PIPELINE CUSTOMIZATION FOR TURKISH DIALOGUE SYSTEMS

2022
MASTER THESIS
COMPUTER ENGINEERING

Abdulhameed ALHINBAZLY

Thesis Advisor
Prof. Dr. Oğuz FINDIK

PIPELINE CUSTOMIZATION FOR TURKISH DIALOGUE SYSTEMS

Abdulhameed ALHINBALY

T.C.
Karabuk University
Institute of Graduate Programs
Department of Computer Engineering
Prepared as
Master Thesis

Thesis Advisor

Prof. Dr. Oğuz FINDIK

KARABUK

August 2022

I certify that, in my opinion, the thesis submitted by Abdulhameed ALHINBAZLY titled “PIPELINE CUSTOMIZATION FOR TURKISH DIALOGUE SYSTEMS” is fully adequate in scope and in quality as a thesis for the degree of Master of Science.

Prof. Dr. Oğuz FINDIK .....
Thesis Advisor, Department of Computer Engineering

This thesis is accepted by the examining committee with a unanimous vote in the Department of Computer Engineering as a Master of Science thesis. August 9, 2022

Examining Committee Members (Institutions)	Signature
Chairman : Prof. Dr. Oğuz FINDIK (KBU)	.....
Member : Prof. Dr. Mustafa Servet KIRAN (KTUN)	.....
Member : Assist.Prof.Dr. Kasım ÖZACAR (KBU)	.....

The degree of Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Prof. Dr. Hasan SOLMAZ .....
Director of the Institute of Graduate Programs

"I declare that all the information within this thesis has been gathered and presented in accordance with academic regulations and ethical principles and I have according to the requirements of these regulations and principles cited all those which do not originate in this work as well."

Abdulhameed ALHINBAZLY

ABSTRACT

M. Sc. Thesis

PIPELINE CUSTOMIZATION FOR TURKISH DIALOGUE SYSTEMS

Abdulhameed ALHINBAZLY

Karabük University
Institute of Graduate Programs
The Department of Computer Engineering

Thesis Advisor:
Prof. Dr. Oğuz FINDIK
August 2022, 45 pages

Natural Language Understanding (NLU) is a crucial part of Dialog Systems. This module consists of a pipeline of components responsible for processing user input, extracting the features of the text, and finally, determining what the user wants to achieve by classifying the text to a predefined representation of user intent. Many NLU pipeline components were primarily developed for the English Language; other languages require a varying degree of customization based on how different the Language is from English. In this study, we customized the NLU pipeline for Turkish, the morphologically rich Language that has unique linguistic properties that are different from English, by implementing custom components specific to the Turkish Language taking advantage of Turkish NLP libraries and pre-trained word embedding models that are available in the literature; then, we conducted a series of comparative analyses of multiple NLU pipeline configurations against two main challenges in dialogue systems: the first challenge is dealing with grammatically incorrect or misspelled user input and the second challenge is the ability of the model to correctly

identify input that contains synonyms or is semantically similar to training data. The obtained results confirm the advantages of using the Turkish Language-specific components over the default ones; the results also show that dealing with Turkish at the sub-word level helps extract more valuable features from the text for better classification results, and finally, the results show the advantages of incorporating state-of-the-art pre-trained language models in the Turkish language processing pipeline to improve the dialog system's robustness to input noise and generalization to unseen data.

Key Words : Natural Language Processing, Natural Language Understanding, Turkish Language Processing, Turkish Dialogue Systems, Intent Classification, Entity Recognition, Text Tokenization, Text Representation, Word Embedding, Language Modeling.

Science Code : 92432

ÖZET

Yüksek Lisans Tezi

TÜRK DİYALOG SİSTEMLERİ İÇİN ARDIŞIK DÜZENİ ÖZELLEŞTİRME

Abdulhameed ALHINBAZLY

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı :
Prof. Dr. Oğuz FINDIK
August 2022, 45 sayfa

Doğal Dil Anlama (NLU), Dialog Systems'ın çok önemli bir parçasıdır. Bu modül, kullanıcı girdisini işlemekten, metnin özelliklerini çıkarmaktan ve son olarak, metni kullanıcı amacının önceden tanımlanmış bir temsiline göre sınıflandırarak kullanıcının ne yapmak istediğini belirlemekten sorumlu bir dizi bileşenden oluşur. Birçok NLU ardışık düzen bileşeni, öncelikle İngiliz Dili için geliştirilmiştir; diğer diller, Dilin İngilizce'den ne kadar farklı olduğuna bağlı olarak değişen derecelerde özelleştirme gerektirir. Bu çalışmada, Türkçe NLP kitaplıklarından ve önceden eğitilmiş kelime yerleştirme modellerinden yararlanarak Türkçe'ye özgü özel bileşenler uygulayarak, morfolojik olarak zengin bir dil olan ve İngilizce'den farklı benzersiz dilsel özelliklere sahip Türkçe için NLU ardışık düzenini özelleştirdik. Ardından, diyalog sistemlerindeki iki ana zorluğa karşı çoklu NLU ardışık düzen konfigürasyonlarının bir dizi karşılaştırmalı analizini gerçekleştirdik: ilk zorluk, dilbilgisi açısından yanlış veya yanlış yazılmış kullanıcı girdisiyle uğraşmak ve ikinci zorluk, modelin girdiyi doğru bir şekilde tanımlama yeteneğidir. eşanlamlıları içerirveya semantik olarak

eğitim verilerine benzer. Elde edilen sonuçlar, Türkçe'ye özgü bileşenleri varsayılan bileşenlere göre kullanmanın avantajlarını doğrulamaktadır; Sonuçlar ayrıca, Türkçe metnin alt kelime düzeyinde ele alınmasının, daha iyi sınıflandırma sonuçları için metinden daha kullanışlı özelliklerin çıkarılmasına yardımcı olduğunu göstermektedir, ve son olarak, sonuçlar, diyalog sisteminin girdi hatalarına ve görünmeyen verilere genellemeye karşı sağlamlığını geliştirmek için en gelişmiş, önceden eğitilmiş dil modellerini Türkçe dil işleme hattına dahil etmenin avantajlarını göstermektedir..

Anahtar Kelimeler : Doğal Dil İşleme, Doğal Dil Anlama, Türkçe Dil İşleme, Türk Diyalog Sistemleri, Niyet Sınıflandırma, Varlık Tanıma, Metin Tokenizasyon, Metin Temsil, Kelime Gömme, Dil Modelleme.

Bilim Kodu **:** 92432

ACKNOWLEDGMENT

I would like to thank my advisor, Prof. Dr. Oğuz FINDIK, for his invaluable advice and assistance in preparing this thesis. His knowledge and experience have motivated me to complete this study.

CONTENTS

Page

Page

Page

LIST OF FIGURES

Page

LIST OF TABLES

Page

ABBREVITIONS INDEX

ABBREVITIONS

NLP : Natural Language Processing
NLU : Natural Language Understanding
NLG : Natural Language Generation
BERT : Bidirectional Encoder Representations from Transformers
GPT-2 : Generative Pretrained Transformer 2
ELMo : Embeddings from Language Models
BOW : Bag of Words
CBOW : Continuous Bag of Words
BONG : Bag of N-Grams
TF-IDF : Term Frequency-Inverse Document Frequency
BPE : Byte Pair Encoding
BPT : Byte Pair Tokenizer
BPM : Byte Pair Embedding
WPC : Word Piece Tokenizer
DIET : Dual Intent and Entity Transformer
WS : Whitespace Tokenizer
WT : Turkish Word-Level Tokenizer
Morph : Turkish Morphological Tokenizer

PART 1

INTRODUCTION

Understanding natural human language is a long-standing goal of Artificial Intelligence. Recent advances in Natural Language Processing (NLP), supported by the evolution of complex deep learning architectures and massive processing power, allow machines a level of language understanding that is never seen before. We still have a long way to go before the machine can comprehend language at the same level as a human, but we are getting closer every day.

Natural Language Understanding (NLU), a subfield of Natural Language Processing, focuses on how machines comprehend natural language [1]. NLU focuses on teaching machines to interpret and comprehend the meaning of the language, whereas NLP typically enables machines to process and understand a substantial amount of unstructured natural language data and convert it to structured data [2].

Dialog Systems (Conversation Agents) use Natural Language Understanding to map individual user utterances to structured abstract representations of what the user wants to achieve [3]. This process is called Intent Classification, a type of Text Classification of user input into predefined classes representing the user intents [4]. The other task of NLU is called Slot Filling, which employs Entity Extraction techniques to identify essential pieces of information within the user utterance (such as places, person names, or dates ...) and pass them to the next module in the system [5]. The second module of Dialog system is Dialog Manager, which manages the conversation flow and uses the extracted information from the NLU module as parameters to query database or external web service and pass the response to the last module, the Natural Language Generation, which returns the response of the dialog system to the user query [6].

The NLU module of the dialog system consists of a pipeline of sub-tasks that process the raw text input, extract the features, classify the text, and extract the named entities.Open-source dialog system implementation frameworks such as Rasa Framework [7] allow for the configuration of the NLU pipeline and provide default components for each task in the pipeline. The default components are generic implementation and only tested and optimized for the English Language; other languages require additional customizations based on how different the Language is from English [8–10].

Turkish is an agglutinative language that heavily depends on adding suffixes to form new words from existing ones [11]. The Turkish Language's rich morphology makes the processing of Turkish text on the word level inefficient because the amount of linguistic information packed into one Turkish word equals a sentence in other languages, such as English [12].

