Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

A question-answering machine learning system for FAQs

Sazzadul Haque

Orientador(es) | Paulo Miguel Quaresma

Teresa Gonçalves

Évora 2021

Universidade de Évora - Escola de Ciências e Tecnologia

Mestrado em Engenharia Informática

Dissertação

A question-answering machine learning system for FAQs

Sazzadul Haque

Orientador(es) | Paulo Miguel Quaresma

Teresa Gonçalves

Évora 2021

A dissertação foi objeto de apreciação e discussão pública pelo seguinte júri nomeado pelo Diretor da Escola de Ciências e Tecnologia:

Presidente	|	Irene Pimenta Rodrigues (Universidade de Évora)
Vogais	|	José Saias (Universidade de Évora) (Arguente) Teresa Gonçalves (Universidade de Évora) (Orientador)

Évora 2021

Acknowledgement

Firstly, from the bottom of my heart, I would like to thank my supervisors, Dr. Paulo Miguel Torres Duarte Quaresma and Dr. Teresa Cristina de Freitas Gonçalves for their consistent support and guidance during the whole thesis. I am extremely grateful to my supervisors, for giving me a friendly environment and perfect guidance, just like a family. I am very thankful for finding such guardians in this foreign land.

I want to thank my parents and my younger sister for believing in me. Their support has helped me to push forward, even when I thought I couldn't achieve my goals.

Lastly, I must thank all my friends for supporting me, especially my international friends who never let me feel I am abroad.

Sazzadul Haque
Évora, April 2021

Contents

List of Figures

List of Tables

Acronyms

BERT Bidirectional Encoder Representations from Transformers

BoW Bag-of-Word

CBOW Continuous Bag of Words

CLESA The cross-language explicit semantic analysis

CNN Convolutional neural network

ESA Explicit Semantic Analysis

FAISS Facebook Artificial Intelligence Similarity Search

FAQ Frequently Asked Questions

FCNN Fully-connected neural network

GLSA Generalized Latent Semantic Analysis

HAL Hyperspace Analogue to Language

IR Information Retrieval

KB Knowledge Base

LCS Longest Common Sub-string

LDA Latent Dirichlet Allocation

LSA Latent Semantic Analysis

NGD Normalized Google Distance

NLP Natural Language Processing

NLTK Natural Language Toolkit

NN Neural Network

PMI-IR Point wise Mutual Information - Information Retrieval

QA Question Answering

RNN Recursive Neural Network

SCO-PMI Second-order co-occurrence point wise mutual information

SemEva Semantic Evaluation

SMC Simple Matching Coefficient

STS Semantic Textural Similarity

SVD Singular value decomposition

TF-IDF Term Frequency Inverse Document Frequency

WHO World Health Organization

Abstract

With the increase in usage and dependence on the internet for gathering information, it's now essential to efficiently retrieve information according to users' needs. Question Answering (QA) systems aim to fulfill this need by trying to provide the most relevant answer for a user's query expressed in natural language text or speech. Virtual assistants like Apple Siri and automated FAQ systems have become very popular and with this the constant rush of developing an efficient, advanced and expedient QA system is reaching new limits.

In the field of QA systems, this thesis addresses the problem of finding the FAQ question that is most similar to a user's query. Finding semantic similarities between database question banks and natural language text is its foremost step. The work aims at exploring unsupervised approaches for measuring semantic similarities for developing a closed domain QA system. To meet this objective modern sentence representation techniques, such as BERT and FLAIR GloVe, are coupled with various similarity measures (cosine, Euclidean and Manhattan) to identify the best model. The developed models were tested with three FAQs and SemEval 2015 datasets for English language; the best results were obtained from the coupling of BERT embedding with Euclidean distance similarity measure with a performance of 85.956% on a FAQ dataset. The model is also tested for Portuguese language with Portuguese Health support phone line SNS24 dataset.

Keywords: Question Answering, Closed Domain QA systems, Similarity Measures, Sentence Embedding, BERT, Unsupervised Learning, Machine Learning

Sumário

Um sistema de pergunta-resposta de aprendizagem automatica para FAQs

Com o aumento da utilização e da dependência da internet para a recolha de informação, tornou-se essencial recuperar a informação de forma eficiente de acordo com as necessidades dos utilizadores. Os Sistemas de Pergunta-Resposta (PR) visam responder a essa necessidade, tentando fornecer a resposta mais relevante para a consulta de um utilizador expressa em texto em linguagem natural escrita ou falada. Os assistentes virtuais como o Apple Siri e sistemas automatizados de perguntas frequentes tornaram-se muito populares aumentando a necessidade de desenvolver um sistema de controle de qualidade eficiente, avançado e conveniente.

No campo dos sistemas de PR, esta dissertação aborda o problema de encontrar a pergunta que mais se assemelha à consulta de um utilizador. Encontrar semelhanças semânticas entre a base de dados de perguntas e o texto em linguagem natural é a sua etapa mais importante. Neste sentido, esta dissertação tem como objetivo explorar abordagens não supervisionadas para medir similaridades semânticas para o desenvolvimento de um sistema de pergunta-resposta de domínio fechado. Neste sentido, técnicas modernas de representação de frases como o BERT e FLAIR GloVe são utilizadas em conjunto com várias medidas de similaridade (cosseno, Euclidiana e Manhattan) para identificar os melhores modelos. Os modelos desenvolvidos foram testados com conjuntos de dados de três FAQ e o SemEval 2015; os melhores resultados foram obtidos da combinação entre modelos de embedding BERT e a distância euclidiana, tendo-se obtido um desempenho máximo de 85,956% num conjunto de dados FAQ. O modelo também é testado para a língua portuguesa com o conjunto de dados SNS24 da linha telefónica de suporte de saúde em português.

Palavras chave: Pergunta-Resposta, sistemas de Pergunta-Resposta de domínio fechado, Medidas de similaridade, *Embbeding* de frases, BERT,

Chapter 1

Introduction

A Question is a sentence or expression that requests specific information. An Answer is a specific and reliable information about a question. A question is always asked for gathering knowledge; on the other hand, the answer is the knowledge.

Before the age of the internet, finding an answer was difficult, but the internet seems to make life easier. There's a need to find an answer on a question? Just search it on the internet, and lots of answers will be presented. Now the problem is that there are so many answers on every possible question, that finding the desired answer is sometimes difficult. Search engines like Google, present the desired answer for a specific questions like *'Capital of Angola'* or *'what is the bitcoin price?'*; but a question like *'what is the bus ticket price from Lisbon to Porto?'* will give countless website links and not the desired answer. The main reason is that a search engine is not a Question Answering(QA) system. To solve this problem Question Answering(QA) systems come to the scene.

A system that can automatically present relevant answers to a natural language query is known as a QA System. Based on the answering domain, a QA system can be divided into two types: open domain, and closed domain. An open domain QA system can answer any kind of question; a closed domain one answers only for a specific domain. Though, open domain systems can accept and answer all types of questions, they are less accurate and more costly (memory, time) when compared to a closed domain one.

Closed domain QA systems are popular for asking Frequently Asked Questions (FAQ). FAQs collate the set of questions and answers that are most important (key questions) for a service provider. For example, the answer to the question *'the ticket price from Lisbon to Porto?'*, can be found on most FAQ section websites that provide bus services. Nonetheless, finding

the desired question from a FAQ can be time consuming and tedious. That's why many organizations are getting interested in closed domain QA systems for providing answers to their FAQs. This thesis proposes a system for a closed domain QA.

1.1 Motivation

The motivation for this thesis is the study and development of a QA model for Botschool[1]. Botschool is a project from Altice Labs[2] that provides a framework of virtual assistants for various enterprises. In 2019 Altice Labs collaborated with the University of Évora for improving the Botschool framework. The system was initially supervised and the challenge of this project was to formulate an unsupervised approach for closed domain QA Systems. This project gave me an opportunity to work on Altice Labs' Botschool.

1.2 Objectives

The aim of this thesis is to propose an unsupervised closed domain QA system for FAQ. To achieve this objective the following tasks are pursued:

- study different sentence representation techniques and similarity approaches;
- survey existing approaches for developing an unsupervised closed domain QA system;
- propose an unsupervised closed domain QA system using modern sentence representation techniques and similarity measures;
- develop an unsupervised closed domain QA system by implementing the proposed approaches;
- test the developed system using datasets of different domains for English and Portuguese languages and,
- identify the best performing model with best test results.

[1] https://botschool.ai/
[2] https://www.alticelabs.com/

1.3 Approach

This work presents an approach for developing an automatic FAQ answering system. To achieve the objective mentioned in the previous section, an unsupervised machine learning approach is proposed. It comprises of two modules:

- sentence representation using BERT and FLAIR embedding;
- similarity measurement between the (embedded) query asked by the user and the (embedded) frequently asked questions using a similarity measure (cosine similarity, Euclidean distance and Manhattan distance).

1.4 Main contributions

The main contributions of this work are:

- a survey and analysis on different state-of-art approaches on sentence representation;
- a survey and analysis of different state-of-art approaches on similarity measures;
- a set of closed QA models (using different sentence representation techniques and similarity measures) and their analysis.

1.5 Thesis Outline

This thesis is comprised of six chapters. Chapter 2 describes the QA system: section 2.1 contains the details and architecture of the QA system and 2.2 and 2.3 describe different sentence representation techniques and similarity measures, respectively. Chapter 3 presents the related work on similarity measurement with different approaches and

Chapter 4 describes the proposed model architecture and the tools used for its development; Chapter 5 presents the results obtained along with a discussion and Chapter 6 is the concluding chapter of this thesis: it comprises a short summary and future work.

Chapter 2

Question Answering

Question Answering is a branch of artificial intelligence in NLP. It aims at building a system that can automatically provide an answer to the user's question expressed in natural language. This chapter aims at describing Question Answering system along with two NLP techniques: sentence representation and similarity measure, which are used for building a question answering system.

2.1 Question Answering System

"The purpose of an QA system is to provide correct answers to user questions in both structured and non-structured collection of data." In Natural Language Processing, Question Answering (QA) is a big research field. The increasing demands of Question Answering(QA) Systems are creating more research opportunities. We are using QA System almost every day either passively or actively.

From telecommunications to websites QA Systems are very popular. There was a time when we used to look for 'Frequently Asked Questions' for general information but it is now outdated. Whenever we want to use a service, a bot popups and asks *"how can I help you?"*. The bot is actually an Artificial Intelligence (AI) QA System. The current challenge for researchers is to improve the bot's ability to answer general queries such that, when a human interacts with a QA System bot, the humans experience is as rich as it is when conversing with another human at the service desk.

2.1.1 Types of Question Answering (QA) Systems

The QA System is generally defined as an AI system that takes question from human in natural language then process the question to give the best fit corresponding answer. A QA System usually returns more than one answer. The retrieved answers are sorted on the scale of relevance to the question by ranking / similarity algorithms.

Based on answering domain, a QA system can be of two types: (1) Open Domain and (2) Close Domain.

Open Domain
An Open Domain QA system as the name suggests, is a QA system that accepts all types of questions as input and, tries to find best suited answer for it. A user has the freedom to ask any questions, from *"how are you?"* to *"what is today's stock exchange rates?"* to this system. Open Domain QA system generally uses a large collection of database for searching answers. As these systems can answer any kind of question, their database is exceptionally huge. The recent open domain QA systems use internet as their database, such as Wikipedia, Yahoo Answer, etc. Apple Siri, Amazon Alexa are some good examples of the same. This kind of QA system usually takes users query in voice input mode, then converts it to text, and searches it on the internet and presents the result.

