

Hiring through Social Networks: A Natural Language Processing Automation of Conversation



By

Shaida Muhammad

Fall-2019-MS-CS 00000317817 SEECS

Supervisor

Dr. Safdar Abbas Khan

Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters
of Science in Computer Science (MS CS)

In

School of Electrical Engineering & Computer Science (SEECS) ,


National University of Sciences and Technology (NUST),

Islamabad, Pakistan.

(July 2023)

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis entitled "Hiring through social networks: A natural language processing automation of conversation" written by SHAIDA MUHAMMAD, (Registration No 00000317817), of SEECS has been vetted by the undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: _____  _____
SEECS

Name of Advisor: Dr. Safdar Abbas Khan _____

Date: 19-Jun-2023 _____

HoD/Associate Dean: _____

Date: _____


Signature (Dean/Principal): _____

Date: _____

Approval

It is certified that the contents and form of the thesis entitled "Hiring through social networks: A natural language processing automation of conversation" submitted by SHAIDA MUHAMMAD have been found satisfactory for the requirement of the degree

Advisor : Dr. Safdar Abbas Khan

Signature: 

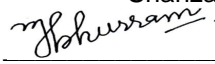
Date: 19-Jun-2023

Committee Member 1: Dr. Muhammad Ali Tahir

Signature: 


Date: 19-Jun-2023

Committee Member 2: Dr. Muhammad Khuram
Shahzad

Signature: 

Date: 19-Jun-2023

Committee Member 3: Dr Adnan Aslam

Signature: 

Date: 20-Jun-2023


Dedication

This thesis is dedicated to all the deserving children who do not have access to quality education especially young girls.

Certificate of Originality

I hereby declare that this submission titled "Hiring through social networks: A natural language processing automation of conversation" is my own work. To the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics, which has been acknowledged. I also verified the originality of contents through plagiarism software.

Student Name: SHAIDA MUHAMMAD

Student Signature:  _____

Acknowledgments

Glory be to Allah (S.W.A), the Creator, the Sustainer of the Universe. Who only has the power to honour whom He please, and to abase whom He please. Verily no one can do anything without His will. From the day, I came to NUST till the day of my departure, He was the only one Who blessed me and opened ways for me, and showed me the path of success. There is nothing which can payback for His bounties throughout my research period to complete it successfully.

Shaida Muhammad

Contents

1	Introduction	1
2	Literature Review	5
2.1	Message Classification	5
2.1.1	Machine Learning based approaches	5
2.1.2	Deep Learning based approaches	10
2.2	Conversational Systems	16
3	Methodology	20
3.1	Message Classification	21
3.1.1	Machine Learning Algorithms	21
3.1.2	Deep Learning Algorithms	28
3.2	Conversational System [RASA Framework]	40
3.2.1	Components	41
3.2.2	Policies	42
3.2.3	Actions	44
3.2.4	Integration by Input/Output Channels	47
3.2.5	RASA Action Server	47
3.2.6	Tracker Store	47
3.2.7	Lock Store	48
3.2.8	File System	48

CONTENTS

3.2.9	RASA X	49
4	Experiments & Results	50
4.1	Message Classification	50
4.1.1	Dataset	50
4.1.2	Experimental Setup	52
4.1.3	Results	53
4.2	Conversational AI System	58
4.2.1	Dataset	58
4.2.2	Experimental Setup	60
4.2.3	Results	61
5	Conclusion and Future Work	64

List of Abbreviations and Symbols

Abbreviations

MSR	Measuring Semantic Relatedness
CDD	Conversation-Driven Development
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
RAM	Random Access Memory
NER	Named-entity Recognition
SVM	Support Vector Machine
LSTM	Long Short Term Memory
SVD	Singular Value Decomposition
KNN	K-Nearest Neighbor
QNLI	Question-answering Natural Language Inference
TF-IDF	Term Frequency-Inverse Document Frequency
NLU	Natural Language Understanding
NLG	Natural Language Generation
SST	Stanford Sentiment Treebank

CONTENTS

CNN	Convolutional Neural Network
DCNN	Dynamic Convolutional Neural Network
RNN	Recurrent Neural Network
BOW	Bag-of-Words
CBOW	Continuous Bag-of-Words
NBoW	Neural Bag-of-Words
DST	Dialogue State Tracker
DSTC	Dialog State Tracking Challenge
SGT	Sequence Graph Transform
DAN	Deep Average Network
NB	Naïve Bayes
HPCA	Hybrid Principal Component Analysis
BLEU	BiLingual Evaluation Understudy
MRPC	Microsoft Research Paraphrase Corpus
RTE	Recognizing Textual Entailment
NNLI	Multi-Genre Natural Language Inference
NNLM	Neural Network Language Model
DT	Decision Tree
CRF	Conditional Random Field
TREC	Text REtrieval Conference
STS	Semantic Textual Similarity

List of Tables

2.1	Semantic-Syntactic Word Relationship test set accuracy	11
2.2	Semantic-Syntactic Accuracy compare to other models	11
2.3	NLU system	17
4.1	Google Colab: Disk space information	52
4.2	Google Colab: CPU information	52
4.3	Google Colab: Memory (RAM) information	52
4.4	Machine Learning Algorithms Results	54
4.5	Feed-forward Neural Network: fixed hyperparameters	54
4.6	Feed-forward Neural Networks: Results	55
4.7	LSTM: fixed hyperparameters	55
4.8	LSTM: Results	56
4.9	Transformed Based Models: Results	57
4.10	Local Setup: CPU Information	60
4.11	Local Setup: Memory Information	61
4.12	Local Setup: Software Requirements	61
4.13	Intent Classification: Results	61
4.14	Stories: Results	62

List of Figures

2.1	BERT vs XLNet vs RoBERTa evaluation on different tasks	16
3.1	Complete System Architecture	20
3.2	ReLU function and ReLU derivative	30
3.3	Sigmoid function and Sigmoid derivative	30
3.4	Standard Network vs Dropout applied Network	31
3.5	Transformer Architecture	32
3.6	Simple Feedforward Network vs Residual Connections Feedforward Network	34
3.7	BERT	36
3.8	BERT	38
3.9	XLNet	40
3.10	RASA overall architecture	41
3.11	Dialogue Transformers Architecture	43
4.1	Google Colab: GPU Information	53
4.2	Intent Classification: Confusion Matrix	62
4.3	Intent Classification: Confusion Matrix	63

Abstract

Linkedin is used by recruiters in the recruitment industry to find candidates for jobs. Recruiters search for candidates, select potential candidates (prospects) and send job advertisement/invitation messages. The messages sent by recruiters to prospects should receive maximum responses/replies from prospects in order to select quality candidates. For maximum responses, the message must contain some quality features like call-to-action, personalization, incentives for connecting, job/company credibility, etc. We used Natural Language Processing techniques to classify the message if it contains quality features. To classify the message, we trained different machine learning and deep learning models. In response to the recruiter's message, the prospect asks some questions about the job, for example the job location, salary, benefits, company information, etc. which the recruiter has to answer manually. In this work, we employed the RASA [68] conversational AI framework to develop natural language understanding and conversation models that allow for automated communication between recruiters and job prospects. This automation frees up the recruiter's time by eliminating the need for them to individually respond to each inquiry about the job.

Keywords: *Conversational AI, Text Classification*

CHAPTER 1

Introduction

The advancement of conversational systems has enabled the development of computer-human communication in a more human-like manner, using languages such as English in both spoken and written form. These systems have evolved significantly over time, starting as rule-based systems and eventually becoming more sophisticated through the incorporation of machine learning techniques. These systems are able to comprehend user queries and provide appropriate responses, and can be broadly classified into two categories: open domain and closed domain.

Open-Domain conversational systems are designed to be more general in nature, allowing users to interact with them as if they were human. These systems are generative, meaning that they generate responses to user queries based on the conversation history and are trained on large datasets of conversations. One notable advancement in open-domain conversational systems is ChatGPT, which has demonstrated promising performance in this area.

ChatGPT is an open-domain conversational system that is capable of conducting a wide range of conversations with users. It has been trained on a large dataset of human-human conversations and is able to generate responses that are natural and coherent in context. Some examples of the types of conversations that ChatGPT can engage in include casual chit-chat, answering questions about various topics, and engaging in more in-depth discussions on specific subjects. ChatGPT is an exciting development in the field of conversational AI and has the potential to greatly improve the way that humans interact with computers.

Closed-Domain, in contrast to open-domain conversational systems, which are de-

signed for general communication, closed-domain systems are specialized for specific tasks and are able to communicate with users within a specific domain. These systems are often task-oriented, meaning that they are designed to perform a specific set of actions or provide information on a specific topic. For example, a closed-domain conversational system might be designed specifically for booking tickets and would not be able to handle inquiries outside of that domain. Because these systems are focused on a specific task, they typically require less data to be effective compared to open-domain systems. Closed-domain conversational systems can be either retrieval-based or machine learning-based. Retrieval-based systems select responses from a pre-defined set of responses based on predetermined rules or policies, while machine learning-based systems use data to train a model to generate responses. Both types of closed-domain systems are able to effectively communicate with users within their defined domain but may struggle to handle inquiries outside of that domain. Closed-domain conversational systems are composed of various components that work together to enable effective communication with users within a specific domain. These systems typically consist of two main components: natural language understanding (NLU) and natural language generation (NLG). The NLU component is responsible for interpreting user queries by predicting the user's intent and extracting relevant structural information. The NLG component then selects an appropriate response from a predefined set of responses and presents it to the user. In some cases, there may be an intermediate step between NLU and NLG in which an API call or database query is made to perform a specific task. The dataset for closed-domain conversational systems typically includes user queries labeled with their intents and structural information (such as named entity recognition tagging) as well as conversations between users and the system (also known as conversation stories).

Conversational systems have a wide range of applications across various domains, including online customer service, entertainment, education, personal assistants, and intelligent quiz systems. In this work, our focus is on the development of an intelligent conversational system for the recruitment domain.

Many companies utilize social and professional networks, such as LinkedIn¹, to hire employees for open positions. LinkedIn, in particular, is a highly popular platform with over 830 million members in more than 200 countries worldwide. When a recruiter posts a job advertisement on LinkedIn, they often receive responses from potential candidates

¹<https://www.linkedin.com/>

expressing interest, disinterest, or no response at all. In an ideal scenario, the recruiter would receive a response from all prospects, enabling them to interview a larger pool of candidates and increase the chances of finding the best employees for the job.

Our goal is to develop a conversational system that can assist recruiters in this process by automating certain aspects of communication with potential candidates.

Many social and professional networks, including LinkedIn, are plagued by spam and bot messages, which can make it difficult for recruiters to effectively reach potential candidates. As a result, prospects may not respond to job advertisements or may view them as spam. One contributing factor to this issue is the low quality of the job advertisement message. In order to maximize the number of responses from prospects, it is important for the recruiter’s message to possess certain key qualities. These quality features are *call to action*, *common ground*, *establish personal or company credibility*, *incentives*, and *personalization* that are defined by the domain experts. To ensure that the job advertisement message will be effective in eliciting a response from prospects, it is necessary to evaluate the quality of the message beforehand. We approached this problem as a multi-label classification task, in which advertisement messages are labeled based on five key qualities that are likely to make them more appealing to prospects. In order to train machine learning models to accurately predict the quality of an advertisement message, we utilized a range of different models, including transformer-based models such as BERT, RoBERTa, and XLNet. By using these models, we aimed to develop a system that could accurately assess the quality of an advertisement message and identify those that are likely to be most effective in getting responses from prospects.

In order to streamline the recruitment process and save time for recruiters, we have developed a machine learning-based conversational system that can automate the communication between recruiters and job prospects. When a recruiter sends a job advertisement message to a large number of people, it can be challenging to keep up with the volume of responses, especially if the message is of high quality and generates a large number of replies. To facilitate this process, our conversational system allows the recruiter to pre-populate certain fields in advance and then handles the conversation until it is completed or hands control back to the recruiter when necessary. We implemented this system using the RASA open source framework, which is a powerful tool for building conversational AI systems. RASA includes a suite of tools and libraries for

natural language understanding, dialogue management, and more, making it an ideal choice for the development of our intelligent conversational system.

The RASA framework is a popular open source toolkit for building conversational AI systems. It includes a range of components and libraries that work together to enable the creation of intelligent chatbots and other conversational systems.

One key component of RASA is the natural language understanding (NLU) module, which is responsible for interpreting and extracting meaning from user inputs. This module includes a number of sub-components that perform tasks such as entity recognition, intent classification, and dialogue state tracking. The NLU module is able to process inputs in a variety of languages and can be customized to understand domain-specific terminology and context.

Another important component of RASA is the dialogue management module, which controls the flow of the conversation and determines how the system should respond to user inputs. This module makes use of machine learning algorithms to predict the next best action for the system to take based on the current context and the user's previous inputs.

RASA also includes a natural language generation (NLG) module, which is responsible for generating appropriate responses for the system to present to the user. The NLG module can be customized to produce responses in a specific style or tone, and can make use of templates or machine learning models to generate more sophisticated responses.

The RASA framework provides a comprehensive set of tools and libraries for building intelligent conversational systems, making it a popular choice for developers in a wide range of domains.

CHAPTER 2

Literature Review

2.1 Message Classification

Message classification is a text classification task where we classify text into 5 different categories, call-to-action, common-ground, establish-credibility, incentive-for-connecting, and personlization.

2.1.1 Machine Learning based approaches

Machine learning based text classification broadly consist of two steps: **Feature Extraction** and **Classification Algorithm**.

Feature Extraction

Natural language is unstructured. It should be converted to a numerical structured representation so that it can be fed to machine learning algorithm. This numerical representation is known as feature vector or word vector. Different techniques are used for generating feature vectors some of them are One-hot vector, Bag of Words (BOW), TF-IDF etc.

One-hot encoding is a simple and intuitive approach for generating feature vectors from natural language data. In this approach, a dictionary of size V is created for the text dataset, with each unique word in the dataset being assigned a corresponding index number. Feature vectors of size V are then generated for all words in the vocabulary, with each one-hot vector containing all zeros except for a single 1 at the position corresponding

to the index of the word in the dictionary. One major drawback of one-hot encoding is that the resulting feature vectors do not contain any information about the context or meaning of the words. Additionally, the distance between words in this encoding cannot be calculated, making it difficult to infer relationships between words. Another issue with one-hot encoding is that the size of the feature vectors is directly proportional to the size of the vocabulary, which can make this approach memory inefficient when working with large datasets. Finally, the resulting one-hot vectors are often very sparse, which can further contribute to the inefficiency of this approach.

The sparsity problem was solved with a new technique known as word co-occurrence matrix. This technique is also known as **Bag of Word (BoW)**. The word co-occurrence matrix is simply a $V \times V$ size matrix that contains word vectors. The word vectors are obtained by looking at the words around the desired words and counting the appearances of other words and putting that value in the specified cell. A context window is defined for the neighbourhood. The bigger the context window size, the more semantics the feature vectors get. These vectors have the same size as one-hot encoding but their size can be reduced via dimensionality reduction algorithms like Principal Component Analysis (PCA), Singular Value Decomposition (SVD), Linear Discriminant Analysis (LDA), and t-Distributed Stochastic Neighbor Embedding (t-SNE). Co-occurrence matrix takes local information around a word into account but doesn't account for global information on whole text documents.

Another technique known as **Term Frequency-Inverse Document Frequency (TF-IDF)** [5] was introduced, which takes care of the global context as well as the local context. TF-IDF performs this in two steps: Term frequency (TF) [2] and Inverse Document Frequency (IDF) [4]. TF is simply the count of how many documents contain the word $tf_{t,d} = \text{count}(t, d)$. It gives us a matrix where columns represent documents and rows represent the words. The formula is modified a bit for normalization and hence we use $tf_{t,d} = \log_{10}(\text{count}(t, d) + 1)$. The IDF term can be calculated by $idf_t = \log_{10}(N/df_t)$ where N is the total number of documents and df_t is the document frequency of term t . The IDF term is used to focus on less-occurring words that can be meaningful. Finally, TF and IDF scores are multiplied to get the final word vector matrix: $w_{t,d} = tf_{t,d} * idf_t$. Other deep learning based approaches like Word2Vec [36], GloVe [48], Doc2Vec [47] etc. are used for feature extraction. They are discussed in detail in section 2.1.2

Classification Algorithm

A classification algorithm takes labeled data as input and train a model on it. Training data is represented by $X = \{x_1, x_2, x_3, \dots, x_m\}$ and $y = \{y_1, y_2, y_3, \dots, y_m\}$ where X is set of training examples and y is the corresponding set of labels. x_i is the i th training example which is an n -dimensional vector and y_i is the label of i th training example. The labels belongs to predefined k classes $\{1, 2, 3, \dots, k\}$. The machine learning algorithm tries approximate a function $f(x)$ which tries to label any new unseen data.