1.1. AIM AND OBJECTIVES

The study aims to construct and train a Turkish Language-based Natural Language Understanding Pipeline that helps design Turkish dialogue systems robust to input noise and can generalize to semantically similar unseen user input.

The objectives of this study are:

- Customizing the Turkish Language Understanding Pipeline by leveraging existing Turkish NLP libraries to develop the NLU components.
- Evaluating the performance of these custom components by conducting a comparative analysis of multiple pipeline configurations against two common challenges in dialog systems: the input with incorrect spelling and the input with unknown words that has a close meaning to the words used in training.

PART 2

LITERATURE REVIEW

In this section, we first review the work related to each NLU pipeline component of the Turkish language for Intent classification and Entity Recognition tasks. Then we review the previous works on optimizing the NLU pipeline for tasks within different languages, focusing on which pipeline components are sensitive to the underlying language morphology and structure.

2.1. NATURAL LANGUAGE UNDERSTANDING OF TURKISH

There is only one study about Turkish language tokenization by Toraman et al. 2022 [12]. In this study, the authors used five tokenizers at various granularity levels, namely Character-level, BPE, WordPiece, Morphological-level, and Word-level tokenization, to compare the effectiveness of various tokenization approaches for the Turkish language. Six downstream tasks—news classification, hate speech detection, sentiment analysis, named entity recognition, semantic text similarity, and natural language inference—were used to evaluate the performance of these tokenizers. The experiments revealed that WordPiece [13] and BPE [14] are the best Turkish tokenizers, but WordPiece outperforms BPE in most tasks. It also demonstrated that the morphological-level tokenizer could compete with state-of-the-art tokenizers, followed by the word-level tokenizer. However, the character-level tokenizer provides no significant benefit.

To overcome the lack of Turkish language datasets for dialog systems, Sahinuc et al. 2020 [5] translated the popular ATIS dataset [15] to the Turkish language using automatic translation and manually reviewed the data for error correction and proper annotation. After preparing the dataset, The BERT multilingual model [16], which has

been pre-trained in over 100 languages, including Turkish, is utilized to perform intent classification and slot-filling tasks.

This study showed the success rates of the training made with this dataset since there has not been a study to create a dataset in this way before in the literature, according to the authors. For intent classification and slot filling, the micro averaged F1-score values are 89 percent and 88 percent, respectively.

As in the previous study, the same process was followed by Yilmaz et al. 2021[4], to adapt five existing English language-based datasets for the task of intent classification in dialog systems to the Turkish language by using Automatic translation. They prepared four NLU pipeline configurations to test the performance of the translated datasets for the Turkish-based Intent classification task. The baseline pipeline uses Bag-Of-Words [17] and TF-IDF [18] sparse featurizers and Support Vector Machine [19] as a classifier, and the other three pipelines used different versions of BERT pre-trained language model: TurkBert[20], which is trained on Turkish language text corpus, mBert, a multilingual model, and the original BERT which is trained on English text. The results reveal that pipelines using the BERT model outperform baseline pipelines even when trained on non-Turkish data, while the BERT model trained exclusively on Turkish corpus outperforms other BERT models.

A large-scale intent detection method for Turkish dialog systems in the banking domain was developed by Dündar et al. 2020 [21]. They compared two state-of-the-art language models, ELMO [22] and BERT [16]; the first one is trained by the authors on Turkish corpora, and the latter is the pre-trained TurkBert from HuggingFace public models, against two pipelines consisting of sparse features and classical machine learning classifiers namely: Naïve Bayes and Support Vector Machines. They collected and annotated 6453 customer messages distributed over 148 intents to evaluate these models for the task of Intent Classification. The result showed that applying text normalization to check the spelling and correct the syntactical mistakes will improve the performance of any classifier. They also concluded that the pre-trained TurkBERT language model [20] provided the best overall classification results.

To solve the lack of open-source training data for developing dialogue agents in the Turkish Language, Arslan et al. 2021 [23] introduced a comparison of Wizard-of-Oz [24] and self-play [25] data collection techniques for the Turkish task-oriented dialogue system. In the Wizard-of-Oz (WOZ) technique, where a human takes the place of a dialogue agent and helps a user to accomplish a goal through conversation, this person is expected to mimic the decision-making process of the dialog agent and give answers according to predefined logical steps, the user at the other end doesn't know that he is talking with a human, the resulting dialogue data is then recorded and saved as a training dataset. For the self-play technique, first, dialogue guides and rules are prepared. Later, these rules are filled with actual phrases by crowd works, and the resulting natural dialogues are then saved as a training dataset. In addition to the above two datasets, A third one is generated using data augmentation techniques; Data augmentation is a technique used in data analysis to increase the amount of data by either adding extra copies of existing ones or by producing new synthetic data. They assess the performances using slot filling scores and intent classification. They employed RASA, an open-source language understanding and dialogue management software, together with the DIET classifier [26], which is an architecture for dialog act and slot recognition based on Transformers, to do these tasks. they apply 5-fold cross-validation to each data set and employ FastText [27] pre-trained word embeddings as Turkish word vectors.

2.2. NATURAL LANGUAGE UNDERSTANDING OF NON-ENGLISH LANGUAGES

In the study by Khan et al. 2021 [9], the authors optimized NLU Pipeline components for a Bangla language-based conversational agent by conducting a comparative analysis of 8 different pipeline configurations. They employed a custom-developed tokenizer for The Bangla language. The results show that although their custom tokenizer performs better than the default word-level tokenizer, However, the pipeline with the BERT-based tokenizer and the BERT Pre-Trained feature extractor delivers the best results overall.

For the Vietnamese language based NLU Pipeline, Nguyen et al. 2021 [8], compared three different pipeline configurations and developed a custom Vietnamese language tokenizer. The results show that the pipeline with the custom tokenizer along with a FastText-based dense feature performed better than the state-of-the-art BERT-based dense feature for the Vietnamese language.

In another study by Hwang et al. 2021 [10], the authors proposed a Korean language NLU pipeline optimization method that includes a Korean-based Tokenizer, bag-of-words sparse feature extractor, and DIET Transformer-based classifier. They performed a comparative analysis with another three sparse feature-based pipeline configurations, showing that the proposed pipeline has the best performance. However, the study did not employ dense feature extractors, so it is unknown how the proposed pipeline performs compared to state-of-the-art dense feature extractors.

PART 3

THEORETICAL BACKGROUND

In this chapter, we present theoretical concepts and text processing techniques related to Natural Language Understanding.

3.1. COMPUTATIONAL LINGUISTICS

Linguistics is the science that studies how Languages work and suggests the analysis of the language on multiple levels [28,29] :

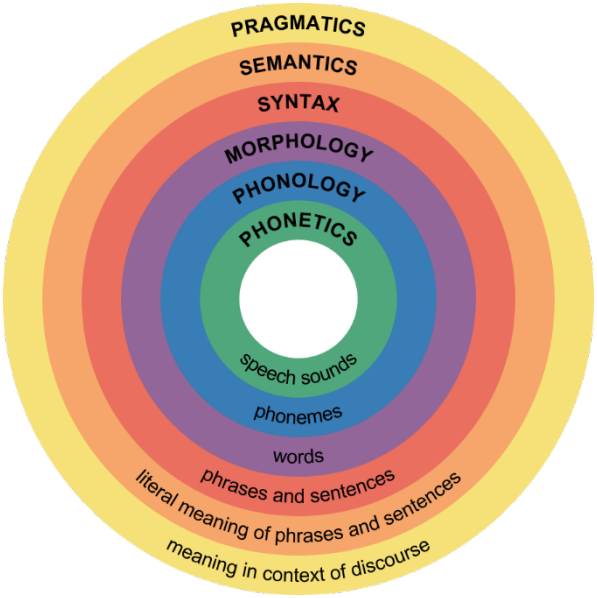

Figure 3.1. Major levels of linguistic structure [28]

- Phonetics: The study of the basic sounds that humans produce and perceive.
- Phonology: How does language organize these sounds on sub-word level
- Morphology: How the words form and relate to each other.
- Syntax: How words are combined together to form phrases and sentence

- Semantics: The study of the meaning of the words
- Pragmatics: How the context affects the meaning.

Computational linguistics is the use of computers in language analysis and processing [29]. Many text-processing techniques allow for language processing up to the syntactic level; In contrast, higher levels of language processing (Semantics and Pragmatics) remain challenging; Natural Language Understanding involves processing the language on semantic and pragmatic levels to extract the correct meaning of the text [1].

3.2. TEXT PROCESSING AND REPRESENTATION:

3.2.1. Text Tokenization

Tokenization is the first step in every text processing task and the first component of the NLU pipeline [30]. Tokenizer splits the input text into tokens, the token is usually a word, but it can be a sub-word [31] or even a character [32]. The total number of unique tokens used to train deep learning models is referred to as the vocabulary size; creating vocabularies that represent the text is the goal of Tokenization [12]. There are many types of Tokenization based on the type of the tokens:

3.2.1.1. Word-Level Tokenization

This method is the most used type of tokenization [30]. It splits the text into individual tokens based on the spaces between the words. Word-level tokenization is rule-based and requires no pre-training. Word-Level tokenization is easy and intuitive, but it has two significant drawbacks [12] :

- When applied to a large text corpus, it produces a huge vocabulary size, and this is a problem since most deep learning models have a limited size of vocabulary, so this tokenizer is not efficient.
- The model trained on a word-level tokenizer easily failed to deal with OOV (Out-Of-Vocabulary) problem when faced with new words during testing time.

