ANALYSIS OF EXISTING NATURAL LANGUAGE ON SEMANTIC SIMILARITY

DIPLOMA THESIS

Yassine Ismaili

Supervisor: doc. Ing. Arnošt Veselý, CSc.

Faculty of Economics and Management

Czech University of Life Sciences Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Ing. Yassine Ismaili

Informatics

Thesis title

Analysis of existing natural language on semantic similarity

Objectives of thesis

The main objective of this research report is to define the meaning of semantic similarity among words and sentences and provide literature reviews, state of the art on semantic similarity and their applications, to provide more details about advantages and drawbacks.

Methodology

This study research concentrates on a measure of semantic similarity of words and sentences of natural language (Wu et al. 2022; D. Chandrasekaran and Mago 2021). Two approaches exist to measure the similarity semantics: The vector approach and the statistical approach. In our case study, BERT (Bidirectional Encoder Representations from Transformers), one of the embedding words techniques is applied (Kumar and Raman 2022), this approach transforms a word into a vector representation of text data using the contexts in which it has learned (Khan et al. 2022). In general, he learned from internet platforms (e.g., Wikipedia, Twitter).

An overview of modern embedded technique will be given in this thesis report and we will also demonstrate its applications in different fields.

In the practice part of this report, we suggest how the embedding technique designed for English can be used also for other languages, for example, the Czech language, and some of the suggested methods will be evaluated on data.

The proposed extent of the thesis

60

Keywords

Deep learning; Embedding words; Semantic similarity

Recommended information sources

Balakrishnan, V., Z. Shi, C.L. Law, R. Lim, L.L. Teh, and Y. Fan. 2022. 'A Deep Learning Approach in Predicting Products' Sentiment Ratings: A Comparative Analysis'. Journal of Supercomputing 78 (5): 7206–26. https://doi.org/10.1007/s11227-021-04169-6.

Duarte, J.M., S. Sousa, E. Milios, and L. Berton. 2021. 'Deep Analysis of Word Sense Disambiguation via Semi-Supervised Learning and Neural Word Representations'. Information Sciences 570: 278–97. https://doi.org/10.1016/j.ins.2021.04.006.

Gao, M., J. Lu, and F. Chen. 2022. 'Medical Knowledge Graph Completion Based on Word Embeddings'. Information (Switzerland) 13 (4). https://doi.org/10.3390/info13040205.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. Deep Learning. Adaptive Computation and Machine Learning Series. Cambridge, MA, USA: MIT Press.

Chandrasekaran, D., and V. Mago. 2021. 'Comparative Analysis of Word Embeddings in Assessing Semantic Similarity of Complex Sentences'. IEEE Access 9: 166395–408. https://doi.org/10.1109/ACCESS.2021.3135807.

Chandrasekaran, Dhivya, and Vijay Mago. 2021. 'Evolution of Semantic Similarity&#x2014;A Survey'. ACM Computing Surveys 54 (2): 41:1–41:37. https://doi.org/10.1145/3440755.

Jaber, A., and P. Martínez. 2022. 'Disambiguating Clinical Abbreviations Using a One-Fits-All Classifier Based on Deep Learning Techniques'. Methods of Information in Medicine. https://doi.org/10.1055/s-0042-1742388.

Jackson, Howard. 1990. Grammar and Meaning: A Semantic Approach to English Grammar. Longman. Jiang, S., W. Wu, N. Tomita, C. Ganoe, and S. Hassanpour. 2020. 'Multi-Ontology Refined Embeddings (MORE): A Hybrid Multi-Ontology and Corpus-Based Semantic Representation Model for Biomedical Concepts'. Journal of Biomedical Informatics 111. https://doi.org/10.1016/j.jbi.2020.103581.

Khan, L., A. Amjad, N. Ashraf, and H.-T. Chang. 2022. 'Multi-Class Sentiment Analysis of Urdu Text Using Multilingual BERT'. Scientific Reports 12 (1). https://doi.org/10.1038/s41598-022-09381-9.

Kumar, P., and B. Raman. 2022. 'A BERT Based Dual-Channel Explainable Text Emotion Recognition System'. Neural Networks 150: 392–407. https://doi.org/10.1016/j.neunet.2022.03.017.

Expected date of thesis defence

2022/23 WS – FEM

The Diploma Thesis Supervisor

doc. Ing. Arnošt Veselý, CSc.

Supervising department

Department of Information Engineering

Electronic approval: 4. 11. 2022

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 28. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Dean

Prague on 30. 03. 2023

Declaration

I declare that I have worked on my master’s thesis titled "Name of the master’s thesis" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the master’s thesis, I declare that the thesis does not break any copyrights.

In Prague on date of submission 30 March 2023

Acknowledgements

I would also like to express my deep and respectful gratitude to my supervisor ***doc. Ing. Arnošt Veselý, CSc.***, for having trusted this work and guided in its realization, for the effort and support he has given to my skill improvements, and for the opportunity that I had to work with him. I would also like to thank my supervisor for providing helpful suggestions and constructive comments.

Also my sincere thanks to ***all professors*** in the Department of information Engineering, I want to express my highest consideration for teaching during my Master's course.

ABSTRACT

The fourth industrial era is reliant on artificial intelligence (AI), and various semantic similarities approaches are constantly evolving. Theses popular techniques are widely applied in medical and health science, industrial and economic management. Similarity evaluation between textual documents is problematic and challenging. For this reason, the semantic text is used to provide a precise and structured representation of text meaning. It is largely used for solving text problems in various domains, as analyzing textual data, sentiment analysis, retrieving information, or extracting knowledge from textual data (text mining).

For this purpose, the contribution of this research is to understand the meaning of semantic similarity among words and sentences and provide state of art, with their applications, to require more details about advantages and drawbacks. Also, in the second part, we aim to test the BERT methodology in the English and Czech languages, which is one of a powerful model for dealing with information retrieval and text summarization. This report describes the use of efficient and user-friendly semantic textual similarity tools to propose optimal techniques and solutions in terms of reducing time, resources, and computing costs.

Keywords Embedding words, Semantic similarity, BERT, Deep learning

Contents

List of Figures

1 Introduction

In the extensive field of artificial intelligence study, deep learning is one of the most popular methods for the extraction of entities based on the organizational diagram of texts written in natural language [Goodfellow et al., 2016]. The deep learning application is rapidly increasing in the last decade, which is starting to convert into a digital revolution era [Duarte et al., 2021, Xiang et al., 2021]. Mainly, the semantic similarity approaches have seen interest in numerous domains [Wu et al., 2022, Chandrasekaran and Mago, 2021], including medical studies [Jiang et al., 2020, Jaber and Martínez, 2022, Gao et al., 2022], computer and engineering sciences [Neves Oliveira et al., 2022, Balakrishnan et al., 2022].

The semantic technique is based on the resemblance of the concept signification and semantic content [Jackson, 1990, Chandrasekaran and Mago, 2021]. The similarity degree measure between the pair texts is an obstacle that has shown the interest of various researchers of natural language processing and information retrieval domains over decades [Martín et al., 2022]. The complicated task has led to numerous techniques to provide similarity measures that are able to account for the largest possible number of features [Mihalcea, 2006].

However, the robustness of such approaches remains limited [H.Gomaa and A. Fahmy, 2013]. Finding semantic similarity methods is quite problematic and requires multiple testing in measurement techniques, for extracting entities from texts written in natural language. For this reason, this research check on how effective are those approaches in the evaluation of optimal solutions.

The Master report's objective is to define the meaning of semantic similarity among words and sentences and provide literature reviews, state of art on semantic similarity and their applications, to provide more details about advantages and drawbacks. The added value of the application use is also discussed in the end of this project report.

The thesis report consisted of three steps: (i) in the first step, we introduce the definition of semantic similarity in the literature reviews and their approaches, to provide more details about advantages and drawbacks.

(ii) we define in the second step the methodology of this thesis report, including the BERT approach (Bidirectional Encoder Representations from Transformers), and some modern embedded techniques. (iii) we designed in the last part, the practical tools of BERT and sentiment analysis using the Python program, by describing some examples of general algorithms and program code, which apply these computations.

Part I

Semantic similarity methodology

2 Literature reviews

2.1 Semantic similarity

Semantic similarity is an influential element of Natural language processing (NLP) and one of the fundamental challenges for many natural language processing uses and associated disciplines.

Text similarity measures have been used in many fields. For example, several semantic web applications, such as community extraction, ontology construction, entity identification for text categorization, word sense disambiguation, and machine translation, detect plagiarism and duplicate documents.

The measure of semantic similarity between words and sentences in natural language is the main topic of this research study. A collection of texts or terms is given a metric developed on how similar their meanings or semantic content are, which is known as a semantic similarity measure [Guo et al., 2013].

Semantic similarity is a problem in automatic language processing, aiming to evaluate the semantic proximity of statements. The notion of similarity can thus be applied to detecting paraphrases or text summarization. This problem has been the subject of many works and evaluation campaigns, such as SemEval [Cer et al., 2017].

Notice that semantic distance is subdivided into two types: semantic similarity and semantic relatedness. The former is a subset of the latter, but the two terms can be applied interchangeably in some contexts, which makes it even more important to be aware of their distinction. Two approaches are included in semantic similarity if there is a synonymy, hyponymy, antonymy, or troponymy between them (Examples: DOCTOR - SURGEON, DARK - LIGHT). (ADD REF)

Two meanings of words are studied semantically related if there is at least one lexical-semantic relationship between them -classical or non-classical (Examples: CHIRURGIAN-SCALPEL, TREE- SHADOW).

Work on automatic similarity computation at the phrasal level relies on resources such as the STS Benchmark [Cer et al., 2017] or SICK [Marelli et al., 2014] corpora that contain annotated sentence pairs with a degree of similarity (STS Benchmark) or a similarity relation (SICK).

With few exceptions, the classic method for determining text similitude between two segments is to utilize a user-friendly lexical matching approach and calculate a resemblance score derived from the number of lexical units that are present in the two segments. Progresses have been made to this simple method which consists in removing empty words, considering only the longest sub-sequence, or weighting or normalization.

Many successful semantic similarity measures are based on methods that are either knowledge-based [Wu and Palmer, 1994, Leacock et al., 1998] or corpus-based [Deerwester et al., 1990]. These models have been well used in language processing assignments as word sense disambiguation [Patwardhan et al., 2003].

2.2 Vector approaches

2.2.1 Semantic vectors

In this part, we distinguish between vector, topological, and statistical approaches.

Semantic vectors are a model of the linguistic notion of the semantic field. Semantic vectors are described by the positive part of a vector space. The aim is to determine the semantics of a word by consulting the other terms used alongside it in sentences. An easy way to do this is to apply vectors to represent the meaning of words, and then use vector similarity measures (as for syntactic similarity). The most difficult thing is to obtain such vectors. It is therefore necessary to build a set of vectors for each word in the dictionary used.

The vectors are denoised in an n-dimensional orthogonal vector space where each base is assigned a unique vocabulary word (so each dictionary entry has a base in the vector space). For each word in the dictionary, we determine a vector in this space, where the component of the vector for each basis is the word occurrence number in the basis that represents it where it shows in the context of the word for which a vector has been constructed.

The similarity between two points/vectors is computed in general as:

- Manhattan distance (Levenshtein distance, L1 norm)

$$dist_{L1}\left(\vec{x},\vec{y}\right)=\sum_{i=1}^{n}\left|x_i-y_i\right| \tag{1}$$

- Euclidean distance (L2 norm)

$$dist_{L2}\left(\vec{x},\vec{y}\right)=\sqrt{\sum_{i=1}^{n}\left(x_i-y_i\right)^2} \tag{2}$$

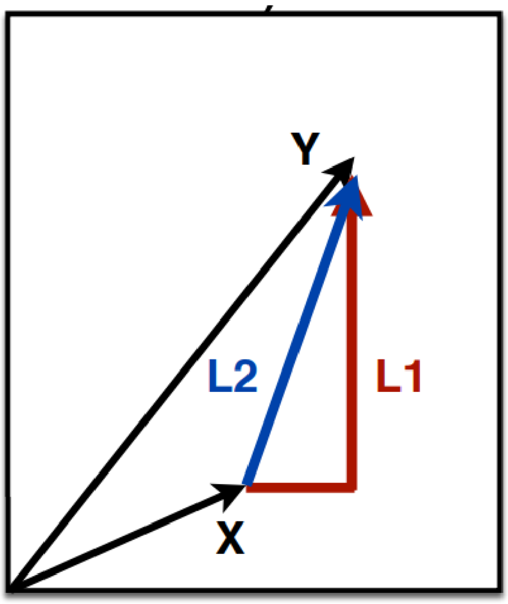


Figure 1: Semantic vector

2.2.2 Bi-clustering

Biclustering, also known as coclustering, is widely used in the field of bioinformatics and has been extended to recommendation systems recommendation systems and text analysis. In sum, these methods apply to any matrix whose cells represent a relationship between rows and columns. If biclustering techniques simultaneously consider the duality that exists between documents and their terms,the techniques derived from LDA rely entirely on a generative model for the dimension of the documents.

Bi-clustering is the process of clustering rows and columns at once of a data matrix, where each row presents a sample and each column represents a gene. The goal of bi-clustering is to determine subsets of genes that are co-expressed across a subset of samples, or subsets of samples that exhibit similar gene expression patterns across a subset of genes.

Bisson and Hussain [2008] presents a method for bi-clustering gene expression data using a regularized Haar-based approach.

The approach presented in the paper[Bisson and Hussain, 2008] is based on the Haar wavelet transform, which is a mathematical tool for decomposing signals into different frequency components. The authors propose a regularized Haar-based approach to bi-clustering, which involves minimizing an objective function that penalizes the complexity of the bi-cluster solution.

The proposed approach is assessed using synthetic and real gene expression datasets, and compared to several other bi-clustering methods. The findings demonstrate that the suggested technique is able to identify bi-clusters that are biologically relevant, and outperforms several other techniques in terms of accuracy and stability. The equation is as follows:

$$sim(di:, dj:) = Fs(di1, dj1) + ... + Fs(dic, djc) \tag{3}$$

where Fs The function used to measure similarity is referred to as a similarity function. At the outset, it is presumed that the SC (similarity coefficient) matrix is set to 1. i.e., sci,i = 1. Thus, the equation is defined as:

$$sim(di, dj) = Fs(di1, dj1).sc11 + ... + Fs(dic, djc).sccc. \tag{4}$$

The equation is later extended to encompass every conceivable combination of words. Conversely, the similarity between two words is identified by the shared documents in which they are present.

Thus, the equations to calculate the similarities are dependent: to obtain SR and SC, each equation must be used iteratively and alternatively and the values updated to be used in the following iterations.

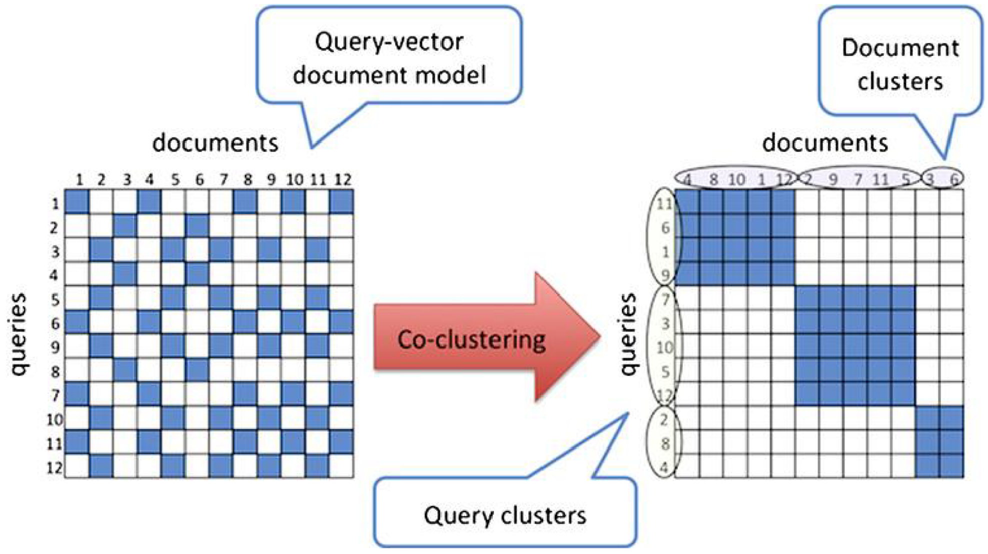


Figure 2: Co-clustering

2.2.3 Topological approaches

Approaches for determining similarity between words based on knowledge depend on a semantic network of words, like WordNet[Fellbaum, 1998]. To estimate the similarity between two words, their positions in the hierarchy of the knowledge base can be utilized. As a matter of fact, the arrangement of the knowledge base is a tree where each node is a concept, concepts can be nouns, verbs, or adjectives. Words have "synsets", which are sets of concepts to which the word can correspond.

Finally, it could be noted that the concepts are more and more abstract and general when we go to the root and that they are more specific when we go to the leaves. Wordnet is a knowledge base or taxonomy whose concepts are in English.

2.2.4 Edge-based

The method of arc-based approach is an intuitive and uncomplicated manner of determining the semantic similarity in a taxonomy. It involves approximating the distance between the relevant nodes, representing the concepts/classes that require comparison, such as the length of an arc.

Considering the multiple dimensions of concepts, the conceptual distance can be conveniently calculated by measuring the geometric distance between the nodes that correspond to the concepts. Clearly, the more similar two nodes are, the shorter the path between them.

2.2.5 Leacock and Chodorow

In their work, Leacock and Chodorow [Leacock et al., 1998] employed only one relation, which was the hyponymy. They modified the formula for path length to account for the fact that the shortest arcs in the hierarchy of hyponymy correspond to the smallest semantic distance.

For instance, the synsets associated with 'sports car' and 'car', which are located lower in the hierarchy, are more similar to each other than 'transportation' and 'instrumentation', which are higher up in the hierarchy, even though both pairs of nodes are separated by only one arc. The nodes are separated by precisely one arc in the hierarchy. The formula for measuring the similarity between the two concepts, $c1$ and $c2$, is expressed below:

$$simlch(c1, c2) = -log(length/(2 \times D)), \tag{5}$$

In this context, $length$ refers to the minimum node numbers between $c1$ and $c2$, and D denotes the highest depth/height of the taxonomy.

2.2.6 Wu and Palmer

The similarity measure introduced by Wu and Palmer [1994] calculates the level of two concepts in WordNet's hierarchy, as well as the level of their lowest shared ancestor (also known as lowest common subsumer). These values are combined to produce a similarity score. as follows:

$$SimWP = 2 \times N \div (N1 + N2) \tag{6}$$

The variables $N1$ and $N2$ represent the count of connections between the concepts and the root of the ontology, while the variable N represents the count of connections between the lowest common subsumer and the root of the ontology R.

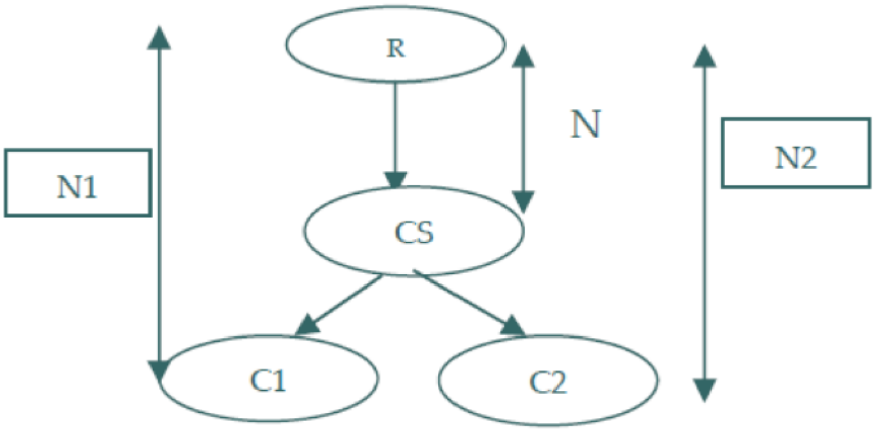


Figure 3: Wu and Palmer similarity metric

2.2.7 Node-based

An approach for determining the similarity between concepts based on their nodes is referred to as an information content-based approach, as described by Resnik [1995] and Li [1998]. In this approach, concepts are represented by nodes in a multidimensional space, and connections between concepts are represented by arcs. The similarity between two concepts is based on how much information they have in common. In this hierarchical structure, the shared information can be identified as a superclass that includes both concepts at the top of the hierarchy.

