University of Passau
Faculty of Computer Science and Mathematics
Chair for Digital Libraries & Web Information Systems

Master Thesis in Informatics

– Evaluating Classification Models Using Question-to-Question Matching In Home Improvement Domain –

submitted by

Ahmad Bilal Sohail

1. Examiner:	Prof. Dr. Markus Endres
2. Examiner:	Prof. Dr. Harald Kosch
Supervisor:	Macedo Sousa Maia
Date:	April 7, 2022

Contents

Abstract

Stackexchange is a widely used question-answer knowledge-based website with many domains; one of them is HomeImprovement. A user can post a query on the Stackexchange forum, and another user can upvote and answer it. Due to the popularity of QA-based websites, they suffer from one common problem: the replication of questions. Given a new query, assuming if questions are similar, answers to the questions should be similar. For this work and the researchers of NLP, we made a dataset for the home improvement domain, which is now available publicly. Categorizing the questions into irrelevant, relevant, and similar is not only required to use this system effectively and efficiently, but it is also required to show the related and linked questions to the users to enhance their knowledge further and to increase the captivity nature of the website. It will also help improve the response time by identifying if the question is already asked and answered while maintaining the quality of the website. A recommendation for the relevant questions is provided for further knowledge in the case of a new question. In contrast, the question is answered by someone. A pair of two questions can be similar, relevant, or irrelevant, which is identified by the help of lexical and semantic similarity. We used word2vec and BERT as word representation/word embeddings in this work. At the same time, we compared eight different classification models ranging from machine learning to deep learning and encoder-based models to classify the text into three classes(irrelevant, relevant, and similar). These models are then evaluated through Precision, Recall F1 score, Micro, Macro, and weighted averages to understand the outcomes of these models better. The RoBERTa-base model with BERT embeddings produced the highest results from the used models. At the same time, SVM with word2vec showed the lowest performance.

Acknowledgments

First and foremost, I would like to thank God for making me persistent and blessing me with enough abilities to complete this work. Secondly, I want to express my gratitude to both University of Passau and Prof. Dr. Markus Endres, the head of the chair Digital Libraries and Web Information Systems at the University of Passau, for offering me a topic in the field that intrigue me the most. I am thankful to him for supervising this thesis work. His suggestions during expose presentation really helped me speed up this work. I wish to thank Prof. Dr. Harald Kosch for his valuable time in reviewing and evaluating my thesis as the second examiner.

It would not have been feasible for me to complete this work in time without the unwavering support and guidance of my advisor Mr. Macedo Sousa Maia. I wish to express my utmost gratitude to him for his continuous guidance at every stage of the thesis project. Furthermore, I thank him for providing me with his insightful suggestions and feedback on the research work. He is an excellent research advisor. His great ideas, advice, and long discussions kept me in the right direction to accomplish the thesis work.

I am also thankful to Google's Colaboratory Service for providing the computational resources which are immensely used in this work. I am very grateful to my childhood friends Dr. Usman and Engr. Salahuddin who helped annotate the dataset democratically with their expertise in electrical engineering.

Special thanks to my parents, who never had a doubt about me. This work would have been impossible without my parents' endless support and encouragement. Likewise, I want to acknowledge my family and friends' endless love and support. I would also like to pay my gratitude to all the front-line workers and the doctors who risked their lives to keep us safe during the COVID19 pandemic.

List of Figures

List of Tables

1 Introduction

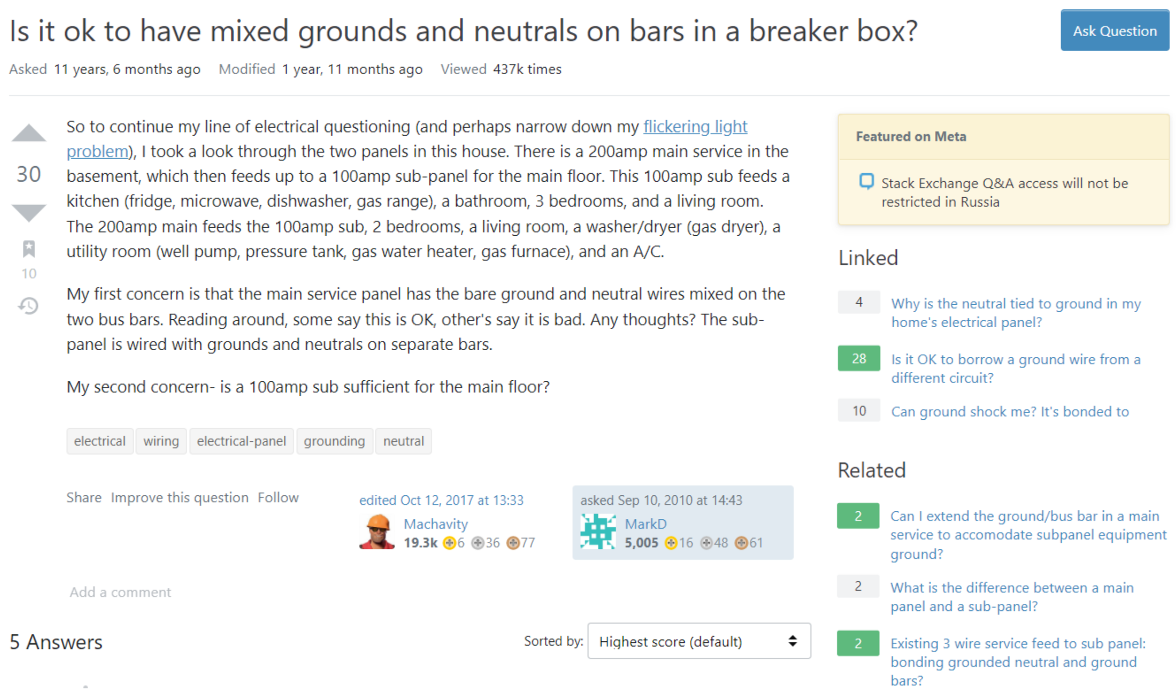
1.1 Context

Home is said to be a basic need of human beings. We have come a long way from caves gradually improving day by day. Today homes are not as simple as they used to be decades ago. With the advent of technology and ongoing research, a lot is gradually being improved over time. This makes the home improvement domain an essential aspect. However unfortunately, this field has not gained enough engagement as it should have. Today everybody needs information to fix or apply new gadgets in their homes independently. This motivated us to discover more and invest our time in this field.

The world has progressed so much that it is not possible for someone to have knowledge of all the domains. Even experts working in some specific domains need help to optimize their solutions further. The same goes for the home improvement domain. For example, one may want to know how to connect fiber wires or put curtains properly on a cupboard wall without damaging it. Others may need help to repair a broken door handle or ceiling tiles. If this kind of help is available immediately, it may save a lot of time and can also be very cost-efficient.

Today, the internet is flooded with a huge amount of data that did not appear overnight. It is the collective effort of yours, mine, and everybody out there using the internet. This data serves different kinds of needs of different kinds of people. We may use this data ranging from learning to entertainment, business to betting, politics to e-commerce, and the list goes on. As data is getting larger and larger with every passing second, we need some technical mechanism to use it efficiently. We use many domain-specific websites to ease our different kinds of tasks. The most well-known of them are "www.qoura.com", "www.stackoverflow.com", and "www.stackexchange.com", which are question-answer-based websites where users ask their desired questions, and other users upvote and answer them. Having the biggest community, Quora is a general-purpose website where one can ask a question from any field related to any topic. On the other hand, Stackoverflow is a domain-specific platform designed for programmers under the umbrella of "www.stackexchange.com".

Like all other platforms, these websites suffer from the problem of duplication due to a massive amount of data. They receive thousands of questions. Not all of them are unique and new, particularly with the increase in the number of questions being asked. Some of them have already been richly answered not one but many times. If duplicate questions are tolerated, they will temper the quality and richness of the platform. It will equally affect all of the persons who are asking questions, answering them, and the viewers (imagine you make a query from google and see ten separate and same kinds of outcomes from the same website). It demands extra patience from the person who is asking questions and extra effort from the person who is answering. The viewer will also be confused about choosing one of all available duplicate questions. Furthermore, duplicate questions restrict the viewers from getting high-quality responses as other users or experts have already provided the answers to the previously asked similar questions and now would be unwilling to respond to the same questions over and over again. This thesis aims to prevent duplicate questioning by finding the questions that are similar in context to the question at hand. This will give a quick solution to questions that have already been asked and provide quality and precise information to the viewer.


Is it ok to have mixed grounds and neutrals on bars in a breaker box?

Asked 11 years, 6 months ago Modified 1 year, 11 months ago Viewed 437k times

30

So to continue my line of electrical questioning (and perhaps narrow down my flickering light problem), I took a look through the two panels in this house. There is a 200amp main service in the basement, which then feeds up to a 100amp sub-panel for the main floor. This 100amp sub feeds a kitchen (fridge, microwave, dishwasher, gas range), a bathroom, 3 bedrooms, and a living room. The 200amp main feeds the 100amp sub, 2 bedrooms, a living room, a washer/dryer (gas dryer), a utility room (well pump, pressure tank, gas water heater, gas furnace), and an A/C.

My first concern is that the main service panel has the bare ground and neutral wires mixed on the two bus bars. Reading around, some say this is OK, other's say it is bad. Any thoughts? The sub-panel is wired with grounds and neutrals on separate bars.

My second concern- is a 100amp sub sufficient for the main floor?

electrical wiring electrical-panel grounding neutral

edited Oct 12, 2017 at 13:33 Machavity 19.3k 6 36 77

asked Sep 10, 2010 at 14:43 MarkD 5,005 16 48 61

Featured on Meta: Stack Exchange Q&A access will not be restricted in Russia

Linked

4 Why is the neutral tied to ground in my home's electrical panel?

28 Is it OK to borrow a ground wire from a different circuit?

10 Can ground shock me? It's bonded to

Related

2 Can I extend the ground/bus bar in a main service to accomodate subpanel equipment ground?

2 What is the difference between a main panel and a sub-panel?

2 Existing 3 wire service feed to sub panel: bonding grounded neutral and ground bars?

5 Answers

Sorted by: Highest score (default)

Figure 1.1: A question from diy.stackexchange.com

1.2 Motivation

As the internet has become the primary source of communication globally, everyone uses the internet to communicate with family, friends, and colleagues. The internet has also made the cast information accessible to internet users. The exponential growth of data over the internet has become normal. It is the need of the day to make a mechanism to handle and use data effectively and efficiently. Text similarity has a lot of applications, for example, plagiarism detection, entailment detection, data retrieval, redundancy removal, e.t.c. Our problem encapsulates almost all of those applications to provide a better solution.

Today, we use so many home gadgets, tools, and products on a daily basis. We can apply and fix all these things ourselves if we have a guiding medium, and of course, we can save so much time and resources by doing it ourselves. For example, people commute from one place to another and need new furniture, for which they need assistance as well, and then to assemble it, they might need help. Similarly, if anything gets broken or goes out of order, they should be able to fix it themselves. Especially this recent covid19 pandemic has taught us many things and among them is to be independent and perform all of the stuff by ourselves. So I believe all these points make home improvement a significant domain for our daily easiness.

With this work, everybody needing guidance or opinions in the home improvement domain will be benefited. This work will make their experience of seeking information smoother and easier. They will be able to get quality information without ambiguous data. In most cases (if a similar question has already been answered), they won't have to wait for someone to answer their query. A quick solution can be to redirect to the previously answered solution to the same query. It will save the time of users as well as data space on the internet by suggesting them similar questions.

Let us have a look at the following examples.

Q1: What is the price of that brown sofa?

Q2: How much should I pay to buy that brown sofa?

Q3: When did America's president take his oath?

Q4: When did Joe Biden announce the biggest economic plan of the decade?

Q5: Which of the red, blue, and black wire should I ground for earthing?

Q6: What is the optimal breadth of a wooden wall?

In the above pair of questions, Q1 and Q2 are similar. They are similar because the meanings of both texts are the same as the price of the sofa is being asked for. Q3 and Q4 are related, as the meanings of both texts are different from each other. Still, they have a common relation between them: the president of the USA. The next pair of questions is not related. Q4 is about earthing the wire, while Q5 is about the breadth of the wooden wall.

Earlier, older methods were being used to find text similarity, which was not accurate enough. New advancements have been observed with time, which gradually made it more precise and easy to handle. Statistical models have been used for such a long time for this purpose. One could find lexical similarities, but they were not good at finding similarities based on actual meanings. Deep learning, Transformers, and BERT have revolutionized these NLP tasks. Bert, along with its derived versions(which provided a trade-off between accuracy and speed), was considered to be SOTA(state of the art) in recent past years.

It is possible to use the enormous amount of unlabeled text on the internet to aid in understanding meaning and sentence structure. Word2vec, which learns phrase embeddings from unlabeled textual content samples, has tried this technique. It learns by predicting a phrase based on its surroundings (CBOW) and predicting surrounding phrases based on a given phrase (SKIP-GRAM). These phrase embeddings are used to expand dictionaries as dimensionality reducers in current techniques such as Tf-IDF, etc. More approaches for capturing sentence-level representations, such as the recursive neural tensor community, are being explored (RNTN). The convolution neural community, which has primarily been employed for image-based classification, has shown to be effective in text classification.

Initially, support vector machines, k nearest neighbors, naive bayes, and tree ensembles produced better results in natural language processing. Later on, deep learning revolutionized computer vision with RNNs at first. Still, later on, the shortcomings of RNNs, such as gradient vanishing and gradient exploding, were taken care of with a new variant of RNN i.e.

LSTM (long short term memory) in 1997. In 2014, a gated recurrent unit (GRU) was proposed to be more straightforward in structure and more practical than LSTM. LSTM and GRU produced much better results as compared to older and simpler machine learning models. The most recent and advanced ones are transformer-based models. Some of them are decoder-based, while some are encoder-based models. BERT and RoBERTa being encoder-based models, are a couple of more suitable models for our task. In this work, we used SVM, random forest, LSTM, GRU, biLSTM, biGRU, BERT, and RoBERTa models for multiclass text classification.

1.3 Goal of this work

1.3.1 Basic Architecture

Breaking our problem into smaller pieces leads us to a simple function that can be efficiently used to see if two pieces of text are similar are not.

$$f(t1, t2) \rightarrow 0 \text{ or } 1 \text{ or } 2$$

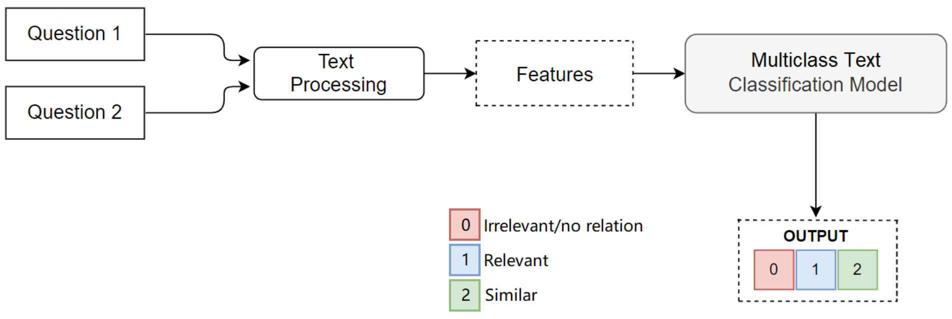


Figure 1.2: A very basic architecture of classification tasks

Given a tuple of two texts, the function returns 2 if they are similar, 1 if they are related, and 0 if they do not possess any relation among them. This architecture is also shown in figure 1.2. We can see that our Classification model gives output in labels 0,1 and 2, which are no relation, related and similar, respectively. A simple similarity detection can be solved with a word comparison approach. However, only simple information retrieval methods i.e. word2vec or tf-idf can be used to get word-based similarity between two given texts. Ranking scores can be given to them, but we need more intelligent methods to achieve a semantic similarity. A lot of research is still going on to obtain semantic similarity efficiently.

1.3.2 Research Questions

- **What are the most important steps to build a new dataset for a new and real-world question-to-question dataset?**

We will see different steps (and their impact on results) of producing a better data frame to implement the machine learning models. The results of a system are hugely influenced by the data we feed to the model

- **How can we validate our newly built dataset?**

Dataset validation is one of the most critical steps of ML problems. Yet, it is often skipped as people usually think that it will slow the pace of the project. However, it is essential to validate the data's accuracy, details, and correct structure to mitigate the defects later. For example, suppose the data model is not built correctly. In that case, multiple issues can arise later, and even the accuracy can also be hugely decreased.

- **Which kind of Model has better performance in question to question matching in the home improvement domain?**

We will try to compare the accuracy of different deep learning and statistical models in this problem keeping things as simple as possible. After experimenting with our data set, we will be working on GBDT from decision tree ensembles if we get a high bias. On the other hand, if we get a high variance, then RFDT will be our go-to choice from statistical models. Our main goal here is to maximize the correctly predicted outputs.

1.4 Structure of this work

This section serves as a road map for this thesis work, ensuring that we meet and fulfill all of our aims and objectives. In addition, it lays out the outline for different topics explained in different chapters of this work.

Chapter 1, introduction, is the first chapter of this work that sets up the topic and approach for the thesis. It describes the working of current QA websites and the issues they face due to their popularity and excessive usage worldwide. It also tells the motivation behind this work. Finally, it proposes models to be used along with the research questions to be tackled in this work. Chapter 2, background, discusses a range of potential models to be used in this work in detail. It explains the ideas of text classification, multiclass classification, and a couple of machine learning models, along with a few neural network models and encoder-based models. One of the major objectives of this work is to find the best model for our dataset, so a deep understanding of the used classification models is required to evaluate them better. As we used eight classification models in total, this will be a long chapter. In the next chapter 3, we discussed some of the worth mentioning work of scientists which is technically relevant to our task. Different scientists proposed different approaches to solve similar problems. Their approaches, along with their results, are breifly discussed in this chapter. Chapter 4 presents the step-by-step approach to building the dataset. Dataset is first built before finally using and finding the best classification model. Chapter 5 first tells about the experimental setup and then elaborates on the usage of different methods along with chosen hyperparameters to provide a proper understanding of the implementation. Finally, we have shown the results in chapter 6 along with the evaluation matrices. In chapter 7, we go through all of our findings, and we try to answer the research questions of this work. We try to present whatever we learned from this work. Finally, chapter8 concludes and describes how this work can be extended in the future.

2 Background

Alan Turing, a well-known computer scientist, and mathematician, in 1950 said that if a machine becomes able to converse with a human and he is unable to differentiate whether he is talking to another person or a machine, then such a machine is believed to be artificially intelligent. This is how the foundation of natural language processing was laid. A system effectively benefiting from NLP can easily understand the query and its meaning, interpret it, identify the best possible actions and respond to the user in his language such that he can easily understand the machine.

Natural Language Processing (NLP) is a well-known field today and has been around for quite some decades now. Today this field is booming, and every researcher wishes to be a part of this evolution. Natural language processing over time evolved with the newer techniques to better understand, interpret and solve human language in a better way. Even though NLP is still relatively a young field in terms of advancement, it has seen exponential growth in progress in the past few years. In the field of AI(artificial intelligence), the recent advancements have had a significant impact on NLP's efficiency and popularity. This is why, now, a lot of researchers, as well as large firms and enterprises, are attracted to research in NLP. It can be categorized as Natural language understanding(NLU) and natural language generating(NLG).

2.1 Natural Language Processing in Home Improvement Domain

Home improvement is a domain that works with informal and casual language and unorganized processes. Efforts are being made to benefit from software and AI to digitally optimize and simplify the DIY processes along with optimal usage of resources. Home improvement has an informal language with an informal syntax, multi-lingual words, and a very diverse vocabulary. All these characteristics of the DIY language make it a challenging subject to exploit the power of natural language processing in this particular field. NLP can help with various challenges in this specific domain, such as extracting relevant information about user queries, automatic error checking, automatic answering of frequent questions, and Assist-Bots as helping hands.

2.2 Language Modeling

Prediction of text based on the previous available text (sequence of words), which is at the heart of Language Modeling, is one of Natural Language Processing's most fundamental challenges. Through the advent of language modeling approaches, such as N-gram, the most used statistical models were probabilistic Language Models or rule-based. The N-gram model is simple enough to limit its performance over longer text. In language modeling challenges, neural language models helped overcome these constraints and surpassed Statistical Language Models. The vanishing gradient problem-plagued recurrent neural networks (RNNs) prevent them from capturing relevant relationships in extended sequences. Long Short Term Memory (LSTM) solved the difficulty of capturing long-term dependencies in sequences other RNNs had.

2.3 Word Embeddings

Natural languages, the fundamental source of our communication, are well understood by us but not by computers. The computer, being a machine, understands numbers in a better way. Keeping these ground truths in mind, NLP suggests converting the text into numbers and vectors for the ease of machines before actually processing them. Word Embeddings are intelligent techniques that in a predefined vector space, map the words to real-valued vectors. It is one of the most popular techniques to represent the vocabulary of a document. As described in the previous section, word representations have two main types context-free representational models and contextualized representational models. Models like Word2vec, FastText, and Glove generate context-free embeddings. These were early state-of-the-art embedding techniques of their times. Later on, ULMFit, ELMo, and flair became popular due to their ability to capture the context of the text.

2.4 Learning The Context Of The Text

One of the essential components of Natural Language Processing is Word representation, and it has a significant impact on performance, particularly for Deep neural network models. Word representations can be created by different types of procedures and a variety of approaches. can be either distributed as word embeddings or fixed as generated by different methods such as "dictionary lookup" and "One-Hot Encoding." The efficiency of Deep Learning approaches in NLP is significant to the use of Distributed Representations. A decent word representation approach can represent the semantics and structure of the natural language.

There are two main types of word representation i.e. Static Word Representation and Contextualised Word Representation. Similar words can have altogether different meanings depending on the context of the text around them. This phenomenon is not regarded by Static word representation models, also known as context-free models. Anytime the same word appears in the text, the exact representation of the word will be generated even if the meaning in a later context is different. On the other hand, a contextualized representation model will generate different representations of the word if the meaning changes with respect to the context. Let us take an example to better illustrate this phenomenon. "Doctors suggest eating an apple every day" and "Apple is going to make significant changes in the design of the upcoming iPhone 14". Context-free models will generate the exact representation of the word apple for both sentences, although the meanings of apple are different in both of the sentences. In the first sentence, the apple is a fruit, while the word apple is an electronics company in the second sentence. Contextualized models will generate different representations of the word Apple. Hence, contextualized representational models yield far better results as compared to context/free models Context can be learned in 2 ways, unidirectional or bidirectional.

J.R Firth, long ago in 1957 said "You shall know a word by the company it keeps" (J.R Firth 1957).

OpenAI GPT generates the word representations only backwards as the context is learned from left to right so it remembers the previous words only. On the contrary, Elmo and ULMFit learn the bidirectional context. They learn forwards and backwards by leaning from right to left and from left to right and later on combining both of the learned representations. They are shallow bidirectional as they use sequence architecture to sequence Recurrent neural networks. On the other hand, Bert, benefiting from the architecture of transformers, learns all the sequences at once which enables it to capture a deep bidirectional context.

2.5 Machine Learning Models

In general, machine learning is a combination of statistics and computer science. The term machine learning means making computers able to learn from data such that the algorithms, after being programmed, can be used to perform a specific task. However, instead of explicitly being programmed, they learn different patterns from the data, understand them and make a model out of this understanding. Finally, this model can make predictions based on learned patterns and understanding even of new and unseen data. There are hundreds of machine learning classifiers proposed until today, but they are categorized into four distinct categories.