3.2.1.2. Character-Level Tokenization

This tokenization method splits the text into its most minor parts, the characters; The vocabulary size of this tokenizer is the number of characters in each language; The advantage of this tokenization is that it can be used with any language, and the model that uses it won't have the OOV issue [32]. However, the major disadvantage of this tokenization is that it failed to present the relations between characters. For example, the characters t and h are frequently encountered in English, and ignoring such valuable information negatively affects the performance of the statistical model [12].

3.2.1.3. Morphological Tokenization

This tokenization method depends on the language-specific grammatical rules to break down the words into linguistic units, such as the word stem and suffixes [11].
This method is specifically applicable to morphologically rich languages such as Turkish, where new words are formed by adding suffixes to a root verb, and the resulting word equals a sentence in other languages [33]. For example, the word “gelemedim” in Turkish means “I could not come” in English, so using a word tokenizer will result in a single token representing a complete sentence.
However, when applying a morphological tokenizer, this word produces the following tokens [gel, em, e, di, m] and denotes the following equivalent English tokens [come, cannot, “past tense suffix,” I].

Although this tokenization method produces linguistically meaningful tokens, however, one disadvantage of this tokenizer is that some lengthy word stems cannot be split further, and this result in less reusable tokens [12], Also, this tokenization method suffers from ambiguity since there may be more than one morphological interpretation for the same word [34].

3.2.1.4. Subword Tokenization

Subword Tokenization is considered a mid-level between character and word-level tokenization method [35]. This tokenization method solves the problems associated with previous tokenization methods; It handles the OOV problem by depending on subwords so any new words can be broken into parts, and these parts are most likely to be found in the vocabulary [13]. It also handles the problem of morphological and character-level tokenizers by forming subwords from the most frequent combinations of characters [36]. Because of these advantages, state-of-the-art language models such as BERT [16] or GPT2 [37] employ this tokenization method. Unlike previous rule-based methods, this method requires training, and there are two main subword tokenization algorithms:

Byte Pair Encoding (BPE)

Byte Pair Encoding (BPE) [38] is a sub-word tokenization algorithm that needs to be trained on a corpus of the language to be tokenized. In this method, the first step is extracting the unique words in the training corpus, and the second step is constructing a base vocabulary from all symbols occurring in the unique words. The last step is building the final vocabulary by merging the symbols that occur together to form a list of the most frequent unique subwords [36].

Word Piece

Word Piece algorithm [13], like BPE, is also used to train a sub-word tokenizer. It works by finding the higher probable combination of symbols (sub-words) that maximize the score for the language model when tested with words not presented in their training data. This tokenization method is frequently used to train BERT language models [39].

3.2.2. Text Representation

Tokenization deconstructed text into simple tokens; still, these tokens are represented by symbols and need to be converted into numerical form to be understandable by machines [40]. The feature extracting methods convert tokens into a vector representation, so they can be processed by statistical methods to extract their properties (or features), find semantic relations between them, and later prepare them as an input to train machine learning or neural networks models to classify them according to the required task [41].

There are two types of text feature representation: Sparse and Dense [42].

3.2.2.1. Sparse Features

These features are represented by sparse vectors in which most elements are zeros. These vectors typically result from hot encoding [41], a bag of words [43], or TF-IDF [44] feature extraction algorithms.

One Hot Encoding

One-Hot Encoding is the simplest vector representation method that represents each word with a vector that has the dimension of the total number of words in the vocabulary set, the represented word has a specific index with the value of 1, and other elements are zeros [41].

Bag Of Words (BOW)

Bag-Of-Words is another sparse vector representation that is like hot-encoding. However, in this case, the represented word has a value according to the number of occurrences in the training corpus [43].

Term Frequency-Inverse Document Frequency (TF-IDF)

TF-IDF is one of the most popular sparse vector representations, and it depends on the Bag-Of-Words representation to calculate the frequencies of all words. The basic idea of TF-IDF is to give a higher score to less frequent words in the document and a lower score or zero to more frequent words; this scoring helps the classification model [44].

Sparse features have the following cons and pros [41] [45] [46]:

Cons:

- Sparsity makes them computationally inefficient because sparse vectors allocate memory for zeroes that don't carry any useful information.
- They are discrete representations treating each token separately, ignoring the linguistic and semantic relation between them.
- The Curse of Dimensionality As the size of the data grows, the sparsity problem worsens since vectors must represent more data, so their dimensions become large.
- Out-Of-Vocabulary, the model will not be able to produce a feature vector for words that do not exist in the training data.
- Intolerance to noise, one of the problems of sparse features is their dependence on the surface form of the tokens, which are usually words, so any change to the words because of mistyping, which is common in dialog systems, is not going to be recognized and considered out-of-vocabulary.

Pros:

- When we want to train our model from scratch on our own domain-specific data when there are no pre-trained dense features available, these features are the only way to train our classification model.
- They contain a lot of zeros; these zeroes are considered a form of regulation for neural models that help reduce the overfitting problem.

- Since linguistic and context information is not considered, Sparse features are independent of the language they represent.
- Some of the problems associated with these features, like intolerance to noise, can be partially solved by using an n-gram-based bag-of-words Featurizer so that when there is a mistyping of a few characters, some parts of the word could survive, and the classification model still be able to recognize the wrong typed words.

2.2.2.2. Dense Features

Each word in these features is represented by a dense vector derived from a pre-trained word embedding or language model; this vector represents the word in a space that contains all the words in the dataset (which is typically a huge dataset, like a Wikipedia corpus), on which the embedding model is trained [47].

Word Embeddings:

Word2vec is the earliest word embedding technique [48]. word2vec embeddings are the weights resulting from training a single-layer neural network, using either Continuous Bag OF Words (CBOW), where the model trained on predicting a center word giving its context as an input, or Skip-Gram, where the model trained on predicting the context given the center word as an input. Figure 3.2.

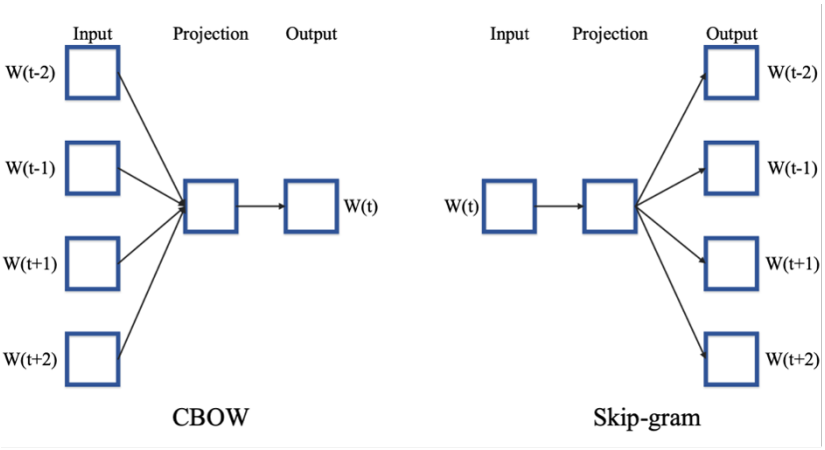

Figure 3.2. Word2Vec Algorithms

The original word2vec algorithm ignores the morphology of words by assigning vectors to each word in the training data [49]. To solve this issue, [31] proposed a novel subword level embedding technique called FastText, based on the skip-gram model but represents each word as a bag of character n-grams.
Byte pair embedding [35] is another sub-word embedding technique that assigns vectors to subword tokens by applying the greedy Byte pair encoding tokenization algorithm.

Language Models

Language models compute the probability distribution across word sequences corresponding to a language's distribution. BERT [16] or GPT2 [37]
Language modeling, including BERT [16], ELMo [22], and GPT-2 [37], forecasts the likelihood that a word will appear in a particular context, the model trains to learn how to infer the missing words by using information from the entire sentence.

Word Embedding VS Language Models

Language models have the following advantages over word embedding [50] [51]:

- Word embeddings fail to capture the different meanings of the word based on its context; language models are trained to capture the different meanings of the same word because they are trained to predict the words given many different contexts as input.
- Word embeddings are trained by a shallow neural networks model (typically a single hidden layer), which only incorporates previous knowledge from the first layer. They also keep the weights representing the word vectors, but the model is discarded. This is not the case in language models that are trained on deep models such as transformers and incorporate way more context knowledge from the surrounding words; also, these models, along with their vectors, are kept after training.

The dense features have the following Pros and Cons [43] [51]:

Pros:

- Dimensionality reduction, Dense vector dimensions are not related to vocabulary size. They are compressed vectors that try to capture as much word information in as few elements as possible.
- Words with close meaning have vectors that are close to each other, so even if the user uses synonyms or even different words than the training data, the model will still be able to predict the correct meaning.
- Context information is incorporated in the vector representation, so the meaning of the word is derived from the surrounding words, so the words that appear in similar contexts have closer vectors.

Cons:

- Training our won word embedding is a time-consuming task; thankfully, there are many pre-trained embeddings available for many languages; still, there is another problem finding an embedding that has domain-specific vocabularies since most of these embeddings are trained on generic or neutral corpus such as Wikipedia corpus.

PART 4

METHODOLOGY

4.1. RASA FRAMEWORK

Rasa is an open-source, python-based, machine learning framework for developing artificial conversational agents [7]. It has a modular and flexible design that enables developers to customize and extend its functionalities.