Resnik [1995] first introduced the concept of information content (IC), demonstrating that the number of specified classes defines an object, such as a word, and that the degree of semantic similarity between two concepts can be measured by the amount of shared knowledge. To determine the relevance of an object, it is necessary to calculate its information content, which is done by determining its frequency in the Wordnet corpus. The information content class value can be estimated by calculating the occurrence probability in a large text corpus. The information content IC of a concept is written as follows:

$$IC(c) = -\lg P(c) \tag{7}$$

The encountering an instance of concept c probability is denoted as $P(c)$.

These methods have certain limitations, such as their reliance on the specific corpus used, which may contain ambiguous or incomplete concepts. Additionally, they yield identical results for any pair of concepts that share the same lowest common subsumer (LCS).

2.2.8 Resnik

The method suggested by Resnik [1995] involves obtaining the information content (IC) of the lowest common ancestor (LCS) of two specified concepts:

$$simres(c1, c2) = IC(LCS(c1, c2)) = -\lg P(LCS(c1, c2)) \tag{8}$$

2.2.9 Lin

The measure proposed by Li [1998] is a normalization of that of Resnik [1995]. The normalization process include the information content (IC) of both concepts.

$$simlin(c1, c2) = 2sIC(LCSc1, c2)/IC(c1) + IC(c2) \tag{9}$$

2.2.10 Jiang and Conrath Enn

Another calculus suggested by Jiang and Conrath [1997], which is also founded on [Resnik, 1995], determines the similarity as follows:

$$simjnc(c1, c2) = 1 \div IC(c1) + IC(c2)2IC(LCS(c1, c2)) \tag{10}$$

2.3 Statistical approaches

Corpus-based similarity measures are distinct from the methods discussed earlier in that they do not necessitate a comprehension of the language's vocabulary or grammar. Latent semantic analysis (LSA), which was developed by [Deerwester et al., 1990],is a notable example of such measures. LSA involves a singular value decomposition (SVD) application to condense co-occurrences of terms in a corpus into a reduced-dimensional representation. The concept of Explicit Semantic Analysis (ESA) by [Gabrilovich, 2007] uses a term/document matrix that differs from the standard vector model by representing abstract concepts as vector dimensions. Other methods or instance the Normalized Google Distance (NGD) [Cilibrasi and Vitanyi, 2007] and Wikipedia (NoW) exist but are not presented in detail.

2.3.1 LSA / PLSA and LDA

[Deerwester et al., 1990] presented latent semantic analysis (LSA) as a means to measure the distance between words or groups of words. Unlike previous techniques that involve creating a word/word co-occurrence matrix, LSA initially generates matrices for word/paragraph, word/document, or word/passage, where a passage is a group of words.

For example, a cell for a word w and a path p contains the frequency of w appearing in p. Next, the matrix's dimension is reduced using singular value decomposition (SVD), representing an unknown summary of concepts. The original word/passage matrix is then recreated from the reduced dimensions.

In probabilistic latent semantic analysis (PLSA) suggested by [Hofmann, 1999] documents in a collection are modeled as a weighted combination of latent topics, denoted as $Z = Z1, ..., ZNz$. Each latent topic is associated with a probabilistic language model $P(w|z)$, which denotes the probability of generating word w from topic z. Each document d in the collection is derived from a weighted mixture of the latent models of the topics, where the generative model PLSA is utilized s the generative model PLSA is:

$$P(d, w) = P(d)\sum_{z} P(w|z)P(z|d) \tag{11}$$

$$P(d, w) = \sum_{z} P(w|z)P(d|z)P(z) \tag{12}$$

2.3.2 The latent Dirichlet distribution

The latent dirichlet distribution (LDA) developed by Blei [2003] is an extension of PLSA. In PLSA, the estimation of $P(zkdi)$ for document di is replaced by a Dirichlet distribution over all possible distributions of latent topics within Z. However, all three methods, including LDA, suffer from the limitation of being "black boxes."

2.3.3 Explicit semantic analysis

Explicit semantic analysis (ESA) is a technique for representing text through vectors that employs Wikipedia as a source of knowledge [Gabrilovich, 2007]. Specifically, ESA represents a single word as a column vector

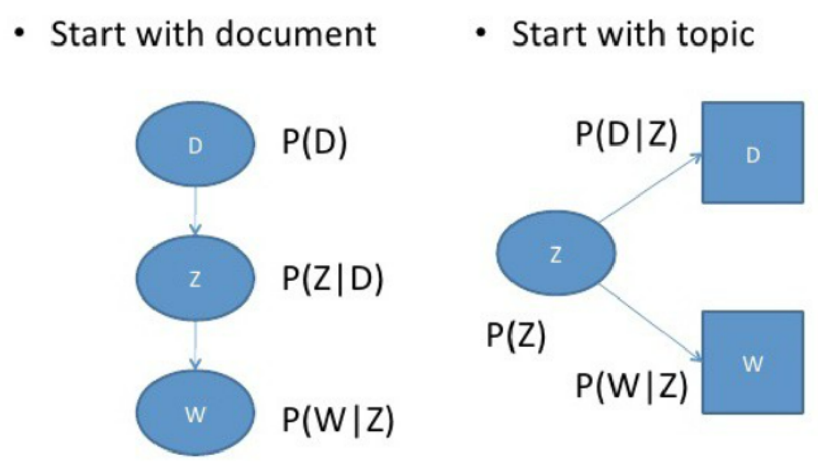


Figure 4: probabilistic latent semantic analysis

in the tf-idf matrix of the corresponding Wikipedia article, while a document is served as the center of the vectors representing its constituent words. Although ESA assumes that Wikipedia articles are "orthogonal", research has demonstrated that it can enhance the effectiveness of information retrieval systems even when applied to non-orthogonal corpora like the Reuters dataset.

2.3.4 Summary

Here we attempt to identify the performance of each measure, whether semantic or syntactic. Grefenstette [2009] praises the merits of the semantic vector approach. According to the authors this approach easily detects similar documents with few errors. Bisson and Hussain [2008] indicates that bi-clustering appears to be better than LSA and even better than the cosine-based measure.

Although Mihalcea [2006] assumes that the semantic methods have similar performance, Budanitsky and Hirst [2006] tends to order them by considering that Jiang and Conrath's method performs better than the others. The sequence of techniques beginning with Lin's method, then Leacock and Chodorow's method, followed by Resnik's method and Jiang and Conrath's method, suggests that the node-based methods achieve superior outcomes when compared to other approaches.

Node-based methods seem to have better results than edge-based methods. Mihalcea [2006] suggests that the metrics developed by Jiang and Conrath, Leacock and Chodorow, Lin, Wu, and Palmer, as well as Resnik, are more effective than vector-based approaches.

Mohler and Mihalcea [2009] demonstrates that knowledge-based and corpus-based measures (e.g. LSA and ESA) produce similar outcomes. However, corpus-based techniques have an advantage over knowledge-based approaches as they are language-independent and allow for the creation of domain-specific corpora with relative ease, compared to using a taxonomy like Wordnet.

Finally, Iosif and Potamianos [2015] describes that the metrics used to measure similarity in the study of Jiang and Conrath and Leacock and Chodorow perform much better than the measures based on Jaccard's coffecient and the Dice index.

2.4 Siamese neural networks

Siamese neural networks are network category that is utilized for extracting features and measuring the similarity between two separate inputs, namely $X1$ and $X2$, that share a certain similarity relationship. These networks are composed of twin networks that are identical in structure and share the same parameters and weights. The twin networks generate vector representations, namely $V1$ and $V2$, which are compared applying a similarity metric, similarly the cosine of the angle between two vectors or the Euclidean distance. Several research works have utilized Siamese neural networks, including [Balntas et al., 2016],[Han et al., 2016], have been interested in image matching processes and have developed siamese neural architectures that take pairs of images. [Han et al., 2016] proposed the MatchNet architecture, which is a two-branch siamese network that first extracts features from each thumbnail pair and then computes the similarity between them using a fully connected network,developed the LIFT (Learned Invariant Feature Transform) that uses a four-branch siamese network to extract feature points and measure similarity using Euclidean distance.[Balntas et al., 2016] employed a neural approach to estimate the geometric transformation between two image pairs. Their neural network consists of a siamese network to match image pairs, followed by a convolutional neural network which estimates the parameters of the geometric transformation between them. Siamese networks are advantageous in that they can learn image similarity effectively.

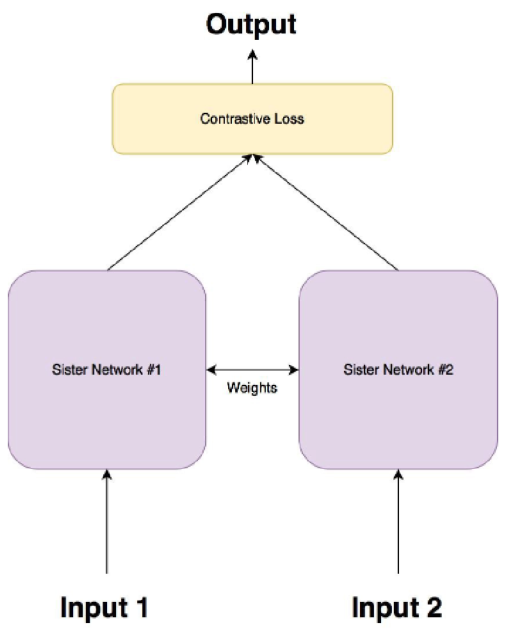


Figure 5: Siamese neural networks

2.5 Machine Learning

Machine learning [Biamonte et al., 2017]is the automatic learning from data without explicit programming. This means that, instead of finding rules to automate a task, we give data to an algorithm that will define these rules itself. These data are representative of the problem we want to solve.

This combination of data and algorithms is what will allow us to develop analytical models to discover new knowledge in the data and extract value from it. And then, build prediction or classification tools to provide decision support.

2.5.1 Situate machine learning

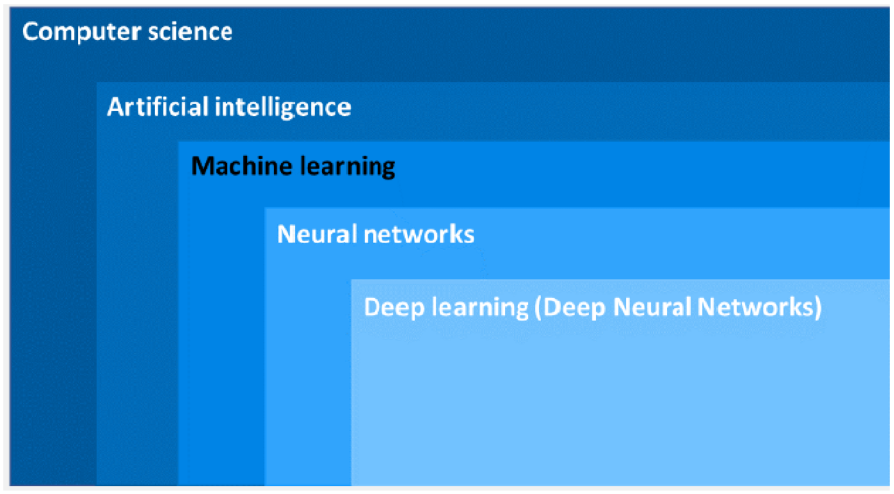


Figure 6: Where machine learning is at

In the field of computer science, machine learning [Shinde and Shah, 2018]is seen as data-driven artificial intelligence. Thus, AI is not necessarily only machine learning, although, the latter can make it more practical. As in the case of language understanding or object recognition.

Neural networks are a sub-category of machine learning, whose information processing model is stimulated by biological nervous systems. As for deep learning, it is an approach to machine learning based on neural networks following a sequence of several levels of abstraction called "layers", hence the name deep learning.

2.5.2 The learning process

The learning process has three main phases: data preparation, model building and the evaluation and validation phase.

The first step in data preparation consists in:

- Collecting data that can be from different sources (data warehouses, web, sensors, etc.) and of different types (numerical, textual, images, SQL, graphs, sequences, etc.).
- Clean, prepare and normalize the data to be able to transmit them to the learning algorithms.

The second step in the model construction comprises:

- Choosing the algorithm best suited to the problem and the nature of training data.
- Performing the training by providing the prepared data to the algorithm to construct a model.
- Evaluate and improve the system according to the results obtained with the test data.

Evaluation and validation This step consists of:

- Observing the efficiency of the model on new data to see if it performs the predictions or rankings according to the desired results.
- Sometimes, ask the opinion of a business expert to validate the model or otherwise see the points that have not been considered.
- Integrate the solution, make predictions, publish the solution, create business strategies.
- Note that one can go back, add new data sets and redo new experiments, to compare the results procured with those of the experiments done before until a satisfactory result is obtained.

2.5.3 Algorithm types

Let's now talk about the different types of algorithms; the choice of the type of algorithm will depend on the problem and the nature of the training data.

- Linear, non-linear: Linear classification algorithms assume that classes can be separated by a hyperplane to answer a problem. In the case of non-linearly separable data, the algorithms are based on non-linear activation functions. In the following example we have three types of flowers (Setosa, Virginica and Versicolor) represented by the length and width of their petals and sepals. The Setosa and Versicolor flowers are very well separated linearly. The two types Versicolor and Virginica share some characteristics even though they are of different categories. So if we use a linear algorithm to classify these data, it separates the data only into two classes and considers the two types Versicolor and Virginica as if they are the same type when they are not. Nevertheless, a non-linear algorithm can find a separation of the three flower types in a non-linear projection.

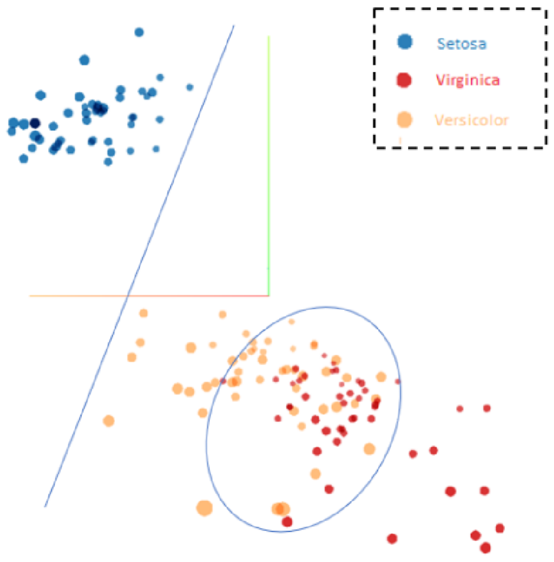


Figure 7: Example of linear and non-linear classification

- Parametric, non-parametric: Parametric algorithms require the prior configuration of a fixed parameter values, the choice of these parameters is made according to one or several hypotheses that are formulated on the distribution of the data. The result obtained will depend on the chosen parameters. Non-parametric algorithms have their own functions that adaptively define the parameters according to the size and nature of the data sets and the training conditions. In this way they are able to match the input representation to the output representation in an appropriate way.
- Incremental, offline: In the case of incremental learning, algorithms build a model from the data, which they can improve incrementally, continuously, if exposed to more data over time. These algorithms are often embedded in systems that respond autonomously. However, off-line algorithms that do not allow for model updating, require total relearning with the original data used, in addition to new data, which may contain new representative features. In the case of language recognition, an offline model that has already learned to recognize French and English but does not understand Spanish cannot learn Spanish separately. It needs a corpus of the Spanish language combined with the French and English corpora to learn all three languages again. An incremental scheme, on the other hand, records the features of the newly detected language and uses them in its next identification attempt.

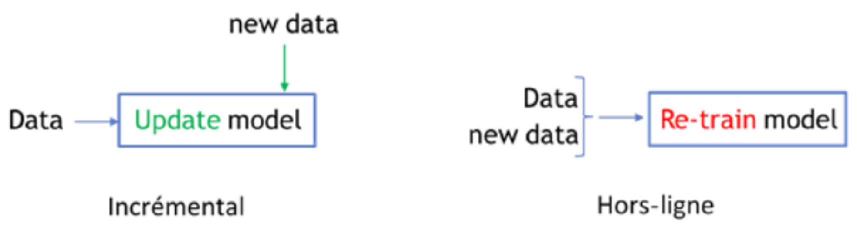


Figure 8: Example of Incremental, offline

- Probabilistic, Geometric: probabilistic algorithms provide the distribution of probabilities of an input example belonging to each class. While geometric algorithms do not model the distribution of classes, but separate the space by functions and return the class associated with the space from which a sample of data comes.

- Algorithm with supervised learning: This type of algorithm receives as training data a set of inputs with their known output equivalents. It is up to them to optimize the "input -> output" relationship and generalize it to unknown inputs.

  Two different types of supervised learning are distinguished:

- Classification: In this case, the principle is to assign a class to each unknown data sample.

- Regression: In this case, the algorithm does not predict a membership class but a continuous value. As for example the case of sales prediction, the algorithm predicts the approximate number of units that will be sold per day.

- Unsupervised Learning Algorithm: Unsupervised learning allows to discover relevant patterns and complex trends in the data. In this case the data is not labeled. So we have no prior information on the output, we are mainly trying to understand the projection of the data in space. Two categories can be distinguished for this type of learning:

- Clustering: Which allows the grouping of large datasets in a way that the objects in the same group, called a cluster, share important similarities with each other.

- Association rules: Which allows to discover regularities in a set of instances in order to detect important co-occurrences of elements of this set.

  This type of learning is for example used for recommendation systems.

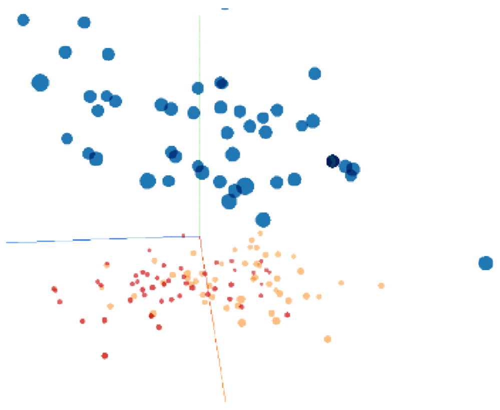


Figure 9: Example of Clustering

3 Bert

In this portion, the structure and training method employed to develop BERT are explained. BERT is a lexical embedding neural model that utilizes a sequence of Transformer layers to generate contextualized representations.Vaswani et al. [2017a].

The training of this model involves two tasks, which are Masked language modeling (MLM) and Next sentence prediction (NSP). The process is divided into two distinct phases throughout all the steps:

- During the pre-training step, the model is trained on both MLM and NSP tasks to acquire the ability to generate contextualized embeddings.
- In the task adaptation phase, BERT is employed as an embedding generator within a larger model that is entirely trained on a specific target task.

3.1 Model description

3.1.1 Segmentation and representation of the input

In contrast to traditional word-based approaches, BERT uses a vocabulary comprising a mixture of words and subwords called wordpieces [Wu et al., 2016]. By dividing unknown words into a series of subwords that belong to the vocabulary, BERT can effectively handle the issue of out-of-vocabulary words. The input plungers are the sum of wordpiece plunges (word/subword), segment plunges (sentence A/sentence B) and positional plunges.During the pre-training phase, BERT systematically takes a couple of sentences as input. These sentences are then segmented either into words or into sub-words, The sequence commences with a distinctive [CLS] symbol and concludes with [SEP] symbols following each sentence.

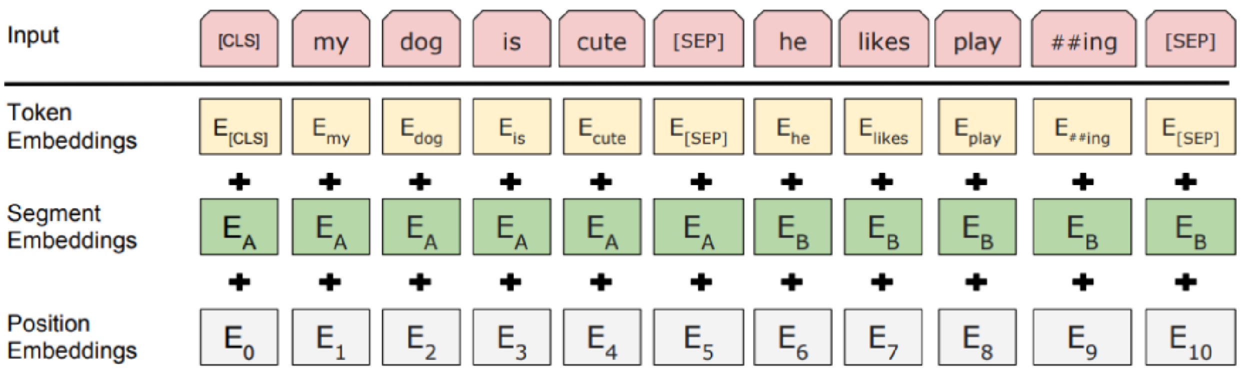


Figure 10: Representation of the BERT input. The + are term to term sums

Each element of the input is then represented by a vector (token embedding) from a folding matrix. Then, in order to inject a notion of position and to distinguish more easily the elements coming from each of the input sentences, we add to this initial vector a position embedding and a segment embedding.

