

Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering



Master's Thesis

**Building an NLP model for classifying short-tail conversational
student query data**

Kural Arasu Venkatesh

© 2023 CZU Prague

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

DIPLOMA THESIS ASSIGNMENT

Kural Arasu Venkatesh, B.E.

Informatics

Thesis title

Building NLP model for classifying short-tail conversational student's query data

Objectives of thesis

The main objective is to build and train Natural Language Processing (NLP) model to classify short-tail conversational dataset into intents (classes). Model is to be used as a part of virtual assistant (chatbot) solution.

The partial goals of this thesis are:

- to characterize the fundamentals of Deep learning and NLP,
- to create a dataset out of student's query.
- to evaluate and optimize the efficiency of the model.
- to compare the NLP model with existing models on the same dataset and provide with objective comparison.

Methodology

To achieve the objectives, there is the need to review literature for similar efforts to define a theoretical framework in which theory of Artificial Intelligence, Natural Language Processing (NLP) and latest advancement in Deep learning with the scope of NLP. Special focus is put on Long-Short-Term-Memory (LSTM) architecture that seems to be the best performing for similar tasks.

For creating a dataset, inputs and archives of queries should be collected from study department and one another approach to conduct the survey with students regarding the queries they have on university and faculty. Linguistic experts and psychologist will be consulted to validate the dataset and to understand the true intent of the individual. Dataset will be classified, labelled and split into training, test and validation sets in order to train the NLP model. The dataset is further pre-processed and featured engineered to convert all non-numerical to numerical. The architecture of neural network is designed based on the theoretical research.

Results are used to compare with existing NLP model. Research will be performed to improve the efficiency of the model using several optimization techniques to implement the model into real time system.

The proposed extent of the thesis

60 – 80 pages

Keywords

Artificial Intelligence, Natural Language Processing, Chatbot, Python, Long-Short-Term-Memory Architecture

Recommended information sources

Bird, S. (2016). Natural language processing with python. O'reilly Media
Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.
Chollet François. (2021) Deep learning with python. Manning
Shaw, Z. (2017). Learn python the hard way: A very simple introduction to the terrifyingly beautiful world of computers and code. Addison-Wesley.
Van Houdt, Greg & Mosquera, Carlos & Nápoles, Gonzalo. (2020). A Review on the Long Short-Term Memory Model. Artificial Intelligence Review.

Expected date of thesis defence

2022/23 SS – FEM

The Diploma Thesis Supervisor

doc. Ing. Jan Tyrychtr, Ph.D.

Supervising department

Department of Information Engineering

Advisor of thesis

Ing. Martin Čejka

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 18. 03. 2023

Declaration

I declare that I have worked on my master's thesis titled "Building an NLP model for classifying short-tail conversational student query data" 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 31/03/2023

Acknowledgments

I would like to thank doc. Ing. Jan Tyrychtr for his guidance and supervision, Ing. Martin Čejka for his extended support, friends and family for their support, and a big shoutout to myself for staying perseverant throughout the process of this thesis.

Building an NLP model for classifying short-tail conversational student query data

Abstract

The objective of this thesis is to construct a Natural Language Processing (NLP) model capable of effectively categorizing students' queries based on their respective intents, thereby enabling their efficient and accurate routing to the appropriate department. To accomplish this, an LSTM model was selected owing to its well-established capacity for handling textual data. Prior to feeding the data into the neural network, pre-processing was carried out to convert the inputs into numerical values that could be processed by the neurons. The training and evaluation of the model were performed using two distinct datasets, with Accuracy being the primary performance metric. Additionally, a comparative analysis was conducted by comparing the outcomes of the LSTM model with those of a pre-existing model in the market, IBM Watson. The results demonstrated that the LSTM model outperformed IBM Watson in accurately classifying students' queries into their respective intents. The proposed classification model has considerable potential for use in university chatbots, as it could substantially aid prospective students and academic institutions in streamlining query organization and improving overall operational efficiency.

Keywords: Artificial Intelligence, Natural Language Processing, Chatbot, Python, Long-Short-Term-Memory Architecture

Vytvoření modelu NLP pro klasifikaci dat konverzačních dotazů studentů s krátkým koncem

Abstrakt

Cílem této diplomové práce je vytvořit model zpracování přirozeného jazyka (Natural Language Processing - NLP), který bude schopen efektivně kategorizovat dotazy studentů na základě jejich konkrétních záměrů a umožní tak jejich rychlé a přesné směrování na příslušné oddělení. K dosažení tohoto cíle byla vybrána architektura LSTM, jejíž vhodnost pro zpracování textových dat je dlouhodobě ověřena. Před vstupem dat do neuronové sítě bylo provedeno předzpracování zajišťující možný a vhodný převod vstupních dat do vektorů vstupujících do navržené neuronové sítě. Trénink a evaluace modelu byly provedeny na dvou odlišných datasetech, přičemž primární výkonnostní metrikou byla přesnost. Kromě toho byla provedena komparativní analýza s komerčním modelem IBM Watson. Výsledky ukázaly, že model LSTM je přesnější v klasifikaci dotazů studentů do konkrétních záměrů v případě trénování přímo pro tento případ použití. Navržený klasifikační model má velký potenciál pro využití v dialozích virtuálních asistentů univerzit, protože může výrazně pomoci budoucím studentům a akademickým institucím při organizaci dotazů a zlepšit celkovou efektivitu provozu.

Klíčová slova: Umělá inteligence, Zpracování přirozeného jazyka, Chatbot, Python, Architektura Long-Short-Term-Memory

Table of Contents

1. INTRODUCTION.....	1
2. OBJECTIVES AND METHODOLOGY	2
2.1 OBJECTIVE.....	2
2.2 METHODOLOGY	2
3. LITERATURE REVIEW	4
3.1 INTRODUCTION	4
3.2 ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, AND DEEP LEARNING	6
3.2.1 <i>Artificial intelligence</i>	6
3.2.2 <i>Acting humanly: The Turing test approach</i>	7
3.2.4 <i>Machine learning methods</i>	8
3.2.5 <i>Deep learning</i>	9
3.3 TEXT CLASSIFICATION METHODS	10
3.3.1 <i>Text representation</i>	11
3.3.2 <i>N-Gram</i>	11
3.3.3 <i>Bag of Words (BoW)</i>	12
3.4 WORD EMBEDDING	13
3.4.1 <i>Word2Vec</i>	13
3.4.2 <i>Continuous Bag-of-Words Model</i>	14
3.4.3 <i>Continuous Skip-Gram Model</i>	15
3.4.4 <i>Traditional method</i>	15
3.4.5 <i>Probabilistic model</i>	15
3.4.6 <i>KNN-based Methods</i>	16
3.5 DEEP LEARNING METHOD.....	17
3.5.1 <i>ReNN-based Methods</i>	17
3.5.2 <i>MLP-based Methods</i>	18
3.5.3 <i>RNN-based Methods</i>	18
3.5.4 <i>CNN-based Methods</i>	19
3.5.5 <i>Attention-based Methods</i>	20
3.5.6 <i>Transformer-based Methods</i>	21
3.6 RECURRENT NEURAL NETWORKS AND THEIR TYPES.....	23
3.6.1 <i>Fully Recurrent Neural Network</i>	23
3.6.2 <i>Recursive Neural network</i>	23

3.6.3	<i>Long short-term memory (LSTM)</i>	24
3.6.4	<i>Long short-term memory (LSTM) architecture</i>	24
3.7	EVALUATION METRICS.....	26
3.7.1	<i>Confusion Matrix</i>	27
3.7.2	<i>Precision & Recall</i>	27
3.7.3	<i>F1-Score</i>	29
4.	PRACTICAL PART	30
4.1	INTRODUCTION	30
4.2	DATASET	30
4.3	ENVIRONMENT.....	31
4.4	CODING	32
4.4.1	<i>Data Preprocessing</i>	36
4.5	MODEL BUILDING	44
4.5.1	<i>Building LSTM Network</i>	44
4.5.2	<i>Compiling the LSTM model</i>	47
4.5.3	<i>Fitting the LSTM model</i>	48
4.5.4	<i>Evaluation of the LSTM model</i>	50
4.5.5	<i>Plotting the LSTM model</i>	51
4.6	COMPARATIVE EVALUATION OF THE ACCURACY BETWEEN IBM WATSON AND LSTM ...	52
4.6.1	<i>Evaluation of intent classification with IBM Watson</i>	58
5.	RESULT	59
6.	CONCLUSION	60
6.1	FUTURE SCOPE.....	60
7.	REFERENCES	62
8.	LIST OF FIGURES	69
8.1	LIST OF FIGURES	69
8.2	APPENDIX.....	70

1. Introduction

Each year, a copious number of individuals enroll in universities to follow their interests and find their strengths. In the present competitive environment, the admissions process, clearing entrance tests, and preparing for interviews might be nerve-racking. Students anticipate easier registration, study administrative procedures, and a smooth transition into the first few experiences required to begin their study program. International students find it challenging to deal with the early hurdles of acclimating to a new place and fitting into the structure of academic programs.

Throughout these procedures, several questions emerge. Some of the most perplexing components would be dealing with errors and questions, answering these questions, and contacting assistance for specific questions. Providing simple navigation to discover answers to these questions and correcting errors would be a wonderful experience that a student would value. If these fundamental issues are addressed, students will have a better experience.

This thesis focuses on developing a method for categorizing questions into their various kinds and departments. Further efforts and improvements in the future will guarantee that inquiries are sent to the appropriate department and that students have the relevant contact information. This model will aid in the effective resolution of such procedures.

To categorize short-tail conversational query datasets into intents, this approach is to experiment and evaluate the performance of Long Short-Term Memory (LSTM) and further compare it with an existing commercial pre-built model (IBM Watson). LSTM is a form of recurrent neural network (RNN) that excels at processing sequential input. It can accept input sequences of varying lengths. This is significant because text inputs can be of varying lengths, whereas standard machine learning models frequently demand fixed-length inputs. LSTM may also learn to extract useful characteristics from the input sequence rather than depending on hand-crafted features. It has been chosen since it suits the requirement of the proposed thesis.

2. Objectives and Methodology

2.1 Objective

The primary aim of this study is to develop and train a Natural Language Processing (NLP) model capable of categorizing short-tail conversational datasets into distinct intents (classes), to be incorporated into a virtual assistant (chatbot) solution.

In addition to this overarching objective, several subsidiary goals have been identified, including the exploration of fundamental principles underlying both Deep Learning and NLP, the creation of a specialized dataset derived from student queries, and the construction of an LSTM model for the purpose of evaluating its efficiency relative to existing models, with the aim of producing an objective comparison.

2.2 Methodology

In order to accomplish the objectives, a comprehensive review of relevant literature is required, focusing on prior studies concerning the intersection of Artificial Intelligence, Natural Language Processing (NLP), and the latest advances in Deep Learning, with a particular emphasis on the Long Short-Term Memory (LSTM) architecture. This review will facilitate the establishment of a theoretical framework for the study, which will provide a comprehensive understanding of the subject matter and its practical implications.

The input queries will be created through surveys, the queries will be categorized to their intents. The queries are pre-processed into tokens, and the associated categories are then one-hot encoded, rendering them amenable to utilization within the context of a neural network model. These data are then shuffled and partitioned into separate training and testing sets, with accuracy serving as the performance metric. The model is constructed employing the Keras library, which incorporates requisite functionality for the development of neural network layers. A long short-term memory (LSTM) neural network is incorporated into the model architecture, with dropout layers added to mitigate the issue of overfitting. Following sequential layering of the neural network, the model is trained with the training dataset, utilizing hyperparameters and callbacks. The model is subsequently evaluated using the test dataset, and the results are then

compared with those obtained from existing commercial pre-built models. The research will also seek to improve model efficiency via the implementation of a range of optimization techniques, with the ultimate goal of integrating the model into a real-time system.

3. Literature Review

3.1 Introduction

Text classification is one of the most important tasks of machine learning and has been widely used in several areas of Natural Language Processing (NLP), Especially with recent breakthroughs in Natural Language Processing (NLP) and text mining, many researchers are now interested in developing applications that leverage text classification methods, Its objective is to design appropriately algorithms to allow computers to extract features and classify texts, classify large amounts of text data manually is time-consuming and challenging to process,

Besides, the accuracy of manual text classification can be easily influenced by human factors, such as fatigue and expertise. Using machine learning methods to automate the text classification procedure is desirable to yield more reliable and less subjective results.

Moreover, this can also help enhance information retrieval efficiency and alleviate the problem of information overload by locating the required information [1].

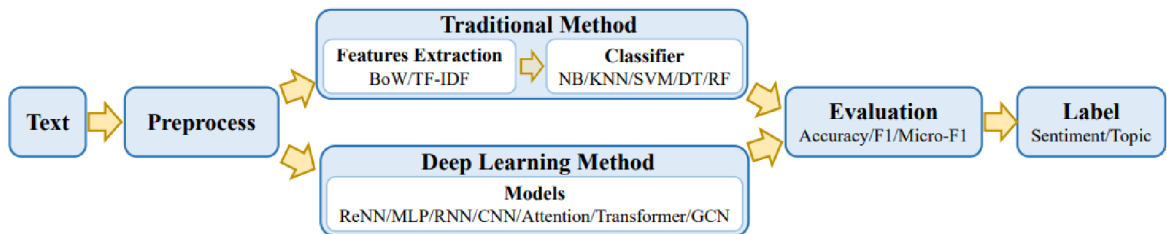


Figure 1. Text classification flowchart

Fig. 1 demonstrates a flowchart of the procedures involved in the text classification, in traditional and deep analysis. Text data is different from numerical, image, or signal data. It requires NLP techniques to be processed carefully.

The first important step is to preprocess text data for the model. Traditional models usually need to obtain good sample features by artificial methods and then classify them with classic machine learning algorithms.

Therefore, the effectiveness of the method is largely restricted by feature extraction. However, different from traditional models, deep learning integrates feature engineering into the model fitting process by learning a set of nonlinear transformations that serve to map features directly to outputs.

From the 1960s until the 2010s, traditional text classification models dominated. Traditional methods mean statistics-based models, like Naïve Bayes (NB) [2], K-Nearest Neighbor (KNN) [3], and Support Vector Machine (SVM) [4].