Open Domain QA system can answer from any domain of question, but it has few drawback also. Finding answer from a very big database(ex: Wikipedia) is very challenging, time consuming and costly. Due to this reason it usually presents short answer to the user. Researcher always face two major challenges when they are working on Open Domain QA system. First, the system may not retrieve the document(golden document) that contain the correct answer, and second, rank algorithm was not able to give top rank to the correct answer. In the second case even though the correct answer is already retrieved but the user gets false result.

Closed Domain
QA Systems build for a specific domain are known as closed domain QA System. Unlike Open domain, Closed domain QA System is dedicated to only certain types of question and answering. For example the chat bot that appears on banking or credit card websites are closed domain QA. Nowadays almost all the websites have this kind of QA System, they are majorly used for answering simple FAQ or user queries regarding the particular website and its services. BASEBALL [22] is the first closed domain QA System which was built in 1961. Since then there were many remarkable work done in the

field of closed domain QA System including LUNAR, ELIZA, etc. [80, 78]

Closed domain question answering system deals with the queries of a specific domain hence their database is relatively smaller. Therefore, they are faster, and give higher accuracy than open domain QA system.

2.1.2 Architecture of Question Answering (QA) System

To understand the architecture of a QA System it is important to know the terminologies associated with it. They are as follows:

- *question phrase* is referred to the part of the question which tells what is looked for,
- *question type* is the classification of the questions into different categories based on its purpose,
- *answer type* refers to the class or set of answers that are sought for the question,
- *candidate passage* is defined as a unit that is retrieved by the system in order to answer the question. The retrieved unit can be anything from sentences to documents or even multimedia,
- *candidate answer* is the set of answers ranked according to its relevance as potential answers. [53]

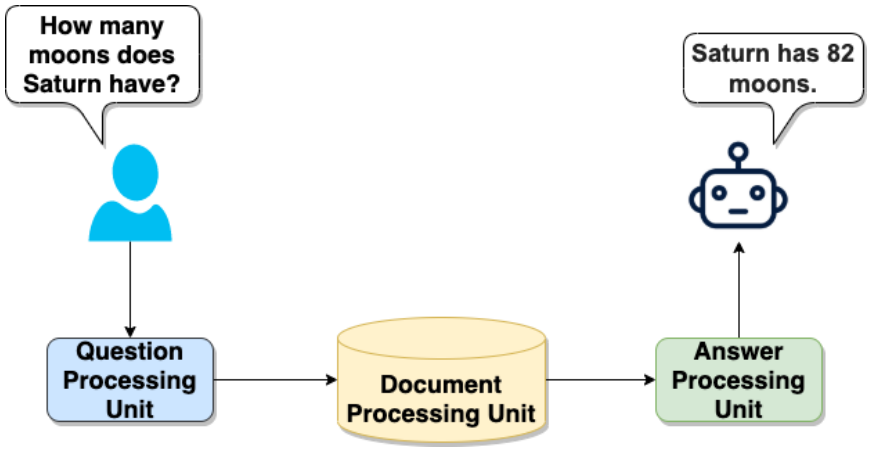

Figure 2.1: QA System Architecture

Figure 2.1 shows the general architecture of a QA system. It contains three major components, namely:

- *Question Processing* from users input.
- *Document Processing* according to user's question in order to generate all possible answers.
- *Answer processing* to present the most reliable answer.

The **Question Processing Unit** receives the question as input from the user in natural language in order to analyze and classify it. Firstly, the question is analyzed to find its type. Questions are generalized on the basis of types of answer they expect, they are factoid, list, definition and complex question [52] . The questions that ask about simple fact and it can be answered in few words like, *Earth has how many moons?*, as knows as factoid questions. The questions that expect group of entities as answers are list questions, for example, *who were the Presidents of USA from 1980 to 2020?.* Definition questions ask for a summary or short passage / paragraph as an answer, like *what is the process of photosynthesis?.* A complex question is the one that is about the information in context and may not have a direct answer, for example *Does Jack has a son?.* For such questions usually merging of passages or sentences are required. Algorithms like Round-Robin, Raw Scoring and Logistic Regression are used for merging the passages [11]. Question classification is a major step in question processing. There are two approaches for question classification, manual and automatic [81]. Hand crafted rules are used in manual classification of the questions. Though classification by hand crafted rules are accurate but its tedious and not flexible. The automatic classification of questions is done using machine learning algorithms by feature extraction. Automatic classification is more flexible, and provides reasonable accuracy with enough training data.

The task of **Document Processing unit** is to extract relevant set of documents, paragraphs or sentences based on the question phrase and question type. This is done with the help of natural language processing techniques. This unit can result a dataset or a neural model that serves as a pool from which the final answer is extracted.

The Answer Processing unit extracts the answers from the retrieved dataset or neural model. It may rank the answer by using ranking algorithm, based on its relevance on the question. It is important to extract short and meaningfully answers to the question thus various extraction techniques [42, 79, 34] are applied in order to deal with ambiguities and get the best fit answer.

Apart from the general architecture, a QA System is also defined by the models that are used for answer extraction, they are as follows:

- *Natural language processing (NLP) QA system* uses NLP techniques like POS tagging, embedding and other machine learning methods to extract answers from the retrieved result of document processing unit.

- *Information Retrieval (IR) QA system* uses search engine to retrieve answers and then apply filters and ranking algorithm on the retrieved dataset to get the best suited answer. It generally follow three steps, they are: retrieve relevant document from huge database, extract candidates answers from retrieved data, and finally rank the candidate answers to detect the most accurate answer. A retriever is used for retrieving and ranking documents. The task of the retriever is to retrieve the data according to the users question, after that the retriever score and rank that data to give the best fit result. To retrieve and rank the data different Information Retrieval (IR) techniques, such as, TF-IDF, Okapi BM25, etc and, rank algorithms are used. Finally a ranker ranks the candidate answer and determine the final answer. For ranking the answers, neural network is used. Neural Network is most efficient than other ranking algorithm, because neural network rank the answer based on the question and its context.

- *Knowledge Base (KB) QA system* is used when a structured dataset is available to find the answers. For example, finding an answer from a well defined entity and relation ontology of medical domain. Ontology can be considered a more refined knowledge base than relational database [5]. In this type of QA System, database queries are used for extracting the answers. To perform database queries on an ontology, structured languages like SPARQL [74] are used.

- *Hybrid QA system* is the one that takes advantage of all the resources available to find and extract the correct answer for the user's question. It is the combination of IR, NLP and KB QA [17]. IBM Watson [50] is a most popular example of this approach.

2.2 Sentence Representation

Sentence representation is a very important task in Natural Language Processing (NLP). It greatly helps in understanding the context of a sentence. It is recognized as one of the core task in NLP. Sentence representation is vastly used in document summarization, machine translation, sentiment analysis, dialogue system, etc. The aim of sentence representation is to encode valuable semantic information of a sentence into real valued vectors. The vector representation of sentences are also used in matching learning and deep learning approach for sentence classification.

Before deep learning approaches emerged, two methods of sentence representation were popularly used, namely: Term Frequency-Inverse Document Frequency(TF-IDF) and Bag-of-Word(BoW). These two methods convert a sentence into a vector depending on the size of the vocabulary. The major drawback of these methods were the huge vector size, sometimes it can even reach millions. The reason for huge vector size is its reliance on the vocabulary size, and, large vocabulary mean more information. Another major drawback is, these methods ignore the structural information and the sequence of words in a sentence, which plays an important role in understanding the context of the sentences. Recent methods for sentence representation solve these problems by using deep learning models such as embedding. TF-IDF, BoW and Embeddings are discussed in this section.

2.2.1 TF-IDF

TF-IDF [31, 57, 58, 56] method computes the vector from a document using two parts, Term Frequency (TF) and Inverse Document Frequency (IDF).

Term Frequency (TF) is measured by counting the occurrences of a term in a document. The probability of a term to be present in a document is higher for large documents than that of short length documents. To normalize this, the frequency of a term is divided by the total number of term of the document.

$$TF = \frac{\text{Number of times term appears in the document}}{\text{Total number of terms in the document}}$$

Inverse Document Frequency (IDF) on the other hand measures the importance of a specific term. TF gives equal importance to all the terms however, stop words such as: "*are*", "*it*", "*the*", etc. have higher occurrence than the terms that posses more meaning. To deal with this IDF assigns weights to the terms, lowers weights are assigned to frequent terms and higher weights are assigned to the terms that occur rarely. The formula to compute IDF is,

$$IDF = \log_e\left(\frac{\text{Total number of documents}}{\text{Number of documents with a specific term in it}}\right)$$

2.2.2 BoW

The Bag of Word (BoW) [23] is the simplest method for representing a sentence as a vector. BoW first makes a vocabulary list from the given document and then assigns 1 to the position of the word where it occurs in

the vocabulary and 0 if the term is not present in a sentence. For example, consider a three sentence document, where,

- S1: 'I like to watch action movies.'
- S2: 'I also like to watch football.'
- S3: 'I dont like horror movies.'

All the unique words are selected for building a vocabulary list thus, the vocabulary list contains 10 words whereas the document contain 19 words. Based on this the vocabulary of the given sentences would be:

$vocabulary = \{I, like, to, watch, action, movies, also, football, don't, horror\}$

Table 2.1 shows the vector representation of the example sentence with BoW model.

	I	*like*	*to*	*watch*	*action*	*movies*	*also*	*football*	*don't*	*horror*
S1	1	1	1	1	1	1	0	0	0	0
S2	1	1	1	1	0	0	1	1	0	0
S3	1	1	0	0	0	1	0	0	1	1

Table 2.1: BoW vector representation

2.2.3 Embeddings

Embeddings are a modern approach for sentence representation. It is a technique that represents word in a vector space. The words that have similar meaning are given similar representation using deep learning methods.

Word2vec
Word2vec is one of the most popular embedding that is used in Natural Language Processing. Word2vec was created by Google research team led by Tomas Mikolo. It learns vector representation of words by constructing a vocabulary from the training text datasets. The output vector file can be used in future as features for the input text corpus. To find the similar words distance measuring algorithms can be performed in the word vectors. Word2vec has two main algorithms for learning the text corpus namely: (1) Continuous Bag of Words (CBOW) [45] and, (2) Skip gram [45].

The **CBOW** model is developed as, it uses continuous distributed representation of the context [45]. This model tries to predict the next word using

context of each word given as input. The context can be any arbitrary number of words. This arbitrary number is determined by the context window. The context window represents the length of neighbour of a given word that will be used as a context.

CBOW model uses a Neural Network (NN) to convert word into vectors. The input layer of this NN takes the context word as input. The size of input vector is same as the size of vocabulary and it is represented as V. The single hidden layer takes the weighted sum from input layer as input. The hidden layer does not have any activation functions, and it consists of N neurons. The output layer takes input from hidden layer's output. Output layer is the only layer where softmax is used as an activation function.

The architecture of **Skip Gram** is similar to that of CBOW. However, unlike CBOW, skip gram attempts to predict the context of the word instead of trying to predict the word in a given context. During the training process, Skip Gram generates different errors for every context word. In order to obtain the resulting vector, an element wise addition is performed on each error vector, which is used during the back propagation process. Another major difference that Skip Gram holds is the working of context window. Skip Gram uses random window size that ranges between one to the maximum window size.