3.1.2 Contextualization and Transform layers

BERT transforms each pair of input sentences into a sequence of vectors according to the procedure described above. At this level, the representation of each element is completely independent of the others. The next step is then to use a series of L layer Transformers layers applying the same transformation in order to iteratively produce representations that are more and more contextualized.

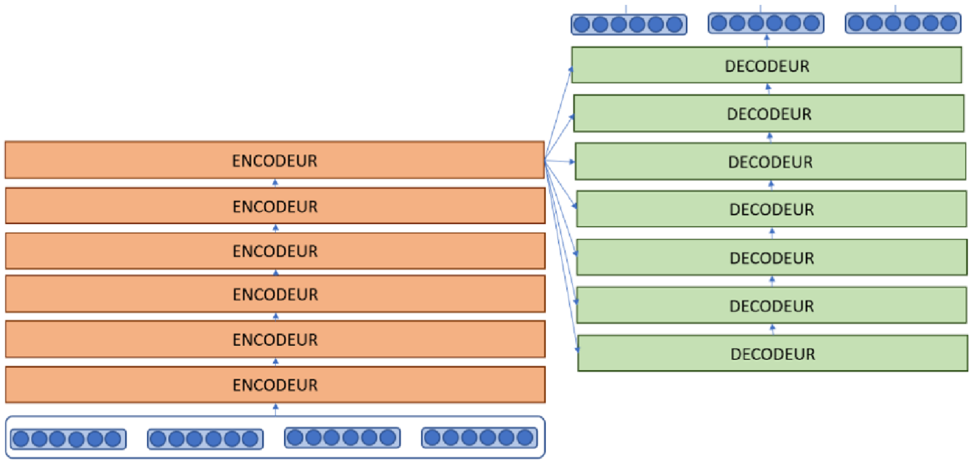


Figure 11: Schematic of a Transformer layer in BERT

- Six stacked encoders (the Nx in the diagram), The architecture comprises of a series of encoders, with each encoder utilizing the output of the previous encoder (except the first one, which employs embeddings). This is then followed by six stacked decoders that apply the output prior decoder and the last encoder as input (with the exception of the first decoder, which only uses the output of the last decoder). It's essential to note that the 12 (or 24, depending on the BERT version) blocks don't have the same weight matrices.

- Each decoder is made up of three layers: a self-attention layer with various heads, thereafter an attention layer utilizing the last encoder's output, and then a fully connected and position-wise FFN (i.e. each element of the output vector of the previous layer is connected to a formal neuron of the FFN input, in the same order as they are in the vector). Each sub-layer has an output layer that adds, sums, the outputs of the layer and the connection to a so-called residual connection (which directly connects the input values of the layer to the output of the layer) and normalizes the whole. FNN: Formal neural layers with a ReLU as activation function, formalized by :

$$FFN(x) = max(0, xW1 + b1)W2 + b2 \tag{13}$$

- Three "key-value" type attention mechanisms. Auto-attention in the encoder, auto-attention with all the elements previously generated in the input of the decoder, then always in the decoder, "masked attention" between the element to be generated in the decoder and all the elements of the encoder. Note that the attention layers have several heads (Multi-head), we will also provide more details in the following section.

- For text generation or simultaneous translation the mechanism is auto-regressive, i.e. a sequence is entered into the first encoder, the output predicts an element, then the whole sequence is passed back through the encoder and the element predicted in the decoder in parallel to generate a second element, then again the sequence in the encoder and all the elements already predicted in the decoder in parallel and so on, until an <end of sequence> is predicted in output.

- In output a probability distribution that predicts the most likely output element.

3.1.3 Multihead attention

Given the global context, the multi-head attention layer weights each representation against the rest of the input representations. This depends on parameters specific to each attention layer which are adjusted to the task at the time of training the model.

Moreover, in order to capture different signals the multi-head attention layer relies on several "heads" that compute each time a different weighting of the inputs. Finally, the set of weights is concatenated

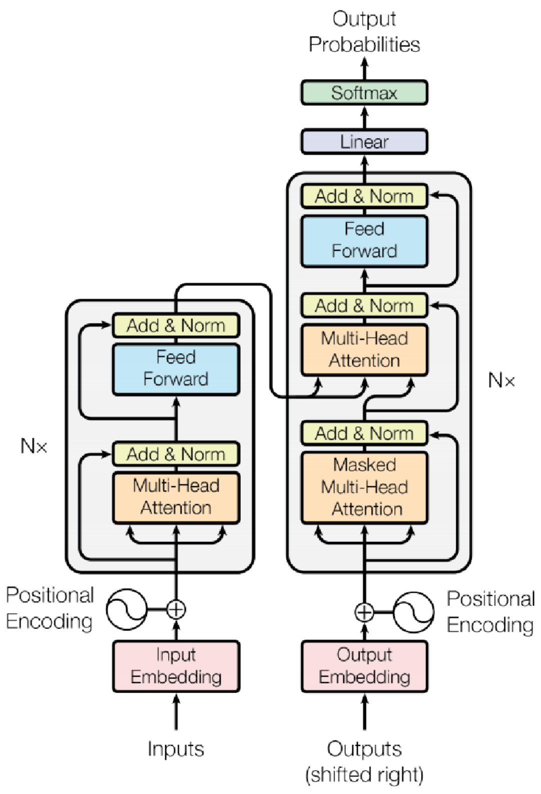


Figure 12: Transformer - model architecture

and then projected to produce the output representations of the attention layer. Each head of attention has its own matrices Wqi, Wki, Wvi.

We concatenate the output of each head to find a matrix of dimension [length of the sequence] $\times$ [dimension of the embeddings, i.e. 512].

X is input matrix, for each head in parallel, $X.Wqi = Qi$ is computed for head i, $X.Wki = Ki$for head i, $X.Wvi = Vi$ for head i. Then, the attention for head is calculated i Attention(Qi, Ki, Vi) with the following formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{14}$$

Where $\sqrt{d_k}$ corresponds to the key and query vector dimension k and q, respectively.

[Q x Transpose of K] is a scalar product between the query vectors and the key vectors. As we have just seen, the more the key will "resemble" the query, the higher the score produced by [Q x Transpose of K] will be for this key. The dk part (=64) is simply there to normalize, it is not always used in attention mechanisms. The softmax will give a probability distribution that will further increase the value for keys similar to the queries, and decrease those for keys dissimilar to the queries.

Finally, the keys correspond to values, when we multiply the previous result by V, the values corresponding to the keys that were "elected" in the previous step are overweighted compared to the other values. Finally, the output is concatenated for each head and multiplied by a matrix W0, of dimensions 512 x 512 ([(number of heads) x (dimension of query or key or value, i.e. 64)]x[dimension of embeddings]), which learns to project the result over an output space of expected dimensions.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{15}$$

where

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

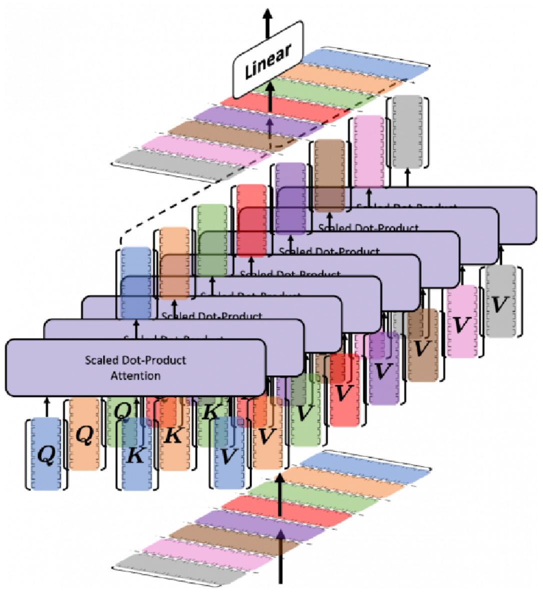


Figure 13: Multi-Head Attention

3.1.4 Inputs

The embeddings: each word is characterized by a vector (column or row of reals), here of dimension 512, which we will note dmodel as in the paper [Vaswani et al., 2017a].

We can add to these embeddings, for each word, the embeddings of a "segment" when it makes sense (for example each sentence is a segment and we want to pass several sentences at the same time, we will then say in which segment each word is).

Then we add "positional encoding", which is a way to encode the place of each element in the sequence. As the length of the sentences is not predetermined, we will use sinusoidal functions giving small values between 0 and 1, to slightly modify the embeddings of each word. The dimension of the position embedding (to be summed with the semantic embedding of the word) is the same as that of the semantic embedding, i.e. 512, to be able to sum term by term. Note that there are many ways to encode the position of an element in a sequence. This gives as input a matrix of size [length of the sequence] x [dimension of the embeddings - 512], see figure 10.

3.1.5 linear layer and final softmax for output probabilities

Let us call S the output matrix of the decoder. We multiply it by a matrix of weights (which can learn) W1. This is entirely interacted layer that simply projects the previous output into a space the size of our vocabulary. $W1$ is the matrix that will extract a word from our vocabulary dictionary. Its dimensions are therefore [dimension of the embeddings, i.e. dmodel] x [number of words in our vocabulary] [Vaswani et al., 2017b].

The Feed Forward output layer passes through a linear layer that deals with a classifier. The classifier is as large as the classe values we acquire. Finally, probability scores will be formed by a softmax layer. The highest probability score index is taken, and that is equivalent to our estimated word.

Afterward, the decoder collects the output, joins it to the decoder's input list, and pursues with decoding again until a token is estimated. In our case, the final class presents the highest probability estimation, which is designated to the end token.

3.2 Training process

In general, we train BERT to produce contextualized representations that are user-friendly for a broad range of NLP tasks. Two tasks are trained:

- a masked language modeling (MLM) task.
- a next sentence prediction (NSP) task.

3.2.1 Masked language modeling

In contrast to classical language modeling tasks, which predict the next word based on the previously observed words, BERT is trained on a task where a random word from the input text is masked, and the objective is to estimate the masked word. This task enables the model to learn highly contextualized representations as the Transformer architecture considers both left and right contexts of the target word. As a result, the representations learned by BERT are more contextualized than those of unidirectional models such as ELMo, which concatenate unidirectional representations.

During this task, the target word can be replaced by a special symbol [MASK], additional random word, or kept as is.

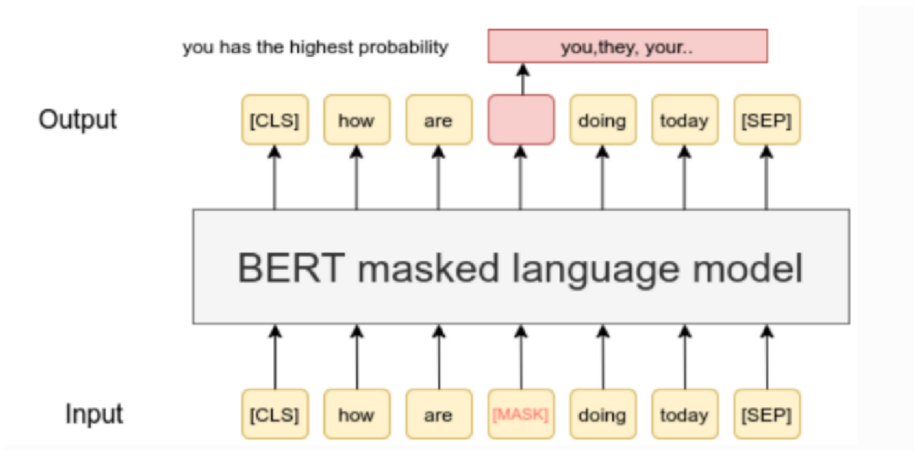


Figure 14: Masked language modeling

3.2.2 Next sentence prediction

Additionally, for the next sentence estimation task, BERT is trained where it indicates whether two input sentences are successive. The purpose of this task is to enhance the efficiency system on tasks that need assessing the relationship between a pair of sentences.

In this task, the [CLS] symbol representation is utilized to classify each input sentence pair and any other classification task post-training.

4 Modern Embedding technique

This segment will provide a summary of contemporary embedding techniques and their various applications in diverse fields. Among the different models of non-contextualized embedding, the best known are Word2Vec and Glove. The latter (referring to the previous sentence or context) disregards the sequence of words and their significance in the text. To convert a word into a vector, it is not necessary to have the complete model but only the list of words and their associated vectors. This makes the transformation of a word into a vector very fast but limited to the words included in this list.

Then came the contextualized embedding models ELMo, one of the best known, considering the position and context of the word in the sentence to create the vector representation. The possibility to contextualize the word has nevertheless a disadvantage: it is necessary to use the model directly to embed to directly use the model to embed a word, contrary to the non contextualized models, generating higher computation times.

The most recent architecture is the Transformers architecture, introduced by [Vaswani et al., 2017b].This architecture has broken the latest literature review records. It is based on an "attention" layer which considerably increases the memory length of the model, allowing contextualization on the whole words of a document. Unlike a basic contextualized model, the Transformers architecture uses an encoder / decoder system, specifically very efficient in translation tasks. The encoder, trained to understand a particular language, creates a neutral vector representation of a word or document.

The decoder, also trained in a particular language (which is not necessarily the same as the encoder), takes this vector and transforms it to obtain the expected result. This architecture allows to graft any decoder on the encoder, making it more easily generalize to all languages. This step of the "neutral" vector,which is not available in the other architectures, forces the encoder to create a vector representation as faithful to the word as possible, due to the attention layer. Currently, the most known and used model of Transformers model is BERT.

4.1 GloVe

GloVe Pennington et al. [2014] is an unsupervised learning algorithm that generates word-vector representations. The algorithm operates on overall word by word co-occurrence statistics from the corpus, producing vector representations that exhibit intriguing linear substructures within the word vector space.

GloVe is a model that applies a weighted log-bilinear approach to solve a least-squares optimization problem. The key insight driving the model is the idea that the ratios of word-word co-occurrence probabilities contain meaningful information. This is why the resulting word vectors are effective for tasks that involve comparing words to one another.

The GloVe algorithm includes the following steps:

1. To generate the $MCOM$ word co-occurrence matrix.Each element in this matrix $MCOM_{ij}$ represents the frequency with which the word denotes the frequency of the occurrence of the word wi in the context of the word wj. To calculate this frequency, a search is conducted for terms within a specified window before and after the term. The words farther away from the term are assigned lower weightage, which is usually determined by a formula [Pennington et al., 2014]. Remoteness represents the distance between two words in a document expressed as a number of words. GloVe distance:

$$weight = 1/distance \tag{16}$$

2. Create flexible restrictions for each pair of words:

$$w_i^T w_j + b_i + b_j = log(MCOM_{ij}) \tag{17}$$

   Where bi and bj are the bias vectors associated with the vectors wi and wj.

3. Define Cost Function GloVe cost function:

$$J = \sum_{i=1}^{n} \sum_{j=1}^{n} f(MCOM_{ij}) w_i^T w_j + b_i + b_j - log(MCOM_{ij})^2 \tag{18}$$

   The GloVe weighting function is defined as the transpose of the vector wi, denoted as T, multiplied by the vector wj, where n represents the size of the vocabulary in terms of word numbers.

$$x = \begin{cases} (MCOM_{ij}/x_{max})^{\alpha} & \text{if} MCOM_{ij} < x_{max} \\ 1 & \text{otherwise} \end{cases} \tag{19}$$

   x_{max} is the maximum value of co-occurrence allowed, otherwise the function returns directly the value 1. In the other cases, the value returned is between 0 and 1 and α is a coefficient to adjust the distribution of weights; a common value is 0,75.

4.2 Word2vec

In recent years, artificial neural approaches are widely used. Word2vec is a heuristic implemented with three layers of neurons [Mikolov et al., 2013]. It is able to represent a word by a dense vector of controlled size. The first layer receives the set of texts in the corpus as vectors. These vectors are built from the sequence of the word indexes. The output of Word2vec is a set of feature vectors that correspond words of this corpus.

Word2vec represents in a first step documents in the form of a sequence and then projects them into the large in the high dimensional description space. Word2vec then groups the similar vector words into a vector space.

The Word2vec artificial neural networks build, for each word, a context window. Within this window, all words are treated equally. It works without human intervention. In this sense, it is an unsupervised algorithm. Even though Word2vec is not classified as a deep neural network, its main purpose is to transform text into a representation that can be understood by neural networks.

Auto-encoders are designed to reproduce their inputs identically. This is of little interest at first sight! However, the knowledge acquired by the hidden layers for this purpose can be very interesting. Word2vec can be compared to an auto-encoder as it encodes each word into a vector representation. The algorithm learns by training the vectors against neighboring words in the input corpus.

There are two methods by which it accomplishes this: one is by utilizing the surrounding context to anticipate a specific word (which is referred to as the Continuous Bag of Words method) (figure 15).

Word2Vec CBOW [Mikolov et al., 2013] formula:

$$-\sum_{t=1}^{T} logP(w_t|w_{t-m}, ...w_{t-1}, w_{t+1}, ..., w_{t+m}), \tag{20}$$

Alternatively, it can predict a particular context by using a word, which is referred to as the skip-gram method target context, called skip-gram (figure 16). The first method is faster than the second, However, when dealing with large data sets, the skip-gram method tends to yield more precise outcomes. The purpose of training the skip-gram model is to identify uninterrupted word representations that are effective in anticipating the neighboring words within a sentence or document. To achieve this, the skip-gram model is designed to minimize the log likelihood error function, given a series of words represented by $w1, w2, w3, ..., wT$. The objective of the Skip-gram model is to minimize the log likelihood error function. Word2Vec Skip-gram [Mikolov et al., 2013] formula:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logP(w_{t+j}|w_t) \tag{21}$$

In this case, T refers to the total word numbers in the given sequence, while m denotes the dimensionality of the context.

4.3 FastText

The fastText approach developed by [Joulin et al., 2016] is an expansion of the Word2vec technique that aims to enhance the vector representations of words for languages with intricate morphology. This is achieved by utilizing a vector representation of n-grams, where a word is corresponded to the summation of its n-gram vectors. In practical terms, rather than directly learning word vectors as input, fastText denotes each word as a collection of character n-grams. To illustrate, let's consider the word "artificial" and set n to 3. In this case, the fastText representation of "artificial" will be determined by nine trigrams: "ar," "art," "rti," "tif," "ifi," "fic," "ici," "ial," and "al." The approach enables us to obtain a representation for a word that wasn't present in the original training data, which is especially useful for smaller corpora. Ultimately, every word or subword within a document is denoted by a vector of real numbers.

A document will be represented by a matrix with two dimensions. The first dimension is the length of the chosen numerical sequence. The matrix's second dimension represents the number of dimensions within the word description space. If the word order relation is well respected in the vectors representing the documents,

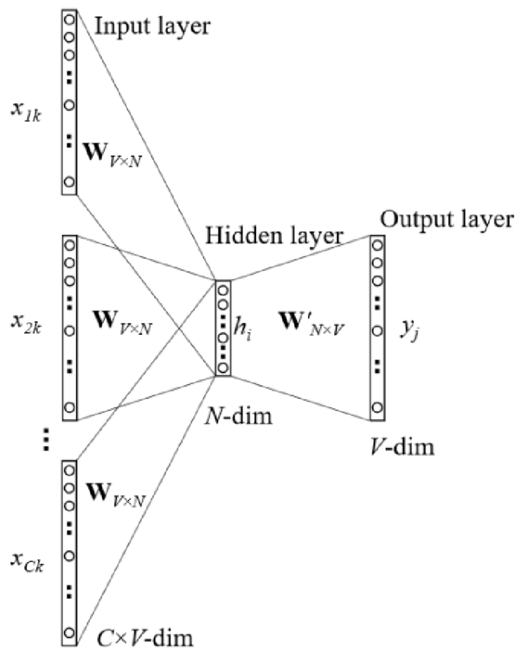


Figure 15: Continuous Bag-of-Word Model

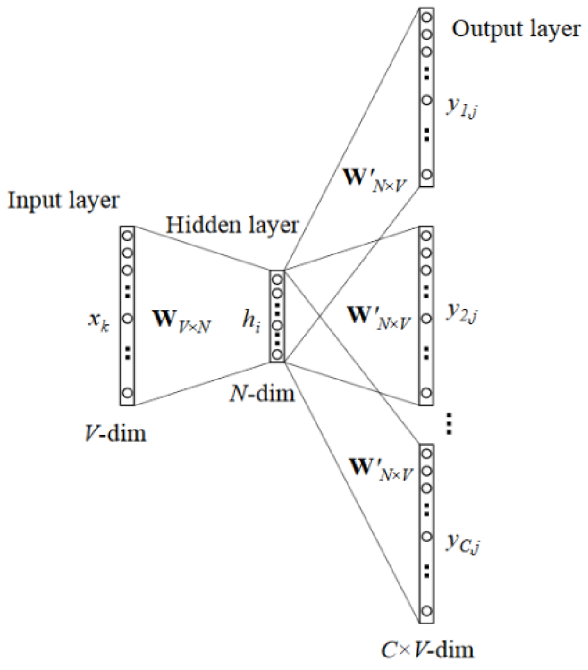


Figure 16: Skip-gram Model

Figure 17: Word2vec models for obtaining word embeddings

the representation of the words is similar in the whole corpus whatever the document is. Thus, like GloVe or Word2vec, fastText is a method for continuous representation of static words.