Compared with the earlier rule-based methods, this method has obvious advantages in accuracy and stability. However, these approaches still need to do feature engineering, which is time-consuming and costly. Besides, they usually disregard the natural sequential structure or contextual information in textual data, making it challenging to learn the semantic information of the words. Since the 2010s, text classification has gradually changed from traditional models to deep learning models. [3]

Compared with the methods based on traditional, deep learning methods avoid designing rules and features by humans and automatically provide semantically meaningful representations for text mining. Therefore, most of the text classification research works are based on Deep Neural Networks (DNNs), which are data-driven approaches with high computational complexity. Few works focus on traditional models to settle the limitations of computation and data. [5]

3.2 Artificial intelligence, machine learning, and deep learning

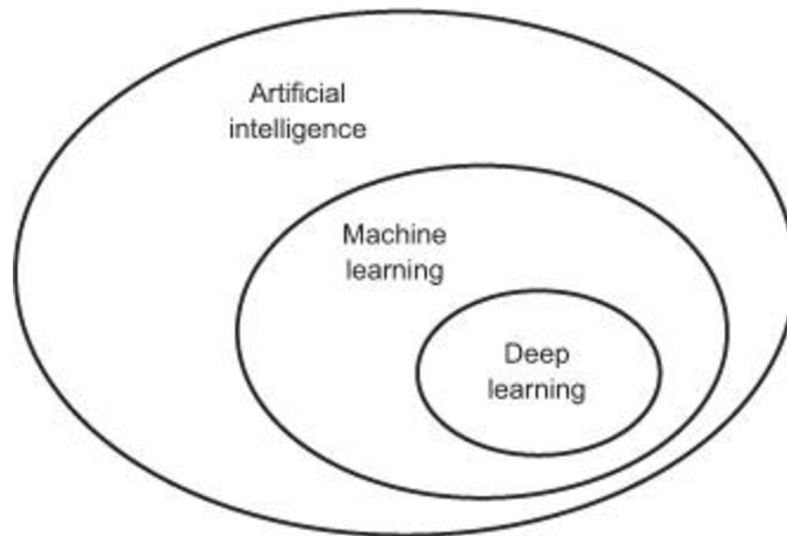


Figure 2. Artificial intelligence, machine learning, and deep learning. [6]

First, we need to define clearly what we’re talking about when we mention AI. What are artificial intelligence, machine learning, and deep learning?

3.2.1 Artificial intelligence

Artificial intelligence was born in the 1950s when a handful of pioneers from the nascent field of computer science started asking whether computers could be made to “think”—a question whose ramifications we’re still exploring today.

A concise definition of the field would be as follows: the effort to automate intellectual tasks normally performed by humans. As such, AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don’t involve any learning. [6]

Early chess programs, for instance, only involved hardcoded rules crafted by programmers and didn’t qualify as machine learning. For a long time, many experts believed that human-level artificial intelligence could be achieved by having programmers handcraft a large set of explicit rules for manipulating knowledge.

This approach is known as symbolic AI, and it was the dominant paradigm in AI from the 1950s to the late 1980s. It reached its peak popularity during the expert systems boom of the 1980s.

Although symbolic AI proved suitable to solve well-defined, logical problems, such as playing chess, it turned out to be intractable to figure out explicit rules for solving more complex, fuzzy problems, such as image classification, speech recognition, and language translation. [8]

3.2.2 Acting humanly: The Turing test approach.

“A computer would deserve to be called intelligent if it could deceive a human into believing that it was human.” Alan Turing

The Turing test measures a machine's capacity to behave intelligently in a way that cannot be distinguished from human behavior. To determine if robots are capable of thinking, British mathematician and computer scientist Alan Turing created the test in 1950. [48]

A human assessor converses in normal language with both a person and a machine as part of the Turing Test. It is unknown to the assessor which side is a machine and which is a person. The Turing Test is claimed to have been passed by a machine if the assessor can consistently tell the difference between the computer's replies and those of a person. [48]

The Turing Test is frequently used as a yardstick to assess the sophistication of AI systems, especially those intended for natural language processing. Nevertheless, some detractors contend that passing the Turing test does not always imply genuine intelligence because a machine may imitate human behavior without truly comprehending the underlying ideas. The Turing Test is still a well-known and significant idea in the field of AI.

3.2.3 Machine learning

Machine learning is a branch of artificial intelligence (AI) and computer science that focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. [7]

Over the last couple of decades, technological advances in storage and processing power have enabled some innovative products based on machine learning, such as Netflix's recommendation engine and self-driving cars.

Machine learning is an important component of the growing field of data science. Using statistical methods, algorithms are trained to make classifications or predictions and to uncover key insights in data mining projects. These insights subsequently drive decision-making within applications and businesses, ideally impacting key growth metrics. [8]

As big data continues to expand and grow, the market demand for data scientists will increase. They will be required to help identify the most relevant business questions and the data to answer them.

3.2.4 Machine learning methods

a. Supervised

Supervised learning [8] typically begins with an established set of data and a certain understanding of how that data is classified.

Supervised learning is intended to find patterns in data that can be applied to an analytics process. This data has labeled features that define the meaning of data.

For example, there could be millions of images of animals and include an explanation of what each animal is and then you can create a machine-learning application that distinguishes one animal from another. By labeling this data about types of animals, you may have hundreds of categories of different species. Because the attributes and the meaning of the data have been identified

b. Unsupervised

Unsupervised learning is best suited when the problem requires a massive amount of data that is labeled. For example, social media applications, such as Twitter, Instagram, Snapchat, and so on all have large amounts of unlabeled data. [8]

Understanding the meaning behind this data requires algorithms that can begin to understand the meaning based on being able to classify the data based on the patterns or clusters it finds. Therefore, supervised learning conducts an iterative process of analyzing data without human intervention.

Unsupervised learning is used with email spam-detecting technology. There are far too many variables in legitimate and spam emails for an analyst to flag unsolicited bulk emails. Instead, machine learning classifiers based on clustering and association are applied to identify unwanted emails.

c. Semi-Supervised

Semi-supervised combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning (with no labeled training data) and supervised learning (with only labeled training data).

Semi-supervised learning aims to alleviate the issue of having limited amounts of labeled data available for training and is motivated by problem settings where unlabeled data is abundant and obtaining labeled data is expensive. [9]

3.2.5 Deep learning

Deep learning is a specific method of machine learning that incorporates neural networks in successive layers to learn from data in an iterative manner. Deep learning is especially useful when you're trying to learn patterns from unstructured data. [10]

Deep learning has lately been shown to be a very powerful tool for a wide range of problems, Deep learning refers to complex neural networks designed to emulate how the human

brain works so computers can be trained to deal with abstractions and problems that are poorly defined. The average five-year-old child can easily recognize the difference between his teacher's face and the face of the crossing guard. In contrast, the computer must do a lot of work to figure out who is who. [6]

A neural network consists of three or more layers: an input layer, one or many hidden layers, and an output layer. Data is ingested through the input layer. Then the data is modified in the hidden layer and the output layers based on the weights applied to these nodes. The typical neural network may consist of thousands or even millions of simple processing nodes that are densely interconnected. The term deep learning is used when there are multiple hidden layers within a neural network. [10]

Using an iterative approach, a neural network continuously adjusts and makes inferences until a specific stopping point is reached, networks and deep learning are often used in image recognition, speech, and computer vision applications.[46]

3.3 Text Classification Methods

Text classification is referred to as extracting features from raw text data and predicting the categories of text data based on such features. Numerous models have been proposed in the past few decades for text classification.

For traditional models, NB [2] is the first model used for the text classification task. Whereafter, generic classification models are proposed, such as KNN [3], SVM [4], and Random Forest (RF) [11], which are called classifiers, and widely used for text classification. Recently, the extreme Gradient Boosting (XGBoost) [12] and the Light Gradient Boosting Machine (LightGBM) [13] have arguably the potential to provide excellent performance. For deep learning models, TextCNN [14] has the highest number of references in these models, wherein a Convolutional Neural Network (CNN) [15] model has been introduced to solve the text classification problem for the first time.

While not specifically designed for handling text classification tasks, the Bidirectional Encoder Representation from Transformers (BERT) [16] has been widely employed when

designing text classification models, considering its effectiveness on numerous text classification datasets.

3.3.1 Text representation

Many researchers have worked on this text feature extraction technique to solve the losing syntactic and semantic relationships between words. Many researchers addressed novel techniques for solving this problem, but many of these techniques still have limitations.

A model was introduced in which the usefulness of including syntactic and semantic knowledge in the text representation for the selection of sentences comes from technical genomic texts. The other solution for the syntactic problem is using the n-gram technique for feature extraction.

3.3.2 N-Gram

The n-gram technique is a set of n-words that occurs “in that order” in a text set. This is not a representation of a text, but it could be used as a feature to represent a text. BOW is a representation of a text using its words (1-gram) which loses their order (syntactic). [18]

This model is very easy to obtain, and the text can be represented through a vector, generally of the manageable size of the text. On the other hand, an n-gram is a feature of BOW for a representation of a text using 1-gram. It is very common to use 2-gram and 3-gram. In this way, the text feature extracted could detect more information in comparison to 1-gram. [27]

An Example of 2-Gram:

After sleeping for four hours, he decided to sleep for another four.

In this case, the tokens are as follows:

{“After sleeping”, “sleeping for”, “for four”, “four hours”, “four he” “he decided”, “decided to”, “to sleep”, “sleep for”, “for another”, “another four”}.

An Example of 3-Gram:

After sleeping for four hours, he decided to sleep for another four.

In this case, the tokens are as follows:

{“After sleeping for”, “sleeping for four”, “four hours he”, “hours he decided”, “he decided to”, “to sleep for”, “sleep for another”, “for another four”}.

3.3.3 Bag of Words (BoW)

The bag-of-words model (BoW model) is a reduced and simplified representation of a text document from selected parts of the text, based on specific criteria, such as word frequency.

The BoW technique is used in several domains such as computer vision, NLP, Bayesian spam filters, as well as document classification and information retrieval by Machine Learning. In a BoW, a body of text, such as a document or a sentence, is thought of as a bag of words. Lists of words are created in the BoW process. [17]

These words in a matrix are not sentences that structure sentences and grammar, and the semantic relationship between these words is ignored in their collection and construction. The words are often representative of the content of a sentence. [46]

While grammar and order of appearance are ignored, multiplicity is counted and may be used later to determine the focus points of the documents.

Here is an example of BoW:

Document:

“As the home to UVA’s recognized undergraduate and graduate degree programs in systems engineering. In the UVA Department of Systems and Information Engineering, our students are exposed to a wide range of range.”

Bag-of-Words (BoW):

{“As”, “the”, “home”, “to”, “UVA’s”, “recognized”, “undergraduate”, “and”, “graduate”, “degree”, “program”, “in”, “systems”, “engineering”, “in”, “Department”, “Information”, “students”, “our”, “exposed”, “wide”, “range”}

Bag-of-Feature (BoF):

Feature = {1,1,1,3,2,1,2,1,2,3,1,1,1,2,1,1,1,1,1,1}

3.4 Word Embedding

Word embedding is a feature learning technique in which each word or phrase from the vocabulary is mapped to an N-dimension vector of real numbers. Various word embedding methods have been proposed to translate unigrams into understandable input for machine learning algorithms. This work focuses on Word2Vec, GloVe, and FastText, three of the most common methods that have been successfully used for deep learning techniques.

Recently, a Novel technique of word representation was introduced where word vectors depend on the context of the word called “Contextualized Word Representations” or “Deep Contextualized Word Representations”.

3.4.1 Word2Vec

The Word2Vec approach uses shallow neural networks with two hidden layers, continuous bag-of-words (CBOW), and the Skip-gram model to create a high-dimension vector for each word. The Skip-gram model dives into a corpus of words w and context c , the goal is to maximize the probability:

$$\arg \max \prod_{w \in T} \left[\prod_{c \in c(w)} p(c | w; \theta) \right]$$

where T refers to Text, and θ is the parameter of $p(c | w; \theta)$.

Figure 3 shows a simple CBOW model which tries to find the word based on previous words, while Skip-gram tries to find words that might come in the vicinity of each word. The weights between the input layer and output layer represent $v \times N$ as a matrix of w .

This method provides a very powerful tool for discovering relationships in the text corpus as well as similarities between words. For example, this embedding would consider the two words such as “big” and “bigger” close to each other in the vector space it assigns them.

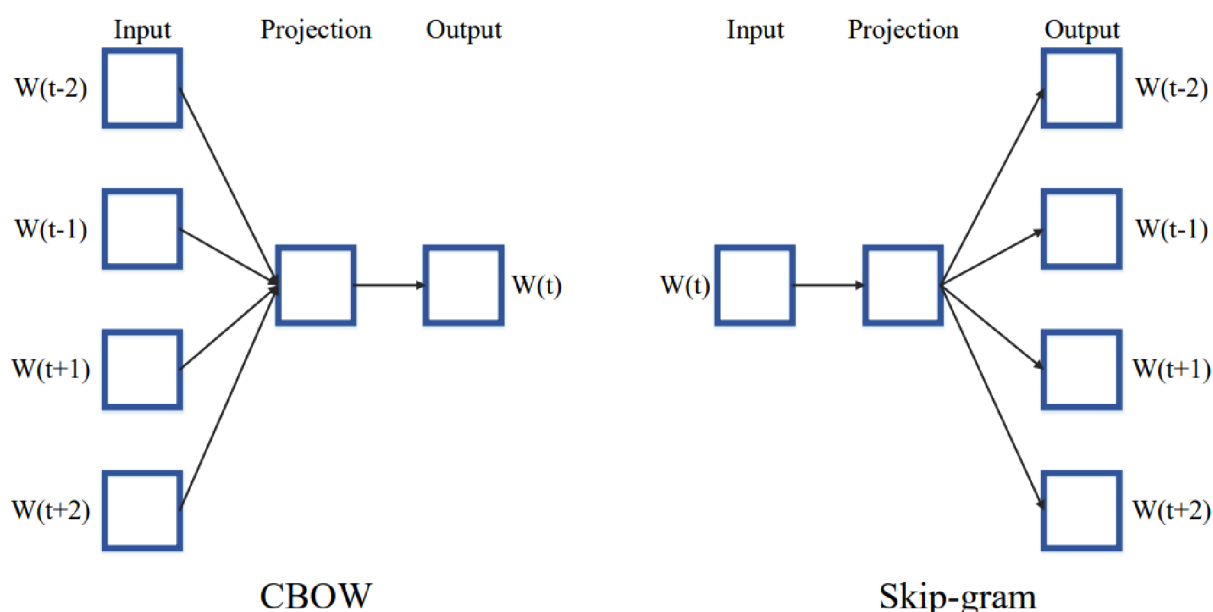


Figure 3. The structure of word2vec, including CBOW and Skip-gram. [17]

3.4.2 Continuous Bag-of-Words Model

The continuous bag-of-words model is represented by multiple words for a given target of words. For example, the word “airplane” and “military” as context words for “air force” as the target word. This consists of replicating the input to hidden layer connections β times which is the number of context words [20]. Thus, the bag-of-words model is mostly used to represent an unordered collection of words as a vector. The first thing to do is create a vocabulary, which means all the unique words in the corpus. The output of the shallow neural network will be that the task is “predicting the word given its context”. The number of words used depends on the setting for the window size (the common size is 4–5 words).

3.4.3 Continuous Skip-Gram Model

Another model architecture that is very similar to CBOW [61] is the continuous Skip-gram model, however, this model, instead of predicting the current word based on the context, tries to maximize the classification of a word based on another word in the same sentence. The continuous bag-of-words model and continuous Skip-gram model are used to keep syntactic and semantic information of sentences for machine learning algorithms.

