames, which is different to F_{AB} and F_{CD}. The incorrect analogies will be generated by using a random sentence of F', which will be used instead of S_D, which will be referred to as F' for the frame and S' for the sentence. This will create an analogy which will be considered invalid.

In order to determine how valid an analogy is a sigmoid activation function could be used, where whether or not an analogy is valid could be given a certain threshold. If an analogy reaches that threshold it would be considered valid. It could also be used in future research as well when determining strong and weak analogies by setting out further thresholds. This activation function also allows for further tweaking based on the analogy being measured where what is considered valid or not could be compared against other similar analogies using the same frames, and what is considered valid in these models.

1.1.2 Analogies

Analogies are employed in FrameNet to create semantic links between frames based on their shared or comparable traits. By revealing parallels between the underlying conceptual structures of different frames, analogies aid in the identification and description of relationships between them. These comparisons can offer insightful information about how meaning is organized and represented in FrameNet.

Based on the nature of the relationship between word pairs, analogies are categorized and grouped. Ancient Greeks were the ones who originally employed proportions, and they applied them to the world of numbers. Arithmetic and geometric proportions are two instances that are noteworthy (Couceiro et al., 2017). The analogical proportion statement, "A is to B as C is to D," is shown by these two examples:

1. A, B, C, D are proportional if $A - B = C - D$ (arithmetic proportion);
2. A, B, C, D are proportional if $\frac{A}{B} = \frac{C}{D}$ (geometric proportion).

Statements of the form "A is to B as C is to D" (usually denoted as $A : B :: C : D$) are analogical proportions. A relationship between two things is equated with another relationship by an analogical proportion. The analogous proportion "A is to B as C is to D" so presents an analogy of proportionality by asserting that the differences between the two items A and B, which are otherwise comparable, are the same as the differences between the two objects C and D which are similar in certain aspects (Prade and Richard, 2014).

An analogical proportion illustrates similar relationships between different pairs. For example, the analogical proportion "A calf is to a bull as a foal is to a stallion" implies the same dissimilarities exist between both pairs - calf and bull, foal and stallion, indirectly referring to a calf as a young bovine. All items in an analogical proportion share a category, here animals, making the analogy robust to changes in sequence. Hence, "A stallion is to a bull as a foal is to a calf" can be deduced from the original analogical proportion (Barbot, N., Miclet, L., Prade, H. 2019).

Analogical Proportions are a quaternary connection that must abide by the following three postulates: $\forall$ a, b, c, d $\in$ X (Lepage, 2004).

1. A : B :: A : B (reflexivity);
2. A : B :: C : D $\rightarrow$ C : D :: A : B (symmetry);
3. A : B :: C : D $\rightarrow$ A : C :: B : D (central permutation).

From these properties, we can infer the following equivalent resultant analogy which is A : B :: C : D.

Given a valid A: B::C: D for the classification, if the analogy of A:B is proportional to the analogy of C:D then it is valid. if the analogy of A:B is not proportional to the analogy of C:D then it will be invalid.

1.2 Database

For this project we used FrameNet, a knowledge graph created in the International Computer Science Institute (ICSI) in Berkeley, California (Fillmore et al., 2004). This lexical database offers a wealth of information for comprehending word meanings in terms of semantic frames. A semantic frame is a conceptual organization that captures the significance of a word or set of words in relation to a certain event or context.

The FrameNet dataset is a corpus of annotated sentences that associate certain words or phrases with particular frames. Each frame has a set of lexical units (words or phrases) that are related to that frame and reflects a certain semantic notion or scenario. The dataset contains details on the frame, its definition, and the lexical units that make up its set. In this chapter, we present the dataset and describe some of the preprocessing carried out before the project's implementation phase.

The key part of FrameNet which is being focused upon for this task is the sentences which demonstrate the meaning of the frame. The lexical units are not of importance in this case as they are not being used. Instead the sentences are being used as a whole. Firstly to prepare the data, the sentences are lemmatized. The sentences are lemmatized in order to remove noise, as well as keeping the vocabulary small. This will be useful in improving the accuracy of the model.

After the words are lemmatized, the next step is to prepare the analogies. Valid analogies and invalid analogies need to be generated to be used in training and testing the model. From the 1222 frames, there are 2869 sentences which appear in the data, which can be used for this task. From these 2869 sentences, there are over 67 billion different analogies, in the form A:B::C:D, which could be generated. The issue here is that the vast amount of these analogies would be invalid, leading to a highly imbalanced data set. Only roughly 123 million analogies would be valid, which is only slightly less than 0.000002% of the total amount of analogies. For this reason the amount of analogies generated should be reduced, as well as for the reason that the volume is too much to be processed in a timely manner by the machine being used.

If every valid pair of sentences is generated, that being in the form A:B, 11122 pairs are created. To match this a similar number of invalid pairs are generated, that being in the form A:B, where A and B are from different frames. Rather than choosing a random selection of invalid pairs, this is chosen methodically, in order for a better representation of all sentences to appear, in order to reduce the chances of overfitting occurring. A similar method again of methodically choosing pairs is used to reduce the number even further, to reduce the amount of analogies for the sake of being manageable by the machine, is done. This leaves the amount of analogies at slightly more than 300,000.

These analogies are then used to create further analogies, by means of central permutation. Central permutation (A:B::C:D $\rightarrow$ A:C::B:D) being a postulation which all analogies must be able to perform in order to be considered valid, means that while the validity of already valid analogies will not change, the same can be said about invalid analogies. The aim of this is for the model to receive a more holistic understanding of analogies, by being able to recognize even analogies which have been transformed. This will be applied to each of the analogies which have been created, leading to over 600,000 analogies for the model to be trained on. After all these steps have been completed the data is ready to be vectorized for training.

Chapter 2

Methodology

2.1 Transfer Learning

A machine learning technique called transfer learning includes applying the knowledge learned from one problem's resolution to enhance performance on subsequent, related, or similar problems. The process of moving knowledge or learned representations from one activity to another is, in other words, what is meant by this. In conventional machine learning, models are typically created from scratch and trained using a big dataset on a particular task. However, this method needs a sizable amount of labeled data as well as processing power. While labeled data is scarce, transfer learning, on the other hand, makes use of prior knowledge from a previous task or domain to increase learning effectiveness and generalization on a new task.A machine learning technique called transfer learning makes use of insight acquired from solving one problem to enhance another.

The process of transfer learning usually involves the following steps:

- Pre-training: To learn broad features or representations that can be helpful for a variety of tasks, a model is trained on a large dataset, often using a supervised learning approach. This pre-training phase is frequently carried out on a sizable dataset, such as ImageNet for computer vision tasks or a sizable corpus of text data for tasks requiring natural language processing. The model gains the ability to distinguish between lower-level features seen in earlier levels, such as edges or textures, and higher-level features found in later layers, such as forms or objects.
- Fine-tuning: After pre-training, the model is fine-tuned or modified using a smaller, more targeted dataset. This dataset typically differs from the pre-training dataset and includes labeled samples pertinent to the current issue. The pre-trained model's later layers are expanded or updated to better suit the new task while the model's earlier layers are frozen or kept fixed during fine-tuning. The model can learn task-specific features faster and with less labeled input by using the previously learned information.

2.2 Benefits of Transfer Learning

Transfer learning has been used to build highly accurate and effective models with little labeled data in a number of fields, including computer vision, natural language processing, and speech recognition. Transfer learning offers several benefits:

- Improved performance with limited data: A big dataset used for pre-training enables the model to acquire broad representations that capture useful patterns, improving performance with little input. When labeled data is scarce, these learnt representations can then be refined on a smaller dataset, producing superior performance than starting from scratch.
- Faster convergence: The pre-trained model has already absorbed fundamental features and structures, which might serve as a useful starting point for the new task and speed up convergence. The model can converge more quickly by fine-tuning from the pre-trained model because it just needs to learn the features related to the job at hand.
- Generalization: Transfer learning aids in the acquisition of more abstract, higher-level elements that are relevant to a variety of tasks or domains. The acquired representations may be more broadly applicable and adapt effectively to fresh, unexplored material.
- Reduced computational resources: The computational resources needed for training are drastically reduced compared to developing a model from scratch since transfer learning uses a pre-trained model.

The pre-trained model must have been trained on a sizable and diverse dataset, and the new task must be connected to or share certain characteristics with the original task for transfer learning to be effective. The transfer may not be as effective and it may be better to train a model from scratch if the pre-training task and the new task are very dissimilar.