In text classification, training data is set of documents $X = \{d_1, d_2, d_3, \dots, d_m\}$ where each document d consist of l words i.e. $d = \{w_1, w_2, w_3, \dots, w_l\}$. The word vectors are obtained by applying a feature extraction algorithm. The word vectors are summed or averages to obtain the document vector. Document level vectors can also be obtained by applying document level feature extraction algorithm like Doc2vec. After generating feature vectors from text document, a classification algorithm is applied to feature vectors to predict the class of text document.

Logistic Regression (LR) is one of the simplest classification algorithm. Logistic Regression is the modified version of Linear Regression. Linear Regression hypothesize that class predictions are the weighted sum of the components of feature vector: $\hat{y}^{(i)} = \theta_0 + \theta_1 x_1^{(i)} + \dots + \theta_l x_l^{(i)}$. LR uses logistic function $\text{logistic}(\eta) = \frac{1}{1 + \exp(-\eta)}$ which is a function of the output of Linear Regression: $P(y^{(i)} = 1) = \frac{1}{1 + \exp(-(\theta_0 + \theta_1 x_1^{(i)} + \dots + \theta_l x_l^{(i)}))}$. LR gives the probability score between 0 and 1 for example x_i . LR is trained by minimizing the negative log likelihood (NLL) 2.1.1 loss function over training examples m :

$$J(\theta) = \frac{1}{m} \left[\sum_{i=1}^m -y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log((1 - h_{\theta}(x^{(i)}))) \right] \quad (2.1.1)$$

Logistic Regression model can be efficiently trained. It is also fast at inference. But It also has some limitations. LR model often overfit when the number of features is greater than the number of examples in the dataset. It draw linear boundaries in the feature space but real world data is highly non-linear and requires non-linear models. Despite of simple nature, LR is used in many text classification tasks including text categorization [62], tweets classification [18], toxic comments classification [76], text-based emotion classification [81], intent classification in chatbot [77], sentiment analysis [70] etc.

Naïve Bayes (NB) is another ML classifier used for classification tasks. NB uses Bayes Theorem [1] which was formulated by Thomas Bayes between 1701–1761 which

is described as equation 2.1.2.

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)} \quad (2.1.2)$$

Naïve Bayes makes an assumption that features are independent from each other which makes it simpler and efficient to train. Given a document d the NB classify the document into set of classes $C = \{1, 2, 3, \dots, c\}$ and return the predicted class \hat{c} . Naïve Bayes can be written as equation 2.1.3 for document classification.

$$\hat{c} = \arg \max_{c \in C} P(d/c)P(c) \quad (2.1.3)$$

Naive Bayes algorithm is used in many areas of text classification which includes spam email classification [19], document classification [50] etc.

One of the simplest and well know multi-class non-parametric classification algorithm is **K-Nearest Neighbour** (KNN). KNN 2.1.4 makes an assumption that an unseen example x can be classified by looking at the feature space of training data. The class labels of k nearest data points in training data will decide the class label of x . The distance between x and training example d can be measure via dot product $Dot(x, d) = \sum_i^n x_i d_i$, cosine similarity $Cosine(x, d) = \frac{x \cdot y}{|x||d|}$, Euclidean distance $\sqrt{\sum_{i=1}^n (x_i - d_i)^2}$, or any other vectors similarity metric.

$$\begin{aligned} f(x) &= \arg \max_j Score(x, C_j) \\ &= \sum_{d_i \in KNN} sim(x, d_i) y(d_i, C_j) \end{aligned} \quad (2.1.4)$$

$f(x)$ is used to predict the class label of new example. KNN requires no training time as it is a lazy learning algorithm. KNN doesn't learn any mapping function during training. The training data is stored and computations are performed at inference time only. New labeled data points can be added anytime to the dataset to improve accuracy. KNN also comes with some limitations. The inference time of KNN is directly proportional to the size of dataset i.e. number of data points and number of dimensions of data points. If the size of dataset it huge, then it would take a lot of time and computation resource for inference which makes it unfeasible to use. Feature vectors should be scaled to produce more accurate results. KNN is sensitive to noisy data so training data must be cleaned before use. Also, it is very difficult to select the value of k . KNN is used for text categorization [51] along with TF-IDF, BBC news text classification [89] where it gets 92% accuracy. KNN is also used for short text classification [31] (tweets,

comments, news headlines etc.) with Pearson Correlation distance where KNN gives better accuracy than NB and SVM.

Support Vector Machine [3] is a non-linear binary classification algorithm that works well when data is high dimensional. SVM does classification by finding a hyperplane in n-dimensional feature space. There could be many hyperplanes between two classes on feature space. Even though all hyperplanes can classify training data but SVM chooses the one that is best suited for unseen data/ SVM chooses hyperplane which has maximum marginal distance. Data points on different sides of the hyperplane are classified into their respective class i.e. 1 or 0. The key point in SVM is support vectors. Support vectors are used to pick the right hyperplane by maximizing the marginal distance. SVM uses hinge loss function $hinge_{loss}(y, f(x)) = \max(0, 1 - y \cdot f(x))$ to compute the cost and optimize the model weights. SVM was tried out on news classification [17], web text classification [73], Sentiment Analysis [67], emotional classification on news headlines [27], fake news detection [88] etc. SVM can also be applied to non-linear data by using a kernel function. Kernel function takes n-dimensional non-linearly separable data and convert it to m-dimensional linearly separable data.

Decision Tree (DT) [7] is another well known non-parametric classification algorithm. DT learns understandable decision rules from features of training data by recursively splitting the features. Unlike SVM, KNN, and LR, DT also works on categorical features. DT start constructing the tree by selecting a feature as root node and then drawing other nodes from it. DT can be used for multi-label/multi-output problems. DT doesn't require scaling of features and fast to compute. DT can overfit easily. Overfitting can be avoided by using pruning or selecting tree depth in advance. DT is also very sensitive to noisy data. The accuracy of DT can be improved by an ensemble technique **Random Forest** (RF) [6]. RF generates multiple t random trees in parallel. Each tree predict a label for x and then final label \hat{y} is decided on votes. The number of trees influence the accuracy of model. Also, the number of trees is directly proportional to the inference time. If the number of trees increase, the inference time increases as well. So the number of tree can also be chosen wisely. DTs are applied on text data for classification e.g. product-review classification [69], arabic text classification [32], sentiment classification of Roman-Urdu [61], restaurant review [80] etc.

2.1.2 Deep Learning based approaches

Machine Learning based text classification approaches were considered quite successful before the arrival of Neural Networks. Feed-forward neural networks were used as feature extractors to get the numerical representation of the text. Each word in the text gets a vector representation with these models.

Word2vec was introduced in [36] where two approaches i.e. CBOW (continues bag of words) and Skip-gram were taken into account to calculate the word vectors or word embedding. Word2vec is a two layer Feed-forward neural network where input is one-hot encoding of vocabulary of size V . In CBOW, given a context windows who's size is pre-defined and a word inside the context window is predicted. A context window of size two means that two past and two future words i.e. $[W_{t-2}, W_{t-1}, W_{t+1}, W_{t+2}]$ are given as input to the model and the model has to predict W_t the center word. The input to the model is the average of one-hot vectors of context words. This Feed-forward network is trained to maximize the likelihood of the center word, given the context window C as shown in equation 2.1.5:

$$\arg \max_{\theta} P(W_{center}/C; \theta) \quad (2.1.5)$$

In Skip-gram, given the center word W_t and the model has to predict the context words C . In case of context window of size two $C = 2$, for W_t , the model will predict $[W_{t-2}, W_{t-1}, W_{t+1}, W_{t+2}]$. Softmax function is used on final layer to convert raw scores into probabilities and then a simple loss function of $y_{pred} - y_{true}$ is used to compute the loss for each context word. The losses are then added to computer overall loss. The skip-gram network is trained to maximize the likelihood of context words given the center word as shown in 2.1.6:

$$\arg \max_{\theta} P(C/W_{center}; \theta) \quad (2.1.6)$$

As a two layer neural network, these models has two weight matrices, W_1 and W_2 . These two weight matrices are concatenated $[W_1 W_2]$ and used as word-embeddings. These models generate word embeddings in a way that semantically closer words are closer to each other in n-dimensional space. The authors of [36] trained word vectors on Google News corpus which contains about 6B tokens and evaluate the model on Semantic-Syntactic Word Relationship test set. Multiple word vector models were trained with different 50, 100, 300, and 600 dimensional vectors and different size dataset. The authors [36] found out that accuracy is directly proportional to the dimensions of word

Dimensionality / Training words	24M	49M	98M	196M	391M	783M
50	13.4	15.7	18.6	19.1	22.5	23.2
100	19.4	23.1	27.8	28.7	33.4	32.2
300	23.2	29.2	35.3	38.6	43.7	45.9
600	24.0	30.1	36.5	40.8	46.6	50.4

Table 2.1: Semantic-Syntactic Word Relationship test set accuracy

Model Architecture	Semantic-Syntactic Word Relationship test set		MSR Word Relatedness Test Set [20]
	Semantic Accuracy [%]	Syntactic Accuracy [%]	
RNNLM	9	36	35
NNLM	23	53	47
CBOW	24	64	61
Skip-gram	55	59	56

Table 2.2: Semantic-Syntactic Accuracy compare to other models

vectors and dataset size. The high dimensional vectors trained on higher dataset size tends to have greater results: see Table 2.1 The accuracy of Word2vec models are better than previous word-embedding models trained on the same dataset with 640 dimensions: see Table 2.2. Word2vec models take local statistics into account to train word-embeddings.

The GloVe model, introduced in the seminal work [48], incorporates both global and local statistics to generate word embeddings, resulting in superior performance compared to other widely-used models. In particular, when evaluated on the word analogy task [36] and word similarity task [34] using various datasets and word embedding sizes, GloVe consistently outperforms models such as *SG*, *CBow*, *SVD* variations, and *LBL* variations. GloVe’s performance has been extensively compared to previously trained models on tasks such as named entity recognition (NER). When pitted against models like *Discrete*, *SVD* variations, *HPCA*, *HSMN*, *CW*, and *CBow*, GloVe consistently emerges as the top performer. These comparisons were made against well-known models proposed in [26] and [20], demonstrating GloVe’s superiority in capturing semantic information and its applicability to various natural language processing tasks. By leveraging both global and local statistics, the GloVe model offers a comprehensive approach to generating word

embeddings that excel in capturing subtle relationships between words. Its impressive performance on tasks such as word analogy and word similarity highlights its ability to effectively represent word meanings and contextual nuances. GloVe’s superiority over previously trained models on the challenging NER task further solidifies its position as a leading word embedding model in the field of natural language processing. The GloVe model was trained on huge corpus including Common Crawl, Wikipedia dump, and Gigaword5 with a total of 42B tokens and 400,000 vocabulary size. In GloVe, co-occurrence matrix X is created with a context window of size 10. GloVe gives two set of word-embeddings W and \tilde{W} that is added to get the final word-vectors.

These pre-trained word-embeddings are used as input to different models. The Deep Average Network (DAN) [55] uses the 300-d of GloVe pre-trained word vectors. The DAN does sentiment analysis by taking text document as input, tokenize it, and then average the embeddings of all tokens in the text. Feed the average embeddings to a simple Feed-forward neural network with multiple hidden layers and perform classification on final layer. DAN takes less time per epoch during training and perform comparable accuracy with other sophisticated models including Neural Bag-of-words (NBOW), RecRNN, TreeLSTM etc. that takes the syntactic information of text into account which requires more training time.

The Doc2vec [47] word-embeddings framework was proposed which learns embeddings on sentence/document level instead of word level. The architecture of Doc2vec is almost the same as Word2vec where given a context and the model predicts the center word. The Doc2vec takes complete document/paragraph as input and generate sentence level embeddings. Two variations of Doc2vec were trained, Distributed Memory (PV-DM) and Distributed Bag-Of-Words (PV-DBOW). PV-DM model is somehow the extension of CBOW. PV-DM takes some random words from document, treat it as context and tries to learn the center word of those random words. The context words vectors are averaged/concatenated and given as input to the model along with paragraph/document ID vector which in unique ID for each document in training set. The PV-DBOW model is somehow similar to Skip-gram model. The PV-DBOW model takes the document id as input and tries to produce the randomly selected words from paragraph/document. The trained embeddings were then applied to sentiment analysis classification and information retrieval tasks. Classification was performed on two datasets: Stanford sentiment treebank [38] and IMDB dataset [22] where it outperformed all other state-of-the-art

approaches on test set.

Recurrent Neural Networks (RNNs) based models along with Long Short-Term Memory (LSTM) were used which takes sequence information and word dependency into account for classifying sequences of text. Vanilla RNNs are effected from vanishing gradient problem. LSTM take care of vanishing gradient problem by introducing memory cell. LSTM also has three gates: input, output, and forget gate which regulate the flow of information from and into the cells of LSTM. LSTM model is good at representing sequential information but it is not good at keeping the syntactic and semantic information. Tree-LSTM [58] was introduced to keep the syntactic information of natural language intact while performing classification. Tree-LSTM was tested out on Stanford Sentiment Treebank [38] dataset for sentiment classification where it performed better than different variants of RNNs, LSTMs, and other different models on fine-grained classification task and comparable accuracy on binary-classification.

Even though LSTM has memory cells and tries to carry information from initial input tokens to final timestamps but often perform poor on very long texts. Multi-Timescale LSTM (MT-LSTM) [56] was introduced to capture very long term dependencies with the help of grouping the hidden states of LSTM with 3-times faster training. MT-LSTM outperform other RNN, CNN, and LSTM based models on Stanford Sentiment Treebank (SST) [38] fine-grained task, TREC [9], and IMDB [22] dataset. Also, get comparable results on SST binary classification task.

Recurrent Nets are good at sequences where the class of the sequence depends on the complete sequence i.e. the class depends on global semantics. The RNNs carries information from time. The CNNs carries information from space. CNNs are good where the class of the sequence depends on local semantics of the sequence. Thus CNNs were also used for text classification. Dynamic Convolutional Neural Network (DCNN) [45] takes the text input embeddings as matrix, performs wide convolution on input matrix and then use dynamic k-max pooling to convert variable length input to fix length feature map for further processing by next convolutional layer. Finally, a fully connected layer is used as classification head. DCNN is tested on SST, TREC and Twitter sentiment dataset where it performed better than baseline models including Naive Bayes, SVM, NBOW, etc.

A simple CNN [46] was used for text classification with different word-embeddings ap-

proaches: random, static (word2vec), non-static (word2vec with training) and multi-channel (two word2vec vectors). The architecture is pretty simple, convolutional layer followed by max-over-time pooling and then a fully connected head for classification with softmax function to calculate probabilities for each class. The model achieve better accuracy than other models on MR [14], SST-2, CR [11], and MPQA [16] dataset. The model also achieve comparable accuracy on SST-1, Subj [12], and TREC [9] dataset.

Character-level CNN [60] is another variant of CNN where the input to the model is one-hot encoding of characters in the sequence. The input length is fixed at 1014 characters. If sequence length is more than 1014 then the first 1014 character would be taken and the remaining will be truncated. If sequence length is less than 1014, then zero vectors are added to the input sequence to make it 1014. Character-level RNN has 6 convolutional layers along with pooling layers and 3 fully connected layers along with dropout layers are stacked to perform text classification. Two networks of the same architecture but with different size of feature maps and different number of hidden nodes are trained on eight different datasets. These models empirically outperform previous machine learning based models, LSTM and word-level CNNs.

Although CNNs perform well by stacking convolutional, pooling and fully connected layers together but sometimes due to pooling operations, CNN lost important spatial information which causes the model to mis-classify text. The information lost in pooling layer issue was addressed by a new type of neural network called Capsule Neural Network (CapsNets) [71]. An activity vector of a capsule represent the attribute of an object. The length of activity vector describes if the object is present and the direction specifies the features of that object. Capsules take the information from different layers till last layer so that no information is lost. CapsNets for text classification were used by [85, 79] where the model performed better than LSTM and CNN based models on MR, Subj, CR and AG dataset.

The RNNs, LSTMs and GRUs were designed to capture long-term dependency in text sequence but it often failed when sequence goes too long. Attention mechanism [44] was introduced to overcome the long-term dependency problem of RecurrentNets. Attention mechanism let the model focus on important areas in the input sequence. The attention mechanism was originally designed for sequence to sequence (seq2seq) problems where the encoder part tries to capture the information of input in a vector called context

vector. The decoder part takes the context vector and tries to produce the output sequence. Attention mechanism was used in Hierarchical Attention Network [65] for text classification. The focus of this work was to classify long sequence of text which consist of multiple sentences. The network consist of a word-based and a sequence-based encoder. The network take the words in the sequence as input, pass it through Bi-directional GRU encoder which acts as word-based encoder and generates attention vectors. The sequence is broken down into sentences and each sentence in the sequence is given to another Bi-direction GRU based encoder along with the attention scores from the previous word-based encoder. The sequence-based encoder has its own attention vectors which helps the model focus on relevant sentences in the sequence. The model perform better on BOW, SVM, CNN, and LSTM based models on Yelp review, IMDB reviews, Yahoo answers and Amazon reviews dataset.