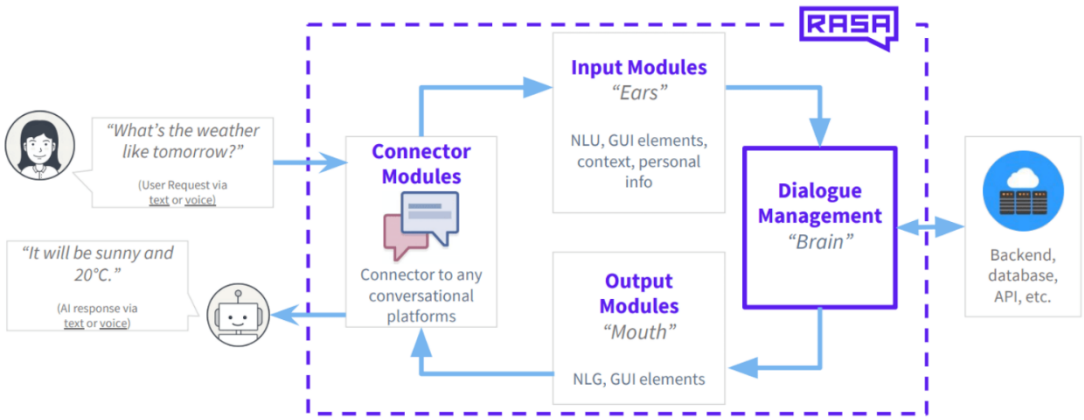

Figure 4.1. RASA Framework Architecture Overview [52]

The following two primary independent parts make up the Rasa Framework [53]:

Rasa NLU is the default library for natural language understanding in the Rasa framework. It attempts to train the chatbot to comprehend the user's message by using statistical methods from Natural Language Processing (NLP) and Machine Learning (ML). Entity extraction and intent classification are the two essential tasks that the library performs on the user input data.

Rasa Core is an ML-based library responsible for dialog management. It attempts to predict the best subsequent action that the chatbot has to take based on a variety of parameters such as the understood user message and the state of the conversation, including its historical actions.

4.1.1. RASA NLU Overview

The main function of the Rasa NLU module is the extraction of structured information in the form of intents and entities from unstructured user's text messages. Intent refers to the goal that the user wants to achieve through his input message in natural language, and entities are the extracted chunks of structured information which help the chatbot to respond correctly [53] [7].

4.1.1.1. Intent Classification

Rasa NLU typically uses a step-by-step machine learning method for intent classification. First, the input message text is tokenized into a bag of words, and the tokens are then given word vector representations through any language-specific pre-trained word vector library. The word vector representations are then fed into a multiclass classifier like the Support Vector Machine (SVM), which labels the user input with a particular intent per the predictive model's confidence score. If no pre-trained dense feature is provided, Rasa NLU will perform supervised learning by feeding the sparse features to the classifier and learning from the training data, and this is useful when we want to train the chatbot from scratch using our domain-specific data [52].

4.1.1.2. Entity Extraction

The entity extraction and intent classification procedure both operate simultaneously after tokenizing the input message and assigning each token a grammatical part of speech. These tokens are then passed to a chunker, which helps train the model to parse multi-worded entities (such as addresses or book titles), and semantically annotates

these chunks using Named-Entity Recognition (NER), with the end result being labeled extracted entities [52].

4.1.1.3. NLU Pipeline Configuration Format:

Rasa supports many pipelines for entity extraction, intent classification, and response selection. Additionally, rasa supports a wide variety of already-trained language models, like BERT [16], GPT-2 [37], and Spacy [54]; Developers can also create their unique components. The configuration file specifies the components that the model will employ to generate predictions depending on input from users, and they are defined in the config.yml file, as in the following example:

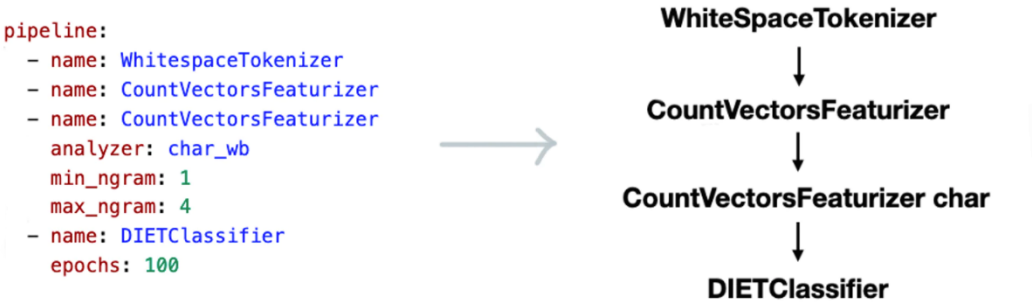

Figure 4.2. RASA NLU Pipeline Configuration Example

In this example, we defined a pipeline configuration by assigning actual component implementation for each NLU task, and we can also specify parameters and hyper-parameters if required. For the tokenizer, we specified WhitespaceTokenizer, which is a rule-based word-level tokenizer. For feature extraction, we defined two types of CountVectorsFeaturizer [55], one that implements a bag-of-word and the other that implements a bag-of-n-gram. We can use the features of both models to train our classifier, which is, in this case, DIET Classifier.

The Dual Intent and Entity Transformer (DIET) [26] is a transformer architecture that can perform both entity recognition and intent classification at the same time. It is specifically designed for conversational agent systems, it has a modular design that

allows experimenting and exchanging different components, and it is around six times faster to train than other state-of-the-art transformer architecture like BERT.

4.2. DATASET

Even though intent classification is a type of text classification, training a dialog system differs from training a text classification model because the dialog system must deal with user-formulated sentences to express a particular goal, whereas, in other text classification, we typically train the models on large bodies of text like articles, product reviews, or tweets [56]. This makes collecting the data for the dialog system is more challenging task and requires human intervention and review to make sure the data is making sense [57]. We have already reviewed previous works on creating datasets for Turkish dialog systems [23]. However, the resulting datasets have few issues not with the quality of their phrases but in the distribution of the examples over intent classes.

For performing the experiments in this work, we build a small talk dataset. Small talk means casual conversation and contains phrases that people usually use when communicating to start a conversation and express their feeling [58].

The reasons for selecting the small talk topic are:

- This study is for evaluating natural language understanding models for Turkish-based chatbots in general, so the scope of the study is domain neutral and should not have any domain-specific vocabularies or acronyms that could affect the universality of the results.
- Small talk functionality is a must-have feature of every AI-based chatbot, and it can greatly improve the user experience because users often start a conversation with a chatbot by greeting or asking general questions, and chatbots should handle these types of conversation gracefully and help users formulate their request by guiding them to asking the right questions [59].

The samples of the dataset are written from scratch to guarantee the examples' quality and the results' accuracy. We selected 12 intent categories, and when it comes to the

ideal number of utterances, there are no rules on the number of required examples per intent because it depends on multiple factors such as the complexity of the chatbot and the domain-specific requirements. Variations in key terms and utterance length were also employed. Also, variations in grammar and punctuation were used, and the main words were positioned differently throughout the utterance.
Consequently, the dataset we made has 250+ sample utterances divided into 12 intents, and we have annotated 63 named entities under three entity types.

4.2.1. Incorrect Spelling Test Data

To evaluate the trained model performance against misspelling errors, we prepared a test data consisting of five examples for each intent and containing the following types of spelling errors that frequently happened during writing [60]:

- Orthographical mistakes, Turkish contains seven diacritic characters consisting of (ç, ı, İ, ğ, ö, ş, ü); these characters are replaced with their ASCII equivalents (c, i, l, g, o, s, u).
- Sound-based mistakes, such as when a single consonant is confused with another single consonant. This problem happened in Turkish as mistakes in constant mutation. For example (kitapı, cevapi instead of kitabı, cevabı).
- Rule-based mistakes, such as errors in suffixes. In the Turkish language, this happens when the suffixes are not written correctly. For example, (istiyom, gidiyo).
- Using chat shortcuts. For example (mrb, nber, slm).
- Writing two words as one. For example (edebilirmiyim, bahsedermisin).
- Randomly omitting characters or adding unnecessary characters.

4.3. TURKISH NLU PIPELINE CUSTOMIZATION

The NLU pipeline is composed of machine learning components that are responsible for processing the user text, extracting the features, and training an intent classifier and entity extractor [52]. Since we are using the open-source Rasa NLU pipeline, we have

a wide range of choices for pipeline components [53]. In the following sections, We customize the NLU Pipeline components for the Turkish language.

4.3.1. Tokenizer Component

Turkish is an agglutinative language in which many new words are generated by adding suffixes to the end of root words [11], so the tokenization method of the Turkish language largely impacts the performance of subsequent feature extraction and classification models [12]. In this study, we tested four types of tokenization algorithms based on the type of tokens necessary for the subsequent components in the pipeline. The following is the discussion of each tokenizer.

4.3.1.1. Default Whitespace Tokenizer

Rasa framework comes with a default word-level tokenizer component called Whitespace Tokenizer [61]. This tokenizer is rule-based and requires no training, and it employs regular expressions to split the text based on the spaces between the words and also performs text cleaning by removing the punctuation marks. Applying this tokenizer to a Turkish text gives the following result:

Table 4.1. Whitespace Tokenizer Example

Input	tamam, yarın arkadaşları ile gelecek
Output	['tamam', 'yarın', 'arkadaşları', 'ile', 'gelecek']

4.3.1.2. Custom Turkish Word Tokenizer

A significant issue for dialog systems in production is receiving wrong-spelled, grammatically incorrect, or slang input from the user. For this reason, we developed an enhanced word-level tokenizer component for the Turkish language. This tokenizer uses the Zemberek Turkish NLP library [62] for word tokenization. Since it is a custom component, we utilized Turkish NLP techniques and added the following useful functionalities:

- Spell and vowel harmony correction and diacritics restoration. This type of input correction is important for the model for a correct prediction since providing incorrect input will result in unknown tokens that don't have a pre-trained vector that represents them.
- Restoring Turkish character encoding. Unfortunately, most programming libraries deal with Turkish letters I, İ as the English i, and this result in an encoding issue that renders affected Turkish words unrecognizable.