2.3 Data Preparation

In activities involving machine learning and natural language processing, data preparation is essential. The performance and accuracy of the models are significantly influenced by the quality of the input data. We will go over the data preparation procedure for a FrameNet dataset obtained from Esteban Marquer in this post. We will specifically look at how to use and operate a preprocessing function that is given to get the data ready for more analysis and modeling.

2.3.1 Data Preprocessing

The provided function, named "create_csv_line" is designed to preprocess and prepare the data of frames in the FrameNet dataset. It takes a list of strings, referred to as "final" as input. The function consists of several steps to clean and transform the data for subsequent analysis.

2.3.2 Data Gathering

Initially, the function extracts relevant information from the input list "final". It iterates through each element of the list, searching for specific patterns to identify relevant attributes. The function looks for the presence of the string "lu =" to determine the type of attribute. Based on the attribute type, the corresponding value is extracted and assigned to the appropriate variable. In this case, the function identifies the attributes "Frame", "Agent", and "Activity" and assigns their respective values to "frame", "agent", and "activity" variables.

2.3.3 Data Cleaning

Once the relevant attributes are extracted, the function proceeds to clean the data. It employs regular expression (regex) operations to remove unwanted characters and symbols from the strings. The regex pattern `[/\^\w\s]` is used to match any non-alphanumeric character or whitespace. These unwanted characters are replaced with a space, effectively removing them from the text. This step ensures that the text is in a consistent format and removes potential noise that could hinder further analysis.

2.3.4 Tokenization and Lemmatization

The cleaned text strings are tokenized using the word_tokenize function of the Natural Language Toolkit (NLTK) module. Tokenization divides the text into tokens, simplifying the processing steps that follow. The tokenized words are then lemmatized using the WordNet lemmatizer from NLTK. By simplifying or dictionarying terms, lemmatization normalizes the text and reduces the dimensionality of the feature space. In this case, lemmatization is carried out with a focus on verbs, demonstrating the intention to pay attention to the verb forms of the words.

2.3.5 String Concatenation and Formatting

The various tokens are reunited to create a single string after undergoing the tokenization and lemmatization processes. In order to preserve the contextual meaning of the words within the string, the processed tokens are concatenated using spaces as separators. The final representation of the text is made suitable for later analysis and modeling thanks to this concatenation. The preprocessing function which is offered, exemplifies a thorough method of data pretreatment for FrameNet datasets. It then extracts the necessary characteristics, cleans the data with regex operations, normalizes the text with tokenization and lemmatization, and concatenates the cleaned tokens into a single string. The output string represents the original data in a preprocessed manner and is prepared for additional analysis, feature extraction, and model training. To maximize the efficiency and quality of subsequent machine learning or natural language processing activities, proper data preparation is crucial.

2.4 Data Arrangement

After preprocessing, we prepared a dataset with two columns ”fName” and ”text” having 2884 samples. Now we created two other empty datasets “df_data_v” having columns ”A” and ”B” and ”df_data_iv” having columns ”C” and ”D”. We added a code snippet that involves the comparison of rows in a dataframe and the subsequent appending of relevant data. The code iterates through the rows of a given dataframe and identifies matching values in a specific column. Upon finding a match, it appends the corresponding values from two other columns to a separate dataframe. The execution time of this process is also measured using the ”time” module.

We demonstrated a nested loop structure, which is used to iterate through the rows of a dataframe named ”df”. The purpose of this iteration is to compare each row with every other row in the dataframe. The outer loop iterates through each row using the ”iterrows()” function, which returns both the index and the row content. Within this loop, the inner loop is also utilized, iterating over the rows of the same dataframe. During each iteration of the inner loop, the code checks if the value in the first column of the current row (row[0]) matches the value in the first column of the row from the outer loop (row1[0]). If a match is found, it indicates that the rows contain related or similar data. Upon identifying a matching pair of rows, the code appends the values from two specific columns (row[1] and row1[1]) to a new dataframe named ”df_data_v”. The ”append()” function is employed with the relevant values and the ”ignore_index=True” parameter, which ensures the appended data receives a new index. The code utilizes the ”%%time” magic command, which is specific to Jupyter Notebook or IPython environments. This command allows for the measurement of execution time. By placing this command at the beginning of the code cell, it records the time taken for the entire code block to execute. We added the same code for second dataset but this time we incorporated a conditional sentence that verifies the value of the variable ”count.” The conditional block is executed if ”count” is 670 or higher. The adding of data is limited by this condition based on a predetermined threshold. The code compares the value in the first column of the current row (row[0]) with the value in the first column of the row from the outer loop (row1[0]) when the condition is satisfied. If the two values do not equal one another, there is a different relationship. In these circumstances, the code uses the ”append()” method to append the values from two specified columns (row[1] and row1[1]) to a new dataframe called ”df_data_iv.” The code raises the value of ”count” by one to maintain a count of consecutively comparable rows if the values in the first columns of the comparison rows are equal. The conditional block resets thanks to this approach after a predetermined number of consecutively comparable rows, enabling further comparisons and potential data adding.

After this, we created a new dataset named "df_data" consisting of four empty columns "A", "B", "C" and "D". We used a nested loop structure to cycle through the rows of the dataframe "df_sample_v." In order to iterate over the rows and provide both the index and the row content, use the "iterrows()" function. We used the "append()" method within the nested loop to append data to a dataframe called "df_data". A new row is added to the dataframe for each pair of rows (row and row1) acquired through the nested iteration. This new row includes the values from the columns of the preceding rows as well as a new column called "value" that has the value set to 1. The appended rows with the columns "A", "B", "C", "D" and "value" are included in the resulting dataframe, "df_data". The values from the first and second columns of the current row are stored in the columns labeled "A" and "B," respectively, whereas the values from the first and second columns of row1 are stored in the columns labeled "C" and "D" respectively. For each row that is appended, the 'value' field is set to 1.

We then used a nested loop structure once more to cycle through the rows of the two dataframes "df_sample_v" and "df_sample_iv." For both dataframes, the "iterrows()" function is used to get the rows and their indices. We used the "append()" method within the nested loop to append data to the dataframe "df_data". A new row is added to the dataframe for each pair of rows (row and row1) acquired through the nested iteration. This new row has the values from the columns of the preceding rows along with a new column called "value" that has the value 0 in it. Finally we saved the dataframe into a csv file named "df_data.csv".

2.5 Data Visualization

In order to improve comprehension, analysis, and decision-making, data visualization refers to the depiction of data in visual representations, such as charts, graphs, and maps. In a formal and professional setting, this paper intends to examine many facets of data visualization, including its significance, approaches, and advantages. Data visualization is essential for presenting complex information in a way that is intelligible and interpretable on a visual level. It makes it possible for stakeholders to draw out important insights from data, spot patterns, and spot trends and anomalies. Data visualization improves comprehension, facilitates good decision-making, and helps to communicate information to both technical and non-technical audiences by converting raw data into aesthetically pleasing and comprehensible visualizations.

2.5.1 Class Distribution

A class distribution graph is a type of visualization that shows how classes or categories are distributed within a dataset. The ability to recognise and comprehend class imbalances is especially useful when dealing with categorization issues. Class distribution graphs, their significance, and their use in data analysis and machine learning are all well explained in this study.

The number or percentage of examples that belong to each class is plotted on the y-axis, and the classes themselves are displayed on the x-axis, to generate a class distribution graph. Depending on the specific visualization method selected, the graph may take different visual shapes, such as bar charts, pie charts, or line plots.

We plotted a class distribution graph of the dataset that has been prepared in the above section. This graph shows the number of records and percentage of data samples belonging to each class.

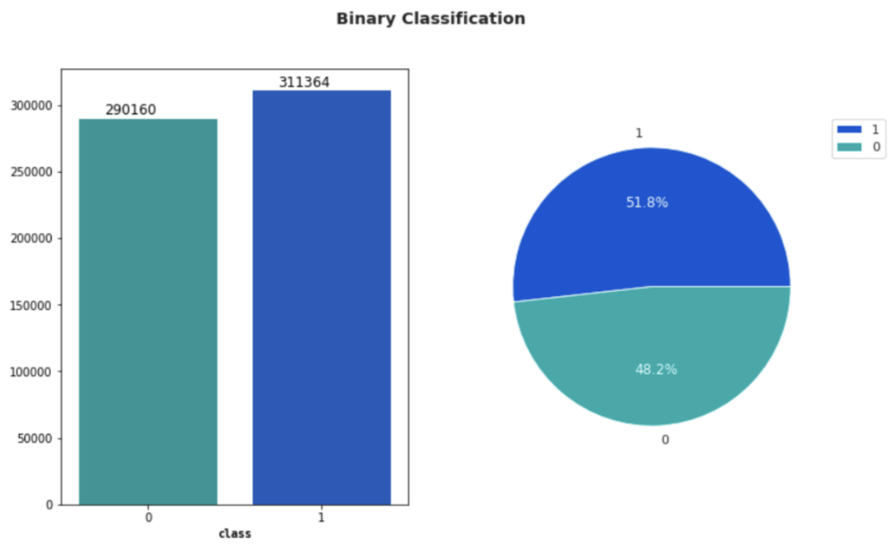


Fig.1: Class distribution graph of the dataset.