As mentioned previously there are two methods to build a text classification system let's discuss them.

3.4.4 Traditional method

Traditional models accelerate text classification with improved accuracy and make the application scope of traditional expand. first is to preprocess the raw input text for training traditional models, which generally consist of word segmentation, data cleaning, and statistics.

The first step towards training a Traditional NLP classifier is feature extraction using methods such as Bag-Of-Words (BOW) [17], N-gram [18], Term Frequency-Inverse Document Frequency (TF-IDF) [19], word2vec [20], and Global Vectors for word representation (GloVe) [21].

3.4.5 Probabilistic model

Naive Bayes classifiers are linear classifiers that are known for being simple yet very efficient, one of the members of that family is Multinomial Naive Bayes (MNB) with a huge advantage, that you can get really good results even when your dataset isn't very large (~ a couple of thousand tagged samples) and computational resources are scarce.

The probabilistic model of naive Bayes classifiers is based on Bayes' theorem, and the adjective naive comes from the assumption that the features in a dataset are mutually independent. In practice, the independence assumption is often violated, but naive Bayes

classifiers still tend to perform very well under this unrealistic assumption [1]. Especially for small sample sizes, naive Bayes classifiers can outperform the more powerful alternatives.

To understand how naive Bayes classifiers work, we must briefly recapitulate the concept of Bayes' rule. The probability model that was formulated by Thomas Bayes (1701-1761) is quite simple yet powerful; it can be written down in simple words as follows:

$$\text{posterior probability} = \frac{\text{conditional probability} \cdot \text{prior probability}}{\text{evidence}} \quad (1)$$

Bayes' theorem forms the core of the whole concept of naive Bayes classification. The posterior probability, in the context of a classification problem, can be interpreted as: "What is the probability that a particular object belongs to class 'i' given its observed feature values?" A more concrete example would be: "What is the probability that a person has diabetes given a certain

3.4.6 KNN-based Methods

KNN algorithm calculates that most of the k nearest neighbors in a feature space belong to a certain category, and the sample also belongs to this category. The algorithm involves several main factors: distance measurement, k-value selection, and so on [6].

First, an experiment of K value selection is performed, and the optimal k value is selected from the seven news text data by a simple cross-validation method. The test set is validated by using the model obtained from the training set to interfere with the sample selection of the training set and the test set.

The results show that the effect is the best when K=4. Secondly, distance measurements are used to measure the distance between individuals in a space. Euclidean distance is the most common distance metric used to measure the absolute distance between points in a multi-dimensional space.

$$Dist(x, y) = \sqrt{\sum_{i=1}^n (x(i) - y(i))^2}$$

Minkowski distance is a common measure of the distance between numerical points, but it is not a distance, but a set of definitions of distance: the Minkowski distance between two n-dimensional variables $A = (x_{11}, x_{12}, \dots, x_{1n})$ and $B = (x_{21}, x_{22}, \dots, x_{2n})$ is defined as

$$D_{12} = \sqrt[p]{\sum_{k=1}^n |X_1 - X_2|}$$

3.5 Deep Learning method

The DNNs consist of artificial neural networks that simulate the human brain to automatically learn high-level features from data, getting better results than traditional models in speech recognition, image processing, and text understanding. Input datasets should be analyzed to classify the data, such as a single-label, multi-label, unsupervised, or unbalanced dataset. According to the trait of the dataset, the input word vectors are sent into the DNN for training until the termination condition is reached.

The performance of the training model is verified by the downstream task, such as sentiment classification, question answering, and event prediction.

3.5.1 ReNN-based Methods

The emergence of the semi-supervised recursive autoencoder (Semi-Supervised RAE) [23] model has raised the recursive neural network (ReNN) to a level, and they combined the recurrent neural network with Autoencoder to form an unsupervised sentence conversion model. Sentences of indeterminate length can be converted into sentence vectors. Socher et al. applied it to the prediction of sentiment label distribution and outperformed the state-of-the-art methods at the time on commonly used datasets.

However, the RAE results are relatively general in terms of accuracy, and the single-word vector model is still insufficient for semantic modelling, especially since the true semantics of deeper phrases cannot be learned well. To improve the inability to capture long phrases and enable the model to understand language more deeply, Socher et al. proposed a novel recursive neural network model MV-RNN [24] for semantic compositionality.

The main changes are assigning a vector and a matrix to each word, and learning an input-specific, non-linear function for computing vector and matrix representations for sentences of arbitrary syntactic type.

This method can well reflect the influence between adjacent nodes, but its disadvantage is that the parameter size depends on the size of the vocabulary and the computational efficiency is low.

3.5.2 MLP-based Methods

The researcher points out that the current processing of natural language processing tasks, such as sentiment analysis, intelligent question answering, and other tasks, are all based on the vectorized representation of text, that is, how to construct an appropriate composition function.

The methods can be mainly divided into two categories: unordered and syntactic, unordered training is fast, but because it is not sensitive to word order, the accuracy is not high; the syntactic method, although the model performance will be better, but the improved performance of the model does not match its cost. Therefore, the author tried to find a combination point, and DAN [25] was born.

The first part of the model was summed and averaged, and a word dropout was also proposed, which is to randomly drop some tokens in the input to increase the robustness of the model. The second part adds the semantic information of the multi-layer nonlinear layers to extract the summed and averaged vectors. Each deep layer of the model is more abstract.

3.5.3 RNN-based Methods

The most fundamental problem with RNN is the short-term memory problem, such as using past chapters in a novel to infer the occurrence of the current chapter, which RNN cannot do.

To solve this problem, LSTM [5] and GRU were born successively. Specifically, RNN cannot capture long-range dependence due to the simple repetition of single neurons, In the backpropagation process of RNN, the weights are adjusted by gradients, calculated by continuous multiplications of derivatives. If the derivatives are extremely small, it may cause a gradient vanishing problem by continuous multiplications.

Long Short-Term Memory (LSTM) [26], the improvement of RNN, effectively alleviates the gradient vanishing problem. It is composed of a cell to remember values on arbitrary time intervals and three gate structures to control information flow. The gate structures include input gates, forget gates and output gates. The LSTM classification method can better capture the connection among context feature words and use the forgotten gate structure to filter useless information, which is conducive to improving the total capturing ability of the classifier.

Tree-LSTM extends the sequence of LSTM models to the tree structure. The whole subtree with little influence on the result can be forgotten through the LSTM forgetting gate mechanism for the Tree-LSTM model. Natural Language Inference (NLI) [27] predicts whether one text's meaning can be deduced from another by measuring the semantic similarity between each pair of sentences.

To consider other granular matchings and matchings in the reverse direction, Wang et al. [28] propose a model for the NLI task named Bilateral Multi-Perspective Matching (BiMPM). It encodes input sentences by the BiLSTM encoder. Then, the encoded sentences are matched in two directions.

The results are aggregated in a fixed-length matching vector by another BiLSTM layer. Finally, the result is evaluated by a fully connected layer.

3.5.4 CNN-based Methods

Convolutional Neural Networks (CNNs) [29] are proposed for image classification with convolving filters that can extract features of pictures. Unlike RNN, CNN can simultaneously apply convolutions defined by different kernels to multiple chunks of a sequence.

Therefore, CNNs are used for many NLP tasks, including text classification. For text classification, the text requires being represented as a vector like an image representation, and text features can be filtered from multiple angles. Firstly, the word vectors of the input text are spliced into a matrix.

The matrix is then fed into the convolutional layer, which contains several filters with different dimensions. Finally, the result of the convolutional layer goes through the pooling layer and concatenates the pooling result to obtain the final vector representation of the text. The category is predicted by the final vector. To try using CNN for the text classification task, an unbiased model of convolutional neural networks was introduced by Kim, called TextCNN [30]. It can better determine discriminative phrases in the max-pooling layer with one layer of convolution and learn hyperparameters except for word vectors by keeping word vectors static.

Training only on labeled data is not enough for data-driven deep models. Therefore, some researchers consider utilizing unlabeled data. Johnson et al. [31] propose a CNN model based on two-view semi-supervised learning for text classification, which first uses unlabeled data to train the embedding of text regions and then labeled data.

DNNs usually have better performance, but it increases the computational complexity. Motivated by this, a Deep Pyramid Convolutional Neural Network (DPCNN) [32] is proposed, with a little more computational accuracy, increasing by raising the network depth. The DPCNN is more specific than Residual Network (ResNet) [33], as all the shortcuts are exactly simple identity mapping without any complication for dimension matching

3.5.5 Attention-based Methods

The text classification algorithm mentioned above is basically sentence-level classification, using long text and chapter-level, although it is also possible to achieve its classification, but the speed and accuracy will decrease. So, some researchers proposed a hierarchical attention [34] classification framework, that is, the model of Hierarchical Attention.

The researchers said that when classifying documents/longer texts, it is not enough to only pay attention to the word granularity, it is necessary to learn attention to each sentence (short

sentence). Different sentences also need to be assigned different weights, and the words in each sentence are also assigned different weights.

The specific process is to first encode each sentence with BiGRU+Att to obtain the sentence vector, and then use BiGRU+Att to obtain the doc-level representation of the sentence vector, and then classify.

The researchers used six datasets, the smallest dataset also contains 33w articles, and the largest dataset has ten times as many as the smallest dataset. The researchers used three combinations, HNAVE and HN-MAX change the method of generating feature vectors of each layer from the weighted summation of Attention to directly doing Average Pooling and Max Pooling, but the effect of Attention is still better than the former two.

Word embedding is the most classical model of word representation in NLP. However, word2vec is essentially a static model. The expression of each word in this model is fixed and does not capture contextual information dynamically.

To solve this problem, ELMO dynamically acquires textual contextual information using bidirectional LSTM. Essentially, ELMo [35] is a model based on a pre-trained language model that dynamically characterizes Word embedding according to the current context. Researchers call this a domain transfer. In this way, the word embedding of words in the current context can be obtained by using the context information of our training data.

The researchers tested SQuAD with a baseline model that is an improved version of the bidirectional attention flow model. After adding ELMo to the baseline model, we get very good data. But ELMo is essentially an RNN, which is weak in feature extraction and training time.

3.5.6 Transformer-based Methods

BERT is a pre-training model proposed by Google AI Research in October 2018. BERT is two-way fine-tuning. Compared to feature-based pre-trained models, fine-tuning makes fewer architectural changes to pre-trained models. [36]

The input of BERT consists of token_embedding, segment_embedding, and position_embedding. BERT obtains generic text representations in the pretraining phase by simultaneously training two pre-training tasks of masked language model and next sentence prediction. Although BERT has its own bidirectional function, BERT consumes a lot of hardware resources.

Reasoning about relationships between multiple text words is involved in many NLP tasks. For example, in extractive question answering, the results are not satisfactory. So Mander et al. made a series of improvements to BERT and proposed SpanBERT [37].

They modified BERT's approach: Google BERT extracted 10 different masks for each sequence during data processing, and the reproduced BERT used a different mask for each epoch. And the setting of generating short sentences according to 10% is removed, and each sample is guaranteed to be 512 in length unless it reaches the end of the file.

After that, three major improvements were made based on BERT: instead of using a random mask method, a certain continuous token was used to mask; a task of predicting the mask by the boundary (Span Boundary Objective) was added; Through experiments, it is found that it is better to abandon the next sentence prediction task and train directly with long text.

In NLP tasks, a good pre-trained model can improve the performance of the model. The current SOTA model has millions or billions of parameters. If you want to expand the model size, you will encounter the limitations of this computer's memory, and the training speed will be limited. There are currently two solutions: model parallelization, and a good memory management mechanism. But both methods have communication overhead.

Therefore, the paper designs an A lite BERT (ALBERT), which uses fewer parameters than BERT. ALBERT [38] overcomes the main obstacle to scaling pre-trained models by introducing two parameter reduction techniques, factorized embedding parameterization and Cross-layer parameter sharing.

And the researchers also introduced an Intersentence coherence loss for sentence coherence modeling to solve the problem of inefficient next-sentence prediction loss in the original BERT.

3.6 Recurrent Neural Networks and their types

The Recurrent Neural Network (RNN) is a network with loops, which allows information to persist in the network. RNN has a feedback connection to the network itself, which allows activations to flow back in a loop, learn sequences, and information to persist.

RNNs are extremely powerful in modeling sequential data, speech, or text and are applied to nonsequential data to train in a non-sequential manner. RNN can be used for image, video captioning, word prediction, word translation, image processing, speech recognition, speech processing [39], natural language processing, music processing applications, etc. There are many types of RNN let's check some of them.

3.6.1 Fully Recurrent Neural Network

Fully recurrent neural network (FRNN) developed in the 1980s, which can learn temporal sequences, either in batch mode or online. FRNN consists of two layers, the input and output layers of linear and non-linear units, resp. The units in the input layer are fully connected to every unit of the output layer by adjustable weights. Each unit has a real-valued time-varying activation function.

The output units have some knowledge of their prior activations, which feedback on the activations to the input layer units. Learning in FRNNs is by mapping input sequences and activations, to another set of output sequences. This continues to feedback to input sequences and finding output sequences over multiple time steps, and over time discover abstract representations.

3.6.2 Recursive Neural network

The network is created in a differentiable graph-like structure by recursively applying the same set of weights to the network in topological order. Such networks are also trained by automatic differentiation [40] in reverse mode. It corresponds to linear chain structure and is used in natural language processing, processing distributed representation of the structure.

Variation of the recursive neural network is Recursive Neural Tensor Network which uses a tensor-based composition function on every network node.

3.6.3 Long short-term memory (LSTM)

LSTM is a system that can learn a task by using deep learning and avoids the vanishing gradient problem [41]. LSTM is normally improved by recurrent gates called "forget" gates and to learn tasks it requires memory of events that happened in history. LSTM can be learned by Connectionist Temporal Classification (CTC) which achieves both alignment and recognition for weights.

a. Gated Recurrent Units

The gating mechanism in RNN was introduced by Kyunghyun Cho [42] (2014). This mechanism lacks an output gate and has fewer parameters than LSTM. Its performance is like that of LSTM on polyphonic music and speech signal modeling.

b. Bi-directional RNN

Bi-directional Recurrent Neural Network predicts each element of a finite sequence based on its past/previous and future/next situation. It works in both directions for processing sequences from left to right and right to left and concatenating their output. This technique is useful when combined with LSTM [43].

3.6.4 Long short-term memory (LSTM) architecture

The LSTM contains special units called memory blocks in the recurrent hidden layer. The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block in the original architecture contained an input gate and an output gate.

The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. Later, the forget gate was added to the memory block [44]. This addressed a weakness of LSTM models preventing them from processing continuous input streams that are not segmented into subsequences.

The forget gate scales the internal state of the cell before adding it as input to the cell through the self-recurrent connection of the cell, therefore adaptively forgetting or resetting the cell's memory. In addition, the modern LSTM architecture contains peephole connections from its internal cells to the gates in the same cell to learn the precise timing of the outputs.