- **Supervised**
- **Semi-supervised**
- **Unsupervised**
- **Reinforcement**

Machine learning models are classified into two types, i.e., shallow machine learning and Deep machine learning. Deep models, as the name suggests, are the ones that consist of deep hidden layers. LSTM, GRU, biLSTM, biGRU, RNN, and CNN are examples of deep learning models. Some well-known shallow machine learning classifiers are support vector machines, logistic regression, random forests, XGBoost, naive bayse, and knn. Our problem is supervised machine learning and involves ternary classification. Basically, SVM is a binary classifier but can be used for multi-classes by choosing a parameter one vs rest(ovr). Here in this work, we will make use of support vector machines and random forest from shallow machine learning models, which are explained in 2.5.1 and 2.5.2 respectively.

2.5.1 Support Vector Machines

SVM, an abbreviated form of support vector machines, is a sort of supervised machine learning technique that can be used for both regression tasks and classification tasks. They improve fundamental machine learning algorithms by adding characteristics that enable them to be more efficient at specific tasks. For example, anomaly detection, handwriting recognition, and text classification are just a few tasks that SVM can help with. SVMs have become an essential pillar of machine learning and significant addition to the toolkit of ML engineers due to their flexibility, high performance, and computing efficiency.

There are two types of SVM, i.e., linear SVM and non-linear SVM. A linear kernel is one where the data points can be separated with the help of a linear separating line, and the data that can be linearly separated is called linearly separable data. On the other hand, if data points of different classes are mixed up in a complex way such that a straight line is unable to separate the classes, then the non-linear SVM is used, and the data is known to be non-linear. This indicates that the algorithm's calculated border does not have to be a straight line. The advantage is that we can catch far more complicated relationships between our data points without executing complex transformations ourselves. The disadvantage is that it takes much longer to train because computationally, it might be much more expensive depending on the data.

Through linear separation, a machine learning model can learn to classify the points into two classes. With the help of different models, different lines can be drawn to separate two classes, but not all of them are equally valuable. In case the line drawn by our model is not in the middle so that it is far away from one class while being closer to another class, the model will behave in a sensitive way to minor divergence from the norm in that class. As a result, if a new point is a little farther away from the training cluster, the ML model, in most cases, will miss-classify it.

2.5.1.1 Linear SVM

- Support Vectors

The closest data points or vectors to the hyperplane that influence its location are referred to as "support vectors." These vectors are known as "support vectors" because they provide support for the hyperplane. SVMs solve classification tasks by utilizing "support vectors." The support vectors (dashed lines in the image 2.1) are equally spaced apart from the primary classification line (solid line). By including support vectors into the edge instances of each class, the SVM generates a machine learning model that is resilient to data that does not precisely match the training examples.

- Hyperplane

In n-dimensional space, several lines/decision boundaries can be used to segregate the classes, but we must select the optimal separating line to categorize the data points better. Such a separating boundary line is known as an optimal boundary. The dimensions of the hyperplane are determined by the total number of classes in the dataset; for example, if there are two classes as we have in the figure 2.1, the hyperplane will be a straight line. However, instead of a straight line, the hyperplane will be a two-dimensional plane in the case of three features. The primary purpose of the SVM algorithm is to find the best possible segregating line that can separate the n-dimensional space into certain distinct classes. Moreover, it could enable us to assign the new data to correct categoriesclasses based on segregating lines in the future. Linear SVM is implemented by finding the optimal hyperplane, and the classifier needs only one line, as shown in the figure 2.1.

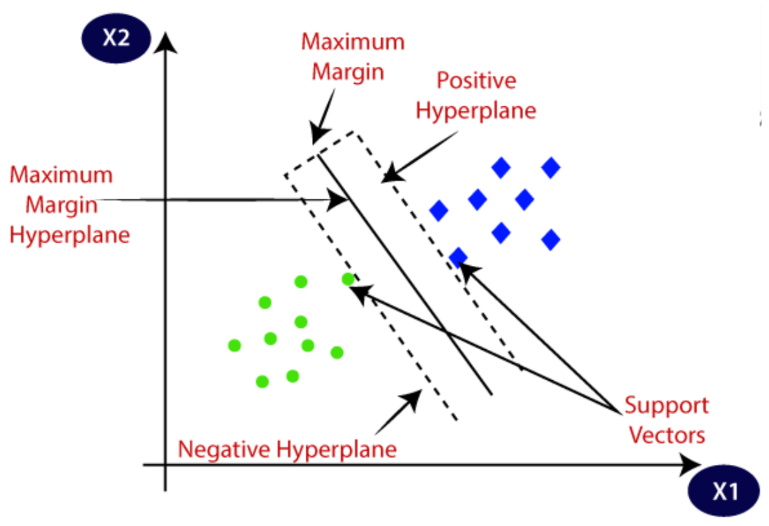


Figure 2.1: Support vectors and hyperplane of SVM algorithm.[Ste17]

2.5.1.2 Non-linear SVM

The data shown in figure 2.1 is straightforward, as the data were easily separable linearly, and we could draw a straight line to split blue and green. Unfortunately, things are rarely that straightforward. Suppose the linear boundary(a single straight line separating both tags) is impossible. A quick possible trick to get out of this situation is introducing a third dimension. Previously, we had only two dimensions: y and x. We establish one more dimension as z and specify it to be calculated in the following manner: z = x^2 + $y2$ (it is the equation for a circle).

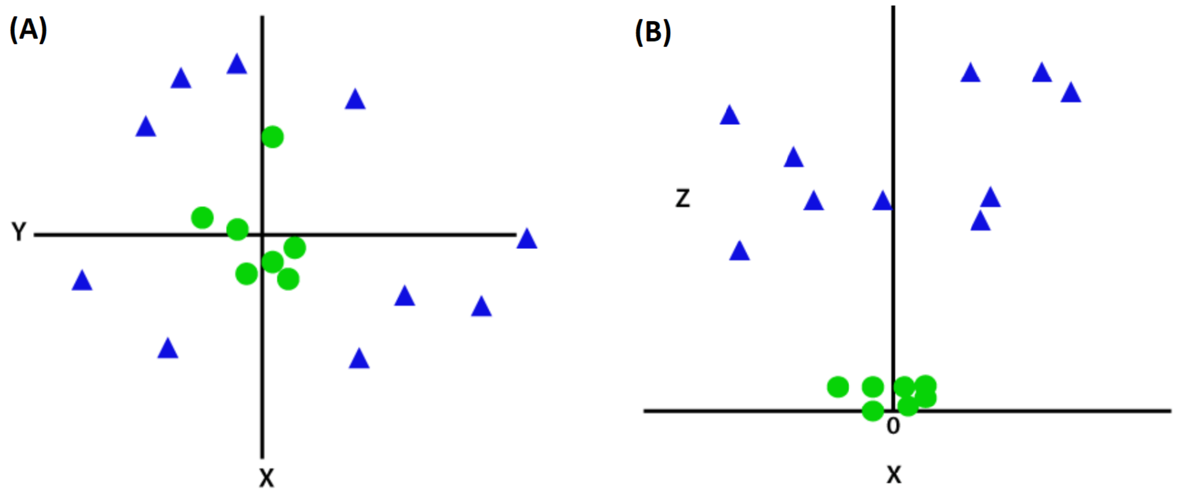


Figure 2.2: Non-linear Data converted to higher dimension

This results in the creation of a three-dimensional space. Then we will be able to disparate the mixed-up data points into distinct classes with the help of the third dimension. We can see in figure 2.2(A) that the data points are mixed up and cannot be separated with a linear line. Here a specific kernel trick, among many others, is used that is to convert to a higher-dimensional space. Figure2.2(B) shows that after converting it to 3 dimensions and it is now possible to separate the data points linearly.

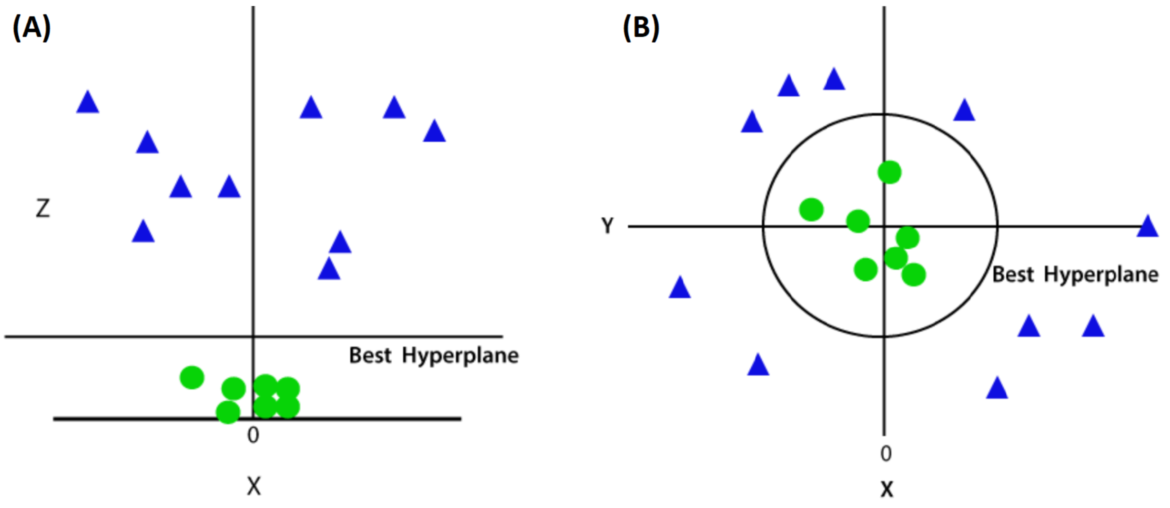


Figure 2.3: Separating the Non-linear Data converted to higher dimension with hyperplane

In figure2.3(A), the data points are separated into two distinct classes with the help of one line. Whereas at the same time, we can get the 2-dimensional view of our data points in figure2.3(B). The hyperplane is in the shape of a circle in 2 dimensions.

2.5.1.3 Multiclass classification with SVM

SVM was initially suggested by Vapnik in his work[JT98] for binary classification, in which a hyper-plane is formed that maximizes the margin between two different classes. However, because many classification tasks, particularly text classification, include many more classes, multiclass classifiers built from a number of binary classifiers are frequently used. The three most prominent multiclass SVM techniques are described here.

1-verses-all

We generate N binary classifiers in total for N separate classes, one of which is positive, and all of the rest are considered negative. A classifier supports each class, classifying its occurrences as positive.

One-verses-one

Using all of the binary pair-wise combinations of the N classes, we create N (N-1)/2 classifiers and ensemble them with the help of the voting technique.

Divide-by-two

At first, the whole set of labels is considered a tree's root node. Then, a binary SVM classifier, just like a binary tree, divides the label set into two subsets, and this process is continued

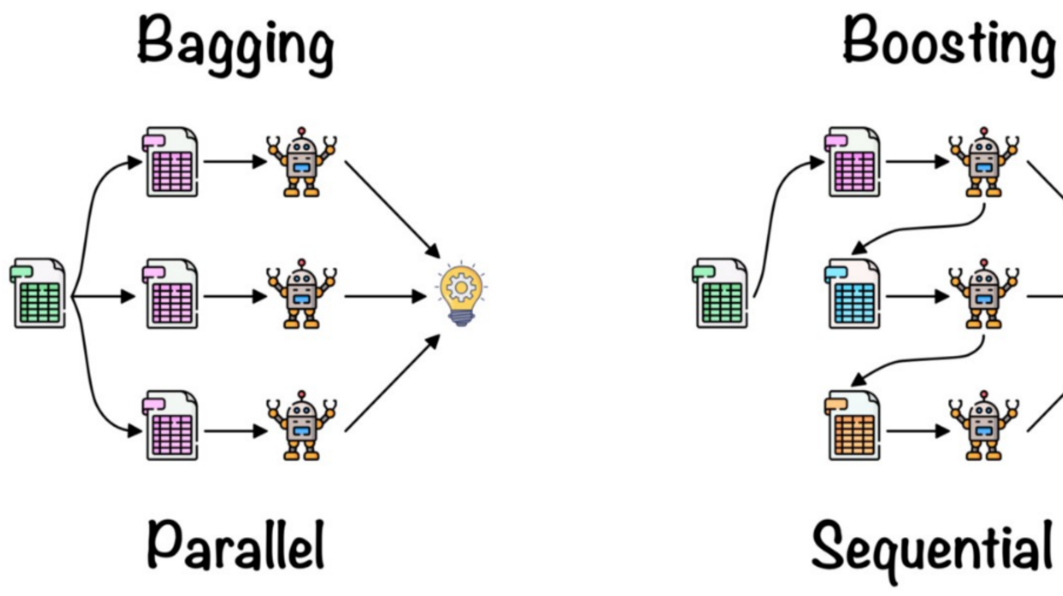


Figure 2.4: Bagging and boosting methods to ensemble.[Sru21]

until all the labels are separate from each other, just like the leaves of a tree. In the end, these N-1 classifiers will generate a decision tree.

2.5.2 Random Forest

The decision tree is an example of a classification problem where the class labels can be "yes" or "no". They were quite helpful. Decision trees can be prone to problems like bias and overfitting. For a specific observation, a classification tree forecasts that it belongs to the training observation class, which occurs most frequently in the region that observation belongs to. A region R_l such that, l = 1,2,...,l is described as $R_l = \{v_{dl} | v_1 < c_i, ..., v_2 < c_c\}$ where feature vector is denoted by v_{dl} and c_c is denoted as conditional cutpoint. Regions are also known as leaves of the tree, and there are l leaves or regions. The basic purpose of the tree is to minimize Gini index and error. The Gini index can be calculated by adding all the squared probabilities of every class and then subtracting them from 1. It is the basic functionality behind the classification decision tree. Combining different models is known as an ensemble, and several different tree ensembles were proposed, and a few of them outperformed decision trees by a significant margin. There are two different methods for tree ensemble, as shown in figure 2.4.

- Bagging

The sample training data generates a different training subset with replacement, and the final output is determined by majority voting. i.e., Random Forest.

- Boosting

Boosting works by creating sequential models such that the model achieves the highest accuracy by combining weak learners with strong learners. XGBoost and AdaBoost are examples of boosting methods.

Random forest uses an ensemble of decision trees to make its predictions. The reason behind calling it a random forest is that it is built by picking up a random sample from the data and then building an ongoing series of decision trees on the subsets. So basically, it essentially creates a whole bunch of decision trees together, which makes them a bigger model. The more decision trees used with different criterion, the better it performs because it increases the prediction accuracy. Furthermore, those will be ignored if one, two, or any of these smaller decision trees are not relevant to a particular feature. Using random forest can be challenging at first because one wants to use many trees, like as many as one can, to get the best predictive accuracy, but one does not want so many trees that it will take a long time to train the model and use a lot of memory space. The random forest algorithm is described in a few steps below and is shown in the figure 2.5.

Step I-

From the original dataset containing n number of records, k number of records are chosen randomly for the random forest.

Step II-

An individual tree for each of the samples chosen in step I is constructed

Step III-

An output for the for each individual tree is calculated

Step IV-

For regression, the final output is calculated by averaging all the outputs of step III, while majority voting is considered for classification.

2.5.2.1 Key Characteristics Of Random Forest

- **Immune to the curse of dimensionality:** As this algorithm selects some features instead of taking all of the features, it becomes immune to the curse of dimensionality by reducing the feature space.
- **Diversity:** Each of the constructed trees differs from the others as only some of the features are selected, which makes them diverse.
- **Train-Test split:** There will always be at least 30% of the data not seen by decision trees, so basically, there is no need for splitting test and train data for the random forest algorithm.
- **Parallelization:** Trees are constructed with the help of different attributes and data, which makes them a perfect example of parallel computing and multithreading.
- **Stability:** this algorithm achieves a better result. The result is based upon averaging of majority voting.

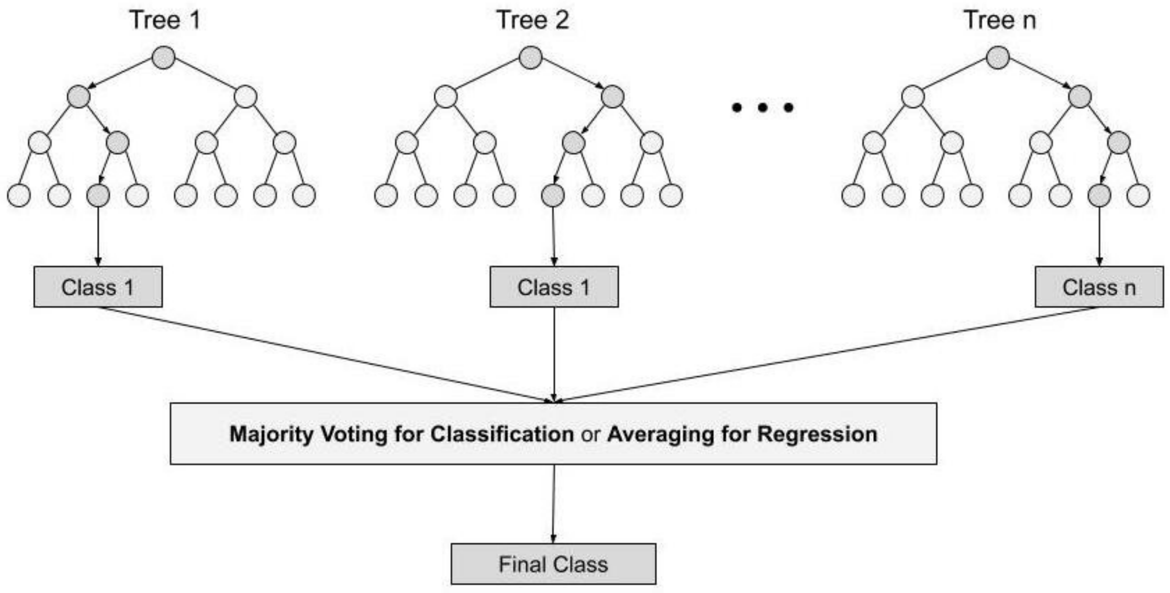

Tree 1
Tree 2
Tree n
Class 1
Class 1
Class n
Majority Voting for Classification or Averaging for Regression
Final Class

Figure 2.5: Random forest using majority voting[Sru21]

2.5.2.2 Key Hyperparameters Of Random Forest

In random forests, hyperparameters are used to either increase the speed of the model or to improve the model's performance and predictive accuracy .

a) **Hyperparameters to increase the predictive power**

- **n_estimators:** Total number of trees the algorithm builds in order to take average or voting from their predictions. A higher number of n_estimators can increase the predicting power and increase the computational time.
- **max_features–:** It determines the maximum number of features// attributes required for any single tree. It is better to stick to the default value for this hyperparameter which calculated by taking the square root of the total number of features present.
- **min_sample_leaf:** After splitting a node, the minimum number of samples that must be present in the leaf node is determined by this hyperparameter.
- **max_depth:** This parameter determines the longest path from the leaf nodes to the root node for every single tree.

b) **Hyperparameters to increase the speed**

- **n_jobs:** It determines the number of computational resources to use. For example, a random forest will employ only two CPU processors if the value is two and so on, but if the value of n_jobs is -1 it uses all available resources.
- **random state–:** Regulates the sample's randomization. when the model is given a specific value of random state and the same training data and the same hyperparameters, the same splits of data will be generated.

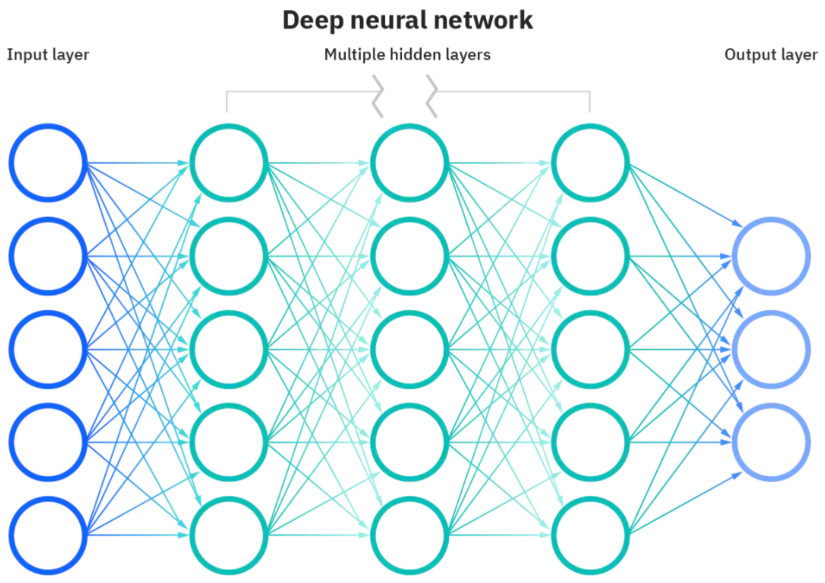


Figure 2.6: Basic architecture showing three different kinds of layers in deep learning models [Kav20]

- **oob score:** Stands for "out of the bag." It is a cross-validation method for the random forest algorithm. One-third of the sample data is separated and put aside. Later on, that chunk is used to evaluate the performance rather than training on it.

2.6 Deep Learning Models

Deep learning algorithms are said to be mathematically challenging and sophisticated progression of machine learning algorithms. It got much of attention recently, and that too for a good reason: the recent advancements have already led to outcomes which were previously unthinkable. These models have produced cutting-edge results in numerous fields, including various NLP applications. For example, deep learning for text and document categorization uses three different deep learning architectures concurrently.

Algorithms that evaluate data with a logic structure comparable to that of a person belong to deep learning. They might use supervised and unsupervised in nature. Deep learning applications use a layered structure of algorithms known as an artificial neural network (ANN). The ANN's architecture is inspired by the biological network of the human brain, resulting in a significantly more competent learning process than typical machine learning algorithms. Consider the ANN from the figure 2.6 as an example. The input layer is the leftmost layer, while the output layer is the rightmost layer. Because their values are not visible in the training set, all intermediate layers are referred to as hidden layers. The real magic of deep neural models lies in these hidden layers, computed values that the network uses. The deeper a network is, the more hidden layers it contains between the output and input layers. A deep neural network is defined as an ANN with at least two or more hidden layers.

Deep learning is now being benefited from in a lot of disciplines. It is used to recognize different items, for example, sign boards, objects, and pedestrians in autonomous driving. The military detects items from the photos or live videos from satellites, for example, they declare safe or risky zones for its personnel with the help of DL. Of course, deep learning is also prevalent in the consumer electronics business. For example, Home assisting systems like Siri , Alexa, Bixby, and Amazon use deep learning algorithms to react to your voice and learn your preferences.

Let us take a more specific example for this subtype of machine learning. Consider Tesla's deep learning system to identify humans, vehicles, and STOP signs in its autonomous driving software. The ANN would detect the an object's essential attributes (usually known as features) in the first stage. In an image, specific structures can be used to represent a feature, such as a point, an edge, or an object. A scientist would have to make handpicked necessary characteristics in a more typical machine learning technique, on the other hand the ANN learns everything on its own and can do so automatically. All the layers learn different things, for example,T the first hidden layer may learn to identify edges, the second how to distinguish colors, and the third might learn to detect more complicated geometrical forms of that object. The other magic behind the success of deep learning algorithms is their ability to learn from their faults. They can efficiently calibrate themselves in case of an excellent forecast or even if they needed to be adjusted after being provided with the training data.

In the case of artificial neural networks (or ANNs), it is a group of neurons with multiple perceptions or inputs in each layer. The network passes information through multiple input nodes in one direction until it reaches the output node. It is the simplest form of neural network. As the information is processed and passed only in one direction, they are also termed feed-forward networks. In this type of neural network, hidden layers are not mandatory but could be included to make their functioning more interpretable. An advantage of ANNs is their ability to store information across an entire network and allow for working with incomplete knowledge. Distributed memory and fault tolerance in ANN are some of the other advantages. Likewise, their disadvantages include huge hardware dependency and occasionally unexplained behavior, which can leave us tormented with results.