2.6 Data Extraction

The description of the FrameNet dataset is given in the following table.

Contents	**Quantity**
Number of samples	601524
Number of variables	5
Number of samples (Valid)	290160
Number of samples (Invalid)	311364

Table 2.1: Description of samples of dataset

In the above table, it can be observed that the dataset that has been finally prepared is large and has a total of 601524 samples. We extracted two small datasets from the large dataset each having 30076 samples. First dataset named "df_d1" has 15535 samples belonging to valid class and 14541 samples belonging to invalid class. The second dataset has 15492 samples belonging to valid class and 14584 samples belonging to invalid class. This is because the large dataset takes a lot of resources and time to train on a Transfer Learning model. We are limited in case of time and resources so we extracted reasonable data which can easily be trained on normal resources within a specific time deadline.

2.7 Architecture Definition

According to the definition, architecture describes the precise layout and composition of a computer model or system. The performance and abilities of models are greatly influenced by architecture in the field of natural language processing (NLP). Sentence-BERT (SBERT) and MiniLM, two well-known NLP architectures, are thoroughly explained in this research. We will examine their structural characteristics, underlying ideas, and the ways in which they have improved natural language creation and processing.

2.7.1 SBERT

A variant of the well-known BERT (Bidirectional Encoder Representations from Transformers) concept, SBERT was created by Reimers and Gurevych (2019). In order to enable sentence embeddings that are semantically relevant, it integrates siamese and triplet network topologies. With self-attention mechanisms, many layers, and transformer-based architecture, SBERT's architecture effectively captures contextual information. SBERT, is a potent framework for creating sentence embeddings. The SBERT algorithm learns fixed-length representations of complete sentences as opposed to conventional word embeddings.

By fine-tuning pre-trained transformer models like BERT on certain tasks, such as sentence similarity or paraphrase identification, this is accomplished.

A two-step procedure is used to train SBERT. BERT is initially trained on vast amounts of unlabeled data to discover contextualized word embeddings. In the second step, supervised tasks like semantic textual similarity or paraphrase identification are used to fine-tune a siamese or triplet network. SBERT learns sentence-level embeddings that encode semantic information through this fine-tuning procedure.

Semantic search, document retrieval, and text classification are just a few of the NLP tasks for which SBERT has demonstrated its high level of efficiency. It provides sentence similarity comparison, making duplicate detection, clustering, and recommendation systems easier to complete.

2.7.2 MiniLM

MiniLM is a condensed version of the well-known BERT model that was first presented by Wang et al. in 2020. It focuses on lowering computing complexity while keeping performance competitive. In comparison to BERT, MiniLM's architecture uses fewer layers, parameters, and self-attention heads, making it lighter and more effective.

MiniLM uses a pre-training and fine-tuning paradigm, like BERT. Masked language modeling (MLM) and next sentence prediction (NSP) objectives are used by MiniLM in the pre-training phase in order to make use of a huge corpus of unlabeled text data. In order to respond to certain downstream tasks, the model is then fine-tuned using task-specific labeled data.

NLP applications have shown the effectiveness of MiniLM, especially when computational resources are scarce. It has been used effectively for applications including named entity identification, language development, and text classification.

2.7.3 Comparative Analysis of BERT and MiniLM

Being an extension of BERT, SBERT inherits its intricate design and needs a lot of processing power for both training and inference. However, MiniLM overcomes this drawback by providing a small architecture that lowers computing requirements without dramatically lowering performance. The performance of SBERT and MiniLM has been impressive across a number of NLP benchmarks. While MiniLM performs well in situations when computational resources are constrained, SBERT offers highly discriminative sentence embeddings.

2.8 Training

2.8.1 Model Training with SBERT

The ”distilbert-base-nli-stsb-mean-tokens” pre-trained model was used to initialize the SBERT model after importing the required libraries. The sentences (A, B, C, and D) were then extracted from a DataFrame (designated as ”df”) and added to a list called ”sentences.” The sentences are subsequently encoded and sentence embeddings are produced using the SBERT tokenizer. ”embeddings” is a NumPy array where the final embeddings are kept. The labels are taken out of the DataFrame (”df”) and put into a NumPy array called ”labels”. The input-output dataset for training the model is a TensorDataset that combines the sentence embeddings and labels. A DataLoader is created from the input dataset to enable effective loading of data in batches during training. There are a certain amount of data samples in each batch. The torch.nn.Module class is used to define a unique binary classification model. It comprises a number of linear layers with dropout regularization and intermediate ReLU activations. Using the specified architecture, a binary classification model instance is produced. The shape of the sentence embeddings is the model's input. For the model's training, we specified the optimizer (Adam) and loss function (cross-entropy). During the training process, these elements decide the objective and the optimisation algorithm. We looked to see if a CUDA-capable GPU was available, otherwise it uses the CPU. We trained the model for 10 epochs. During each epoch, the model is put in training mode. The code iterates over the DataLoader, zeroing the gradients, forwarding the inputs through the model, calculating the loss, performing backpropagation, and updating the model's parameters. At the end, we have also recorded training loss and accuracy.

2.8.2 Fine tuning with SBERT

For a particular objective, we concentrated on optimizing a Sentence-BERT (SBERT) model. Several modifications have been made to the prior BERT model in order to accommodate the search for the ideal parameters. To achieve this, The best parameters discovered during the search are defined to be stored in a dictionary called best_params. It has fields for validation accuracy, optimizer, loss function, batch size, and learning rate. For the purpose of storing validation accuracies for various parameter combinations, a list called ”val_acc_list” is made. Learning_rates, batch sizes, optimizers, and loss_fns are a few of the defined parameter search spaces. We used stacked loops to iterate through all permutations of the provided parameter search spaces. The binary classifier model is recreated inside the loops using the current parameter setting. The best parameters are updated if the current combination yields a higher validation accuracy than the previous best. After the parameter search is complete, the best parameters, including the learning rate, batch size, optimizer, loss function, and validation accuracy, are stored in the best_params dictionary. After performing the fine tuning on SBERT and storing the values of best parameters, we trained the SBERT again with these best parameters and stored the accuracy on validation data.

2.8.3 Model Training with MiniLM

A dataframe with the name df that is a subset of df_d2 and contains a portion (5%) of the original data is loaded. "microsoft/MiniLM-L12-H384-uncased" is the model name used to load the MiniLM model and tokenizer. The labels variable stores the long-typed labels that were extracted from the dataframe. The tokenizer concatenates and tokenizes the sentences from the dataframe, and then creates a CustomDataset instance with the tokenized sentences and labels. For batch-wise training, the CustomDataset object is encased in a DataLoader. For the model parameters, an optimizer (AdamW) and a learning rate scheduler (StepLR) are defined. When a GPU is available, the model is transferred to that device. Ten training epochs are run in a loop. The model is put in training mode for each epoch. The inputs and labels are transported to the device, and the data is iterated over in batches. The model's forward pass is computed using the inputs and labels after the gradients have been cleared. Gradients are back propagated through the model after the loss has been computed. By contrasting the anticipated labels with the actual labels, training accuracy is calculated. Corresponding lists are used to store the training accuracy and loss. The model is switched to evaluation mode at the conclusion of each epoch. Accuracy and validation loss are calculated similarly to the training loop. The accuracy and loss of the validation are kept in related lists. The scheduler for learning rate is updated. The training loop continues for the specified number of epochs. During training, the code prints the epoch number, training accuracy, validation accuracy, training loss, and validation loss.

2.8.4 Fine tuning with MiniLM

This time, learning rate, batch size, and optimizer are three hyperparameters that are taken into account. The batch sizes are [16, 32], the learning rates [1e-5, 1e-4], and the optimizers ["Adam", "SGD"] are all configured. To test various combinations, we run a nested loop over these hyperparameters. For each parameter combination in the hyperparameter search cycle, a fresh MiniLM model instance is built. On the basis of the present parameter configuration, the optimizer is defined. Calculated and stored are training accuracy, training loss, validation accuracy, and validation loss. The values of the current parameters are changed in best_params if the validation accuracy is higher. After fine-tuning the MiniLM and recording the best parameter values, we trained the MiniLM once more using these values, and we recorded the accuracy on validation data.