GloVe

Global Vector (GloVe) is an unsupervised machine learning algorithm for getting vector representation of words. It was released in 2014 and is developed by researches from Stanford University.[49] GloVe captures the meaning of words by using word co-occurrence method. Word co-occurrence statistics defines the meaning to words by tacking into account how often two words appear together is a document. For example word '*ice*' occurs more frequently with '*solid*' then it does with '*gas*' and the word '*water*' co occurs more frequently with '*liquid*' then with '*solid*'.

BERT

Bidirectional Encoder Representations from Transformers (BERT) is a methods of pre-training language representations to create a model for "general purpose language understanding" [14]. The created models are then used for various NLP tasks, like question answering. BERT was first released in 2018 by Google. It is the first unsupervised, bidirectional deep learning embedding model.

As the name suggest, BERT tries to understand the context of the word from both the directions (left and right). For example, in a sentence '*I made*

a bank deposit', the context of the word '*bank*' is based on both left context (I made a..) and right context (deposit). BERT outperforms the previous models for word embedding, including word2vec and GloVe as they generate single representation of each word in vocabulary, meaning, the word '*bank*' for both '*river bank*' and '*bank transfer*' have the same representation.

BERT model is pre-trained using large text corpus like wikipedia. The model is trained by masking approximately 15% of the words in an input sentence, and then aims at predicting the masked word. For example:

Input Sentence : I made a [MASK1] deposit. Then I went [MASK2]. Label: [MASK1] = bank, [MASK2] = home

Masking of the words is done through a deep bidirectional Transformer encoder. BERT can also be trained on single language corpus to learn the co-relation between sentences for predicting the next input sentence.

FLAIR
FLAIR text embedding library was proposed in 2018 [4]. It allows the user to combine various character, word and document embeddings including Flair, BERT and ELMo. Flair embedding is referred to as Contextual string embedding which is based on character level language modelling. It represents words as a sequence of characters. The word representation is obtained by its surrounding context, meaning, words in different context like in the phrases '*eye ball*' and '*base ball*', the word '*ball*' has different representation.

2.3 Similarity Measures

The measures of finding how close two pieces of text are is known as text similarity measures. Text similarity is an important task in the building applications like, question answering, essay scoring, topic detection, information retrieval, text classification, etc. Texts can be close or similar to each other in two behaviours, lexical or semantic. Words are lexically similar to each other if they have the same character sequence, like, eat and ate. Words are semantically similar to each other if they possess the same meaning or often used in same context, like, amazing and wonderful. Both lexical and semantic similarity are used depending on the nature of the task. For instance, finding words that rhyme with each other, character similarity in words is required, like, *top*, *shop*, *pop*, *mop*, etc. However, for text understanding semantic similarity is more significant. For instance, the sentence *"What are the tourist attraction in Rome?"* and *"What are the best places to visit in Rome?"* only have *"Rome"* in common but expect the same answer.

There are multiple algorithms that help in finding text similarity. Based on their nature, they are divided into three major types, namely:

- String Based Similarity,
- Knowledge Based Similarity,
- Corpus Based Similarity.

There are various algorithms in all the categories, some of the popular ones are discussed in their respective section.

2.3.1 String Based Similarity Methods

String based similarity measures work on string or character sequences. A metric is used to calculate the distance between two text for approximate string matching or comparison, the metric is called string similarity metric or string distance metric. Based on their operation, string based similarity is further divided into two categories, namely: character based and token based similarity measure.

Character Based Similarity Measures
Character based similarity measure algorithms are the ones that work on character sequences for finding the similarity / dissimilarity between text. Some of the famous character based similarity measure are discussed under this section.

The **Longest Common Substring** (LCS) algorithm [73] finds the longest common chain of characters between two strings. It computes the similarity between two text by comparing their longest common sub-sequence. It is calculated by the following equation:

$$LCSubstr(S_1, S_2) = max_{1 \leq i \leq m, 1 \leq j \leq n} LCSuff(S1_{1,2,...i}, S2_{1,2,..j})$$

here m and n are the length of the string S_1 and S_2 respectively, and $LCSuff$ is a function to find the longest common suffix of every probable prefixes of S_1 and S_2.

Jaro [68, 20] is an algorithm to calculate the distance for measuring similarity between two strings. It is based both on number and order of the common character sequences between two strings. The Jaro distance score is 0 if there is no similarity and 1 if the complete match is found. It takes into account spelling variation and majorly used for record linkage. The

Jaro distance (d_j) between two string S_1 and S_2 is defined by the following equation:

$$d_j = \begin{cases} 0 & \text{if } m = 0, \\ 1/3\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

where m is the number of matching character, t is half of number transposition. Two characters of string S_1 and S_2 are matched only when they are same and positioned close to each other. The distance between characters should be not more than $\frac{max(|s1|.|s2|)}{2} - 1$.

Jaro winkler [68, 20] is an extension of Jaro distance algorithm. It is semantic similarity measure for two strings, that is useful for short string matching. It makes use of prefix scale to compute the rating of the strings that are matched from the beginning for defined prefix length. The higher the Jaro Winker distance (d_{jw}) between the two strings the more similar they are. It is calculated by the following equation:

$$d_{jw} = d_j + (lp(1 - d_j)),$$

where l is the prefix length and p is the prefix scale.

Damerau-Levenshtein [33, 25] is an algorithm to find the distance between two strings by calculating the number of transformations that are required to convert one string to another. The conversion is done by operations like, insertion, deletion or substitution of one character or changing the position of two adjacent characters. The least number of operations required the closer the strings are.

Needleman-Wunsch [15, 76] is a dynamic programming algorithm and the first of its kind for biological sequence matching. It is an alignment method that applies global alignment to find the best fit arrangement over two biological sequences. It is an optimal matching algorithm. It is used when the two sequences are of same length, and posses considerable level of similarity.

Smith-Waterman [15, 12] is a variant of Needleman-Wunsch algorithm. It is also a dynamic programming algorithm for finding the alignment of biological sequences, but instead of global it uses local alignment over conserved domain to find the best fit arrangement in sequences. To find the similarity it compares parts of the string and optimizes the results. It is useful for very large dissimilar sequences that contain chunks of similarity.

N-gram [32, 82] is a probabilistic language algorithm majorly used for predicting the next term in the sequence. It generates the sequence of n items for a given text. N-gram can be used for segmenting both characters and

tokens. For example, bi-gram consists of two items in a sequence of text, character bi-grams of '*work*' are '*wo*', '*or*', '*rk*'. Similarly, token bi-grams of '*I work for fun*' are '*I work*', '*work for*', '*for fun*'. The similarity is calculated by dividing the similar number of n grams by the total number of n grams.

Token Based Similarity Measures

Token or term based similarity measure algorithms work on term or word sequences for text matching. Some of the famous token based similarity measures are discussed in this section.

Cosine similarity measure [59, 51] is the similarity between two non zero vectors. It finds the angle of cosine between them, if the two vectors have same direction their cosine similarity is 1 and if vectors are at 90° their similarity is 0. Euclidean dot product is used to measure the cosine of vectors, the following equation shows the formula for the same.

$$a \cdot b = \|a\|\|b\| \cos\theta.$$

For two vectors A and B the cosine similarity is defined by the following equation:

$$\cos\theta = \frac{A\dot{B}}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

here A_i and B_i are the components of A and B vectors.

Block Distance [70] is also known as Manhattan distance, city block distance, snake distance or $L1$. It is defined as the distance that needs to be covered while moving from one data point to another, if grid path is followed. The block distance (d_1) between two points (a and b) is calculated by adding the difference of their components, as shown in the following equation:

$$d_1(a,b) = \|a-b\|_1 = \sum_{i=1}^{n} |a_i - b_i|$$

Euclidean distance [59] or $L2$ distance is the measure of distance between two points in Euclidean space[1]. The position of points are Euclidean vectors and distance is the square root of the sum of squared differences between

[1]Euclidean space is an elemental space in geometry and can have any non negative integer dimensions.

corresponding elements. The Euclidean distance (d_2) for two points (a and b) is defined as following equation:

$$d_2 = d(a,b) = d(b,a) = \sqrt{\sum_{i=1}^{n}(b_i - a_i)^2}$$

The **Jaccard Similarity Coefficient** [21, 59] is also known as the Jaccard index. It is used to find the similarity and dissimilarity between two finite set of strings. The similarity via Jaccard coefficient is calculated as the common terms, or intersection of terms that are divided by the union of terms in the two strings. The dissimilarity is computed by subtracting the Jaccard index of similarity from 1. For two set of strings(P and Q), Jaccard index (J) is defined as following equation:

$$J(P,Q) = \frac{|P \cap Q|}{|P \cup Q|} = \frac{|P \cap Q|}{|P| + |Q| - |P \cup Q|}$$

Simple matching coefficient [59] is also a vector based approach for computing similarity and dissimilarity between two strings. It counts the number of terms which are non zero vectors. It accepts strings as a collection of n binary attributes and calculates the squared Euclidean distance between two binary vectors. The SMC for string S and T is defined as following equation:

$$\begin{aligned} SMC &= \frac{\text{Number of matching attributes}}{\text{Total number of attributes}} \\ &= \frac{s_{00} + s_{11}}{s_{00} + s_{01} + s_{10} + s_{11}} \end{aligned}$$

where s_{00} are the total attributes as 0 in S and T, s_{11} are the total attributes as 1 in S and T, s_{01} are the total attributes as 0's and 1's in S and T and likewise, s_{10} are the total attributes as 1's and 0's in S and T.

Dice's Coefficient [46] is also known as Sorensen-Dice index or Sorensen index. It is majorly used to find the presence or absence of data in the dataset. It is calculated as twice the count of similar terms divided by the total terms of the compared set / string. The following equation shows Dice's coefficient for set X and Y.

$$QS = \frac{2|X \cap Y|}{|X| + |Y|}$$

$|X|$ and $|Y|$ are the number of elements of set X and Y, QS is the quotient

of similarity that ranges between 0 and 1. To calculate the similarity of two strings (P and Q), the Dice's coefficient is computed by bigrams as shown in the following equation:

$$sim = \frac{2n_t}{n_p + n_q}$$

where, n_t is the count of common bigram, and n_p and n_q are the total number of bigrams in string P and Q.

2.3.2 Knowledge Based Similarity Methods

Knowledge based similarity methods are majorly used for finding the semantic similarity between two strings. It is called knowledge based as it makes the use of underlying information sources for measuring similarity. Information source may consist of ontology, dictionaries, thesauri, or lexical databases like WordNet, Wiktionary, etc. These fundamental data sources help in finding structured representation of concepts, terms along with their meanings and semantic relations. Thus, providing highly efficient and unambiguous semantic measure [72]. Based on the types of approaches used, Knowledge based similarity measure is further divided into three categories namely: feature based, edge counting based and information content based methods. They are discussed in the sections respectively.

Edge Counting Methods

Edge counting methods are based on counting the edges of an ontology graph to calculate the similarity between them. An ontology graph is a graph which connects words taxonomically. The distance between terms / words define their similarity, the less the distance the more similar they are. *Path measure* [52] finds the similarity via edge counting methods based on two deductions. First, the words that are present at deep down in the graph hierarchy contain more specific meanings, and second, the words from first deduction are more likely to be similar to each other even if they have same distance as the other pairs that represent more generic concepts. Thus, claiming that similarity is inversely proportional to the shortest path length between two terms [11].

A Least Common Subsumer (LCS) is a common ancestor that is shared between two terms in an ontology graph. *Wup measure* [81] is the one take into account the depth of the terms along with the depth of their LCS in the graph and count the edges between each term to compute the similarity. In this measure the shortest path(sim_{path}) between two words (w_1 and w_2) is

defined as:

$$sim_{path}(w_1, w_2) = \frac{1}{1 + min\_len(w_1, w_2)}$$

and the wup measure(sim_{wup}) defined as:

$$sim_{wup}(w_1, w_2) = \frac{2depth(w_{lcs})}{depth(w_1) + depth(w_2)}$$

where, w_{lcs} is the LCS of w_1 and w_2.