One of the most popular models proposed to date for image recognition is Convolutional Neural Network. A variation of multilayer perceptrons is used in CNN and contains any number of pooled convolutional layers or fully connected layers. Convolutional layers create feature maps which record a region from the image. Those regions are further broken down into rectangles and sent out for nonlinear processing, In image recognition problems, they yield very high accuracy. Their capability of sharing weights automatically detecting essential features without any human supervision makes them superior to ANNs. On the other hand, CNN's are not capable of encoding the orientation and position of the object. They also unable to be spatially invariant to the input data, and similarly, a lot of training data is required in order for it to work efficiently.

Recurrent Neural networks (or RNNs) are the more complex algorithm among all we discussed until now. They have been the favorite deep neural network for data mining and text classification tasks. In RNN, after processing the nodes, the output is saved and fed back into the model (hence they don't pass the information in one only direction) to update, which is not a case in feed-forward networks. While continuing the computation and implementation of operations, every node of the RNN model behaves as a memory cell. In the event of an error in the prediction of the network, the system learns by itself, and it continuously attempts to make a correct prediction with the help of back-propagation.

An RNN remembers all of the information over timeline. As it also remembers previous inputs, this model proved to be very fruitful for predicting time series. The effective pixel neighborhood is extended withthe combinations of RNN and convolutional layers. Some of their drawbacks include: They have gradient exploding and gradient vanishing problems. Training an RNN is a challenging task. If the activation function used is tanh or relu, processing long sequences become a limitation.

2.6.1 Long Short Term Memory

In 1997 the long short term memory (LSTM) was first introduced by Hochreiter and Schmidhuber[HS97] and had captured the attention of many researchers so far. LSTM falls under the umbrella of Recurrent Neural Networks (RNNs). Other than the input layer, RNN neurons have a link to the preceding neuron state. RNNs proved to be helpful, especially for the data that can benefit from a contextual view or is sequential. As a result, RNNs may be pretty effective in text categorization. RNN retains all information in the data over time. Earlier, vanilla RNN proved to be a successful addition to deep learning but sooner, an increasingly challenging dilemma while dealing with gradient-based learning algorithms was faced. The issue arose when the time it takes to capture dependencies increased while constructing a model [BSF94].

A memory cell in the architecture of LSTM is used to store information. It computes the input gate, forget gate, and output gate to handle this memory. Over a long distance, LSTM units can transmit an essential feature that appeared early in the input sequence. As a result, possible long-distance dependencies are captured [BCA15]. LSTM is an intelligent solution to the RNN's problem of vanishing and exploding gradient due to which the information was being lost.

Forget gate:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{2.1}$$

Input gate:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{2.2}$$

Candidate:

$$\tilde{C_t} = \tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right) \tag{2.3}$$

Output gate:

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{2.4}$$

Hidden state:

$$h_t = o_t * \tanh C_t \tag{2.5}$$

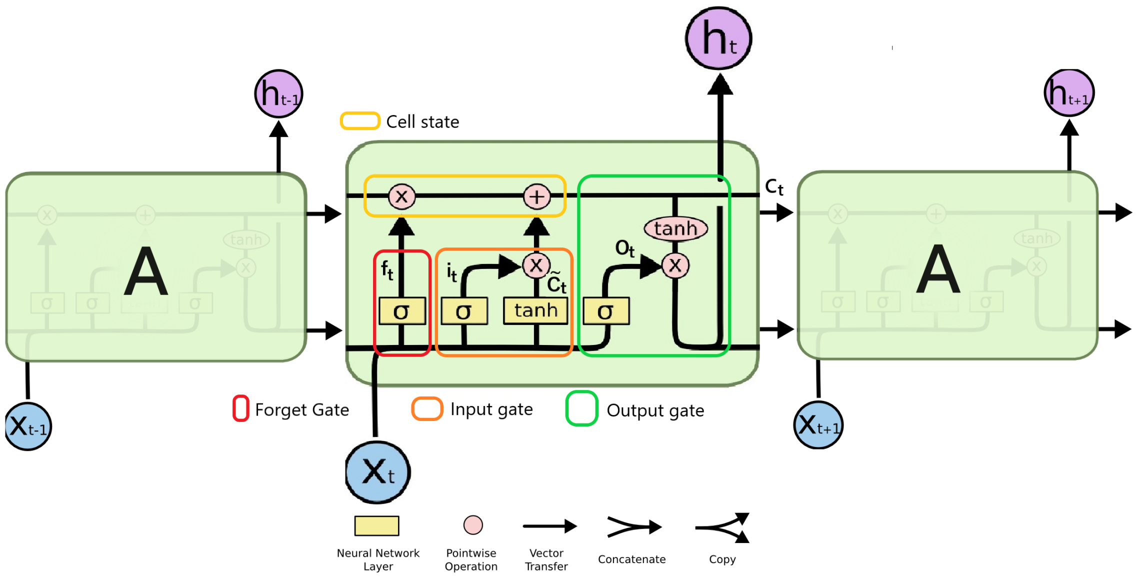


Figure 2.7: The repeating module for sequences in an LSTM contains four interacting layers

Memory state:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C_t} \qquad (2.6)$$

As shown in figure 2.7, there are three gates in the LSTM cell: Input gate, forget gate, and output gate. In the above equation, W is the weight matrix while x_t means input at a particular time t. h_t represents the hidden state at timestamp t, while f, I, C,o refer to forget gate, input gate, memory cell state, and output gate respectively. Each of the gates takes the hidden state and the current input X as inputs. In the forget gate, equation 2.1 calculates if the information is to be forgotten or to be preserved for further utilization. In the input gate, equation 2.2 and equation 2.3 refer to input with sigmoid and candidate with tanh layers in order to decide which information is to be stored. Finally, $\tilde{C_t}$ modifies the cell state with a tanh activation. With a zero-centered range and some operations, Tanh distributes the gradients pretty well, which allows the information of cell state to flow longer avoiding exploding or vanishing. In the output gate, equation 2.4 along with updating the hidden state with equation 2.5 and getting information from the cell stated decides which part of the information to output as a result. As the last step in the LSTM cell, equation 2.6 updates the current cell state for the usage of the next cell as the previous cell state.

2.6.2 Gated Recurrent Unit (GRU)

The Gated recurrent unit (GRU) is a newer type of recurrent neural network that is comparable to an LSTM. Kyunghyun Cho [Cho+14] presented the Gated Recurrent Unit (GRU) in 2014 as a generalized variant of RNN, providing a gating mechanism that deals with both vanishing and exploding gradients, like LSTM. Instead of using the memory cell state to store information, GRU uses a hidden state to transmit information. A GRU, on the other

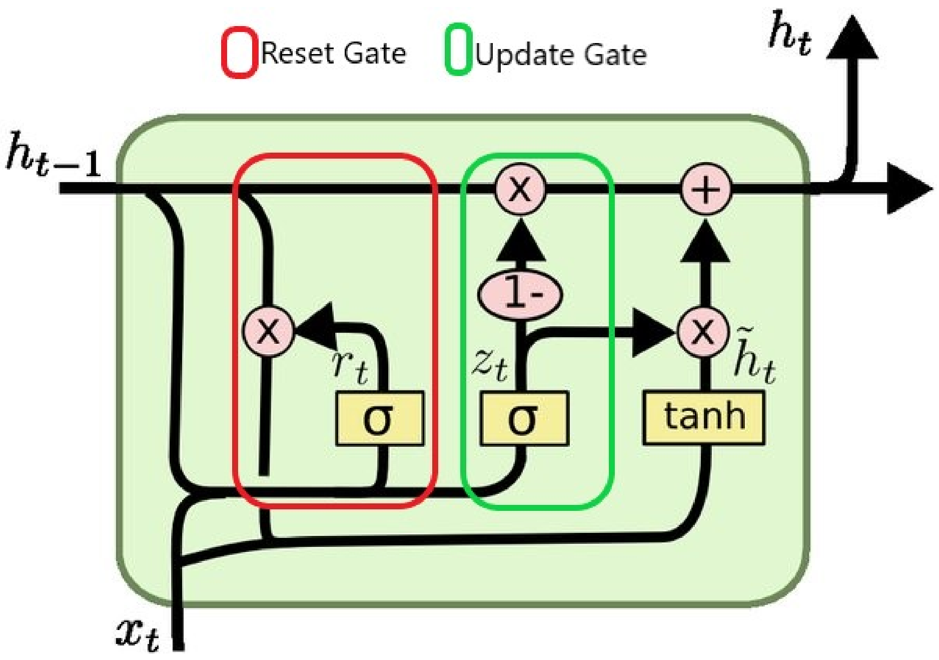


Figure 2.8: Gates and functions of a GRU cell

hand, differs from an LSTM in that it has two gates and no internal memory, as depicted in the figure 2.8. Only two gates are also included, which are a reset gate, and an update, and they are used to govern the flow of data through each hidden unit. The functionality of the update gate is quite similar to that of LSTM's forget input gate. It determines about the information whether it should be inlcuded or discarded, The reset gate is a gate that determines how much information from the past should be erased. The GRU features gating units that influence the flow of information inside the unit, similar to the LSTM unit but without a distinct memory cell. That is how a GRU functions. Due to the fact that GRUs employ fewer tensor operations which in return makes them a bit faster to train than LSTMs. Researchers and engineers typically test both to see which one is best for their desired tasks. The following equations are used to calculate each hidden state at time-step t.

Reset gate:

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{2.7}$$

Update gate:

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{2.8}$$

New memory:

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right) \tag{2.9}$$

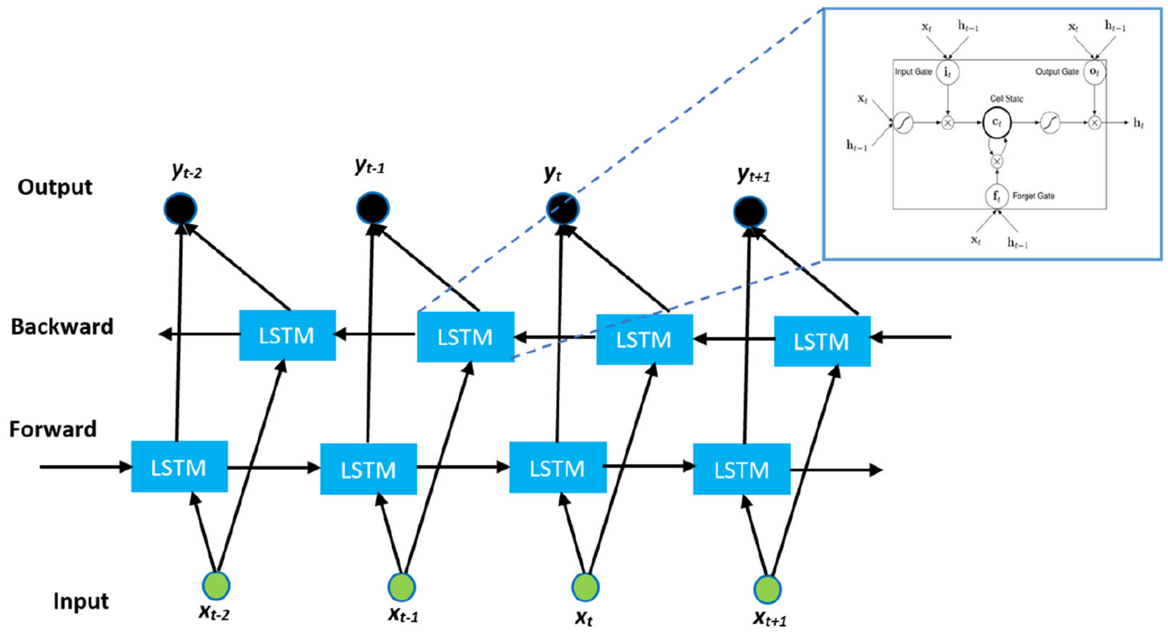


Figure 2.9: Stack of forward lstm and backward lstm layers [RWN+21]

Final memory:

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \tag{2.10}$$

In the equations 2.7, 2.8, 2.9, and 2.10, W represents the weight of that specific input $*$ refers to element wise multiplication. tanh is the tangent function converting the number into values between -1 and 1 while σ is a sigmoid function that converts its inputs between the values of 1 and 0. r_t represents the vector of reset ate and z_t the vector of update gate. h_t refers to final output of a cell at timeperiod of t.

2.6.3 Bi-directional Long Short Term

To enhance the capability and overcome the shortcomings of LSTM, Schuster and Paliwal[SP97] proposed BiLSTM in +1997 shortly after LSTM was proposed. LSTM can predict the next word with the help of past contexts that are the previous words, which was not a very helpful technique in many use cases. To better understand this phenomenon, let us take an example. "He studied in a __". It is challenging for an LSTM model to predict the empty blank by considering all the previous words. On the other hand, "He studied in a __ and after four years graduated as an engineer" now, it has become apparent that the blank can be filled with the word "university". This became possible by considering a word's past and future context under consideration. Hence, biLSTM is simply a bi-directional LSTM that stacks two separate RNNs that capture the information forward and backward. We can say that one works as if it has information about the past, and the other works as if it has information about the future. Having access to both of these information makes it pretty easy to produce way better results.

$$h_t^f = \tanh\left(W_{xh}^f x_t + W_{hh}^f h_{t-1}^f + b_t^f\right) \tag{2.11}$$

$$h_t^b = \tanh\left(W_{xh}^b x_t + W_{hh}^b h_{t+1}^b + b_t^b\right) \tag{2.12}$$

$$y_t = W_{hy}^b h_t^b + W_{hy}^b h_t^f + b_y \tag{2.13}$$

Equation 2.11 shows the hidden state in the forward direction while equation 2.12 calculates the hidden state in backward direction. Moreover finally, the output is generated by concatenating both of the hidden states as shown in 2.13. BiLSTM produces way better results in the tasks where context is important as compared to simple LSTMs, but the computation speed is increased as the number of hidden cells is doubled.

2.6.4 Bi-Gated Recurrent Unit (biGRU)

The concept behind bidirectional gated recurrent units is similar to that of biLSTMs, that is, taking advantage of bidirectional context awareness to produce better results. First, two layers of GRU are stacked upon each other to capture the past and future context of the text. One layer captures the context in the forward direction while the other captures it backward. Then both of these are concatenated to produce the output.

$$\overleftarrow{h_t} = GRU(x_t, \overleftarrow{h}_{t-1}) \tag{2.14}$$

$$\overrightarrow{h_t} = GRU(x_t, \overrightarrow{h}_{t-1}) \tag{2.15}$$

$$h_t = (\overleftarrow{h}_t, \overrightarrow{h}_t) \tag{2.16}$$

Equation 2.14 calculate the hidden state values from right to left direction while equation 2.15 calculated in the opposite direction. Later on, both of the values for hidden states are concatenated in order to generate the output values. The architecture of bi-directionality can be understood with the help of the figure of biLSTM 2.9 as the architecture is the same, but the cells are GRU instead of LSTM. BiGRU is preferred over BiLSTM as the former is an enhanced variant of later by merging the input gate and forget gate into one gate, which is the reset gate. Similarly, biGRU does not contain special memory state cells but instead manages them only with the help of hidden states, which makes it more robust, resource-efficient, and speedy.

2.7 Importance Of Pretraining

Pretraining is an easy task that is carried out only once and can be described as it prevents us from a very common problem of reinventing the wheel and instead offers us learned parameters to invest time and energy to enhance them further. Learned weights can be used for various NLP tasks in the future with just simple fine-tuning, saving a great deal of time and resources. By using this technique, current state-of-the-art NLP models have become able to perform better in many NLP tasks. An unsupervised pre-training is carried out using unlabeled text as input to train a Neural Network to learn common language features, which can be later reused for a supervised NLP task. As described in their semi-supervised sequence learning paper [ML15], the authors developed two distinct ways of training RNNs with unlabelled data. Traditional language models aim to predict the next word, which is the first strategy. In the second method, a sequence is read into a vector and then predicted with the help of sequence autoencoding. These two strategies make it possible to pre-train a Neural Network to learn parameters unsupervised, which it can then apply to future supervised learning tasks.

The unlabelled text is abundant in the shape of news, books, online articles, scripts, and blogs compared to the very scarcity of labeled text in specific domains. The availability of unlabelled text makes it easier to pre-train on an enormous amount of data. The more it is trained on data, the better results it will generate later. OpenAI's GPT merged the notion of unsupervised pre-training with Transformer Network [Ash+17] to gain SOTA results on different tasks of natural language processing. This idea of unsupervised pre-training is quite similar to that of computer vision's pre-training, which achieved far better results after pre-training on various unlabeled pictures of different objects and creatures. The GPT's results showed that it was a great success in different NLP tasks. A few examples of the most popular models are GPT, BERT, XLNet, and GPT Two were trained on 0.8 billion, 3.3 billion, 32.89 billion, and 40 billion tokens respectively.

2.8 The Era of Transformer

In 2014 Bahdanau, along with his fellow researchers, proposed an attention mechanism in their work [BCB14]. Inspired by their proposed attention mechanism, Vaswani and a few other researchers from Google, in their paper [Ash+17] "attention is all you need" proposed a new neural network model which was based totally on the attention mechanism. This new proposed model was named transformers. They were also a better option to make use of the GPU efficiently due to their ability to operate in parallel. Soon researchers started to prioritize transformers over the previous best performing model BiLSTM. The transformers were able to read the whole sequence at once after getting rid of convolution and recurrence. Their ability to read the whole sequence at once made them the first choice for models like BERT, which is explained in a later section, as they get better context awareness by learning bi-directional context at once.

2.8.1 The Architecture Of A Transformer

Unlike LSTM, GRU, and other RNNs that use recursion and backpropagation, the Transformer relies entirely on the Attention Mechanism. As illustrated in Figure 2.1, the transformer model is made up of two parts for executing seq2seq tasks: an encoder and a decoder block. The encoder-decoder architecture is similar to that of prior sequence to sequence models such as LSTM. However, unlike LSTM, the mechanism used is self-attention rather than recursions. An encoder takes the text in the shape of a sequence of words as inputs along with the position encoding. It generates a numerical representation of each word, also known as feature vectors. The feature vector of any word contains the information about its position and surroundings along with the information of that specific word. An encoder can also be used as a standalone model, and the most famous encoder model is BERT. Similarly, a decoder can also be used as a standalone model. The example is t5. The encoder transfers its generated output i.e. numerical representation to the decoder. The decoder accepts the representation generated by the encoder and the normal input sequence and positional encoding. Then it decodes to generate an initial word. The model has feature vectors from the encoder and an initial word decoded by the decoder by this time. The encoder is not required any further. From here on, the decoder can work auto-regressive. Using the features vectors and initially generated word, it can keep on outputting the word until a stopping criterion is met. Positional encoding is a method for maintaining the order in which sequences are given to a transformer model, and it is explained briefly in the 2.8.3 section.

As described in Section 2.8.2, the Self-Attention Mechanism is used by both Encoders and Decoders in the Transformer Model. The Encoder block contains multiple stacked encoder layers, which are shown by Nx as depicted in fig. Each of these layers tries to discover different attributes of the NLP pipeline. For example, the first layer might be capturing the partsofspeech(POS) tags. The second layer might be capturing semantic roles. The third might capture dependencies and relationships, and so on. Each layer of the encoder contains a Multi-Head Self-Attention sub-layer, followed by the residual learning part and normalization, which is further followed by a fully connected feed-forward network and both the residual learning and normalization. Add, and the normalization layer helps traverse the information for a greater length. As shown on the right side of Figure 2.10, the decoder (just like an encoder) is a stack of multiple decoder layers represented by Nx. In comparison to the encoder, each decoder layer contains an extra multi-head attention sub-layer in addition to the two encoder sub-layers. This additional sub-layer is utilized to compute attention based on the left block's output.

Each sub-layer of the decoder is responsible for residual learning and normalization. In addition to the encoded input sequence from the encoder, the decoder takes the predicted output sequence as input. Multi-Head Attention sub-layers in decoders are masked to stop the decoder from attending to the output sequences that have not yet been predicted. Only the predicted output is visible at any given time. After applying the Softmax function, a prediction is made by the Transformer Model that the next token predicted by the decoder will be based on the Output Probabilities for the following tokens.

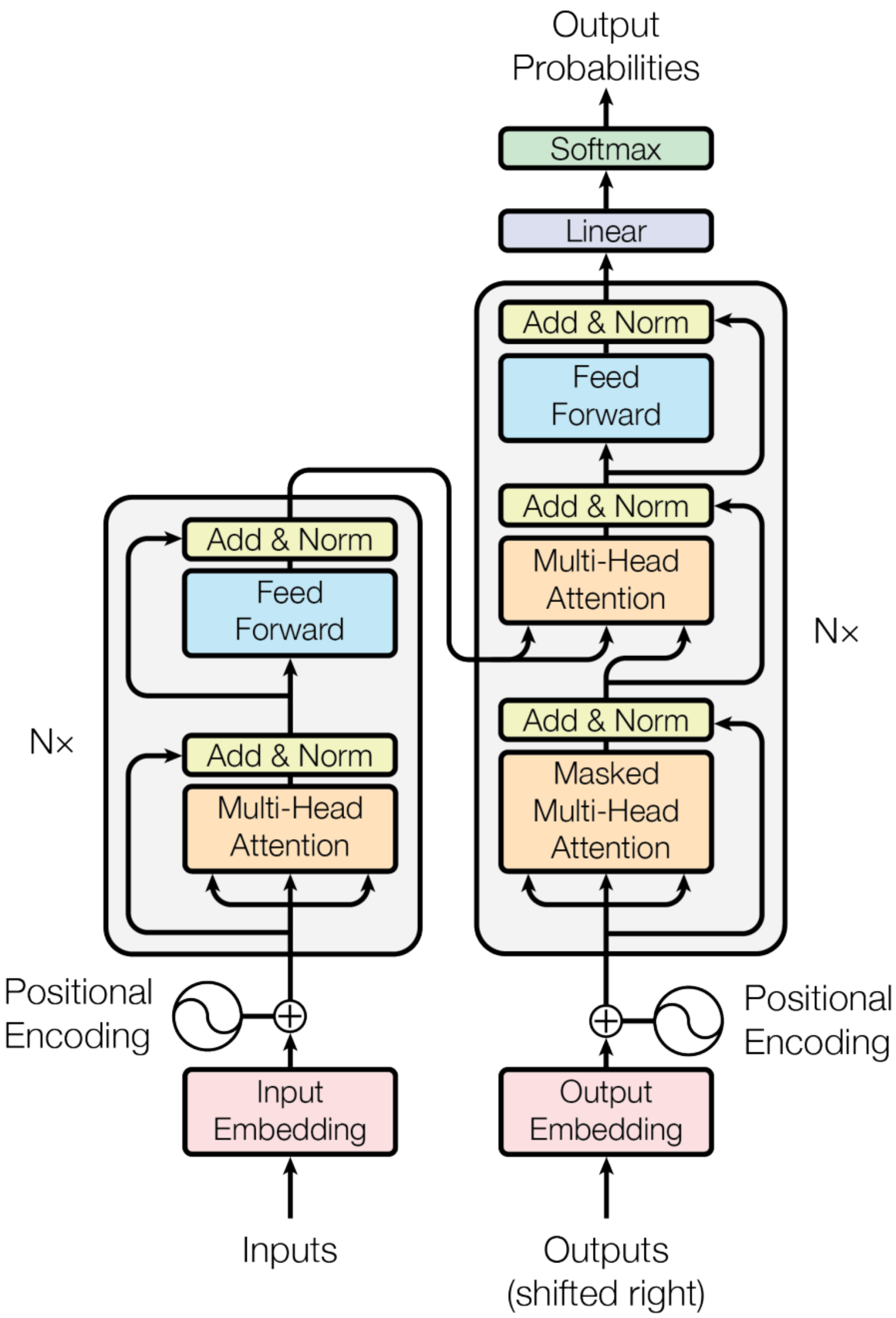


Figure 2.10: The architecture of transformer [Ash+17]

2.8.2 Self Attention

Attention and self-attention are closely related topics as the latter was driven by the former. The attention made its way into NLP in 2015, one year after being proposed. It revolutionized the concept of translation into other languages by machines. For example, let us take the sentence" The Indus water treaty was signed in 1960. It was agreed upon between Pakistan and India". One way to translate this sentence from English to German will be to translate one word at a time into its equivalent word in German, but that is going to do a lousy translation. As in German, the order of words to make a sentence is different from English. Moreover, in German, every noun is supposed to be masculine, feminine, or neutral, and the surrounding words are affected by the gender of the nouns in that sentence. The earlier translation was being done with the help of this sequential technique through RNNs. In translation, attention helps by looking at all the words in the sentence before making any decision. In their paper [BCB14] authors showed which words were making an impact on the translated words with the help of heat maps. Models learned which words to pay attention to while translating any word with the help of excessive training on the vast training data. They learned the words' gender, plurality, grammar, and structure of the languages they were trained on. Earlier attention was being used with RNN, but after the paper "Attention is all you need" LSTM became a past with the advent of transformers.