The RecurrentNets including LSTM and GRU takes input sequence word by word. The next word is dependent on the information from the previous word. This makes RNNs unsuitable for parallel computation which causes inefficient utilization of the modern GPU. Transformer [72] architecture was introduced to address the parallelism issue. Transformer architecture takes the complete sequence for computation and uses positional embedding mechanism to remember the position of words in the sequence. Transformer introduced new type of attention mechanism known as self-attention. Self-attention tries to focus on the dependency and relevance of words in the input sequence with each other. Transformer is encoder-decoder model and was trained for WMT-14¹ machine translation task and it outperformed all previous state-of-the-art models with a better margin of BLEU score on WMT-14 task. The encoder part of Transformer can be used for text classification task.

BERT [74] i.e. a transformers based model set new state-of-the-art performance on different tasks including Question/Answering, Named Entity Recognition (NER) and text classification task. Two variations of BERT was trained: **BERT_{BASE}** and **BERT_{LARGE}**. **BERT_{BASE}** consist of 110M parameter with 12 layers with a hidden size of 768 and 12 self-attention heads. **BERT_{LARGE}** is relatively bigger model which has 24 layers with hidden size of 1024 and 16 self-attention heads. BERT models are trained in two phases. (1) pre-training (2) fine-tuning. Pre-training step performs a masked-language modeling task on unsupervised data. Fine-tuning step takes the weights of language

¹<https://www.statmt.org/wmt14/>

model and train it for downstream tasks. BERT perform better than all other previous state-of-the-art models including OpenAI GPT-1 [75]. XLNet [86], another transformer based model with the same architecture as BERT but trained on 113GB of data that helps the model perform better than BERT on all tasks but with the expense of 5-times more training time than BERT and hardware resources. Another extension to BERT known as RoBERTa [83] was trained. RoBERTa perform better than both BERT and XLNet on different tasks. RoBERTa was trained on 160GB of data that is bigger than both BERT and XLNet. See figure 2.1

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	90.7	83.5	95.2	92.6	68.6	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5

Figure 2.1: BERT vs XLNet vs RoBERTa evaluation on different tasks

These models can be fine-tuned on custom datasets. BERT, RoBERTa, and DistilBERT were fine-tuned for text classification on legal domain dataset [82] where they performed better than attention-based LSTM and GRU baseline models [82]. RoBERTa and DistilBERT dominated BERT model on different accuracy metrics.

In our work, we will be fine-tuning BERT, RoBERTa, XLNet and DistilBERT for message classification on our dataset and will produce baseline results on message classification dataset.

2.2 Conversational Systems

There are two majorly two types conversational systems: Task-oriented and Non-task oriented.

Non-task oriented conversational systems are often called chatbots. They are open domain systems where user can ask any query. Two approaches are taken into account.

(1) A generative based network [24, 63, 78] where the diverse utterances are generated.

	Book	a	flight	to	London	for	16-December-2022	at	9:00AM
Slots/Entities	O	O	O	O	GPE	O	DATE	O	TIME
Intent	order_flight								

Table 2.3: NLU system

Previous dialogues are given as context. (2) A retrieval based system [39, 33] where the utterances are selected from predefined set of utterances.

Task-oriented conversational systems are domain specific conversational systems. These systems are end-to-end systems and are made-up of a few components. (1) **Natural Language Understanding** (NLU) parse the query received from the user into some semantic pieces. (2) **Dialogue State Tracker** (DST) keeps track of the current state of dialogue along with the history states. (3) **Dialogue policy learning** decides the next action based on current dialogue state. (4) **Natural Language Generation** (NLG) maps the selected action to its surface and generates the response.

NLU in conversational systems is performed via two tasks: intent detection, and entity extraction. **Intent detection** [25, 28, 52, 30, 49] is text classification task where the user query/utterance is classified into pre-defined classes. The intent of user query inform the system that what the user is asking about i.e. what is the intention of the user. **Entity extraction** or **Slot filling** is named entity recognition task where the semantic information from user query are collected into structured format. Entities are also pre-defined like intents. NER is treated a sequence labeling problem where each word in the user utterance is assigned a tag. Conditional Random Fields (CRF) [90] models are used to set baselines for Entity extracting. However, Deep Belief Networks (DBNs) [25] tends to perform better than CRF based baseline models. RecurrentNets [35, 53, 42] were also used for NER tasks. A user query which says *Book a flight to London on 16-December-2022 at 9:00AM* gives an intention of booking a flight which means that user is asking for a pre-defined intent *book_flight*. The Entity extractor will extract *London* as location from the text. Table 2.3 shows how the user query is interpreted by NLU system. Geopolitical entity *London*, Time entity *9:00AM*, Date entity *16-December-2022* are extracted with the intent *order_flight* that helps the NLU system understand the what the user actually wants. The outputs are given to DST for further processing.

Dialogue State Tracking (DST) component keep track of the dialogue state H_t at time t . The user goal might change from the initial settled goal to a new one or the initial goal might be slightly modified due to some reasons caused by the system or user himself. For example, when the system receive the above user utterance, the state of the system is to order book flight to specific location on specific date and time. But if no tickets are available and the system replies that *"sorry there are not tickets available for 9:00AM, would you like to book for 10:00AM"* then depend on the user next utterance the state should be changed. If the user says *"yes"*, then the system state would be changed accordingly and ticket booking should pursue further. If the user says *"no, cancel it"* then the system state would change accordingly and the ticket book should be stopped. These states were managed through rule based systems where the user has to follow specific patterns in querying the system. E-form [8] algorithm was introduced where the user doesn't have to follow specific pattern. The Dialog State Tracking Challenge (DSTC) [41] was introduced to evaluate state tracking systems. A rule-based system [40] that used some domain independent rules with probability and was tested out on DSTC where it gets comparable accuracy with other machine learning based models. CRF was used [37] where the model outperformed 1-best tracking approach. A Feed-forward network [29] with 3 hidden layers was used with a sliding window to output some sequence of probability distributions for state tracking where the model achieve comparable performance. RecurrentNets based model [57] was trained on dataset from multiple domains. Regardless of the data available for a specific domain, the model get state-of-the-art results.

Dialogue Policy decides next action based on the output of DST. Dialogue policies are learned using rule-based system, supervised learning or reinforcement learning [43, 54] methods are used. In the case of *book_flight* example, if the ticket flight is available at the user required time, the system would perform an action of booking a flight ticket for the user. These actions are in some form of abstract symbolic representation.

These actions are converted to natural language utterances through **Natural Language Generation** (NLG) component. A natural language utterance must convey the desired meaning [15]. The utterance must be syntactically correct. The utterance should have no ambiguity in the context. The actions are converted to intermediate representation and then utterance is generated through surface realization [13, 10]. A conditional LSTM was trained [59] for NLG where the slot values are used as input in

the form of one-hot encoding to the model and the model generates natural language utterance. LSTM based generated utterances have more variety and are more natural. The LSTM based approach achieved better BLEU score than rule-based and other algorithms. Context Aware CA-LSTM [66] encoder-decoder sequence-to-sequence model was used to get state-of-the-art results. CA-LSTM uses attention mechanism to better capture the dependency. CA-LSTM gets better on human and quantitative evaluation. An end-to-end system [64] was proposed where the model takes dialogue history as input and product new utterance based on that. An LSTM-based intent network generates a distributed representation and belief trackers handles slot-value pairs. Database operator generates database query from belief states. A system action vector is generated from belief state and intent representation by policy network. Finally, an attention based LSTM network is used to generate the natural language utterance conditioned on action vector.

Methodology

The system architecture is designed to incorporate message classification and a conversational agent as its key components, as depicted in Figure 3.1. This holistic approach ensures a comprehensive and integrated system that effectively handles the entire conversational process. The message classification component is responsible for accurately categorizing incoming messages, enabling the system to understand the intent and context of user inputs. Meanwhile, the conversational agent component utilizes advanced techniques to generate meaningful and contextually appropriate responses. By combining these components, the end-to-end system architecture creates a seamless and intelligent conversational experience for users.

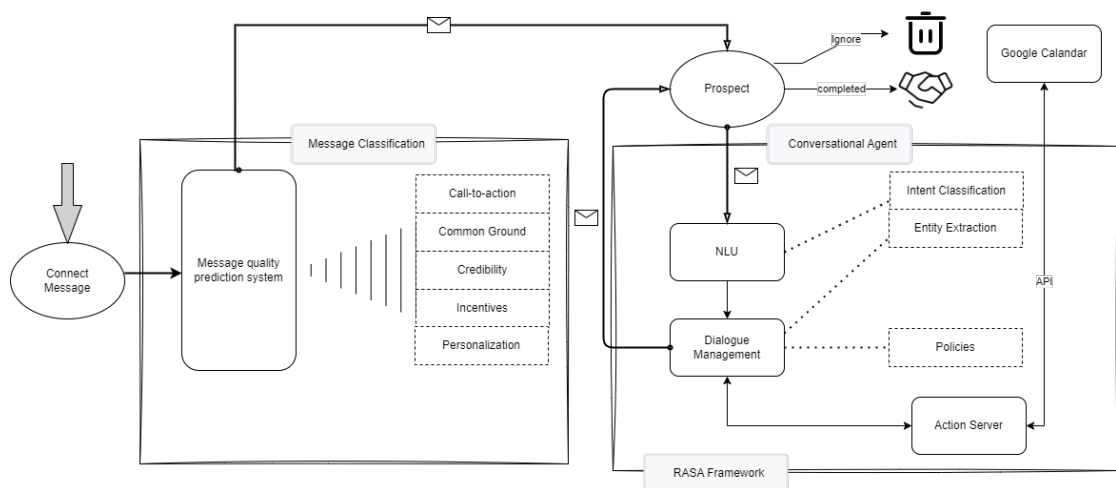


Figure 3.1: Complete System Architecture

3.1 Message Classification

Message classification components shows the strength of message sent from the recruiter to the prospect. Message classification is multi-output classification problem where a text sequence x is assigned a zero or more classes from a set of 5 classes $Y = \{call - to - action, common - ground, establish - credibility, incentive - for - connecting, personlization\}$. A message having all these five classes available is considered high quality message. In this work, we have tried out different Machine Learning and Deep Learning based approaches on message classification task.

3.1.1 Machine Learning Algorithms

We have trained 6 famous machine learning algorithms on our dataset: Decision Tree, Linear SVM, Logistic Regression, Gaussian Naive Bayes, K-Nearest Neighbor, and Random Forest. We have used 300 and 100 dimension GloVe word embeddings. We calculated sentence embeddings as average/mean of word vectors in the sentence.

Decision Tree

The Decision Tree algorithm stands as a stalwart of supervised learning, adeptly addressing both classification and regression tasks. It operates by constructing a tree-like model, leveraging the provided data features to guide its decision-making process. By recursively partitioning the data based on feature conditions, this algorithm navigates the intricacies of the problem space, yielding insightful outcomes for prediction and pattern recognition. With its versatility and intuitive structure, the Decision Tree algorithm stands as a reliable tool in the repertoire of machine learning, empowering practitioners to extract meaningful insights from complex datasets. The goal is to create a model that can make predictions based on the features of a given data point by following the path through the decision tree. The algorithm starts at the root node of the tree and evaluates the available features to determine which branch to follow. It continues this process until it reaches a leaf node, which is where the prediction is made. The tree is constructed by splitting the data into subsets based on the most significant attributes/features. The splitting is done in a way such that the purity of the subset increases with each split. The measure of purity is typically the entropy or the Gini index.

Entropy is a measure of impurity, or the amount of randomness in the data. If the data is completely pure, the entropy is zero. If the data is equally distributed across all classes, the entropy is at its maximum value. The Gini index is a measure of how often a randomly chosen element in the set would be incorrectly labeled, assuming that the labels are randomly distributed among the elements. The Gini index ranges from 0 (perfectly pure) to 1 (not pure at all). The splitting of data continues until we reach a leaf node or a node where the entropy or the Gini index is zero or until a pre-defined stopping condition is reached.

The decision tree algorithm has several advantages:

- It is easy to understand and interpret, as the tree structure is visually appealing and the decision tree itself is easy to understand.
- It is a white box type of algorithm, meaning that the decision tree model can be easily explained to humans.
- It can handle high dimensional data and large datasets, as it is not necessary to normalize the data or to transform it in any way.
- It can be used to identify the most significant features in a dataset.

However, decision trees also have some disadvantages:

- They can be prone to overfitting, especially if the tree is allowed to grow too deep.
- They can be unstable, meaning that small changes in the data can result in a completely different tree being generated.
- They can be biased if the data is not evenly distributed among the classes.

Linear SVM

Linear Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification and regression tasks. The goal of a linear SVM is to find the hyperplane in an N-dimensional space (where N is the number of features) that maximally separates the classes. The algorithm starts by finding the hyperplane that has the greatest distance (also known as the margin) between the nearest points of the two classes. This distance is known as the margin, and the hyperplane is known as the

maximum-margin hyperplane. The optimization problem tries to find the values of w and b that minimize the Euclidean norm of the weight vector while ensuring that the points are correctly classified (i.e., the distance between the hyperplane and the nearest points is greater than or equal to 1).

To solve this optimization problem, the algorithm uses the "hinge loss" function, which measures the distance between the predicted label and the true label. The hinge loss is defined as:

$$L(y, f(x)) = \max(0, 1 - yf(x))$$

Where y is the true label and $f(x)$ is the predicted label. The optimization problem is then solved using an optimization algorithm such as gradient descent or the primal-dual method.

Linear SVMs have several advantages:

- They are efficient and scale well to large datasets.
- They are effective in high-dimensional spaces, and they are particularly good at classifying datasets with many more negative than positive examples.
- They are easy to interpret, as the weight vector and bias term can be used to explain the decision boundary.

However, linear SVMs also have some disadvantages:

- They can be sensitive to the scaling of the features, and it is recommended to scale the features before training the model.
- They are not suitable for data that is not linearly separable, as the maximum-margin hyperplane will not be able to accurately classify the data points.
- They do not provide probability estimates, which can be useful in some applications.

Logistic Regression

Logistic Regression is a supervised learning algorithm that can be used for classification tasks. It is a linear model that is used to predict a binary outcome (i.e., a outcome

that can only have two possible values, such as 0 or 1). The goal of logistic regression is to find the parameters (also known as weights or coefficients) of the linear model that best predict the probability of a given data point belonging to a particular class. The probability that a data point belongs to a particular class is calculated using the logistic function, which is defined as equation 3.1.1:

$$f(x) = \frac{1}{1 + e^{-(w^T x + b)}} \quad (3.1.1)$$

Where p is the probability of the data point belonging to the positive class, w is the weight vector, x is the feature vector of the data point, and b is the bias term. The weight vector and bias term are learned during the training process by minimizing the error between the predicted probabilities and the true labels of the data points. The error is typically measured using the cross-entropy loss function, which is defined as equation 3.1.2:

$$L = -(y \log(p) + (1 - y) \log(1 - p)) \quad (3.1.2)$$

Where y is the true label (either 0 or 1) and p is the predicted probability. The cross-entropy loss function is used because it is a continuous, smooth function that is easy to optimize using an optimization algorithm such as gradient descent. The logistic regression algorithm has several advantages:

- It is a simple and fast algorithm that is easy to implement.
- It is highly interpretable, as the weights and bias term can be used to explain the decision boundary.
- It provides probability estimates, which can be useful in some applications.

However, logistic regression also has some disadvantages:

- It is a linear model, so it cannot capture non-linear relationships between the features and the target.
- It can be sensitive to the scaling of the features, and it is recommended to scale the features before training the model.
- It can be prone to overfitting if there are a large number of features relative to the number of data points.

Gaussian Naive Bayes

Gaussian Naive Bayes is a classification algorithm that uses Bayes' theorem to make predictions. It is called "naive" because it assumes that all the features are independent of one another, which is rarely the case in real-world data. Despite this assumption, the algorithm often performs well in practice. The goal of the Gaussian Naive Bayes algorithm is to predict the probability of a data point belonging to a particular class, given its features. This is done using Bayes' theorem 3.1.3, which states that:

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \quad (3.1.3)$$

Where $P(y|x)$ is the posterior probability of the class (y) given the features (x), $P(x|y)$ is the likelihood of the features given the class, $P(y)$ is the prior probability of the class, and $P(x)$ is the prior probability of the features. In the case of the Gaussian Naive Bayes algorithm, the likelihood of the features is assumed to be a Gaussian distribution. This means that the probability of a given feature is calculated using the mean and standard deviation of the feature in the training data. The algorithm then makes predictions by calculating the posterior probability for each class and choosing the class with the highest probability.