4.4 Application of Word Embedding

The areas of application of artificial intelligence in our daily life are almost unlimited. Therefore, as an AI-related technology,word embedding can help in many tasks.We have grouped these applications into three main families, which correspond to document reading supports, document production tools, and finally man-machine interfaces.

4.4.1 Document processing

Word embeddings are commonly used to enhance human comprehension of the vast resources available in natural language. This is achieved through various applications, including:

- Machine translation (or machine translation assistance). This application, which historically prompted the first research efforts in NLP, it is important to note that even if complete domain-independent translation is still out of reach, it is possible to obtain good specialized translators (technical fields), which are an effective way to prepare for manual intervention by a translator. There are also working environments that provide lexical resources (extensive bilingual dictionaries) for translation.
- The search for interesting documents in documentary databases. The proliferation of documentary search tools of documentary research tools on the web, which process millions of requests every day, show the importance of the demand in this field. The performance of these engines shows how far we can go in this field. More and more tools also provide spontaneous search tools for potentially interesting addresses (based on user profiles), or automatic monitoring of publications in given fields.
- Routing, classification or automatic indexing of electronic documents are application variants of the document search paradigm.
- More complex is the task of finding (or producing on demand) precise answers to the user's questions ("question-answer" task). Here again, technical solutions exist.

4.4.2 Production of documents

If so many electronic documents are available today, it is because someone wrote them. In the field of text production support (text generation), word embedding applications are also numerous:

- Self-correcting" keyboards (for example for the disabled);

- Optical character recognition; Many commercial systems are available today, with very satisfactory performances: Recognita, Omnipage, ScanWorX... ;

- Automatic generation of documents from formal specifications; In fact, many sectors of activity involve the massive production of very stereotyped texts from more or less formal specifications (legal texts, database exploration reports, statistical analysis reports, technical documentation, etc.). For this class of documents, it is perfectly possible to automatically generate, if not completely final texts, at least preliminary versions that will be finalized by human editors. We find in these applications the same dialectic as in applications designed to facilitate document management. On the one hand, there are applications with broad coverage, which essentially use lexical resources, with tolerant access functions (allowing the correction of errors) to the lexicon: this is the case of applications that revolve around spelling correction. On the other hand, applications that integrate higher level processing mechanisms (typically generation), but which only work for much more restricted domains.

4.4.3 Natural interfaces

Last application domain, which is undoubtedly the one in which the demand for linguistic processing is the strongest, the domain of natural interfaces such as:

- Natural language querying of databases (natural language translation SQL) or web search engines. Many applications of this type are beginning to be set up on the web.

- Voice interfaces, which implement, in a variable manner depending on the application, modules for speech recognition, speech synthesis, dialogue generation and management, access to knowledge bases each of these modules requires specific processing (morpho-syntactic disambiguation and syntagm identification for synthesis, stochastic grammars for speech recognition). The first "mass-market" voice dictation systems are starting to arrive on the market (IBM's Via Voice, Dragon Dictate, and many others). In the next few years, the integration of a speech processing API in Windows should literally explode the market for voice technologies, and the market for speech technologies. Many commercial services already exist, or are close to commercialization that use all of these technique (recognition-dialogue-synthesis), for various applications such as "hands-free" computers, telephone reading of mails electronic, ticket reservation (train, plane), the list is almost limitless.

Part II

Practical part

5 Semantic similarity applications

In the practice part of this report, we suggest how the embedding technique designed for English can be used also for other languages, for example, the Czech language, and some proposed methods will be evaluated on data. To apply this to the STS (Semantic Textual Similarity) task, we used pre-trained language models like BERT and CzechBERT, which are designed to generate high-quality sentence embeddings that capture the semantic meaning of the text.

Initially, we will explore how the Sentence Transformers library can enable a BERT model to learn the STS task. This library simplifies the process of fine-tuning a pre-trained BERT model on a dataset consisting of pairs of sentences and their corresponding similarity scores. The objective is to train the model to generate embeddings for two sentences and then estimate their similarity score. Moreover, we aim to compute the similarity matrix for a given list of sentences to display the computed similarity scores between each sentence pair.

Before proceeding, we will introduce the STS task and the datasets used for this purpose. Specifically, we will employ the STS benchmark dataset, which contains pairs of sentences with similarity scores ranging from 0 to 5.

In the second objective, the CzechBERT model is utilized for evaluating the sentiment analysis in Czech text. The purpose is to conduct sentiment analysis on latest review by producing embeddings for Czech sentences through the CzechBERT model and subsequently refining it with the CSFD dataset, comprising Czech film reviews. The model's enhanced version is then utilized to gauge the similarity between Czech sentences, a pivotal component of sentiment analysis. The model can anticipate the sentiment of the new review, i.e., positive, negative, or neutral, by matching the similarity between the newest review and previously labeled instances in the CSFD dataset.

5.1 Sentence Transformers library

The Sentence Transformers library is an open-source Python library that allows an easy-to-use interface for training, using transformer models such as BERT for various NLP tasks. It was developed by Nils Reimers and Iryna Gurevych from the UKP Lab at Technische Universität Darmstadt.

The library provides pre-trained models that can be fine-tuned on specific datasets for tasks such as text classification, semantic similarity, and question answering. In particular, the library provides a simple and efficient way to fine-tune BERT on the STS (Semantic Textual Similarity) task.

To fine-tune a pre-trained BERT model on the STS task using the Sentence Transformers library, one needs to provide a dataset of sentence pairs with associated similarity scores. The library takes care of the rest, including tokenizing the sentences, generating embeddings for each sentence using the BERT model, and training a new model to predict the score of the similarity between the two embedding sentences.

The fine-tuned model is applied to calculate similarity scores between any two sentences, even those that were not in the training dataset. This makes the Sentence Transformers library a powerful tool for a broad range of NLP tasks that involve comparing the semantic similarity between sentences.

The Transformers library is a powerful resource for researchers and developers in the machine learning domains. It offers a wide range of APIs and tools that provide easy access to advanced pre-trained models. The use of these pre-trained models can significantly reduce the time, resources, and computing expenses required to train a model from scratch.

Another advantage of the Transformers library is its framework interoperability, which allows for different framework uses such as PyTorch, TensorFlow, and JAX at different stages of a model's life. This flexibility enables users to train a model in one framework using just three lines of code and then load it into another framework for inference. Additionally, the models can be exported to formats such as ONNX and TorchScript, which are suitable for deployment in production environments.

The $!pipinstallsentence - transformers$ command is used in a command prompt or terminal to install the Sentence Transformers library, a Python package for generating high-quality sentence embeddings using pre-trained transformer models. $Thepip$ command is a package manager for Python that simplifies the process of installing and managing Python packages. With $pipinstall$, you can easily download and install a package from the Python Package Index (PyPI), which in this case is to install the sentence-transformers package.

5.2 Description of the fine-tuning process

Fine-tuning a BERT model involves including a pre-trained BERT model and training it further on a specific task. The pre-trained BERT model has been trained on large amounts of text data and has learned to encode the meaning of words and sentences in a language model. However, it has not been specifically trained for a particular task. Fine-tuning adapts the pre-trained model to the task at hand by adjusting its weights and biases to better fit the specific dataset.

The process of fine-tuning a BERT model involves several steps, including data preparation, model configuration, training, and evaluation. First, the dataset needs to be prepared by splitting it into training and validation sets and preprocessing it by tokenizing the text and converting it into numerical representations that can be fed into the model. Next, the pre-trained BERT model is configured for a particular task by inserting a new classification layer or modifying the existing layers to fit the output requirements. The model is then trained on the training set using backpropagation and gradient descent to update the weights and biases, minimizing the loss between the predicted and actual values.

The trained model is assessed on the validation set to verify its performance and make adjustments if necessary. It may involve tuning the hyperparameters, as the learning rate, batch size, or epoch numbers, or adjusting the model architecture.

Once the model is fine-tuned and validated, the specific task can be applied.

Fine-tuning a pre-trained BERT model on a dataset of sentence pairs with associated similarity scores involves training the model to predict the similarity score between two sentences, given their embeddings. The process starts with loading the pre-trained BERT model and then fine-tuning it on the specific task at hand using the Sentence Transformers library. During fine-tuning, the pre-trained model weights are updated using backpropagation to minimize the difference between the estimated and the actual similarity scores in the training dataset.

The fine-tuning process involves several steps, including preparing the data by tokenizing and encoding the sentences, defining the loss function, and setting the hyperparameters. Once the model is trained, the performance of the model is cheked by evaluating the validation. If the performance is not satisfactory, the hyperparameters can be adjusted, and the fine-tuning procedure is repeated until the desired performance is achieved.

fine-tuning a pre-trained BERT model involves updating its parameters to adapt to the specific task by minimizing the loss between predicted and actual similarity scores. The purpose is to generate a model that can accurately predict the similarity between two sentences given their embeddings.

5.3 Goal of the fine-tuning process:

The main goal of fine-tuning a BERT model for the STS task is to train the model to accurately predict the similarity score between two input sentences. The model does this by generating embeddings for each sentence and then calculating a similarity score based on the similarity between the two embeddings. These embeddings are learned representations of the input sentences that capture their meaning and context, and are generated by passing the input sentences through the pre-trained BERT model.

During the fine-tuning process, the weights of the pre-trained BERT model are updated using backpropagation to minimize the difference between the predicted similarity scores and the actual similarity scores in the training dataset. This process involves repeatedly presenting the model with pairs of input sentences and their associated similarity scores and adjusting the model's weights to improve its prediction accuracy. By the end of the fine-tuning process, the model should be able to accurately predict the similarity score between two sentences based on their embeddings.

5.4 Task 1

This part discusses how the Sentence Transformers library can be used to train a pre-trained BERT model on a dataset of sentence pairs and their similarity scores. The aim is to generate embeddings for the sentences and then estimate their similarity score. Additionally, the similarity matrix for a list of sentences will be computed to show the similarity scores between each pair of sentences.

5.4.1 Dataset for task 1

The dataset selected for this task is known as the "stsb multi mt" dataset May [2021], which is an NLP dataset consisting of pairs of sentences in multiple languages. The name of the dataset is derived from the fact that it contains sentence pairs in several languages (hence the "multi" in its name) and comes equipped with corresponding similarity scores. The dataset was created with the objective of serving as a resource for cross-lingual semantic similarity tasks and was made publicly available as part of the WMT19 shared task on parallel corpus filtering and cleaning. The sentences in the dataset were procured from various domains and subsequently translated into multiple languages through the use of machine translation. The similarity scores were determined via human annotations using a scale varying from 1 to 5, where higher scores determine higher similarity between the sentence pairs. With over 20,000 sentence pairs available in 14 different languages, the "stsb multi mt" dataset is an invaluable resource for research endeavors in cross-lingual NLP.

5.4.2 Explication of how Sbert model determine Similarity Between a Pair of Texts

Transformers-based models such as BERT, SBERT, or RoBERTa can determine the similarity between two texts by computing the similarity between their respective embeddings. These embeddings are vector representations of text that capture the meaning of words in a continuous vector space, which is computed by passing the text through the encoder of the transformer model. The first step is to convert the input text into a sequence of tokens using a process called tokenization. For BERT models, tokenization involves splitting the text into small chunks, adding special tokens like [CLS] and [SEP], and appending [PAD] tokens based on the maximum sequence length. Once the input is tokenized, the model generates an embedding vector for each token with 768 dimensions. For classification tasks using BERT, the embedding vector for the [CLS] token is typically used as input to a final softmax or sigmoid layer acting as a classifier. Using the workflow

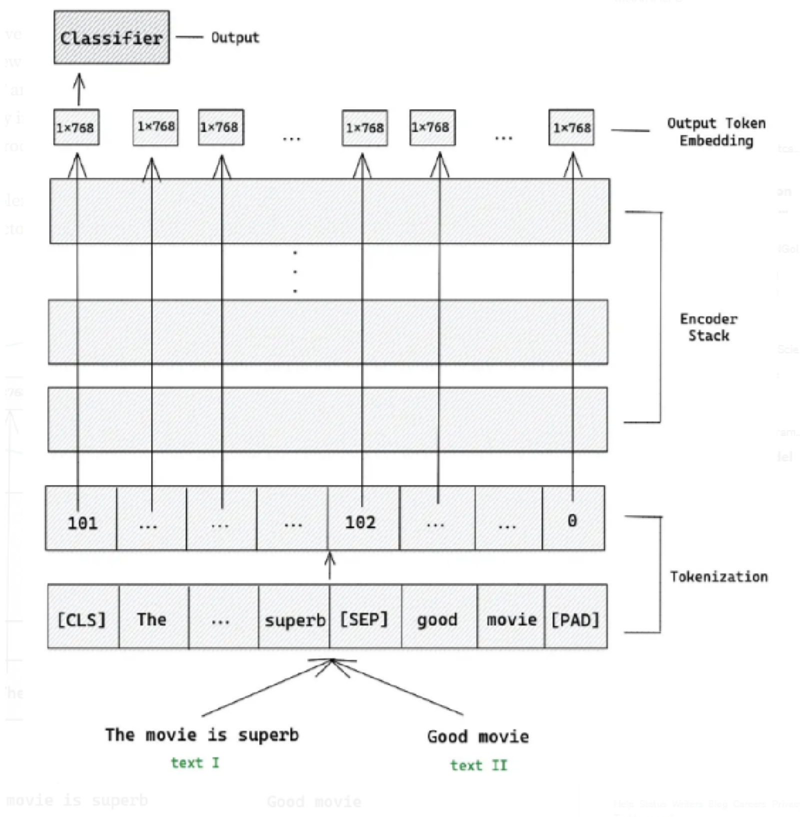


Figure 18: BERT Workflow for STS task

described above, BERT achieved outstanding results on the STS benchmark. However, there is a significant limitation to this approach: scalability. To illustrate, suppose we have a fresh piece of text and want to find the most similar entry to it in our 100,000-entry database. If we utilize the BERT architecture described earlier, we must compare the new text to every entry in the database, which entails tokenization and a forward pass 100,000 times. The primary challenge with scalability is that BERT produces an embedding vector for each token, rather than one for each sentence or piece of text. Assuming BERT could provide us with a useful embedding at the sentence level, we could store the embedding for each entry in our database. Then, when a new piece of text arrives, we would only need to compare its sentence embedding with the sentence embeddings of each entry in our database using cosine similarity, a much quicker method.

This is the issue that Sentence BERT (SBERT) aims to address. SBERT can be thought of as a refined version of BERT that employs a siamese-type model architecture, as illustrated below.

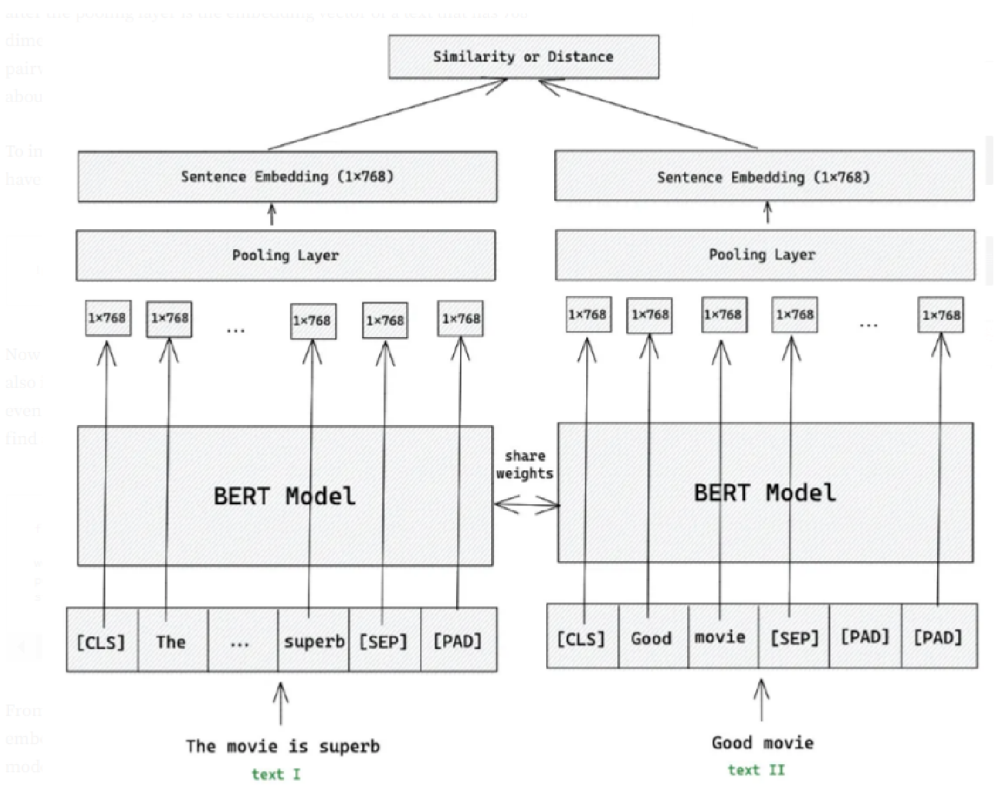
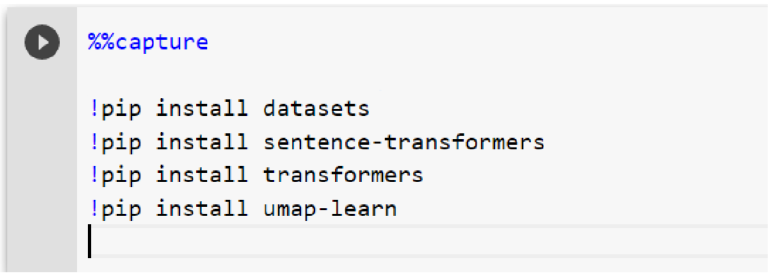


Figure 19: Architecture of SBERT model

5.4.3 Implementation of SBERT model

Algorithm to use Hugging Face's Transformers library to implement SBERT using pre-built models:

- First, we installed the Transformers library and imported the necessary modules.

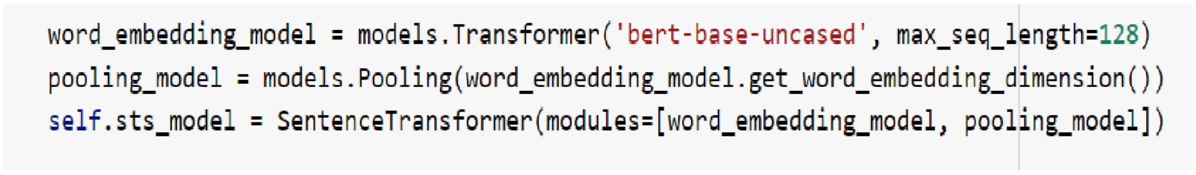


```
%%capture

!pip install datasets
!pip install sentence-transformers
!pip install transformers
!pip install umap-learn
```

Figure 20: Transformers library and necessary modules

The datasets package provides a collection of datasets commonly used in natural language processing (NLP) and machine learning research. The sentence-transformers package is a Python library that allows users to compute embeddings for sentences or documents using various pre-trained models, including BERT and other transformer-based architectures. The transformers package provides a range of pre-trained transformer models for NLP tasks, including BERT and GPT-2. The umap-learn package is a Python library that provides an implementation of Uniform Manifold Approximation and Projection (UMAP), a machine learning algorithm used for dimensionality reduction and visualization.