On the other hand, if we want our model to learn semantics and other patterns in the language, we need self-attention instead of attention. For example, in computer vision, neural networks did a fantastic job by learning different parameters of the objects, for example, recognizing shapes, edges, and even detailed and tiny structures. If the internal representation is learned better, the results produced in any related tasks are also better. The same is the case here. If the model is well aware of the internal representation, structures, grammar, and patterns, it is expected to perform accordingly. Let us take two sentences.

1- Server and I met today, and it was a pleasant meeting.

2- The server could not pass the load test and failed to respond in time due to massive traffic.

We have "Server" as a common word, which means different in both sentences. We being humans, can easily differentiate between both of the meanings with the help of their context, but for a machine, it is not an easy task. Nevertheless, with the help of Self-Attention, it becomes just another task for the machine to understand the different meanings of the same word due to their surrounding words. For example, a self-attention model might attend to the words met and meet to process "Server" as a human. In contrast, in the second sentence, it might attend to load-test and respond to process "Server" as a physical IT server different from a human. Recurrence over the hidden states of typical recurrent neural networks in long sequences loses part of the information. The authors suggested using an attention mechanism instead. It makes the model quite fast as we do not have to perform backpropagation which is done in several steps. The encoder takes the whole input sequence and reads it in one go. The decoder gives the probabilities of the following potential word with the help of the softmax function in a single iteration. Attention is beneficial as it provides parallel computing to save time and is also a preferred method to benefit from the parallel computation of TPUs and GPUs.

As described earlier, the attention mechanism helps the model select the words to pay more

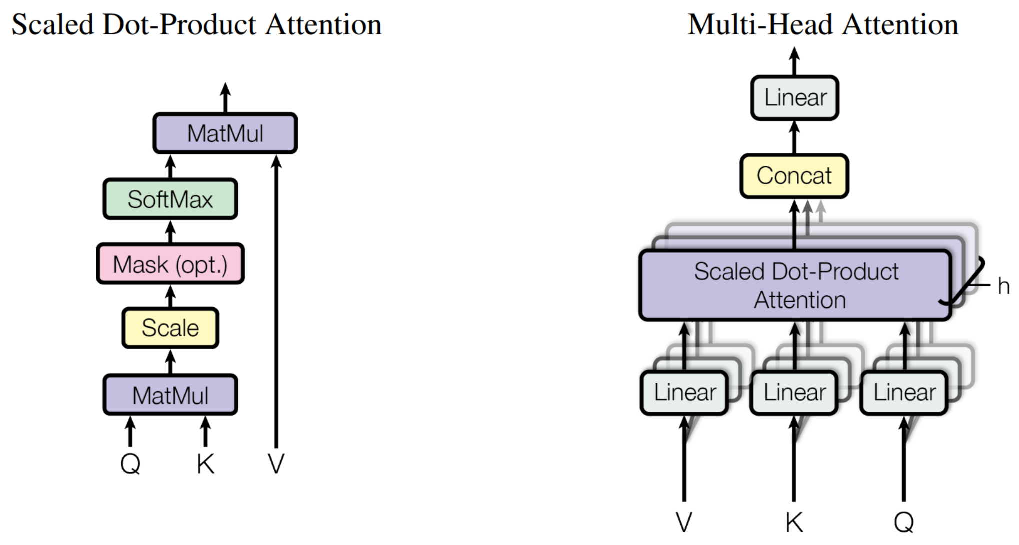


Figure 2.11: Scaled dot-product attention and multihead attention of transformer [Ash+17]

attention to while making any prediction. How does it work is described as follows. The first step is to create three vectors K, V, and Q(key, value, and query) for each input provided to the encoder. These vectors are created by multiplying the input sequence with the weighting matrices. Weighting matrices are learned in the training phase and are different for each encoder and decoder layer. K and V vectors are generated in the encoder block, while the Q vector is generated in the first multi-head attention layer of the decoder block from the output it has so far. These three vectors are transferred to the next multi-head attention sub-layer within the decoder. In the second step, scores are calculated from these three vectors. For a single sentence, let us take the first word. We have to score all other words in the sentence against the first word of that sentence. This score decides the amount of emphasis given to other words in the sentence while encoding the first word and so on. This score is calculated with the help of the dot product of Key vector K and Value vector V of all the words. Among many other attention mechanisms, the authors of the transformer chose a scaled dot product because of its ability to utilize the least resources, such as time and memory. After taking the dot product, a scaling factor is used before finding the probabilities of the output, which is why it became famous with the term scaled dot product. If we calculate the score of the second word in the sentence concerning the first word, it will be calculated by the dot product of Q1 and K2. The third step is to scale the result of the dot product with a factor of $\sqrt{d_k}$. In the next step, softmax is taken, and here the value of the scaling factor becomes evident because otherwise, it could suffer from the vanishing gradient problem. In the fifth step, to increase the score of relevant words and deplete all the others, the softmax probabilities are multiplied with vector V. In the last step, the weighted values are summed up to output a vector from the self-attention layer for the first word. All of the steps until here can be summarised with the equation2.17 from [Ash+17]. Any of the two described ways in their paper can be used to sum up the multi-head attention layer to produce an output vector of this layer which can then be forwarded to the feed-forward neural network.

$$\text{Attention(K,V,Q)} = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.17}$$

2.8.3 Positional Encoding

As shown in the figure, both the encoder and decoder blocks use positional encoding just after taking inputs. Unlike RNN, which works on sequential data, the transformers are based on an attention mechanism with no concept of sequence or timestamps. Hence we have to introduce positional encoding to remember the order of the input as the attention mechanism is a feed-forward layer that processes the whole input in parallel. The concept was proposed initially in [Geh+17] by the name of position embedding but later became a significant part of transformers. It is basically a tensor vector of the size same as the input and contains information about the relative distances of the words. A word vector is generated from the embedding by applying the positional encoding and containing the location information about the sentence, called the "context".

Authors of [Ash+17] used absolute positional encoding with the help of sine and cosine functions which are arguments for reducing bias over one hot encoding technique. Later, when the bias was still found, further research from [SUV18] proposed relative positional encoding in 2018. The explicit representations of relative position in the self-attention mechanism are used, which increases the performance of the architecture of the transformer dramatically. By introducing tiny changes such as adding the new vector to the dot product of K and V matrices with the input, relative positional encoding was generated. Refer to their paper for better understanding. In terms of concept, relative positional encoding permits self-attention to examine arbitrary relationships between any two words of the provided input. A vital benefit of this approach is that a token's position in a sequence is not based on its absolute position but only on its relative distance from another token. Consequently, attention should be more robustly generalized beyond the length of sequences in training. He proved the supremacy of this technique by producing way better results as compared to using the absolute positional encoding.

2.9 Bidirectional Encoder Representations from Transformers Model

Bidirectional Encoder Representations from Transformers, more famous by their abbreviated name "BERT", is a breakthrough deep learning language model that was introduced in the 4th quarter of 2018 by google researchers in their paper [Dev+18]. It is an open-source ML framework that can make computers intelligent enough to understand ambiguous words by a contextual understanding of the text. It is based on the transformer, which is a relatively newer deep learning model that links all the output elements with all of the input elements to have a better understanding of the text. Earlier language models used representations that were learned sequentially in one direction, either forward (left-to-right) or backward (right-to-left), depending on the direction in which they learned the context. However, as evident from

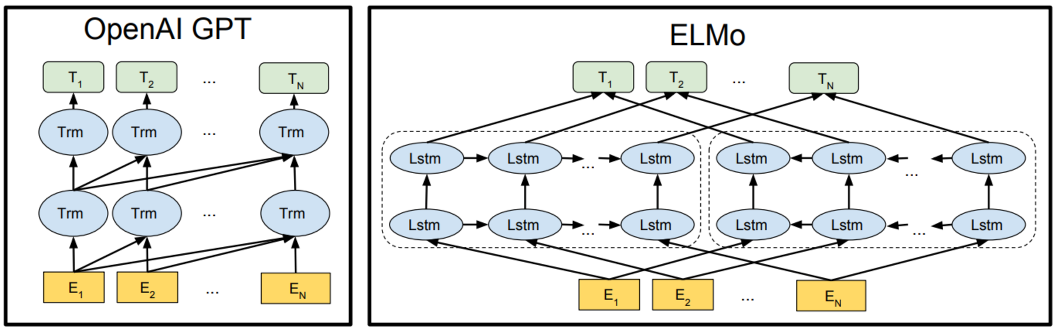


Figure 2.12: Pretraining architecture of GPT(Unidirectional) and ELMo(shallow bi-directional) [Dev+18]

the name, the representations in BERT are learned in a bi-directional way and are captured at once. This ability makes any words well aware of all of the surrounding words.

In order to meet the needs of a wide variety of Natural Language Processing problems, BERT utilizes advanced Language Modeling techniques to produce SOTA results. Its fundamental design is that of Transformer [Ash+17], a modern Neural Network. BERT additionally incorporates the usage of Transfer Learning by performing unsupervised pretraining on plain text. Simple finetuning could be used to apply the learning obtained during the pretraining process to different Natural Language Processing tasks. Among OpenAI's famous models, GPT is unidirectional, and later on, GPT-2 and GPT-3 were also released. A shallow bidirectional model is produced by concatenating 2 LSTMs trained separately for left-to-right and right-to-left directions. The pretraining architecture of GPT and ELMo are shown in the figure 2.12. In contrast, BERT is the model that learns bidirectional deep context concurrently.

2.9.1 Architecture of BERT

The transformer, as explained in section 2.8.1, consists of 2 blocks an encoder and a decoder. The encoder block takes the input and processes it into some representations. The Decoder block predicts the output based on further processing of those representations. We aim to construct a Language Model(LM), so we only need the encoder component. Because of this, BERT makes use of only the encoder block of the Transformer Neural Network as a core component. BERT consists of a stack of Transformer Encoders with several layers. There are two different configurations of pre-trained BERT Models presented in [Dev+18]: $BERT_{LARGE}$ (24 layers, 1024 hidden states, and 16 Attention Heads) and $BERT_{BASE}$ (12 layers, 768 hidden states, 12 Attention Heads). Layers are the number of encoders used in the stack. $BERT_{LARGE}$ and $BERT_{BASE}$ both have 340M, and 110M parameters learned, respectively.

An illustration of the flow of word information in the BERT Model's pretraining architecture is shown in Figure 2.13. The input information starts from the first layer containing embeddings

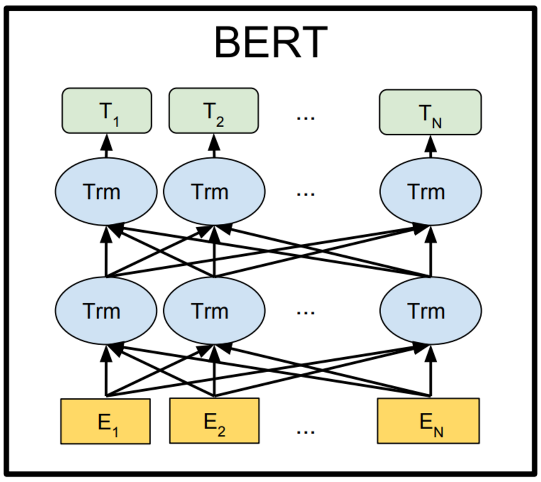


Figure 2.13: Pretraining architecture of BERT(deep bi-directional) [Dev+18]

of the input. Then, it is transmitted to all the intermediate layers depicted as "Trm" until the last layer produces the output T_1,T_2,..T_N. A complex attention algorithm computes multi-head attention on the word representation from the preceding layer to generate intermediate states that are equivalent in size. Each word representation is learned by looking at its past and future context, as shown by the arrows in Figure 2.13. For comparison, $BERT_{BASE}$ was selected to have the exact same model size as OpenAI GPT [Dev+18].

2.9.2 Pretraining of BERT

Pretraining is a time-consuming and costly process, but it is usually done only once. The procedure of pre-training and its importance is described in section 2.7. A considerable quantity of plain text is used for unsupervised training of an artificial neural network with particular pre-training objectives. In the creation of BERT, several Encoder layers were used as described in Section 2.9.1. It was pre-trained using around one million training steps over a very lengthy period of time. $BERT_{BASE}$ is pre-trained on 800 million words of plain text from the BookCorpus, while $BERT_{LARGE}$ is pre-trained on 2500 million words of text collected from English Wikipedia, both of which are unlabelled text. The two unsupervised tasks BERT pre-training are Masked Language Modeling(MLM) and next sentence prediction(NSP), as described in the section 2.9.3.

Pre-training in the BERT Model is fairly a resource-demanding job. It might take up to four days on Google's TPUs. Standard BERT models are pre-trained on the texts of multiple different domains, and usually, they are sufficient and can produce the best results in the majority of tasks. In contrast to just relaying on the pre-trained standard BERT model, additional pre-training using domain-specific corpora for a specific task can help boost performance in the domain-specific tasks. For example, the plain English text of 3.3 billion tokens is used to train English BERT. The implementation for performing both BERT tasks to execute pre-training of any of the available BERT Models is already provided. There are two stages to the pre-training process:

a) **Data creation for Pretraining**[1]

   The data from plain text files is transformed into the TFRecord format. TensorFlow's TFRecord is used to save sequences of binary records.

b) **Run Pretraining**[2]

   Pretraining of BERT is run on both its tasks such as MLM and NSP. The TFRecord format data generated in the step (a) is used here to run pre-training.

2.9.3 Tasks of BERT

Pretraining is done with specific objectives, which can also be called tasks. BERT is pre-trained in a deep bi-directional way which is achieved with the help of two unsupervised tasks i.e. masked language modeling and next sentence prediction.

2.9.3.1 Masked Language Modeling

The concept of masking originated from Taylor's paper [Tay53] termed as cloze task way back in 1953. Bi-directional models are considered to be Superior to shallow bi-directional and uni-directional models. The traditional language models used uni-directional context. In BERT, masking is a method of randomly masking a small proportion of tokens with a special token MASK and later on predicting those masked tokens based on their bi-directional context. Randomly selected tokens of input are masked, which helps achieve bi-directional context simultaneously in the pre-training of the BERT model. In this task of pre-training, BERT converts the randomly selected 15% of tokens to the specific token `[MASK]`. Then, to predict those tokens, BERT learns the bi-directional context of the text without using expensive training methods. An example from the home-improvement dataset is provided below to understand the masked language modeling.

Input Sentence: The electric power line has three wires: live, neutral, and earth.

Masked Input Sentence: The electric `[MASK]` line has three wires : live , `[MASK]` and earth .

During this task of pre-training, a problem was faced. The masked tokens were never seen in the fine-tuning to BERT, making a mismatch. To get rid of this problem, BERT made it biased. After experimentation, the best strategy to make the data biased was: Choose 15% tokens randomly, convert 80% of the chosen token to `[MASK]`, convert 10% of the chosen words to some random tokens, and keep the 10% of the chosen tokens unchanged.

2.9.3.2 Next Sentence Prediction(NSP)

Many NLP tasks related to natural language inference and question answering sentence entailment are strongly dependent on a better understanding of relationships between sentences. Masked language modeling captures the relationships between words while the next sentence

[1] `https://github.com/google-research/bert/blob/master/create_pretraining_data.py`
[2] `https://github.com/google-research/bert/blob/master/run_pretraining.py`

prediction understands the relationship between sentences and helps BERT learn them. For example, NSP is provided with a pair of sentences, and its task is to determine if the second sentence follows the first sentence or not, based on their semantic properties.

In the pre-training phase, sentence pairs are generated from the corpus so that in 50% of the pairs, sentence II comes after sentence I in the corpus and is given the label "IsNext". Whereas for the rest of 50% pairs, sentence II of the pair is randomly selected from the corpus providing the label "NotNext".2.9.5 will explain how three different kinds of input representations are used in BERT and how will BERT be able to differentiate between sentences. As of now, `[SEP]` will show the ending of any sentence while `[CLS]` that is inserted at the beginning of the input representation will be used to make the prediction for the pair. Experiments from [Dev+18] show that results were worse in all of the tasks where NSP was not used.

2.9.4 Tokenization in BERT

In the BERT model, embeddings are made by using end-to-end wordPiece tokenization and vocabulary, which already has around 30,000 pre-defined tokens. This algorithm is based on LinMaxMatch. Word2Vec and glove had the problem of out-of-vocabulary, which can be efficiently solved by using the subword segmentation algorithm of wordPiece. This technique pre-tokenizes the text based on white spaces and punctuations into words. If there is any new or unknown word, instead of converting it to null or out/-of/-vocab, the workpiece further splits them into sub-pieces until all known vocab words are discovered from that unknown word, and the rest of the pieces are converted.

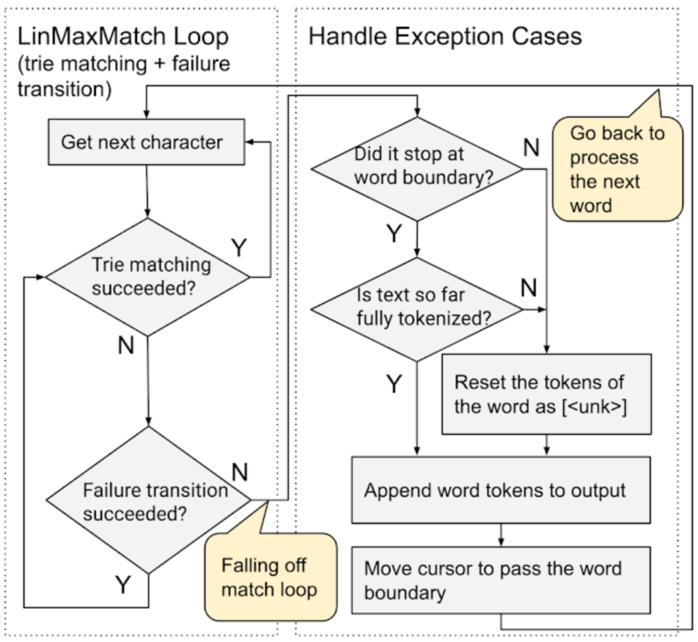

LinMaxMatch Loop
(trie matching + failure transition)
Handle Exception Cases
Get next character
Trie matching succeeded?
Y
N
Failure transition succeeded?
N
Y
Falling off match loop
Did it stop at word boundary?
N
Go back to process the next word
Y
Is text so far fully tokenized?
N
Y
Reset the tokens of the word as [<unk>]
Append word tokens to output
Move cursor to pass the word boundary

Figure 2.14: Flowchart of end-to-end wordPiece tokenization [Xin21]

To produce the input embeddings, the implementation of end-to-end tokenization and encoding can be found on [3]. An example from the home-improvement dataset is provided below to understand the end-to-end wordPiece tokenization used in BERT.

Input Text: I grounded the red wire by mistake for earthing hahahu

Tokenized Input: I grounded the red wire by mistake fot earth ##ing haha ##hu

Encoded Input: 101 1046 14045 2006 24009 28430 2056 17398 4560 8420 28904 7953 28986 102

In the above example, a token "earthing" is neither a regular word from the dictionary nor is it present in the vocabulary of wordPiece. Hence it is split into two tokens, earth, and ##ing both of which are present in wordPiece vocabulary. The same goes for the word "hahahu". We can see two additional tokens `[101]` and `[102]`. BERT uses two additional `[CLS]` and `[SEP]`. `[CLS]` is encoded to `[101]` and is used at the beginning of input as the very first token. Similarly `[SEP]` is encoded to `[102]` and is used after the last token of a sentence. This token helps to separate different sentences from one another in a paragraph. End-to-end wordPiece tokenization is a combination of pre-tokenization and WordPiece that converts it into a single linear-time pass, as shown in the flowchart diagram 2.14. It reduces the computational speed along with the model latency while boosting up the tokenization process, making it 8x faster than the previous tokenizers[Xin21].

2.9.5 Input Representation

BERT is able to represent both the single sentence as well as sentence pair unambiguously in a single token which makes it suitable for various NLP tasks. A sentence is considered to be a single input sequence which might be a single regular sentence or a pair of sentences in the case of NLI tasks [Dev+18]. Input sequences can be up to 512 tokens in length but not more than it. As described earlier `[CLS]` is used at the beginning of the sentence in both the single and sentence pair tasks. In the former, it is initialized with a `[CLS]` token and then the input tokens, and at the end, a `[SEP]` token. In the latter, after initializing with a `[CLS]` token, input tokens from the first sentence are placed, then comes the `[SEP]` token, followed by the tokens from the second sentence with one more `[SEP]` token at the end. In contrast to other models, BERT uses more than one embedding layer in the shape of token embeddings, segment embeddings, and position embeddings which can be seen in the figure. Moreover, 15% of the tokens are masked as discussed in 2.9.3.1, so input representation satisfies the pre-requisites for both of the tasks of pre-training.