Table 4.2. Custom Turkish Word Tokenizer Example

Input	tmm, yrn arkadşları ıla gelcek
Whitespace Tokenizer Output	['tmm', 'yrn', 'arkadşları', 'ıla', 'gelcek']
Custom Turkish Word Tokenizer Output	['tamam', 'yarın', 'arkadaşları', 'ile', 'gelecek']

4.3.1.3. Custom Turkish Morphological Tokenizer

In a previous work [12], a morphological level tokenizer for the Turkish language was adopted and compared with other types of tokenizers. The study found that its performance was competitive with other state-of-the-art subword tokenization techniques. When we attempted to test this tokenizer, we found no ready-to-use morphological tokenizer for the Turkish language. However, the authors of the mentioned study hinted that they used the Zemberek library morphological analyzer [62], parsed its output, and extracted the word morphological parts as tokens.

Table 4.3. shows an example of the mentioned morphological analyzer in action:

Table 4.3. Turkish Morphological Tokenizer Example

Input	veremedim
Analyzer Output	[vermek:Verb] ver:Verb+eme:Unable+di:Past+m:A1sg
Tokenizer Output	['ver','eme','di','m']

The analyzer outputs the infinitive form of the verb [vermek] in Turkish, the inability suffix "eme" and the past tense suffix "di" and the first-person pronoun "m".

We implemented our version of the Turkish morphological tokenizer by extending the functionality of our custom Turkish word tokenizer. We get the word tokens output, apply a morphological analysis for every single word token, parse the analysis results, and extract the individual word parts as tokens.

However, the morphological analysis is not that straightforward for some words, for example: "akşamlar"

Table 4.4. Morphological Analysis of the word “akşamlar”

	Morphological Analyses	**Possible Tokens**
First Analysis	[akşamlamak:Verb] akşamla:Verb+r:Aor+A3sg	['akşamla','r']
Second Analysis	[akşamlamak:Verb] akşamla:Verb|r:AorPart→Adj	['akşamla','r']
Third Analysis	[akşam:Noun, Time] akşam:Noun+lar:A3pl	['akşam','lar']

The morphological analyzer provides all the possible morphological analyses for the word. In this example, we have three analyses for the input word "akşamlar "

- The first analysis means that the word is the verb "akşamla" + the suffix of the present tense (geniş zaman)+ Third Person Singular (inferred by the absence of other personal plural pronouns)
- The second analysis adds another piece of information to the analysis of the first line, which states that the verb + aorist suffix can be used as an adjective in the Turkish language. (such as geçer zaman = passing time)
- The third analysis recognizes "akşam" as a noun that represents the time + the plural suffix (-lar) [the most likely analysis]

So that there are two possible tokenization outputs out of this word:
['akşamla','r']

['akşama','lar']

In the Zemberek library [62], applying morphological analysis to a word generate an object of the type WordAnalysis; each Word analysis object has a list of SingleAnalysis objects, each one representing a different analysis of the same word. In our tokenizer implementation, we select one SingleAnalysis output for each word token because we cannot generate more than one tokenizer output for the same input sentence.

4.3.1.4. Custom Turkish Subword Tokenizer

Unlike other tokenizer types, these tokenizers need training. We trained two subword tokenizers, namely the BPE tokenizer and Word Piece tokenizer, on our dataset. To improve generalization, we also added a Turkish Wikipedia dumb [63] to the training data.

4.3.2. Featurizer Component

As we discussed later in the background section, we have two main types of features, Sparse and Dense, thus we have two types of feature extraction components:

4.3.2.1. Sparse Featurizer

In the following experiments, we employed a sparse feature extraction component provided by the Rasa framework called CountVectorFeaturizer. The CountVectorFeaturizer uses Sklearn's CountVectorizer [55] to generate a bag-of-words representation of the user's message. This feature extractor can be configured to generate sparse vectors for either the token or the character n-grams. By default, the component generates a bag-of-word representation by counting the number of occurrences of each token in the training data; the tokens are based on the type of the used tokenizer. For example, if a word-level tokenizer is used, then a vector representing the frequency of the word is generated for each token, and when a sub-word-level tokenizer has employed, a vector representing the frequency of the sub-

word token is generated. The other configuration for this component allows calculating the frequencies of n-grams within the token boundaries; this configuration requires two parameters that set the lower and upper limit of the range of n-values for different character n-grams to be extracted [64].

4.3.2.2. Dense Featurizers

Dense feature extractors require a pre-trained word embedding or language model; hence the choice of the type of dense features is subject to the availability of such a pre-trained model for the Turkish Language since training a model from scratch is a time-consuming task. Luckily, we have many publicly available pre-trained models for the Turkish Language [27] [65] [20]; some of them are from the word embedding official repository of pre-trained models that are ready for download and use; these models are trained on a large amount of data and are ready to use in our NLU pipeline. The table below shows a summary of the three Turkish pre-trained models that are used in this study.

Table 4.5. Dense Features Models

Dense Feature Extraction Component	**Model Type**	**Model Information**
FastTextFeaturizer	n-grams Embedding	This is the official FastText [27] trained model for Turkish. It was trained using CBOW with position-weights of 300, character n-grams of 5, a window of 5, and 10 negatives on Wikipedia. Model Size: 4.3 GB.
BytePairFeaturizer	Subword Embedding	Byte Pair Embedding [65]pre-trained Turkish subword embeddings, based on Byte-Pair

		Encoding (BPE) and trained on Wikipedia. Model Size: 11 MB.
LanguageModelFeaturizer (BERT)	Language model	We used BERTurk [20], a community-driven BERT model for Turkish. It was trained on the Turkish Wikipedia and OSCAR corpus with a total text data size of 35 GB. Model Size: 545 MB.

4.3. TURKISH NLU PIPELINE CONFIGURATION OPTIONS

The second objective of this study is to evaluate the performance of the above custom components by conducting a comparative analysis of multiple pipeline configurations against incorrectly spelled user input and input with synonyms that are not present in the training data. The pipeline configurations are listed in the following table:

Table 4.6. Turkish NLU Pipeline Components Options

Tokenizers	**Sparse Featurizers**	**Dense Featurizer**
Whitespace Tokenizer	CountVectorFeaturizer	FastText Featurizer
Turkish Word Tokenizer	(Bag of Words)	BytePair Featurizer
Turkish Morph Tokenizer	CountVectorFeaturizer	Language Model
Turkish BPE Tokenizer	(Bag of n-grams)	Featurizer
Turkish WPC Tokenizer		

We will use DIET Classifier for intent classification and Entity Extraction for all the experiments. Since only the tokenizer and feature extractor components depend on the language, the classifier only receives the resulting vectors, so there is no need to test multiple classifiers.

4.4. PIPELINE PERFORMANCE EVALUATION

Precision, recall, and F1 scores, common performance and accuracy metrics for classification, were computed for the evaluation.
The following coincidence matrix table shows the correlation between the expected and actual values.

Table 4.7. Coincidence Matrix

	ACTUAL POSITIVE	**ACTUAL NEGATIVE**
PREDICTED POSITIVE	True positive (TP)	False positive (FP)
PREDICTED NEGATIVE	False negative (FN)	True negative (TN)

The actual values are the ones determined by the user's intent, while the predicted values are the replies the chatbot provides. False negatives happen when the chatbot cannot forecast the intent, while false positives happen when it believes it has recognized the intent correctly but is incorrect. Precision, recall, and F1-score were computed based on these classifications, allowing for comparison of the NLU pipelines.

4.4.1. Precision

The precision of a classifier is its ability not to classify a negative sample as positive. Precision is the result of dividing the number of true positives (TP) by the total number of predicted positives, which is made up of the true positives (TP) and false positives (FP). The precision value is between 0, the worst, and 1, the highest.

Precision = TP / (TP + FP)

4.4.2. Recall

The recall, also called sensitivity, is the classifier's capacity to locate all the positives. It is calculated by dividing the number of true positives (TP) by the number of actual

positives, which is made up of the true positives (TP) and false negatives (FN). Recall also returns a value between 0, the minimum, and 1, where 1 is the maximum.

Recall = TP / (TP + FN)

4.4.3 F1-Score

The F1-score, which is a harmonic mean between precision and recall and indicates that they both contribute equally to the score, can be calculated using the following formula:

F1-Score = 2 x (precision x recall) / (precision + recall)

The F1-score is always a number between 0 and 1, with 1 being the best result that can be achieved.

PART 5

RESULTS DISCUSSIONS AND CONCLUSION

5.1. TESTING NLU PIPELINE ROBUSTNESS

In the following series of experiments, we trained NLU models based on pipeline configurations shown in the tables below. The tables are grouped by the type of the extracted features, and each row represents an NLU pipeline consisting of a Tokenizer and Featurizer components. And for all pipelines, we used the DIET classifier for both intent classification and entity extraction tasks. After training, we tested each trained model on examples introduced in the training data but with different spelling errors.

