IDENTIFYING EVENT NUGGETS IN TURKISH NEWS TEXTS USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING METHODS

by

Mehmet Durna

B.S., Computer Engineering, Boğaziçi University, 2014

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University
2019

ACKNOWLEDGEMENTS

It has been a very long and adventurous journey for me, but now I am finally here, writing this acknowledgement part for my thesis, and this is a thrilling moment indeed. There may be only one person's name written on the front page of a thesis but in fact it takes the support of the whole family, circle of friends and many academicians to successfully finish a thesis. Today, I am grateful to all those people who took part in supporting me.

First of all, I would like to express my special appreciation to my thesis supervisor Associate Professor Dr. Arzucan Özgür for being an incredible support and guide through this journey, who introduced me to the wonders of Natural Language Processing and gave me the reason to complete my bachelors in Computer Engineering and even to specialize in the same field, being a complete turning point in my life.

I would also like to thank the members of my committee, Professor Dr. Tunga Güngör, with whom I shared so many great lectures, and Assistant Professor Dr. Reyhan Aydoğan who literally helped me to code for the first time in my life in the CMPE150 course and now will be here in my thesis committee for the latest leg of my academic journey, for kindly accepting to be my committee members and share this final moment in my MS years.

I would like to dedicate this thesis to my family, Yılmaz Durna, Sevilay Durna and Şahander K. who gave me the motivation to make changes in life and the support to keep on. I would like to also thank all my extended family, my uncles, my aunt, my cousins, their spouses and my nephews and nieces. I am grateful that I could share this moment with them all. I wish I could also have shared this joy with my dearest grand-aunts Mine Kara and Sabahat Kazaz and my grandfather Mustafa K. who all have taught me so much in life.

Last but not least, I am deeply thankful to my friends who have assisted me and encouraged me throughout my thesis. I would like to express my thanks to Gökhan Hınçal and Ekin Aydın who did not necessarily make things easier but they for sure made it better , Amer Maraqa who believed in me and was always here for me from the very beginning, Alina Lotfullina who always inspired and encouraged me to do more and better, Oğuzhan Murat Çakmak who gave me the motivation when I needed from very far, Dr. Derya Soydaner who gave me valuable insights for my thesis and Assistant Professor M. Şükrü Kuran who helped to get back on my thesis and restart.

ABSTRACT

IDENTIFYING EVENT NUGGETS IN TURKISH NEWS TEXTS USING NATURAL LANGUAGE PROCESSING AND MACHINE LEARNING METHODS

An event nugget is the smallest textual instance that marks the existence of an event. Detecting event nuggets in a given text opens door to further research and many practical applications such as automatic classification of the events within a given text. Therefore, it has been studied extensively for some languages including English, Spanish and Chinese. In this thesis, event nugget detection and event type classification for Turkish are studied for the first time. Due to lack of annotated data for event nugget detection in Turkish, we developed a new annotated data set for this task. In this thesis we describe how we manually annotated our data set as well as our system to identify event nuggets in Turkish news texts.

The data set consists of words from Turkish news texts. Each word in the data set is manually annotated in terms of sequence type, nugget type, realis value and whether the event nugget is the main event, thus enabling us to make analysis on this data set for event nugget detection, event type classification, realis classification and main event detection. We made use of language specific features like morphological features and dependency parser features in Turkish as well as some other features. We aimed to see the effect of language specific features on this kind of analysis. We also experimented with different machine learning algorithms to find the best fitting model for our tasks. After having completed our experiments, we have shown that Turkish specific morphological features, dependency tree related features as well as word embeddings enabled us to achieve better results.