[3]`https://github.com/google-research/bert/blob/master/tokenization.py`

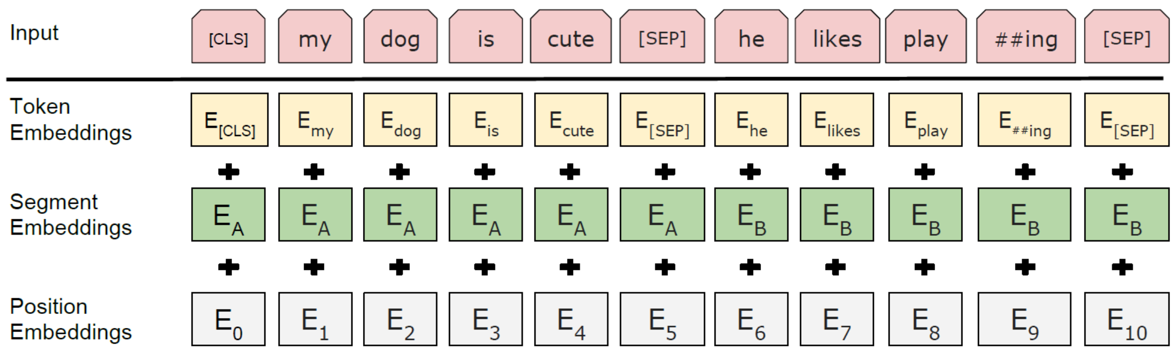

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} $E_{\#\#ing}$ $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

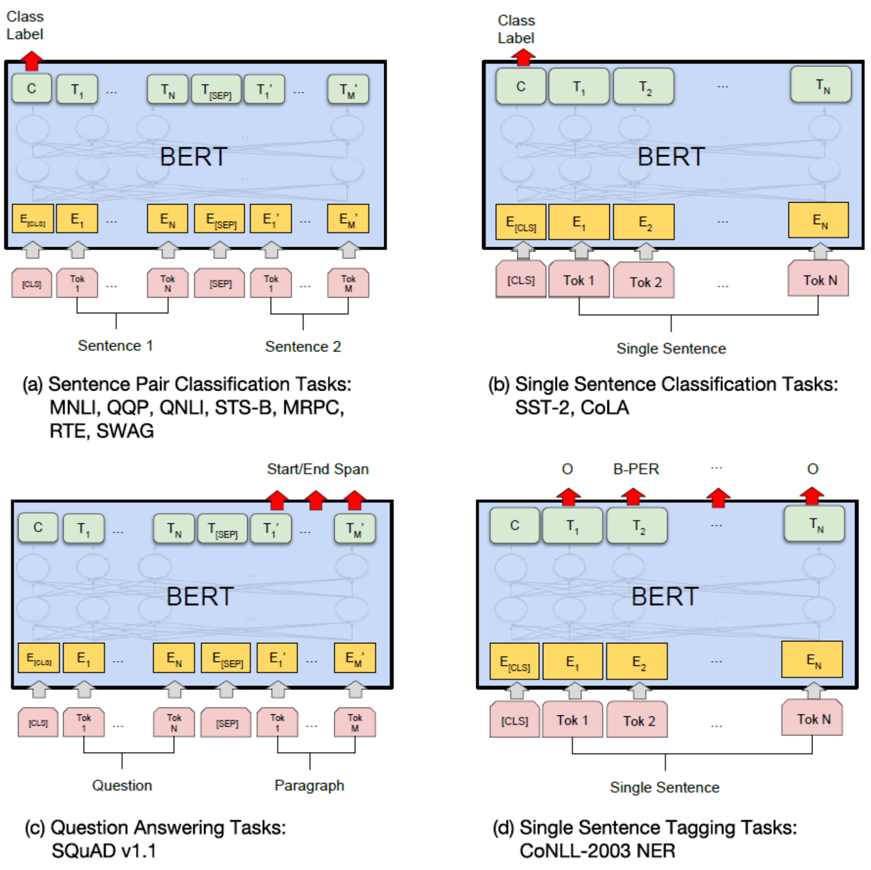
Figure 2.15: Three embedding layers used in BERT, from[Dev+18]

The token embedding layer contains vectors of the tokens and special tokens. Each vector has a dimension of 768, a specific number used in BERT. So after getting the tokens from the wordPiece tokenizer, they are converted to 768-dimensional vectors represented in the token embedding layer. Segment embeddings give distinct embeddings to both of the sentences' tokens i.e. "0" to all the tokens from the first sentence while "1" to all the tokens from the second sentence. Similarly, positional embedding layers gives distinct position to every token of the whole input sequence. All three layers have 768-dimensional vectors for every single token. The visualization of BERT's input representation is shown in the figure 2.15. All three layers add up to generate the input Embeddings. `[CLS]` is used in the input representation for prediction in classification tasks during finetuning of BERT.

2.9.6 Finetuning the model

We can use BERT for ant specified tasks in three ways i.e. building BERT according to our tasks, using pre-trained models, or using pre-trained models with fine-tuning. The first choice demands many resources, while the second is very resource-efficient as the model is already pre-trained on massive data. In contrast, the third choice is the trade-off between a bit more resources and higher accuracy. Hyperparameters are usually kept the same as pre-training in fine-tuning except for three. Epochs, the learning rate of the optimizer, and batch size could be changed according to our data to increase the accuracy further. It is observed that the effect of hyperparameter tuning is directly proportional to the size of the dataset. Fine-tuning is a speedy process, so implementing an exhaustive search to find the best parameters for our dataset should not be an issue.

Fine-tuning can be carried out for several tasks, as shown in the figure 2.16. The single sentence classification task is shown as (b). BERT layers in the input representation are combined to give an output representation. The hidden state in `[CLS]` contains information about classification, which is transmitted to the classification layer in the shape of aggregated sequence representation. Softmax activation is used to find the probabilities of each class at the end. Sentence pair classification tasks are shown in figure 2.16 as (a). Natural language inference tasks are suitable for this kind of fine-tuning. The procedure is quite similar to that



Class Label
C
T_1
T_N
T_[SEP]
T_1'
T_M'
BERT
E_[CLS]
E_1
E_N
E_[SEP]
E_1'
E_M'
[CLS]
Tok 1
Tok N
[SEP]
Tok 1
Tok M
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
T_2
Tok 2
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Figure 2.16: Finetuning BERT for different kind of tasks. Taken from original paper,[Dev+18]

of (a), except it has a pair of texts which is separated by `[SEP]` token. Again `[CLS]` is used to make classification. In contrast, BERT follows the procedure shown in figure (d) to perform token classifications such as Named entity recognition tasks. Here instead of transmitting `[CLS]`, the hidden state of every token is transmitted to the classification layer. If a token was split into multiple parts in a word tokenizer, only one of them(the first one) is transmitted to be classified. The fourth type of fine-tuning is for question answering tasks, as shown in the (c)bottom left of the figure 2.16. It follows a slightly different procedure and learns "start vector" and "end vector" to find the answer from the paragraph for the given question.

3 Related Work

Text similarity has become a hot topic, and a lot of researchers have been investing their time in this domain. During the early periods of text similarity, it centered around FAQ databases which the service provider already answered to save time for both parties. Different techniques were being used to optimize FAQ(frequently asked questions). Song, Feng, and Wenyin [Son+07] proposed similarity as a combination of both semantic similarity and statistic similarity. They used cosine-product to implement statistic similarity and Wordnet to see the distance between words to calculate semantic similarity. They showed that when using the combination of both static and semantic similarity, the results were improved compared to using either of them. The main disadvantage of using WordNet is that words have a limitation, and they can only be compared with the same type of words i.e. nouns with nouns and verbs with verbs.

Later on, a few metrics(HSO, LESK) were proposed where words could be compared with other types, but they were very slow. Finally, Milhai and Vasile [LR12] identified another problem with WordNet, which is Word Sense Disambiguation(WSD), where finding a correct sense of the word in the text is still a difficult task as there are a lot of senses of the same comment. Milhai and Vasile constructed pair of words and then computed the overall similarity through a weighted sum. Finally, they normalized the output with a weighted input text length. By experiments, they proved that using the weights and normalizing increased efficiency for almost all the WordNet metrics they used. The problem of WSD can be overcome to some extent by always using the first sense of the word.

Pawar Hingmire and Palshikar[PHP14] produced their algorithm by combining Jaccard similarity along with context similarity. Context similarity was obtained by multiplying similarity with a common context factor. In this way, when they used LMsim(Jaccard similarity + context similarity), they overcame both similarities' limitations as smaller common context results in high Jaccard. Still, lower context similarity and many co-occurring words will produce higher context similarity but lower Jaccard similarity. Sharma [Sha+19] compared trigrams with many linear and neural network models to find matching questions in the quora question pairs dataset and found that Neural network models outperformed linear models and, more specifically, Continuous Bag of Words got the best results among all.

Author of paper [Kam+18] made a model for finding the most similar questions to a new question for any QA website. They tried to implement the other popular models proposed until that time to make a comparison between their model and the others. They named their model 1D-SLcQA, which was composed of twin LSTM neural networks, one for the question and the other for the potentially related question. They used glove embeddings, and after making the LSTM matrix, a 1D convolutional layer was used along with max-pooling and ReLU activation function. Then after applying a fully connected layer, they used the softmax function to predict the probabilities. Their results suggested that translation models performed better than baseline methods but worse than topic-based methods. Deep learning models with parameter sharing proved to be more beneficial than those without sharing. The average precision, mean reciprocal rank, and precision at 1 showed that their model outperformed all of the others. With the help of LSTM and CNN layer, their model was able to capture more features and semantic patterns.

Kaggle is a well-known website dedicated to data science. A lot of competitions are organized by Kaggle, especially for data science projects. For example, in 2017, a competition was organized on quora questions pairs with the winning amount of 25000 dollars. Around 3290 teams participated in that competition. It was a huge success for NLP, and since then, the popularity of natural language processing has been growing exponentially.

Ansari and Sharma, in their work [AS20] compared some of the well-performing machine learning and deep learning models to find the duplicate questions in the Quora questions pairs (QQP) dataset. They used KNN, decision trees, random forest, extra trees, gradient boost, XGBoost, and AdaBoost from machine learning models. Along with the original features (X and y of the dataset) and basic features, the authors also utilized fuzzy, distancing, and vector features. In addition, TF-IDF was used as a scoring mechanism for the words. From neural networks, they used LSTM as the foundational model for classification along with Glove embedding, dense layers, 1DCNN, 1D global max pooling, batch normalization, dropout, preLU, and sigmoid activation function, as they had only two output classes. XGboost performed best from machine learning models with 78.81% accuracy and 75.97% f1 score when using TF-IDF at the word level. In contrast, 82.42% accuracy and 80.44% f1 score when TF-IDF was used at the character level. They also made four models of different architectures with LSTM and other NN layers. Model 1 used simple word embedding with LSTM for both of the questions. Then after merging them, they used batch normalization, dense layer, PreLU, Dropout, Batch normalization, Dense layer, and sigmoid activation. Model 1 produced way better results compared to the machine learning model as expected with 81.33%. In addition, their accuracy increased with the complexity of the model. Model 4, being the most complex model, gave an accuracy of 85.82 %.

Natural languages are easy for humans to understand but, reasonably, not for computers. Computers being machines, understand numbers more appropriately. That is why if processing and working on natural language text is required, it is understandable to convert them into numbers. One of the early successful trials of finding semantic similarity between text was made through the vector space model [SWY75] dating back to 1975. Without converting the text into vectors, it is not possible for ML/DL models to process it. Word embedding is a technique of converting the text into a vector for processing later. Chawla, along with her fellow researchers, made a comparative analysis of semantic similarity word embedding techniques [CAK21] where they evaluated based on the performance and computational resources of a few popular embedding techniques. They compared TF-IDF, word2vec, sent2vec, doc2vec, and FastText. The dataset was appropriately preprocessed, and the same distancing measures were used to keep the comparison fair. Appropriate thresholds for distancing measures were set after careful experimentation. All of these five embedding techniques were evaluated on two datasets, Quora Question Pairs (QQP) and Plagiarized Short Answers (PSA). FastText from Facebook dominated the results in all of the evaluating matrices i.e, accuracy, precision, recall, f1 score, and ROC curves. The results of FastText were the highest and the most resource-efficient among all of the compared, while sent2vec was the most resource-demanding. On the other hand, TF-IDF being the oldest and most straightforward of the lot, still produced an almost similar result to the others [CAK21]. All of these results can be increased by combining other techniques like negative sampling and normalization.

Another effort was made to evaluate three prominent classifiers by Shah and his fellow researchers in [Sha+20]. They evaluated k nearest neighbors, random forest, and logistic regression. They used a BBC news dataset to classify the given text into different categories i.e. Business, entertainment, politics, sports, and technology. The text was appropriately preprocessed and cleaned before use, and TF-IDF was used as input representation. These three classification models were evaluated on the accuracy, precision, recall,f1-score, support, and confusion matrix. All of the three ML models gave accuracy above 90%. Logistic regression performed the best with an accuracy of 97% while random forest produced 93% and kNN 92%. The performance of any model is dependent on many factors, for example, the quality of the text in data and the complexity of tasks. They used the BBC news dataset, and news has very structured and well-formed text without any noise and slang. Secondly, predicting the genre is quite an easy task and usually gives excellent results depending on the data.

After the adoption of word embedding techniques, researchers tried to improve the performance of word embeddings. WMD performs well on text classification and semantic similarity tasks. Still, due to its below-par super cubic time complexity, it is usually not preferred to be used in time-efficient tasks. On the other hand, the time complexity of soft cosine similarity is quadratic in the worst-case scenario. In [Nov+20], authors used two different techniques to regularize word embeddings. They used quantization and orthogonalization. Their results were calculated with the help of two document similarity measures mentioned above, WMD and SCM, in 6 standard datasets. With the help of quantization, they were able to reduce the average kNN test error from 60.57 to 48.86 for soft VSM. Still, WMD was unable to get any benefit as it uses only the most similar word instead of using all words in the document. Similarly, orthogonalization reduced the processing time of soft VSM by 2.73x with worst-case time complexity from quadratic to linear. Again, quantization helped slow down the operational time by1.8x. Authors argue that the WMD to be slowed down on average by 7450x [Nov+20]. They proved the superiority of soft VSM over WMD, which was believed to be state-of-the-art at that time. Khan has elaborated the whole architecture of BERT very precisely in his work [Kha21]

4 Dataset: Home-Improvement

4.1 Data Crawling For Home-Improvement

The performance of any model is hugely dependent on the quality of the dataset used. After a lot of research, we could not find any kind of built dataset in DIY (do-it-yourself) or home improvement. So the data had to be crawled from a QA website in order to make a dataset. We used "www.diy.stackexchange.com" as stack exchange is the biggest and the most famous website for question answering having the highest number of questions asked in different domains on the web. The raw data was crawled from the home-improvement domain stack of the website. As shown in the code snippet below, the whole data is in JSON, with keys and values. The data was read as a JSON object. Json library offers the keys() function, which can return all the keys in the data.

The raw data has a lot of attributes assigned to each questionID. Attributes of a questionID are question, context, upvotes, tags, edit time, edit lint, owner link, related questions, and linked question. We have converted the raw data from JSON into a pandas data frame for further processing. We will use only questionID, question, context, related question, and linked question, so we filtered only these attributes. We extracted 56431 discrete questions without any null or duplicate questions. The URL of the question was used as the key. The value was all other attributes where the names of the attributes were used as keys again with their actual values as the value of the key-value pair.

```
{
  "https://diy.stackexchange.com/questions/202518": {
    "question": "Can I use four-wire cable from the panel to
       create a two-circuit duplex outlet?",
    "context": "I am planning on adding a server to one of the
       closets in my house. In order to do that I need to run
       electrical to the location of the server. For severs they
        recommend having 2 independent lines of power, for
       redundancy, that way if one breaker trips the server won'
       t go down. The server has 2 power supplies with
       independent cords. I am wondering if I can do a single 24
       0v run from my breaker box to the location of the server
       and then separate the 240 to 120 and into 2 outlets, also
        can I run it into 2 outlet with 2 plugs, and would there
        be any code violations?\nThis is what I'm thinking,
       except this is all a 4-wire cable instead of 4 separate
       wires.\n\n",
    "upvotes": "2",
    "tags": [
      "https://diy.stackexchange.com/questions/tagged/electrical",
      "https://diy.stackexchange.com/questions/tagged/wiring",
      "https://diy.stackexchange.com/questions/tagged/receptacle",
      "https://diy.stackexchange.com/questions/tagged/120-240v"
    ],
    "q_user_edit_time": "2020-09-03 13:52:52Z",
    "q_user_edit_link": "https://diy.stackexchange.com/users/35141",
    "q_user_owner_link": "https://diy.stackexchange.com/users/122896/austinthemighty",
    "related_questions": [
      "https://diy.stackexchange.com/questions/77451",
      "https://diy.stackexchange.com/questions/89453",
      "https://diy.stackexchange.com/questions/132432",
      "https://diy.stackexchange.com/questions/154509",
      "https://diy.stackexchange.com/questions/155899/",
      "https://diy.stackexchange.com/questions/159391",
      "https://diy.stackexchange.com/questions/176714",
      "https://diy.stackexchange.com/questions/183598"
    ],
    "linked_questions": [
      "https://diy.stackexchange.com/questions/202550",
      "https://diy.stackexchange.com/questions/124585",
      "https://diy.stackexchange.com/questions/183698"
    ]
  },
```

4.2 Dataset Building

4.2.1 Question-to-Question Pairs

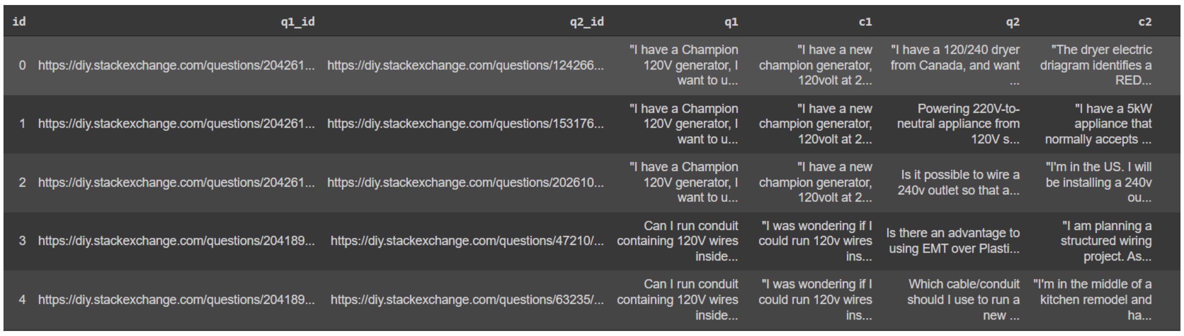
At this point, we have crawled the data and then dropped some of the attributes. Before we actually build the dataset, it is important to understand the data we have for now.

a) **QuestionID:**

   It is the url for questions on the website, as it is unique so it is used the key while drawling the data while all other information about that question is the value for that specific key value pair.

b) **Question:**

   The actual text of the question, er are the most interested in. For a specific QuestionID, question is key and the text string is value



id	q1_id	q2_id	q1	c1	q2	c2
0	https://diy.stackexchange.com/questions/204261...	https://diy.stackexchange.com/questions/124266...	"I have a Champion 120V generator, I want to u...	"I have a new champion generator, 120volt at 2...	"I have a 120/240 dryer from Canada, and want ...	"The dryer electric driagram identifies a RED...
1	https://diy.stackexchange.com/questions/204261...	https://diy.stackexchange.com/questions/153176...	"I have a Champion 120V generator, I want to u...	"I have a new champion generator, 120volt at 2...	Powering 220V-to-neutral appliance from 120V s...	"I have a 5kW appliance that normally accepts ...
2	https://diy.stackexchange.com/questions/204261...	https://diy.stackexchange.com/questions/202610...	"I have a Champion 120V generator, I want to u...	"I have a new champion generator, 120volt at 2...	Is it possible to wire a 240v outlet so that a...	"I'm in the US. I will be installing a 240v ou...
3	https://diy.stackexchange.com/questions/204189...	https://diy.stackexchange.com/questions/47210/...	Can I run conduit containing 120V wires inside...	"I was wondering if I could run 120v wires ins...	Is there an advantage to using EMT over Plasti...	"I am planning a structured wiring project. As...
4	https://diy.stackexchange.com/questions/204189...	https://diy.stackexchange.com/questions/63235/...	Can I run conduit containing 120V wires inside...	"I was wondering if I could run 120v wires ins...	Which cable/conduit should I use to run a new ...	"I'm in the middle of a kitchen remodel and ha...

Figure 4.1: question pairs generated from the initial question records

c) Context:

It is the further explanation of the the question and can be accessed by the key "context" under questoinID

d) Related questions:

It is a list of questionIDs present which are believed to have some sort of relationship with the specific question.

e) Linked questions:

Like related questions, it is a list of questionIDs present which are believed to have a higher degree of relationship with the specific question.

For our dataset, we needed pairs of questions (question-to-question). First of all, we made pairs of questions, such as each question with each of its related questions in a separate pair. We had questionIDs of other questions in the related question, which we had to search again in the dataset to maintain the integrity before making a pair. The exactly same process was carried out with linked questions of each questionID. The number of questions pairs we got was 438692. Suppose question A has question B in its related questions. Question B will also have a question A in its related questions section, and the same goes for linked questions. So when we made pairs, their were some pairs that were duplicates. So all of the duplicate pairs were removed. The dataset was saved as a ".tsv" file instead of a ".csv" file as we have a lot of commas in our text, and we do not want them to mess up with the columns. A data frame from our unlabelled dataset is shown in the figure 4.1

4.2.2 Annotation of Data

By now, we have created a dataset, but it is not labeled. There is no available labeled dataset of the home-improvement domain. So in order to help the scientific community with a high-quality dataset in this domain, totally manual annotation was required to maintain a high standard. Almost 90% are related to electricity and electrical engineering. Hence the manual annotation was carried out democratically by the author(a computer scientist) and two other engineers (from electrical engineering).

At first, 8000 records were annotated, but the dataset was not balanced. Later on, 299 more records (generated from the linked question: having a high degree of relevance) records were annotated in an effort to make the dataset balanced. We have three classes. Similar questions are annotated with 2. Relevant question pairs are annotated with 1. Whereas irrelevant question pairs are annotated with 0. In total, 8299 records were manually annotated to make the first dataset in the home improvement domain that will also be made public to the scientific community.

5 Methods

5.1 Data Exploration

In our dataset, we have 8299 records in total containing question pairs along with their contexts i.e. q1, c1, q2, c2, and annotation where q is question, c is context, and annotation is their label based on the extent of their similarity. The dataset contains three kinds of classes. Similar question pairs are annotated as 2, pairs with some degree of relationship are annotated as relevant. In contrast, others with no relevance are annotated as 0. We have 2213 instances of similar(class 2) pairs of questions, 3606 question pairs are relevant(class 1), and 2480 instances of irrelevant (class 0) question pairs. Is it shown in the bar graph 5.1.