- We then loaded a pre-trained transformer model for SBERT. In our case, we used the "bert-base-uncased" model.



```
word_embedding_model = models.Transformer('bert-base-uncased', max_seq_length=128)
pooling_model = models.Pooling(word_embedding_model.get_word_embedding_dimension())
self.sts_model = SentenceTransformer(modules=[word_embedding_model, pooling_model])
```

Figure 21: loaded a pre-trained transformer model for SBERT

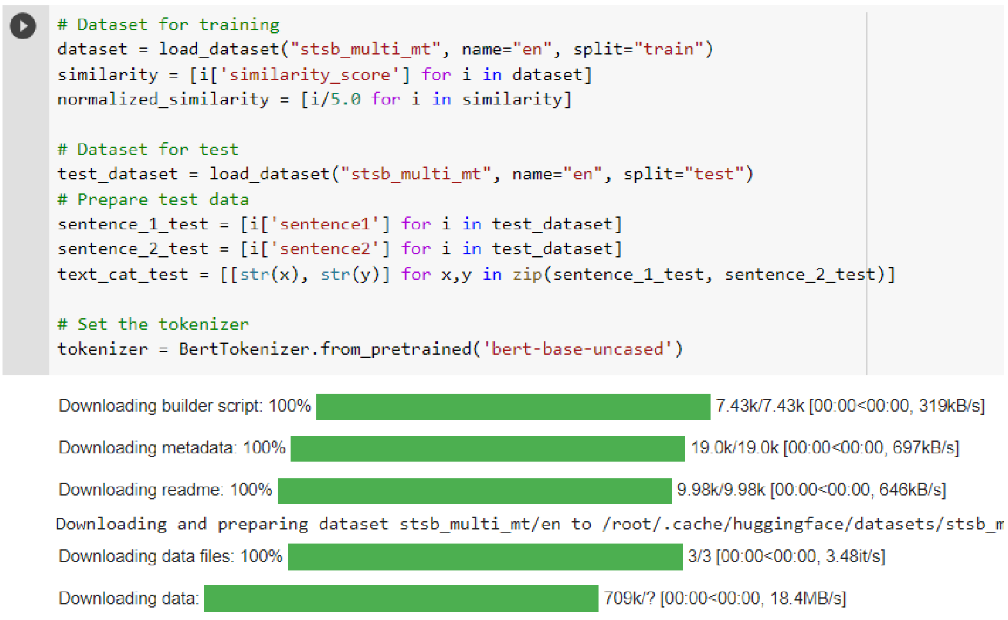
The line word embedding model = models.Transformer('bert-base-uncased', max seq length=128) creates an instance of the Transformer class from the sentence transformers library with the bert-base-uncased pre-trained model as the base model for embedding the input text.

Here, bert-base-uncased refers to a pre-trained BERT model released by the Google Research team, which is trained on a huge corpus of English text and can be fine-tuned for numerous natural language processing tasks. The 'uncased' variant means that the model was trained on lowercased text, so it can handle both lowercase and uppercase input text.

The max seq length argument specifies the maximum number of tokens (i.e., words or subwords) allowed in each input sequence. In this case, the maximum sequence length is set to 128, which is the maximum input length allowed by the BERT model.

The resulting word embedding model object can be used to embed input text using the pre-trained BERT model, which can be applied for numerous natural language processing tasks such as text classification or sentence similarity scoring.

- Load a dataset and generate embeddings for each sentence:



```
# Dataset for training
dataset = load_dataset("stsb_multi_mt", name="en", split="train")
similarity = [i['similarity_score'] for i in dataset]
normalized_similarity = [i/5.0 for i in similarity]

# Dataset for test
test_dataset = load_dataset("stsb_multi_mt", name="en", split="test")
# Prepare test data
sentence_1_test = [i['sentence1'] for i in test_dataset]
sentence_2_test = [i['sentence2'] for i in test_dataset]
text_cat_test = [[str(x), str(y)] for x,y in zip(sentence_1_test, sentence_2_test)]

# Set the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
Downloading builder script: 100%  7.43k/7.43k [00:00<00:00, 319kB/s]
Downloading metadata: 100%  19.0k/19.0k [00:00<00:00, 697kB/s]
Downloading readme: 100%  9.98k/9.98k [00:00<00:00, 646kB/s]
Downloading and preparing dataset stsb_multi_mt/en to /root/.cache/huggingface/datasets/stsb_m
Downloading data files: 100%  3/3 [00:00<00:00, 3.48it/s]
Downloading data:  709k/? [00:00<00:00, 18.4MB/s]
```

Figure 22: Load a dataset

This code block involves the preparation of data for training and testing a machine learning model for sentence similarity.

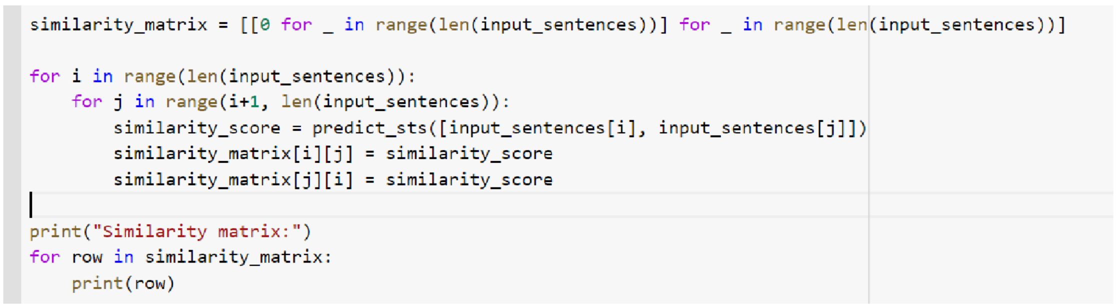
load dataset is a function from the datasets package that loads the $stsb_multi_mt$ dataset in English language for training and testing. split parameter specifies the data split, which is 'train' for training data and 'test' for testing data. The test dataset is loaded from the same 'stsb multi mt' dataset for testing the trained model.

The sentence 1 test and sentence 2 test lists are created by extracting the first and second sentences of each pair from the test dataset.

The text cat test list is created by combining the sentences of each pair as a list of two strings.

The tokenizer is set to the BertTokenizer from the $bert - base - uncased$ pre trained model. It is used to tokenize the input sentences into a format that can be understood by the model.

- Next we define a function to compute similarity between pairs of sentence:



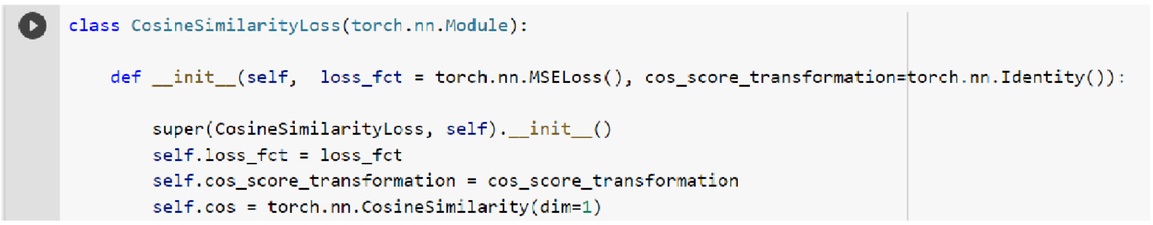
```
similarity_matrix = [[0 for _ in range(len(input_sentences))] for _ in range(len(input_sentences))]

for i in range(len(input_sentences)):
    for j in range(i+1, len(input_sentences)):
        similarity_score = predict_sts([input_sentences[i], input_sentences[j]])
        similarity_matrix[i][j] = similarity_score
        similarity_matrix[j][i] = similarity_score

print("Similarity matrix:")
for row in similarity_matrix:
    print(row)
```

Figure 23: Function to compute similarity between pairs of sentence

there is a list of dictionaries called dataset, where each dictionary represents a data point and contains a key-value pair for the similarity score of that data point (i.e., similarity score).

The function then calculates the cosine similarity between the output sentence embeddings from the trained model using PyTorch's cosine similarity function, and returns the similarity score as a float. To scale the similarity score between 0 and 1, we can normalize it by dividing each similarity score by the maximum similarity score (which is 5 in this case), which may be useful for some types of analysis or modeling.

- We also define a Loss Function:



```
class CosineSimilarityLoss(torch.nn.Module):

    def __init__(self,  loss_fct = torch.nn.MSELoss(), cos_score_transformation=torch.nn.Identity()):

        super(CosineSimilarityLoss, self).__init__()
        self.loss_fct = loss_fct
        self.cos_score_transformation = cos_score_transformation
        self.cos = torch.nn.CosineSimilarity(dim=1)
```

Figure 24: Loss Function

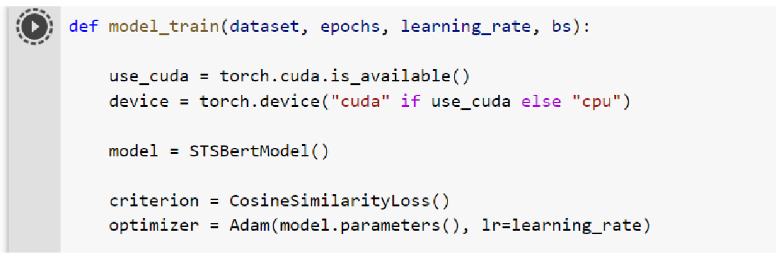
During training, the model's predicted similarity scores are compared to the ground truth similarity scores (which are typically obtained through human annotation) using a loss function. The loss function measures the discrepancy between the estimated and actual scores and provides a gradient that can be used to update the model's parameters via backpropagation.

The choice of the loss function is important because it determines how the model will be optimized and what kind of errors it is more likely to minimize. In an STS task, a common choice of the loss function is the mean squared error (MSE) loss or the cosine similarity loss, which penalizes the model for making large errors in predicting the similarity score.

By using a loss function during training, the model can learn to minimize its errors and generalize to new, unseen data. Without a loss function, the model would have no guidance on how to update its parameters and would likely perform poorly on new data. Therefore, using a suitable loss function is essential for training accurate and robust models for STS tasks.

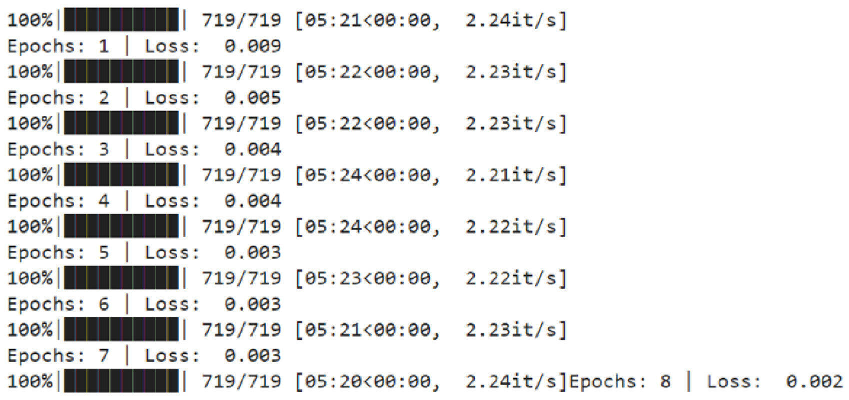
- Model Training:

With the architecture of the model, the data loader, and the loss function in place, it is now time to initiate the training process for the model.



```
def model_train(dataset, epochs, learning_rate, bs):

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    model = STSBertModel()

    criterion = CosineSimilarityLoss()
    optimizer = Adam(model.parameters(), lr=learning_rate)
```

Figure 25: Training script



```
100%|██████████| 719/719 [05:21<00:00,  2.24it/s]
Epochs: 1 | Loss:  0.009
100%|██████████| 719/719 [05:22<00:00,  2.23it/s]
Epochs: 2 | Loss:  0.005
100%|██████████| 719/719 [05:22<00:00,  2.23it/s]
Epochs: 3 | Loss:  0.004
100%|██████████| 719/719 [05:24<00:00,  2.21it/s]
Epochs: 4 | Loss:  0.004
100%|██████████| 719/719 [05:24<00:00,  2.22it/s]
Epochs: 5 | Loss:  0.003
100%|██████████| 719/719 [05:23<00:00,  2.22it/s]
Epochs: 6 | Loss:  0.003
100%|██████████| 719/719 [05:21<00:00,  2.23it/s]
Epochs: 7 | Loss:  0.003
100%|██████████| 719/719 [05:20<00:00,  2.24it/s]Epochs: 8 | Loss:  0.002
```

Figure 26: Training progress

Figure 27: Model training for 8 epochs

During training, the model is updated with batches of input data from the DataLoader object. Each batch is processed by the model to produce an output, which is then compared to the expected output using the cosine similarity loss function. The gradients are computed and the optimizer updates the model weights accordingly.

The function starts by checking if a CUDA-enabled GPU is available for training, and sets the device accordingly. It then initializes an STSBertModel and a CosineSimilarityLoss object, and an optimizer of type Adam to train the model.

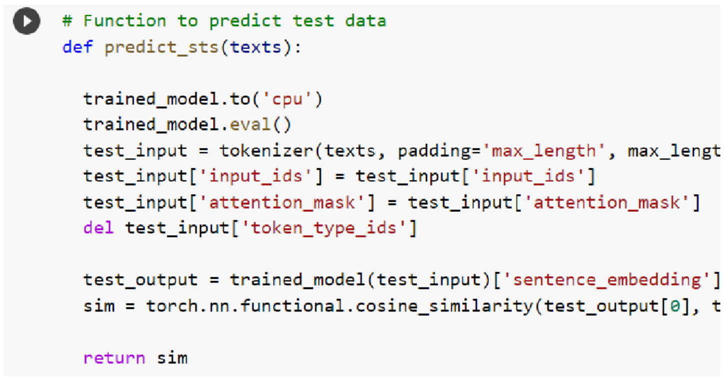
The input dataset is converted to a DataSequence object and a DataLoader object with a specified batch size, which will feed the data to the model during training. If a GPU is available, the model and the loss function are transferred to the device.

After each epoch, the function calculates the average loss and prints it to the console. The trained model is returned as output.

In the main program, the function is called with the specified hyperparameters (EPOCHS, LEARNING RATE, BATCH SIZE) and a dataset, and the resulting trained model is stored in the trained model variable.

- Model Prediction:

Once the model has been trained, it can be utilized to make predictions on data that it has not seen before. The function named predict sts that can be used to predict the similarity score between a pair of text inputs



```
# Function to predict test data
def predict_sts(texts):

  trained_model.to('cpu')
  trained_model.eval()
  test_input = tokenizer(texts, padding='max_length', max_lengt
  test_input['input_ids'] = test_input['input_ids']
  test_input['attention_mask'] = test_input['attention_mask']
  del test_input['token_type_ids']

  test_output = trained_model(test_input)['sentence_embedding']
  sim = torch.nn.functional.cosine_similarity(test_output[0], t

  return sim
```

Figure 28: Model Prediction

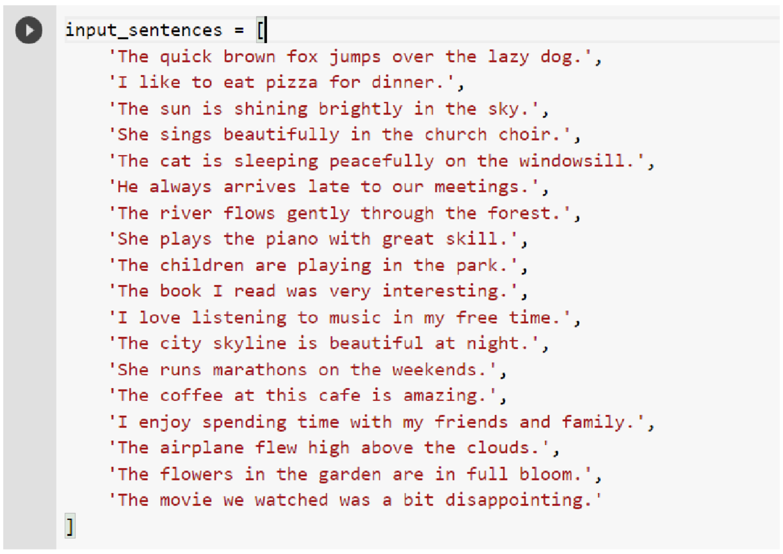
using a trained model.

The input to this function is a list of two text strings, and the output is a similarity score between these two strings computed using the cosine similarity function provided by the PyTorch library.

The function uses the trained model that has already been trained on a dataset of text pairs to perform the prediction task. It first pre-processes the input text pairs using the same tokenizer used during training, and then passes them through the trained model to obtain the sentence embeddings for each text.

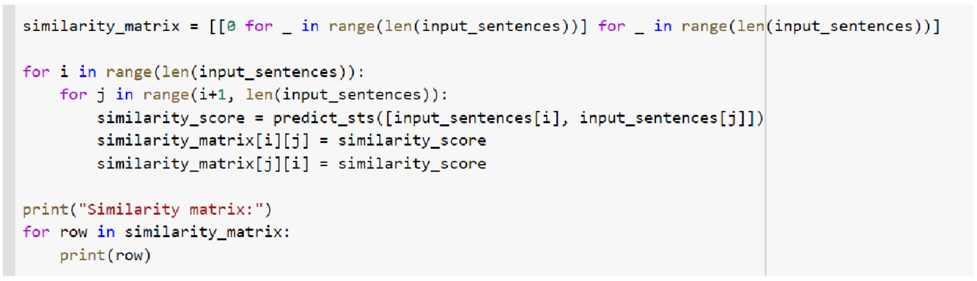
These embeddings are then used to compute the cosine similarity between the two sentence embeddings, which is returned as the final output from the function.

- Compute a similarity matrix for a list of input sentences using our Model Prediction.



```
input_sentences = [
    'The quick brown fox jumps over the lazy dog.',
    'I like to eat pizza for dinner.',
    'The sun is shining brightly in the sky.',
    'She sings beautifully in the church choir.',
    'The cat is sleeping peacefully on the windowsill.',
    'He always arrives late to our meetings.',
    'The river flows gently through the forest.',
    'She plays the piano with great skill.',
    'The children are playing in the park.',
    'The book I read was very interesting.',
    'I love listening to music in my free time.',
    'The city skyline is beautiful at night.',
    'She runs marathons on the weekends.',
    'The coffee at this cafe is amazing.',
    'I enjoy spending time with my friends and family.',
    'The airplane flew high above the clouds.',
    'The flowers in the garden are in full bloom.',
    'The movie we watched was a bit disappointing.'
]
```

Figure 29: List of input sentences



```
similarity_matrix = [[0 for _ in range(len(input_sentences))] for _ in range(len(input_sentences))]

for i in range(len(input_sentences)):
    for j in range(i+1, len(input_sentences)):
        similarity_score = predict_sts([input_sentences[i], input_sentences[j]])
        similarity_matrix[i][j] = similarity_score
        similarity_matrix[j][i] = similarity_score

print("Similarity matrix:")
for row in similarity_matrix:
    print(row)
```

Figure 30: Similarity matrix

Figure 31: Similarity matrix for a list of input sentences

We calculate a similarity matrix for a list of input sentences. The similarity matrix is a square matrix that shows the pairwise similarity scores between each pair of sentences in the input. then we stored the similarity score in the corresponding positions of the similarity matrix, and the matrix is printed at the end of the code.

```
Similarity matrix:
[0, 0.07025889307260513, 0.06101253256201744, 0.033337630331516266, 0.3684455454349518, 0.09503860771656036,
[0.07025889307260513, 0, 0.06570456922054291, 0.12823045253753662, 0.224593386054039, 0.20449170470237732, -
[0.06101253256201744, 0.06570456922054291, 0, 0.23878790438175201, 0.2654830813407898, 0.1217852234840393, 6
[0.033337630331516266, 0.12823045253753662, 0.23878790438175201, 0, 0.07604498416185379, 0.07041212916374207
[0.3684455454349518, 0.224593386054039, 0.2654830813407898, 0.07604498416185379, 0, 0.14825411140918732, -0.
[0.09503860771656036, 0.20449170470237732, 0.1217852234840393, 0.07041212916374207, 0.14825411140918732, 0,
[-0.013806970790028572, -0.036927685141563416, 0.05961782485246658, 0.15353769063949585, -0.0663362890481948
[0.07743165642023087, 0.09114041179418564, 0.16399772465229034, 0.40763503313064575, 0.16067177057266235, 0.
[0.051970306783914566, -0.05681580677628517, 0.18176448345184326, 0.14497420191764832, 0.09876997768878937,
[-0.010429946705698967, 0.18153679370880127, 0.19895398616790771, 0.13530415296554565, 0.132354736328125, 0.
[0.09173563122749329, 0.16771315038204193, 0.20934639871120453, 0.18761686980724335, 0.13008028268814087, 0.
[0.08582543581724167, 0.15398447215557098, 0.53094881772995, 0.14225931465625763, 0.2522158920764923, 0.1794
[0.000871581956744194, 0.0773485004901886, 0.08888892829418182, 0.12021949887275696, 0.047218207269907, 0.09
[-0.07540380209684372, 0.3348945379257202, 0.07005488872528076, 0.07568389177322388, 0.07680727541446686, 0.
[0.015655633062124252, 0.1914234757423401, 0.25659358501434326, 0.14036895334720612, 0.08287211507558823, 0.
[0.015310859307646751, -0.006279872730374336, 0.3101968467235565, 0.1725974977016449, -0.005197509191930294,
[-0.0780826210975647, -0.12745751440525055, 0.260928213596344, 0.1853557825088501, 0.03777708858251572, 0.01
[0.01863013580441475, 0.21735334396362305, 0.129944309592247, 0.08492538332939148, 0.06496191024780273, 0.19
```

Figure 32: Similarity matrix

- Generates a heatmap to visualize the similarity matrix of a list of input sentences:

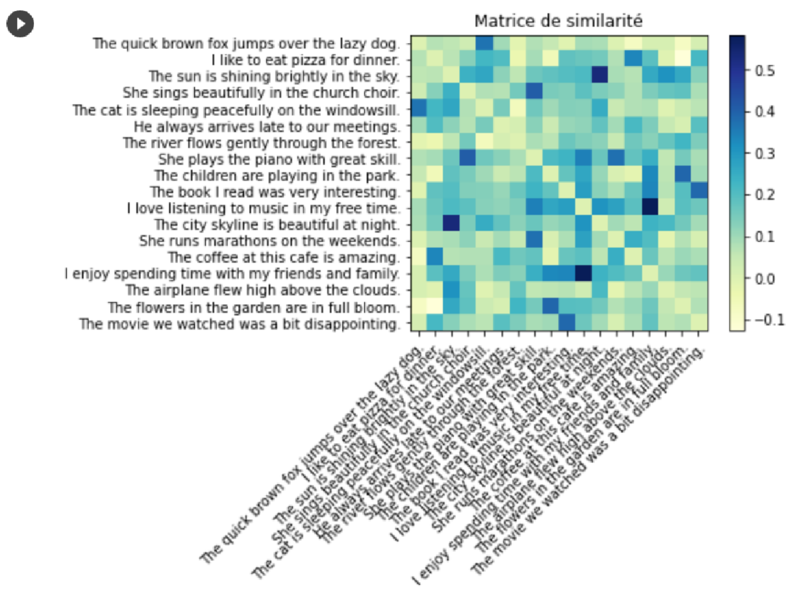


Figure 33: Heatmap to visualize the similarity matrix

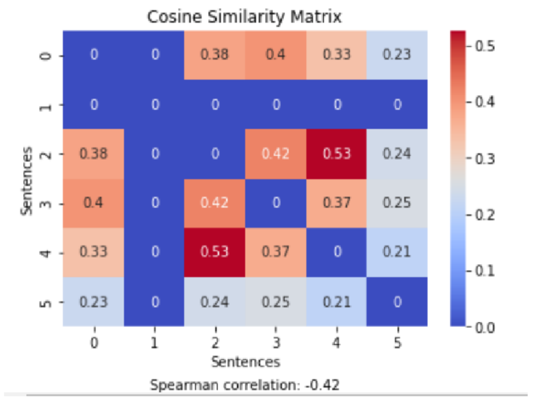
Here is a heatmap to visualize the similarity matrix of a list of input sentences. The darker colors represent a higher similarity score between two sentences, while the lighter colors indicate lower similarity scores.