Table 5.1. Bag-Of-Words Sparse Features Pipelines Robustness Results

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + BOW	0.493	0.367	0.352	0.868	0.526	0.61
WT + BOW	0.969	0.966	0.966	**1.0**	**0.947**	**0.97**
Morph + BOW	**0.986**	**0.983**	**0.983**	0.876	0.909	0.871
BPT + BOW	0.666	0.666	0.635	0.511	0.791	0.592
WPC + BOW	0.592	0.666	0.603	0.518	0.711	0.6

Table 5.2. Bag-of-n-grams Sparse Features Pipelines Robustness Results

Pipeline	Intent Classification Results			Entity Extraction Results		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + BONG	0.968	0.95	0.95	**1.0**	0.842	0.911
WT + BONG	**1.0**	**1.0**	**1.0**	**1.0**	0.9	**0.945**
Morph + BONG	**1.0**	**1.0**	**1.0**	0.863	**1.0**	0.92
BPT + BONG	0.91	0.866	0.856	0.463	0.729	0.552
WPC + BONG	0.885	0.833	0.826	0.617	0.692	0.641

The first and second groups of experiments (Table 5.1. and Table 5.2) employed the five types of tokenizers with two types of sparse features; The first type of sparse features came from Bag-of-Words (BOW) sparse featurizer that converts the unique tokens from word or sub-word tokenizers into sparse vectors, keeping the tokens intact. The second type of sparse features came from applying Bag-of-n-grams (BONG), which assign sparse vectors to n-gram of characters (specified in the settings between 2 and 5 within the word boundaries), which means that regardless of the input tokens resulting from the tokenizer, the specified number of n-grams are assigned sparse vectors. The sparse pipelines' results clearly show the advantage of implementing text normalization and spelling correction in Turkish Word and Morphological Tokenizers; these tokenizers were able to revert the misspelled sentences into their correct format. The results also show that using n-gram sparse vectors significantly improved the results of all pipelines, especially the word level tokenizers, which means that we can bring the advantages of using sub-word level tokenizers to word level ones by using sub-word level feature extraction.

Table 5.3. Dense Features Pipelines Robustness Results

Pipeline	Intent Classification Results			Entity Extraction Results		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + FT	0.85	0.799	0.788	0.887	0.684	0.74
WT + FT	**0.976**	**0.966**	**0.967**	**0.95**	**0.85**	**0.895**
Morph + FT	0.9	0.883	0.873	0.796	0.818	0.789

BPM	0.5	0.416	0.37	0.605	0.157	0.247
BERT	0.864	0.816	0.794	0.947	0.842	0.881

In the third group of experiments (Table 5.3.), we replaced sparse features with dense features from Turkish pre-trained word embedding or language models. We also matched tokenizers with suitable models to convert their tokens into dense vectors. The word level tokenizers can only be matched by the FastText model because this is the only model that has vectors for word tokens; we also used the FastText model for the morphological tokenizer because FastText model is trained on n-grams, so we have a high probability of finding vectors that match morphological tokens, Byte Pair sub-word embedding, and BERT language model do not need separate tokenizers because they can assign dense vectors to the matching sub-words directly from word tokens. Dense features can only be helpful for incorrect spelling if their pre-trained models are trained on incorrect spelling data and can map the vectors representing those misspelled words to their correct form. The results of dense features pipelines show varying degrees of success in handling the misspelling errors per model if we disregard the tokenizers that use spelling correction. BERT model, for example, has better results than other models because BERT is trained on contextual information to assign vectors to words, meaning that misspelled word context information can help assign the misspelled word dense vectors close to their correct forms. FastText model's huge size was also helpful in mapping misspelled words to correct ones, whereas the small-size Byte Pair embedding model was the worst model to handle misspelling errors.

Table 5.4. Sparse and Dense Features Pipelines Robustness Results

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + BONG + FT	0.967	0.966	0.967	**1.0**	0.684	0.798
WT + BONG + FT	**1.0**	**1.0**	**1.0**	**1.0**	**0.9**	**0.945**

Morph + BONG + FT	**1.0**	**1.0**	**1.0**	0.787	0.863	0.818
BONG + BPM	0.917	0.866	0.852	0.736	0.21	0.324
BONG + BERT	0.972	0.966	0.964	**1.0**	0.894	0.941

In the final group of experiments (Table 5.4.), we took advantage of an important feature of the DIET classifier: the capacity to learn from both sparse and dense features. We selected bag-of-n-gram sparse features for their better performance in previous tests and the same dense feature models from the last experiment. Internally DIET classifier learns from training fast-forward neural networks on the sparse feature, and the resulting sparse vectors of this training, along with dense vectors from the pre-trained models, are then passed to the transformer layers, as can be seen in Figure 5.1; this means that each word (more specifically, its sub-parts) in training has both a sparse vector and dense vectors to represent it, this gives the classifier more features per words for better training. The obtained results confirm the advantages of this approach by scores for FastText and BERT models that are nearly equivalent to the pipelines that use additional spell correction functionality, which means that combining sub-word level sparse features with pre-trained dense feature produce models more robust to spelling mistakes, without any spell checking or text normalization preprocessing.

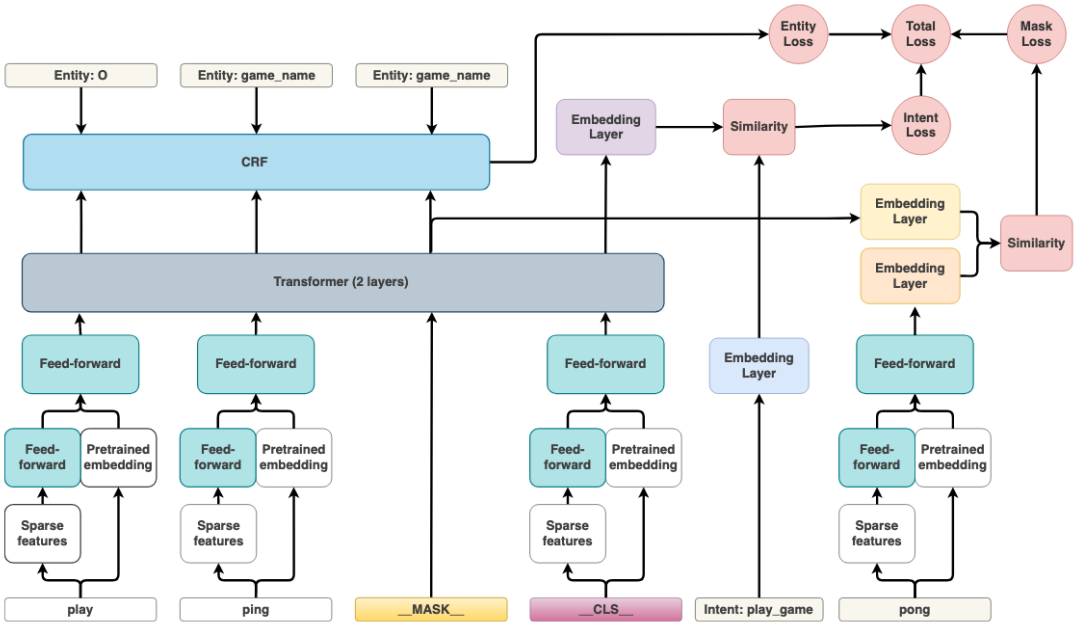

Figure 5.1. The Architecture of DIET Classifier [26]

5.2. TESTING NLU PIPELINE GENERALIZATION

Cross Validation is a resampling technique that uses various data subsets to test and train a model over a number of iterations. We performed a 5-fold cross-validation test for multiple NLU pipeline configurations in the following experiments. Five folds mean that for each run, we split the dataset into 80% train / 20% test sets, then train the model on the 80% subset and test it on the remaining 20% subset. This process repeats five times; each time, a trained model is created, tested, and discarded. The average evaluation metrics are then calculated for all runs. The test set of each run has samples that had similar meanings to training samples. This comprehensive test help evaluate the model generalization to unseen data.

The results of cross-validation tests are presented in the following tables, grouped by the type of extracted features as in the experiments of the previous section.

Table 5.5. Bag-Of-Words Pipelines Cross-Validation Results

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + BOW	0.866	0.858	0.858	0.661	0.615	0.636
WT + BOW	0.873	0.858	0.861	0.746	0.707	0.712
Morph + BOW	0.883	0.882	0.881	0.73	0.565	0.634
BPT + BOW	**0.922**	**0.919**	**0.918**	0.808	0.735	0.768
WPC + BOW	0.91	0.906	0.906	**0.825**	**0.748**	**0.783**

Table 5.6. Bag-of-n-grams Pipelines Cross-Validation Results

Pipeline	Intent Classification Results			Entity Extraction Results		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + BONG	0.922	0.919	0.919	0.83	0.5	0.62
WT + BONG	**0.932**	**0.927**	**0.927**	**0.844**	0.476	0.593
Morph + BONG	0.921	0.919	0.918	0.636	0.493	0.55
BPT + BONG	0.93	**0.927**	**0.927**	0.731	**0.617**	**0.65**
WPC + BONG	0.931	**0.927**	0.926	0.709	0.549	0.586

The results of the two types of sparse features presented in Table 5.5 and Table 5.6 above show that sub-token level bag-of-n-grams sparse features generally provide better generalization results for the intent classification. In contrast, the token-level bag-of-words sparse features have better results for the entity classification task. We also notice that the difference in the results of bag-of-n-grams pipelines is minimal and nearly has the same performance metrics, which means that exclusively using bag-of-n-grams sparse features voids the effect of the tokenizer; whereas the effect of the tokenizers is more pronounced when using bag-of-words sparse features, where it is clear that sub-word level tokenizers have the advantage over word-level ones when considering the performance in both intent classification and entity extraction tasks, since sparse features, in general, depend on how similar the words in test samples are to their train sample counterparts, and sub-words have a better chance of similarity between test and train samples.