Chapter 3

Evaluation and Results

3.1 Evaluation

We evaluated our proposed system with 4 evaluations that are training results, validation results, comparison between accuracies and losses of SBERT and MiniLM on validation These evaluations are conducted once the training of the model is complete. The training of the model is done in different settings and hyper-parameters. The best hyper-parameters are selected for the model to be trained.

3.2 Results

We trained both the SBERT and MiniLM models using the best hyper parameters on a dataset named ”df_d1”.

The following graphs shows the training and validation accuracy of fine-tuned SBERT.

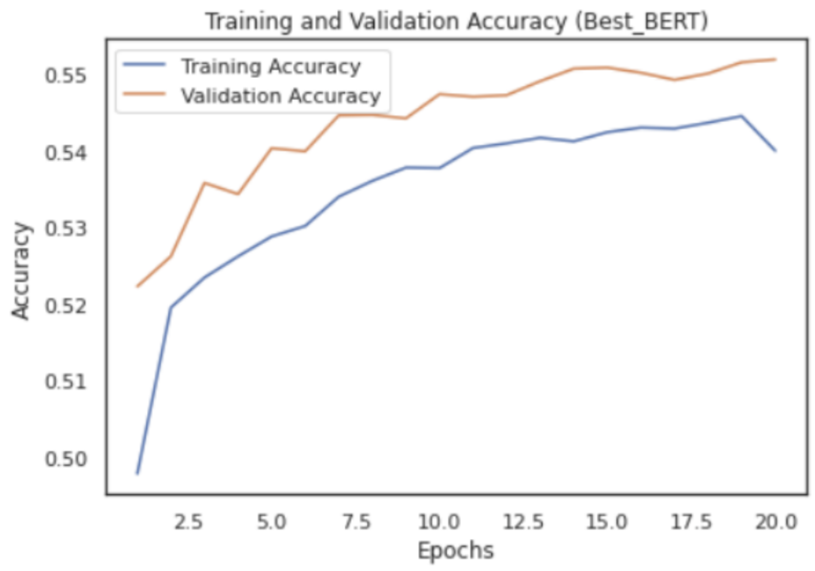


Fig.2: Training and Validation Accuracy of SBERT.

The graph shows that the maximum accuracy achieved by fine tuned SBERT on training data is 54% and the maximum accuracy gained after 10 epochs on validation data is 55.11% means that the model is performing not so well but acceptable. We also recorded the training and validation loss on fine tuned SBERT and visualized it in the following graph.

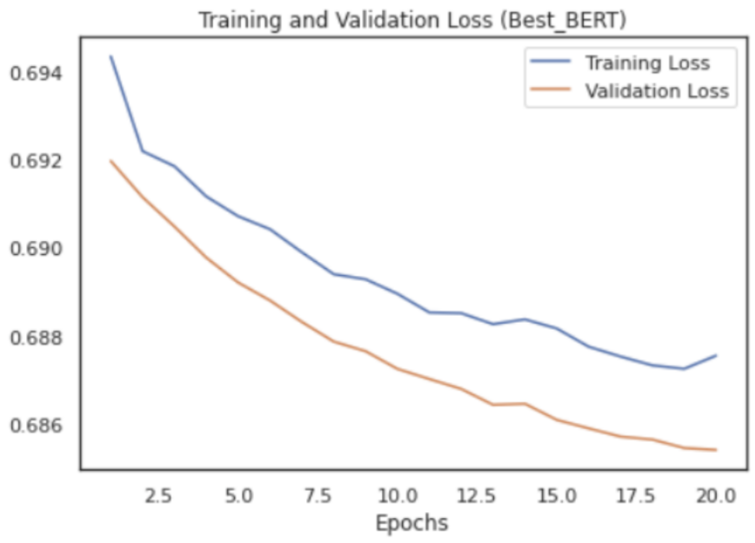


Fig.3: Training and Validation Loss of SBERT.

The graph shows that training and validation loss on fine tuned SBERT decreases from 0.69 to 0.68 respectively.

We can find the performance of fine tuned MiniLM with the help of accuracy and loss graphs. The following graph represents the information about the accuracy achieved by MiniLM on best hyperparameters.

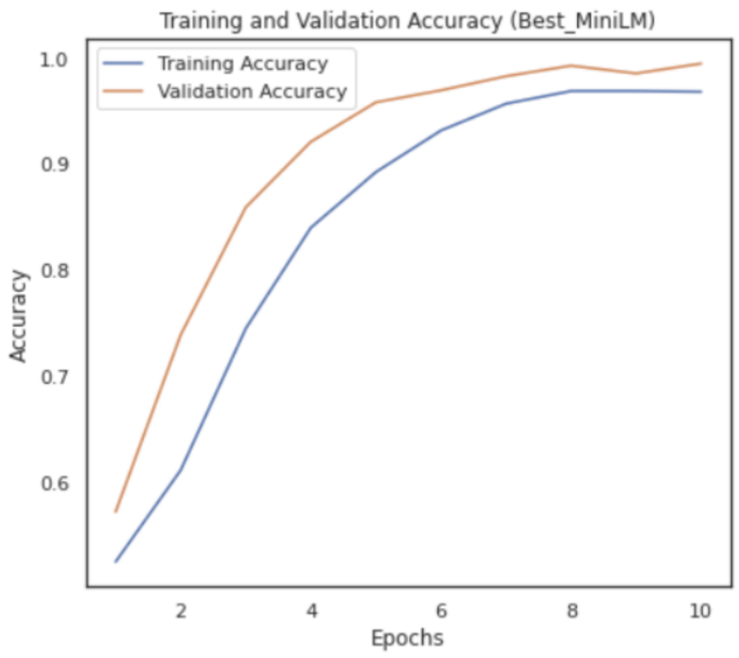


Fig.4: Training and Validation Accuracy of MiniLM.

We can observe that after fine tuning the MiniLM has achieved a high accuracy which is 99% on validation data. Similarly, the following loss graph gives us the performance information of fine tuned MiniLM.

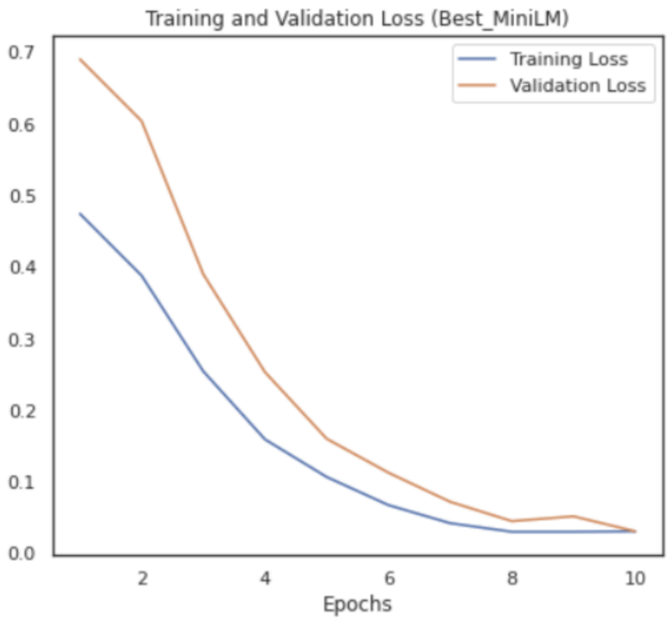


Fig.5: Training and Validation Loss of MiniLM.

As it is depicted in the graph, the training and validation loss values are very low which is approximately 0.031 means the model is giving very accurate predictions on unseen text data.

Conclusion

In this study, we explored the application of two pre-trained models, SBERT and MiniLM, on the FrameNet dataset. Additionally, we performed fine-tuning on both models and observed significant improvements in accuracy, particularly with the MiniLM model. The utilization of SBERT allowed us to capture contextualized embeddings for textual representations within the FrameNet dataset. This enabled us to leverage the semantic information encoded in the sentences and frames, facilitating improved performance in downstream tasks. By fine-tuning the SBERT model, we were able to further optimize its performance on FrameNet, resulting in enhanced accuracy which is 55% and more precise semantic representations.

However, the real breakthrough came with the application of the MiniLM model. MiniLM, with its powerful language modeling capabilities, showcased exceptional performance after fine-tuning on the FrameNet dataset. The fine-tuned MiniLM model not only achieved higher accuracy which is 99% but also demonstrated a deeper understanding of the semantic frames present in the data. This outcome suggests that MiniLM is well-suited for capturing the intricate semantic nuances present in FrameNet and effectively modeling the relationships between words and frames. The success of fine-tuning the MiniLM model on FrameNet underscores the significance of domain-specific fine-tuning in natural language processing tasks. The ability to adapt a pre-trained model to a specific dataset, such as FrameNet, allows for the exploitation of domain-specific features and patterns, resulting in improved performance and better representation learning.