The x-axis and y-axis labels show the input sentences, and the color bar on the right side of the heatmap indicates the similarity score scale.

This visualization can be useful in several natural language processing tasks such as text classification, clustering, and information retrieval, where measuring the similarity between sentences or documents is essential. It can help in identifying patterns, trends, and relationships between different texts. For example, it can be used to cluster similar documents together or to identify duplicate content.

5.4.4 Results task1

we can evaluate how well the cosine similarity method is performing in capturing the expected similarities between the input sentences. By comparing the similarity scores generated by the cosine similarity matrix to the gold standard(The gold standard is a matrix of expected similarity scores for each pair of sentences in the input sentences list) using the Spearman correlation coefficient.



	0	1	2	3	4	5
0	0	0	0.38	0.4	0.33	0.23
1	0	0	0	0	0	0
2	0.38	0	0	0.42	0.53	0.24
3	0.4	0	0.42	0	0.37	0.25
4	0.33	0	0.53	0.37	0	0.21
5	0.23	0	0.24	0.25	0.21	0

Figure 34: The Spearman correlation coefficient

The result of the Spearman correlation coefficient is -0.4212541096870377, which suggests a moderate negative correlation between the cosine similarity scores and the gold standard values. This means that the cosine similarity scores do not perfectly match the gold standard values, but there is some level of agreement between the two. The heatmap plot generated by the code visualizes the similarity matrix, and shows that the cosine similarity scores are generally higher for sentences that are more similar according to the gold standard values. When computing BERT-based models like bert-base-uncased on Google Colab,we face several difficulties due to limited resources, timeouts, connection and compatibility issues, and data preprocessing errors.These issues can affect the accuracy and performance of BERT-based models and require significant resources and time to train and evaluate properly.

Note this is a simple illustration of how to compute the similarity between textual documents using cosine similarity and how to evaluate the similarity scores using a gold standard

The results obtained from the code provided in the previous section can be used in various natural language processing (NLP) applications such as text classification, information retrieval, clustering, duplicate content detection, and chatbots/virtual assistants.

In text classification, the similarity matrix can be used to identify and classify similar texts into specific categories such as news articles, social media posts, or customer reviews.

In information retrieval, the similarity matrix can be used to rank documents based on their similarity to a query text, helping to retrieve the most relevant documents.

In clustering, the similarity matrix can help in grouping similar documents or sentences into clusters, which can be useful in various applications such as document summarization, sentiment analysis, or topic modeling.

Finally, in chatbots and virtual assistants, the similarity matrix can be used to measure the similarity between user queries and the existing knowledge base or FAQ, which can help in providing accurate and relevant responses to users.

5.5 Task 2

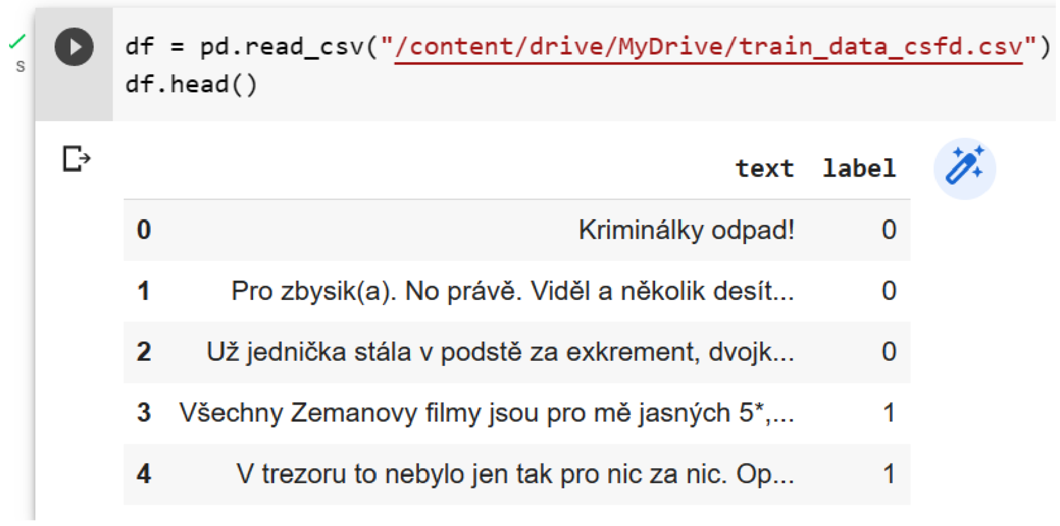
The objective is to evaluate sentiment analysis in Czech text using the bert-base-multilingual-cased model. This is done by producing sentence embeddings and refining them with the CSFD dataset of Czech film reviews. The enhanced model is then used to determine the similarity between Czech sentences, which is a crucial component of sentiment analysis. By comparing the similarity between the new review and previously labeled instances in the CSFD dataset, the model can predict the sentiment of the new review as positive, negative.

5.5.1 Dataset for task 2

The dataset selected for this task is the CSFD dataset, also known as the Czech-Slovak Film Database, which is a dataset containing film reviews and metadata about movies, primarily in the Czech and Slovak languages. The dataset is commonly used for natural language processing (NLP) and sentiment analysis research.

The CSFD dataset [Habernal and Brychcín, 2013] consists (90k reviews) of two main parts: the movie metadata and the user reviews. The movie metadata includes information such as the movie title, year of release, director, and genre. The user reviews include the text of the review, a rating of the movie (from 1 to 10), and the username of the reviewer.

The CSFD dataset is publicly available and can be downloaded from various sources, such as Kaggle and the UCI Machine Learning Repository. It contains tens of thousands of reviews and metadata for movies, making it a useful resource for researchers and data scientists interested in NLP and sentiment analysis.



```
df = pd.read_csv("/content/drive/MyDrive/train_data_csfd.csv")
df.head()
```

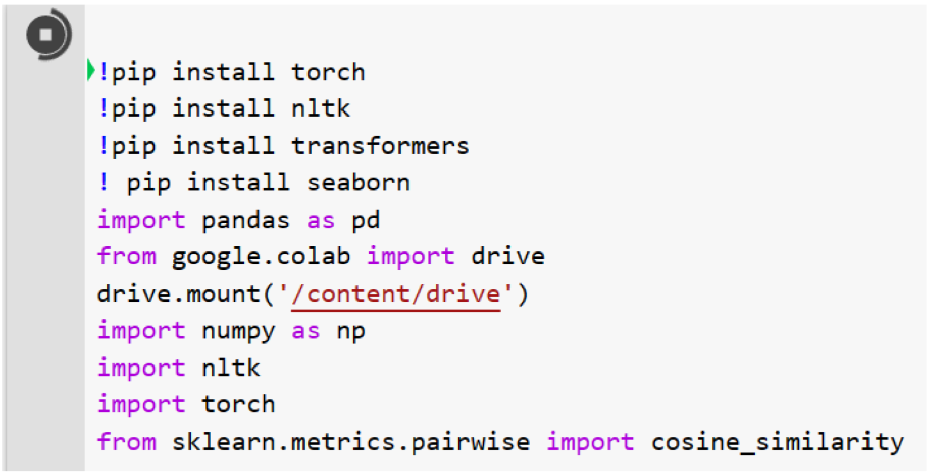
	text	label
0	Kriminálky odpad!	0
1	Pro zbysik(a). No právě. Viděl a několik desít...	0
2	Už jednička stála v podstě za exkrement, dvojk...	0
3	Všechny Zemanovy filmy jsou pro mě jasných 5*,...	1
4	V trezoru to nebylo jen tak pro nic za nic. Op...	1

Figure 35: The CSFD dataset

5.5.2 Implementation of bert-base-multilingual-cased

bert-base-multilingual-cased is a pre-trained BERT model that was trained on a multilingual corpus, which includes Czech language data. Therefore, it can be used for Czech language tasks along with many other languages.

- Install necessary packages - torch, nltk, transformers, and seaborn



```
!pip install torch
!pip install nltk
!pip install transformers
! pip install seaborn
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
import numpy as np
import nltk
import torch
from sklearn.metrics.pairwise import cosine_similarity
```

Figure 36: Torch, nltk, transformers, and seaborn

!pip install torch: This command uses the pip package manager to install the PyTorch library, which is a popular open-source machine learning framework used for building and training neural networks.

!pip install nltk: This command installs the Natural Language Toolkit (NLTK) library, which is a collection of tools and resources for natural language processing in Python.

!pip install transformers: This command installs the Transformers library, which is a powerful open-source library developed by Hugging Face for training and using state-of-the-art language models like BERT and GPT-2.

!pip install seaborn: This command installs the Seaborn library, which is a Python data visualization library based on Matplotlib. import pandas as pd: This line imports the Pandas library, which is a popular library used for data manipulation and analysis.

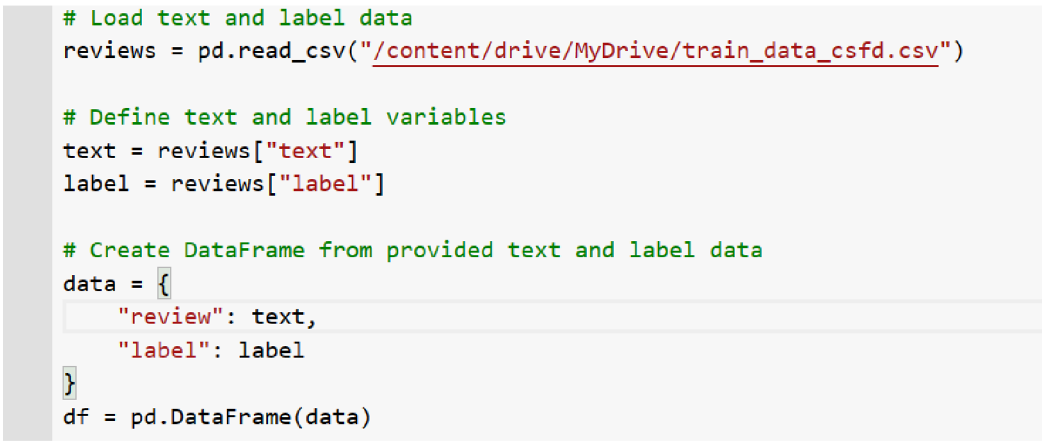
from google.colab import drive: This line imports the Google Colaboratory library, which provides access to Google Drive from within a Colab notebook.

drive.mount('/content/drive'): This line mounts my Google Drive so that we can access files and directories stored in our Drive account from within the Colab notebook.

import numpy as np: This line imports the NumPy library, which is a powerful library for working with arrays and matrices in Python.

import nltk: This line imports the NLTK library that we installed earlier, so that we can use its tools and resources for natural language processing.

- Load the reviews dataset from Google Drive as a pandas DataFrame
- Define text and label variables using the reviews DataFrame
- Create a new DataFrame by combining text and label data



```
# Load text and label data
reviews = pd.read_csv("/content/drive/MyDrive/train_data_csfd.csv")

# Define text and label variables
text = reviews["text"]
label = reviews["label"]

# Create DataFrame from provided text and label data
data = {
    "review": text,
    "label": label
}
df = pd.DataFrame(data)
```

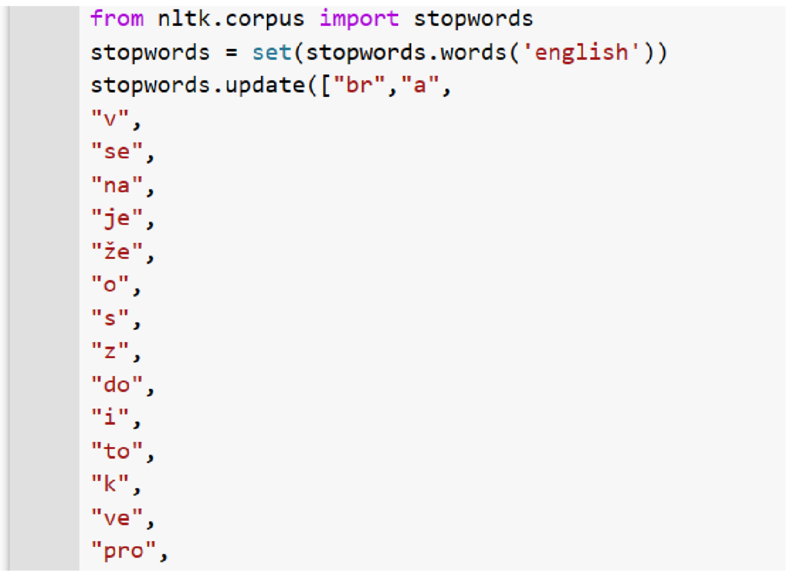
Figure 37: Reviews dataset from Google Drive as a pandas DataFrame

We read a CSV file called "train data csfd.csv" from the Google Drive folder "/content/drive/MyDrive", using the pandas library, and assigns it to a pandas DataFrame object called reviews. The CSV file contain two columns: "text" and "label".

Next,we create two variables called text and label, which correspond to the "text" and "label" columns in the reviews DataFrame, respectively.

Finally,we create a new DataFrame object called df from the text and label variables. This new DataFrame has two columns: "review" (which contains the text data from the "text" column in reviews) and "label" (which contains the label data from the "label" column in reviews). In other words, df is a reformatted version of the original reviews DataFrame.

- Define a list of stopwords to be removed from the text



```python
from nltk.corpus import stopwords
stopwords = set(stopwords.words('english'))
stopwords.update(["br","a",
"v",
"se",
"na",
"je",
"že",
"o",
"s",
"z",
"do",
"i",
"to",
"k",
"ve",
"pro",
```

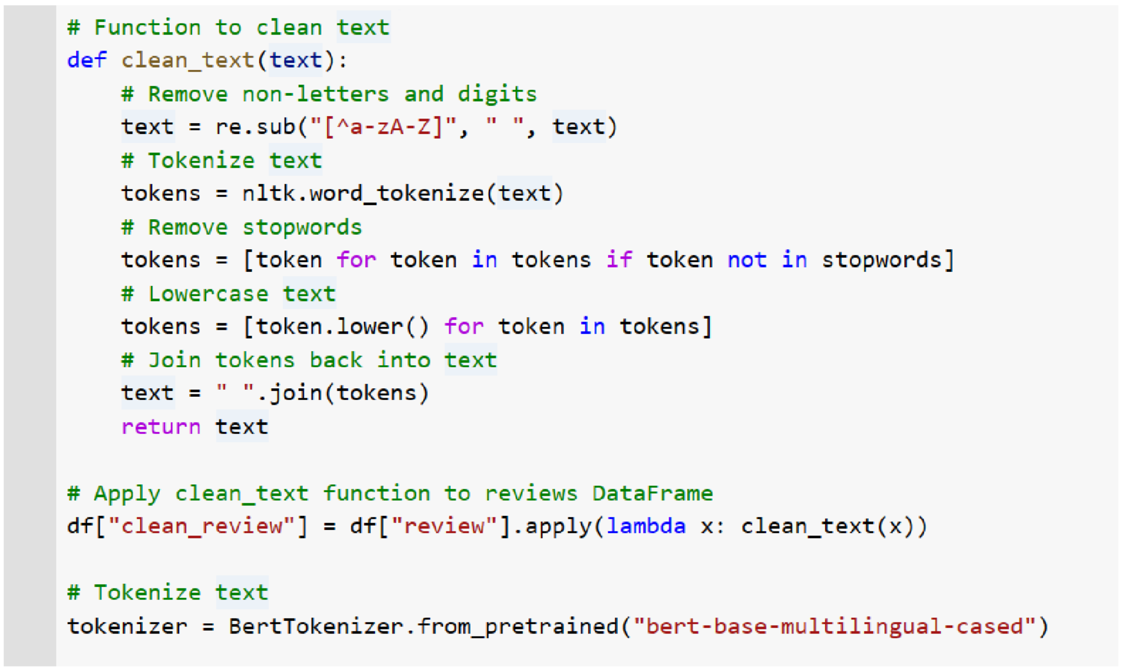
Figure 38: List of stopwords to be removed from the text

the set of stopwords is defined as a variable named "stopwords", which is initialized with the set of English stopwords imported from the "STOPWORDS" module. The code then adds additional Czech stopwords to this set using the "update()" method. These additional Czech stopwords include common words such as "a" (and), "se" (reflexive pronoun), "na" (on), "je" (is), "o" (about), "s" (with), "z" (from), "do" (to), "i" (also), "tak" (so), and many others.This will help to focus on the more important and less common words in the text, which can provide a better understanding of the content.

- Define a function to clean text data by removing non-letters, digits, and stopwords, and tokenize the text data

This code defines two functions that are used for cleaning and tokenizing text.

The first function, named "clean text", takes a text string as input and performs several text cleaning steps. It first removes any characters that are not letters or digits using regular expressions. It then tokenizes the text into separate words using the Natural Language Toolkit (nltk) "word tokenize" function. Next, it removes any stopwords from the text by comparing the tokens to a previously defined set of stopwords. It then converts all remaining tokens to lowercase and finally joins them back together into a single text string before returning the cleaned text.



```python
# Function to clean text
def clean_text(text):
    # Remove non-letters and digits
    text = re.sub("[^a-zA-Z]", " ", text)
    # Tokenize text
    tokens = nltk.word_tokenize(text)
    # Remove stopwords
    tokens = [token for token in tokens if token not in stopwords]
    # Lowercase text
    tokens = [token.lower() for token in tokens]
    # Join tokens back into text
    text = " ".join(tokens)
    return text

# Apply clean_text function to reviews DataFrame
df["clean_review"] = df["review"].apply(lambda x: clean_text(x))

# Tokenize text
tokenizer = BertTokenizer.from_pretrained("bert-base-multilingual-cased")
```

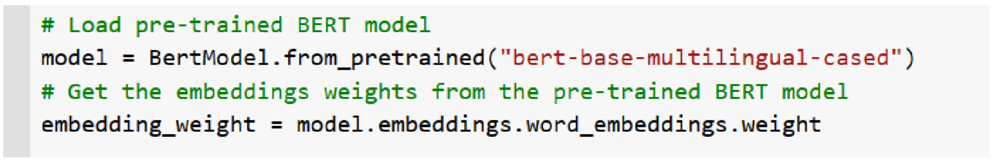
Figure 39: Function to clean text data by removing non-letters, digits, and stopwords

The second function, named "tokenize text", takes a text string as input and uses the BERT tokenizer from the "bert-base-multilingual-cased" pre-trained model to encode and tokenize the text. The function returns a dictionary with the encoded text and associated attention mask, which can be used as input to a BERT model.