Feature Based Methods

Feature based methods take in consideration word / term features to calculate the semantic similarity between them. Both common and not common features are taken into account. The alike features add to the increase in similarity, and the contrasting features add to the dissimilarity of the terms. In the context of knowledge based methods, features are properties of words, like its meaning, neighbouring concepts, etc. [72]. The meaning of a word defined in a dictionary is known as word's gloss. Gloss based similarity measures take into account a glossary / dictionary to find similar terms by the overlap of words in gloss. The more common words in gloss the more similar two terms are. The *Leak measure* [6], calculate the similarity between terms by calculating the overlap of words in their gloss and concept gloss to exploit their relatedness in WordNet ontology. The relatedness score is defined as the sum of the squares of the overlap lengths. Similarly [29] proposed a method of feature based semantic similarity that calculates the overlap of words in respective Wikipedia pages to find similar words. A Vector pair approach also uses WordNet to find the similarity. It creates a cooccurrence matrix for every word from the dataset that occur in the WordNet gloss. Then every gloss / concept is represented with a vector that is the average of these cooccurrence vectors.

Feature based similarity measures are heavily dependent on ontology which contain required feature information. This dependency has both advantages and disadvantages. The system will be very efficient and accurate if the ontology is fully incorporated with semantic features, however, most of the ontology only provide taxonomic relations in the context of semantic features which leads to reduced efficiency [72].

Information Content Based Methods

Information Content (IC) is referred to the information that is derived from the word when it appears in context of something [71]. IC defines the specificity of the words by using Inverse Document Frequency (IDF). IDF is estab-

lished on an information theory principle that says that, the more specific the word is, the less they appear in a document. Based upon this IC is defined as:

$$IC(c) = \log^{-1} p(c)$$

where $p(c)$ is the probability of encountering the term 'c' in a document 'C' [42]. The higher the IC the more specific and well described the term is. Information content based methods take advantage of IC to determine the similarity. The *res measure* [55] finds the similarity based on the concept that if two terms share a LCS then they posses more information as the IC of LCS is higher. It means IC of the terms is equal to that of their LCS and hence is always greater than or equal to zero. Then for terms (c_1 and c_2) res measure (res) is define as:

$$res(c_1, c_2) = IC_{lcs}.$$

Lin [38], and Jiang and Conrath [28] proposed the extensions of res measure. *Lin measure* takes into account the twice the IC value of LCS for two terms and divides it with the combined IC values of the terms. This is done to exploit the attributes of individual information from both terms along with the common information they share with LCS. However, the *jcn measure* by Jiang and Conrath calculates the distance between terms by adding up the IC values of the two terms and then subtracting it with the twice of IC value of their LCS. After that the similarity is computed using the following equation:

$$jcn(c_1, c_2) = \frac{1}{1 + dis_{jcn}(c_1, c_2)},$$

where $dis_{jcn}(c_1, c_2)$ is defined as:

$$dis_{jcn}(c_1, c_2) = IC_{c1} + IC_{c2} - 2IC_{lcs}$$

One or more ontologies can be used in information content based methods, thus depending on IC methods can be categorized as mono or multi ontological methods.

2.3.3 Corpus Based Similarity Methods

A corpus is a large collection of texts (written or spoken). Corpus based similarity methods exploit the information from a large corpus or dataset to measure the semantic similarity. These methods work on the deduction that similar words occur together in a document more frequently, rather than utilizing any external ontology or dictionary for finding semantic relatedness. Word embedding is widely used in these methods to identify the underlying similarities between words in training corpus. There are many methods for calculating corpus based similarity. Some of the famous ones are discussed under this sections.

Latent Semantic Analysis
Latent Semantic Analysis (LSA) [34] is the most popular corpus based semantic similarity measure. It works on the principle that words that appear in similar context or among same piece of texts share common meaning. To calculate the similarity first a co-occurrence matrix is constructed where, rows represent unique words, columns represent paragraphs and their corresponding cell represent the word count in respective paragraph. As the matrix is formulated with a large corpus its dimensions are huge. Thus, a mathematical technique, singular value decomposition (SVD) [79] is used for reducing the matrix's dimensions. SVD preserves the similarity structure among the words by only reducing the number of columns and retaining the rows. Then each word is represented as a vector and the similarity is calculated by finding the cosine angel between them.

Generalized Latent Semantic Analysis
Generalized Latent Semantic Analysis (GLSA) [43] is an extension of LSA. When the word vectors are generated from the corpus which is heterogeneous in nature, then the performance of LSA degrades [5]. GLSA method was proposed to overcome this problem. It is based on semantically motivated pair-wise term for finding the word, sentence or document vector. This approach can combine any type of measure for semantic association of terms along with different dimensionality reduction techniques. To give the final result, term document matrix is computed by combining the weights of the term vectors in a linear fashion.

Explicit Semantic Analysis
The Explicit Semantic Analysis (ESA) [18] method is based on measuring similarity with the aid of Wikipedia concepts. This method is useful when working on arbitrary or unrestricted natural language text [17]. It uses

machine learning techniques to represent terms as high dimensional vectors. All the Wikipedia concepts are represented as an attribute vector of the words that comprise the concept. Then TF-IDF is used to link every vector with its associated concepts. A threshold is set for linking the concepts with vectors. By this, the input text is converted into a weighted vector of concepts called "*interpretation vectors*" and similarity between them is calculated using the cosine measure.

Hyperspace Analogue to Language

Hyperspace Analogue to Language (HAL) [40, 41] method is built using word co-occurrence matrix in semantic space. A group of words is considered as a "*window*". The rows and columns of the matrix correspond to the words and the associated cells represent the association weight between the words. Associated weight between words is calculated by sliding the window size which can vary depending upon the corpus. The association weight is inversely proportional to the distance from the focused word. This is done based on the deduction that words that are at close neighbours to the focused word tends to provide more semantic to the focused word then the words that are farther. Dimensionality of the matrix can be reduced by dropping the columns with low entropy and the similarity is measured using Euclidean or Block distance between word vectors.

Normalized Google Distance

Normalized Google Distance (NGD) [13] method is based on Google search engine. It calculates the similarity by taking into consideration the results obtained when a set of keywords are queried with Google search engine. This method is built on the deduction that words that occur together more frequently in web pages are close / similar to each other. Thus, the words that have similar meaning have shorter Google distance between them then to those that have different meanings or used in different context. Normalized Google distance is defined as:

$$NGD(t_1, t_2) = \frac{max\left\{\log f(t_1), \log f(t_2) - \log f(t_1, t_2)\right\}}{\log G - min\{\log f(t_1), \log f(t_2)\}}$$

where,$t_1$and t_2 are the two terms, functions $f(t_1)$ and $f(t_2)$ return the number of hits from the Google search engine when the respective term is searched, function $f(t_1, t_2)$ returns the number of hits when the terms $t_1$and t_2 are searched together and G is the total number of pages searched by the Google. Hence if the two terms occur independently but never occur together then the NGD between them is infinite, and if they always occur together then the distance between them is zero, or equivalent to the coefficient between

x and y squared.

Apart from these some methods for finding corpus based similarity are Word-Alignment models [69], Pointwise Mutual Information - Information Retrieval (PMI-IR) [74], Second-order co-occurrence pointwise mutual information (SCO-PMI) [26, 27], Latent Dirichlet Allocation (LDA) [66], Dependency-based models [3], Word-attention models [36] and The cross-language explicit semantic analysis (CLESA) [8].

Chapter 3

State of the Art

In the early days of textual similarity, two pieces of text were assumed to be similar if they contained the same character sequence. With the advancement of technology, it was understood that two snippet of text can be similar even when they don't posses the same alphabetic order, like synonyms. Bag of Words (BOW) and Term Frequency - Inverted Document Frequency (TF-IDF) are accounted as early works in finding similarity between words, sentences and documents. These measures led the path for machine learning and deep learning methods to find semantic similarity in text. Before this, core mathematical formulas were used to accomplish this task [1, 37, 55]. This chapter reviews some related work which use deep learning and hybrid methods for finding textual similarity.

3.1 Deep Learning Methods

Neural Networks (NN) have shown very good potential in the field of Natural Language Processing. They are also used in finding similarity between texts. This section discusses in details about the sate of art work for determining semantic similarity between sentences using recurrent, recursive, and convolutional neural network.

3.1.1 Recurrent and Recursive Neural Networks

To study the effectiveness of recurrent and recursive neural networks, [60] combines these neural networks with word vector representations to predict the semantic similarity of sentence and phrase pairs. Both the neural networks take structured set of words or tokens as input, but both the models

treat its input differently. For word (w) to vector $L(w)$ conversion, GolVe embedding is used. The dimension of vectors for per-trained GolVe is either 50 or 100, and it uses Wikipedia and Gigaword for training the model. SemEval 2015 data was used for testing the system.

Recurrent Neural Networks

Recurrent Neural Networks are very powerful deep learning model. It takes sequence of tokens as input. [60] uses every token sequence to update the hidden state. The formula of hidden state (h_i) for a sequence of tokens $(w_1, w_2, w_3, ...w_n)$ is:

$$h_i = \int (U.h_{i-1} + W.L[w_i] + b)$$

Where,

- U, W and b are parameters learned by the model.
- $\int$ is a non-linear function, applied element-wise to the vector of its arguments.

For each training example, the h_0 is set to zero vector. For two sequences of tokens, $(w_1, w_2, ...w_n)$ and $(v_1, v_2, ...v_m)$ the above equation generates two hidden vectors $h_n^{(w)}$ and $h_m^{(v)}$. These two vectors then produce a vector of probability (p) that determines if the sentence pair belongs to a corresponding similarity category or not, the following equation is used for the same:

$$p = softmax(W_s.[\frac{h_n^w}{h_m^v}] + b_s)$$

where

- W_s and b_s are parameters learned by model.
- $softmax$ is the function defined on k-dimensional vectors x by:

$$[softmax(x)]_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

Finally the proposed model is trained using the cross-entropy loss, given a vector of probabilities p for a training pair of sentences. If the correct

similarity category corresponds to index i of p, the model will incur the loss $L = -\log(p_i)$.

$U, W$and W_s are initialised using a random Gaussian in the interval of [-0.01, +0.01], and b and b_s are initialised as zero vector.

Recursive Neural Network
Recursive Neural Networks(RNN) are used in most of the NLP tasks because it can take advantages of known linguistic structure. Rather than working on a sequence of tokens, RNN takes binary tree as an input. For this NN to be able to generate a parse tree, [60] have trained "CKY parser on the QuestionBank and Penn treebanks using pyStatParser". Then, the parse trees are converted to binary tree using simple heuristics. For each leaf of binary tree with word w the GloVe vector $L[w]$ is assigned. A vector is calculated for each node N with left child N_L and right child N_R using the formula:

$$h_N = \int (W_L.h_{N_L} + W_R.h_{N_R} + b)$$

where,

- W_L, W_R and b are parameters learned by the model.
- $\int$ is a non-linear function, applied element-wise to the vector of its arguments.

Using the above equation [60] generates two vector $h_{S_1}$and h_{S_2} for two sentences $S_1$and S_2. The rest of the model is designed exactly similar to the Recurrent Neural Network, as mentioned in 3.1.1 with the replacement of $h_n^{(w)}$with h_{S_1} and $h_m^{(v)}$with h_{S_2}.