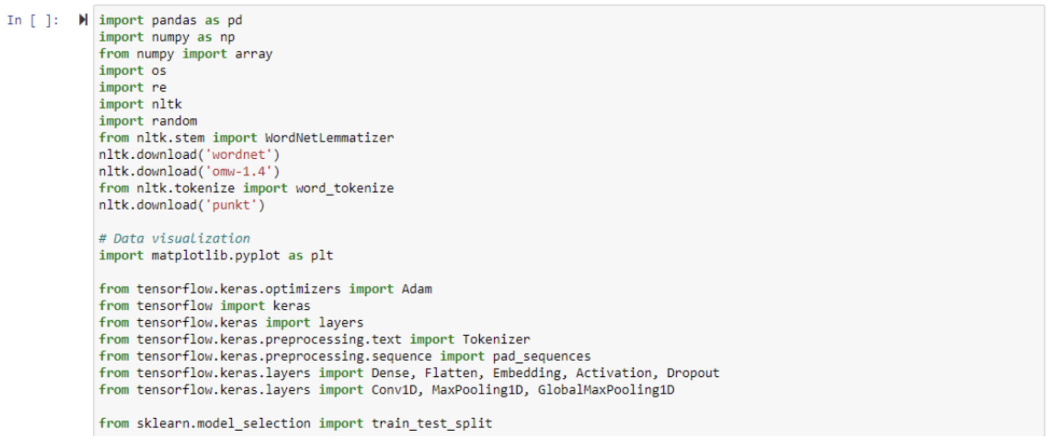
In conclusion, this study contributes to the growing body of research on leveraging pre-trained models and fine-tuning techniques for semantic frame analysis. The advancements made in accurately representing and understanding semantic frames in the FrameNet dataset have significant implications for a wide range of natural language processing applications, including sentiment analysis, information extraction, and argumentation mining. The findings of this study pave the way for future research endeavors, encouraging the exploration of more sophisticated techniques and models for advancing the field of semantic frame analysis and its practical applications.

Bibliography

1. Barboto, G., Miclet, L., & Richard, G. ,2014,. Integrating FrameNet and WordNet for Frame Identification. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016).*
2. Coucerio, E. ,2017. FrameNet: An Overview of Its Construction and Applications. In *M. Kipp & E. Schulte im Walde (Eds.)*, FrameNet in Action: The Case of Attitude Verbs (pp. 1-20). Language Science Press.
3. Fillmore, C.J., Ruppenhofer, J., & Baker, C.F. ,2004. Framenet and representing the link between semantic and syntactic relations. Computational Linguistics and Beyond, C. R. Huang and W. Lenders, eds., pp. 19-62, *Institute of Linguistics, Academia Sinica*, pp. 19–59.
4. Lim, S., Prade, H. & Richard, G., 2021. Classifying and completing word analogies by machine learning. *International Journal of Approximate Reasoning*, 132, pp.1-25. DOI :`https://doi.org/10.1016/j.ijar.2021.02.002`
5. Miclet, L., Cortes, C., & Vapnik, V. ,2005. Efficient Confidence Intervals for Support Vector Machines. In *Proceedings of the 22nd International Conference on Machine Learning (ICML 2005).*
6. Palmer, M., Gildea, D., & Kingsbury, P. ,2005. Exploiting FrameNet for Semantic Role Labeling. *Computational Linguistics*, 31(1), 1-28.
7. Prade, H. & Richard, G. ,2021, August. Analogical Proportions: Why They Are Useful in AI. In *IJCAI* (pp. 4568-4576), DOI : `https://doi.org/10.24963/ijcai.2021/621`
8. Reimers, N., & Gurevych, I. ,2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 3982-3992. DOI : `https://doi.org/10.48550/arXiv.1908.10084`
9. Stab, C., & Gurevych, I. ,2014. FrameNet as a Resource for Argumentation Mining. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (ACL).*. DOI : `10.18653/v1/W18-5211`
10. Wang, W., Xu, R., Qiu, X., & Liu, X. ,2020. MiniLM: Deep Self-Attention Distillation for Task-Agnostic Compression of Pretrained Transformers. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics (ACL)*, 2738-2748. DOI : `https://doi.org/10.48550/arXiv.2002.10957`
11. Wiebe, J., Wilson, T., & Cardie, C. ,2004. FrameNet+: A Manually Annotated Lexical Resource for Sentiment Analysis. In *Proceedings of the 4th International Conference on Language Resources and Evaluation (LREC).*

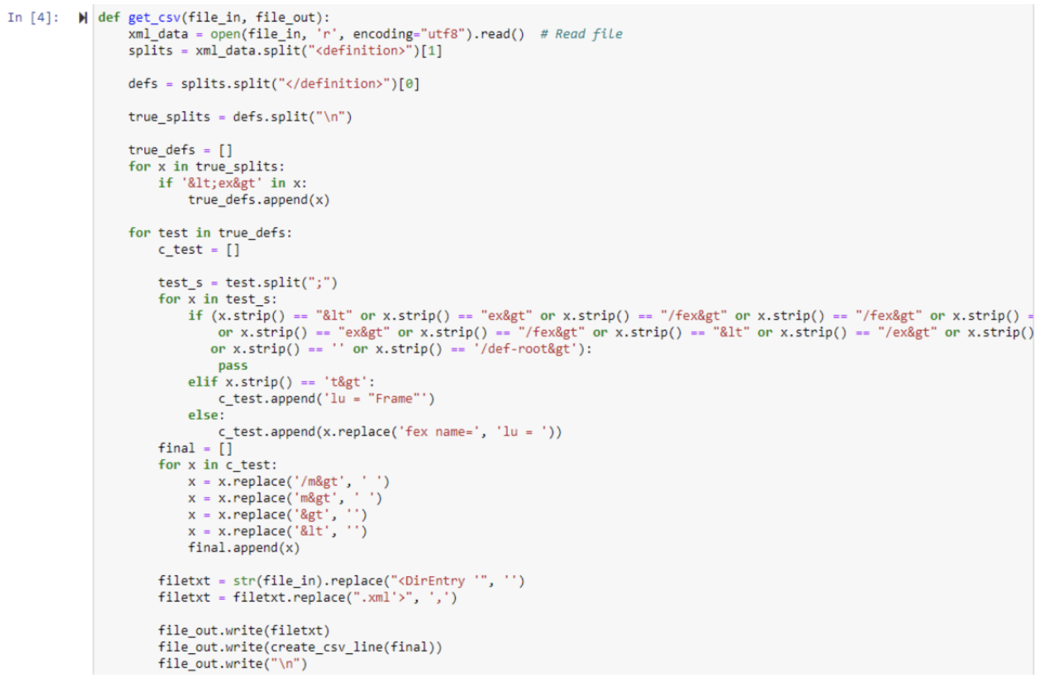
Appendix

Importing Libraries



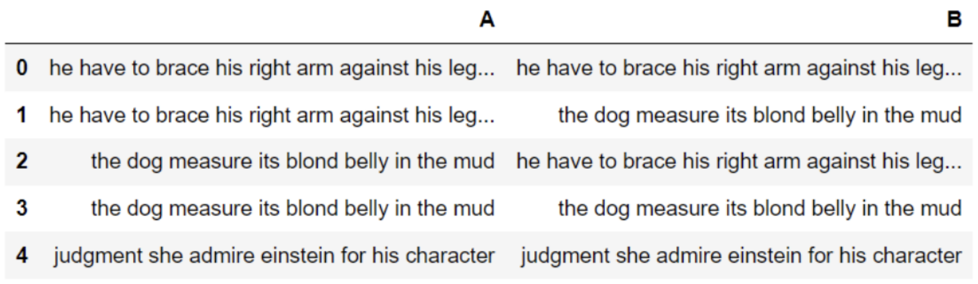
```
In [ ]:  import pandas as pd
         import numpy as np
         from numpy import array
         import os
         import re
         import nltk
         import random
         from nltk.stem import WordNetLemmatizer
         nltk.download('wordnet')
         nltk.download('omw-1.4')
         from nltk.tokenize import word_tokenize
         nltk.download('punkt')

         # Data visualization
         import matplotlib.pyplot as plt

         from tensorflow.keras.optimizers import Adam
         from tensorflow import keras
         from tensorflow.keras import layers
         from tensorflow.keras.preprocessing.text import Tokenizer
         from tensorflow.keras.preprocessing.sequence import pad_sequences
         from tensorflow.keras.layers import Dense, Flatten, Embedding, Activation, Dropout
         from tensorflow.keras.layers import Conv1D, MaxPooling1D, GlobalMaxPooling1D

         from sklearn.model_selection import train_test_split
```