The Gaussian Naive Bayes algorithm has several advantages:

- It is easy to implement and run, as it only requires a small amount of training data to make predictions.
- It is fast and efficient, as it only requires a small amount of computation to make predictions.
- It can handle a large number of features, as it assumes that all the features are independent of one another.

However, the Gaussian Naive Bayes algorithm also has some disadvantages:

- It assumes that the features are independent, which is rarely the case in real-world data.
- It is sensitive to the scaling of the features, and it is recommended to scale the features before training the model.

- It can perform poorly on datasets with categorical features, as the algorithm assumes that all features are continuous.

K-Nearest Neighbor

K-Nearest Neighbors (KNN) is a classification algorithm that is based on the idea of identifying the K data points in a training set that are closest to a given data point and using those data points to make a prediction. It is a type of instance-based learning, as it does not learn a model from the training data but simply stores the training data and makes predictions based on the stored data.

The algorithm works as follows:

1. The training data is stored in memory.
2. When a prediction is needed for a new data point, the distance between the new data point and all the stored data points is calculated using a distance metric such as Euclidean distance.
3. The K data points that are closest to the new data point are identified.
4. The prediction is made based on the majority class of the K nearest neighbors.

The value of K is a hyperparameter that is chosen by the user and controls the number of neighbors that are used to make the prediction. A larger value of K will result in a smoother decision boundary, as more data points will be considered when making the prediction. However, a larger value of K can also result in a slower prediction time, as more distance calculations are needed.

KNN has several advantages:

- It is simple to implement and understand.
- It does not make any assumptions about the data, so it is flexible and can work with a wide variety of data types.
- It can handle large datasets, as the prediction time is independent of the size of the dataset.

However, KNN also has some disadvantages:

- It can be computationally expensive, as the distance between the new data point and all the stored data points must be calculated.
- It can be sensitive to the scaling of the features, and it is recommended to scale the features before training the model.
- It can be sensitive to the choice of K , as a poor choice of K can result in poor performance.

Random Forest

Random Forest is an ensemble learning algorithm that is used for both classification and regression tasks. It is an improvement over bagging (bootstrapped aggregation) algorithms, as it decorrelates the trees by training them on different subsets of the data.

The algorithm works as follows:

1. A number of decision trees are trained on different subsets of the training data.
2. The data is split into subsets by selecting a random sample of the features for each split.
3. Each decision tree is trained on a different subset of the training data, and the predictions of the trees are combined to make a final prediction.
4. The final prediction is made by either taking the majority vote (for classification tasks) or the average of the predictions (for regression tasks).

Random Forest has several advantages:

- It is a fast and accurate algorithm that is easy to implement and tune.
- It can handle a large number of features, as it selects a random subset of the features for each split.
- It is resistant to overfitting, as the decorrelation of the trees reduces the variance of the model.
- It provides a good approximation of the feature importance, as it calculates the feature importance by averaging the feature importance of all the trees in the forest.

However, Random Forest also has some disadvantages:

- It can be sensitive to the scaling of the features, and it is recommended to scale the features before training the model.
- It can be difficult to interpret, as the decision trees in the forest are not easily explainable.
- It can be computationally expensive, as a large number of decision trees must be trained.

3.1.2 Deep Learning Algorithms

A feed-forward neural network is a type of artificial neural network that is used for both classification and regression tasks. It consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the input data, and the output layer produces the predictions. The hidden layers process the data and transmit it to the next layer. The network is trained using an optimization algorithm such as gradient descent, which adjusts the weights and biases of the network to minimize the error between the predicted output and the true output. The error is typically measured using a loss function such as mean squared error (MSE) for regression tasks or cross-entropy loss for classification tasks.

The feed-forward process can be described mathematically as follows:

The input data, x , is transformed using the weight matrix W , and the bias vector b with equation 3.1.4:

$$z = Wx + b \quad (3.1.4)$$

The transformed data is passed through an activation function, f , which is used to introduce non-linearity into the model as equation 3.1.5:

$$a = f(z) \quad (3.1.5)$$

The output of the activation function is passed through the output layer 3.1.6 to produce the final prediction, y :

$$y = g(a) \quad (3.1.6)$$

Where y is the predicted output, a is the output of the activation function, and z is the transformed data. The activation function, f , is typically a non-linear function such as

sigmoid, tanh, or ReLU, and the output function, g , is typically a linear function such as the identity function or the softmax function.

The weights and biases of the network are adjusted during the training process by minimizing the error between the predicted output and the true output using an optimization algorithm such as gradient descent. The weights and biases are updated according to equation 3.1.7 and 3.1.8:

$$w = w - \alpha \nabla w L \quad (3.1.7)$$

$$b = b - \alpha \nabla b L \quad (3.1.8)$$

We have trained simple feed-forward neural networks with different hyperparameters configuration settings. We have tried out 100 and 300 dimensions of GloVe word embeddings. The sentence embeddings are calculated as component wise mean of word embeddings in the sentence. The network contains five head for each classification category. We used ReLU activation function (see Figure 3.2 and equation 3.1.9) on each layer as empirically it is empirically proved to work well. ReLU is also simple and fast.

$$ReLU(x) = \max(0, x) \quad (3.1.9)$$

The derivative function (see Figure 3.2 and equation 3.1.10) of the ReLU is also simple but it is undefined when $x = 0$. When $x = 0$ derivative is considered as 0.

$$\frac{d}{dx} ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \end{cases} \quad (3.1.10)$$

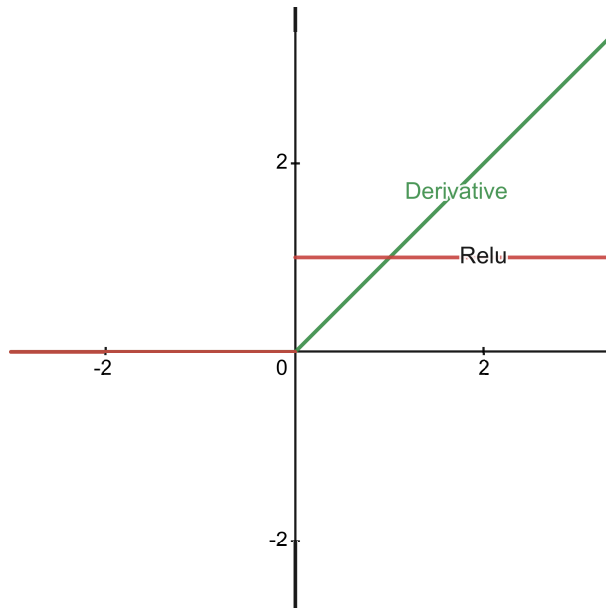


Figure 3.2: ReLU function and ReLU derivative

Each classification head in the network is a binary classifier having sigmoid function (see Figure 3.3 and equation 3.1.11) as loss function. The cumulative loss is the sum of the loss on all classification heads.

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (3.1.11)$$

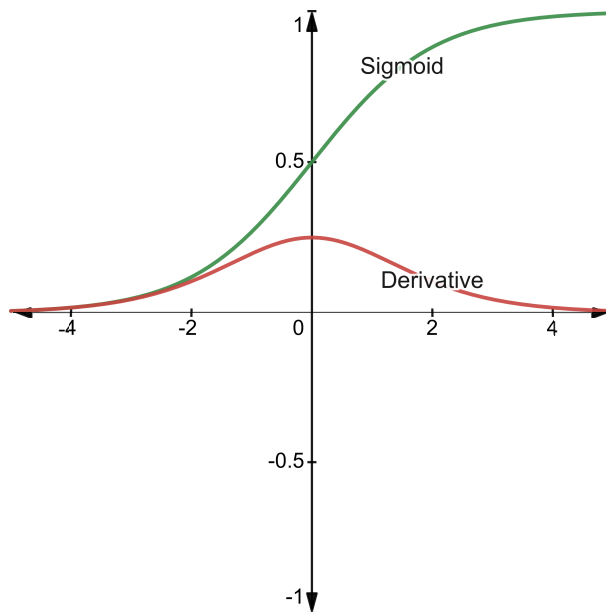


Figure 3.3: Sigmoid function and Sigmoid derivative

The derivative of sigmoid function 3.1.12 is calculated as:

$$d\text{Sigmoid}(x)/dx = \text{Sigmoid}(x)(1 - \text{Sigmoid}(x)) \quad (3.1.12)$$

We used dropout with probability 0.5 on different layers of the network. Dropout helps in regularizing the network. Dropout simply drops neuron from the network with specified probability, see the comparison of standard network and dropout network in figure 3.4

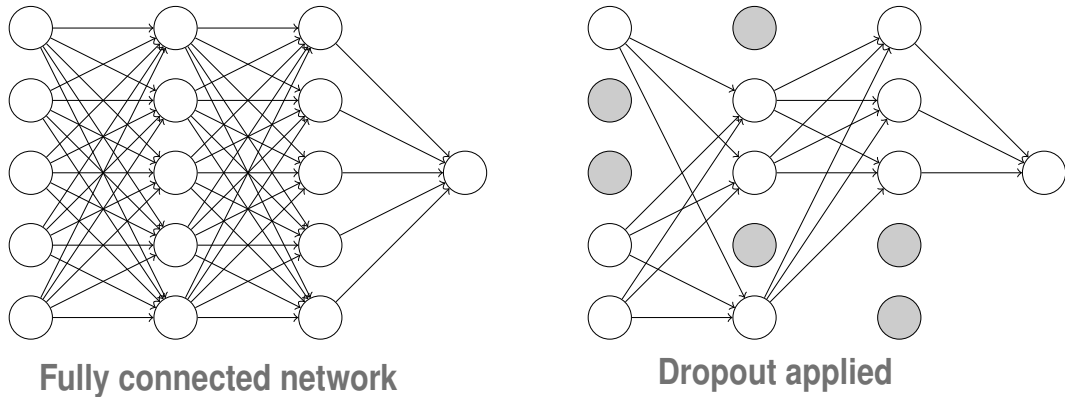


Figure 3.4: Standard Network vs Dropout applied Network

We used Adam optimizer to optimize the weights of the network. Adam is the combination of RMS Prop and Momentum algorithms.

We also trained transformer based models that are RoBERTa, BERT, and XLNet. Transformer based model uses transformer blocks as basic component. Transformer block uses self-attention mechanism to give attention to relevant parts of the sequence: see Figure 3.5 utilized the Adam optimizer to optimize the weights of the network. Adam is an optimization algorithm that combines the concepts of RMSprop and Momentum. RMSprop is a technique that adapts the learning rate based on the moving average of squared gradients, while Momentum helps accelerate gradient descent in the relevant direction.

In addition, we trained transformer-based models including RoBERTa, BERT, and XLNet. Transformer-based models employ transformer blocks as fundamental components. The transformer block incorporates a self-attention mechanism, enabling it to focus on relevant parts of the sequence. This mechanism allows the model to capture dependencies and relationships effectively, leading to improved performance in various natural language processing tasks. Refer to Figure 3.5 for an illustration of the transformer block architecture.

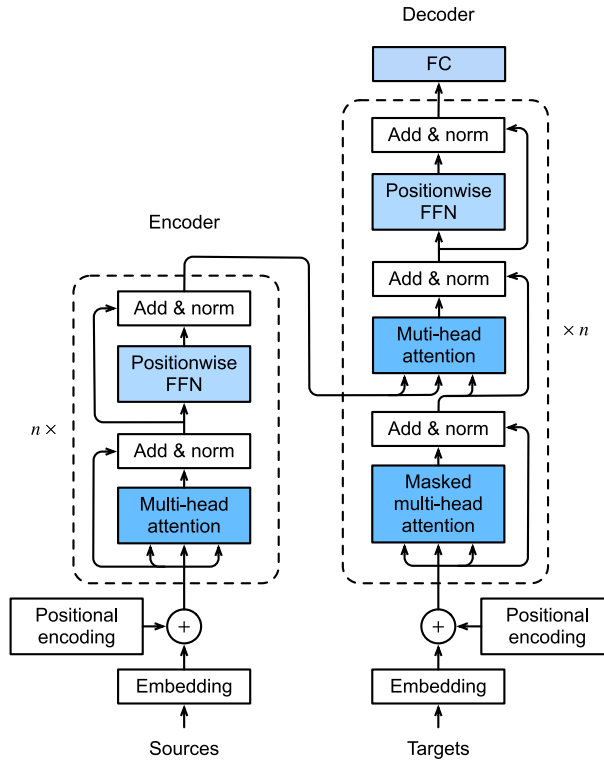


Figure 3.5: Transformer Architecture

Transformer has an encoder and a decoder network. For our task, we only use the encoder part of the network.

The first layer of transformer-based models is the input embedding layer, which takes in the input ids of the input text from the vocabulary of the tokenizer. These input ids correspond to the unique identifiers for each word in the vocabulary. During training, the input embedding layer learns the embeddings for each word in the vocabulary, allowing the model to represent the meaning and context of the words in a numerical form. One notable difference between transformer-based models and other types of neural networks, such as recurrent neural networks, is that the input text tokens are processed in parallel by the input embedding layer rather than being processed in a sequential manner. To maintain the sequence information of the input, the transformer-based model includes additional components such as self-attention layers that are capable of capturing the dependencies between the tokens. The position information for each token in the sequence is calculated by equation 3.1.13 and 3.1.14.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{3.1.13}$$

$$PE_{(\text{pos}, 2i+1)} = \cos(\text{pos}/10000^{2i/d_{\text{model}}}) \quad (3.1.14)$$

To incorporate positional information into the input representation of transformer-based models, we can use positional vectors that have the same dimensions as the token embeddings. These positional vectors encode the relative or absolute position of each token within the sequence. By concatenating the positional vectors with the target token embeddings, we can generate positional input embeddings that include both the semantic information of the tokens and their position within the sequence. This allows the model to more effectively capture the dependencies between the tokens and understand the context in which they appear. The use of positional input embeddings can therefore improve the performance of transformer-based models on various natural language processing tasks.

After encoding the positional information to each token, the input matrix X (text as sequence of tokens with positional information) is converted to query, key, and value matrix. Each row in the query, key, and value matrix correspond to the tokens in the sequence. Query, key, and value matrices has their own weight matrices that are learned during the training process by backpropagation. see equation 3.1.15, 3.1.16, and 3.1.17.

$$\text{query} = X \times W_{\text{query}} \quad (3.1.15)$$

$$\text{key} = X \times W_{\text{key}} \quad (3.1.16)$$

$$\text{value} = X \times W_{\text{value}} \quad (3.1.17)$$

The attention score is calculated by equation 3.1.18 from query, key, and value matrices.

$$Z_i = \text{softmax}\left(\frac{Q_i K_i}{\sqrt{d_k}}\right) V_i \quad (3.1.18)$$

The attention scores describes that how much attention each token in the text sequence will give to every other token in the same text sequence. The query, key, value matrices and attention score combined is called self attention mechanism. The self-attention mechanism can be converted to multi-head attention by using more than one query, key, and value matrices and their own corresponding weight matrices. The output of each head is concatenated to get output vector. A linear layer is applied to the output vector. The linear layer combined with multiple self-attention heads is called multi-head attention.

The output vector of multi-head attention is point-wise added to positional input embeddings to for Residual Connection. The Residual Connection helps in faster convergence and tackle vanishing gradient problems by skipping some layers and feeding the input directly to the later layers in the network: see Figure 3.6.

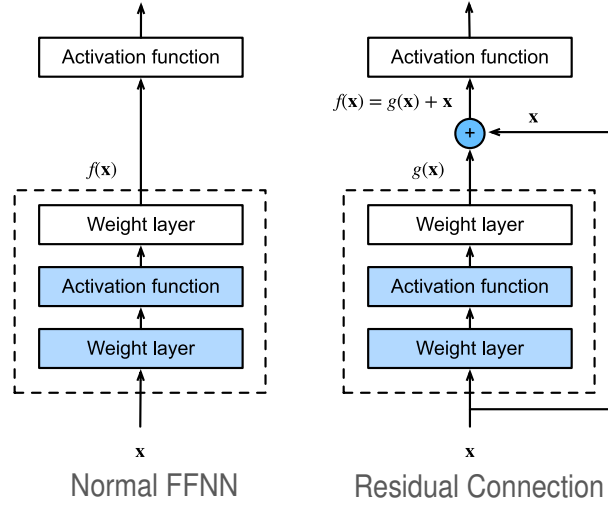


Figure 3.6: Simple Feedforward Network vs Residual Connections Feedforward Network

The output of residual layer is passed through a *Layer Normalization*. The Layer Normalization takes the output/activations x_i having K dimensions, of each individual training example and normalize it to mean 0 and standard deviation 1 by using equation 3.1.20, 3.1.20 and 3.1.21 and convert it to y_i vector having dimensions K .

$$\mu_i = \frac{1}{K} \sum_{k=1}^K x_{i,k} \quad (3.1.19)$$

$$\sigma_i^2 = \frac{1}{K} \sum_{k=1}^K (x_{i,k} - \mu_i)^2 \quad (3.1.20)$$

$$\hat{x}_{i,k} = \frac{x_{i,k} - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad (3.1.21)$$

After normalization, the normalized activations are passed through a linear layer with learnable parameters γ and β : see equation 3.1.22.

$$y_i = \gamma \hat{x}_i + \beta \equiv \text{LN}_{\gamma,\beta}(x_i) \quad (3.1.22)$$

The Layer Normalization is used to stabilized the training process.