Table 5.7. Dense Pipelines Cross-Validation Results

Pipeline	Intent Classification Results			Entity Extraction Results		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + FT	0.945	0.943	0.943	0.936	**0.892**	0.912
WT + FT	0.914	0.91	0.91	0.915	0.861	0.887
Morph + FT	0.812	0.813	0.811	0.839	0.701	0.754

PBM	0.853	0.842	0.844	**0.983**	0.83	0.898
BERT	**0.96**	**0.959**	**0.959**	0.97	0.876	**0.913**

The results of cross-validating dense feature pipeline models are shown in Table 5.7. In contrast to sparse features, which depend on the words (or sub-words) surface-level similarity, dense features represent similar words by close vectors. Hence, the performance score indicates the pre-trained model's capacity to capture semantically similar words between train and test samples. The first thing to notice in the results is the difference between whitespace and custom Turkish word tokenizers; they should have similar performance, given that the data is free from spelling errors. Digging deeper into the logs of the custom tokenizer, we noticed that sometimes applying spell correction to correct words could replace the word with another with a similar shape but a different meaning. For example, the Turkish word "kaçlısın" which is a different way of asking "how old are you?" than the most common "kaç yaşndasın," is wrongly converted to "kaçışın" which means "escape." These errors prove that spell checking is like morphological analysis; it is not perfect and is prone to ambiguous interpretations. The spell-checking problem is also presented in the morphological tokenizer, which suffers from the errors of morphological analysis since we only select a single random analysis from a set of usually multiple possible analyses for a single word. BERT-based dense features language model, which is unlike other traditional word embeddings techniques, takes context information into account to determine similarity produced the best overall classification results, it is followed closely by FastText n-gram embedding, and the lowest performance came from Byte Pair sub-word embedding.

Table 5.8. Spare and Dense Features Pipelines Cross-Validation Results

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
WS + SPARSE + FT	0.932	0.931	0.93	**1.0**	**0.861**	**0.923**
WT + SPARSE + FT	0.938	0.935	0.935	0.959	0.815	0.878

Morph + SPARSE + FT	0.936	0.935	0.934	0.822	0.723	0.761
SPARSE + BPM	0.939	0.939	0.939	0.983	0.769	0.858
SPARSE + BERT	**0.964**	**0.963**	**0.963**	0.985	0.784	0.858

In the final series of experiments, we tested pipelines with sparse and dense features in an attempt to take advantage of both features; since sparse features depend on word surface similarity, and dense features depend on vector distance to determine the semantic relatedness between train and test words. The results are shown in Table 5.8; the sparse features used in these pipelines came from bag-of-words and bag-of-n-grams featurizers because, unlike misspelling experiments, no sparse featurizer outperformed the other in cross-validation tests, so we used features from both. The results show that some pipelines (the third and the fourth rows in the table above) got better results by combining dense and sparse features, while others did not improve. The reason is that the improved pipelines have tokens not matched by vectors from the pre-trained model in the previous experiment, so they are not used in training the classifier. This experiment solved this problem by assigning sparse features for these tokens that do not have dense vectors, and the result of their respective pipelines got better.

5.3. COMPARING RESULTS WITH LITERATURE

In this section, we compared the results obtained by testing multiple pipeline configurations against existing results from previous work [23] that was reviewed in the Literature review section. This work followed two data collection methods, Wizard-Of-OZ and Self Play, to create two datasets for Turkish-based dialogue systems.

The authors of the study performed a 5-fold cross-validation using a pipeline consisting of FastText [27] pre-trained dense features and DIET classifier [26], so to determine which pipeline could enhance the reported results, we reviewed the dataset and found that it contains a lot of non-Turkish words such as restaurants' names and locations these words has a low probability of being matched by dense vectors from pre-trained word-embeddings model, also based on findings from previous experiments we proved that using Language models allows incorporating context information that improves the classification results.

Table 5.9. WOZ_TR Dataset Results Optimization

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
FT (Original)	-	-	0.84	-	-	0.77
SPARSE + FT	0.925	0.923	0.924	0.778	0.806	0.791
BERT	0.929	0.930	**0.93**	0.808	0.793	**0.799**

Table 5.10. Self_Play_TR Dataset Results Optimaization

Pipeline	**Intent Classification Results**			**Entity Extraction Results**		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score
FT (Original)	-	-	0.82	-	-	**0.89**
SPARSE + FT	0.93	0.928	0.929	0.857	0.873	0.864
BERT	0.935	0.935	**0.935**	0.868	0.887	0.876

The obtained results (Tables 5.9, 5.10) confirmed our observations. The first proposed pipeline got better results since new words that don't have matching vectors can be

assigned sparse vectors, whereas, in the original pipeline, those words are assigned just a single generic vector for all unknown words. The second proposed pipeline also provides better results because language models have better performance than word embedding.

5.4. SUMMARY

The Turkish language is an agglutinative language with a rich morphological structure. This study presented a processing pipeline that considers the language qualities while designing a natural language understanding module for a Turkish language-based dialog system. The implemented pipeline is tested extensively for robustness against common spelling mistakes and the generalization to semantically similar unseen data. For incorrect spelling errors, we showed that this problem could be handled either by adding text correction functionality or by employing sub-word level tokenizers and a pre-trained language model; where the tokenizer can reduce the effect of errors by utilizing the correct parts of the misspelled word, and the pre-trained language model can use context information to understand the meaning of the word regardless of its surface form. For generalization, we performed cross-validation for multiple pipeline configurations to select the best-performing pipeline. The results clearly show the advantage of using a pre-trained language model over word embedding to find semantically similar words since language models incorporate context information to determine the correct meaning of the words, while traditional word embeddings lack this information and sometimes it failed to assign dense vectors to tokens and need a sparse vector to fill the gap.

REFERENCES

1. Navigli, R., "Natural Language Understanding: Instructions for (Present and Future) Use", ***Proceedings Of The Twenty-Seventh International Joint Conference On Intelligence, IJCAI-18***, 5697–5702 (2018).

2. Valin, R. D. van, "From NLP to NLU", (2016).

3. Galitsky, B., "Chatbot Components and Architectures", Developing Enterprise Chatbots, ***Springer International Publishing***, 13–51 (2019).

4. Yilmaz, E. H. and Toraman, C., "Intent Classification based on Deep Learning Language Model in Turkish Dialog Systems", ***2021 29th Signal Processing And Communications Applications Conference (SIU)***, 1–4 (2021).

5. Sahinuc, F., Yucesoy, V., and Koc, A., "Intent Classification and Slot Filling for Turkish Dialogue Systems", ***2020 28th Signal Processing And Communications Applications Conference, SIU 2020 - Proceedings***, (2020).

6. Adamopoulou Eleni and Moussiades, L., "An Overview of Chatbot Technology", ***Artificial Intelligence Applications And Innovations***, 373–383 (2020).

7. Bocklisch, T., Faulkner, J., Pawlowski, N., and Nichol, A., "Rasa: Open Source Language Understanding and Dialogue Management", ***ArXiv Preprint ArXiv:1712.05181***, (2017).

8. Nguyen, T. and Shcherbakov, M., "Enhancing Rasa NLU model for Vietnamese chatbot Proactive Decision Support Systems Design View project", ***International Journal Of Open Information Technologies***, 9 (1): 33–36 (2021).

9. Khan, F. S., Mushabbir, M. al, Irbaz, M. S., and Nasim, M. A. al, "End-to-End Natural Language Understanding Pipeline for Bangla Conversational Agents", ***2021 20th IEEE International Conference On Machine Learning And Applications (ICMLA)***, 205–210 (2021).

10. Hwang, M. H., Shin, J., Seo, H., Im, J. S., and Cho, H., "KoRASA: Pipeline Optimization for Open-Source Korean Natural Language Understanding Framework Based on Deep Learning", ***Mobile Information Systems***, 2021: (2021).

11. Oflazer, K., "Morphological Processing for Turkish", Turkish Natural Language Processing, ***Springer International Publishing***, Cham, 21–52 (2018).

12. Toraman, C., Yilmaz, E. H., Şahinuç, F., and Ozcelik, O., "Impact of Tokenization on Language Models: An Analysis for Turkish", ***ArXiv Preprint ArXiv:2204.08832v1***, (2022).

13. Schuster, M. and Nakajima, K., "Japanese and Korean voice search", ***2012 IEEE International Conference On Acoustics, Speech And Signal Processing (ICASSP)***, 5149–5152 (2012).

14. Heinzerling, B. and Strube, M., "BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages", ***Proceedings Of The Eleventh International Conference On Language Resources And Evaluation ({LREC} 2018)***, (2018).

15. Hemphill, C. T., Godfrey, J. J., and Doddington, G. R., "The ATIS Spoken Language Systems Pilot Corpus", ***Proceedings Of The Workshop On Speech And Natural Language***, 96–101 (1990).

16. Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K., "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding", ***Proceedings Of The 2019 Conference Of The North American Chapter Of The Association For Computational Linguistics: Human Language Technologies***, 1: 4171–4186 (2019).

17. Zhang, Y., Jin, R., and Zhou, Z.-H., "Understanding bag-of-words model: a statistical framework", ***International Journal Of Machine Learning And Cybernetics***, 1 (1): 43–52 (2010).

18. Havrlant, L. and Kreinovich, V., "A simple probabilistic explanation of term frequency-inverse document frequency (tf-idf) heuristic (and variations motivated by this explanation)", ***International Journal Of General Systems***, 46 (1): 27–36 (2017).