Data Fetching from Frames



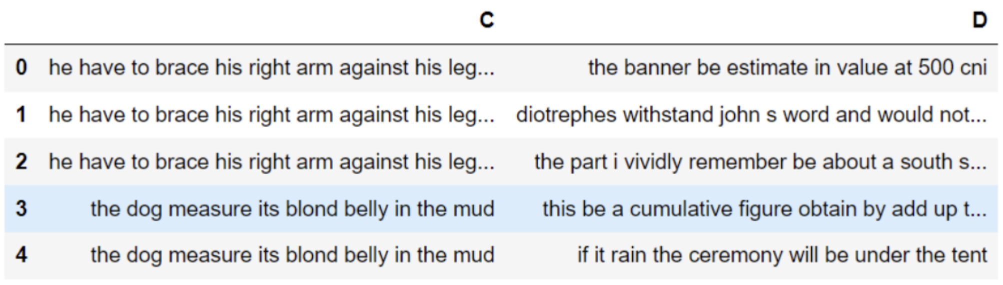
```
In [4]:  def get_csv(file_in, file_out):
             xml_data = open(file_in, 'r', encoding="utf8").read()  # Read file
             splits = xml_data.split("<definition>")[1]

             defs = splits.split("</definition>")[0]

             true_splits = defs.split("\n")

             true_defs = []
             for x in true_splits:
                 if '&lt;ex&gt' in x:
                     true_defs.append(x)

             for test in true_defs:
                 c_test = []

                 test_s = test.split(";")
                 for x in test_s:
                     if (x.strip() == "&lt" or x.strip() == "ex&gt" or x.strip() == "/fex&gt" or x.strip() == "/fex&gt" or x.strip() =
                         or x.strip() == "ex&gt" or x.strip() == "/fex&gt" or x.strip() == "&lt" or x.strip() == "/ex&gt" or x.strip()
                        or x.strip() == '' or x.strip() == '/def-root&gt'):
                         pass
                     elif x.strip() == 't&gt':
                         c_test.append('lu = "Frame"')
                     else:
                         c_test.append(x.replace('fex name=', 'lu = '))
                 final = []
                 for x in c_test:
                     x = x.replace('/m&gt', ' ')
                     x = x.replace('m&gt', ' ')
                     x = x.replace('&gt', '')
                     x = x.replace('&lt', '')
                     final.append(x)

                 filetxt = str(file_in).replace("<DirEntry '", '')
                 filetxt = filetxt.replace(".xml'>", ',')

                 file_out.write(filetxt)
                 file_out.write(create_csv_line(final))
                 file_out.write("\n")
```

Data Preparation part 1



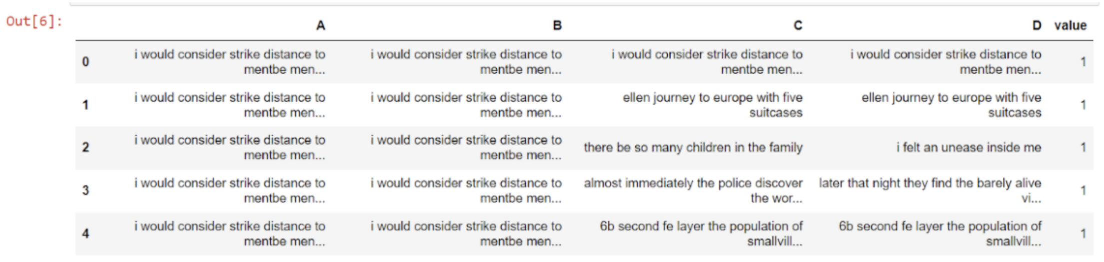
	A	B
0	he have to brace his right arm against his leg...	he have to brace his right arm against his leg...
1	he have to brace his right arm against his leg...	the dog measure its blond belly in the mud
2	the dog measure its blond belly in the mud	he have to brace his right arm against his leg...
3	the dog measure its blond belly in the mud	the dog measure its blond belly in the mud
4	judgment she admire einstein for his character	judgment she admire einstein for his character

Data Preparation part 2



	C	D
0	he have to brace his right arm against his leg...	the banner be estimate in value at 500 cni
1	he have to brace his right arm against his leg...	diotrephes withstand john s word and would not...
2	he have to brace his right arm against his leg...	the part i vividly remember be about a south s...
3	the dog measure its blond belly in the mud	this be a cumulative figure obtain by add up t...
4	the dog measure its blond belly in the mud	if it rain the ceremony will be under the tent

Final Dataset

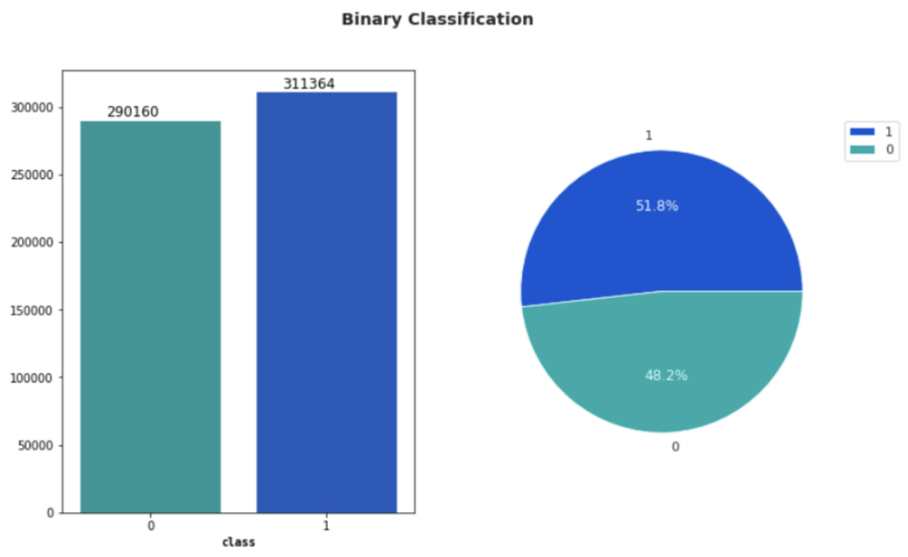


Out[6]:

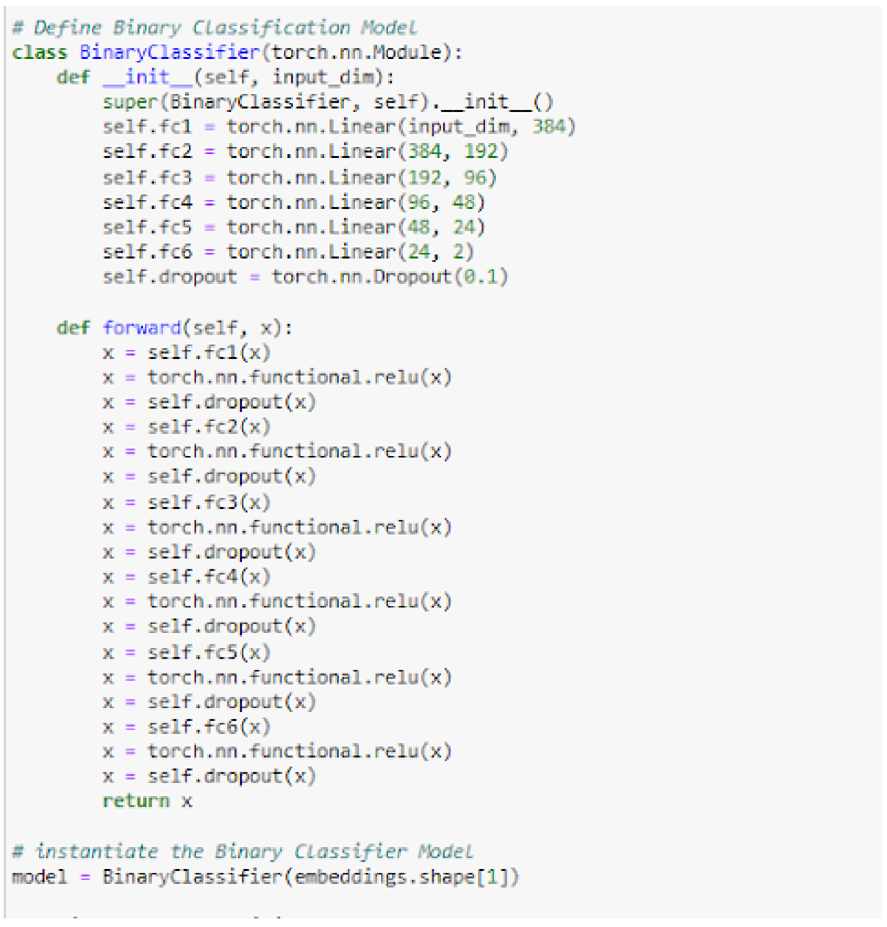
	A	B	C	D	value
0	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	1
1	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	ellen journey to europe with five suitcases	ellen journey to europe with five suitcases	1
2	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	there be so many children in the family	i felt an unease inside me	1
3	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	almost immediately the police discover the wor...	later that night they find the barely alive vi...	1
4	i would consider strike distance to mentbe men...	i would consider strike distance to mentbe men...	6b second fe layer the population of smallvill...	6b second fe layer the population of smallvill...	1