3.1.2 Convolutional Neural Networks

Yang Shao [65] uses convolutional neural network (CNN) to determine Semantic Textural Similarity (STS). Five components of a convolutional neural network (CNN) are proposed to accomplish this task, they are:

- Enhance GloVe word vectors by adding handcrafted features.
- Transfer the enhanced word vectors to a more proper form by a convolutional neural network.

- Max pooling over every dimension of all word vectors to generate semantic vector.
- Generate semantic difference vector by concatenating the element-wise absolute difference and the element-wise multiplication of two semantic vectors.
- Transfer the semantic difference vector to the probability distribution over similarity scores by a fully-connected neural network.

Operation performed for data pre-processing are, tokenisation by Natural Language Toolkit (NLTK) [10], punctuation removal, converting all words to lower-case, and all sentences are given a static length ($l = 30$) with zero vectors [24]. After pre-processing the data, word to vector conversion was done using pre-trained GloVe [48] embedding system. If a word does not exist in the pre-trained embedding system, then it set to the zero vector. After sentence embedding, three features were added to the sentences vector. They are as follows:

- If both sentence have common word, a $TRUE$ flag is added to that specific word vector, otherwise a $FALSE$ flag is added.
- If a word is a number and the same number appears in the other sentence, add a $TRUE$ flag to the word vector of the matching number in each sentence, otherwise add $FALSE$ flag.
- The part-of-speech (POS) tag of every word according to NLTK is added as a one-hot vector.

After features enhancing, the vectors are feed to a one layer convolutional Neural Network (CNN). This CNN has $n = 300$ one dimensional filters. This dimension is the same as of the enhanced word vectors. Relu [47] activation function, and max pooling [61] are used in CNN to transform word embedding to sentence embedding.

Then[65] creates a semantic difference vector by the following formula:

$$\vec{SDV} = (|\vec{SV1} - \vec{SV2}|, \vec{SV1} \bullet \vec{SV2})$$

Where,

- $\vec{SDV}$ = the semantic difference vector.
- $SV1, SV2$ = the semantic vectors of the two sentences.

- $\bullet$ = Hadamard product which generate the element-wise multiplication of two semantic vectors.

This semantic difference vector (600 dimension) is fed to a two layer fully-connected neural network (FCNN) for probability distribution over six similarity lanes used by STS. The first layer of FCNN uses 300 units and its activation function is tanh. The second layer of FCNN produces the similarity label probability distribution with six units, using softmax activation function and calculating the similarity score.

CNN for finding textual similarity is also used in [77]. In this work, to calculate similarity for words that are semantically related but possess different spelling, like: *'irrelevant'*, and *'not related'*, each word is represented by a distributed vector . A semantic matching vector is calculated for each word vector based on all the word vectors in the other sentences. Then a semantic matching vector is used to decompose all the word vectors into similar and dissimilar components. Then CNN is used for feature extraction from similar and dissimilar vectors. Lastly, similarity assessment function is applied to get the final similarity score over feature vectors.

In [77], given a pair of sentences S and T, the task is to calculate a similarity score $sim(S,T)$. The pre-trained embedding of Mikolov(2013) is used to transform the sentences (S and T) into sentence matrices $S = [s_1, ..., s_i, ..., s_m]$ and $T = [t_1, ..., t_j, ..., t_n]$ where, s_i and t_j are d-dimension vectors of the corresponding words, and m and n are sentence length of S and T respectively.

To check if the sentences are a paraphrases of each other, every word is treated as a primitive semantic unit, and a semantic matching vector si for each word s_i is calculated by composing part, or full word vectors in the other sentence (T). Then a similarity matrix $A_{m\times n}$, where each element $a_{(i,j)} \epsilon A_{m\times n}$ is calculated by the cosine similarity between words s_i and t_j. Three different semantic matching functions were used over $A_{m\times n}$, namely: Global, local-w and max are used to calculate the semantic matching vector. Here, (w is the size of the window that is consider to be centered at k (the most similar word position)

To calculate the word similarity of phrases, [77] considers all the words in a phrase. For example: *'sockeye'* means *'red salmon'*, and when *'sockeye'* goes to this system it will be matched with *'salmon'* only. To solve this problem [77] has proposed a Decomposition method. Decomposition is the breaking of semantic matching vectors of a phrase into two components. They are identified as: similar component $s_i +$ (or t_j+), and dissimilar component $s_i -$ (or t_j-). The Decomposition function is as follows:

$$[s_i^+; s_i^-] = \int_{decomp} (s_i, \hat{s_i})$$

$$[t_j^+; t_j^-] = \int_{decomp} (t_j, \hat{t_j})$$

The significance of the dissimilar parts alone between two sentences has a great effect of their similarity [67]. Taking this into consideration, the similar component matrix and dissimilar component matrix were composed into a feature vector. To generate this feature vector two-channel convolutional neural networks(CNN) has been used with max pooling and for the filters various order of n-grams are used such as, unigram, bigram and trigram.

To compute similarity from two feature vectors the following formula is used:

$$sim(S, T) = \int_{sim} (\vec{S}, \vec{T})$$

where,

- S and T are a pair of sentence.
- A linear function used for sum up all the features and *sigmoid* function used to calculate similarity score between 0 to 1.

3.2 Hybrid Methods

3.2.1 Combining statistical and supervised methods

In [7] authors propose a system to measure Semantic Textual Similarity(STS) by combining various methods for similarity measurement. Usage of multiple methods for STS based on surface-level and has been proposed in 2007 by [44], [35], [19]. Authors have proposed two major limitations of these methods, namely, 1. Measures are typically used in separation. Thereby, the assumption is made that a single measure inherently captures all text characteristics which are necessary for computing similarity. 2. existing measures typically exclude similarity features beyond content per se, thereby implying that similarity can be computed by comparing text content exclusively, leaving out any other text characteristics. In this research authors are focusing on second issue.

The system is based on DKPro[1]. In the data pre-processing phase, data is tokenised and lemmatised using Tree-Tagger implementation [62]. After that multiple similarity methods are applied in the pre-processed data to generate the results. The methods are: Character / word n-grams where $n = 2, 3, ..15$ is used, Longest common subsequence, Longest common substring. Vector space model such as Explicit Semantic Analysis(ESA) [19] is used. For the vector space model to calculate textural similarity, Jaccad similarity method has been used. Lexical Substitution System based on supervised word sense disambiguation [9] is used.

After that the feature combination step uses the pre-computed similarity scores and combines their log-transformed values using linear regression classifier from the WEKA tool [3] with 10-fold cross validation.

3.2.2 Combining lexical semantic net with deep learning semantic model

In [2] authors have presented three systems for measuring the semantic similarity between two sentence. For testing the system English semantic textual similarity (STS) dataset of 2015 and 2016 were used. The first system depends on corpus statistics with lexical database and the second system on deep learning semantic model. The third system is a linear combination of first and second system. Authors have used five dataset for testing the systems. They are: answer-answer, headlines, plagiarism, post editing and question-question. For the data pre-processing, contractions like URL, e-mail addresses and parenthetical expressions using WordNet antonyms were removed. Also, the stops word were removed and the data was lemmatized. But, stop words for answer-answer and question-question data-set were not removed, because many pairs only contain stop words, for eg: *'Can you do this?, You can do this too.'*

Method 1. First method calculates similarity between sentence by the combination of semantic similarity and syntactic similarity. After pre-processing, firstly word vector are joined (JWV) by collecting unique words that occur in the sentence pair. For computing semantic similarity each sentence is mapped into a vector as: if ith word in JWV is present in the sentence assign value 1 to the ith entry, otherwise calculate the semantic similarity score between the ith word and each word in the sentence and then select the highest score to the ith entry using WordNet with the equation:

[1]DKPro is a collection of software components for natural language processing based on the apache UIMA framework.

$$s(W_i, W_j) = e^{al}.\frac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}}$$

Where l is the length of shortest path between words w_i and w_j, which returns a measure of depth in WordNet, and α, β are constant with the value of 0.2 and 0.45 respectively.

If $s(w_i, w_j)$ is greater than threshold value 0.2 than a value has been set to the score, otherwise 0 has been set to score. Then the ith entry is normalized by the equation:

$$I(W) = 1 - \frac{\log(n+1)}{\log(N+1)}$$

Where N is the total number of words in the corpus and n is the frequency of the word W in the corpus. After that semantic similarity (S_{sem}) has been calculated by cosine similarity.

To compute syntactic similarity, each sentence is mapped to a syntactic vector. The dimension of syntactic vector is same as the size of JWV. If the ith word in the JWV occurs at the jth position in the sentence, then the value of the ith entry in the vector is j. If the ith word does not exist in the sentence, the value of the ith entry is the position of the most similar word obtained from WordNet using $s(w_1, w_2)$. Syntactic similarity is calculated by the following equation:

$$S_{syn} = 1 - \frac{|o_1 - o_2|}{|o_1 + o_2|}$$

Where o_1 and o_2 represent syntactic vector for sentences s_1 and s_2 respectively.

Final sentence pair similarity is defined by combining semantic similarity and syntactic similarity as follows:

$$S(s_1, s_2) = \omega S_{sem} + (1 - \omega) S_{syn}$$

The value of $\omega = 0.85$ has been used.

Method 2. The second method is built on a Deep Structured Semantic Model (DSSM). This approach is a deep learning method and it maps short textual strings, such as sentences, to feature vector in a low dimensional

semantic space. DSSM used deep neural network architecture to represent a sentence in the semantic vector space. To reduce dimension of bag of word vector, DSSM employs a novel word hashing method. The hashing method attach a starting and an ending mark for each word and split the word into letter trigrams. For example: *girl-> #girl# -> #gi, gir, irl, rl#*. Then each word represented by a vector of letter trigrams is used as input to the deep neural network. The deep neural network layers divided into three parts, they are: word hashing layer, hidden layers and top layer and the layer functions are follows:

$$l_1 = W_1 x$$

$$l_i = \int (W_i l_{i-1} + b_i), i = 2, 3, ...N - 1$$

and

$$y = \int (W_N l_{N-1} + b_N)$$

where,

- x = is the input term vector
- y = the output vector
- $l_i (i = 1, 2, ...N - 1)$ = is the hidden layers
- W_i = is the ith weight matrix
- b_i = is the ith bias, and
- $\int(.)$ = is the tan activation function.

The word hashing layer generates the feature vector then feed to the hidden layers and top layer generate the semantic feature vector. The cosine similarity has been performed on each pair of texts semantic feature vector to measure the semantic similarity between the pair.

Method 3. The third method is the linear combination of method 1 and method 2 as follows:

$$S_{method3} = \alpha . S_{method1} + \beta . S_{method2}$$

Where α and β are constant with the value of 0.5.

Chapter 4

Proposed Methodology

This chapter proposes the system architecture of a closed domain Question Answering system along with the tools and techniques that were analysed and used for building the same.

4.1 System Architecture

The proposed system architecture consists of six modules namely: embedded corpus, input, question processing unit, similarity measuring unit, answer database and output. Figure 4.1 shows the system architecture diagram.

Embedding Corpus. This unit acts as the database of the targeted vectored question sets. For example, if the system is used for the University of Évora FAQ system, firstly, the system requires a set of sample questions and their corresponding answer. Secondly, the system embeds all the questions using the Question Processing Unit. Thirdly, embedded questions are mapped to their corresponding vector according to the question serial number. Finally it saves this information in the disk. This is a one time task, meaning, it is done once when the system is established in a device.