After defining these functions, the code applies the "clean text" function to a Pandas DataFrame named "df" that contains a column of text reviews. The cleaned reviews are stored in a new column named "clean review" in the same DataFrame.

Finally, the "tokenize text" function is defined and can be used later on to tokenize any text that needs to be input to a BERT model.

- Load the pre-trained BERT model and get the embeddings weights from the model



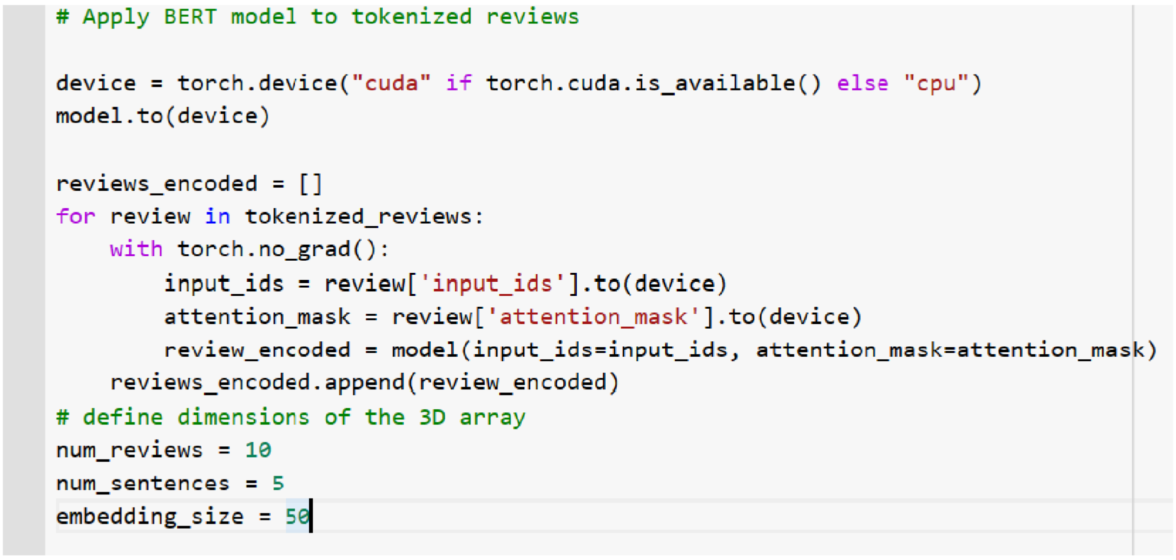
```python
# Load pre-trained BERT model
model = BertModel.from_pretrained("bert-base-multilingual-cased")
# Get the embeddings weights from the pre-trained BERT model
embedding_weight = model.embeddings.word_embeddings.weight
```

Figure 40: Load the pre-trained BERT model

We load a pre-trained BERT model named "bert-base-multilingual-cased" using the from pretrained method of the BertModel class. The model is loaded from the Hugging Face Transformers library After loading the pre-trained model, the code extracts the weights of the embedding layer by accessing the word embeddings attribute of the model's embeddings layer. The embedding layer is responsible for converting input text into a numerical representation that can be used as input to the model.

The embedding weight variable is assigned the tensor of embedding weights. These weights can be used in a variety of downstream NLP tasks that require word embeddings, such as sentiment analysis, text classification,

- Apply the BERT model to tokenized reviews to obtain the encoded reviews



```python
# Apply BERT model to tokenized reviews

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

reviews_encoded = []
for review in tokenized_reviews:
    with torch.no_grad():
        input_ids = review['input_ids'].to(device)
        attention_mask = review['attention_mask'].to(device)
        review_encoded = model(input_ids=input_ids, attention_mask=attention_mask)
    reviews_encoded.append(review_encoded)
# define dimensions of the 3D array
num_reviews = 10
num_sentences = 5
embedding_size = 50
```

Figure 41: Apply BERT model to tokenized reviews

In the first two lines of code, the program initializes a variable called device that selects the available device as a CUDA-enabled GPU (if available) or CPU otherwise. The model.to(device) line moves the model to the selected device.

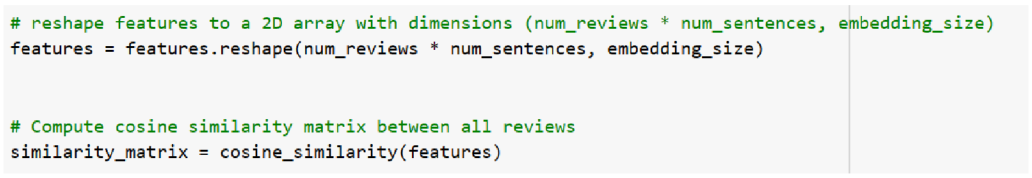
The program then creates an empty list called reviews encoded to store the encoded reviews.

The following code uses a for loop to process each review in tokenized reviews. Inside the loop, input ids and attention mask are extracted from the current review dictionary, converted to tensors, and then moved to the selected device using the to() method. The torch.no grad() context is used to disable the gradient calculation, which helps to reduce memory usage during inference.

Then, the BERT model is used to encode the review by passing the input ids and attention mask tensors to the model, and the encoded review tensor is assigned to review encoded. The review encoded tensor is then appended to the reviews encoded list.

Finally, the code defines three variables num reviews, num sentences, and embedding size. These variables describe the dimensions of a 3D array that can be used to store the encoded reviews. Specifically, num reviews indicates the number of reviews, num sentences indicates the maximum number of sentences per review, and embedding size indicates the size of the embedding vector produced by the BERT model.

- Compute cosine similarity matrix between all reviews

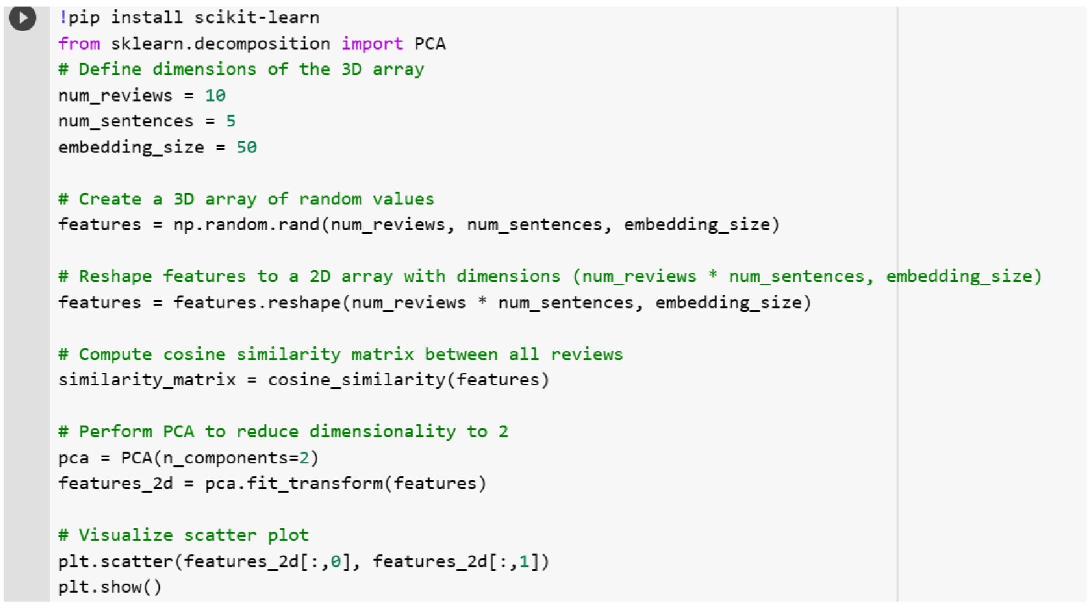


```
# reshape features to a 2D array with dimensions (num_reviews * num_sentences, embedding_size)
features = features.reshape(num_reviews * num_sentences, embedding_size)


# Compute cosine similarity matrix between all reviews
similarity_matrix = cosine_similarity(features)
```

Figure 42: Cosine similarity matrix

The cosine similarity() function from the scikit-learn library is used to compute the cosine similarity matrix between all pairs of sentences in all reviews using the features array as input. The resulting similarity matrix is a square matrix where each element (i,j) represents the cosine similarity between the ith and jth review.

- Visualize the similarity matrix



```
!pip install scikit-learn
from sklearn.decomposition import PCA
# Define dimensions of the 3D array
num_reviews = 10
num_sentences = 5
embedding_size = 50

# Create a 3D array of random values
features = np.random.rand(num_reviews, num_sentences, embedding_size)

# Reshape features to a 2D array with dimensions (num_reviews * num_sentences, embedding_size)
features = features.reshape(num_reviews * num_sentences, embedding_size)

# Compute cosine similarity matrix between all reviews
similarity_matrix = cosine_similarity(features)

# Perform PCA to reduce dimensionality to 2
pca = PCA(n_components=2)
features_2d = pca.fit_transform(features)

# Visualize scatter plot
plt.scatter(features_2d[:,0], features_2d[:,1])
plt.show()
```

Figure 43: Visualize scatter plot

We perform dimensionality reduction using principal component analysis (PCA) and visualizes the results as a scatter plot.

The dimensions of the 3D array are defined, where num reviews represents the number of reviews, num sentences represents the number of sentences per review, and embedding size represents the size of the embedding for each sentence.

A 3D array of random values is created using np.random.rand() function, with the shape (num reviews, num sentences, embedding size).

The features 3D array is reshaped into a 2D array of shape (num reviews * num sentences, embedding size) using the reshape() function. The cosine similarity matrix is computed between all reviews using the cosine similarity() function from scikit-learn.

PCA is performed on the features of the reviews to reduce the dimensionality to 2, using the PCA() function from scikit-learn.

The 2D features obtained from PCA are visualized as a scatter plot using the scatter() function from matplotlib.

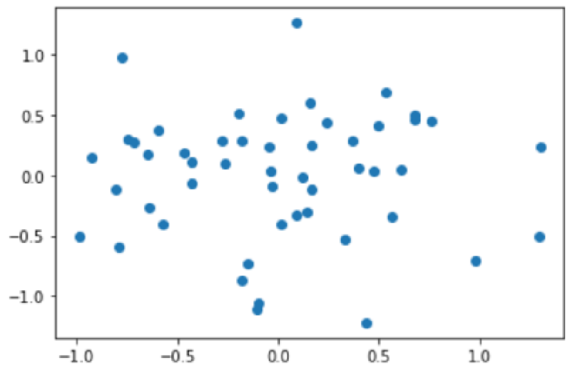


Figure 44: Scatter plot of the 2D features obtained from performing PCA on the random features of the reviews

The scatter plot shows a visualization of a high-dimensional data set (specifically, a set of embeddings) after applying Principal Component Analysis (PCA) to reduce the dimensionality to two.

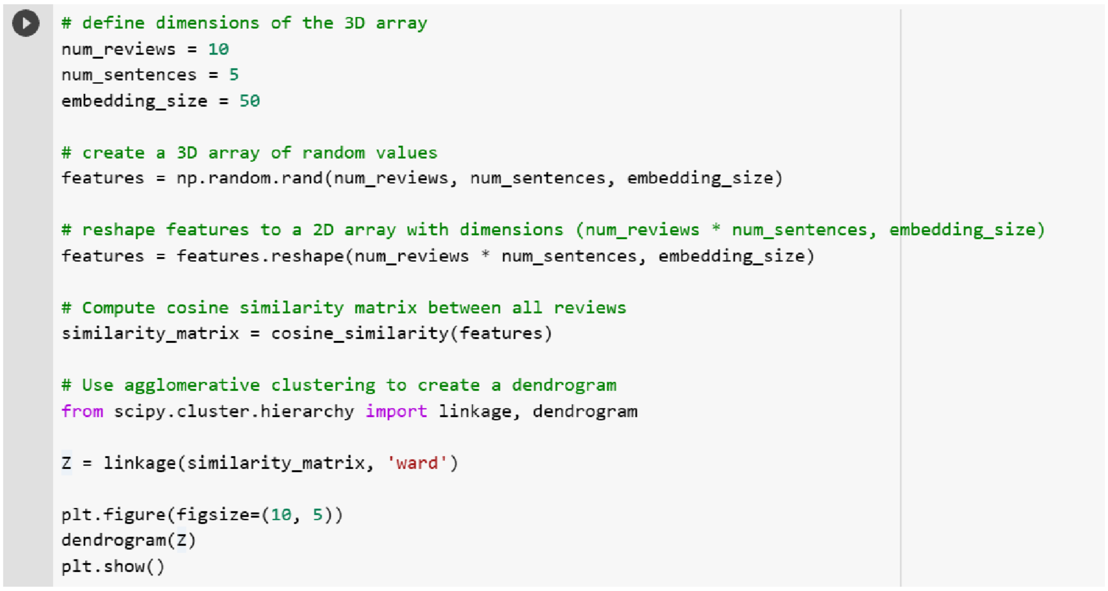
Each point in the plot represents an embedding (or a combination of multiple embeddings) for a given review and sentence. The X-axis and Y-axis of the plot represent the two principal components that capture the largest amount of variance in the original data set.

The plot allows us to visualize any patterns or clusters in the data set. If there are distinct groups or clusters of points, this could indicate that certain reviews or sentences share similar features or characteristics. On the other hand, if the points are scattered randomly throughout the plot, it may suggest that there are no clear patterns or relationships among the data.

Note that the scatter plot generated by this code is using cosine similarity as a measure of similarity between the embeddings. Therefore, points that are closer to each other in the plot are more similar to each other in terms of cosine similarity.

5.5.3 Sentiment analysis tasks involving new reviews

In the context of sentiment analysis, agglomerative clustering is a popular method for grouping similar reviews together based on their underlying semantic content. To achieve this, a dendrogram is generated using the cosine similarity matrix computed for all reviews, which allows for the identification of reviews that share similar sentiment. By clustering together reviews that exhibit similar sentiment, the approach facilitates the analysis of common themes or topics within the reviews, and provides insights into the aspects of the product or service that are being discussed. Therefore, agglomerative clustering is a valuable tool for sentiment analysis tasks involving new reviews.



```python
# define dimensions of the 3D array
num_reviews = 10
num_sentences = 5
embedding_size = 50

# create a 3D array of random values
features = np.random.rand(num_reviews, num_sentences, embedding_size)

# reshape features to a 2D array with dimensions (num_reviews * num_sentences, embedding_size)
features = features.reshape(num_reviews * num_sentences, embedding_size)

# Compute cosine similarity matrix between all reviews
similarity_matrix = cosine_similarity(features)

# Use agglomerative clustering to create a dendrogram
from scipy.cluster.hierarchy import linkage, dendrogram

Z = linkage(similarity_matrix, 'ward')

plt.figure(figsize=(10, 5))
dendrogram(Z)
plt.show()
```

Figure 45: Apply BERT model to tokenized reviews

We generates a dendrogram using agglomerative clustering to perform sentiment analysis on a collection of reviews.

First, the dimensions of a 3D array are defined, where num reviews denotes the number of reviews, num sentences denotes the number of sentences in each review, and embedding size denotes the size of the word embedding used in the analysis.

Next, a 3D array is created with random values using NumPy. This array represents the embedding vectors for each word in each sentence of each review.

The features array is then reshaped into a 2D array with dimensions (num reviews * num sentences, embedding size), which allows for the calculation of cosine similarity between all reviews.

The cosine similarity matrix is then computed using the cosine similarity function from Scikit-learn. This matrix represents the similarity between each pair of reviews in terms of their word embeddings.

The linkage function from SciPy is used to perform agglomerative clustering on the similarity matrix, and the resulting dendrogram is plotted using the dendrogram function. Finally, the dendrogram is displayed using plt.show().

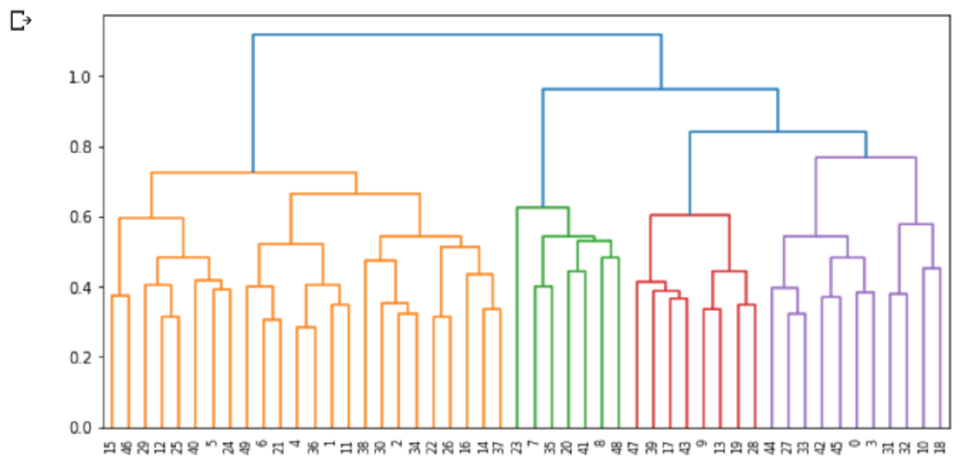


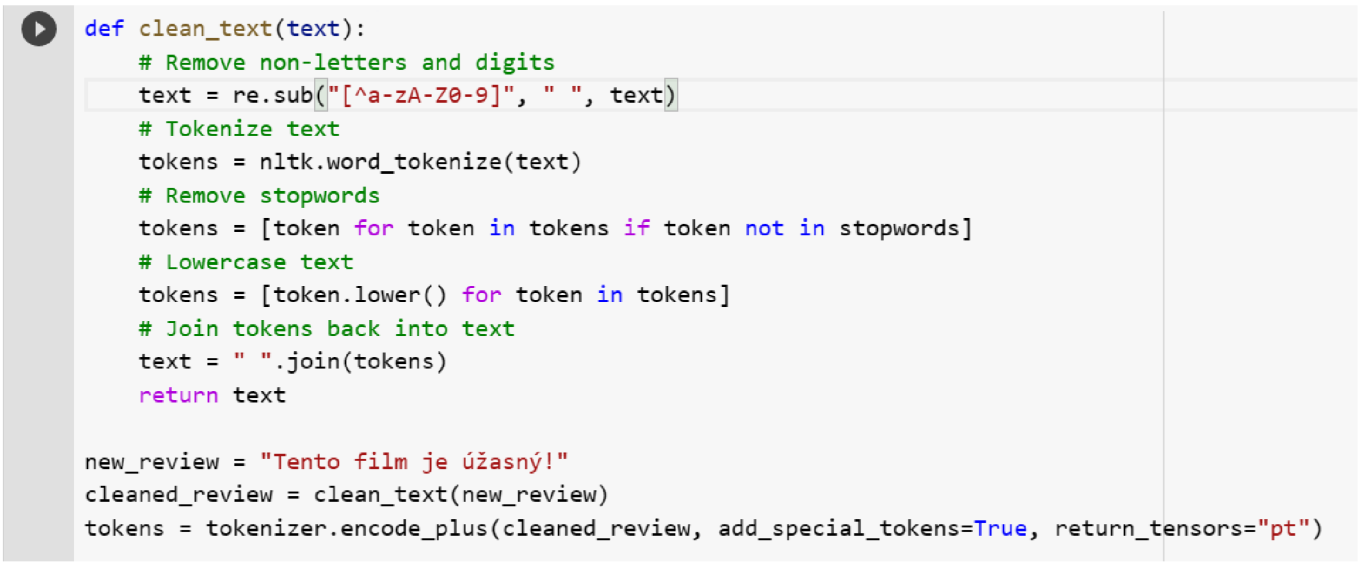
Figure 46: Structure of the dendrogram and the relationships between the reviews

The resulting dendrogram shown by the given code represents a hierarchical clustering of the reviews based on their cosine similarity. The dendrogram is a tree-like diagram that shows the arrangement of the clusters of the reviews based on their similarity.