Dataset Statics

```
Out[7]: Text(0.5, 0.98, 'Binary Classification')
```



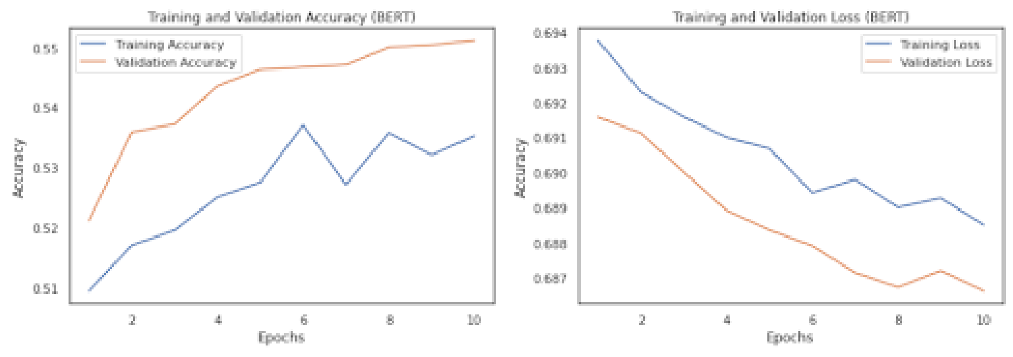
SBERT Model Building



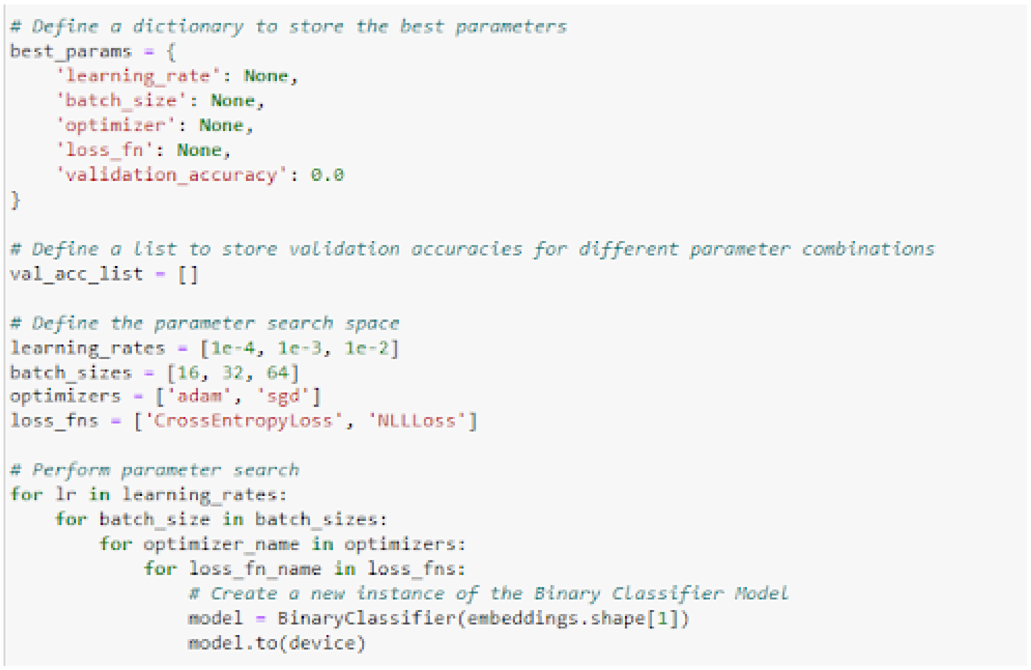
```
# Define Binary Classification Model
class BinaryClassifier(torch.nn.Module):
    def __init__(self, input_dim):
        super(BinaryClassifier, self).__init__()
        self.fc1 = torch.nn.Linear(input_dim, 384)
        self.fc2 = torch.nn.Linear(384, 192)
        self.fc3 = torch.nn.Linear(192, 96)
        self.fc4 = torch.nn.Linear(96, 48)
        self.fc5 = torch.nn.Linear(48, 24)
        self.fc6 = torch.nn.Linear(24, 2)
        self.dropout = torch.nn.Dropout(0.1)

    def forward(self, x):
        x = self.fc1(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc2(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc3(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc4(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc5(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        x = self.fc6(x)
        x = torch.nn.functional.relu(x)
        x = self.dropout(x)
        return x

# instantiate the Binary Classifier Model
model = BinaryClassifier(embeddings.shape[1])
```

Simple SBERT Performance Graphs

```
<matplotlib.legend.Legend at 0x7fc127811510>
```

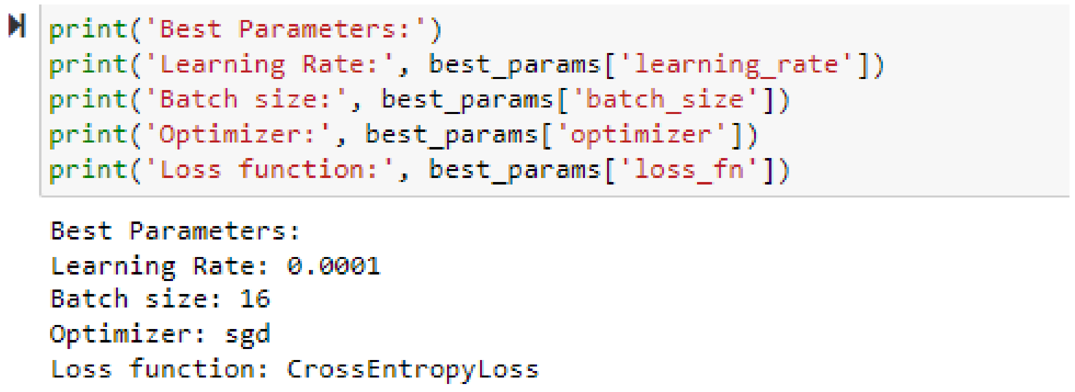


SBERT Fine Tuning



```
# Define a dictionary to store the best parameters
best_params = {
    'learning_rate': None,
    'batch_size': None,
    'optimizer': None,
    'loss_fn': None,
    'validation_accuracy': 0.0
}

# Define a list to store validation accuracies for different parameter combinations
val_acc_list = []

# Define the parameter search space
learning_rates = [1e-4, 1e-3, 1e-2]
batch_sizes = [16, 32, 64]
optimizers = ['adam', 'sgd']
loss_fns = ['CrossEntropyLoss', 'NLLLoss']

# Perform parameter search
for lr in learning_rates:
    for batch_size in batch_sizes:
        for optimizer_name in optimizers:
            for loss_fn_name in loss_fns:
                # Create a new instance of the Binary Classifier Model
                model = BinaryClassifier(embeddings.shape[1])
                model.to(device)
```