ÖZET

DOĞAL DİL İŞLEMESİ VE MAKİNE ÖĞRENMESİ YÖNTEMLERİ İLE TÜRKÇE HABERLERDE OLAY GÖSTERGESİ TESPİTİ

Metinde bir olayın geçtiğini gösteren en küçük kelime grubuna olay göstergesi denmektedir. Şu ana kadar İngilizce, İspanyolca ve Çince gibi dillerde çalışılmıs olan olay göstergelerinin tespiti bize daha çok araştırma yapılmasını ve bir metinde otomatik olay sınıflandırması gibi birçok uygulamayı mümkün kılmaktadır. Bu tez Türkçe için ilk olay göstergesi tespiti ve olay göstergesi türü sınıflandırması çalışmasını sunmaktadır. Olay göstergesi tespiti için Türkçe bir veri kaynağımız olmadığından dolayı bu problem için kendimiz bir veri kümesi oluşturduk. Bu tezde veri kümesini nasıl etiketlediğimizi ve Türkçe haber metinleri için olay göstergesi tespiti ve olay göstergesi türü sınıflandırması yapan sistemimizi tanıttık.

Veri kaynağı çeşitli Türkçe haber sitelerinden alınmış haberlerin içindeki kelimelerden oluşmaktadır. Veri kümesindeki her kelime elle dizi türü, gösterge türü, gösterge alt türü, realis değeri ve ana olay olup olmamasına göre işaretlenmiştir. Böylelikle bu veri kümesi olay göstergesi tespiti, olay türü sınıflandırması, realis değeri ve ana olay sınıflandırması çalışması yapmamıza olanak sağlamaktadır. Bu işaretlenmiş veri kümesi üzerinde çeşitli sınıflandırma metotları denedik. Türkçe'ye özgü morfolojik ve bağlılık ayrıştırma özelliklerinin yanı sıra diğer özelliklerden de yararlandık. Bunu yaparken dile özgü özelliklerin sınıflandırmada nasıl bir etkisi olduğunu görmeyi amaçladık. Farklı makine öğrenmesi algoritmalarını kullanarak olay göstergesi tespiti, olay göstergesi türü sınıflandırması, realis değeri bulma ve ana olay tespiti için en başarılı modeli bulmaya çalıştık. Çalışmamızın sonunda Türkçe'ye özgü morfolojik ve bağlılık ayrıştırma özelliklerinin ve kelime temsillerinin sonuçlarımızı iyileştirdiğini gözlemledik.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

DCS	Dice coefficient score
g	Sigmoid activation function of a logistic regression model
GL	List of gold standard mentions
h	Logistic regression hypothesis expectation
J	State transition matrix of a hidden Markov model
M	Mapping of dice scores with event mentions
T_G	Token of an event mention in gold standard mention list
T_S	Token of an event mention in system mention list
x	Feature of a data instance
y	Label of a data instance
z	Output of a net input function
β	Best parameters to make to decision boundary in logistic regression

LIST OF ACRONYMS/ABBREVIATIONS

AI	Artificial Intelligence
BRNN	Bidirectional Recurrent Neural Networks
CNN	Convolutional Neural Networks
CRF	Conditional Random Field
DSC	Dice Similarity Coefficient
ESA	Explicit Semantic Analysis
KBP	Knowledge Base Population
Maxent	Maximum Entropy
ML	Machine Learning
NCNN	Non-consecutive Convolutional Neural Networks
NER	Named Entity Recognition
NLP	Natural Language Processing
OvR	One versus the Rest
prec	Precision
RNN	Recurrent Neural Networks
Smote	Synthetic Minority Over-sampling Technique
SRL	Semantic Role Labeling
SVM	Support Vector Machine
TAC	Text Analysis Conference
XGBOOST	Extreme Gradient Boosting

1. INTRODUCTION

With the advancement of technology, our reach has expanded to the globe. Now if we want to deliver a message to the other side of the world, to the people we do not know in person, this can be done in seconds. Our messages can even be heard by people we do not know at all. We are not limited to the news that are in newspapers sold in the nearby kiosk anymore, we can reach the published news on the other side of the world, moreover unpublished unofficial news are within our reach through social media.

This ease of getting information and news has enabled and even pushed us to look for more news sources to get the news from different perspectives. The abundance of news seems to have grabbed attention of the commercial world. A lot of different applications were made and are still being made to gather news and display on screen.