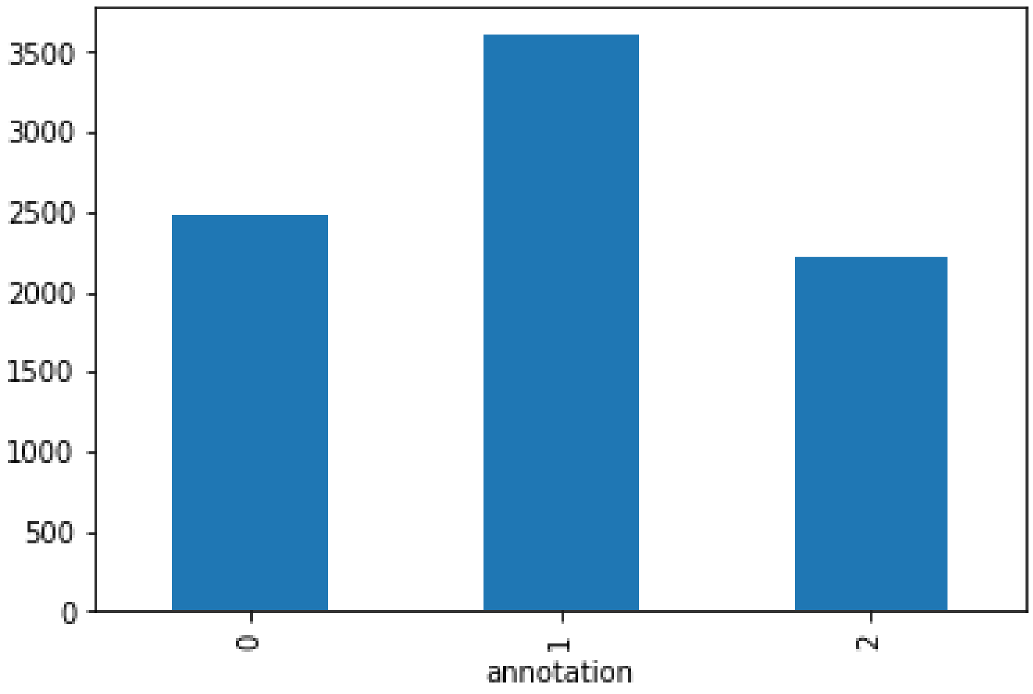


Figure 5.1: Class distribution of our dataset . 0:irrelevant, 1:relevant, 2:similar

In 8299 rows, 3496 questions are unique, and they do not appear again in any other question pairs. At the same time, one question that appeared the most in pairs appeared in 44 pairs in total. A log histogram 5.1 shows the number of occurrences of questions in the dataset. We can see that most of the questions appeared less than 27 times, and only one question was repeated 44 times. In contrast, almost 3700 questions appeared only, so they were unique in

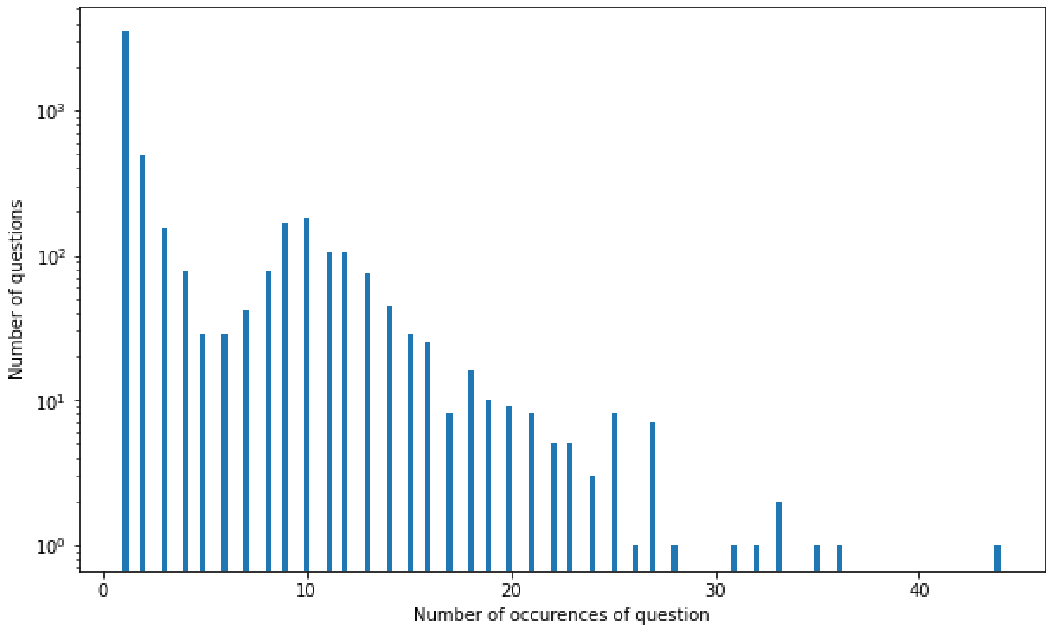


Figure 5.2: Log-histogtam of question appearance counts

our dataset rest of the questions appeared multiple times in pairs. One thing is to be kept in mind that we do not have any duplication in pairs. So when we say that the question appeared multiple times, it appears in a pair with different questions but not with the same questions.

In our dataset, the minimum number of characters in question 1 or question 2 of any pair is 15 characters, including white spaces. In contrast, the maximum number of characters in a question is 152. Likewise, we have 31 tokens(words) at the max in any question, whereas the least number of tokens in a question is just 2. We have 8299 question1 and 8299 question2 in our dataset, which combine to be 16598 questions in total.

Out of 16598 questions, we have 27 questions that comprise only two tokens, while 291 questions are comprised of only three tokens. We have 885 questions where the number of tokens is less than five. On the other hand, 334 out of 16598 questions have more than 19 tokens. The graph 5.3 shows the number of occurrences of total characters in a question. All of this information is necessary to understand the dataset's behavior while evaluating and setting the length for inputs in different models.

5.2 Dataset Preprocessing

Preprocessing is considered to be a significant step in the performance of a model. We came to know that the data is highly unstructured during data exploration. We have to work on preprocessing to enhance data quality to perform. The preprocessing stage is shown in 5.5 as

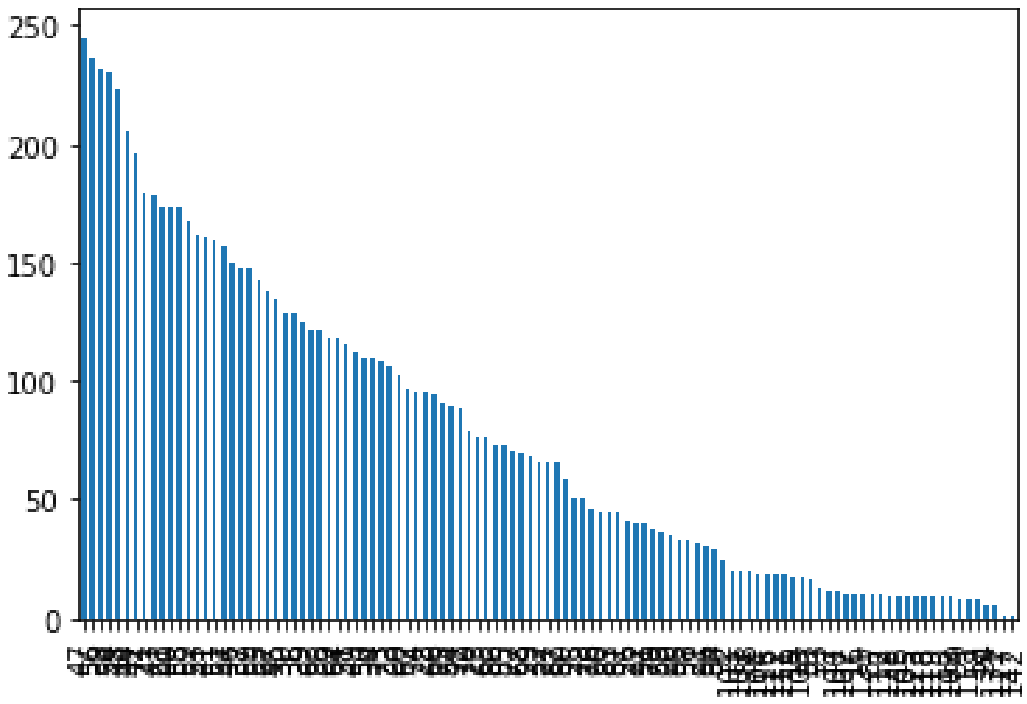


Figure 5.3: number of occurrences for different lengths of questions

a flow chart. Some of these steps are described below.

a) Tokenization:

It is a process of getting the smallest linguistic units from the text, for example, disjoints words, phrases, and symbols from a chunk of text which might be of any length. Word tokenizer from NLTK library is used for ML and NN models, while BERT tokenizer is used for BERT models.

b) Lowercasing:

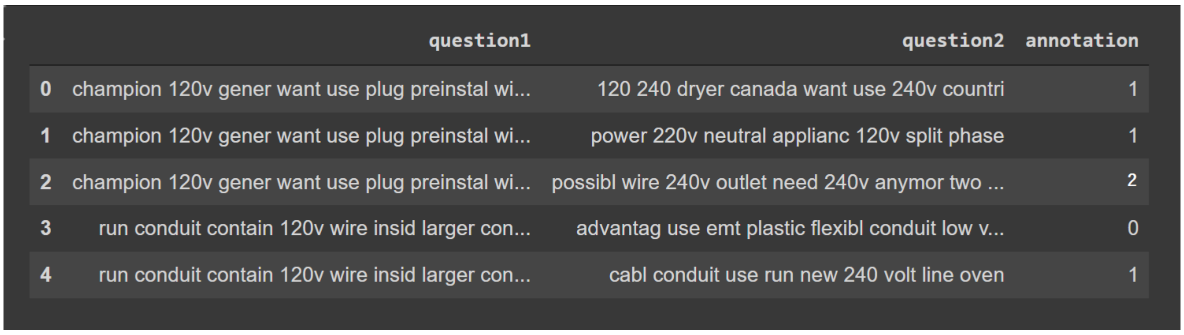
All of the text is lowercased to maintain the uniformity of the text as well as to reduce the dimensionality. For example, "Wire" and "wire" are two separate tokens in the text, but if we lowercase the text, both of them will be represented by a single token, "wire".

c) Stopword:

In the text, different words have a different levels of significance. Usually, there are a lot of words that have very minimum to zero importance in the whole context. It is better to get rid of them to simplify the model to classify and reduce the dimensionality. Stopwords from the most popular NLTK library are used to remove from our dataset.

d) HTML tags:

As the data is crawled from a website, It contains HTML tags as well. For example, the "<br>" tag was found the most among some other tags in our dataset. All of the found HTML tags are removed.



	question1	question2	annotation
0	champion 120v gener want use plug preinstal wi...	120 240 dryer canada want use 240v countri	1
1	champion 120v gener want use plug preinstal wi...	power 220v neutral applianc 120v split phase	1
2	champion 120v gener want use plug preinstal wi...	possibl wire 240v outlet need 240v anymor two ...	2
3	run conduit contain 120v wire insid larger con...	advantag use emt plastic flexibl conduit low v...	0
4	run conduit contain 120v wire insid larger con...	cabl conduit use run new 240 volt line oven	1

Figure 5.4: The dataframe generated from the preprocessed dataset containing only the required columns

e) **Contractions:**

In English, all words are usually used in contracted form. For example, ”cannot” is also written as ”can’t”. Again, two different tokens for a word appear differently in the text. Some of the most commonly used contractions are manually picked to be converted into the standard form.

f) **Noise removal:**

Generally, the informal text contains a lot of noise in the shape of punctuation, smilies, and special characters. All of them are removed with the help of regular expression.

g) **Lemmatization:**

Words have different forms of the verb and by altering a few characters can also be used as adjectives. The best way to deal with all those forms and variants is to convert all of the instances of their variants to one single standard form. Stemming converts a word by removing a few of the last characters, which is sometimes not a valid word. In contrast, lemmatization converts the word to a standard dictionary word. WordNet lemmatization is used here.

The preprocessed data was loaded into the pandas and the head of the dataframe is shown in figure 5.4

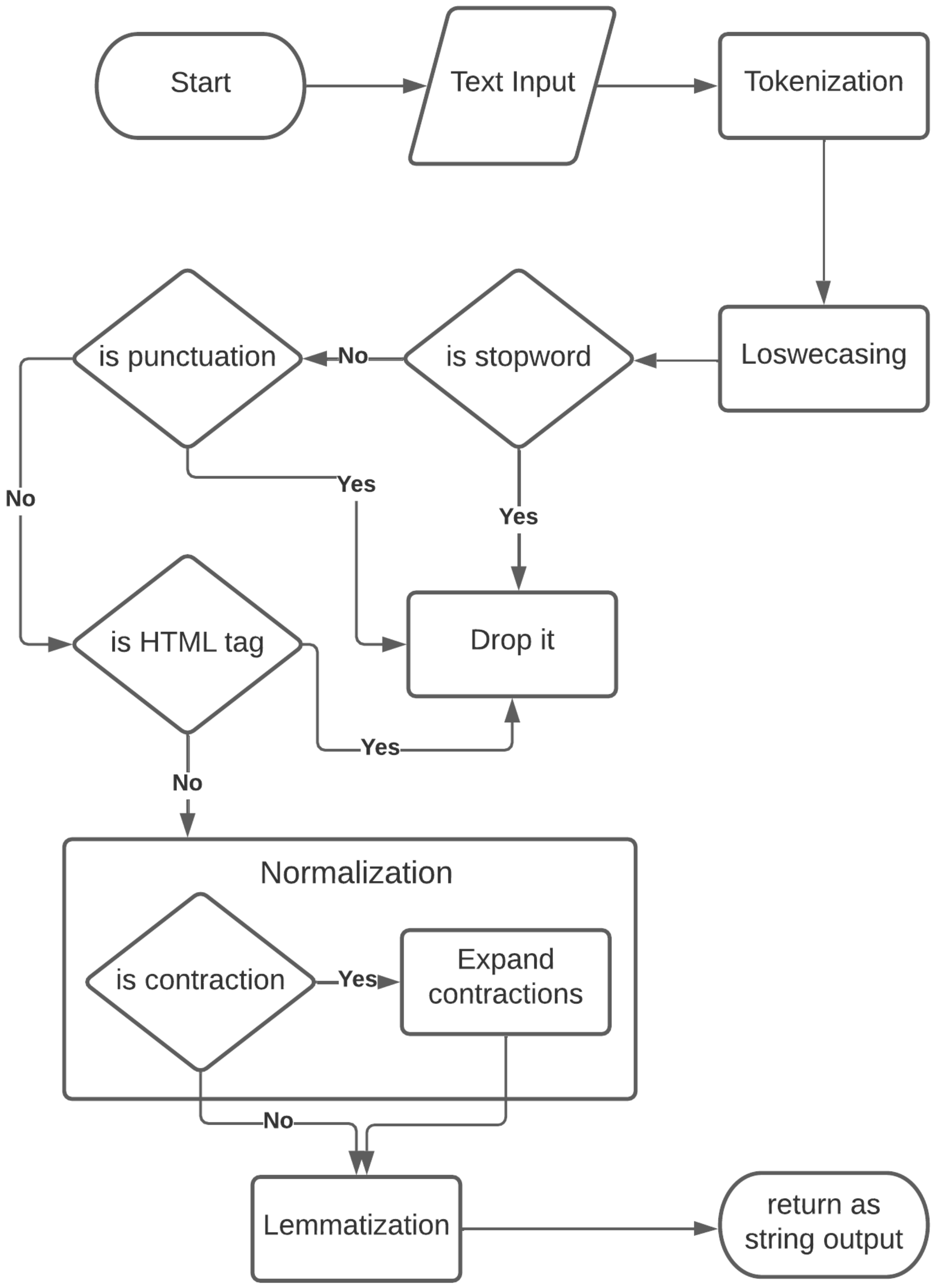


Figure 5.5: The flow diagram for preprocessing the data

5.3 Architecture of models

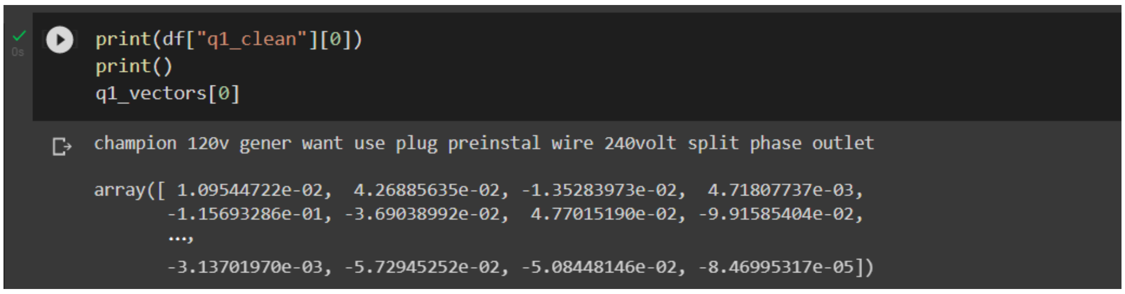
A generic architecture of this work is shown in 5.6. The first part was dataset creation which we have already discussed in 4. Next, we crawled the data and then made question pairs from it. Later on, we manually annotated sufficient records to complete the first dataset in the home-improvement domain. The second part of the architecture is about preprocessing, which is an essential process for any NLTK tasks and is discussed in 5.2. Finally, we tokenized the text, lowercased the whole data, removed stopwords & noise, and lemmatized the text to bring some level of uniformity.

In the next section of our architecture, we can we the classifying models. We used three types of models i.e. machine learning models, neural network models, and BERT models. We used eight models in total: random forest, support vector machine, LSTM, GRU, bi-directional LSTM, bi-directional GRU, RoBERTa, and BERT-base-uncased. We used word2vec and BERT embeddings with these models. The architecture of ML NN and BERT models is different from each other.

5.3.1 Word2vec

Word2vec is a numerical representation of the text that aid classification models to understand the text. It has been a prevalent choice for to generate the word embeddings of text. Word embedding is briefly explained in 2.3. It works based on two methods i.e. skip-gram and continuous bag of words(CBOW). Skip-gram tries to predict the context based on the provided main word, whereas CBOW tries to predict the main word based on the provided context of that main word. Both of these models can be seen as the flipped versions of each other. Authors of the models suggested that CBOW works well for short data to mid-sized data while skip-gram performs well for mid to huge datasets. Word2vec with skip-gram is used with a size of 256, meaning each text token is represented by 256 different numbers. In our experiments, we used the pre-trained word2vec model from the gensim library to generate a word representation of the text.

As word2vec will make 256-dimensional output for a single token, we have to reduce the dimensions to make it work well for sentences instead of words. So we take a weighted average of the word representations of words in a sentence to represent the word2vec vector of the sentence, which will be used to compare the similarity of the questions later on. An example from the built dataset is shown in the picture 5.7



```
print(df["q1_clean"][0])
print()
q1_vectors[0]
```

```
champion 120v gener want use plug preinstal wire 240volt split phase outlet

array([ 1.09544722e-02,  4.26885635e-02, -1.35283973e-02,  4.71807737e-03,
       -1.15693286e-01, -3.69038992e-02,  4.77015190e-02, -9.91585404e-02,
       ...,
       -3.13701970e-03, -5.72945252e-02, -5.08448146e-02, -8.46995317e-05])
```

Figure 5.7: Numeric representation of a sentence from word2vec.

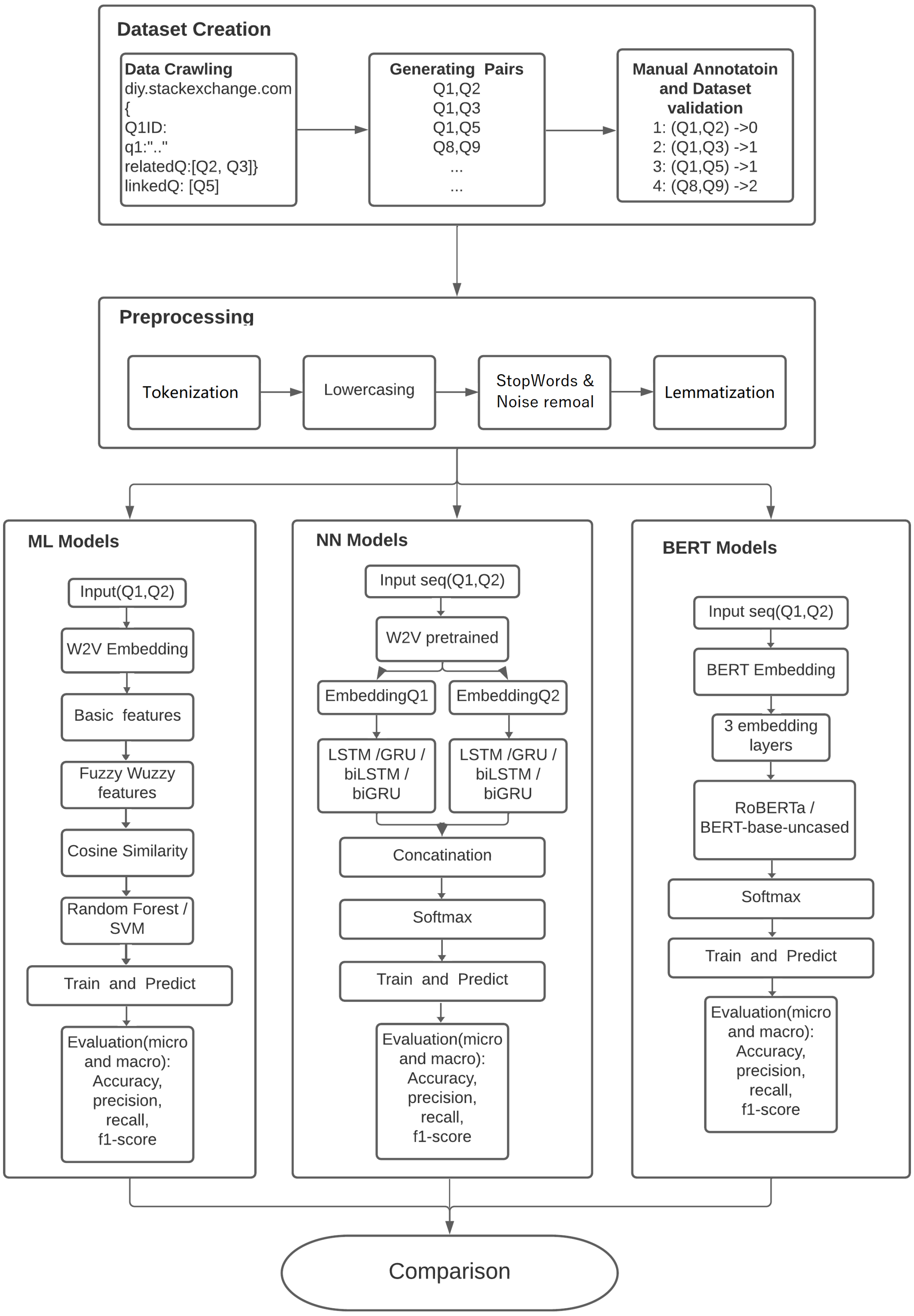


Figure 5.6: Generic architecture of this whole work

5.3.2 Cosine Similarity

Cosine similarity is an important metric used to compute the similarity of texts regardless of their lengths. It is basically measured by computing the angles of vectors after projecting them into multi-dimensional space. Each word is represented by its vector. The similarity of two words, unlike Euclidean distance, which is captured by the magnitudes of vectors, can be computed with the help of the cosine of the angle between their vectors. Smaller the angle means higher the similarity.

$$cos\varphi = \frac{\overrightarrow{A} \cdot \overrightarrow{B}}{||\overrightarrow{A}||||\overrightarrow{B}||} \tag{5.1}$$

Equation 5.1 shows how the cosine similarity is computed. It is the basic formulae of computing cosine similarity, but we used cosine distance from the spacy library. Cosine distance can be computed by subtracting cosine similarity from 1.

5.3.3 Feature Engineering

Hand-crafted features are the backbone of the machine learning models. Neural networks learn all the patterns themselves, but for machine learning, the selected features play a significant role in the performance of the models. We used some of the basic features i.e. length of q1, length of q2', the difference in their lengths, length of characters in q1, length of characters in q2, number of words in q1, number of words in q1, and common words in q1 and q2. These are very basic features that are used in the majority of the tasks in NLP.

We also made use of Fuzz from fuzzywuzzy library. They perform amazingly in text comparison tasks such as plagiarism detection, entailment detection and text similarity etc. From fuzz, we used `fuzz_qratio`, `fuzz_WRatio`, `fuzz_partial_ratio`, `fuzz_partial_token_set_ratio`, `fuzz_partial_token_sort_ratio`, `fuzz_token_set_ratio`, `fuzz_token_sort_ratio`. As their names suggest, these features checks the ratio of similarity in both of the text. For example "Ahmad runs so fast," and "so so so so fast, runs Ahmad" both of these texts have `fuzz_token_set_ratio` = 100

5.3.4 SVM model

Support vector machines are used as the first classification model of machine learning. SVM is comparatively less complex, and that's why it is pretty fast as compared to other models. SVM was basically designed for binary classification and actually did not have three or more classes by default. But due to its high performance on binary classification, an approach for converting multi-classification into a binary classification is used to use SVM for multiclass classification. Two examples of this approach are one-vs-all (OVA) and one-vs-one. OVA approach converts a multiclass classification task into one binary classification task per class. On the other hand, the OVO approach converts a multi-classification task into one binary classification task per each pair of classes. OVO is believed(by the majority) to perform better, but still, it depends on the nature of the task.