Input. Input is natural language text that is taken from the user as a question. Pre-processing of the input text, such as tokenization and stop word removal (between others), should be avoided. This is because sometimes stop words play a vital roll in short questions and by removing them this valuable information can be erased. For example, in the question *'What if I am not available?'*, after removing stop words, the question becomes *'what available?'* changing the meaning of the question.

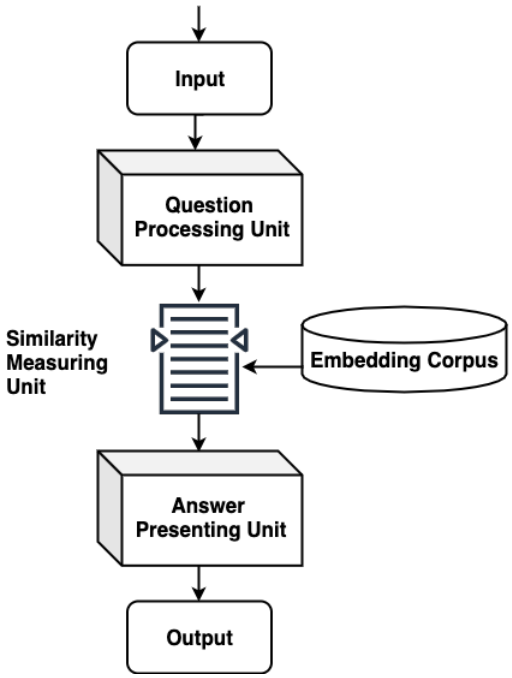

Figure 4.1: Diagram of System Architecture

Question Processing Unit. The Question Processing Unit takes the original text as input and feeds it to an embedder to convert into its vector representation. The main task of this unit is to convert a sentence into a $m \times n$ dimension vector using the context of the sentence, where m is the number of questions and n is the number of dimensional space.

Similarity Measuring Unit. The output of the Question Processing unit is fed as the input to this unit. This unit calculates the similarity between the query vector and the embedded questions using similarity measurement methods. The query vectors similarity is calculated to every question vector resulting into a similarity score. The best scored vector is presumed as the desired question.

Answer presenting Unit. The answer presenting unit takes the index of best scored questions from Similarity Measuring unit as its input and presents the answer according to the index number (The question and its corresponding answer have same index number).

4.2 Tools and Approaches

The proposed system was developed using Python(v3.7.5). The vast libraries and ease of use has made the Python programming language popular not only among developers but also among researchers. Artificial Intelligence technologies such as Neural Networks, Embeddings, Image processing, etc are easy to implement using Python.

This section describes the tools used for accomplishing the tasks of (1) sentence to vector transformation and (2) measuring similarity between sentences.

4.2.1 Tools for Sentence Transformation

Two embedding models were tested for transforming sentences into vectors, namely: Sentence Transformer [54] and FLAIR [4].

Sentence Transformer. Sentence Transformer is a Python embedding library. The architecture is same as BERT's embedding. It is a very efficient library for finding semantic text similarity and semantic search. Sentence Transformer includes over 100 language models. English and Portuguese models were been used for the development of this work.

It includes a set of different pre-trained models for each language. Figure 4.2 shows the top 19 most accurate models. The best four models for Semantic Textual Similarity were selected. They are:

- *roberta-large-nli-stsb-mean-tokens*
- *roberta-base-nli-stsb-mean-tokens*
- *bert-large-nli-stsb-mean-tokens*
- *distilbert-base-nli-stsb-mean-tokens*
- *neuralmind/bert-base-portuguese-cased*

The stated accuracy performance for all the above mentioned models is over 85% [39]. From the above mentioned methods, *roberta-base-nli-stsb-mean-tokens* was selected for English Language. This model has a performance score of 85.44% and was previously stated that is faster than the other three language models (2300 sent./ sec on V100 GPU) [75].

A pre-trained BERT language model *neuralmind/bert-base-portuguese-cased* was used for Portuguese dataset. This model has achieved the state-of-art performance for the Named Entity Recognition, Sentence Textual Similarity

Model Name	Base Model	Pooling	Training Data	STSb Performance (Higher = Better)	Speed (Sent. / Sec on V100 GPU)
stsb-roberta-large	roberta-large	Mean Pooling	NLI+STSb	86.39	830
stsb-roberta-base	roberta-base	Mean Pooling	NLI+STSb	85.44	2300
stsb-bert-large	bert-large-uncased	Mean Pooling	NLI+STSb	85.29	830
stsb-distilbert-base	distilbert-base-uncased	Mean Pooling	NLI+STSb	85.16	4000
stsb-bert-base	bert-base-uncased	Mean Pooling	NLI+STSb	85.14	2300
paraphrase-xlm-r-multilingual-v1	XLM-R	Mean Pooling	Paraphrase Data	83.50	2300
paraphrase-distilroberta-base-v1	distilroberta-base	Mean Pooling	Paraphrase Data	81.81	4000
nli-bert-large	bert-large-uncased	Mean Pooling	NLI	79.19	830
nli-distilbert-base	distilbert-base-uncased	Mean Pooling	NLI	78.69	4000
nli-roberta-large	roberta-large	Mean Pooling	NLI	78.69	830
nli-bert-large-max-pooling	bert-large-uncased	Max Pooling	NLI	78.41	830
nli-bert-large-cls-pooling	bert-large-uncased	CLS Token	NLI	78.29	830
nli-distilbert-base-max-pooling	distilbert-base-uncased	Max Pooling	NLI	77.61	4000
nli-roberta-base	roberta-base	Mean Pooling	NLI	77.49	2300
nli-bert-base-max-pooling	bert-base-uncased	Max Pooling	NLI	77.21	2300
nli-bert-base	bert-base-uncased	Mean Pooling	NLI	77.12	2300
nli-bert-base-cls-pooling	bert-base-uncased	CLS Token	NLI	76.3	2300
average_word_embeddings_glove.6B.300d	Word Embeddings: GloVe	Mean Pooling	-	61.77	34000
average_word_embeddings_komninos	Word Embeddings: Komninos et al.	Mean Pooling	-	61.56	22000

Figure 4.2: Top 19 BERT language model

and Recognizing Textual Entailment. It is trained with Brazilian Portuguese Language. [67]

FLAIR. FLAIR is a collection of NLP and embedding libraries. FLAIR embedding library supports many embedding techniques along with pre-trained language models. GloVe embedding with news-X English pre-train language model is used from this library. The news-X language model is trained with web, Wikipedia, Subtitles and News data. Over 1 billion word corpus have been used to train this language model [16].

4.2.2 Tools for Similarity measurement

For measuring similarity between sentences, three similarity measures were used, namely: cosine similarity, Euclidean distance and Manhattan distance. These measures were applied with the help of two libraries, FAISS and Scikit-learn.

Facebook Artificial Intelligence Similarity Search (FAISS) [30] was developed by Facebook research group, and is used for efficient similarity searching and clustering of dense vectors. It consists of different similarity measure algorithms. It creates a data structure from a given vector (in this case embedded corpus) in the RAM. This data structure is called '*index*'. To build an index it is mandatory to provide the dimension of the vector. Every time a new query comes, FAISS searches the index with the given query. This phenomena makes FAISS faster and efficient.

For testing cosine similarity and euclidean distance measures, three FAISS index models were used, namely:

- **IndexFlatL2.** IndexFlatL2 calculates similarity between two vector using euclidean distance.
- **IndexFlatIP.** IndexFlatIP uses cosine similarity to calculate the similarity between two vectors.
- **IndexIVFFlat.** IndexIVFFlat divides the corpus vector into k clusters and performs cosine similarity for the query vector.

Scikit-learn [63] is an open source Python library for using machine learning algorithms. It is used for calculating the Manhattan distance.

Chapter 5

Evaluation of the Proposed System

This chapter aims at describing the results that were achieved upon performing experiments on the proposed Question Answering System. The chapter also details the datasets that were used for the experiments and, the setup that was required to conduct the experiments on different datasets.

5.1 Dataset overview

The experiments were designed and evaluated on Portuguese and English Language dataset. This section briefly describes the datasets used for both languages.

5.1.1 Portuguese dataset

The Portuguese dataset has 269,662 sentences from the Portuguese Health support phone line SNS24, about medical symptoms of diseases which corresponds to 59 diseases. From this dataset 80% (215,729 sentences) data is used for building vector model and 20% (53,933 sentences) data is used for testing the model. 5.1 shows a few examples of the Portuguese dataset.

5.1.2 English dataset

For testing and evaluating the proposed system for the English language the following datasets were used: SemEval 2015 and three FAQ datasets [64].

No	*Symptoms of Diseases*	*Diseases*
1	Retenção urinária há 5 horas	PROBLEMA URINARIO
2	Tosse ha uma semana	Tosse
3	Edema facial e febre há 12 horas	PROBLEMA NA FACE
4	Febre e tosse ha 8 dias	Tosse
5	Sangue vivo na fralda	PROBLEMA URINARIO

Table 5.1: Portuguese Dataset (sns24 motivo classe)

SemEval 2015 dataset. Semantic Evaluation (SemEval) is series of international conference on NLP research that happens every year. It aims at improving the state-of-art technologies / methods on semantic analysis and contributing to the creation of quality dataset to address the new challenges in natural language semantics. One of the SemEval's task is Semantic Textual Similarity(STS), in this task researchers aim to contribute different methods for STS task. SemEval dataset consists of two parts, one for training and another for testing. It scores the sentence on the scale of zero to five, where, 5 means both sentence are same, and 0 means they are totally different from each other. For the testing of the proposed system, SemEval 2015 dataset is used. SemEval 2015 dataset consists of 2012, 2013 and 2015 similarity datasets. To test the proposed system the score range of zero to one is selected, where, score 1 means that the two sentence are most similar with each other and 0 means that the sentences are completely different. Table 5.2 and Table 5.3 shows few examples of SemEval's dataset and test dataset respectively.

No	*dataset*
1	A cat standing on tree branches
2	A large boat in the water at the marina
3	A passenger train waiting in a station
4	An Apple computer sitting on the floor
5	A jockey riding a horse

Table 5.2: Sample Data of SemEval 2015

FAQ dataset. FAQ dataset comprises of real life frequency asked question that were collected from different websites FAQ page. The test dataset is a collection of question that are asked by users. Three domains of FAQ dataset are used for testing the model, namely:

- RomeDataset: It consists of tourism FAQ for Rome, Italy.

No	*Query set*
1	A black and white cat is high up on tree branches
2	A large boat on the sea
3	A passenger train sits in the station
4	A Macintosh computer sitting on the floor
5	A jockey riding a horse in a pen

Table 5.3: Sample Query set of SemEval 2015

- COVID19 dataset: FAQ on Covid19 virus, collected form World Health Organization (WHO) website[1].
- Altice Labs dataset: It was provided by Altice Labs. This dataset consists of FAQ from their customer care services.

5.2 Experimental setup

The system returns the best n questions indexes, where n is a natural number ($n = 1, 2, 3, \dots$) and 1 is the best score. The index(es) of the best question(s) is(are) returned as output. It was tested that the desired answer is always on the top 5 questions($n = 5$). The experiments are performed using two embedding methods: BERT(2.2.3) and FLAIR(2.2.3), and 4 similarity methods: Cosine [59, 51], Euclidean [59], Manhattan [70] and cosine with cluster(4.2.2). Experiment setup comprises of coupling each embedding with all the four similarity methods in order to find the best pair of embedding and similarity method.