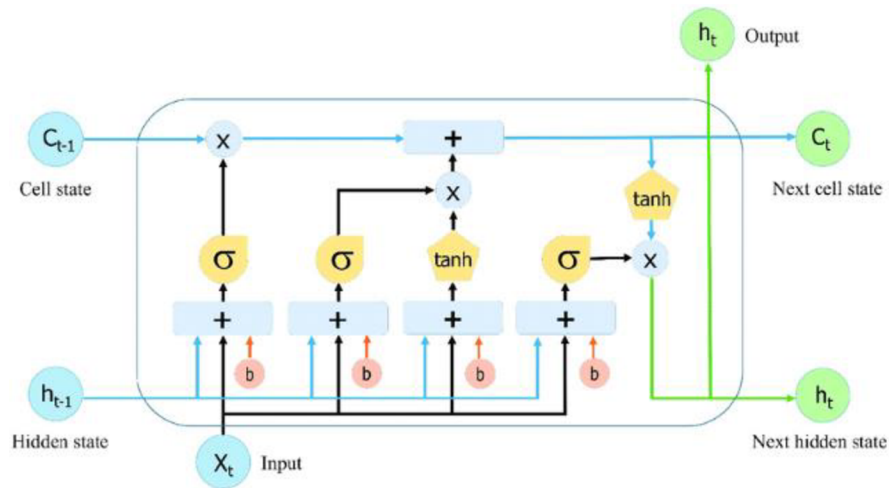


Figure 4. LSTM architecture. [26]

A simple LSTM network consists of the following components:

a. Forget Gate

One of the main properties of the LSTM is to memorize and recognize the information coming inside the network and to discard the information which is not required by the network to learn the data and predictions. This gate is responsible for this feature of the LSTM.

It helps in deciding whether information can pass through the layers of the network. There are two types of input it expects from the network one of them is the information from the previous layers and another one is the information from the presentation layer.

b. Input Gate

The input gate helps in deciding the importance of the information by updating the cell state. where the forget gate helps in the elimination of the information from the network input gate decides the measure of the importance of the information and helps the forget function in elimination of the not important information and other layers to learn the information which is important for making predictions.

c. Cell state

The weight gained information goes through the cell state where this layer calculates the cell state. In the cell state, the output of the forget gate and input gate gets multiplied by each other. The information which has the possibility of dropping out gets multiplied by near-zero values.

d. Output Gate

It is the last gate of the circuit that helps in deciding the next hidden state of the network in which information goes through the sigmoid function. The updated cell from the cell state goes to the tanh function then it gets multiplied by the sigmoid function of the output state. Which helps the hidden state to carry the information.

3.7 Evaluation Metrics

There are many metrics that come in handy to test the ability of any multi-class classifier and they turn out to be useful for:

- comparing the performance of two different models.
- analyzing the behavior of the same model by tuning different parameters.

Many metrics are based on the Confusion Matrix since it encloses all the relevant information about the algorithm and classification rule performance [45].

3.7.1 Confusion Matrix

The confusion matrix is a cross table that records the number of occurrences between two raters, the true/actual classification, and the predicted classification, as shown in Figure 5. For consistency reasons throughout the paper, the columns stand for model prediction whereas the rows display the true classification.

The classes are listed in the same order in the rows as in the columns, therefore the correctly classified elements are located on the main diagonal from top left to bottom right and they correspond to the number of times the two raters agree.

		PREDICTED classification				Total
		Classes	a	b	c	
ACTUAL classification	a	6	0	1	2	9
	b	3	9	1	1	14
	c	1	0	10	2	13
	d	1	2	1	12	16
Total		11	11	13	17	52

Figure 5. Example of a confusion matrix. [45]

In the following paragraphs, we review two-class classification concepts, which will come in handy later to understand multi-class concepts.

3.7.2 Precision & Recall

These metrics will act as building blocks for Balanced Accuracy and F1-Score formulas. Starting from a two-class confusion matrix:

		PREDICTED		Total
		Classes	Positive (1)	
ACTUAL	Positive (1)	TP = 20	FN = 5	25
	Negative (0)	FP = 10	TN = 15	25
Total		30	20	50

Figure 6. Two-class Confusion Matrix. [45]

The Precision is the fraction of True Positive elements divided by the total number of positively predicted units (column sum of the predicted positives). True positives are the elements that have been labeled as positive by the model and are actually positive, while False Positives are the elements that have been labeled as positive by the model but are actually negative.

$$precision = \frac{TP}{TP + FP}$$

Precision expresses the proportion of units our model says are Positive and, they are Positive. In other words, Precision tells us how much we can trust the model when it predicts an individual as Positive. The Recall is the fraction of True Positive elements divided by the total number of positively classified units (row sum of the actual positives). False negatives are the elements that have been labeled as negative by the model, but are positive.

$$Recall = \frac{TP}{TP + FN}$$

The Recall measures the model's predictive accuracy for the positive class: intuitively, it measures the ability of the model to find all the Positive units in the dataset. Hereafter, we present different metrics for the multi-class setting, outlining the pros and cons, with the aim to provide guidance to make the best choice.

3.7.3 F1-Score

Also, F1-Score assesses the classification model's performance starting from the confusion matrix, it aggregates Precision and Recall measures under the concept of harmonic mean.

$$F1 - Score = \left(\frac{2}{precision^{-1} + recall^{-1}} \right) = 2 \cdot \left(\frac{precision \cdot recall}{precision + recall} \right)$$

The formula of the F1-score can be interpreted as a weighted average between Precision and Recall, where F1-score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall are equal to the F1-score and the harmonic mean is useful to find the best trade-off between the two quantities [45].

4. Practical Part

4.1 Introduction

Based on the research, there are many latest approaches for NLP applications with an efficient output, but the architecture which changed the game of word embedding was LSTM. Therefore, the proposed approach consists of using a Long Short Term Memory Neural Network for word embedding and categorization of queries.

4.2 Dataset

The data used in this approach was created manually by surveying students from various faculties and partially created by using ChatGPT. The dataset contains two attributes, Query and Category.

	A	B	
1	Query	Category	
2	What type of visa do international students need to study in the czech republic?	VISA	
3	How can I apply for a student visa?	VISA	
4	How long does it take to process a student visa application?	VISA	
5	What documents do I need to provide when applying for a student visa?	VISA	
6	Is there a fee for a student visa?	VISA	
7	How long is a student visa valid for?	VISA	
8	Can I work while on a student visa?	VISA	
9	Can I bring my family with me on a student visa?	VISA	
10	Can I change my school or program while on a student visa?	VISA	
11	Can I apply for a student visa if I have a criminal record?	VISA	
12	Where is the test centre located?	Test center	
13	Where can I find the test centre?	Test center	
14	In which building is the test centre located?	Test center	
15	How do I get to the test centre?	Test center	
16	When is the test centre open?	Test center	

Figure 7. Student query dataset. [own work]

The dataset consists of 210 rows of queries and their respective categories. The queries have been categorized into 14 different categories with a minimum of 10 queries per category.

```
df.Category.value_counts()
```

Test center	51
admission	21
study management	17
technical support	16
lockers	13
accomodation	11
menza	11
VISA	10
library	10
sports	10
lost and found	10
emergency	10
activities and events	10
laundry	10

Name: Category, dtype: int64

Figure 8. Categories of Data. [own work]

4.3 Environment

Google Colab has been chosen as the coding environment as it is very similar to Jupyter Notebook.

It is a cloud-based platform that provides a free, convenient, and easy-to-use environment for working with data science and machine learning projects. With Google Colab, users can access powerful hardware resources, such as GPUs and TPUs, without having to worry about the cost and maintenance of the hardware.

Google Colab also comes with many pre-installed data science libraries and frameworks, such as TensorFlow, PyTorch, and scikit-learn, and can be imported at any point in time during coding., making it easy for users to get started with their projects quickly.

Moreover, Google Colab is synced with a google account which allows users to collaborate with others in real-time and provides features like version control, code sharing, and commenting.

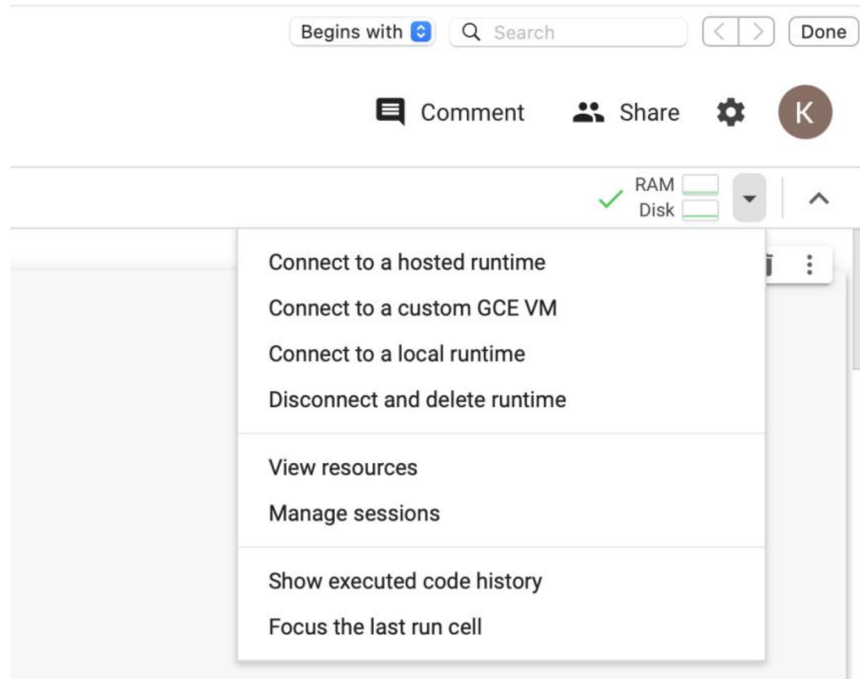


Figure 9. Google Colab environment. [own work]

4.4 Coding

Firstly, all the necessary libraries need to be imported to Colab's ipynb workspace. If necessary, any libraries can be imported into any part of the code.

```

Multi-Class Text Classification using LSTM.ipynb ☆
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

import pandas as pd
import matplotlib.pyplot as plt
from keras.preprocessing.text import Tokenizer
from keras.utils import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
from nltk.corpus import stopwords
from nltk import word_tokenize

```

Figure 10. Library imports in Google Colab. [own work]

To import the data into the workspace, google colab's inbuilt function "files" could be used. Executing the below code opens a dialogue box to select the file from the local drive.

```
from google.colab import files
uploaded = files.upload()
```

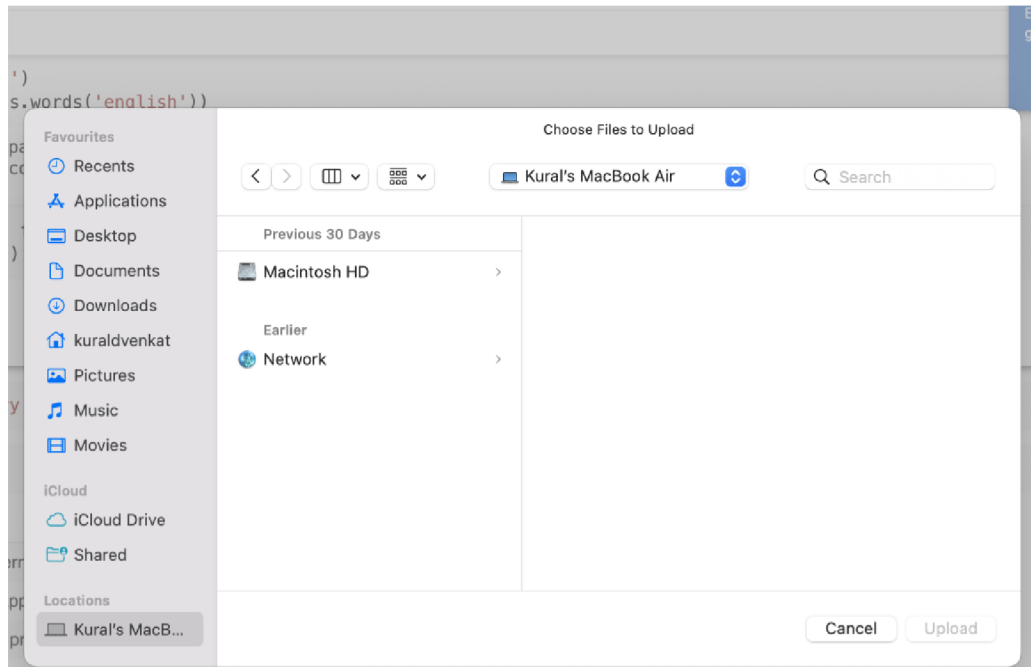


Figure 11. Files upload function in Google Colab. [own work]

```
query.xlsx(application/vnd.openxmlformats-officedocument.spreadsheetml.sheet) -
```

15877 bytes, last modified: n/a - 100% done.

```
Saving query.xlsx to query.xlsx
```

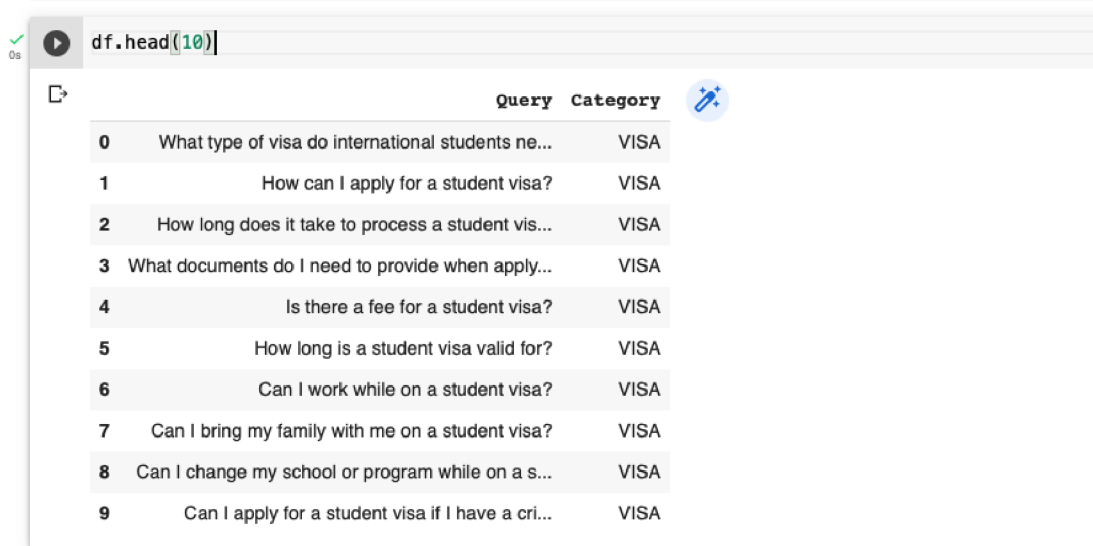
This will add “query.xlsx” to Google Colab’s environment base directory.

```
df = pd.read_excel('query.xlsx')
```

To load the selected xlsx file as Data Frame, Pandas has a function called **read_excel()**.