We used "decision_function_shape = OVO" in SVM from the sklearn library to use it for multiclass classification. In addition, we fed word2vec representations of texts, along with basic and fuzzywuzzy features described in 5.3.3 into our SVM model. We also used the spacy library's cosine distance as an essential classifying feature.

5.3.5 Random Forest model

Random forest has been a popular choice for classification tasks. Quora, one of the biggest QA websites, still uses random forest algorithm to find out the duplicates of the questions. Random forest takes a little bit higher time complexity, but the performance is worth it. Here, in this work, we used GridSearchCv to find the best hyperparameters for max_depth and n_estimators, both of which are already described in 2.5.2.2. From GridSearchCv, we found out that max_depth = 300 and n_estimators = 900 produced the best results in our dataset. We used the basic features, fuzzywuzzy features, word2vec representations, and cosine distance for random forest from the sklearn library and the best hyperparameters.

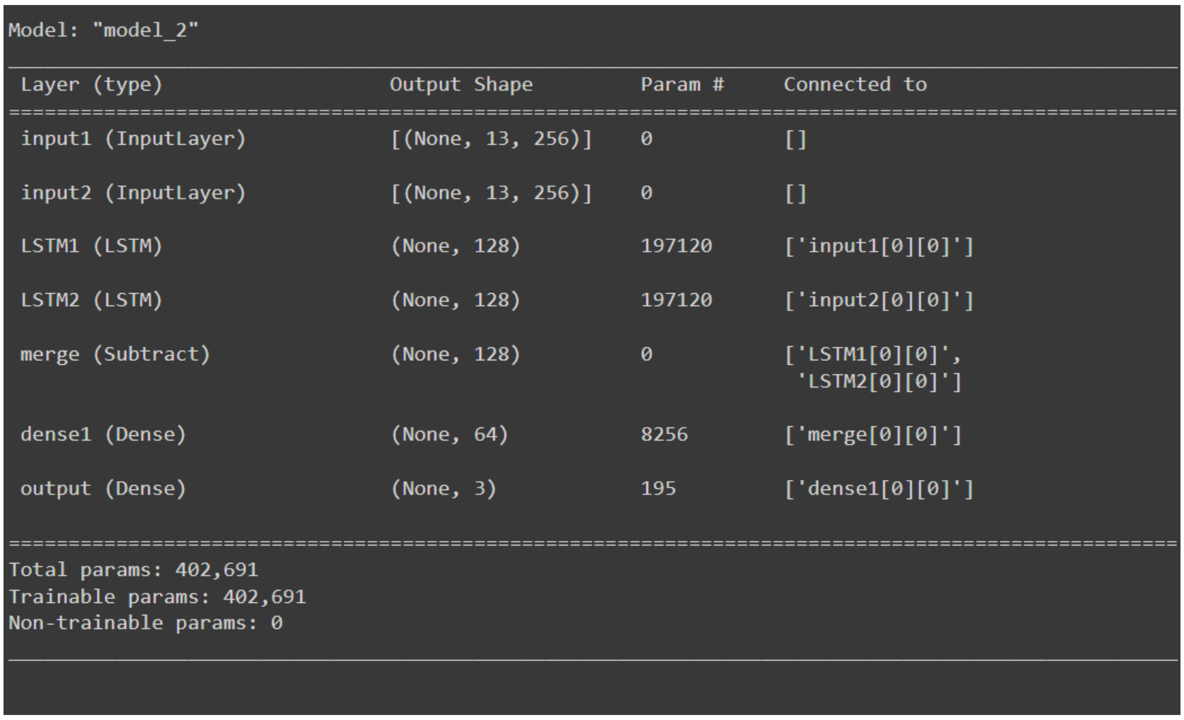
5.3.6 LSTM model

LSTM works on sequences instead of separate inputs feeding one by one. So for LSTM, we convert all of the q1 docs into one list of the sequence of all the instances, and the same goes for other input. We make a sequence of q2 as well. The longest sequence instance is 22 tokens long. We set the maximum input length to 13 as only 2.82% records are longer than 13 while 6413 records which are 77.27%, are shorter than 10. If we keep increasing the maximum input length, these 77.27% will have more meaningless padding. So padding is added to sequences to make them of constant length (maximum input length). We set the size of a batch 64 and a total of 16 epochs. From the input sequence, word2vec representations are generated.

We created two models for q1 and q2. Each model was fed with a word2vec embedding layer and an LSTM layer. LSTM layer of 128 length with dropout 0.45, recurrent dropout 0.1, and activation relu are used. Then both of the models are merged, and after using one dense layer, softmax function computed the probabilities of all three classes. Adam optimizer is used to compile the final LSTM model. The model summary is shown 5.8.

max_input_length	**Instances**	**Coverage**
≦22	8299	100%
≦17	8290	99.89%
≦16	8262	99.55%
≦15	8252	99.43%
≦14	8176	98.52%
≦13	8065	97.18%
≦12	7904	95.24%
≦11	7616	91.77%

Table 5.1: Coverage of question **pairs** with respect to the maximum length



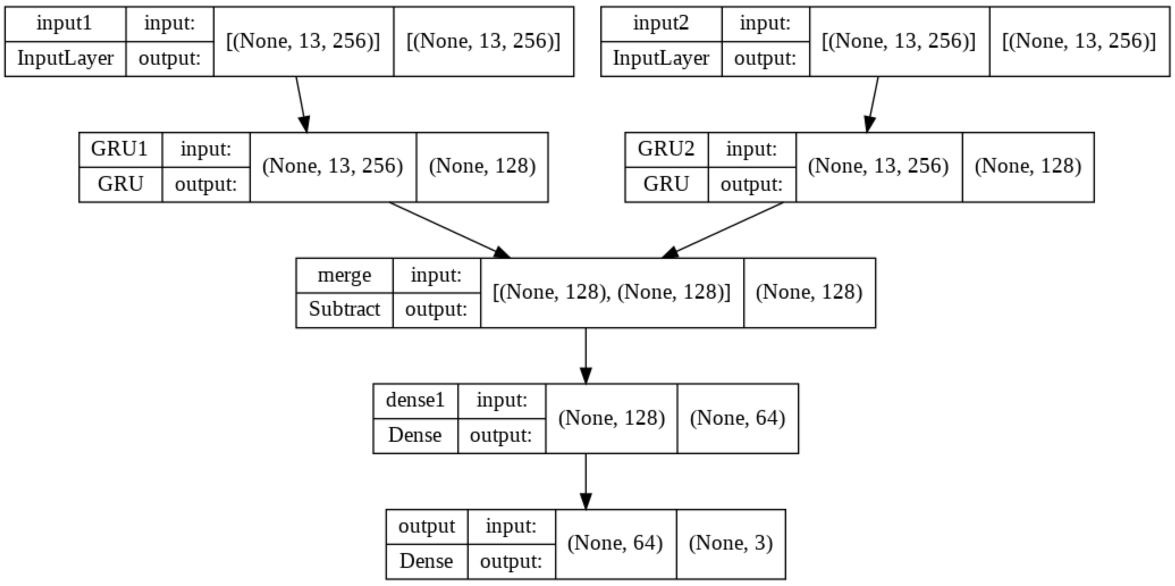
```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input1 (InputLayer)            [(None, 13, 256)]    0           []

 input2 (InputLayer)            [(None, 13, 256)]    0           []

 LSTM1 (LSTM)                   (None, 128)          197120      ['input1[0][0]']

 LSTM2 (LSTM)                   (None, 128)          197120      ['input2[0][0]']

 merge (Subtract)               (None, 128)          0           ['LSTM1[0][0]',
                                                                  'LSTM2[0][0]']

 dense1 (Dense)                 (None, 64)           8256        ['merge[0][0]']

 output (Dense)                 (None, 3)            195         ['dense1[0][0]']

==================================================================================================
Total params: 402,691
Trainable params: 402,691
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5.8: Summary of LSTM + word2vec based model.

5.3.7 GRU model

A Gated Recurrent Unit(GRU) is an enhanced version of RNN. It is relatively faster and more efficient than LSTM as its mechanism is less complex. GRU is explained in 2.6.2. GRU also works with input sequences, and again we convert all the q1 documents into one sequence. In contrast, all of the q2 documents are converted into another sequence. Padding is used to make all instances of a sequence of equal lengths. As we are using the same data, we will stick to the maximum string length of 13. A table of different options for maximum_input_length is given at 5.1 . Maximum string length can also be set to the length of the longest input text from q1 and q2, which is also a general approach. But after data exploration, we realized that a huge subset of the q1 and q2 has a length smaller than 10. The more we increase the maximum length, the more padding we will have to add to all of them.

Again we created two separate models for q1 sequences and q2 sequences. Both of the models are fed with word2vec representation of the sequences along with a GRU layer, each having 128 length 0.45 dropout and 0.1 as recurrent dropout with relu activation. Both of the models are merged, and after one dense layer, we have the output layer as the final layer, which uses the softmax activation method to predict the probabilities of each class. The optimizer used is adam. The main difference between the LSTM and the GRU models is their parameters. LSTM has 402,691 trainable parameters while GRU has 304,893, which is almost 25% less than those of LSTM, which is due to the fact that GRU is an enhanced version of LSTM, reducing the complexity. The model of GRU is plotted in 5.9

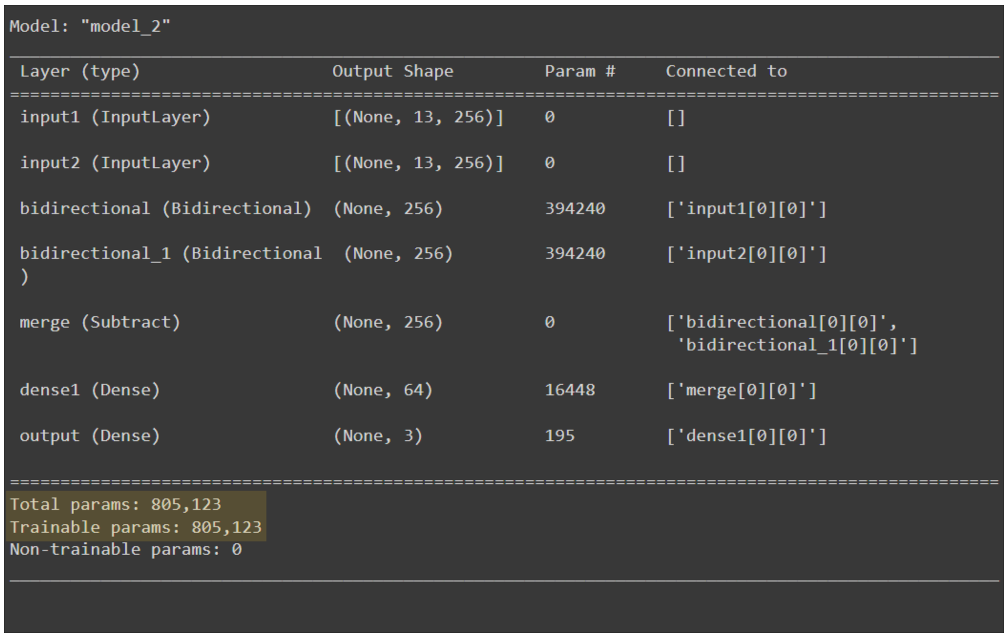


input1 InputLayer input: output: [(None, 13, 256)] [(None, 13, 256)]
input2 InputLayer input: output: [(None, 13, 256)] [(None, 13, 256)]
GRU1 GRU input: output: (None, 13, 256) (None, 128)
GRU2 GRU input: output: (None, 13, 256) (None, 128)
merge Subtract input: output: [(None, 128), (None, 128)] (None, 128)
dense1 Dense input: output: (None, 128) (None, 64)
output Dense input: output: (None, 64) (None, 3)

Figure 5.9: Compiled model of GRU

5.3.8 Bi-directional LSTM and GRU models

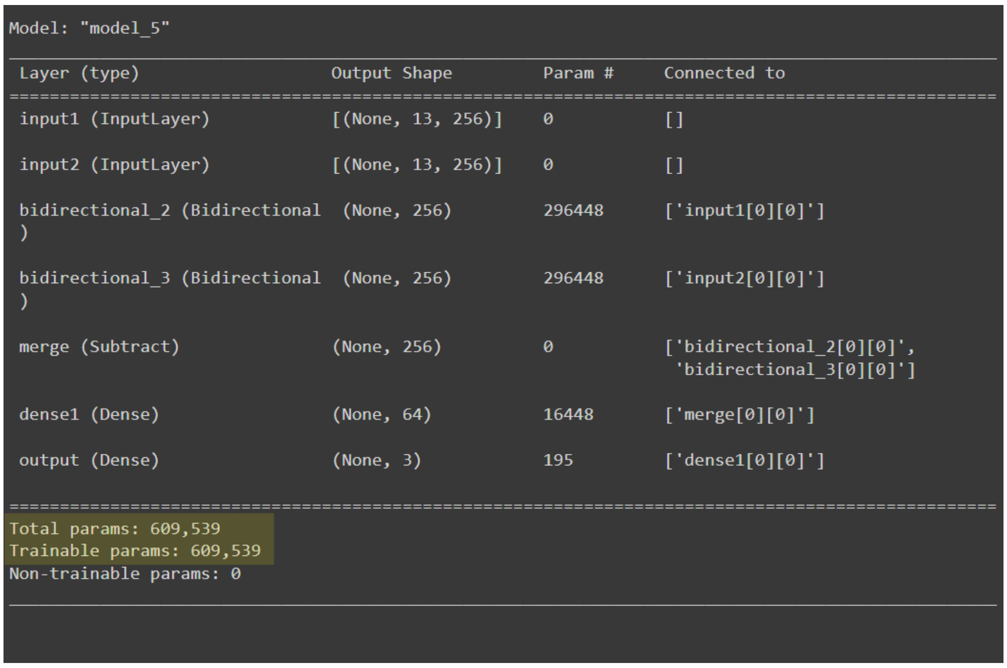
Bi-directional models i.e. bi-directionalLSTM and bi-directionalGRU are explained in sections 2.6.3 and 2.6.4 respectively. These models are pretty much the same as of LSTM and GRU except that we add bi-directionality to them to capture the context both from left-to-right and right-to-left.



```
Model: "model_2"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input1 (InputLayer)            [(None, 13, 256)]    0           []

 input2 (InputLayer)            [(None, 13, 256)]    0           []

 bidirectional (Bidirectional)  (None, 256)          394240      ['input1[0][0]']

 bidirectional_1 (Bidirectional  (None, 256)         394240      ['input2[0][0]']
 )

 merge (Subtract)               (None, 256)          0           ['bidirectional[0][0]',
                                                                  'bidirectional_1[0][0]']

 dense1 (Dense)                 (None, 64)           16448       ['merge[0][0]']

 output (Dense)                 (None, 3)            195         ['dense1[0][0]']

==================================================================================================
Total params: 805,123
Trainable params: 805,123
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5.10: Summary of Bi-directional LSTM + word2vec based model.