19. Noble, W. S., "What is a support vector machine?", ***Nature Biotechnology***, 24 (12): 1565–1567 (2006).

20. "Dbmdz/Bert-Base-Turkish-Uncased · Hugging Face", **https://huggingface.co/dbmdz/bert-base-turkish-uncased** (2022).

21. Dündar, E. B., Kiliç, O. F., Çekiç, T., Manav, Y., and Deniz, O., "Large scale intent detection in turkish short sentences with contextual word embeddings", ***IC3K 2020 - Proceedings Of The 12th International Joint Conference On Knowledge Discovery, Knowledge Engineering And Knowledge Management***, 1: 187–192 (2020).

22. Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L., "Deep contextualized word representations", ***Proceedings Of The 2018 Conference Of The North American Chapter Of The Association For Computational Linguistics: Human Language Technologies***, 1: 2227–2237 (2018).

23. Arslan, D. and Eryigit, G., "Evaluation of wizard-of-Oz and self-play data collection techniques for turkish goal-oriented dialogue agents", ***2021 International Conference On INnovations In Intelligent SysTems And Applications, INISTA 2021 - Proceedings***, (2021).

24. Okamoto, M., Yang, Y., and Ishida, T., "Wizard of Oz Method for Learning Dialog Agents", ***Cooperative Information Agents V***, 20–25 (2001).

25. Shah, P., Hakkani-Tür, D., Tür, G., Rastogi, A., Bapna, A., Nayak, N., and Heck, L., "Building a Conversational Agent Overnight with Dialogue Self-Play", ***ArXiv Preprint ArXiv:1801.04871***, (2018).

26. Bunk, T., Varshneya, D., Vlasov, V., and Nichol, A., "DIET: Lightweight Language Understanding for Dialogue Systems", ***ArXiv Preprint ArXiv:2004.09936*** , (2020).

27. "Word Vectors for 157 Languages · FastText", **https://fasttext.cc/docs/en/crawl-vectors.html** (2022).

28. Scott-Phillips, T., "Pragmatics and the aims of language evolution", ***Psychonomic Bulletin & Review***, 24: (2016).

29. Hausser, R., "Foundations of computational linguistics: Human-computer communication in natural language, third edition", ***Foundations Of Computational Linguistics: Human-Computer Communication In Natural Language, Third Edition***, 1–518 (2014).

30. Webster, J. J. and Kit, C., "Tokenization as the Initial Phase in NLP", (1992).

31. Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T., "Enriching Word Vectors with Subword Information", ***Transactions Of The Association For Computational Linguistics***, 5: 135–146 (2016).

32. McNamee, P. and Mayfield, J., "Character N-Gram Tokenization for European Language Text Retrieval", ***Information Retrieval***, 7 (1): 73–97 (2004).

33. Baykara, B. and Güngör, T., "Abstractive text summarization and new large-scale datasets for agglutinative languages Turkish and Hungarian", ***Language Resources And Evaluation***, 56 (3): 973–1007 (2022).

34. Külekci, M. O., "Turkish Word Segmentation Using Morphological Analyzer", ***Eurospeech.2001***, 1053–1056 (2001).

35. Heinzerling, B. and Strube, M., "BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages", ***Proceedings Of The Eleventh International Conference On Language Resources And Evaluation ({LREC} 2018)***, (2018).

36. Sennrich, R., Haddow, B., and Birch, A., "Neural Machine Translation of Rare Words with Subword Units", ***Proceedings Of The 54th Annual Meeting Of The Association For Computational Linguistics***, 1: 1715–1725 (2016).

37. Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I., "Language Models are Unsupervised Multitask Learners", (2019).

38. Shibata, Y., Kida, T., Fukamachi, S., Takeda, M., Shinohara, A., Shinohara, T., and Arikawa, S., "Byte pair encoding: a text compression scheme that accelerates pattern matching", (1999).

39. Muller, B., Sagot, B., Seddah, D., and Muller BenoˆıtBenoˆıt Sagot Djamé Seddah, B., "Enhancing BERT for Lexical Normalization", ***Proceedings Of The 5th Workshop On Noisy User-Generated Text (W-NUT 2019)***, 297–306 (2019).

40. Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., and Tan, S., "Between words and characters: A Brief History of Open-Vocabulary Modeling and Tokenization in NLP", ***ArXiv Preprint ArXiv:2112.10508***, (2021).

41. Jurafsky, D. and Martin, J. H., "Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition", 1st. Ed., ***Prentice Hall PTR***, USA, (2000).

42. Luan, Y., Eisenstein, J., Toutanova, K., and Collins, M., "Sparse, Dense, and Attentional Representations for Text Retrieval", ***Transactions Of The Association For Computational Linguistics***, 9: 329–345 (2021).

43. Kowsari, K., Meimandi, K. J., Heidarysafa, M., Mendu, S., Barnes, L., and Brown, D., "Text classification algorithms: A survey", ***Information (Switzerland)***, 10 (4): (2019).

44. Salton, G. and Buckley, C., "Term-weighting approaches in automatic text retrieval", ***Information Processing & Management***, 24 (5): 513–523 (1988).

45. Drikvandi, R. and Lawal, O., "Sparse Principal Component Analysis for Natural Language Processing", ***Annals Of Data Science***, (2020).

46. Li, X., Wang, Y., and Ruiz, R., "A Survey on Sparse Learning Models for Feature Selection", ***IEEE Transactions On Cybernetics***, 52 (3): 1642–1660 (2022).

47. Levy, O. and Goldberg, Y., "Neural Word Embedding as Implicit Matrix Factorization", ***Advances In Neural Information Processing Systems***, 27: 2177–2185 (2014).

48. Mikolov, T., Chen, K., Corrado, G., and Dean, J., "Efficient Estimation of Word Representations in Vector Space", ***1st International Conference On Learning Representations, {ICLR} 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings***, (2013).

49. Cotterell, R. and Schütze, H., "Morphological Word Embeddings", ***The Proceedings Of NAACL 2015 (Denver, June).*** , (2019).

50. Gomez-Perez Jose Manuel and Denaux, R. and G.-S. A., "Understanding Word Embeddings and Language Models", A Practical Guide to Hybrid Natural Language Processing: Combining Neural Models and Knowledge Graphs for NLP, ***Springer International Publishing***, Cham, 17–31 (2020).

51. Almeida, F. and Xexéo, G., "Word Embeddings: A Survey", ***ArXiv Preprint ArXiv:1901.09069***, (2019).

52. Bagchi, M., "Conceptualising a Library Chatbot using Open Source Conversational AI", ***DESIDOC Journal Of Library & Information Technology***, 40: 329–333 (2020).

53. Sharma, R., "An Analytical Study and Review of open source Chatbot framework, Rasa", ***International Journal Of Engineering Research & Technology (IJERT)***, V9 (6): 1011–1014 (2020).

54. Honnibal, M. and Johnson, M., "An Improved Non-monotonic Transition System for Dependency Parsing", ***Proceedings Of The 2015 Conference On Empirical Methods In Natural Language Processing***, 1373–1378 (2015).

55. "Sklearn.Feature_extraction.Text.CountVectorizer — Scikit-Learn 1.1.1 Documentation", **https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html** (2022).

56. Schuurmans, J. and Frasincar, F., "Intent Classification for Dialogue Utterances", ***IEEE Intelligent Systems***, 35 (1): 82–88 (2020).

57. Larson, S., Mahendran, A., Peper, J. J., Clarke, C., Lee, A., Hill, P., Kummerfeld, J. K., Leach, K., Laurenzano, M. A., Tang, L., and Mars, J., "An Evaluation Dataset for Intent Classification and Out-of-Scope Prediction", ***ArXiv Preprint ArXiv:1909.02027v1***, 1311–1316 (2019).

58. Coupland, J., "Small talk: Social functions", ***Research On Language And Social Interaction***, 36 (1): 1–6 (2003).

59. Babel, F., Kraus, J., Miller, L., Kraus, M., Wagner, N., Minker, W., and Baumann, M., "Small Talk with a Robot? The Impact of Dialog Content, Talk Initiative, and Gaze Behavior of a Social Robot on Trust, Acceptance, and Proximity", ***International Journal Of Social Robotics***, 13 (6): 1485–1498 (2021).

60. Elliott, G. and Johnson, N., "All the right letters-just not necessarily in the right order. Spelling errors in a sample of GCSE English scripts", ***The British Educational Research Association Conference, Edinburgh, September 2008***, (2008).

61. Rafla, A. and Kennington, C., "Incrementalizing RASA's Open-Source Natural Language Understanding Pipeline", ***ArXiv Preprint ArXiv:1907.05403v1***, (2019).

62. Afşın Akın, A. and Dündar Akın, M., "Zemberek, an open source NLP framework for Turkic Languages", 1–5 (2007).

63. "Turkish Wikipedia Dump | Kaggle", **https://www.kaggle.com/datasets/mustfkeskin/turkish-wikipedia-dump** (2022).

64. KANARIS, I., KANARIS, K., HOUVARDAS, I., and STAMATATOS, E., "WORDS VERSUS CHARACTER N-GRAMS FOR ANTI-SPAM FILTERING", ***International Journal On Artificial Intelligence Tools***, 16 (06): 1047–1067 (2007).

65. "BPEmb", **https://bpemb.h-its.org/** (2022).

RESUME

Abdulhameed ALHINBAZLY graduated from Aleppo university in 2006 with a bachelor’s degree in computer engineering. He worked for over 10 years as a software developer in Saudi Arabia. In 2017 he moved to Turkey and studied master's degree in Computer Engineering at Karabük University.