The output of previous step is passed through a point-wise feed-forward neural network which is basically layer of two stacked linear layers applied to each position in the sequence. The output of the first linear layer is also passed through ReLU activation function: see equation 3.1.23

$$\text{FFN}(y_i) = \max(0, y_i W_1 + b_1) W_2 + b_2 \quad (3.1.23)$$

The point-wise feed-forward component is used to process the information further. The point-wise feed-forward layers also uses the residual connection and layer normalization to stabilize the training process and help the model in quick convergence.

All these components combined are collectively called the encoder of transformer-based models. Classification heads are attached to the output of this layer. In our case, a classification head with 5 heads is connected for each different category. The classification head consist of a linear layer. We use BERT, RoBERTa, and XLNet that are transformed-based architecture on our task.

BERT

BERT (Bidirectional Encoder Representations from Transformers) 3.7 is a powerful transformer-based model that has achieved state-of-the-art results on a wide range of natural language processing tasks. BERT is designed to process sequential data by considering the context from both the left and the right sides of a token, allowing it to capture the dependencies between the tokens in a more effective manner than models that only consider a single context. To achieve this bidirectional context, BERT makes use of self-attention layers that are capable of attending to any position in the input sequence.

One key feature of BERT is its use of transformers, which are a type of neural network architecture that is particularly well-suited to processing sequential data. Transformers use self-attention mechanisms to calculate the relationships between the tokens in the input sequence, allowing them to capture long-range dependencies and attend to relevant contextual information. This makes them an ideal choice for tasks such as language translation, question answering, and language modeling.

In addition to its use of transformers and self-attention layers, BERT also includes a number of other innovative features that contribute to its effectiveness. For example, it is

trained using a special masking technique that allows it to make predictions for masked tokens based on the context provided by the surrounding tokens. This pre-training procedure allows BERT to learn general-purpose language representations that can be fine-tuned for specific tasks. Overall, BERT is a highly effective and widely-used model for natural language processing, and has been successful in a range of applications.

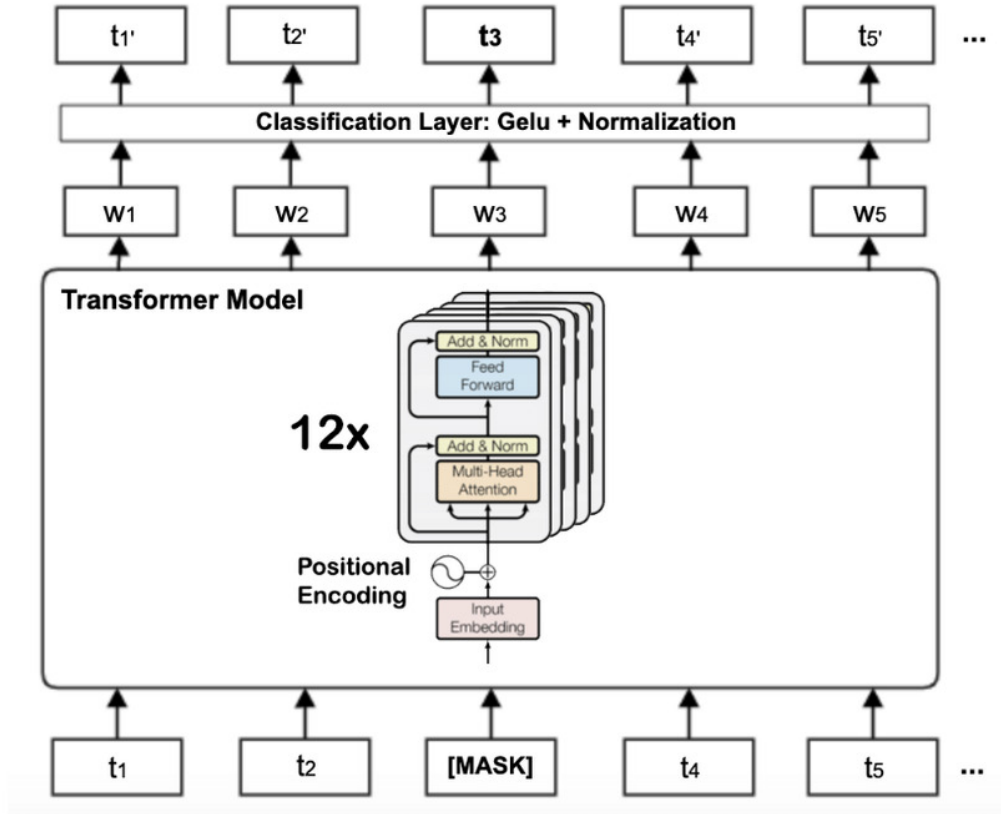


Figure 3.7: BERT

RoBERTa

RoBERTa (Robustly Optimized BERT Pretraining Approach) is a transformer-based model that was developed to improve upon the BERT model. It was introduced in a paper by Facebook AI in 2020 and has since achieved strong results on various natural language processing tasks.

Like BERT, RoBERTa is trained on a large dataset of unannotated text and is designed to learn general-purpose language representations that can be fine-tuned for specific tasks. However, RoBERTa differs from BERT in a number of key ways that are intended to improve the model's performance.

One key difference between RoBERTa and BERT is the size of the dataset used for pretraining. BERT was trained on a dataset of approximately 3.3 billion words, while RoBERTa was trained on a much larger dataset of approximately 160 billion words. This larger dataset allows RoBERTa to learn more comprehensive and robust language representations, which can improve its performance on various tasks.

Another difference between RoBERTa and BERT is the way in which the models are trained. BERT uses a masking technique in which it is trained to predict the identities of randomly masked tokens based on the context provided by the surrounding tokens. RoBERTa, on the other hand, uses a different masking strategy in which it is trained to predict the next sentence given a sequence of tokens from the current sentence. This training approach allows RoBERTa to better capture the relationships between sentences and improve its performance on tasks that require understanding of the broader context.

RoBERTa also includes a number of other modifications that are designed to improve the model's performance. For example, it uses a larger batch size and longer sequences during training, and applies a dynamic masking strategy that adjusts the masking probability based on the length of the input sequences. These and other modifications help RoBERTa to learn more effective language representations and achieve strong results on various natural language processing tasks.

RoBERTa is a powerful transformer-based model that builds upon the successes of the BERT model and introduces a number of improvements that are designed to enhance its performance. It has demonstrated strong results on various natural language processing tasks and is widely used by researchers and practitioners in the field.

We add a classification head for both BERT and RoBERTa to use it as a classifier. see [Figure 3.8](#)

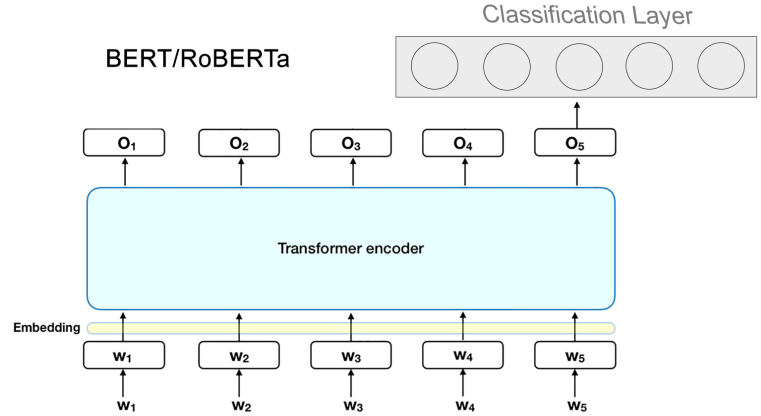


Figure 3.8: BERT

RoBERTa

XLNet

XLNet (eXtremely Large Neural Network) 3.9 is a transformer-based model that was developed to improve upon the performance of existing language models such as BERT and GPT-2. It was introduced in a paper by researchers at Carnegie Mellon University and Google in 2019 and has since achieved strong results on a variety of natural language processing tasks.

One key feature of XLNet is its use of a novel attention mechanism called permutation language modeling. In traditional language models, the goal is to predict the next word in a sequence given the previous words. However, in permutation language modeling, the model is trained to predict any word in the sequence given all of the other words in the sequence. This allows XLNet to better capture the dependencies between the words in the sequence and improve its performance on tasks such as language translation and question answering.

Another important feature of XLNet is its use of autoregressive modeling, which means that the model is trained to predict each word in the sequence based on all of the previous words. This is in contrast to the masked language modeling approach used in BERT, in which the model is trained to predict a masked word based on the surrounding context. Autoregressive modeling allows XLNet to capture more comprehensive dependencies between the words in the sequence, which can improve its performance on various natural

language processing tasks.

In addition to its use of permutation language modeling and autoregressive modeling, XLNet also includes a number of other innovations that are designed to improve its performance. For example, it uses a variant of the transformer architecture that is optimized for autoregressive modeling, and includes a number of modifications to the self-attention mechanism that are intended to improve the model’s ability to capture long-range dependencies. These and other features help XLNet to learn effective language representations and achieve strong results on a range of natural language processing tasks.

Overall, XLNet is a powerful and innovative transformer-based model that has demonstrated strong performance on a variety of natural language processing tasks. It builds upon the successes of previous models such as BERT and GPT-2 and introduces a number of new features that are designed to further improve its ability to understand and process natural language. One key difference between XLNet and BERT and RoBERTa is the way in which the models are trained. BERT and RoBERTa are trained using a masked language modeling approach, in which the models are trained to predict the identities of randomly masked tokens based on the context provided by the surrounding tokens. XLNet, on the other hand, is trained using a permutation language modeling approach, in which the model is trained to predict any word in the sequence given all of the other words in the sequence. This allows XLNet to better capture the dependencies between the words in the sequence and improve its performance on tasks such as language translation and question answering. Another key difference between XLNet and BERT and RoBERTa is the use of autoregressive modeling. In autoregressive modeling, the model is trained to predict each word in the sequence based on all of the previous words. This is in contrast to the masked language modeling approach used in BERT and RoBERTa, in which the model is only trained to predict a masked word based on the surrounding context. Autoregressive modeling allows XLNet to capture more comprehensive dependencies between the words in the sequence, which can improve its performance on various natural language processing tasks.

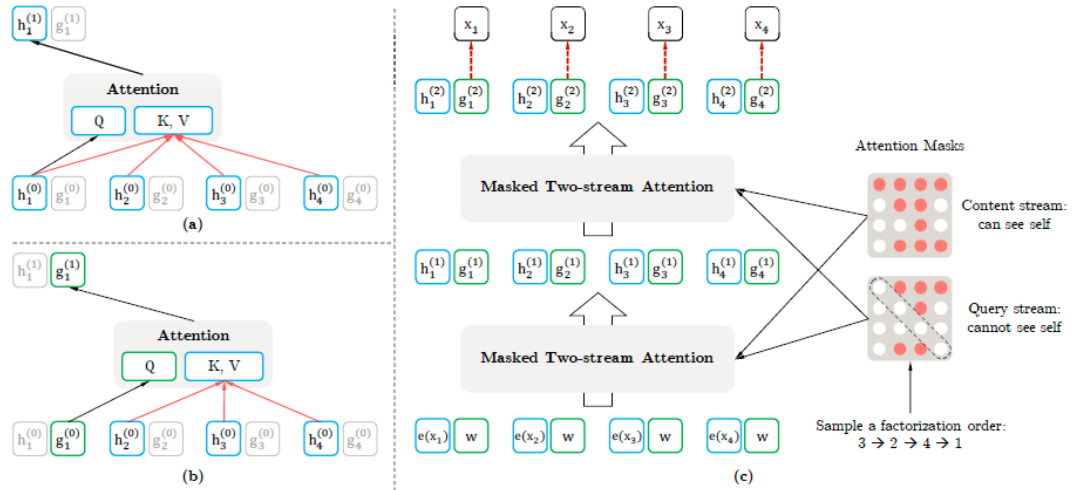


Figure 3.9: XLNet

3.2 Conversational System [RASA Framework]

RASA (Open Source Dialogue Engine for Conversational AI) is a powerful and flexible framework for building intelligent conversational systems. It allows developers to create chatbots and other conversational systems that can understand and respond to user input in a natural and human-like manner. The RASA Open Source 3.10 has two big components: NLU Pipeline and Dialogue Policies. At the core of RASA, NLU Pipeline, is a set of machine learning-based components that handle tasks such as natural language understanding and generation. These components are trained on data annotated with information about the user’s intentions and the structure of the input text. This allows RASA to understand the user’s intent and respond appropriately, even when the user’s input is ambiguous or contains errors. The Dialogue Policies takes next action according to the context of conversation. In addition to its machine learning-based components, RASA also includes a number of tools and features that make it easy for developers to build and deploy conversational systems. These include a visual interface for creating and managing conversations, support for a wide range of popular messaging platforms and channels, and integration with other tools and services.

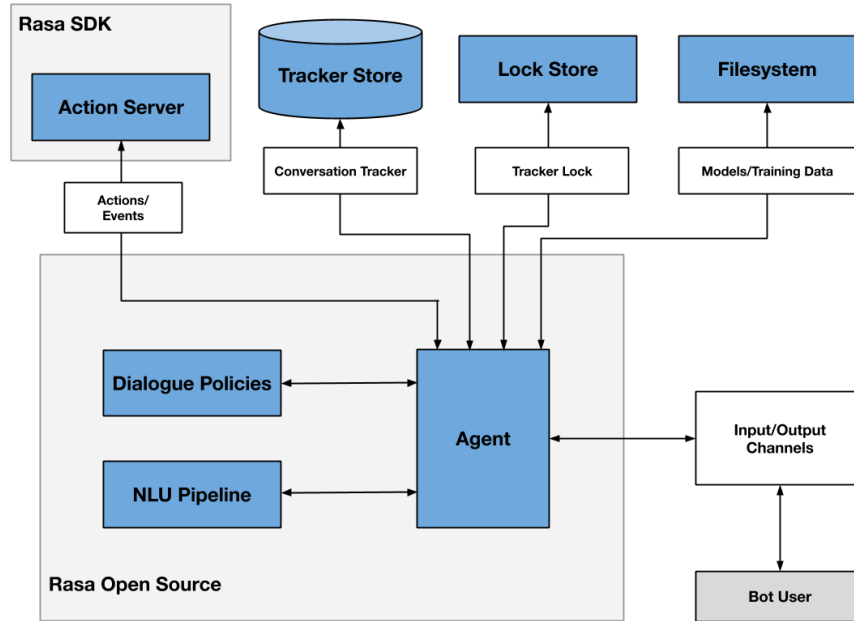


Figure 3.10: RASA overall architecture

3.2.1 Components

Components in RASA are integrated in a single pipeline for understanding of the users input. Components can be defined for different tasks. For example, MITIE [21] can be used for feature extraction and entity extraction in the pipeline. MITIE comes with pre-trained language models, however it can be fine-tuned on custom datasets. SpacyNLP can be integrated into the pipeline to perform certain functions. Different **Tokenizers** can be integrated into the pipeline, for example, WhitespaceTokenizer, JiebaTokenizer (specifically for Chinese language), MitieTokenizer, and SpacyTokenizer. Featurizers component helps RASA to convert text to word-vectors. Featurizer can be either sparse and dense. Certain featurizers can be used with RASA. Some dense featurizers are MitieFeaturizer, SpacyFeaturizer, and ConveRTFeaturizer. Some sparse featurizers are RegexFeaturizer, CountVectorsFeaturizer, and LexicalSyntacticFeaturizer. A featurizer can either return word-vector or sentence-vector for an input text. The LanguageModelFeaturizer can be used to get word-vectors with BERT, GPT, XLNet, RoBERTa etc. from HuggingFace ¹ models.

Intent Classifiers can be defined as a component of the pipeline. Certain classifiers can

¹<https://huggingface.co/>

be added including `MitieIntentClassifier` [21], `LogisticRegressionClassifier` [23], `SklearnIntentClassifier` [23], `KeywordIntentClassifier`, `DIETClassifier` [87], and `.`. Some classifiers uses their own embeddings, while other used embeddings defined in the pipeline. Each classifier has its own training settings which can be configured as of user or task requirements.