In figure 5.10 we can see that the total parameters of biLSTM are 805,123, which are double than the parameters of LSTM, which were 402691. The same is the case of BiGRU 5.11 having 609,539 parameters which are also double than those of GRU. First, it is due to the fact that we are using bidirectional here, so the params get doubled as the context is learned both ways instead of only one. Secondly, the model for biLTSM and biGRU is the same. Yet, biGRU has fewer parameters as its mechanism is more straightforward as it uses two gates instead of three in LSTM.



```
Model: "model_5"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape          Param #     Connected to
==================================================================================================
 input1 (InputLayer)            [(None, 13, 256)]     0           []

 input2 (InputLayer)            [(None, 13, 256)]     0           []

 bidirectional_2 (Bidirectional  (None, 256)          296448      ['input1[0][0]']
 )

 bidirectional_3 (Bidirectional  (None, 256)          296448      ['input2[0][0]']
 )

 merge (Subtract)               (None, 256)           0           ['bidirectional_2[0][0]',
                                                                   'bidirectional_3[0][0]']

 dense1 (Dense)                 (None, 64)            16448       ['merge[0][0]']

 output (Dense)                 (None, 3)             195         ['dense1[0][0]']

==================================================================================================
Total params: 609,539
Trainable params: 609,539
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 5.11: Summary of Bi-directional GRU + word2vec based model.

5.3.9 BERT based RoBERTa and BERT-base-uncased

BERT-Base, Uncased: 12-layer, 768-hidden, 12-heads, 110M parameters RoBERTa was developed by Facebook, and as the name suggests, it is based on the BERT model. BERT was pre-trained on 16 GB of text, whereas RoBERTa was pre-trained on 16GB of BERT text + 144GB of more text. BERT-base-uncased has 12 layers, 768 hidden states, 12 heads, and 110 million parameters. Whereas RoBERTa-base has 12 layers, 768 hidden states, and 12 heads but 125 million parameters. That is due to the different methods of RoBERTa for masking. We used both of these models with maximum_input_length to the maximum length from the data as these models can handle them efficiently.

6 Results

6.1 Experimental Setup

In this section, the used tools, libraries, and frameworks are discussed, along with the computational resources. All of the codes were written and executed on Google Colaboratory[1]. Google collab offers limited resources of Tesla t4 GPU, which was used whenever found available. Google collab is an excellent alternative to jupyter notebook, which is a web application for creating and sharing computational documents which are usually used for data science and python projects comprising of equations, computational output, and visualizations along with explanatory text within one file.

The code is written in python language, and the version is 3.9. Some of the worth mentioning libraries used are gensim, NLTK. keras, tensorflow, sklearn, matplotlib. Google collaboratory offers virtual RAM and GPU. Nevertheless, the machine used is MSI gs63 laptop containing an i7 processor with a power of 2.80GHz. The RAM of the used machine is 16 gigabytes. The dataset used is question pairs home-improvement which we first created before use. Most of the time, collab's TPU or GPU was used, but whenever it was unavailable, the laptop's GPU, Nvidia 1050ti was used.

6.2 Results and Evaluation

6.2.1 SVM and RF

SVM and RF algorithms are less complex as compared to neural network and transformer-based models. Due to their simplicity, the training time of these models is also very minor. The results of this work are the average of 3 runs. SVM, on average, took 20 seconds for training, while random forest took 2 minutes on average. Figure 6.1 shows that the outcome is hugely influenced and dominated by class 1, that is relevant questions. It is due to the fact that our dataset is dominated by class 1 with a massive share of almost 46% in the whole dataset.

Table 6.1 shows the precision, recall, and f1 scores at macro, micro, and weighted level, along with the accuracy. For example, SVM achieved an accuracy of 44.36% while table 6.2 shows the accuracy of random forest, which is a little bit higher, 44.89%. In these tables, it is evident that the precision and recall of class 2(similar) are very low, making the f1-score very low as the results are influenced by class 1(relevant).

[1]https://colab.research.google.com/

	Precision	Recall	F1-Score	Support
Irrelevant	42%	35%	38%	517
Relevant	48%	68%	56%	702
Similar	29%	16%	21%	441
Macro avg	40%	40%	39%	1660
weighted avg	41%	44%	41%	1660
Accuracy			44.36%	1660

Table 6.1: Evaluation matrix of support vector machine model

	Precision	Recall	F1-Score	Support
Irrelevant	42%	32%	36%	517
Relevant	47%	76%	58%	702
Similar	40%	13%	20%	441
Macro avg	43%	40%	38%	1660
weighted avg	44%	45%	41%	1660
Accuracy			44.89%	1660

Table 6.2: Evaluation matrix of random forest model

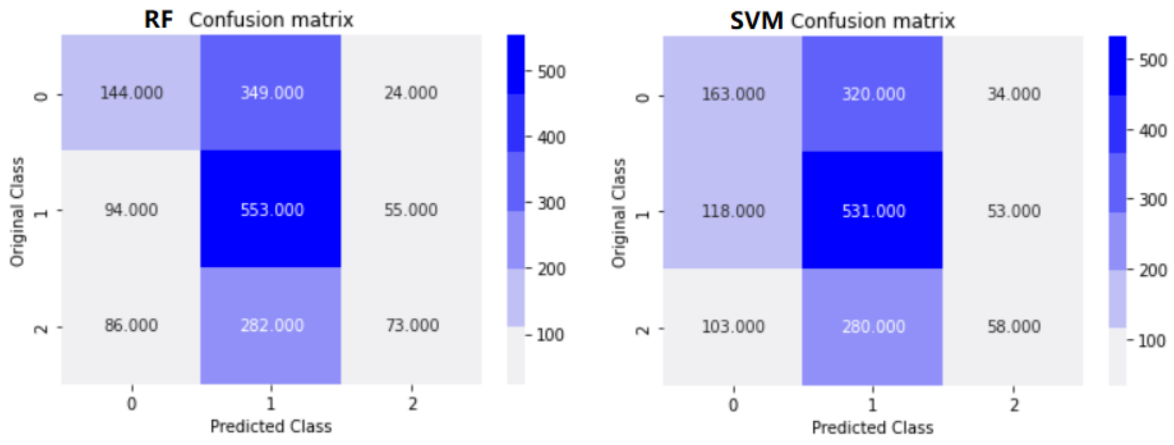


Figure 6.1: Confusion matrices of random forest and support vector machine models

6.2.2 LSTM and Bi-LSTM

LSTM has been a choice to go in the recent past for many NLP tasks due to its performance over other models. In 6.2 shows the accuracy and loss of LSTM and bi-directional LSTM. We can see in the figure that bi-LSTM achieves higher accuracy as compared to bi-LSTM, while is loss is decreased more for bi-directional LSTM. The final average accuracy for LSTM is 45.84% while bi-LSTM achieved 48.01%. The tables 6.3 for LSTM and 6.4 for the bi-LSMT show that the LSTM is still suffering from the dominance of one class due to an imbalanced dataset. At the same time, bi-LSTM, on the other hand, slightly decreased the dominance of class 1(relevant), producing a higher accuracy as it is bi-directional.

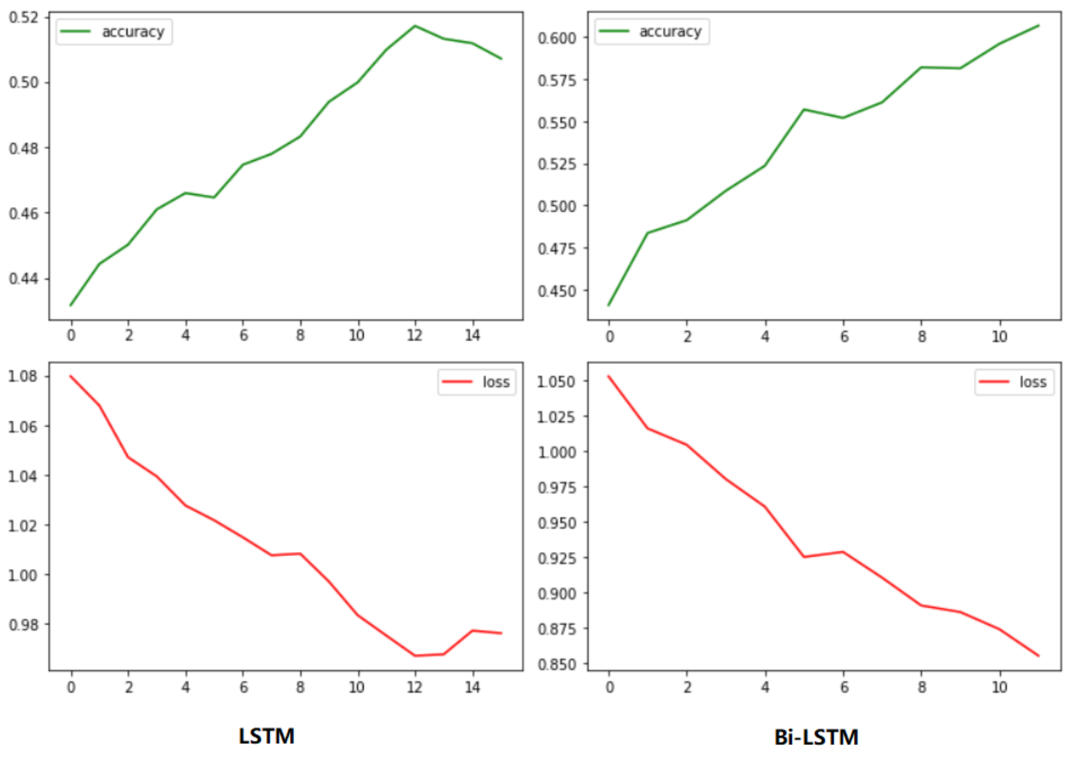


Figure 6.2: Accuracy and loss of LSTM(left) and bi-LSTM(right)

	Precision	**Recall**	**F1-Score**	**Support**
Irrelevant	45%	31%	37%	496
Relevant	48%	74%	58%	713
Similar	35%	18%	24%	451
Macro avg	43%	41%	40%	1660
weighted avg	44%	46%	43%	1660
Accuracy			45.84%	1660

Table 6.3: Evaluation matrix of LSTM model

	Precision	**Recall**	**F1-Score**	**Support**
Irrelevant	42%	45%	44%	496
Relevant	51%	66%	58%	743
Similar	48%	19%	27%	421
Macro avg	47%	44%	43%	1660
weighted avg	48%	48%	46%	1660
Accuracy			48.01%	1660

Table 6.4: Evaluation matrix of Bi-LSTM model

6.2.3 GRU and Bi-GRU

GRU produced better results as compared to LSTM, and bi-GRU outperformed bi-LSTM. However, the margin was not so significant. Bi-GRU achieved higher performance than bi-LSTM, while bi-GRU took less time than the latter. The accuracy and loss of GRU and bi-GRU are shown in 6.2.3. The accuracy of GRU is 46.21% as shown in table 6.5 and bi-GRU achieved 48.33% on average of 3 runs shown in the evaluation matrix 6.6. The evaluation matrices show that the bi-GRU produced a more balanced result and decreased the influence of the dominant class(1-relevant) as compared to all of the other models discussed by now.

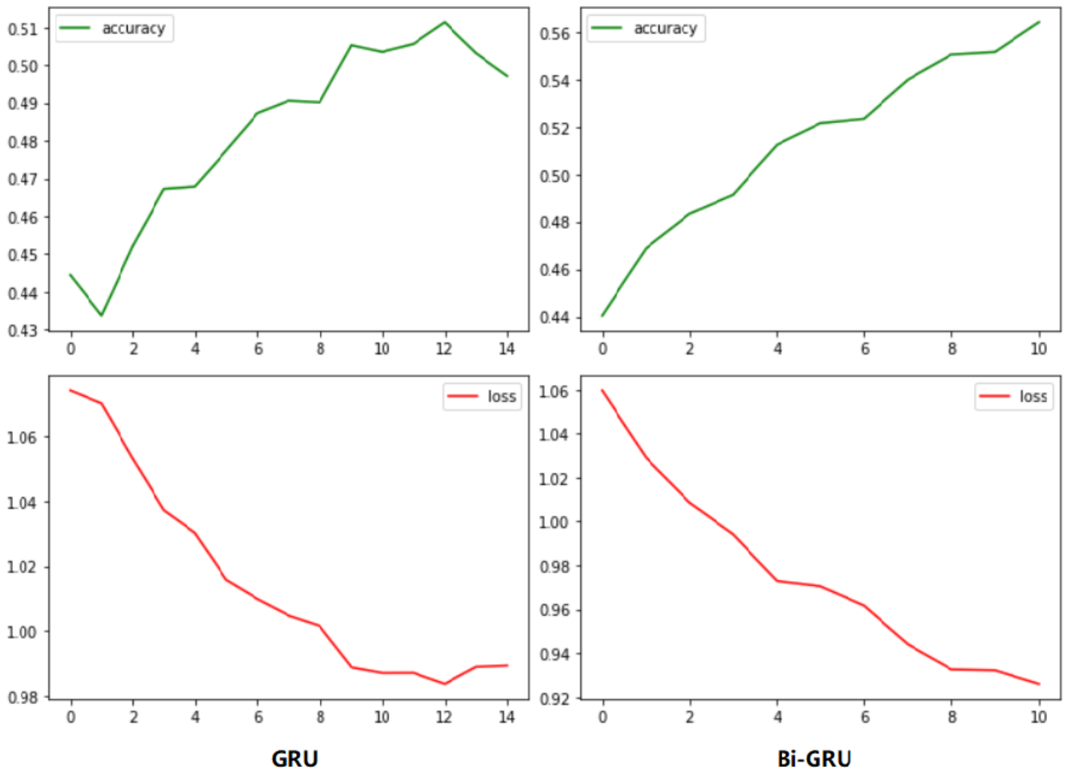


Figure 6.3: Accuracy and loss of GRU(left) and bi-GRU(right)

	Precision	Recall	F1-Score	Support
Irrelevant	41%	40%	41%	477
Relevant	49%	68%	57%	731
Similar	40%	18%	25%	452
Macro avg	43%	41%	40%	1660
weighted avg	44%	46%	43%	1660
Accuracy			46.21%	

Table 6.5: Evaluation matrix of GRU model

	Precision	Recall	F1-Score	Support
Irrelevant	47%	35%	41%	502
Relevant	48%	72%	59%	704
Similar	47%	33%	35%	454
Macro avg	47%	46%	45%	1660
weighted avg	48%	49%	46%	1660
Accuracy			48.33%	1660

Table 6.6: Evaluation matrix of bi-GRU model

6.2.4 BERT and RoBERTa

The performance of BERT and RoBERTA has been above par on this dataset, thanks to its attention mechanism. Due to attention heads, BERT-based models can efficiently handle longer sequences with minimal error. Tables 6.7 and 6.8 depict that all of the classes produced better results. Although our dataset is imbalanced and has almost 46% of data as class "relevant"(out of three classes) yet the results are equally good for all of the classes, unlike previous models where most of the data(including class 2 and class 0) was being predicted as class 1. The attention mechanism provides different importance to each word in the sequence to better understand the relationships among words. As a result, we get better and well-spread results.

Bert-base-uncased is resource hungry, and it took almost 28mins to train one epoch, so we were restricted to fewer epochs, yet it achieved 51.93% accuracy. RoBERTa-base, on the other hand, produced an accuracy of 52.54%. As BERT based models can handle longer sequences, the maximum input length was set to the longest pre-processed input sequence, which is 22.

	Precision	Recall	F1-Score	Support
Irrelevant	50%	62%	55%	485
Relevant	56%	48%	51%	721
Similar	53%	44%	44%	454
Macro avg	50%	49%	50%	1660
weighted avg	53%	52%	51%	1660
Accuracy			51.93%	1660

Table 6.7: BERT-base-uncased evaluation

	Precision	Recall	F1-Score	Support
Irrelevant	53%	55%	54%	480
Relevant	53%	59%	56%	733
Similar	48%	37%	42%	447
Macro avg	51%	50%	51%	1660
weighted avg	52%	52%	52%	1660
Accuracy			52.54%	1660

Table 6.8: RoBERTa evaluation

Since we have an imbalanced dataset, all of the classes are equally important to us. Macro averaging assign the same importance to all available classes, making it a smarter choice for our dataset. Tables from 6.1 to 6.8 show the macro and micro precision, recall, and f1-score of all of the used eight models. On the other hand, micro averaging is always exactly the same for precision-recall and accuracy and is shown as accuracy. The weighted average is calculated by assigning weights according to the support(number of occurrences).

Table 6.9 shows the accuracy of all of the eight models used in this word. Support vector machines and random forest, the oldest and simplest models, produced below-par results. Neural networks like LSTM and GRU somewhat increased the accuracy. Still, the main difference became evident by using bidirectional layers for LSTM and GRU. Finally, Encoder-based models, BERT and RoBERTa produced the best results among all of the used ones. These two models use an attention mechanism to learn relationships among words better while learning the context in both directions. These two models used wordPiece for tokenization, and three layers of embedding, including positional encoding, helped them better understand the data.

	Accuracy on avg
SVM	44.39
RF	44.89
LSTM	45.84
GRU	46.21
BiLSTM	48.01
BiGRU	48.33
BERT-base-uncased	51.93
RoBERTa-base	52.54

Table 6.9: Accuracy of all the used models

7 Discussion

Based on the methods in chapter 5 and result in chapter 6, we can discuss the key findings of this work in order to answer the research questions. Building a dataset from any domain is addressed in section 7.1 while validating a newly built dataset is discussed in section 7.2. Finally, after having a detailed look at the results and evaluating matrices, the comparison of different classification models is discussed in section 7.3.

7.1 Building Dataset

Research Question 1: What are the most important steps to build a new dataset for a new and real-world question-to-question dataset?

When there is no dataset suitable for our task, we usually build a new dataset. The same was the case with our work, there was no public dataset available in the home improvement domain. So the first thing to keep in mind before actually building a dataset is to verify the downstream requirements. In our case, after scrapping the data from the web, we needed only a question text column for this work but keeping the scientific community in mind, we stored the context of the questions as well. Hopefully, the classification models will be able to classify very long text efficiently in the near future, so we made room for them in our dataset by saving the context in our dataset. Identification of the important fields was the next step. For example, the upVotes seem irrelevant, but they give us an idea about the popularity of that question. Similarly, the tags provide us with an idea about the text category, which can be used in different tasks. The next step should be to convert the data according to the shape required for our tasks. We needed pairs of texts, so we paired each question with the questions mentioned in their related and linked sections. Another important factor to keep in mind while building the dataset is the consistency of the data. For example, the annotation should be in one format i.e. numeric. Suppose we annotate the data somewhere in numbers(0,1, or 2) and somewhere else with strings (zero, one, or two). In that case, the data type won't be consistent. The creation of this dataset is essential for our work and the scientific community as there is no dataset available to work on the home improvement domain. That is exactly why no work has been done in classifying text of this domain. It is possible to use this dataset in the future as the context of the question (which is not required in our work) is also preserved for future usage.

7.2 Validating newly built Dataset

Research Question 2: How can we validate our newly built dataset?

The performance of different tasks hugely depends on the quality of data it uses. Validation plays a significant role in maintaining or enhancing the quality of data. Validation is carried out in a number of steps. Validation eliminates the risk of making decisions based on imperfect data, and hence the project defects are mitigated. The first step is to generate a random sample to observe data validation. Data duplication, if found, is removed, and uniqueness is checked. For example, the id of one question should not be the same as any of the other questions. Next, the data format is checked against multiple values of the columns. For example, there should be one format for date whenever it is used. Similarly, the data types should be consistent for any specified column i.e. the annotation should either be made in numbers or strings but not in both. We also check the dataset for empty and null values. All the points discussed so far are observed in validating our dataset. Some other factors might be observed according to the type of dataset, for example, checking the range values and boundary values etc. Data validation is usually carried out along with data analysis or in the data preprocessing stage.

7.3 Performance of Classification models

Research Question 3: Which kind of model has better performance in question to question matching in the home improvement domain?

This work uses different machine learning, neural network, and transformer-based models to classify text into three categories based on semantic similarity. The results and evaluation matrices showed that RoBERTa was the top-performing model with an accuracy above 52% while BERT-base-uncased stood second. BiGRU and BiLSTM were next and almost equally good. Then came the GRU and LSTM models, while after random forest, SVM was the least performing classification model. Let's look at the precision recall and f1 scores of the classes separately. We can see that RoBERTa and BERT have done an amazing job for each class and predicted them in equal proportion. The next model, BiGRU, suffered a little bit by getting 33% recall for class 2(similar) while 71% recall for class 1(relevant). The results suffered more for models LSTM, random forest, and SVM, respectively. SVM resulted in the least performance with accuracy just of 44.36% suffered the most with an imbalanced dataset. The results were dominated by class 1(relevant), with 16% recall for class 2(similar) while 68% for class 1(relevant). Out of 441 samples for class 2, only 69 were correctly predicted as class 2. So RoBERTa and BERT are the best models for multi-class text classification tasks in the home improvement domain.

8 Conclusion and Future Work

We have carried multiclass text classification with the help of word2vec and BERT embeddings on a number of machine learning and deep learning models. The results showed that BERT based models are better for text classification even if the model is not pre-trained on the same domain text. In this work, we used pre-trained BERT, pre-trained RoBERTa, and pre-trained word2vec as embeddings for classifying models. Glove and word2vec models produce almost the same results when used for the same data and same models. However, the older classification models suffered from the imbalance dataset, and their results showed the influence of dominating class. Although BERT based models produce better performance, they are understandably more resource hungry. Usually, they demand more time to train as well.

A logical reason for this finding is that BERT and RoBERTa used the word embeddings from BERT while all other models used the pre-trained word2vec weights. word2vec has a shortcoming of out of vocabulary words due to its lower capacity to represent words. As the word2vec model was not trained on our domain and in our dataset, there was a huge domain-specific vocabulary. Therefore, there were a lot of words being marked as out of vocab. One easy solution to this problem is pre-train the word2vec model on a huge home improvement unlabeled data corpus.

The second reason for comparatively low accuracy has very unstructured data. These models were pre-trained on very structured and formal data, for example, Wikipedia and books corpus. On the other hand, The data we got from stack exchange is very informal and unstructured, containing a lot of slang and irregular words. The grammar used was also not good and formal, and above all, there was a massive proportion of typing mistakes and misspelled words. As it is a standard user/consumer text, it is already evident to have many problems in the text. The solution to this problem is again pre-training on the domain-specific data and filtering only high-quality, structured, and formal text.

We stick to the models in their very basic and default shape in this work. We did not try to enhance the accuracy rather than capture their performance without using any other fancy techniques to exhibit better performance. Of course, their performance can be increased in a number of ways. First of all, we can pre-train the models on domain-specific corpora. It is a time-consuming process, but the time elapsed in this process is worth its performance. Secondly, we can use a structured text to understand the patterns better while fitting the model. We must also make use of negative sampling and text normalization properly. We can also use noise reduction, Named Entity(NE), Latent Semantic Indexing(LSI), Latent Semantic Analysis(LSA), dissimilarity matrices, and so on. Finally, the models, especially deep neural networks, can better perform with the conjunction of other layers or suitable models.

In the future, the research, especially in home improvement domain, can be extended to the latest state-of-the-art (SOTA) models for example, XL-Net, ELMo, ULMfit, and GPT-3. Plain text is first converted to TF record file format for pre-training the BERT models. In the future, this work can also be extended by considering the context text along with the question text from this dataset. The dataset is already provided with context attributes for both instances of the pairs. At the moment, if we want to use context, we will have to use some summarizing techniques (i.e. Text Rank, Sentence scoring) for the context values to truncate the context to make a smaller summary of the context text as the text in context is longer in length and current models cannot classify them with high accuracy. Moreover, due to the intense research in this field, it is reasonable to hope for new models to be publicly available for longer texts with higher accuracy. The dataset created is already annotated with respect to the text of context columns and is ready to be used in the future with only question columns of the pairs as in this work or with both question and context columns. This work opens a path to additional possibilities such as for the better adaptation of models in the home improvement domain.

Bibliography

[AS20] Navedanjum Ansari and Rajesh Sharma. “Identifying semantically duplicate questions using data science approach: A quora case study”. In: *arXiv preprint arXiv:2004.11694* (2020).

[Ash+17] Vaswani Ashish et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[BCA15] Shamim Biswas, Ekamber Chadda, and Faiyaz Ahmad. “Sentiment analysis with gated recurrent units”. In: *Department of Computer Engineering. Annual Report Jamia Millia Islamia New Delhi, India* (2015).

[BCB14] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: (2014).

[BSF94] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. “Learning long-term dependencies with gradient descent is difficult”. In: *IEEE transactions on neural networks* 5.2 (1994), pp. 157–166.

[CAK21] Shrutika Chawla, Preeti Aggarwal, and Ravreet Kaur. *Comparative Analysis of Semantic Similarity Word Embedding Techniques for Paraphrase Detection.* Tech. rep. EasyChair, 2021.

[Cho+14] Kyunghyun Cho et al. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv:1406.1078* (2014).

[Dev+18] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).

[Geh+17] Jonas Gehring et al. “Convolutional Sequence to Sequence Learning”. In: *Proceedings of the 34th International Conference on Machine Learning.* Ed. by Doina Precup and Yee Whye Teh. Vol. 70. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1243–1252. URL: `https://proceedings.mlr.press/v70/gehring17a.html`.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[JT98] Joachims and Thorsten. “Text categorization with Support Vector Machines: Learning with many relevant features”. In: *Machine Learning: ECML-98.* Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 137–142. ISBN: 978-3-540-69781-7.

[Kam+18] Avinash Kamineni et al. “Siamese LSTM with convolutional similarity for similar question retrieval”. In: *2018 international joint symposium on artificial intelligence and natural language processing (isai-nlp).* IEEE. 2018, pp. 1–7.

[Kav20] Eda Kavlakoglu. "AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?" In: (2020). URL: `https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks`.

[Kha21] Zohaib Khan. "Comparing the Performance of NLP Toolkits and Evaluation measures in Legal Tech". 2021. URL: `https://www.academia.edu/45351967/Comparing_the_Performance_of_NLP_Toolkits_and_Evaluation_measures_in_Legal_Tech`.

[LR12] Mihai C Lintean and Vasile Rus. "Measuring Semantic Similarity in Short Texts through Greedy Pairing and Word Semantics." In: *Flairs conference*. 2012, pp. 244–249.

[ML15] Andrew M and Quoc V Le Dai. "Semi-supervised sequence learning". In: *Advances in neural information processing systems* 28 (2015).

[Nov+20] Novotny et al. "Text classification with word embedding regularization and soft similarity measure". In: *arXiv preprint arXiv:2003.05019* (2020).

[PHP14] Sachin Pawar, Swapnil Hingmire, and Girish Palshikar. "LMSim: Computing Domain-specific Semantic Word Similarities Using a Language Modeling Approach". In: *Proceedings of the 11th International Conference on Natural Language Processing*. 2014, pp. 101–106.

[RWN+21] Md Rahman, Yutaka Watanobe, Keita Nakamura, et al. "A bidirectional LSTM language model for code evaluation and repair". In: *Symmetry* 13.2 (2021), p. 247.

[Sha+19] Lakshay Sharma et al. "Natural Language Understanding with the Quora Question Pairs Dataset". In: *CoRR* abs/1907.01041 (2019). arXiv: `1907.01041`. URL: `http://arxiv.org/abs/1907.01041`.

[Sha+20] Kanish Shah et al. "A comparative analysis of logistic regression, random forest and KNN models for the text classification". In: *Augmented Human Research* 5.1 (2020), pp. 1–16.

[Son+07] Wanpeng Song et al. "Question Similarity Calculation for FAQ Answering". In: *Proceedings of the Third International Conference on Semantics, Knowledge and Grid*. SKG '07. USA: IEEE Computer Society, 2007, pp. 298–301. ISBN: 0769530079. DOI: `10.1109/SKG.2007.247`. URL: `https://doi.org/10.1109/SKG.2007.247`.

[SP97] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[Sru21] Sruthi. "Understanding Random Forest". In: (2021). URL: `https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/`.

[Ste17] Bruno Stecanella. "Support Vector Machines (SVM) Algorithm Explained". In: (2017). URL: `https://monkeylearn.com/blog/introduction-to-support-vector-machines-svm/`.

[SUV18] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-attention with relative position representations". In: *arXiv preprint arXiv:1803.02155* (2018).

[SWY75] Gerard Salton, Anita Wong, and Chung-Shu Yang. “A vector space model for automatic indexing”. In: *Communications of the ACM* 18.11 (1975), pp. 613–620.

[Tay53] Wilson L Taylor. ““Cloze procedure”: A new tool for measuring readability”. In: *Journalism quarterly* 30.4 (1953), pp. 415–433.

[Xin21] Denny Zhou Xinying Song. “A Fast WordPiece Tokenization System”. In: (2021). URL: `https://ai.googleblog.com/2021/12/a-fast-wordpiece-tokenization-system.html`.

A Abbreviations

NLP	Natural Language Processing
ML	Machine Learning
DL	Deep Learning
TF-Record	TensorFlow Record
RF	Random Forest
LR	Logistic Regression
SVM	Support Vector Machines
SVC	Support Vector Classification
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
GRU	Gated Recurrent Unit
biGRU	Bi-directional Gated Recurrent Unit
biLSTM	Bi-directional Long Short Term Memory
CNN	Convolutional Neural Network
BERT	Bidirectional Encoder Representations from Transformers
LM	Language Modeling
MLM	Masked Language Modeling
NER	Named-Entity-Recognition
NLTK	Natural Language Toolkit
GPU	Graphics Processing Unit
TPUs	Tensor Processing Units
JSON	JavaScript Object Notation
NE	Named-Entity
W2V	Word2vec
Tf-idf	Term Frequence, Inverse Document Frequency
LSI	Latent Semantic Indexing
LSA	Latent Semantic Analysis
Transformer-XL	Transformer - (Extra Long)
XLNet	Extra Long Network (Named after Transformer-XL)
GPT	Generative Pre-trained Transformer
ELMo	Embeddings from Language Models
ULMFiT	Universal Language Model Fine-tuning
GPT	Generative Pre-trained Transformer

B Code and Dataset

The code and Dataset can be accessed from my github repository link provided bellow. The repository is private at the moment. It will be public once the evaluation of this thesis complete. Both the code of this work and the dataset created and used will be accessible to anyone with the following link.

https://github.com/Ahmad-billz/Question-to-Question-Matching-Using-Semantic-Relatedness-In-Home-Improvement-Domain

Declaration of Academic Integrity / Eidesstattliche Erklärung

Ich erkläre, dass ich die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe und dass alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind. Mit der aktuell geltenden Fassung der Satzung der Universität Passau zur Sicherung guter wissenschaftlicher Praxis und für den Umgang mit wissenschaftlichem Fehlverhalten vom 31. Juli 2008 (vABlUP Seite 283) bin ich vertraut. Ich erkläre mich einverstanden mit einer Überprüfung der Arbeit unter Zuhilfenahme von Dienstleistungen Dritter (z.B. Anti-Plagiatssoftware) zur Gewährleistung der einwandfreien Kennzeichnung übernommener Ausführungen ohne Verletzung geistigen Eigentums an einem von anderen geschaffenen urheberrechtlich geschützten Werk oder von anderen stammenden wesentlichen wissenschaftlichen Erkenntnissen, Hypothesen, Lehren oder Forschungsansätzen.

Passau, 7. April 2022

Ahmad Bilal Sohail

I hereby confirm that I have composed this scientific work independently without anybody else's assistance and utilising no sources or resources other than those specified. I certify that any content adopted literally or in substance has been properly identified. I have familiarised myself with the University of Passau's most recent Guidelines for Good Scientific Practice and Scientific Misconduct Ramifications from 31 July 2008 (vABlUP Seite 283). I declare my consent to the use of third-party services (e.g., anti-plagiarism software) for the examination of my work to verify the absence of impermissible representation of adopted content without adequate designation violating the intellectual property rights of others by claiming ownership of somebody else's work, scientific findings, hypotheses, teachings or research approaches.

Passau, 7. April 2022

Ahmad Bilal Sohail