Pandas support various file formats such as CSV, Excel, SQL, JSON, etc. making the library versatile.

Now the variable “df” is a data frame of Queries and Categories. Hereafter, this variable will be used to view data or to perform various descriptive statistics.



```
df.head(10)
```

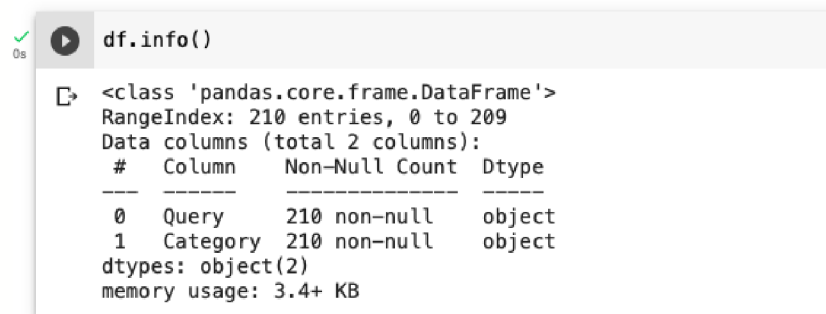
	Query	Category
0	What type of visa do international students ne...	VISA
1	How can I apply for a student visa?	VISA
2	How long does it take to process a student vis...	VISA
3	What documents do I need to provide when apply...	VISA
4	Is there a fee for a student visa?	VISA
5	How long is a student visa valid for?	VISA
6	Can I work while on a student visa?	VISA
7	Can I bring my family with me on a student visa?	VISA
8	Can I change my school or program while on a s...	VISA
9	Can I apply for a student visa if I have a cri...	VISA

Figure 12. Top 10 rows of the dataset. [own work]

The **df.head()** method will accept integer parameters and display the data up to the given index. In this case, it displays the first ten rows of the dataframe.

It is important to check missing values in any dataset and if there are any missing values “Imputation” can be performed on the null values, since this dataset is manually created using surveys and chatGPT, it does not contain null values.

`df.info()` is used to view basic information of the dataset.



```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 210 entries, 0 to 209
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ---
 0   Query      210 non-null   object
 1   Category   210 non-null   object
dtypes: object(2)
memory usage: 3.4+ KB
```

Figure 13. Basic information about the data. [own work]

To check the descriptive statistics “describe” function can be called, since the dataset has no numerical column, there are not many statistics descriptions to be analyzed.



```
df.describe()
```

	Query	Category
count	210	210
unique	210	14
top	What type of visa do international students ne...	Test center
freq	1	51

Figure 14. Statistical information about the data. [own work]

The above image explains the number of unique entities in both Query and Category Columns. There are no duplicate entries in Query. And there are a total of 14 unique categories.

To view all the unique categories, the “value_count” function should be called in the Category column.

```
df.Category.value_counts()
```



```
df.Category.value_counts()
Test center          51
admission           21
study management    17
technical support   16
lockers             13
accomodation        11
menza               11
VISA                10
library             10
sports              10
lost and found      10
emergency           10
activities and events 10
laundry             10
Name: Category, dtype: int64
```

Figure 15. Categories and their count. [own work]

From the above image, it can be inferred that the dataset is imbalanced with the “Test center” category with a count of 51 and some other categories count as meager as 10.

4.4.1 Data Preprocessing

After the initial inference and analysis, it is concluded that the data frame does not require any more imputation or alteration. Data standardization and normalization do not apply to this approach as there is no numerical column.

Since the model only understands numerical values, the subsequent step to be done is pre-processing the data and converting the query sentences and their categories into numerical data by using standard procedures.

```
✓ 0s df[['Query']]
0    [what, type, of, visa, do, international, stud...
1          [how, can, apply, for, student, visa]
2    [how, long, does, it, take, to, process, stude...
3    [what, documents, do, need, to, provide, when,...
4          [is, there, fee, for, student, visa]
      ...
205          [where, can, rent, presenter]
206 [is, it, possible, to, borrow, laptop, for, te...
207 [how, to, turn, on, the, air, conditioning, in...
208 [how, to, turn, off, the, air, conditioning, i...
209 [who, takes, care, of, the, temperature, in, t...
Name: Query, Length: 210, dtype: object
```

Figure 16. Queries as an array of tokens. [own work]

Since the computer system is sensitive to the case of the alphabet, this creates ambiguity in identifying the same words with different cases as the same word with the same meaning. To avoid this issue all the letters must be written in lowercase.

All the punctuations and special characters will be removed from the corpus as it does not affect the intent of the query.

Stopwords are common words that appear frequently in natural language text, such as "the", "and", "of", "to", "in", etc. These words do not carry much meaning and can often be removed without significantly affecting the overall understanding of the text.

One of the main reasons for removing stopwords in data modelling is to reduce the dimensionality of the data. By removing these common words, the overall vocabulary size is reduced, which can make the analysis more efficient and effective. This is particularly important for models that rely on counting or frequency-based methods, such as bag-of-words models, where the presence or absence of certain words can affect the outcome.

Removing stopwords can also help improve the quality of the analysis by reducing noise and increasing the signal-to-noise ratio. By eliminating words that are unlikely to be relevant to the analysis, we can focus on the words that are more likely to carry important information.

Overall, removing stopwords is a common preprocessing step in natural language processing and can help improve the efficiency and effectiveness of data modelling.

For all the above-mentioned preprocessing procedures Gensim would be the best choice as it is an open-source library for unsupervised topic modeling, document indexing, retrieval by similarity, and other natural language processing functionalities.

The **gensim.parsing.preprocessing** module provides a set of functions for preprocessing text data in natural language processing applications. Here are some of the most used functions in this module:

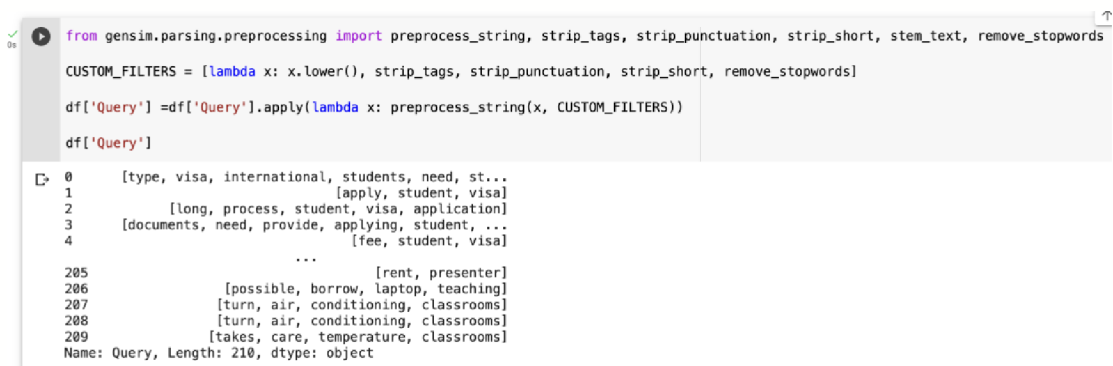
1. **strip_tags(text):** Removes HTML tags from the given text.
2. **strip_punctuation(text):** Removes punctuation marks from the given text.
3. **strip_multiple_whitespaces(text):** Replaces multiple whitespace characters with a single whitespace in the given text.
4. **strip_numeric(text):** Removes digits from the given text.
5. **remove_stopwords(text):** Removes stop words (commonly occurring words such as "the", "and", "in", etc.) from the given text.
6. **stem_text(text):** Stems the words in the given text using Porter stemming algorithm.
7. **preprocess_string(text):** This applies a series of preprocessing steps, including converting the text to lowercase, removing tags, punctuation, and digits, and removing stop words.

```
from gensim.parsing.preprocessing import preprocess_string,  
strip_punctuation, strip_short, remove_stopwords
```

```
CUSTOM_FILTERS = [lambda x: x.lower(), strip_punctuation,
strip_short, remove_stopwords]
```

```
df['Query'] =df['Query'].apply(lambda x:
preprocess_string(x, CUSTOM_FILTERS))
```

The above lines will preprocess the corpus so that there won't be any punctuation, stop words, and white spaces.



```
from gensim.parsing.preprocessing import preprocess_string, strip_tags, strip_punctuation, strip_short, stem_text, remove_stopwords

CUSTOM_FILTERS = [lambda x: x.lower(), strip_tags, strip_punctuation, strip_short, remove_stopwords]

df['Query'] =df['Query'].apply(lambda x: preprocess_string(x, CUSTOM_FILTERS))

df['Query']
```

0	[type, visa, international, students, need, st...
1	[apply, student, visa]
2	[long, process, student, visa, application]
3	[documents, need, provide, applying, student, ...]
4	[fee, student, visa]
...	...
205	[rent, presenter]
206	[possible, borrow, laptop, teaching]
207	[turn, air, conditioning, classrooms]
208	[turn, air, conditioning, classrooms]
209	[takes, care, temperature, classrooms]

Name: Query, Length: 210, dtype: object

Figure 17. Implementation of Gensim's Preprocessing function. [own work]

The next step is to encode the Categories, As mentioned, most machine learning or deep learning algorithms require numerical input, so categorical data must be encoded to be used in these models. Encoding ensures that the categorical data is represented in a consistent and standardized format.

These variables can have many categories, which can result in many variables in the model. Encoding these variables can reduce the dimensionality of the data and make the model more efficient.

Categories can have an arbitrary order, meaning that assigning numerical values to categories based on arbitrary criteria can introduce unintended biases into the model.

Categorical variables can be classified into two types: ordinal and nominal.

Ordinal variables are categorical variables where the categories have a natural ordering or hierarchy. On the other hand, nominal variables are categorical variables where the categories do not have a natural ordering.

The distinction between ordinal and nominal variables is important because different encoding methods are used for each type of variable. For ordinal variables, we can use label encoding or ordinal encoding to encode the categories numerically, whereas, for nominal variables, we typically use one hot encoding or binary encoding.

Since our categories are multiclass and do not have any natural ordering or hierarchy the most suitable would be one hot encoding.

```
0s ✓ ▶ Y = pd.get_dummies(df['Category']).values
print(Y)

[[0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]
 [0 0 0 ... 0 0 1]]
```

Figure 18. Transforming categories to one hot encoding. [own work]

Subsequently, it is crucial to create the word index or vocabulary that is used to convert the text data into a sequence of integers.

The **Keras.preprocessing.text.Tokenizer** class is a tool in the Keras library that is used to vectorize a corpus of text into a matrix of token counts. It essentially transforms a list of strings into a matrix of integers, where each integer represents the count of a particular word in the text corpus.

The Tokenizer class provides several useful methods for preparing text data for models, including:

1. **fit_on_texts(texts):** Updates the internal vocabulary based on a list of texts. This method creates a dictionary of words and their respective counts in the text corpus. This method is necessary because it creates the word index or vocabulary that is used to convert the text data into a sequence of integers. Without this step, it would not be possible to use text data as input to a model. Additionally, the fit_on_texts() method ensures that each word is represented by a unique integer index, which is essential for accurate text analysis and classification
2. **texts_to_sequences(texts):** Transforms a list of texts into a list of sequences of integers. Each integer corresponds to a word in the dictionary created by the fit_on_texts method.
3. **texts_to_matrix(texts, mode):** Transforms a list of texts into a matrix, where each row represents a text and each column represents a word in the dictionary created by fit_on_texts method. The value in each cell represents the count of the corresponding word in the text, and the mode parameter specifies the type of matrix to be returned (e.g., binary, count, tfidf, etc.).

```
[33] MAX_NB_WORDS = 1000
      MAX_SEQUENCE_LENGTH = 32
      EMBEDDING_DIM = 100

      tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\\]^_`{|}~', lower=True)
      tokenizer.fit_on_texts(df['Query'].values)
      word_index = tokenizer.word_index
      print('Found %s unique tokens.' % len(word_index))

Found 335 unique tokens.
```

Figure 19. Tokenizing the queries. [own work]

With the above code, arrays of word tokens are transformed into arrays of the integer sequence.

The number of Words (MAX_NB_WORDS) for this model is set to 1000 as the corpus has only 335 unique tokens. This parameter specifies the maximum number of words to keep in the vocabulary.

MAX_SEQUENCE_LENGTH is set to 32 as the corpus does not contain long sentences, this would be the input dimension in the embedding layer while building the model.

Words that are not included in the vocabulary will be treated as out-of-vocabulary (OOV) words. This parameter is set to None by default, which means that all words will be included in the vocabulary.

```
X = tokenizer.texts_to_sequences(df['Query'].values)
print("Before Padding: ", X)

X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print("\nAfter Padding: ", X)

print('\nShape of data tensor:', X.shape)

Before Padding: [[121, 8, 47, 6, 14, 27, 122, 123], [15, 5, 8], [28, 21, 5, 8, 16], [124, 14, 125, 48, 5, 8], [49, 5, 8],
After Padding: [[ 0  0  0 ... 27 122 123]
[ 0  0  0 ... 15  5  8]
[ 0  0  0 ...  5  8 16]
...
[ 0  0  0 ... 69 70 36]
[ 0  0  0 ... 69 70 36]
[ 0  0  0 ... 74 335 36]]

Shape of data tensor: (210, 32)
```

Figure 20. Padding the tokens to uniform the length. [own work]

The **pad_sequences** function is used to transform a list of sequences into a 2D Numpy array of a specified length by adding padding or truncating the sequences.

It is important to split the data into train and test sets because it helps us to evaluate the performance of our model on unseen data.

The primary purpose of a model is to make accurate predictions on new, unseen data. If we train our model on the entire dataset and then test it on the same data, it will likely overfit the data, which means it will memorize the patterns in the training data instead of generalizing to new data. This will result in poor performance when it encounters new data, even if it performs well on the training data.

By splitting the data into a training set and a test set, we can train the model on the training set and evaluate its performance on the test set. This allows us to estimate how well the model will perform on new, unseen data. If the model performs well on the test set, we can

have more confidence that it will generalize well to new data. If the model performs poorly on the test set, we can adjust our model or choose a different model altogether before deploying it to make predictions on new data.

Overall, splitting a dataset into a training set and a test set is a critical step in machine learning that allows us to evaluate the performance of our model on unseen data and ensure that it can generalize well to new data.

To do that, the *train_test_split* function is used, it splits the dataset into a training set and a test set. The parameters of the **train_test_split** function is:

train_test_split(*arrays, **options)

1. ***arrays**: The first parameter is one or more arrays that we want to split into a training set and a test set. Typically, this will be an array of features and an array of corresponding labels.
2. **test_size**: This parameter determines the size of the test set. It can be a float (between 0.0 and 1.0) which represents the proportion of the dataset to include in the test set, or an integer which represents the absolute number of samples to include in the test set. By default, it is set to 0.25, meaning that 25% of the dataset will be used for testing.
3. **train_size**: This parameter determines the size of the training set. It can be a float (between 0.0 and 1.0) which represents the proportion of the dataset to include in the training set, or an integer which represents the absolute number of samples to include in the training set. By default, it is set to *None*, which means that the complement of the test set will be used for training.
4. **random_state**: This parameter sets the random seed for the data shuffling, which ensures that the data is split in the same way every time the function is called. This can be an integer or a random state object. By default, it is set to *None*.