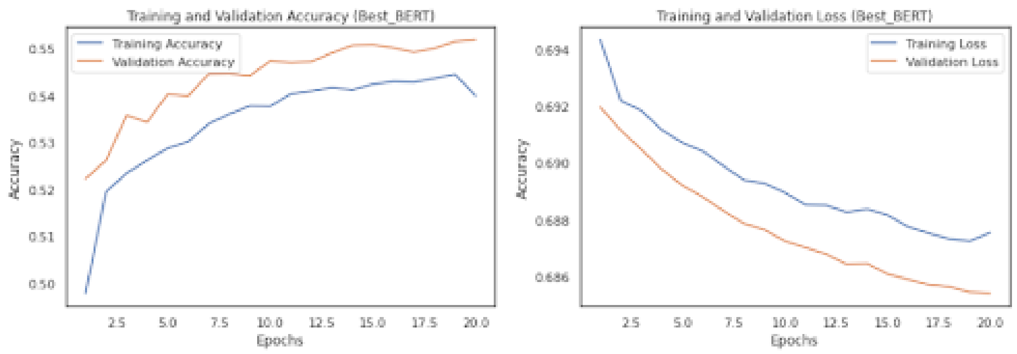
Best Parameter of SBERT



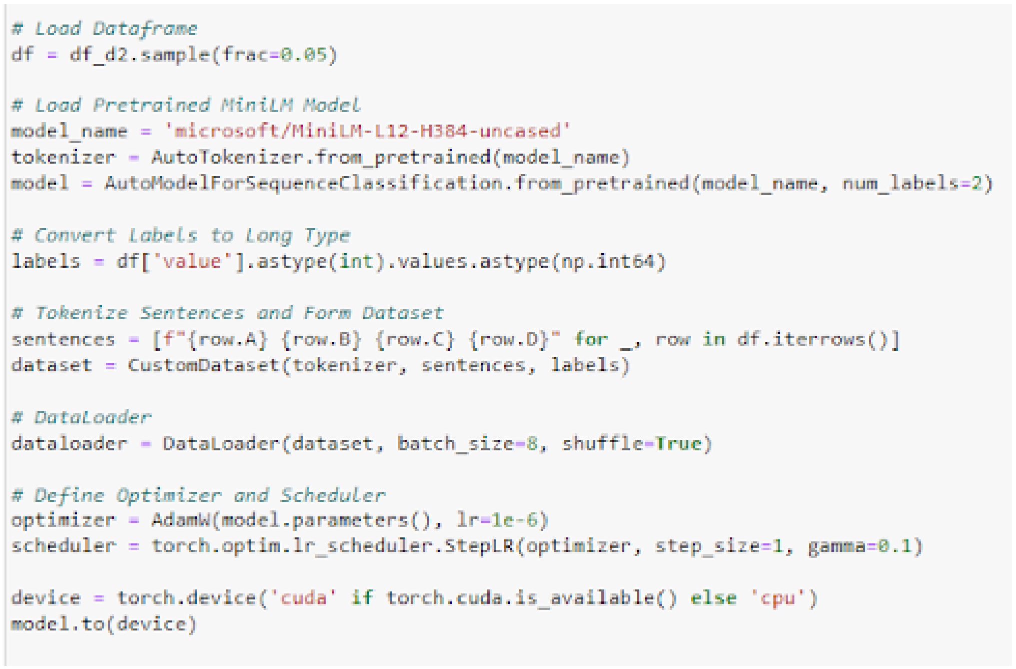
```python
print('Best Parameters:')
print('Learning Rate:', best_params['learning_rate'])
print('Batch size:', best_params['batch_size'])
print('Optimizer:', best_params['optimizer'])
print('Loss function:', best_params['loss_fn'])
```

```
Best Parameters:
Learning Rate: 0.0001
Batch size: 16
Optimizer: sgd
Loss function: CrossEntropyLoss
```

Fine Tuned SBERT Performance Graphs

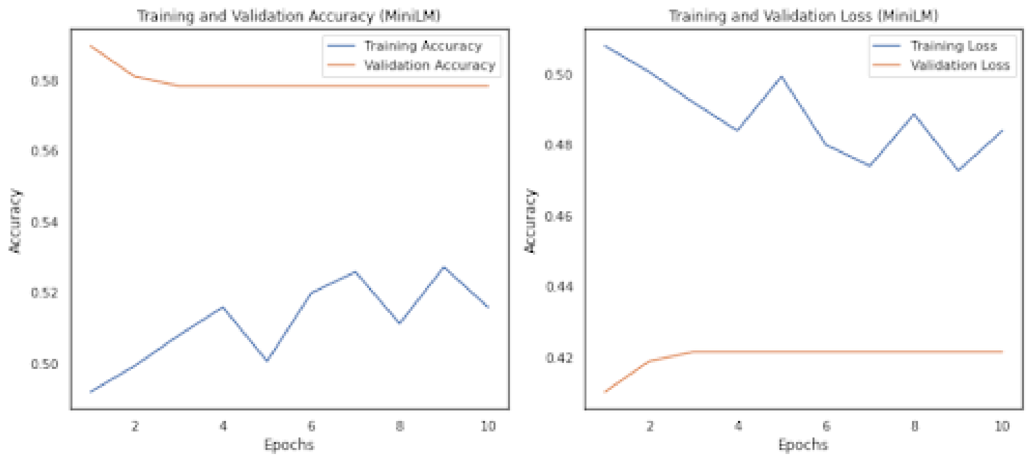


MiniLM Model building

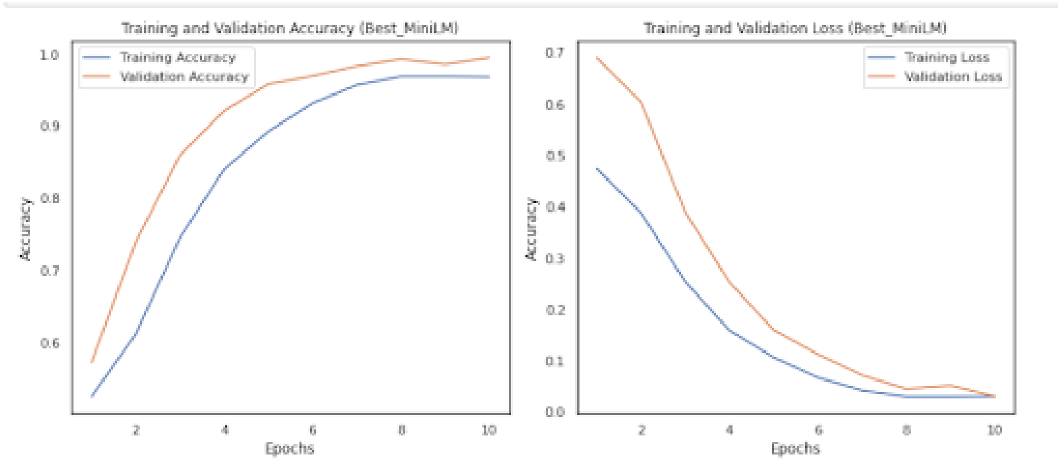


```python
# Load Dataframe
df = df_d2.sample(frac=0.05)

# Load Pretrained MiniLM Model
model_name = 'microsoft/MiniLM-L12-H384-uncased'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)

# Convert Labels to Long Type
labels = df['value'].astype(int).values.astype(np.int64)

# Tokenize Sentences and Form Dataset
sentences = [f"{row.A} {row.B} {row.C} {row.D}" for _, row in df.iterrows()]
dataset = CustomDataset(tokenizer, sentences, labels)

# DataLoader
dataloader = DataLoader(dataset, batch_size=8, shuffle=True)

# Define Optimizer and Scheduler
optimizer = AdamW(model.parameters(), lr=1e-6)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=1, gamma=0.1)

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

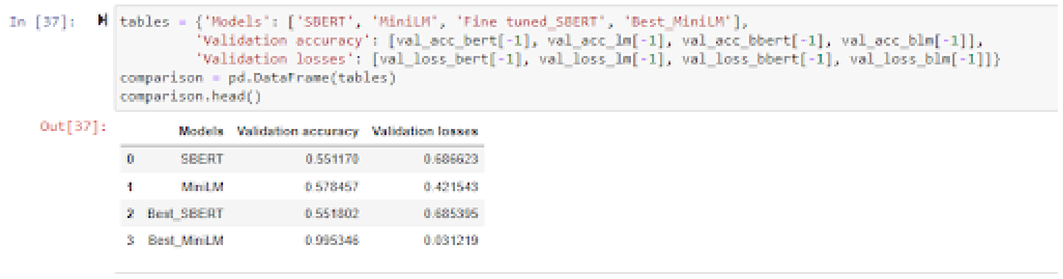
Performance Graphs of MiniLM



Performance Graphs of Fine Tuned MiniLM



Model Comparison



```
In [37]:  tables = {'Models': ['SBERT', 'MiniLM', 'Fine tuned_SBERT', 'Best_MiniLM'],
                    'Validation accuracy': [val_acc_bert[-1], val_acc_lm[-1], val_acc_bbert[-1], val_acc_blm[-1]],
                    'Validation losses': [val_loss_bert[-1], val_loss_lm[-1], val_loss_bbert[-1], val_loss_blm[-1]]}
          comparison = pd.DataFrame(tables)
          comparison.head()
```

Out[37]:

	Models	Validation accuracy	Validation losses
0	SBERT	0.551170	0.686623
1	MiniLM	0.578457	0.421543
2	Best_SBERT	0.551802	0.685395
3	Best_MiniLM	0.995346	0.031219