5.2.1 Portuguese dataset

For the Portuguese dataset firstly, 80% data is used to build the vector model using BERT embedding. Then, for the rest of the 20% data, each sentence is used to calculate the similarity with the embedded sentences using similarity measure methods.

The experiments were performed in two ways, namely: 'Top1' and 'Top3'. Top1 answer selects the first answer that was predicted by the system and then compares it with the "Diseases". If the query sentence and the result has the same Diseases class, then it is saved as a 'match' section and otherwise it is saved as 'not match'. Top3 selects the first three answers that were

[1]https://www.who.int/emergencies/diseases/novel-coronavirus-2019/question-and-answers-hub

predicted by the system and compares each answer with Diseases class. If any of the selected answer's diseases class matches with the query's diseases class, then it is saved as 'match' and otherwise it is saved as 'not match'.

5.2.2 English dataset

The English dataset are relatively small as compared to the Portuguese dataset. The queries are created manually for testing.

The dataset are tested by two embedding models namely: BERT and FLAIR. The pre-trained language model "roberta-base-nli-stsb-mean-tokens" is used for BERT and, "news-X" language model is used for FLAIR.

5.3 Results

5.3.1 Portuguese dataset

This section comprises of the results that were achieved by performing the experiments on the Portuguese language dataset by combining BERT and FLAIR embedding with cosine, euclidean and, cosine with cluster similarity measures. The results for Top1 and Top3 are calculated by number of match and not match found among the total queries.

Table 5.4 shows the results obtained by BERT embedding for Top1 and Table 5.5 shows the results obtained by BERT embedding for Top3.

Table 5.6 shows the results obtained by FLAIR embedding for Top1 and, Table 5.7 shows the results obtained by FLAIR embedding for Top3.

	Match	***Not Match***	***Match(%)***	***Not Match(%)***
Cosine	29,170	24,763	54.086%	45.914%
Euclidean	34,925	19,008	64.756%	35.244%
Cosine with cluster	34,510	19,423	63.987%	36.013%

Table 5.4: Portuguese Dataset Result(Top1 Result) - BERT

5.3.2 English dataset

The results are obtained from FAQ and SemEval dataset by combining BERT and FLAIR embedding with cosine, euclidean, Manhattan, and cosine with

	Match	***Not Match***	***Match(%)***	***Not Match(%)***
Cosine	38,087	15,846	70.619%	29.381%
Euclidean	43,102	10,831	79.918%	20.082%
Cosine with cluster	42,732	11,201	79.232%	20.768%

Table 5.5: Portuguese Dataset Result(Top3 Result) - BERT

	Match	***Not Match***	***Match(%)***	***Not Match(%)***
Cosine	20,593	33,340	38.183%	61.817%
Euclidean	28,003	25,930	51.922%	48.078%
Cosine with cluster	25,490	28,443	47.262%	52.738%

Table 5.6: Portuguese Dataset Result(Top1 Result) - FLAIR

cluster similarity measures. The results obtained from different model sets are evaluated in this section.

FAQ dataset: FAQ dataset consists of questions with their corresponding answers. Based on the domain of the dataset, a sample test dataset was created. It consist of queries that a user can have regarding the particular topic. As the system is designed to return the most suitable answer, the correctness of the answers are evaluated manually. The results are calculated using the following formula:

$$result = \frac{total\ correct\ answer}{total\ samplequestion\ in\ the\ test\ dataset.}$$

Table 5.8 shows the result of using BERT embedding with similarity measures.

Table 5.9 shows the result of using FLAIR GloVe embedding with the similarity measures.

SemEval dataset: As SemEval dataset does not contain any specify questions and answers, the above mentioned formula does not apply in this sce-

	Match	***Not Match***	***Match(%)***	***Not Match(%)***
Cosine	35,140	18,793	65.155%	34.845%
Euclidean	38,724	15,209	71.800%	28.200%
Cosine with cluster	37,178	16,755	68.934%	31.066%

Table 5.7: Portuguese Dataset Result(Top3 Result) - FLAIR

	Cosine	***Euclidean***	***Manhattan***	***cosine with cluster***
Rome dataset	78.947%	84.241%	84.241%	78.947%
Covide19	78.261%	86.956%	82.609%	73.913%
Altice Labs dataset	70.000%	75.000%	75.000%	66.412%

Table 5.8: FAQ Dataset Result - BERT

	Cosine	***Euclidean***	***Manhattan***	***cosine with cluster***
Rome dataset	36.842%	42.105%	52.632%	36.842%
Covide19	80.950%	71.428%	76.190%	71.428%
Altice Labs dataset	55.000%	55.000%	65.000%	50.000%

Table 5.9: FAQ Dataset Result - FLAIR with GloVe

nario. Thus, for comparing the predicted answer with the answers present in the dataset, the assortment is done using three conditions, namely:

- Relevant sentence
- Satisfactory sentence
- Irrelevant sentence

Relevant set consists of a sentences that is most similar to the sample sentence. Satisfactory set consists of two sentences that are not very similar but, are not completely dissimilar as well with the sample sentence. If the proposed system returns any of the sentence in Relevant set then it is scored as 1, if it returns the sentence in Satisfactory set then it is scored as 0.5 and, otherwise score 0 is assigned.

Table 5.10 shows three example of Relevant and Satisfactory set for SemEval dataset.

50 sample sentences from the dataset were tested. The results are calculated as:

$$result = \frac{Score\ of\ the\ sentence}{Total\ number\ of\ sentences}$$

Table 5.11 shows the results of using BERT and FLAIR GolVe embedding with the similarity measures on sample SemEval dataset.

Sentence	*Relevant Sentence*	*Satisfactory Sentence*
A jockey riding a horse in a pen.	A jockey riding a horse in a pen.	A jockey riding a horse. A woman equestrian riding a horse.
Bird with a green head and white chest perched on a tree branch.	Green and white bird perched on tree branch.	A small red and gray bird perched amongst the bare branches of a tree. A small bird perched on an icy branch.
A passenger train sits in the station.	A passenger train waiting in a station.	The train sits at the train station. A train at the train station.

Table 5.10: An example Relevant and Satisfactory set for SemEval dataset

	Cosine	*Euclidean*	*Manhattan*	*cosine with cluster*
BERT	74.242%	72.727%	74.242%	74.242%
FLAIR	69.697%	72.727%	71.212%	69.697%

Table 5.11: SemEval dataset result

5.4 Discussion

This section discusses the performance of model sets that were obtained from FAQ and SemEval dataset for English Language and Portuguese dataset for Portuguese language.

5.4.1 FAQ dataset

As this dataset is tightly with the proposed models, the discussion on the performance results from this dataset will help in selecting the best model for the proposed system. The procedure that was followed to select the best model is:

- comparing the results between embedding,
- selecting the best performed embedding,
- comparing the results between similarity measures for the selected embedding,

- selecting the best performed similarity measure,
- selecting the best performed model.

BERT vs FLAIR

The performance results obtained by combining BERT embedding with different similarity measures 5.8, ranges between 70% to 86.956%. The lowest result were obtained with Cosine similarity on Altice Labs dataset, and the best result was with Euclidean distance on Covid 19 dataset. On the other hand, the performance results obtained by combining FLAIR embedding with different similarity measures range between 36.842% to 80.95%. Cosine on Rome dataset gave considerably lower results but on the contrary, cosine also gave the best result on Covid 19 dataset.

Based on the range and fluctuation of results it can be determined that BERT embedding performs better then FLAIR for the proposed system.

Comparison between similarity measures

As BERT embedding has performed better than FLAIR 5.4.1, comparison between the results of similarity measures is done with BERT embedding.

The table 5.8 shows that Euclidean distance gives the highest performance with BERT in all the datasets. Its performance was highest with Covid 19 dataset, 86.956%, and lowest with Altice Lab's dataset, 75%. Yet, among the performances of other measures with Altice labs's data, Euclidean distance performs the best.

With BERT embedding, it can be seen that Manhattan and Euclidean perform equally on Rome and Altice Labs dataset. But, for Covid 19 dataset Euclidean distance performs better than Manhattan.

Based on the above analysis, it can be assumed that BERT + Euclidean Distance is the best performing model for the proposed unsupervised closed domain QA system.

5.4.2 SemEval dataset

The results from testing the model on SemEval dataset 5.11 shows that BERT embedding performs better than FLAIR embedding for the proposed system. However, cosine and Manhattan similarity measures have performed slightly better (by 1.515%) than Euclidean distance for SemEval dataset.

5.4.3 Portuguese dataset

For Portuguese language dataset, the best result is given by BERT embedding with Euclidean distance similarity measure. Hence it can be determined that BERT embedding is better then FLAIR embedding for Portuguese language and it works best with Euclidean distance similarity measure.

By obtaining the results from different language models, it can be determined that BERT + Euclidean distance is the best performing model for the proposed unsupervised closed domain QA system.

Chapter 6

Conclusions and Future Work

This chapter concludes the thesis by giving a summary of the work done followed by its future work.

6.1 Summary

The objective of the thesis was to propose and develop an optimized and effective unsupervised closed domain QA system. To achieve this objective, a detailed study and analysis was carried out on several techniques for developing a QA System. A survey was made on the recent techniques for sentence representation and similarity measurement. From the extensive survey on each of the topics, the best performing two sentence representation techniques: BERT and FLAIR with four similarity measures methods: Cosine, Euclidean, Manhattan, and cosine with the cluster were selected for developing the unsupervised QA system.

Then, a set of methods were developed by combining BERT and FLAIR with similarity measures. Each model was examined with three datasets: FAQ, SemEval and Portuguese data-set. Finally, the best performing model was identified: BERT sentence representation along with the Euclidean distance on Covid-19 FAQ dataset. This model gives the best performance with the highest accuracy of 86.956%.

It was also determined that BERT embedding performs better and faster then FLAIR and Euclidean and Manhattan similarity methods perform better then cosine similarity.

6.2 Future work

The proposed system can still be improved in several ways. The proposed model is currently working with English and Portuguese queries, but it is possible to develop a model to work with multilingual queries. To achieve this, a study on different language models needs to be conducted. From this thesis it was learned that if the language model is not trained with enough data then its performance will be considerably low. In such a case, a new language model needs to be made for that specific language.

Deep learning techniques were tried for the proposed model. Beside this deep learning method, QA system can be developed using Knowledge based, Information Retrieval, Hybrid approaches, etc. to determine the best approach for developing an automatic closed domain QA system.

Finally, token based similarity methods are used for finding similarity between two questions. Hybrid methods [7] already show good potential for measuring similarity between sentences. The system can be improved by trying hybrid methods to finding question similarity.

6.3 End note

The journey of the QA system started with the BASEBALL QA system from 1961 [22]. This system was developed with an IR technique. From then until now the technologies have evolved drastically.

Before two decades, statistical sentence representation like BoW, TF-IDF were very popular approaches for developing a QA system, but deep learning changed the concept of the QA system, and now we have high performance QA systems, like IBMs Watson. With recent development on deep learning approaches, the field of QA system is taking a new turn, and is in its way for reaching new heights of technological advancement.

Bibliography

[1] A method for measuring sentence similarity and its application to conversational agents.

[2] Naveed Afzal, Yanshan Wang, and Hongfang Liu. MayoNLP at SemEval-2016 task 1: Semantic textual similarity based on lexical semantic net and deep learning semantic model. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 674–679, San Diego, California, June 2016. Association for Computational Linguistics.

[3] Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. NAACL '09, page 1927, USA, 2009. Association for Computational Linguistics.

[4] Alan Akbik, Duncan Blythe, and Roland Vollgraf. Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649, 2018.