The test size is set to 0.2% which means 20 percent of the total dataset will be used for testing, which is 42 rows out of 210 rows.

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.20, random_state = 42, shuffle=True)
print(X_train.shape, Y_train.shape)
print(X_test.shape, Y_test.shape)

(168, 32) (168, 14)
(42, 32) (42, 14)
```

Figure 21. Splitting the dataset into train and test data. [own work]

4.5 Model Building

4.5.1 Building LSTM Network

To implement the LSTM network Keras is a popular choice, It has a user-friendly deep learning library that provides a high-level interface for building and training neural networks. It allows the building of a wide range of neural network architectures, including LSTMs, with customizable layers and activation functions. So, it's easy to combine different types of layers to create more complex models.

Keras is built on top of TensorFlow, a high-performance deep-learning library. It has the advantage of TensorFlow's efficient computation and parallelization capabilities to train LSTMs faster.

```
[168] model = Sequential()
      model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
      model.add(SpatialDropout1D(0.2))
      model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
      model.add(Dense(14, activation='softmax'))
      model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
      print(model.summary())
```

```
Model: "sequential_31"
-----
```

Layer (type)	Output Shape	Param #
embedding_31 (Embedding)	(None, 32, 100)	100000
spatial_dropout1d_32 (SpatialDropout1D)	(None, 32, 100)	0
lstm_33 (LSTM)	(None, 100)	80400
dense_28 (Dense)	(None, 14)	1414

```
-----
Total params: 181,814
Trainable params: 181,814
Non-trainable params: 0
-----
None
```

Figure 22. Building LSTM model. [own work]

The Sequential model is a linear stack of layers that allows you to easily build and train neural networks. It is created by instantiating the Sequential class and can add layers to the model using the `.add()` method.

In this approach, there will be 4 layers:

a. Embedding

This layer is used to convert integer-encoded input sequences into dense vectors, which can be used as input to a neural network model.

A sequential model is created and an Embedding layer is added to it. The **input_dim** parameter specifies the size of the vocabulary, i.e., the maximum integer index that can be used as a word index, In this case, `input_dim = 1000`.

The **output_dim** parameter specifies the dimensionality of the dense embedding vector, which will be 100.

The **input_length** parameter specifies the length of the input sequences(`X.shape[1]`), which is set to 32 while converting texts to sequence.

When the model is trained, the Embedding layer learns an embedding for each word in the vocabulary, mapping each word to a dense vector of `output_dim` dimensions. During training, the weights of the embedding layer are updated based on the error between the predicted outputs and the true outputs.

The output of the embedding layer is a 3D tensor with shape **(batch_size, input_length, output_dim)**, which can be fed into other layers in the model for further processing.

b. Spatial Dropout

The **SpatialDropout1D** layer is a type of dropout regularization to prevent overfitting in the model by randomly dropping out (setting to zero) some of the feature maps during training.

This layer randomly sets the input feature maps to zero with a certain probability (dropout rate) for each channel, which helps to prevent the model from relying too heavily on any particular set of features. This helps the model to generalize better to new, unseen data.

In this case, the dropout percentage is set to 20% because the size of the input is very less compared to traditional modelling. So, it's important to use as much information from the input data.

c. LSTM

The LSTM layer is designed to address the issue of vanishing gradients that can occur in traditional RNNs. The LSTM layer includes a memory cell that can retain information over a long period of time and selectively forget or remember specific pieces of information as needed.

In this model, LSTM layer has been added with 100 units, which takes as input a 3D tensor with shape **(batch_size, timesteps, input_dim)**. It can also have a dropout and recurrent dropout function, here it has been set to 0.2 which is 20%.

d. Dense

After the LSTM layer, the Dense layer should be with **num_classes** units and a **softmax** activation function, which is commonly used for classification tasks.

The purpose of a dense layer is to transform the input data into a format that is suitable for the final output layer of the network. Each neuron in the dense layer transforms the input data linearly, followed by a non-linear activation function, such as ReLU or sigmoid.

The softmax activation function is commonly used for classification tasks, where the output of the network represents the probability distribution over the possible classes.

The parameter `num_classes` should be 14 units as the data has 14 different categories, this means the output layer will have 14 neurons each neuron representing each category of the query.

4.5.2 Compiling the LSTM model

The **compile** method is used to configure the learning process of a deep learning model before training. This method specifies the loss function, the optimizer, and the metrics to use during training and evaluation.

The loss function is specified as '**categorical_crossentropy**' for the model, which is commonly used for multiclass classification tasks.

The optimizer is specified as '**adam**', which is a popular optimization algorithm that adapts the learning rate during training.

Finally, '**accuracy**' has been specified as the metric to use during training and evaluation.

4.5.3 Fitting the LSTM model

The model needs to be fitted with the training data using the 'fit' function. This method takes as input the training data, as well as various hyperparameters that control the learning process, such as the number of epochs, the batch size, and the optimizer.

To train the model using the **fit** method, training data (**X_train** and **Y_train**), as well as the batch size, the number of epochs, and the validation data (**X_test** and **Y_test**) should be passed.

During training, the model will update its weights based on the gradient of the loss function with respect to the weights, using the optimizer specified in the **compile** method.

The metrics specified in the **compile** method will be computed and reported after each epoch of training.

```
import datetime
import tensorflow as tf

epochs = 20
batch_size = 15

log_dir = "logs/fit/" +
datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback =
tf.keras.callbacks.TensorBoard(log_dir=log_dir,
histogram_freq=1)

history = model.fit(X_train, Y_train, epochs=epochs,
batch_size=batch_size, validation_split=0.2,
                    callbacks=[tensorboard_callback,
EarlyStopping(monitor='val_loss', patience=3,
min_delta=0.0001)])
```

Epochs were set to 20 and batch size is set to 15. These optimal values for batch size and epochs were determined through a process of experimentation involving various alternative values.

```
import datetime
import tensorflow as tf

epochs = 20
batch_size = 20

log_dir = "logs/fit/" + datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=log_dir, histogram_freq=1)

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2,
                    callbacks=[tensorboard_callback, EarlyStopping(monitor='val_loss', patience=3, min_delta=0.0001)])
```

Epoch 1/20
7/7 [=====] - 7s 325ms/step - loss: 2.6217 - accuracy: 0.1716 - val_loss: 2.5980 - val_accuracy: 0.2059
Epoch 2/20
7/7 [=====] - 3s 419ms/step - loss: 2.5350 - accuracy: 0.2463 - val_loss: 2.5568 - val_accuracy: 0.2059
Epoch 3/20
7/7 [=====] - 2s 234ms/step - loss: 2.4351 - accuracy: 0.2463 - val_loss: 2.5387 - val_accuracy: 0.2059
Epoch 4/20
7/7 [=====] - 2s 237ms/step - loss: 2.4094 - accuracy: 0.2463 - val_loss: 2.5103 - val_accuracy: 0.2059
Epoch 5/20
7/7 [=====] - 2s 245ms/step - loss: 2.3765 - accuracy: 0.2463 - val_loss: 2.4986 - val_accuracy: 0.2059
Epoch 6/20
7/7 [=====] - 2s 236ms/step - loss: 2.3397 - accuracy: 0.2463 - val_loss: 2.4467 - val_accuracy: 0.2059
Epoch 7/20
7/7 [=====] - 2s 239ms/step - loss: 2.2676 - accuracy: 0.2463 - val_loss: 2.4003 - val_accuracy: 0.2059
Epoch 8/20
7/7 [=====] - 2s 238ms/step - loss: 2.1963 - accuracy: 0.2463 - val_loss: 2.3621 - val_accuracy: 0.2059
Epoch 9/20
7/7 [=====] - 3s 433ms/step - loss: 2.1005 - accuracy: 0.2910 - val_loss: 2.2657 - val_accuracy: 0.2353
Epoch 10/20
7/7 [=====] - 2s 240ms/step - loss: 2.0033 - accuracy: 0.3209 - val_loss: 2.1520 - val_accuracy: 0.2647

Figure 23. Experimenting with epochs and batch_value for finding optimal value. [own work]

a. Callbacks

i. TensorBoard

TensorBoard is a powerful visualization tool that can be used to monitor and debug deep learning models in real-time.

TensorBoard callback has been initialized, specifying the directory where the logs should be written. During training, the callback will automatically write logs to this directory, including information about the training loss, validation loss, and other metrics.

ii. EarlyStopping

The EarlyStopping callback is a Keras callback that can be used to automatically stop training when a monitored metric has stopped improving. This can be useful to prevent overfitting and to save time during training.

On creating an **EarlyStopping** callback, specify the monitored metric as **'val_loss'**, which means the validation loss will be monitored during training.

The **min_delta** is set to **0.001**, which will only consider a change in the metric as an improvement if it is greater than **0.001**.

And **patience** is set to **3**, which means that the model will wait for 5 epochs before stopping training if the monitored metric has not improved. Training will be stopped early if the monitored metric (**val_loss**) does not improve for 3 consecutive epochs.

```
history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size, validation_split=0.2,
                    callbacks=[tensorboard_callback, EarlyStopping(monitor='val_loss', patience=3, min_delta=0.001)])
```

```
Epoch 1/20
9/9 [=====] - 6s 164ms/step - loss: 2.6227 - accuracy: 0.1866 - val_loss: 2.6062 - val_accuracy: 0.2059
Epoch 2/20
9/9 [=====] - 1s 94ms/step - loss: 2.4976 - accuracy: 0.2463 - val_loss: 2.5757 - val_accuracy: 0.2059
Epoch 3/20
9/9 [=====] - 1s 94ms/step - loss: 2.4503 - accuracy: 0.2463 - val_loss: 2.5154 - val_accuracy: 0.2059
Epoch 4/20
9/9 [=====] - 1s 88ms/step - loss: 2.3984 - accuracy: 0.2463 - val_loss: 2.4920 - val_accuracy: 0.2059
Epoch 5/20
9/9 [=====] - 1s 90ms/step - loss: 2.3318 - accuracy: 0.2463 - val_loss: 2.4948 - val_accuracy: 0.2059
Epoch 6/20
9/9 [=====] - 1s 148ms/step - loss: 2.2690 - accuracy: 0.2463 - val_loss: 2.3799 - val_accuracy: 0.2059
Epoch 7/20
9/9 [=====] - 1s 163ms/step - loss: 2.1642 - accuracy: 0.2985 - val_loss: 2.2893 - val_accuracy: 0.2353
Epoch 8/20
9/9 [=====] - 1s 110ms/step - loss: 2.0131 - accuracy: 0.3284 - val_loss: 2.1912 - val_accuracy: 0.2353
Epoch 9/20
9/9 [=====] - 1s 87ms/step - loss: 1.8914 - accuracy: 0.3881 - val_loss: 2.1261 - val_accuracy: 0.2941
Epoch 10/20
9/9 [=====] - 1s 91ms/step - loss: 1.7402 - accuracy: 0.4328 - val_loss: 2.0678 - val_accuracy: 0.3235
Epoch 11/20
9/9 [=====] - 1s 90ms/step - loss: 1.6062 - accuracy: 0.4925 - val_loss: 1.9775 - val_accuracy: 0.3529
Epoch 12/20
9/9 [=====] - 1s 90ms/step - loss: 1.4861 - accuracy: 0.6716 - val_loss: 1.9140 - val_accuracy: 0.4412
Epoch 13/20
9/9 [=====] - 1s 90ms/step - loss: 1.3005 - accuracy: 0.7015 - val_loss: 1.8029 - val_accuracy: 0.4412
Epoch 14/20
9/9 [=====] - 1s 87ms/step - loss: 1.1620 - accuracy: 0.7463 - val_loss: 1.6569 - val_accuracy: 0.5294
Epoch 15/20
9/9 [=====] - 1s 89ms/step - loss: 0.9582 - accuracy: 0.7985 - val_loss: 1.5608 - val_accuracy: 0.6176
Epoch 16/20
9/9 [=====] - 1s 91ms/step - loss: 0.7882 - accuracy: 0.8433 - val_loss: 1.4354 - val_accuracy: 0.6176
Epoch 17/20
9/9 [=====] - 1s 96ms/step - loss: 0.6553 - accuracy: 0.9030 - val_loss: 1.3307 - val_accuracy: 0.7059
Epoch 18/20
9/9 [=====] - 1s 92ms/step - loss: 0.5359 - accuracy: 0.9478 - val_loss: 1.1821 - val_accuracy: 0.6765
Epoch 19/20
9/9 [=====] - 1s 90ms/step - loss: 0.4726 - accuracy: 0.9403 - val_loss: 1.1922 - val_accuracy: 0.6765
Epoch 20/20
9/9 [=====] - 1s 105ms/step - loss: 0.3601 - accuracy: 0.9627 - val_loss: 1.0592 - val_accuracy: 0.7353
```

Figure 24. Training of the model. [own work]

4.5.4 Evaluation of the LSTM model

After training a model, it is important to evaluate its performance on a separate test set to get an estimate of how well it can generalize to new data.

In Keras, the *evaluate* method can be used to evaluate the performance of a trained model on a test set. The *evaluate* method returns the values of the specified metrics on the test set.

```

✓ [31] accr = model.evaluate(X_test,Y_test)
0s print('Test set\n Loss: {:.3f}\n Accuracy: {:.3f}'.format(accr[0],accr[1]))

2/2 [=====] - 0s 15ms/step - loss: 0.6667 - accuracy: 0.8810
Test set
Loss: 0.667
Accuracy: 0.881

```

Figure 25. Evaluation of the trained model with test set. [own work]

After evaluating the model on a separate test set using the evaluate method, passing in the test data (X_test and Y_test). The evaluate method returns the values of the specified metrics (in this case, the loss and accuracy) on the test set.