Each leaf node in the dendrogram represents a single review, and the height of each node indicates the distance between the clusters being merged. The longer the branch, the less similar the clusters are.

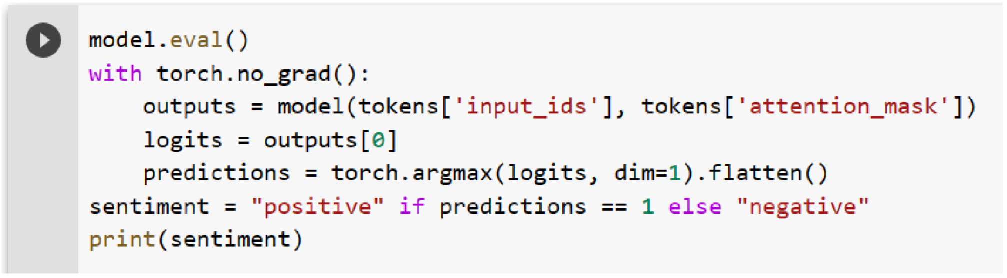
The dendrogram can be useful for visualizing the relationships between the reviews and identifying groups of reviews that are similar to each other. The clustering can also be used for tasks such as text classification, topic modeling, or recommendation systems. The resulting dendrogram may look different each time the code is run, as the random values generated for the 3D array features will be different each time

We define a function called clean text which takes a string of text as input and performs several text preprocessing steps. These include removing non-alphanumeric characters, tokenizing the text into words, removing stopwords, converting words to lowercase, and joining the cleaned tokens back into a single string. The second part of the code then takes the cleaned text and tokenizes it using the tokenizer.encode plus function from the Hugging Face Transformers library, producing a PyTorch tensor that can be used as input to a machine learning model.



```python
def clean_text(text):
    # Remove non-letters and digits
    text = re.sub("[^a-zA-Z0-9]", " ", text)
    # Tokenize text
    tokens = nltk.word_tokenize(text)
    # Remove stopwords
    tokens = [token for token in tokens if token not in stopwords]
    # Lowercase text
    tokens = [token.lower() for token in tokens]
    # Join tokens back into text
    text = " ".join(tokens)
    return text

new_review = "Tento film je úžasný!"
cleaned_review = clean_text(new_review)
tokens = tokenizer.encode_plus(cleaned_review, add_special_tokens=True, return_tensors="pt")
```

Figure 47: Preprocessing and encoding of new review



```python
model.eval()
with torch.no_grad():
    outputs = model(tokens['input_ids'], tokens['attention_mask'])
    logits = outputs[0]
    predictions = torch.argmax(logits, dim=1).flatten()
sentiment = "positive" if predictions == 1 else "negative"
print(sentiment)
```

Figure 48: Sentiment analysis on some input text using a pre-trained BERT model

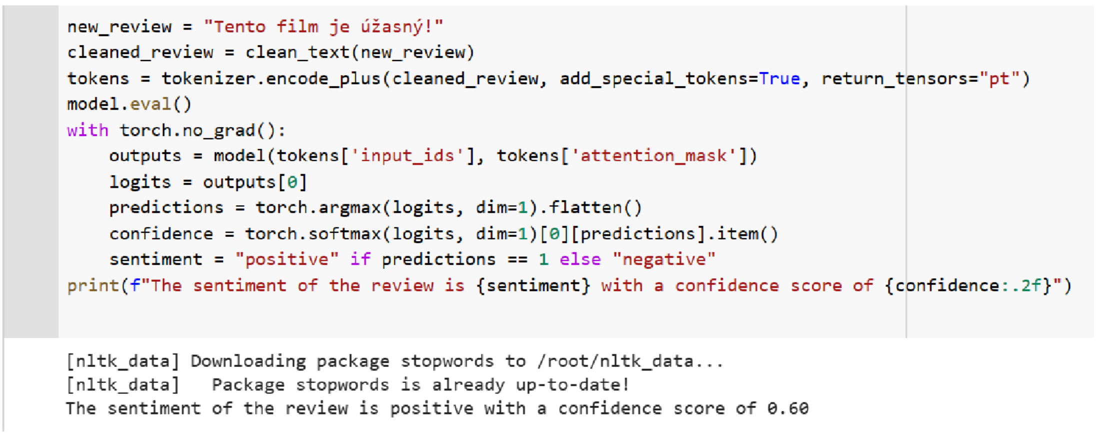
We perform sentiment analysis on some input text using a pre-trained BERT model. predictions = torch.argmax(logits, dim=1).flatten(): Calculates the predicted class for each input sample by finding the index with the highest value in the logits tensor. The flatten() function converts the predictions from a 2D tensor to a 1D tensor.

sentiment = "positive" if predictions == 1 else "negative": Assigns "positive" to the sentiment variable if the predicted class is 1, otherwise "negative".

print(sentiment): Prints the predicted sentiment label for the input text.

5.5.4 Results task 2

When computing BERT-based models like bert-base-multilingual-cased on Google Colab,we face several difficulties due to limited resources, timeouts, connection and compatibility issues, and data preprocessing errors.These issues can affect the accuracy and performance of BERT-based models and require significant resources and time to train and evaluate properly.

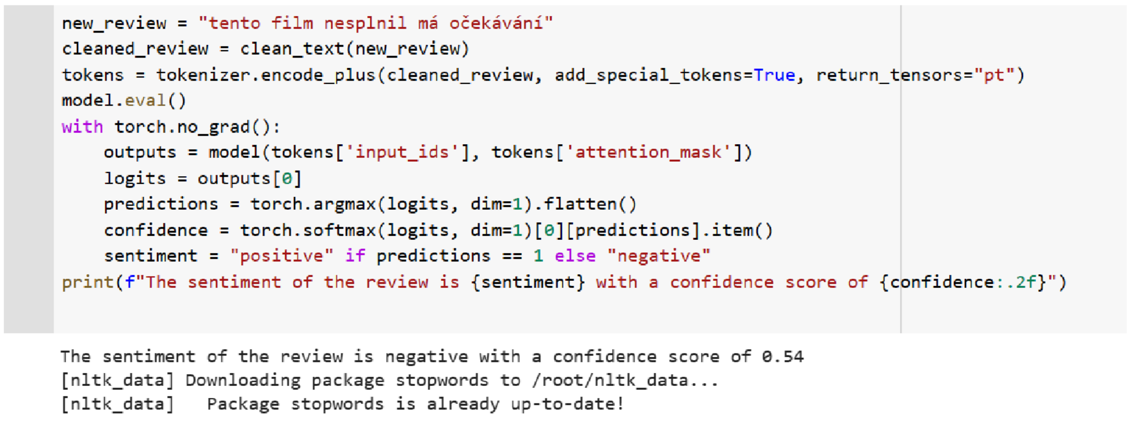


```
new_review = "Tento film je úžasný!"
cleaned_review = clean_text(new_review)
tokens = tokenizer.encode_plus(cleaned_review, add_special_tokens=True, return_tensors="pt")
model.eval()
with torch.no_grad():
    outputs = model(tokens['input_ids'], tokens['attention_mask'])
    logits = outputs[0]
    predictions = torch.argmax(logits, dim=1).flatten()
    confidence = torch.softmax(logits, dim=1)[0][predictions].item()
    sentiment = "positive" if predictions == 1 else "negative"
print(f"The sentiment of the review is {sentiment} with a confidence score of {confidence:.2f}")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
The sentiment of the review is positive with a confidence score of 0.60
```

Figure 49: Sentiment analysis:Positive new review

This result suggests that the sentiment analysis model predicted a positive sentiment for the given movie review, with a confidence score of 0.60. The confidence score is a measure of how certain the model is about its prediction. In this case, a confidence score of 0.60 means that the model is somewhat confident in its prediction, but not completely certain.



```
new_review = "tento film nesplnil má očekávání"
cleaned_review = clean_text(new_review)
tokens = tokenizer.encode_plus(cleaned_review, add_special_tokens=True, return_tensors="pt")
model.eval()
with torch.no_grad():
    outputs = model(tokens['input_ids'], tokens['attention_mask'])
    logits = outputs[0]
    predictions = torch.argmax(logits, dim=1).flatten()
    confidence = torch.softmax(logits, dim=1)[0][predictions].item()
    sentiment = "positive" if predictions == 1 else "negative"
print(f"The sentiment of the review is {sentiment} with a confidence score of {confidence:.2f}")
```

```
The sentiment of the review is negative with a confidence score of 0.54
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Figure 50: Sentiment analysis:Negative new review

This result suggests that the sentiment analysis model predicted a negative sentiment for the given review, with a confidence score of 0.54. In this case, a confidence score of 0.54 means that the model is somewhat confident in its prediction, but not completely certain.

Semantic similarity is used to identify reviews with similar meanings by computing the cosine similarity measure between different reviews. The metric ranges from 0 to 1, with 0 indicating no similarity and 1 indicating perfect similarity. Similar reviews can be clustered together to analyze the overall sentiment of the dataset, identify common themes, and gain insights into patterns. Positive and negative reviews may use similar language and expressions to convey their sentiment. Furthermore, the applications of semantic similarity techniques go beyond sentiment analysis in reviews. These techniques can be used in various fields such as information retrieval, recommendation systems, and natural language generation. The results of this research highlight the potential benefits of incorporating semantic similarity techniques into these applications.

6 Conclusion

In conclusion, this thesis has explored the effectiveness of semantic similarity measurement techniques in extracting entities from texts written in natural language, and their applications in different fields.

The uses of semantic similarity methods go beyond sentiment analysis in reviews. The findings demonstrated that semantic similarity can be useful in identifying reviews that convey similar meanings, and thus can be used to analyze the overall sentiment of a dataset. By computing the cosine similarity between reviews and clustering them based on their similarity matrix, common themes, and patterns can be identified, providing valuable insights.

However, the limitations of memory and resources that prevented us from testing the accuracy of our models have also been acknowledged.

From an application point of view, this contribution proposes a general methodology guideline that can be applied in various areas and domains such as information retrieval, and natural language generation, recommendation systems, for example, building a chatbot that can recommend movies based on the similarity of their reviews to the user's preferred movies.

Suggesting that there is still room for improvement and further experimentation.

7 References

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, Cambridge, MA, USA, November 2016. ISBN 978-0-262-03561-3.

J.M. Duarte, S. Sousa, E. Milios, and L. Berton. Deep analysis of word sense disambiguation via semi-supervised learning and neural word representations. *Information Sciences*, 570:278–297, 2021. ISSN 0020-0255. doi:10.1016/j.ins.2021.04.006.

R. Xiang, E. Chersoni, Q. Lu, C.-R. Huang, W. Li, and Y. Long. Lexical data augmentation for sentiment analysis. *Journal of the Association for Information Science and Technology*, 72(11):1432–1447, 2021. ISSN 2330-1635. doi:10.1002/asi.24493.

F. Wu, M. Du, C. Fan, R. Tang, Y. Yang, A. Mostafavi, and X. Hu. Understanding Social Biases Behind Location Names in Contextual Word Embedding Models. *IEEE Transactions on Computational Social Systems*, 9(2):458–468, 2022. doi:10.1109/TCSS.2021.3106003.

D. Chandrasekaran and V. Mago. Comparative Analysis of Word Embeddings in Assessing Semantic Similarity of Complex Sentences. *IEEE Access*, 9:166395–166408, 2021. doi:10.1109/ACCESS.2021.3135807.

S. Jiang, W. Wu, N. Tomita, C. Ganoe, and S. Hassanpour. Multi-Ontology Refined Embeddings (MORE): A hybrid multi-ontology and corpus-based semantic representation model for biomedical concepts. *Journal of Biomedical Informatics*, 111, 2020. doi:10.1016/j.jbi.2020.103581.

A. Jaber and P. Martínez. Disambiguating Clinical Abbreviations Using a One-Fits-All Classifier Based on Deep Learning Techniques. *Methods of Information in Medicine*, 2022. doi:10.1055/s-0042-1742388.

M. Gao, J. Lu, and F. Chen. Medical Knowledge Graph Completion Based on Word Embeddings. *Information (Switzerland)*, 13(4), 2022. doi:10.3390/info13040205.

B.S. Neves Oliveira, A. Fernandes De Oliveira, V. Monteiro De Lira, T. Linhares Coelho Da Silva, and J.A. Fernandes De MacÊdo. HELD: Hierarchical entity-label disambiguation in named entity recognition task using deep learning. *Intelligent Data Analysis*, 26(3):637–657, 2022. ISSN 1088-467X. doi:10.3233/IDA-205720.

V. Balakrishnan, Z. Shi, C.L. Law, R. Lim, L.L. Teh, and Y. Fan. A deep learning approach in predicting products' sentiment ratings: a comparative analysis. *Journal of Supercomputing*, 78(5):7206–7226, 2022. doi:10.1007/s11227-021-04169-6.

Howard Jackson. *Grammar and Meaning: A Semantic Approach to English Grammar*. Longman, 1990. ISBN 978-0-582-02875-3. Google-Books-ID: cr54AAAAIAAJ.

Alejandro Martín, Javier Huertas-Tato, Álvaro Huertas-García, Guillermo Villar-Rodríguez, and David Camacho. FacTeR-Check: Semi-automated fact-checking through semantic similarity and natural language inference. *Knowledge-Based Systems*, 251:109265, September 2022. ISSN 0950-7051. doi:10.1016/j.knosys.2022.109265. URL https://www.sciencedirect.com/science/article/pii/S0950705122006323.

Rada Mihalcea. Corpus-based and Knowledge-based Measures of Text Semantic Similarity. 2006.

Wael H.Gomaa and Aly A. Fahmy. A Survey of Text Similarity Approaches. *International Journal of Computer Applications*, 68(13):13–18, April 2013. ISSN 09758887. doi:10.5120/11638-7118. URL http://research.ijcaonline.org/volume68/number13/pxc3887118.pdf.

Yuan Guo, Yinghong Peng, and Jie Hu. Research on high creative application of case-based reasoning system on engineering design. *Computers in Industry*, 64(1):90–103, January 2013. ISSN 0166-3615. doi:10.1016/j.compind.2012.10.006. URL https://www.sciencedirect.com/science/article/pii/S0166361512001765.

Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 Task 1: Semantic Textual Similarity - Multilingual and Cross-lingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, 2017. doi:10.18653/v1/S17-2001. URL http://arxiv.org/abs/1708.00055. arXiv:1708.00055 [cs].

Marco Marelli, Stefano Menini, Marco Baroni, Luisa Bentivogli, Raffaella Bernardi, and Roberto Zamparelli. A SICK cure for the evaluation of compositional distributional semantic models. In *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, pages 216–223, Reykjavik, Iceland, May 2014. European Language Resources Association (ELRA). URL http://www.lrec-conf.org/proceedings/lrec2014/pdf/363_Paper.pdf.

Zhibiao Wu and Martha Palmer. Verb Semantics and Lexical Selection, June 1994. URL http://arxiv.org/abs/cmp-lg/9406033. arXiv:cmp-lg/9406033.

Claudia Leacock, Martin Chodorow, and George A. Miller. Using Corpus Statistics and WordNet Relations for Sense Identification. *Computational Linguistics*, 24(1):147–165, 1998. URL https://aclanthology.org/J98-1006. Place: Cambridge, MA Publisher: MIT Press.

Scott Deerwester, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American Society for*

Information Science, 41(6):391–407, 1990. ISSN 1097-4571. doi:10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-4571%28199009%2941%3A6%3C391%3A%3AAID-ASI1%3E3.0.CO%3B2-9.

Siddharth Patwardhan, Satanjeev Banerjee, and Ted Pedersen. Using Measures of Semantic Relatedness for Word Sense Disambiguation. In Alexander Gelbukh, editor, *Computational Linguistics and Intelligent Text Processing*, Lecture Notes in Computer Science, pages 241–257, Berlin, Heidelberg, 2003. Springer. ISBN 978-3-540-36456-6. doi:10.1007/3-540-36456-0_24.

Gilles Bisson and Fawad Hussain. Chi-Sim: A New Similarity Measure for the Co-clustering Task. In *2008 Seventh International Conference on Machine Learning and Applications*, pages 211–217, San Diego, CA, USA, 2008. IEEE. ISBN 978-0-7695-3495-4. doi:10.1109/ICMLA.2008.103. URL http://ieeexplore.ieee.org/document/4724977/.

Christiane Fellbaum. A Semantic Network of English: The Mother of All WordNets. In Piek Vossen, editor, *EuroWordNet: A multilingual database with lexical semantic networks*, pages 137–148. Springer Netherlands, Dordrecht, 1998. ISBN 978-94-017-1491-4. doi:10.1007/978-94-017-1491-4_6. URL https://doi.org/10.1007/978-94-017-1491-4_6.

Philip Resnik. Using Information Content to Evaluate Semantic Similarity in a Taxonomy, November 1995. URL http://arxiv.org/abs/cmp-lg/9511007. arXiv:cmp-lg/9511007.

Dingquan Li. An Information-Theoretic Definition of Similarity. 1998.

Jay J. Jiang and David W. Conrath. Semantic Similarity Based on Corpus Statistics and Lexical Taxonomy, September 1997. URL http://arxiv.org/abs/cmp-lg/9709008. arXiv:cmp-lg/9709008.

Evgeniy Gabrilovich. Computing Semantic Relatedness Using Wikipedia-based Explicit Semantic Analysis. 2007.

Rudi Cilibrasi and Paul M. B. Vitanyi. The Google Similarity Distance, May 2007. URL http://arxiv.org/abs/cs/0412098. arXiv:cs/0412098.

Thomas Hofmann. Probabilistic latent semantic indexing. In *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 50–57, Berkeley California USA, August 1999. ACM. ISBN 978-1-58113-096-6. doi:10.1145/312624.312649. URL https://dl.acm.org/doi/10.1145/312624.312649.

David M Blei. Latent Dirichlet Allocation. 2003.

Edward Grefenstette. Towards a Formal Distributional Semantics: Simulating Logical Calculi with Tensors. 2009.

Alexander Budanitsky and Graeme Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1):13–47, 2006. doi:10.1162/coli.2006.32.1.13. URL https://aclanthology.org/J06-1003.

Michael Mohler and Rada Mihalcea. Text-to-Text Semantic Similarity for Automatic Short Answer Grading. In *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*, pages 567–575, Athens, Greece, March 2009. Association for Computational Linguistics. URL https://aclanthology.org/E09-1065.

Elias Iosif and Alexandros Potamianos. Similarity computation using semantic networks created from web-harvested data. *Natural Language Engineering*, 21(1):49–79, January 2015. ISSN 1351-3249, 1469-8110. doi:10.1017/S1351324913000144. URL https://www.cambridge.org/core/journals/natural-language-engineering/article/abs/similarity-computation-using-semantic-networks-created-from-webharvested-data/62E3431D60D1296CAF2907B6C8689E6B. Publisher: Cambridge University Press.

Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Procedings of the British Machine Vision Conference 2016*, pages 119.1–119.11, York, UK, 2016. British Machine Vision Association. ISBN 978-1-901725-59-9. doi:10.5244/C.30.119. URL http://www.bmva.org/bmvc/2016/papers/paper119/index.html.

Song Han, Huizi Mao, and William J. Dally. Deep Compression: Compressing Deep Neural Networks with Pruning, Trained Quantization and Huffman Coding, February 2016. URL http://arxiv.org/abs/1510.00149. arXiv:1510.00149 [cs].

Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, September 2017. ISSN 1476-4687. doi:10.1038/nature23474. URL https://www.nature.com/articles/nature23474. Number: 7671 Publisher: Nature Publishing Group.

Pramila P. Shinde and Seema Shah. A Review of Machine Learning and Deep Learning Applications. In *2018 Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pages 1–6, August 2018. doi:10.1109/ICCUBEA.2018.8697857.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017a. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, October 2016. URL http://arxiv.org/abs/1609.08144. arXiv:1609.08144 [cs].

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need, December 2017b. URL http://arxiv.org/abs/1706.03762. arXiv:1706.03762 [cs].

Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics. doi:10.3115/v1/D14-1162. URL https://aclanthology.org/D14-1162.

Tomas Mikolov, Quoc V. Le, and Ilya Sutskever. Exploiting Similarities among Languages for Machine Translation, September 2013. URL http://arxiv.org/abs/1309.4168. arXiv:1309.4168 [cs].

Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. FastText.zip: Compressing text classification models, December 2016. URL http://arxiv.org/abs/1612.03651. arXiv:1612.03651 [cs].

Philip May. Machine translated multilingual sts benchmark dataset. 2021. URL https://github.com/PhilipMay/stsb-multi-mt.

Ivan Habernal and Tomáš Brychcín. Unsupervised improving of sentiment analysis using global target context. In *Proceedings of RANLP 2013*, page TBD. Association for Computational Linguistics, 2013. URL TBD.