[5] Rie Kubota Ando. Latent semantic space: Iterative scaling improves precision of inter-document similarity measurement. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, page 216223, New York, NY, USA, 2000. Association for Computing Machinery.

[6] Satanjeev Banerjee and Ted Pedersen. Extended gloss overlaps as a measure of semantic relatedness. *IJCAI-2003*, 05 2003.

[7] Daniel Bär, Chris Biemann, Iryna Gurevych, and Torsten Zesch. UKP: Computing semantic textual similarity by combining multiple content similarity measures. In **SEM 2012: The First Joint Conference on Lexical and Computational Semantics – Volume 1: Proceedings of the main conference and the shared task, and Volume 2: Proceedings of the Sixth International Workshop on Semantic Evaluation (SemEval 2012)*, pages 435–440. Association for Computational Linguistics, 7-8 June 2012.

[8] Fabio Benedetti, Domenico Beneventano, Sonia Bergamaschi, and Giovanni Simonini. Computing inter-document similarity with context semantic analysis. *Information Systems*, 80:136 – 147, 2019.

[9] Chris Biemann. Creating a system for lexical substitutions from scratch using crowdsourcing. In *Language Resources and Evaluation*, volume Vol. 47, 03 2013.

[10] Steven Bird, Ewan Klein, and Edward Loper. *Natural Language Processing with Python*. O'Reilly Media, Inc., 1st edition, 2009.

[11] D. Chandrasekaran and Vijay Mago. Evolution of semantic similarity - a survey. *ArXiv*, abs/2004.13820, 2020.

[12] Michael Kinkley Yuma Tou Cherie Ruan, Erik Hoberg. Smith-waterman algorithm sequence alignment with dynamic programming. [Online; accessed 02-November-2020].

[13] R. L. Cilibrasi and P. M. B. Vitanyi. The google similarity distance. *IEEE Transactions on Knowledge and Data Engineering*, 19(3):370–383, 2007.

[14] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[15] Trinity College Dublin. Needleman-wunsch algorithm for sequence similarity searches, 2020. [Online; accessed 02-November-2020].

[16] flairNLP. Flair embeddings, 2018. [Online; accessed 02-November-2020].

[17] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. IJCAI'07, page 16061611, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc.

[18] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. volume Vol. 6, 01 2007.

[19] Evgeniy Gabrilovich and Shaul Markovitch. Computing semantic relatedness using wikipedia-based explicit semantic analysis. volume Vol. 6, 01 2007.

[20] geeksforgeeks. Jaro and jaro-winkler similarity, 2020. [Online; accessed 02-November-2020].

[21] Stephanie Glen. Jaccard index / similarity coefficient, 2020. [Online; accessed 02-November-2020].

[22] Bert F. Green, Alice K. Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: An automatic question-answerer. IRE-AIEE-ACM '61 (Western), page 219224, New York, NY, USA, 1961. Association for Computing Machinery.

[23] Zellig S. Harris. Distributional structure. WORD, 10(2-3):146–162, 1954.

[24] Hua He, Kevin Gimpel, and Jimmy Lin. Multi-perspective sentence similarity modeling with convolutional neural networks. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1576–1586, Lisbon, Portugal, September 2015. Association for Computational Linguistics.

[25] Frank Hofmann. Levenshtein distance and text similarity in python, 2018. [Online; accessed 02-November-2020].

[26] Aminul Islam and Diana Inkpen. Semantic text similarity using corpus-based word similarity and string similarity. *ACM Trans. Knowl. Discov. Data*, 2(2), July 2008.

[27] Md. Aminul Islam and Diana Inkpen. Second order co-occurrence PMI for determining the semantic similarity of words. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC'06)*, Genoa, Italy, May 2006. European Language Resources Association (ELRA).

[28] Jay J. Jiang and David W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *Proceedings of the 10th Research on Computational Linguistics International Conference*, pages 19–33, Taipei, Taiwan, August 1997. The Association for Computational Linguistics and Chinese Language Processing (ACLCLP).

[29] Yuncheng Jiang, Xiaopei Zhang, Yong Tang, and Ruihua Nie. Feature-based approaches to semantic similarity assessment of concepts using wikipedia. *Information Processing Management*, 51(3):215 – 234, 2015.

[30] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. *arXiv preprint arXiv:1702.08734*, 2017.

[31] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[32] Grzegorz Kondrak. N-gram similarity and distance. pages 115–126, 10 2005.

[33] Praveen Kumar, Narendra, Bibhu Vimal, Md Islam, and Bhardwaj Shashank. Approximate string matching algorithm. *International Journal on Computer Science and Engineering*, 2, 05 2010.

[34] T. Landauer and S. T. Dumais. A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge. *Psychological Review*, 104:211–240, 1997.

[35] Thomas Landauer, Peter Foltz, and Darrell Laham. An introduction to latent semantic analysis. *Discourse Processes*, 25:259–284, 01 1998.

[36] Yuquan Le, Zhi-Jie Wang, Zhe Quan, Jiawei He, and Bin Yao. Acv-tree: A new method for sentence similarity modeling. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 4137–4143. International Joint Conferences on Artificial Intelligence Organization, 7 2018.

[37] Y. Li, Z. A. Bandar, and D. Mclean. An approach for measuring semantic similarity between words using multiple information sources. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):871–882, 2003.

[38] Dekang Lin. An information-theoretic definition of similarity. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 296304, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[39] Y. Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, M. Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

[40] Kevin Lund and Curt Burgess. Producing high-dimensional semantic space from lexical co-occurence. *Behavior Research Methods Instruments Computers*, 28:203–208, 06 1996.

[41] Kevin Lund, Curt Burgess, and Ruth Atchley. Semantic and associative priming in high-dimensional semantic space. pages 660–665, 01 1995.

[42] Goutam Majumder, Partha Pakray, Alexander Gelbukh, and David Pinto. Semantic Textual Similarity Methods, Tools, and Applications: A Survey. *ComputaciÃy Sistemas*, 20:647 – 665, 12 2016.

[43] Irina Matveeva, Gina-Anne Levow, Ayman Farahat, and Christiaan Royer. Term representation with generalized latent semantic analysis. *Recent Advances in Natural Language Processing IV: Selected Papers from RANLP 2005*, 292, 01 2007.

[44] Rada Mihalcea, Courtney Corley, and Carlo Strapparava. Corpus-based and knowledge-based measures of text semantic similarity. volume 1, 01 2006.

[45] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv e-prints*, page arXiv:1301.3781, January 2013.

[46] Dr Candace Makeda Moore. Dice similarity coefficient, 2020. [Online; accessed 02-November-2020].

[47] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. ICML'10, page 807814, Madison, WI, USA, 2010. Omnipress.

[48] Jeffrey Pennington, Richard Socher, and Christopher Manning. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar, October 2014. Association for Computational Linguistics.

[49] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[50] Martin Potthast, Benno Stein, and Maik Anderka. A wikipedia-based multilingual retrieval model. In Craig Macdonald, Iadh Ounis, Vassilis Plachouras, Ian Ruthven, and Ryen W. White, editors, *Advances in Information Retrieval , 30th European Conference on IR Research, ECIR 2008, Glasgow, UK, March 30-April 3, 2008. Proceedings*, volume 4956 of *Lecture Notes in Computer Science*, pages 522–530. Springer, 2008.

[51] Selva Prabhakaran. Cosine similarity understanding the math and how it works (with python codes), 2020. [Online; accessed 02-November-2020].

[52] R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1):17–30, 1989.

[53] Sudha Rao and Hal Daumé III. Learning to ask good questions: Ranking clarification questions using neural expected value of perfect information. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2737–2746, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[54] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[55] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. *ArXiv*, abs/cmp-lg/9511007, 1995.

[56] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information Processing Management*, 24(5):513 – 523, 1988.

[57] Gerard Salton, Edward A. Fox, and Harry Wu. Extended boolean information retrieval. *Commun. ACM*, 26(11):10221036, November 1983.

[58] Gerard Salton and Michael J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, Inc., USA, 1986.

[59] Jayesh Salvi. Familiarity with coefficients of similarity, 2019. [Online; accessed 02-November-2020].

[60] Adrian L. Sanborn. Deep learning for semantic similarity. 2015.

[61] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. pages 92–101, 01 2010.

[62] Helmut Schmid. Probabilistic part-of-speech tagging using decision trees. In *Proceedings of the International Conference on New Methods in Language Processing*, 1994.

[63] scikit learn. scikit-learn machine learning in python. [Online; accessed 02-November-2020].

[64] SemEval-2015. Semeval-2015 task 2: Semantic textual similarity. [Online; accessed 02-November-2020].

[65] Yang Shao. HCTI at SemEval-2017 task 1: Use convolutional neural network to evaluate semantic textual similarity. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 130–133, Vancouver, Canada, August 2017. Association for Computational Linguistics.

[66] Roberta A. Sinoara, Jose Camacho-Collados, Rafael G. Rossi, Roberto Navigli, and Solange O. Rezende. Knowledge-enhanced document embeddings for text classification. *Knowledge-Based Systems*, 163:955 – 971, 2019.

[67] Fábio Souza, Rodrigo Nogueira, and Roberto Lotufo. BERTimbau: pretrained BERT models for Brazilian Portuguese. In *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*, 2020.

[68] statisticaloddsandends. What is jaro/jaro-winkler similarity?, 2019. [Online; accessed 02-November-2020].

[69] Md Arafat Sultan, Steven Bethard, and Tamara Sumner. DLS@CU: Sentence similarity from word alignment and semantic vector composition. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 148–153, Denver, Colorado, June 2015. Association for Computational Linguistics.

[70] Fred E. Szabo. The linear algebra survival guide. In Fred E. Szabo, editor, *The Linear Algebra Survival Guide*, pages 219 – 233. Academic Press, Boston, 2015.

[71] David Sánchez and Montserrat Batet. A semantic similarity method based on information content exploiting multiple ontologies. *Expert Systems with Applications*, 40(4):1393 – 1399, 2013.

[72] "David Sánchez, Montserrat Batet, David Isern, and Aida Valls". Ontology-based semantic similarity: A new feature-based approach. *Expert Systems with Applications*, 39(9):7718 – 7728, 2012.

[73] Michigan Tech. Testing text similarity, 2020. [Online; accessed 02-November-2020].

[74] Peter D. Turney. Mining the web for synonyms: Pmi-ir versus lsa on toefl. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, pages 491–502, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[75] UKPLab. Sentencetransformer pretrained models, 2019. [Online; accessed 02-November-2020].

[76] Stanford University. Needleman-wunsch algorithm, 2020. [Online; accessed 02-November-2020].

[77] Zhiguo Wang, Haitao Mi, and Abraham Ittycheriah. Sentence similarity learning by lexical decomposition and composition. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1340–1349, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee.

[78] Joseph Weizenbaum. Elizaa computer program for the study of natural language communication between man and machine. *Commun. ACM*, 9(1):3645, January 1966.

[79] Wikipedia contributors. Singular value decomposition — Wikipedia, the free encyclopedia, 2020. [Online; accessed 02-November-2020].

[80] W. A. Woods. Progress in natural language understanding-an application to lunar geology. In *Managing Requirements Knowledge, International Workshop on*, volume 42, pages 441–450, Los Alamitos, CA, USA, jun 1973. IEEE Computer Society.

[81] Zhibiao Wu and Martha Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics*, ACL '94, page 133138, USA, 1994. Association for Computational Linguistics.

[82] A Ydobon. N-gram to quantify similarity between sentences, 2020. [Online; accessed 02-November-2020].