Entity Extractors can be integrated into the pipeline to extract structural information such as names, locations, date, time etc. from the user utterance. `MitieEntityExtractor` [21], `SpacyEntityExtractor`, `CRFEntityExtractor`, `DucklingEntityExtractor`, `DIETClassifier` for entity extraction, and `RegexEntityExtractor` can be used as entity extractors. `EntitySynonymMapper` can be used to map entities to their synonyms. `EntitySynonymMapper` requires entity to be extracted by an entity extractor beforehand.

Response selection is a critical component of any conversational AI system, as it determines the appropriate response to a user’s input. In the RASA framework, the `Response Selector` is a module that selects the best response from a set of candidate responses based on the context of the conversation and the user’s input. The `Response Selector` uses a machine learning model to score and rank the candidate responses, and selects the one with the highest score as the final response. This model is trained on a dataset of conversation examples, which includes both the user’s input and the appropriate response for each example. There are several factors that the `Response Selector` takes into account when scoring and ranking the candidate responses. These include the relevance of the response to the user’s input, the tone and style of the response, and the context of the conversation. One of the key challenges in response selection is handling out-of-scope inputs, which are inputs that the system is not trained to handle. In these cases, the `Response Selector` may need to fall back on a default response or ask the user for clarification.

3.2.2 Policies

`Policies` decides at each step that which action should be taken by conversational AI system in contrast to each utterance of a user. RASA uses different approaches for this tasks. A deep learning based model called `Dialogue Transformers` [84] can be used to train `TEDPolicy` and `UnexpectTEDIntentPolicy` to pick the next action. `Dialogue Transformer` 3.11 is a transformers based architecture.

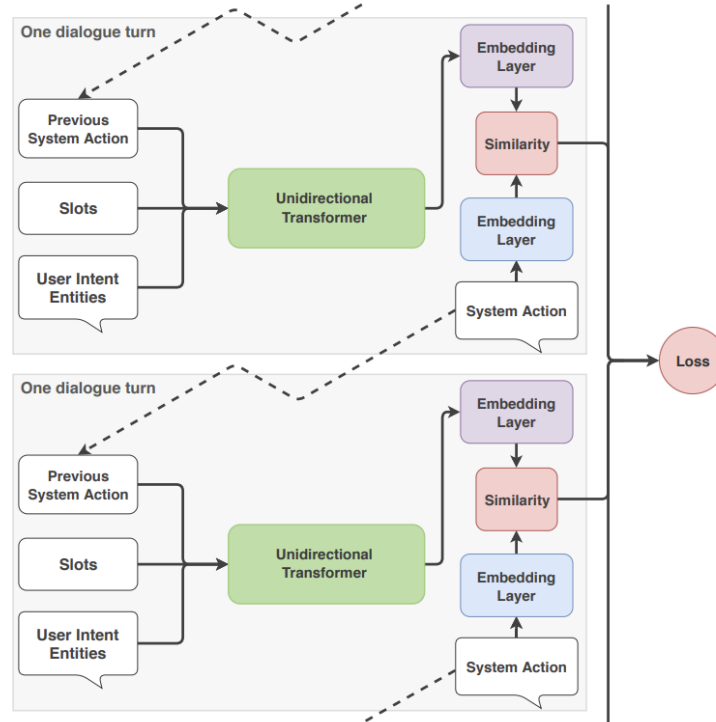


Figure 3.11: Dialogue Transformers Architecture

First, the user inputs and actions are featurized and given to the transformer encoder along with the previous system actions in dialogue history. Dialogue Transformer decides how much attention to put on each dialogue turns in the dialogue history. The model is trained to maximize the dot product similarity of system actions.

Memoization Policy predict next action by remembering the stories of the training data. If the conversation follows the same story pattern as given in the data, the Memoization Policy will predict the same next action with probability 1. If conversation does not follow the same pattern of training data then Memoization Policy will predicting nothing.

A Rule-based Policy is a type of machine learning policy that uses a set of hand-coded rules to determine the appropriate action to take in a given dialogue state. The rules are defined by the developer, and may be based on a variety of factors such as the user's input, the context of the conversation, and the available actions. Rule-based policies can be useful for simple dialogue systems, or for handling specific, well-defined tasks. However, they can be difficult to maintain and scale as the complexity of the dialogue system increases. Additionally, rule-based policies may not be able to adapt to new or unexpected input, as they are limited to the rules that have been explicitly coded by the

developer. For example, If a user say "Good Bye" then the system would definitely say "Good Bye" or something related but never something like "Hello" or "How can I help you?" etc. RASA uses Rule Policy for these cases in conversation where the conversation follow strict patterns.

3.2.3 Actions

In the RASA framework, actions are responses that the conversational AI system can take in response to user inputs. Actions can take many different forms, including simple text responses, API calls, database queries, or the presentation of forms to the user. One of the key benefits of using actions in a conversational AI system is that they allow the system to perform a wide range of tasks beyond simply generating text responses. For example, actions can be used to retrieve information from external sources, update databases, or interact with other systems. This makes it possible to build sophisticated and highly functional conversational AI systems that can provide a wide range of services to users. There are several types of actions that are supported by the RASA framework, including: Text responses: Simple text responses are the most basic type of action, and are used to provide information or replies to user inputs. Text responses can be static or dynamic, depending on the needs of the system.

API calls: Actions can be used to make calls to external APIs or systems, which can be useful for retrieving or updating information or interacting with other systems.

Database queries: If the conversational system has access to a database, actions can be used to retrieve or update information stored in the database.

Forms: Actions can be used to present forms to the user and collect information from them. This can be useful for gathering data from the user or gathering feedback. RASA forms are utilized to gather structured information from users. These forms enable the specification of a series of questions that a chatbot should ask in a specific order until all required information has been collected. RASA forms are particularly useful when a significant amount of structured information needs to be gathered from users or when the format of the collected information is crucial. For instance, a RASA form could be utilized to gather a user's name, address, phone number, and email address. To utilize a RASA form, it must first be defined in the form of a configuration file or through code. Once defined, the form can be triggered and used to gather the desired information

from the user. RASA forms are implemented with the RASA NLU (Natural Language Understanding) and Core libraries, which are open-source libraries for building chatbots and conversational AI. RASA forms are an essential component of the RASA framework, a set of tools and libraries for building and deploying chatbots and conversational AI applications. A form must be activated before in rules or stories. For example, a specific intent will trigger the form. When all slot values are filled, the form is automatically deactivated.

RASA form validation is a feature that allows to verify that the user has provided the required information when filling out a form within a RASA conversational system. This can be useful for collecting specific pieces of information from the user, such as their name, email address, or phone number. To implement form validation in RASA, it requires to create a form within the conversational flow. This can be done using the Form Action class, that allows to define the fields that the user has to fill out and the types of responses that is expected to receive. For example, one might create a form with fields for the user's name, email address, and phone number, and specify that a text based response is expected to receive for each field. Once the form is defined, `validate` method can be used to verify that the user has provided the required information. This method takes a dictionary of user responses as input and returns a dictionary of validation results. If the required fields are filled out correctly, the validation results will be `True`. If any of the required fields is not been filled out or filled out incorrectly, the validation results will be `False`. Form validation results can be used to control the flow of the conversation. For example, if the validation results are `True`, the next step in the conversation will be proceed. If the validation results are `False`, the user is required to provide the missing or incorrect information before continuing. Form validation is a powerful tool that ensure to receive the necessary information from the user in order to provide the best possible service.

Slots: RASA slots are a crucial aspect of the RASA framework for building chatbots and conversational AI applications. They are used to store information that is gathered during a conversation with a user and to track the state of the conversation. In the context of chatbots, slots can be used to store a wide range of information, including user input, conversation history, and the results of calculations or other processing. For example, a chatbot that is designed to help users book flights may use slots to store the destination, departure date, and number of passengers that the user has specified.

One of the key benefits of using slots is that they allow chatbots to maintain context and to remember important information across multiple turns in a conversation. This is important because it enables the chatbot to provide a more natural and seamless conversation experience for the user. There are several types of slots that can be used in the RASA framework, including categorical slots, float slots, and text slots. Categorical slots are used to store information that belongs to a finite set of categories, such as the names of cities or the types of products that a chatbot can offer. Float slots are used to store numerical values, such as prices or quantities, and text slots are used to store free-form text input from the user. In addition to these types of slots, the RASA framework also provides a range of tools and features for managing and manipulating slots, such as the ability to set default values for slots and to reset the values of slots at specific points in a conversation.

Custom actions: Developers can also define custom actions that perform specific tasks or perform actions that are not supported by the built-in action types.

If a user says "Hello", the system will reply with something like "Hi, how are you?". The action returned by system is a simple text. A response text can have place holders from slots. For example, a user utters that "Hello, I'm John". The system can extract John from the user utter and reply with a custom response like "Hello John, how can I be of service?". A response can have multiple messages. RASA can be integrated into different channels like Slack, Facebook messenger, Twitch etc. so different messages for each channel can be added as a response. Rich responses can be produced by adding buttons to the response message. Images can be added as response message along with text. Date pickers can be sent to user as response to select a date from it rather than typing. A response can also be called from custom action functions.

Custom Actions are used when the action needs some custom logic to be performed before returning the response to the user. For example, If a user ask about weather of a location, then the system will call some weather API to fetch the weather information of the specified place. Or if a user ask to add something to his calendar then the conversational system will perform some database query to insert data into the database. If a user needs to check his bank balance, then conversational system will call the bank API to check his balance.

There are also some default actions with the ability to be customized in RASA which

helps user in many situations. Some examples of default actions are *action_listen*, *action_restart*, *action_session_start*, *action_back*, *action_unlikely_intent* etc. These default actions can be modified as of system requirements.

3.2.4 Integration by Input/Output Channels

There are several ways in which the RASA conversational system can be integrated into an existing application or platform with Input/Output Channels 3.10. One option is to use the RASA chat widget, which allows users to interact with the system directly from the application interface. Another option is to use the RASA X utility, which provides a web-based graphical user interface through which users can access the system. In addition, RASA can be integrated with custom chat widgets using the REST channels. Alternatively, RASA can also be integrated with popular social and professional networks, such as Facebook Messenger, Slack, Telegram, Twilio, and Google Hangout Chats, as well as RocketChat. This versatility allows RASA to be used in a wide range of contexts and environments, making it a powerful tool for building sophisticated conversational systems.

3.2.5 RASA Action Server

The RASA action server is a tool that allows RASA Open Source to execute custom actions. When the action server receives a request, it processes the provided conversation information, such as the conversation ID and tracker data, and performs the requested action. This action could involve performing a database query, making an API call, or any other form of processing. The action server can be implemented using any programming language, and the RASA SDK 3.10 is included with the RASA installation, although other action servers can be used as long as they support the required API. The action server is an essential part of the RASA Open Source system and allows it to perform a wide range of custom actions in order to better serve the needs of the user.

3.2.6 Tracker Store

Tracker Store component of figure 3.10 is used for storing the users conversations. By default, RASA uses InMemoryTrackerStore mechanism that stores the conversations in the memory. When the Bot is restarted or the conversation time ends, the conversations

are removed from memory, and no records if left. It works in some scenarios but sometimes permanent storage of conversation is required. RASA provides `SQLTrackerStore`, `RedisTrackerStore`, `MongoTrackerStore`, and `DynamoTrackerStore` where a simple configuration helps to connect to the database for conversations. Besides that, custom conversation trackers can also be created.

3.2.7 Lock Store

Rasa incorporates a mechanism known as the "ticket lock" to maintain the accurate sequencing of incoming messages associated with a given conversation ID. This system serves the purpose of upholding conversation integrity and consistency, particularly when multiple Rasa servers operate concurrently as replicated services. The "ticket lock" approach functions by temporarily securing conversations during the active processing of messages. Upon the arrival of a message tied to a specific conversation ID, Rasa ensures its proper order of processing by acquiring a lock specifically assigned to that conversation. This precautionary measure prevents potential disruptions to the conversation's sequence caused by simultaneous message processing. Through the implementation of the ticket lock mechanism, Rasa effectively expands its scalability by operating multiple server instances in parallel and distributing the workload among them. Moreover, this mechanism empowers clients to transmit messages pertaining to a particular conversation ID to any server instance with the assurance that the messages will be processed in the correct order, irrespective of the server responsible for handling them.

3.2.8 File System

RASA maintains a proper Filesystem [3.10](#) for training data and models. The training, testing, and validation data are given in the YAML file format. The models can be stored on local disk, on a local disk, on a remote, or cloud server. The models can be accessed via HTTP from a URL from a server with a small configuration in `endpoints.yml`. RASA is compatible with AWS S3, Azure Blob Storage, and Google Cloud Storage with a setting a few environment variables.

3.2.9 RASA X

RASA X² is a graphical interface designed to support Conversation-Driven Development (CDD). It is built on top of the open-source RASA framework and helps developers gather data from users in order to build more effective chatbots and conversational AI systems. One of the primary challenges of building conversational systems is that it is impossible to predict with 100% accuracy what a user will say or ask. RASA X helps developers overcome this challenge by providing a tool that makes it easy to collect data from users and use it to improve the performance of their chatbots. In this way, RASA X helps developers build more effective and engaging conversational systems that are better able to meet the needs of their users.

²RASA X is no longer an Open Source Product.

Experiments & Results

Different experimental setups were used for message classification and conversational system.

4.1 Message Classification

In this section, we present the dataset, experimental setup, and classification algorithms results.

4.1.1 Dataset

Message classification dataset has 97,710 messages with their labels for 5 classes. Each message can have labels **call to action**, **common ground**, **establish credibility**, **incentive for connecting**, and **personalization**.

Call to action means that the prospect will most likely reply to this message.

Common ground means that the recruiter and the prospect might have worked in the same company before, or having some common educational background, or same hobbies, or same favorite things. It can be anything which is common between both the recruiter and the prospect.

Establish credibility means that the recruiter established credibility for his work, or the company for which he is hiring candidates, or the role/job.

Incentive for connecting means that the recruiter writes about the incentives and benefits that the company will be giving him for his role.

Personalization shows that how much the message is personalized. For example, If the recruiter writes his name, the prospect name, or mention his current/previous company, or job experience then it is a personalized message. The following message is a direct example from the dataset which is having call to action, establish credibility, incentive for connecting, and personalization labels.

this is (firstname) from (my_company), the only data science solutions provider in the region. i think given your work experience at (their company name) you should be a part of this revolution. are you connected to the (my_company). our company is growing its team each day and we would like to add to it even more. how about if i discuss this role further with you?

The message is basically a templated message and values can be filled from the LinkedIn API. The sentence piece *how about if i discuss this role further with you?* is majorly responsible for *call to action* label. The *(firstname)*, *(my_company)* and *i think given your work experience at (their company name)* makes the message personalized. The sentence piece *the only data science solutions provider in the region* gives credibility to the company. *our company is growing its team each day* adds somehow incentives to the prospect.

There are total 332 unique sentence pieces in the dataset. Also, there are total 723 unique words when we tokenize all messages with gensim tokenizer ¹. Number of unique sentence pieces and unique words are small which makes the data distribution condensed in the dataset. This makes the problem very easy for us and we always get close to 100% accuracy when when train any model on it. To overcome this problem, we added some newly annotated data with diverse distribution to the test set. We made the test set smaller so that the model generalizes well on new data. 94,010 data samples goes to train-set and 3,700 becomes part of the test set. It means that 4% data becomes part of the test-set while the remaining goes into training set. The newly annotated data is taken from real world recruiters is kept into the test-set.

¹<https://tedboy.github.io/nlps/generated/generated/gensim.utils.tokenize.html>

4.1.2 Experimental Setup

We used Google Colab for all machine learning and deep learning experiments. Google Colab ² is a cloud platform provided by Google Research. As the time of this research, google provide three different plans for colab: **Colab free**, **Colab Pro pro**, and **Colab Pro+**. We used the free version of colab for training our models. Even though the specification for free version of colab changes with time and also depends on the availability of the CPU, however, the hard disk space 4.1, memory information 4.3, CPU information 4.2, and GPU information 4.1 is as follows.

	Space	Used	Available
On CPU	108GB	39GB	70GB
On GPU	79GB	39GB	40GB

Table 4.1: Google Colab: Disk space information

	Model name	CPU MHz	cache size	CPU cores
CPU: 0	Intel(R) Xeon(R) CPU @ 2.20GHz	2200.162	56320 KB	1
CPU: 1	Intel(R) Xeon(R) CPU @ 2.20GHz	2200.162	56320 KB	1

Table 4.2: Google Colab: CPU information

MemTotal	MemFree	MemAvailable
~13.30 GB	~10.61	~12.46

Table 4.3: Google Colab: Memory (RAM) information

²<https://colab.research.google.com/>

```

Wed Jun 29 09:31:29 2022
+-----+
| NVIDIA-SMI 460.32.03    Driver Version: 460.32.03    CUDA Version: 11.2    |
+-----+-----+-----+
| GPU  Name          Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap|      Memory-Usage | GPU-Util  Compute M. |
|                               |                  |              MIG M. |
+-----+-----+-----+
|   0   Tesla T4            Off     | 00000000:00:04:0 Off  |            0         |
| N/A   47C    P8             9W / 70W |  0MiB / 15109MiB |      0%      Default |
|                               |                  |              N/A     |
+-----+-----+-----+

+-----+
| Processes:                                |
| GPU   GI    CI          PID    Type   Process name          GPU Memory |
|   ID   ID   ID              |          |                  |         Usage |
+-----+-----+-----+
| No running processes found                |
+-----+

```

Figure 4.1: Google Colab: GPU Information

We could turn the GPU on or off in our colab environment. The CPU and memory specifications are same if we turn GPU on or off, however, Disk space specifications changes with turning on or off the GPU.