```

2/2 [=====] - 0s 15ms/step - loss:
0.6667 - accuracy: 0.8810
Test set
Loss: 0.667
Accuracy: 0.881

```

The LSTM model built using the input data and the above-mentioned approach and parameter resulted in a loss of 0.8810 and 88% of accuracy on unseen data.

4.5.5 Plotting the LSTM model

The below is graphs, plotted using TensorBoard for epoch vs accuracy and epoch vs loss, it can be inferred that accuracy is linearly improving in each epoch. While the loss is decreasing on each epoch.

This linearity is because of EarlyStopping callback, usually, once the model reaches the optimum epoch, it will start to overfit and results in a parabolic curve on accuracy and loss.

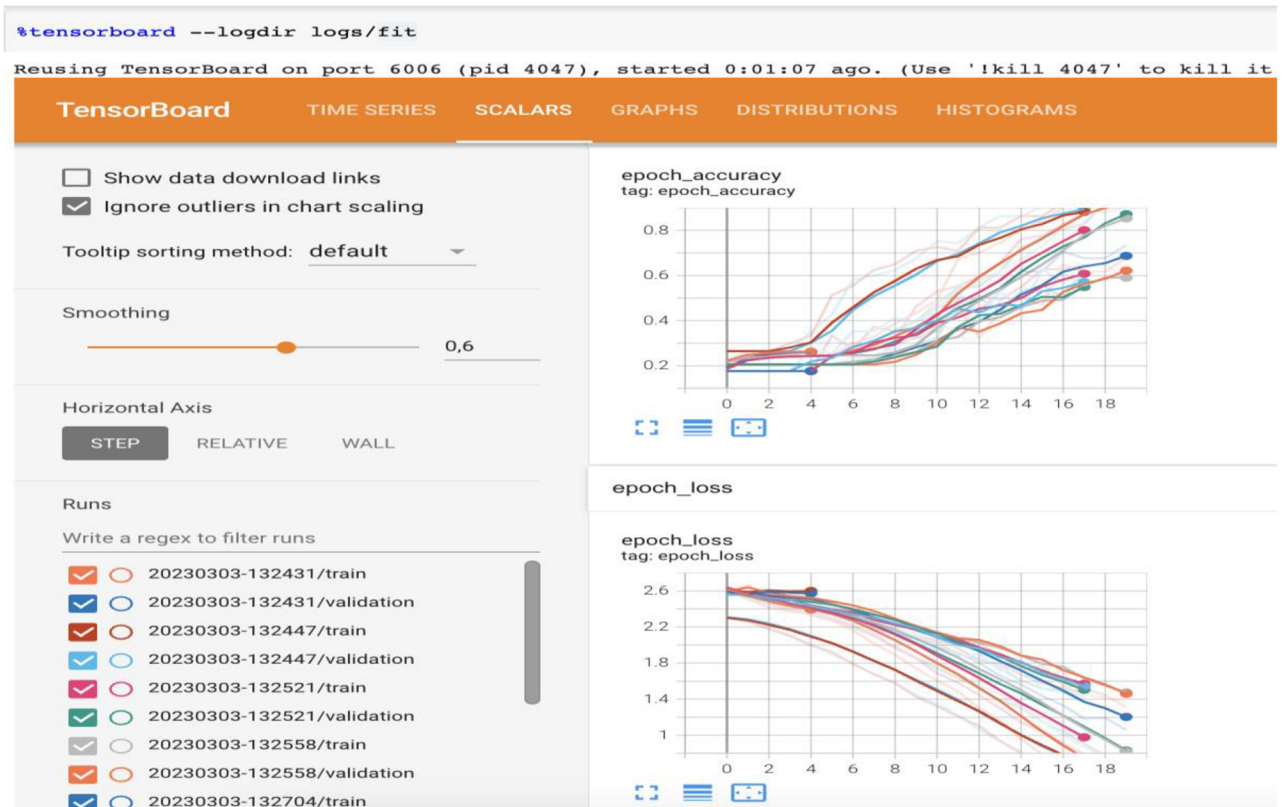


Figure 26. Tensorboard plotting of LSTM model training. [own work]

4.6 Comparative evaluation of the accuracy between IBM Watson and LSTM

To compare the efficiency of the LSTM model with a commercial pre-trained model, IBM Watson has been chosen.

IBM Watson is an exceptional platform that employs cutting-edge natural language processing (NLP) techniques to effectively classify massive amounts of textual data. Its sophisticated machine learning algorithms enable it to accurately identify and categorize diverse text data, including social media posts, news articles, and customer feedback, among others.

Watson's text classification abilities are fueled by its advanced deep learning models, which are expertly trained on extensive data sets to recognize intricate patterns and correlations in text. This empowers Watson to offer businesses valuable insights into customer behavior, sentiment analysis, and other critical metrics that can be utilized to enhance their products and services.

IBM Watson is an invaluable resource for any organization that desires to glean deeper insights into its text data and make informed decisions based on its findings.

Account in the IBM cloud is set up, Watson Assistant Service resource is created, and the virtual assistant instance is created including a dialog skill that is linked with the virtual assistant instance.

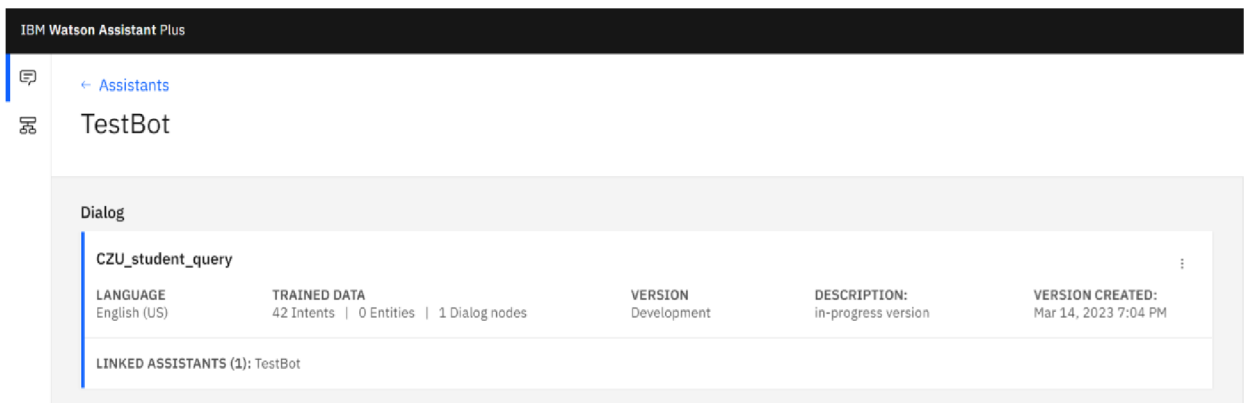


Figure 27. IBM Watson setup. [own work]

The dataset is shuffled and splits as train and test datasets. Similar to the LSTM model, 80% of the data is taken for training and 20% for testing.

```
[ 4] cross_1 = shuffle(df)
      cross_2 = shuffle(df)
      cross_3 = shuffle(df)
      cross_4 = shuffle(df)
      cross_5 = shuffle(df)
```

```
cf = df['Category'].value_counts().reset_index()
cf = cf.rename(columns={'Category': 'Count', 'index': 'Category'})
print(cf.shape)
cf.head(2)
```

(14, 2)

	Category	Count
0	Test center	51
1	admission	21

Figure 28. Shuffling and data pre-processing for IBM Watson. [own work]

```

0s ▶ train = pd.DataFrame()
test = pd.DataFrame()
for i in range(cf.shape[0]):
    split_number = int(round(cf['Count'][i]*0.8))
    category_examples = cross_1[cross_1['Category'].str.match(cf['Category'][i]).reset_index()]
    counter = 0
    for index, row in category_examples.iterrows():
        if counter <= split_number:
            train = train.append(row)
        else:
            test = test.append(row)
        counter = counter + 1

```

Figure 29. Splitting the data into train and test datasets. [own work]

Since the chosen environment is Google's colab, it does not come with a pre-installed `ibm_watson` library. So, it is necessary to install it in colab's runtime environment.

```

[ ] !pip install ibm-watson

```

```

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: ibm-watson in /usr/local/lib/python3.9/dist-packages (7.0.0)
Requirement already satisfied: python-dateutil>=2.5.3 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: ibm-cloud-sdk-core==3.*,>=3.3.6 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: requests<3.0,>=2.0 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: websocket-client==1.1.0 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: PyJWT<3.0.0,>=2.4.0 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: urllib3<2.0.0,>=1.26.0 in /usr/local/lib/python3.9/dist-packages (from ibm-watson)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.9/dist-packages (from python-dateutil)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/dist-packages (from requests)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.9/dist-packages (from requests)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-packages (from requests)

```

Figure 30. Installing `ibm_watson` library to Google Colab environment. [own work]

And it is crucial to import all the necessary libraries to make an API call to IBM Watson's service and consume the result.

HTTP Basic Authentication is a straightforward method to secure web pages, APIs, and other resources. In IBM Watson, an API key is used for authentication.

The client resends the request with the username and password encoded in the "Authorization" header. However, there are security limitations to HTTP Basic Authentication. The credentials are sent in plain text, making them vulnerable to interception. Additionally, there is no way to log out of a session once authenticated, other than closing the browser or terminating the API client.

```

✓ [11] import pandas as pd
      import requests
      import json
      from requests.auth import HTTPBasicAuth
      from ibm_watson import AssistantV2
      from ibm_cloud_sdk_core.authenticators import IAMAuthenticator
      from ibm_watson import ApiException

```

Figure 31. Importing necessary libraries for IBM Watson API. [own work]

```

[10] API_KEY = ""
      SERVICE_URL = "https://api.eu-de.assistant.watson.cloud.ibm.com"
      API_APP_URL = "https://api.eu-de.assistant.watson.cloud.ibm.com/instances/6833047d-680e-4bd1-ace3-2a9e87387e14"
      API_ASSISTANT_ID = "bfe2fe18-c68e-4f54-9891-7bbcc978c76a"
      API_VERSION = "2021-11-27"

```

Figure 32. Parameters for IBM Watson API call. [own work]

The next step is to set up IBM Watson's AssistantV2, IBM Watson Assistant is an IBM cloud-based chatbot and virtual assistant platform. It enables developers to create and deploy conversational interfaces for a variety of platforms, including websites, messaging applications, mobile devices, and smart speakers.

The most recent version of IBM Watson Assistant is v2, which represents a major improvement over the previous version (formerly known as Watson Conversation). The below code will create a session with IBM Watson's assistant service with the supplied parameters.

```

✓ [13] authenticator = IAMAuthenticator(API_KEY)
1s      assistant = AssistantV2(
          version=API_VERSION,
          authenticator = authenticator
        )

      assistant.set_service_url(SERVICE_URL)
      try:
          response = assistant.create_session(
              assistant_id=API_ASSISTANT_ID
          ).get_result()
      except ApiException as ex:
          print("Method failed with status code " + str(ex.code) + ": " + ex.message)

      session_id = json.dumps(response['session_id'])
      print(session_id)

"79258e67-babf-438c-8266-8ec48f7f204d"

```

Figure 33. Creating session with IBM Watson Assistant. [own work]

Once the session is set up, the assistant is now ready to receive the Queries via API and will respond with the predicted Category. The below code is written to get the original query, predicted intent, and confidence as the response from the Assistant service.

```

▶ example_count = test.shape[0]
correctly_detected = 0
incorrects = []

for i in range(example_count):
    query = test['Query'][i]
    ground_true = test['Category'][i].replace(' ', '_')
    print(query, ground_true)
    try:
        response = assistant.message(
            assistant_id=API_ASSISTANT_ID,
            session_id=session_id.strip('\"'),
            input={
                'message_type': 'text',
                'text': query
            }
        ).get_result()

        answer = json.dumps(response['output']['generic'][0]['text'])
        if len(response["output"]["intents"]) > 0:
            intent = json.dumps(response['output']['intents'][0]['intent'])
            confidence = json.dumps(response['output']['intents'][0]['confidence'])
            print(intent, confidence)
            if(str(intent).strip('\"') == ground_true):
                correctly_detected = correctly_detected + 1
            else:
                incorrects.append(str(query + ',' + intent + ',' + ground_true))
        else:
            incorrects.append(str(query + ',[]' + ',' + ground_true))
    except ApiException as ex:
        print("Method failed with status code " + str(ex.code) + ": " + ex.message)

```

Figure 34. Predicting categories for queries with IBM Watson via API. [own work]

```

☞ I missed my deadline at the test center, am I missing my shot? Test_center
"Test_center" 1
How can I cancel my reservation in advance? Test_center
"Test_center" 1
Can I bring some supplies with me? Test_center
"Test_center" 1
Can the calculator be used in the test centre? Test_center
"Test_center" 1
Will I get paper and pencil for the test at the test centre? Test_center
"Test_center" 1
Where can I go to take the test? Test_center
"Test_center" 0.9349506497383118
Can I get some samples to test at the test centre? Test_center
"Test_center" 1
I missed a test date, I failed my test? Test_center
"Test_center" 1

```

Figure 35. Response received from IBM Watson Assistant service with input query, its predicted intent, and confidence score.

[own work]

4.6.1 Evaluation of intent classification with IBM Watson

From the IBM Watson service, the predicted intents and their confidence score are returned as shown in Figure.34

```
▶ print('Number of test examples: ', example_count)
print('Correctly detected: ', correctly_detected)
print('Incorrectly detected: ', example_count - correctly_detected)
print('Accuracy: ', correctly_detected / example_count)
```