4.1.3 Results

We performed multiple experiments with machine learning and deep learning based algorithms with different sets of hyperparameters and logged the best empirical accuracy score.

Machine Learning Algorithms

The results of the experiments are given in table 4.4.

Algorithm	Input Size	Word_embeddings	Accuracy %
DecisionTree	300	GloVe	53.05
DecisionTree	100	GloVe	53.35
LinearSVM	100	GloVe	85.19
LinearSVM	300	GloVe	90.92
LogisticRegression	100	GloVe	83.97
LogisticRegression	300	GloVe	89.78
GaussianNB	100	GloVe	28.73
GaussianNB	300	GloVe	29.84
KNeighborsClassifier	300	GloVe	76.59
RandomForest	100	GloVe	71.22

Table 4.4: Machine Learning Algorithms Results

LinearSVM with 300-dimension embeddings size gives the best accuracy of 90.92%.

Feed-forward Neural Networks

We performed multiple experiments with Feed-forward networks with different hyper parameters but we also kept some hyperparameters constant in all experiments. Table 4.5 shows the hyperparameters of FFNN.

Word Embeds	n_layers	n_neuron	ActFunc	Batch size	Loss function	Optimizer	LR
Glove	4	[32,64,128,256]	ReLU	1024	Sigmoid	Adam	0.001

Table 4.5: Feed-forward Neural Network: fixed hyperparameters

These hyperparameters are kept fixed as their counter parts works almost the same. Keeping these hyperparameters also reduce our search space. We used GloVe embeddings as it generally gives best results over other embedding algorithms like Word2vec. Generally, a neural network with large number of layers results in better performance but we have data where data distribution is smaller so we have to keep minimum number of layer. We heuristically select four hidden layers in our network. Empirically speaking, research shows that ReLU works best as activation function in hidden layers. Sigmoid

loss function is better a option when we have 2 number of classes. Adam optimizer is the gold standard of all optimization algorithms and learning rate is selected as 0.001 arbitrarily.

The result of Feed-forward neural networks experiments are in table 4.6.

Epochs	Input Size	Accuracy %	Dropout
10	100	79.73	No
10	300	85.73	No
30	100	72.22	on layer (1,2,3,4)
30	300	87.13	on layer (1,2,3,4)
30	100	89.03	on layer (2,3,4)
30	300	92.76	on layer (2,3,4)
30	100	89	on layer (3,4)
30	300	92.67	on layer (3,4)
30	100	90.54	on layer (4)
30	300	92.24	on layer (4)

Table 4.6: Feed-forward Neural Networks: Results

The larger number of embedding size means more information. This is the reason that 300 dimension embeddings perform better than their 100 dimension counter part experiments. Applying dropout on the final layers increases the accuracy and prevent the model from overfitting.

Long-short Term Memory (LSTM)

LSTM models with different hyperparameters are trained and logged the accuracy. Some hyperparameters are kept constant (see table 4.8) as it does not have significant impact on the accuracy of the model.

Epochs	Input Size	n_neuron	ActFunc	Loss Function	Dropout	Optimizer	LR	Gradient Clipping
10	100	512	ReLU	Sigmoid	Yes	Adam	0.001	Yes [5]

Table 4.7: LSTM: fixed hyperparameters

Different experiments were performed and found that the 10 number of epochs are enough for our task. Messages in our dataset are shorted so we keep the input size to 100 tokens to tackle all input sizes. Number of neuron in hidden layers are selected as 512 as the problem in hand is not that complex so we do not need any extra neurons as it will increase the training time and the risk of overfitting. ReLU is used as activation function as research shows that it empirically converges faster. We also used the Dropout with probability 0.5 to overcome the problem of overfitting. Adam optimizer is chosen as it helps the model to converge faster and overcome local minima problem up to great extent. Learning rate is selected as 0.001 arbitrarily. We used gradient clipping because the LSTM models are often faces exploding gradient problem due to its backpropagation in time nature during parameters optimization. If gradient size on a neuron gets bigger value than 5 then that gradient value will be cut down to 5. If gradient value is smaller than 5 then it will remain the same. We do not use any word-embeddings here, the model learn the embeddings during training itself.

num_LSTM_layers	Bi-Direction	Batch_size	Accuracy %
1	No	1024	89.94
2	No	1024	93.97
3	No	1024	94.03
4	No	1024	93.95
1	Yes	1024	36.21
2	Yes	512	93.08
3	Yes	512	91.59
4	Yes	512	94.14

Table 4.8: LSTM: Results

We trained simple LSTM and Bi-directional LSTM networks with different number of layers and the result are logged in Table 4.8. We used 1024 batch size when the network was relatively smaller but when the number of parameters increased due to increasing number of layers or changing the nature of the network to bi-directional, we used 512 batch size as we have limited amount of memory on our experimental environment. Increasing the number of layers increased the capacity of the model which increases the accuracy. We get our best accuracy score of 94.14% when number of layers are 4 and

the network is bi-directional. The bi-directional nature of the model give the ability to look for the context from the future words in the sentence as well. The simple LSTM just look for the context and relationship in the previous words in the sentence.

Transform based models

We trained three famous transformer based models: BERT, RoBERTa, and XLNet.

Model	Epochs	Input Size	Batch_size	Accuracy %	training time	GPU
RoBERTa	1	128 tokens	8	94.84	3 h 49 m 05 s	K80
BERT	1	256 tokens	8	95.67	4 h 34 m 57 s	K80
XLNet	1	256 tokens	8	95.35	1 h 01 m 51 s	T4

Table 4.9: Transformed Based Models: Results

We trained transformer based models i.e. RoBERTa, BERT, and XLNet for 1 epoch and the model converged due to the simplicity of the problem. The messages lengths are shorter so we kept the input token size as 128 or 256. We kept batch size as 8 because these networks have millions of parameters and we only have 12GB of memory available for our model and data. Initially, we were assigned the K80 GPU when we were training RoBERTa and BERT models which is slower than the T4 GPU. When we were training the XLNet, the Google Colab has assigned T4 GPU for it hence the training time of XLNet is around 4 times shorter than the training time of BERT and RoBERTa. Result are logged in Table ?? for comparison. Accuracy of these models are closer to each other. All models uses attention mechanism to better capture the context and semantic relationship between words. We get highest accuracy of 95.67% from BERT model but the training time is slightly higher than RoBERTa which are trained on the same K80 GPU. Our best model from the accuracy point of view is **BERT** among all other machine learning and deep learning based models.

4.2 Conversational AI System

4.2.1 Dataset

There is no such dataset available for conversation system of our work, so we created our own dataset. The dataset of conversational system contains two types of data: **NLU data** and **Conversation stories**.

NLU data contains prospects text messages examples with their intent (class) and entities. There are total 35 intent classes. Some of them are *ask_salary*, *job_requirements*, *meeting_schedule*, *company_info*, *role*, etc.

ask_salary means when the prospect ask for salary information. Some examples from dataset are:

- What's the salary for this role?
- what is the expected salary for this job?
- may I know about the salary for this opportunity?
- may i ask about the salary?
- it would be good if i get the idea about the salary for this job.

job_requirements intent means that when the prospect ask about the requirements of the job. Some example from the dataset are:

- can you share the requirements for this Post?
- what are the job requirements?
- what is the required qualification?
- what is the required experience?
- minimum required experience?

meeting_schedule intent means that when the prospect schedule a meeting with the recruiter. Some example from the dataset are:

- I've schedule it for Wednesday 13th this week.
- Sure. I have set the time on Friday from 10:00 PM to 10:30 PM. Can you please tell me about the company and its background for the better context?
- I just noticed that there isn't any options for 3pm, so I picked 9:30pm for May 19,2021

- What about May 19, 2021 at 3PM Pakistan time?
- Hi let's connect at 9 pm on Tuesday May 18 if you have that slot open.

In the *company_info* intent, the prospect asks about the company. Just like that, in the *role* intent the prospect asks about the job/position/role? There are total 783 number of examples in the dataset. Some of the examples are human written while some are generated using Machine Learning with the help of paraphrasing by using pre-trained paraphrase model Pegasus ³.

Conversation stories are conversations patterns between the prospect and the recruiter. A typical conversation starts with greetings and ends with goodbye message. The simplest conversation story is when the recruiter sends job advertisement/offer message to the prospect and the prospect is not interested in the offer due to some reasons: see the following story from dataset.

- story: not interested path
- steps:
- intent: not_interested
- action: utter_share_opportunity

The first message received by the conversational system from the prospect is classified as not_interested intent. Then the system takes an action and message the prospect to share the job opportunity with others. A more complicated conversation story would be like, when the recruiter sends job offer/advertisement message to the prospect and the prospect accept the offer but ask for complete job description. The conversational system send the job description and the prospect affirms it. The conversational system ask about meeting time and the prospect suggest a time slot. Conversational system confirms the meeting. The conversational system ask the prospect if she/he has other question to ask, the prospect reply with a deny message and the conversation ends with a good bye message from the system. See the story below which is taken from the dataset:

- story: Affirm and meeting scheduleed
- steps:

³https://huggingface.co/tuner007/pegasus_paraphrase

- intent: interested+job_description
- action: action_job_description
- intent: affirm
- action: utter_ask_meeting_time
- intent: meeting_schedule
- action: action_meeting_confirmed
- action: utter_anything_else
- intent: deny
- action: utter_goodbye

We have defined total of 14 conversation stories in our stories dataset which is enough for our work.

4.2.2 Experimental Setup

We use a Laptop with the following specifications for our experiments. CPU information 4.10 and RAM 4.11 information are as follows.

CPU	Intel(R) Core(TM) m3-7Y30 CPU @ 1.00GHz
Base speed:	1.61 GHz
Sockets:	1
Cores:	2
Logical processors:	4
Virtualization:	Enabled
L1 cache:	128 KB
L2 cache:	512 KB
L3 cache:	4.0 MB

Table 4.10: Local Setup: CPU Information

Memory	8.0 GB
Speed:	1600 MHz
Form factor:	Row of chips
Hardware reserved:	116 MB

Table 4.11: Local Setup: Memory Information

The software tools in 4.12 are used:

Rasa SDK Version ⁴	3.1.1
PyCharm ⁵	2022.1 (Community Edition)
Python Version ⁶	3.8.13
Operating System	Windows-10-10.0.19044-SP0
Duckling ⁷	

Table 4.12: Local Setup: Software Requirements

The conversational system is trained with these resources.

4.2.3 Results

RASA trains two models for conversational system. An NLU model for intent classification and a core model for stories in conversations. We calculated the accuracy, precision, recall, and f1-score of the model: see table 4.13.

	Accuracy	Precision	Recall	F1-score
Macro	0.99	0.99	0.98	0.98
Micro	0.99	0.99	0.99	0.99
Weighted	0.99	0.99	0.99	0.98

Table 4.13: Intent Classification: Results

The confusion matrix of intent classification model is shown in 4.2.

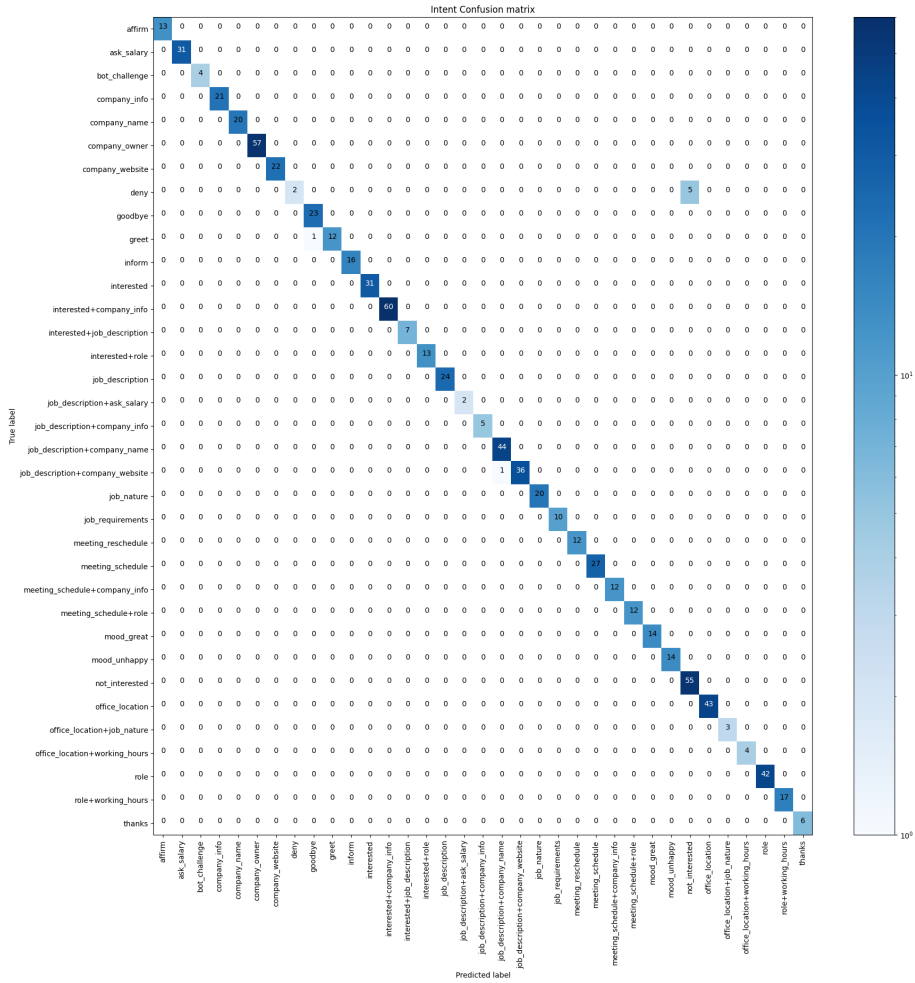


Figure 4.2: Intent Classification: Confusion Matrix

The model achieves a very good accuracy, precision, recall, and f1-score. The conversation between recruiter and prospect is often limited to the domain. This causes the problem to be relatively easier to solve for the model. Also, majority of the data is generated with machine learning which also causes the model to achieve high scores.

The core model is trained on stories. Core model also achieves better results: see table 4.14.

	Precision	Recall	F1-score
Macro	0.88	0.87	0.87
Micro	0.96	0.92	0.94
Weighted	0.94	0.92	0.92

Table 4.14: Stories: Results

The confusion matrix of the story model is shown in figure 4.3.

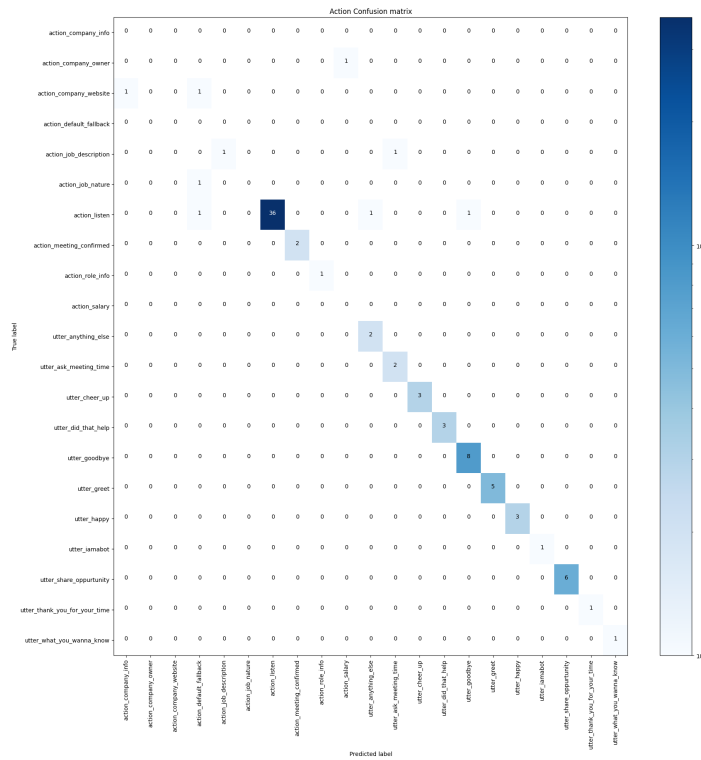


Figure 4.3: Intent Classification: Confusion Matrix

The stories models also achieves better evaluation scores as the domain is not very broader. The conversations are not very sparse and hence stories are closer to each other.