```
Number of test examples: 27
Correctly detected: 22
Incorrectly detected: 5
Accuracy: 0.8148148148148148
```

Figure 36. Printing the metrics for evaluation. [own words]

Based on the results of predicting the intent for the queries in the test dataset, it was found that 22 out of 27 queries were correctly classified with their respective intent, resulting in 5 incorrectly classified intents. The Accuracy metric was calculated using the formula "correctly_detected / example_count," yielding a value of 0.8148 (i.e., 81.48%).

5. Result

The performance of two deep learning models for intent classification, LSTM, and IBM Watson, was evaluated on a dataset of student queries. Both models were trained and tested using the same queries and categories, and their performance was compared based on accuracy.

The evaluation results showed that the LSTM outperformed IBM Watson with an accuracy of 88.1% compared to IBM Watson's accuracy of 81.4%. This indicates that the LSTM is better at predicting in our case of specific training sets for students' queries than IBM Watson.

6. Conclusion

In conclusion, the comparison of the LSTM and IBM Watson demonstrated that the LSTM trained using a dataset of 210 rows is better in terms of accuracy for predicting the intent of students' queries. This suggests that the LSTM will be the better choice to implement in the university chatbot.

LSTM model was trained with custom architecture and proper use of Dropouts. The model is further optimized and hyperparameters were tuned to increase the accuracy. And the overfitting problem was prevented by early stopping. These factors could be the contributing reasons for better accuracy.

It's worth noting that while accuracy is an important metric, there may be other factors to consider when selecting a deep learning model for a specific application. For example, IBM Watson may have lower accuracy but could be faster or require fewer computational resources, making it more suitable for certain business cases. The result might vary with a different and large set of data.

Future research could investigate the performance of other deep learning models or explore the use of BERT models to further improve intent classification accuracy. Overall, the findings of this study contribute to the growing body of research on deep learning models for natural language processing applications.

6.1 Future Scope

- a. LSTM is an excellent model that can handle complex data, but having the capacity of handling numerous rows has not been sufficiently utilized by the 210 rows in this model. In the future, additional surveys could gather additional data, train the model as per this new extensive data, and be made more effective.

- b. There are recent technologies in NLP, with profound architecture and vast libraries, for example, BERT and Transformers. Perhaps this data model can be trained and tested to be accurate and better suited for analyzing similar data types.

- c. Employing it in a real-time chatbot can produce more variety of data and categories to tune the model, which will be appropriate and accurate for solving real-time issues.

7. References

- [1] DHARMAJEE RAO, D.T.V. and RAMANA, K.V. A novel approach for efficient training of Deep Neural Networks. *Indonesian Journal of Electrical Engineering and Computer Science*. 2018. Vol. 11, no. 3p. 954. DOI 10.11591/ijeecs.v11.i3.pp954-961.
- [2] MARON, M. E. Automatic indexing: An experimental inquiry. *Journal of the ACM*. 1961. Vol. 8, no. 3p. 404–417. DOI 10.1145/321075.321084.
- [3] WU, Yingquan, IANAKIEV, Krassimir and GOVINDARAJU, Venu. Improved K-nearest neighbor classification. *Pattern Recognition*. 2002. Vol. 35, no. 10p. 2311–2318. DOI 10.1016/s0031-3203(01)00132-7.
- [4] JOACHIMS, Thorsten. Text categorization with support vector machines: Learning with many relevant features. *Machine Learning: ECML-98*. 1998. P. 137–142. DOI 10.1007/bfb0026683.
- [5] ALY, Rami, REMUS, Steffen and BIEMANN, Chris. Hierarchical multi-label classification of text with Capsule Networks. *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: Student Research Workshop*. 2019. DOI 10.18653/v1/p19-2045.
- [6] CHOLLET, FRANCOIS. *Deep learning with python*. S.l. : O'REILLY MEDIA, 2021.
- [7] What is machine learning? *IBM* [online]. [Accessed 19 March 2023]. Available from: <https://www.ibm.com/topics/machine-learning>
- [8] MUELLER, John and MASSARON, Luca. *Machine learning*. Hoboken, NJ : John Wiley & Sons, 2021.

- [9] ZHU, Xiaojin (Jerry). Semi-supervised Learning Literature Survey. *MINDS@UW Home* [online]. 1 January 1970. [Accessed 19 March 2023]. Available from: <https://minds.wisconsin.edu/handle/1793/60444>
- [10] BENGIO, Yoshua. *Learning deep architectures for AI*. Hanover, MA: Now Publishers, 2009.
- [11] GENUER, Robin and POGGI, Jean-Michel. Random forests. *Use R!* 2020. P. 33–55. DOI 10.1007/978-3-030-56485-8_3.
- [12] CHEN, Tianqi and GUESTRIN, Carlos. XGBoost. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2016. DOI 10.1145/2939672.2939785.
- [13] MACHADO, Marcos Roberto, KARRAY, Salma and DE SOUSA, Ivaldo Tributino. LIGHTGBM: An effective decision tree gradient boosting method to predict customer loyalty in the finance industry. *2019 14th International Conference on Computer Science & Education (ICCSE)*. 2019. DOI 10.1109/iccse.2019.8845529.
- [14] KIM, Yoon. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. DOI 10.3115/v1/d14-1181.
- [15] ALBAWI, Saad, MOHAMMED, Tareq Abed and AL-ZAWI, Saad. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*. 2017. DOI 10.1109/icengtechnol.2017.8308186.
- [16] DEVLIN, Jacob, CHANG, Ming-Wei, LEE, Kenton and TOUTANOVA, Kristina. Bert: Pre-training of deep bidirectional Transformers for language understanding. *ACL Anthology* [online]. [Accessed 19 March 2023]. Available from: <https://aclanthology.org/N19-1423/>

- [17] ZHANG, Yin, JIN, Rong and ZHOU, Zhi-Hua. Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*. 2010. Vol. 1, no. 1-4p. 43–52. DOI 10.1007/s13042-010-0001-0.
- [18] CAVNAR, W. B. and TRENKLE, J. [PDF] n-gram-based text categorization: Semantic scholar. [PDF] *N-gram-based text categorization* | *Semantic Scholar* [online]. 1 January 1994. [Accessed 19 March 2023]. Available from: <https://www.semanticscholar.org/paper/N-gram-based-text-categorization-Cavnar-Trenkle/49af572ef8f7ea89db06d5e7b66e9369c22d7607>
- [19] AUTHOR: FATIH KARABIBER PH.D. IN COMPUTER ENGINEERING, FATIH KARABIBER PH.D. IN COMPUTER ENGINEERING, PSYCHOMETRICIAN, Editor: Rhys and LEARNDATASCI, Editor: Brendan Founder of. TF-idf - term frequency-inverse document frequency. *Learn Data Science - Tutorials, Books, Courses, and More* [online]. [Accessed 19 March 2023]. Available from: <https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/>
- [20] MIKOLOV, Tomas, CHEN, Kai, CORRADO, Greg and DEAN, Jeffrey. Efficient estimation of word representations in vector space. *arXiv.org* [online]. 7 September 2013. [Accessed 19 March 2023]. Available from: <https://arxiv.org/abs/1301.3781>
- [21] PENNINGTON, Jeffrey, SOCHER, Richard, and MANNING, Christopher. Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. DOI 10.3115/v1/d14-1162.
- [22] RISH, I. An empirical study of the naive Bayes classifier. *Semantic Scholar* [online]. 1 January 1970. [Accessed 19 March 2023]. Available from: <https://www.semanticscholar.org/paper/An-empirical-study-of-the-naive-Bayes-classifier-Rish/3cd1df035ce7a022eefaf3190627ffcd5e43eca3>
- [23] UNIVERSITY, Richard Socher Stanford, SOCHER, Richard, UNIVERSITY, Stanford, UNIVERSITY, Jeffrey Pennington Stanford, PENNINGTON, Jeffrey, UNIVERSITY, Eric H. Huang Stanford, HUANG, Eric H., UNIVERSITY, Andrew Y. Ng Stanford, NG, Andrew Y., UNIVERSITY, Christopher D. Manning Stanford, MANNING, Christopher D., GENEVA,

University of, TECHNOLOGY, Massachusetts Institute of, UNIVERSITY, Macquarie and METRICS, Other MetricsView Article. Semi-supervised recursive autoencoders for predicting sentiment distributions: Proceedings of the conference on empirical methods in natural language processing. *DL Hosted proceedings* [online]. 1 July 2011. [Accessed 19 March 2023]. Available from: <https://dl.acm.org/doi/10.5555/2145432.2145450>

[24] SOCHER, Richard, HUVAL, Brody, MANNING, Christopher D. and NG, Andrew Y. Semantic compositionality through recursive matrix-vector spaces. *ACL Anthology* [online]. [Accessed 19 March 2023]. Available from: <https://aclanthology.org/D12-1110/>

[25] IYYER, Mohit, MANJUNATHA, Varun, BOYD-GRABER, Jordan and III, Hal Daumé. Deep unordered composition rivals syntactic methods for text classification. *ACL Anthology* [online]. [Accessed 19 March 2023]. Available from: <https://aclanthology.org/P15-1162/>

[26] HOCHREITER, Sepp and SCHMIDHUBER Jürgen. *Long short term memory*. München : Inst. für Informatik, 1995.

[27] BOWMAN, Samuel R., ANGELI, Gabor, POTTS, Christopher and MANNING, Christopher D. A large annotated corpus for Learning Natural Language Inference. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 2015. DOI 10.18653/v1/d15-1075.

[28] WANG, Zhiguo, HAMZA, Wael and FLORIAN, Radu. Bilateral multi-perspective matching for natural language sentences. *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*. 2017. DOI 10.24963/ijcai.2017/579.

[29] ALBAWI, Saad, MOHAMMED, Tareq Abed and AL-ZAWI, Saad. Understanding of a convolutional neural network. *2017 International Conference on Engineering and Technology (ICET)*. 2017. DOI 10.1109/icengtechnol.2017.8308186.

- [30] KIM, Yoon. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. DOI 10.3115/v1/d14-1181.
- [31] JOHNSON, Rie and ZHANG, Tong. Semi-supervised Convolutional Neural Networks for text categorization via region embedding. *arXiv.org* [online]. 1 November 2015. [Accessed 19 March 2023]. Available from: <https://arxiv.org/abs/1504.01255>
- [32] JOHNSON, Rie and ZHANG, Tong. Deep pyramid convolutional neural networks for text categorization. *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2017. DOI 10.18653/v1/p17-1052.
- [33] HE, Kaiming, ZHANG, Xiangyu, REN, Shaoqing and SUN, Jian. Identity mappings in deep residual networks. *Computer Vision – ECCV 2016*. 2016. P. 630–645. DOI 10.1007/978-3-319-46493-0_38.
- [34] YANG, Zichao, YANG, Diyi, DYER, Chris, HE, Xiaodong, SMOLA, Alex and HOVY, Eduard. Hierarchical Attention Networks for document classification. *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2016. DOI 10.18653/v1/n16-1174.
- [35] PETERS, Matthew E., NEUMANN, Mark, IYYER, Mohit, GARDNER, Matt, CLARK, Christopher, LEE, Kenton and ZETTLEMOYER, Luke. Deep contextualized word representations. *arXiv.org* [online]. 22 March 2018. [Accessed 19 March 2023]. Available from: <https://arxiv.org/abs/1802.05365>
- [36] DEVLIN, Jacob, CHANG, Ming-Wei, LEE, Kenton and TOUTANOVA, Kristina. Bert: Pre-training of deep bidirectional Transformers for language understanding. *arXiv.org* [online]. 24 May 2019. [Accessed 19 March 2023]. Available from: <https://arxiv.org/abs/1810.04805>
- [37] JOSHI, Mandar, CHEN, Danqi, LIU, Yinhan, WELD, Daniel S., ZETTLEMOYER, Luke and LEVY, Omer. Spanbert: Improving pre-training by representing and predicting spans.

Transactions of the Association for Computational Linguistics. 2020. Vol. 8, p. 64–77. DOI 10.1162/tacl_a_00300.

[38] LAN, Zhenzhong, CHEN, Mingda, GOODMAN, Sebastian, GIMPEL, Kevin, SHARMA, Piyush and SORICUT, Radu. Albert: A lite bert for self-supervised learning of language representations. *arXiv.org* [online]. 9 February 2020. [Accessed 19 March 2023]. Available from: <https://arxiv.org/abs/1909.11942>

[39] SAK, Haşim, SENIOR, Andrew and BEAUFAYS, Françoise. Long short-term memory recurrent neural network architectures for large-scale acoustic modeling. *Interspeech 2014*. 2014. DOI 10.21437/interspeech.2014-80.

[40] GRIEWANK, Andreas and WALTHER, Andrea. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. Philadelphia, PA: Society for Industrial & Applied Mathematics, 2009.

[41] PICHOTTA, Karl and MOONEY, Raymond. Learning statistical scripts with LSTM recurrent neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*. 2016. Vol. 30, no. 1. DOI 10.1609/aaai.v30i1.10347.

[42] CHO, Kyunghyun, VAN MERRIENBOER, Bart, GULCEHRE, Caglar, BAHDANAU, Dzmitry, BOUGARES, Fethi, SCHWENK, Holger and BENGIO, Yoshua. Learning phrase representations using RNN encoder–decoder for statistical machine translation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014. DOI 10.3115/v1/d14-1179.

[43] GRAVES, Alex and SCHMIDHUBER, Jürgen. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*. 2005. Vol. 18, no. 5-6p. 602–610. DOI 10.1016/j.neunet.2005.06.042.

[44] GERS, F.A. Learning to forget: Continual prediction with LSTM. *9th International Conference on Artificial Neural Networks: ICANN '99*. 1999. DOI 10.1049/cp:19991218.

- [45] SHMUELI, Boaz. Multi-class metrics made simple, part II: The F1-score. *Medium* [online]. 21 July 2022. [Accessed 19 March 2023]. Available from: <https://towardsdatascience.com/multi-class-metrics-made-simple-part-ii-the-f1-score-ebe8b2c2ca1>
- [46] BIRD, Steven, KLEIN, Ewan and LOPER, Edward. *Natural language processing with python: Analyzing text with the natural language toolkit*. Beijing: O'Reilly, 2009.
- [47] Van Houdt, Greg & Mosquera, Carlos & Nápoles, Gonzalo. (2020). A Review on the Long Short-Term Memory Model. *Artificial Intelligence Review*.
- [48] TURING, A. M. (1950, October 1). I.—COMPUTING MACHINERY AND INTELLIGENCE. *Mind*, *LIX*(236), 433–460. <https://doi.org/10.1093/mind/lix.236.433>

8. List of Figures

8.1 List of Figures

Figure 1. Text classification flowchart	4
Figure 2. Artificial intelligence, machine learning, and deep learning. [6]	6
Figure 3. The structure of word2vec, including CBOW and Skip-gram. [17]	14
Figure 4. LSTM architecture. [26]	25
Figure 5. Example of a confusion matrix. [45]	27
Figure 6. Two-class Confusion Matrix. [45]	28
Figure 7. Student query dataset. [own work]	30
Figure 8. Categories of Data. [own work]	31
Figure 9. Google Colab environment. [own work]	32
Figure 10. Library imports in Google Colab. [own work]	32
Figure 11. Files upload function in Google Colab. [own work]	33
Figure 12. Top 10 rows of the dataset. [own work]	34
Figure 13. Basic information about the data. [own work]	34
Figure 14. Statistical information about the data. [own work]	35
Figure 15. Categories and their count. [own work]	36
Figure 16. Queries as an array of tokens. [own work]	37
Figure 17. Implementation of Gensim's Preprocessing function. [own work]	39
Figure 18. Transforming categories to one hot encoding. [own work]	40
Figure 19. Tokenizing the queries. [own work]	41
Figure 20. Padding the tokens to uniform the length. [own work]	42
Figure 21. Splitting the dataset into train and test data. [own work]	44
Figure 22. Building LSTM model. [own work]	45
Figure 23. Experimenting with epochs and batch_value for finding optimal value. [own work]	49
Figure 24. Training of the model. [own work]	50
Figure 25. Evaluation of the trained model with test set. [own work]	51
Figure 26. Tensorboard plotting of LSTM model training. [own work]	52
Figure 27. IBM Watson setup. [own work]	53
Figure 28. Shuffling and data pre-processing for IBM Watson. [own work]	53

Figure 29. Splitting the data into train and test datasets. [own work]	54
Figure 30. Installing ibm_watson library to Google Colab environment. [own work]	54
Figure 31. Importing necessary libraries for IBM Watson API. [own work]	55
Figure 32. Parameters for IBM Watson API call. [own work]	55
Figure 33. Creating session with IBM Watson Assistant. [own work]	56
Figure 34. Predicting categories for queries with IBM Watson via API. [own work]	57
Figure 35. Response received from IBM Watson Assistant service with input query, its predicted intent, and confidence score. [own work]	57
Figure 36. Printing the metrics for evaluation. [own words]	58

8.2 Appendix

1. Query.xlsx
2. Multi_Class_Text_Classification_using_LSTM.ipynb
3. IBM Watson.ipynb