Conclusion and Future Work

This research is focused on maximizing the throughput of recruiter by automating the conversation between the candidate and recruiter by an end-to-end conversational AI system. The end-to-end conversational system is developed using RASA Open Source framework. The conversational system consists of two major components: Natural Language Understanding (NLU) and Natural Language Generation (NLG). The NLU components understands the users (candidate) messages and the NLG system responds to those messages.

This research also covers another important aspect of this system that is increasing the chance to get high quality candidates for the open position in company by measuring the quality of advertisement message.

The work done in this thesis is extensively broader where from determining the quality of a message till the scheduling meeting on Google Calendar is performed automatically. But this work can be extent to add additional functionality to the system. In our system, the first message (advertisement message) sent by the recruiter is written manually. The future work can include a system where the first message should be generated by AI system and the recruiter won't have to write it. The recruiter would just fill a form which contain the job name, company name, and other required field etc. and the system will generate custom messages for each individual prospect.

Bibliography

- [1] Egon S Pearson. “Bayes’ theorem, examined in the light of experimental sampling”. In: *Biometrika* (1925), pp. 388–442.
- [2] Hans Peter Luhn. “A statistical approach to mechanized encoding and searching of literary information”. In: *IBM Journal of research and development* 1.4 (1957), pp. 309–317.
- [3] V Vapnik and A Ya Chervonenkis. “A class of algorithms for pattern recognition learning”. In: *Automat. i Telemekh* 25.6 (1964), pp. 937–945.
- [4] Karen Sparck Jones. “A statistical interpretation of term specificity and its application in retrieval”. In: *Journal of documentation* (1972).
- [5] Gerard Salton and Christopher Buckley. “Term-weighting approaches in automatic text retrieval”. In: *Information processing & management* 24.5 (1988), pp. 513–523.
- [6] Tin Kam Ho. “Random decision forests”. In: *Proceedings of 3rd international conference on document analysis and recognition*. Vol. 1. IEEE. 1995, pp. 278–282.
- [7] David M Magerman. “Statistical decision-tree models for parsing”. In: *arXiv preprint cmp-lg/9504030* (1995).
- [8] David Goddeau et al. “A form-based dialogue manager for spoken language applications”. In: *Proceeding of Fourth International Conference on Spoken Language Processing. ICSLP’96*. Vol. 2. IEEE. 1996, pp. 701–704.
- [9] Xin Li and Dan Roth. “Learning question classifiers”. In: *COLING 2002: The 19th International Conference on Computational Linguistics*. 2002.
- [10] Marilyn A Walker, Owen C Rambow, and Monica Rogati. “Training a sentence planner for spoken dialogue using boosting”. In: *Computer Speech & Language* 16.3-4 (2002), pp. 409–433.

- [11] Minqing Hu and Bing Liu. “Mining and summarizing customer reviews”. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*. 2004, pp. 168–177.
- [12] Bo Pang and Lillian Lee. “A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts”. In: *arXiv preprint cs/0409058* (2004).
- [13] Amanda Stent, Rashmi Prasad, and Marilyn Walker. “Trainable sentence planning for complex information presentations in spoken dialog systems”. In: *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL-04)*. 2004, pp. 79–86.
- [14] Bo Pang and Lillian Lee. “Seeing Stars: Exploiting Class Relationships for Sentiment Categorization with Respect to Rating Scales”. In: *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL’05)*. Ann Arbor, Michigan: Association for Computational Linguistics, June 2005, pp. 115–124. DOI: [10.3115/1219840.1219855](https://doi.org/10.3115/1219840.1219855). URL: <https://aclanthology.org/P05-1015>.
- [15] Amanda Stent, Matthew Marge, and Mohit Singhai. “Evaluating evaluation methods for generation in the presence of variation”. In: *international conference on intelligent text processing and computational linguistics*. Springer. 2005, pp. 341–351.
- [16] Janyce Wiebe, Theresa Wilson, and Claire Cardie. “Annotating expressions of opinions and emotions in language”. In: *Language resources and evaluation* 39.2 (2005), pp. 165–210.
- [17] Fabrice Colas and Pavel Brazdil. “Comparison of SVM and some older classification algorithms in text classification tasks”. In: *IFIP International Conference on Artificial Intelligence in Theory and Practice*. Springer. 2006, pp. 169–178.
- [18] Alexander Genkin, David D Lewis, and David Madigan. “Large-scale Bayesian logistic regression for text categorization”. In: *technometrics* 49.3 (2007), pp. 291–304.
- [19] Haiyi Zhang and Di Li. “Naive Bayes text classifier”. In: *2007 IEEE International Conference on Granular Computing (GRC 2007)*. IEEE. 2007, pp. 708–708.

BIBLIOGRAPHY

- [20] Ronan Collobert and Jason Weston. “A unified architecture for natural language processing: Deep neural networks with multitask learning”. In: *Proceedings of the 25th international conference on Machine learning*. 2008, pp. 160–167.
- [21] Davis E. King. “Dlib-ml: A Machine Learning Toolkit”. In: *Journal of Machine Learning Research* 10 (2009), pp. 1755–1758.
- [22] Andrew Maas et al. “Learning word vectors for sentiment analysis”. In: *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*. 2011, pp. 142–150.
- [23] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.
- [24] Alan Ritter, Colin Cherry, and Bill Dolan. “Data-driven response generation in social media”. In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2011.
- [25] Li Deng et al. “Use of kernel deep convex networks and end-to-end learning for spoken language understanding”. In: *2012 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2012, pp. 210–215.
- [26] Eric H Huang et al. “Improving word representations via global context and multiple word prototypes”. In: *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2012, pp. 873–882.
- [27] DK Kirange and RR Deshmukh. “Emotion classification of news headlines using SVM”. In: *Asian Journal of Computer Science and Information Technology* 5.2 (2012), pp. 104–106.
- [28] Gokhan Tur et al. “Towards deeper understanding: Deep convex networks for semantic utterance classification”. In: *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE. 2012, pp. 5045–5048.
- [29] Matthew Henderson, Blaise Thomson, and Steve Young. “Deep neural network approach for the dialog state tracking challenge”. In: *Proceedings of the SIGDIAL 2013 Conference*. 2013, pp. 467–471.

BIBLIOGRAPHY

- [30] Po-Sen Huang et al. “Learning deep structured semantic models for web search using clickthrough data”. In: *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2013, pp. 2333–2338.
- [31] Khushbu Khamar. “Short text classification using kNN based on distance function”. In: *International Journal of Advanced Research in Computer and Communication Engineering* 2.4 (2013), pp. 1916–1919.
- [32] Mohammad S Khorsheed and Abdulmohsen O Al-Thubaity. “Comparative evaluation of text classification techniques using a large diverse Arabic dataset”. In: *Language resources and evaluation* 47.2 (2013), pp. 513–538.
- [33] Zhengdong Lu and Hang Li. “A deep architecture for matching short texts”. In: *Advances in neural information processing systems* 26 (2013).
- [34] Minh-Thang Luong, Richard Socher, and Christopher D Manning. “Better word representations with recursive neural networks for morphology”. In: *Proceedings of the seventeenth conference on computational natural language learning*. 2013, pp. 104–113.
- [35] Grégoire Mesnil et al. “Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding.” In: *Interspeech*. 2013, pp. 3771–3775.
- [36] Tomas Mikolov et al. “Efficient estimation of word representations in vector space”. In: *arXiv preprint arXiv:1301.3781* (2013).
- [37] Hang Ren et al. “Dialog state tracking using conditional random fields”. In: *Proceedings of the SIGDIAL 2013 Conference*. 2013, pp. 457–461.
- [38] Richard Socher et al. “Recursive deep models for semantic compositionality over a sentiment treebank”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1631–1642.
- [39] Hao Wang et al. “A dataset for research on short-text conversations”. In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 935–945.
- [40] Zhuoran Wang and Oliver Lemon. “A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed

- information”. In: *Proceedings of the SIGDIAL 2013 Conference*. 2013, pp. 423–432.
- [41] Jason Williams et al. “The Dialog State Tracking Challenge”. In: *Proceedings of the SIGDIAL 2013 Conference*. Metz, France: Association for Computational Linguistics, Aug. 2013, pp. 404–413. URL: <https://aclanthology.org/W13-4065>.
- [42] Kaisheng Yao et al. “Recurrent neural networks for language understanding.” In: *Interspeech*. 2013, pp. 2524–2528.
- [43] Steve Young et al. “Pomdp-based statistical spoken dialog systems: A review”. In: *Proceedings of the IEEE* 101.5 (2013), pp. 1160–1179.
- [44] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural machine translation by jointly learning to align and translate”. In: *arXiv preprint arXiv:1409.0473* (2014).
- [45] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A convolutional neural network for modelling sentences”. In: *arXiv preprint arXiv:1404.2188* (2014).
- [46] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1746–1751. DOI: [10.3115/v1/D14-1181](https://doi.org/10.3115/v1/D14-1181). URL: <https://aclanthology.org/D14-1181>.
- [47] Quoc Le and Tomas Mikolov. “Distributed representations of sentences and documents”. In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.
- [48] Jeffrey Pennington, Richard Socher, and Christopher D Manning. “Glove: Global vectors for word representation”. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.
- [49] Yelong Shen et al. “Learning semantic representations using convolutional neural networks for web search”. In: *Proceedings of the 23rd international conference on world wide web*. 2014, pp. 373–374.
- [50] Hiroshi Shimodaira. “Text classification using naive Bayes”. In: *Learning and Data Note 7* (2014), pp. 1–9.

- [51] Bruno Trstenjak, Sasa Mikac, and Dzenana Donko. “KNN with TF-IDF based framework for text categorization”. In: *Procedia Engineering* 69 (2014), pp. 1356–1364.
- [52] D Yann et al. “Zero-shot learning and clustering for semantic utterance classification using deep learning”. In: *International Conference on Learning Representations (cited on page 28)*. 2014.
- [53] Kaisheng Yao et al. “Spoken language understanding using long short-term memory neural networks”. In: *2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE. 2014, pp. 189–194.
- [54] Heriberto Cuayáhuitl, Simon Keizer, and Oliver Lemon. “Strategic dialogue management via deep reinforcement learning”. In: *arXiv preprint arXiv:1511.08099* (2015).
- [55] Mohit Iyyer et al. “Deep unordered composition rivals syntactic methods for text classification”. In: *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)*. 2015, pp. 1681–1691.
- [56] Pengfei Liu et al. “Multi-Timescale Long Short-Term Memory Neural Network for Modelling Sentences and Documents”. In: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sept. 2015, pp. 2326–2335. DOI: [10.18653/v1/D15-1280](https://doi.org/10.18653/v1/D15-1280). URL: <https://aclanthology.org/D15-1280>.
- [57] Nikola Mrkšić et al. “Multi-domain dialog state tracking using recurrent neural networks”. In: *arXiv preprint arXiv:1506.07190* (2015).
- [58] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. “Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks”. In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, July 2015, pp. 1556–1566. DOI: [10.3115/v1/P15-1150](https://doi.org/10.3115/v1/P15-1150). URL: <https://aclanthology.org/P15-1150>.

- [59] Tsung-Hsien Wen et al. “Semantically conditioned lstm-based natural language generation for spoken dialogue systems”. In: *arXiv preprint arXiv:1508.01745* (2015).
- [60] Xiang Zhang, Junbo Zhao, and Yann LeCun. “Character-level convolutional networks for text classification”. In: *Advances in neural information processing systems* 28 (2015).
- [61] Muhammad Bilal et al. “Sentiment classification of Roman-Urdu opinions using Naïve Bayesian, Decision Tree and KNN classification techniques”. In: *Journal of King Saud University-Computer and Information Sciences* 28.3 (2016), pp. 330–344.
- [62] ST Indra, Liza Wikarsa, and Rinaldo Turang. “Using logistic regression method to classify tweets into the selected topics”. In: *2016 international conference on advanced computer science and information systems (icacsis)*. IEEE, 2016, pp. 385–390.
- [63] Iulian Serban et al. “Building end-to-end dialogue systems using generative hierarchical neural network models”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.
- [64] Tsung-Hsien Wen et al. “A network-based end-to-end trainable task-oriented dialogue system”. In: *arXiv preprint arXiv:1604.04562* (2016).
- [65] Zichao Yang et al. “Hierarchical attention networks for document classification”. In: *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2016, pp. 1480–1489.
- [66] Hao Zhou, Minlie Huang, and Xiaoyan Zhu. “Context-aware Natural Language Generation for Spoken Dialogue Systems”. In: *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*. Osaka, Japan: The COLING 2016 Organizing Committee, Dec. 2016, pp. 2032–2041. URL: <https://aclanthology.org/C16-1191>.
- [67] Munir Ahmad, Shabib Aftab, and Iftikhar Ali. “Sentiment analysis of tweets using svm”. In: *Int. J. Comput. Appl* 177.5 (2017), pp. 25–29.
- [68] Tom Bocklisch et al. “Rasa: Open source language understanding and dialogue management”. In: *arXiv preprint arXiv:1712.05181* (2017).

- [69] Tomas Pranckevičius and Virginijus Marcinkevičius. “Comparison of naive bayes, random forest, decision tree, support vector machines, and logistic regression classifiers for text reviews classification”. In: *Baltic Journal of Modern Computing* 5.2 (2017), p. 221.
- [70] WP Ramadhan, STMT Astri Novianty, and STMT Casi Setianingsih. “Sentiment analysis using multinomial logistic regression”. In: *2017 International Conference on Control, Electronics, Renewable Energy and Communications (ICCREC)*. IEEE. 2017, pp. 46–49.
- [71] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. “Dynamic routing between capsules”. In: *Advances in neural information processing systems* 30 (2017).
- [72] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).
- [73] Zhiquan Wang and Zhiyi Qu. “Research on Web text classification algorithm based on improved CNN and SVM”. In: *2017 IEEE 17th International Conference on Communication Technology (ICCT)*. IEEE. 2017, pp. 1958–1961.
- [74] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [75] Alec Radford et al. “Improving language understanding by generative pre-training”. In: (2018).
- [76] Mujahed A Saif et al. “Classification of online toxic comments using the logistic regression and neural networks models”. In: *AIP conference proceedings*. Vol. 2048. 1. AIP Publishing LLC. 2018, p. 060011.
- [77] Muhammad Yusril Helmi Setyawan, Rolly Maulana Awangga, and Safif Rafi Efendi. “Comparison of multinomial naive Bayes algorithm and logistic regression for intent classification in chatbot”. In: *2018 International Conference on Applied Engineering (ICAE)*. IEEE. 2018, pp. 1–5.
- [78] Chen Xing et al. “Hierarchical recurrent attention network for response generation”. In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 32. 1. 2018.
- [79] Wei Zhao et al. “Investigating capsule networks with dynamic routing for text classification”. In: *arXiv preprint arXiv:1804.00538* (2018).

BIBLIOGRAPHY

- [80] Masrur Adnan, Riyanarto Sarno, and Kelly Rossa Sungkono. “Sentiment analysis of restaurant review with classification approach in the decision tree-j48 algorithm”. In: *2019 International Seminar on Application for Technology of Information and Communication (iSemantic)*. IEEE. 2019, pp. 121–126.
- [81] Fahad Mazaed Alotaibi. “Classifying text-based emotions using logistic regression”. In: (2019).
- [82] Ilias Chalkidis et al. “Large-scale multi-label text classification on EU legislation”. In: *arXiv preprint arXiv:1906.02192* (2019).
- [83] Yinhan Liu et al. “Roberta: A robustly optimized bert pretraining approach”. In: *arXiv preprint arXiv:1907.11692* (2019).
- [84] Vladimir Vlasov, Johannes EM Mosig, and Alan Nichol. “Dialogue transformers”. In: *arXiv preprint arXiv:1910.00486* (2019).
- [85] Min Yang et al. “Investigating the transferring capability of capsule networks for text classification”. In: *Neural Networks* 118 (2019), pp. 247–261.
- [86] Zhilin Yang et al. “Xlnet: Generalized autoregressive pretraining for language understanding”. In: *Advances in neural information processing systems* 32 (2019).
- [87] Tanja Bunk et al. “Diet: Lightweight language understanding for dialogue systems”. In: *arXiv preprint arXiv:2004.09936* (2020).
- [88] Md Gulzar Hussain et al. “Detection of bangla fake news using mnb and svm classifier”. In: *arXiv preprint arXiv:2005.14627* (2020).
- [89] Kanish Shah et al. “A comparative analysis of logistic regression, random forest and KNN models for the text classification”. In: *Augmented Human Research* 5.1 (2020), pp. 1–16.
- [90] Guokan Shang et al. “Speaker-change aware CRF for dialogue act classification”. In: *arXiv preprint arXiv:2004.02913* (2020).