Moreover, the fact that gathering news from many different sources so that some analyses and research can be made became appealing to the scientific world as well. The field seemed very promising with numerous Natural Language Processing (NLP) applications possible. One of these applications, and a challenging one is to understand why the news is written. What is the main reason that this news exists? What events are mentioned in the news? What kind of events are they? Did the events mentioned actually occur?

In order to detect what the news are about, we need to identify the events that are mentioned in the news text. Also, to understand what kind of events they are, we need to find a way to classify the events. When events are successfully classified in the way that we desire, numerous other applications will be possible. Imagine you would like to gather all dates of public meetings of two politicians in a specific place between given years. When the events inside the news are successfully classified, that the event is about a meeting and that the event actually took place, the aforementioned problem will be easier and faster to solve. As far as new machine learning solutions for different

problems coming out with the advancement of modern processors are concerned, one of the most studied problems stands out as event nugget detection.

There are plenty of data for Event Nugget Detection and Classification tasks in English, mostly provided by National Institute of Standards and Technology (NIST). NIST had been hosting event nugget detection and event nugget type classification in their Text Analysis Conference (TAC) until 2017.

However for the Turkish Language, there does not exist an already available data set for this task. Also, to the best of our knowledge, this task has not been studied for Turkish news before. In order to work on this task further, we had to create our own data set. We have manually annotated news texts for different categories.

Our research focuses on the detection of event nuggets that are mentioned in a given Turkish news text and the classification of the detected event nuggets. We also attempt to identify the main event of the news, i.e which event is the reason that the news is written. In addition, we also attempt to detect whether a given event has actually happened. The nature of the study is highly complex, since we will deal with natural language processing. Also the resources and the available libraries for Turkish are significantly less than those for English.

This thesis has the following two main contributions.

(i) A data set of Turkish news texts has been manually annotated for event nuggets.
(ii) By using different machine learning models and Turkish-specific features such as morphological features, a baseline for event nugget detection for Turkish news text has been formed for future research. Our results show that using Turkish-specific morphological and dependency parser features, as well as word embeddings significantly increase event nugget detection performance.

1.1. Terminology

In order to go further with the topic, some terminology should be introduced so that we can talk about the topic further in detail and avoid any confusion that may occur.

The first word to be defined is an *event*. Events are things that take place involving different entities like people and objects [1]. Events can also be a change of a state in a given place and time [2]. An example of a simple event is given below.

Example 1.1.1. *Bombalama** sabah gerçekleşti.
Translation: The *bombing** took place in the morning.

Event mention is the occurrence of an event in a text. Documents can have numerous event mentions which refer to a single event [3].

Event nugget can be defined as the smallest textual instance that gives away the occurrence of an event [4]. An event nugget that contains only one word is sometimes called as an *event trigger*. In the example above, the event nugget, also called the event trigger in this case, is italic and marked with a star.

An event nugget can contain one or more words. An example of an event nugget that consists of more than one word is given below. Event nuggets can be verbs, nouns, or adjectives [1].

Example 1.1.2. Saatlerce *esir* tutuldular**.
Translation: Several people were *held* hostage** for hours.

A sentence can have more than one event nugget as well as a document may have multiple event nuggets that refer to the same event. In the example below, two different event mentions can be seen. The event nuggets of the events are italic and marked with a star.

Example 1.1.3. *Bombalamadan** sonra saatlerce *esir* tutuldular**.
Translation: Several people were *held* hostage** for hours after the *shooting**.

The event nuggets may be categorized depending on the meaning they embody, i.e. their event type. The event types also can be classified in many ways, however we will use the classification stated in [5]. An event nugget can belong to nine main types and 38 different subtypes [5]. The possible event types and subtypes are given in Table 1.1 [6].

Table 1.1. Event and Subevent Types.

Events	**Subevents**
Life Events	be-born , marry , divorce , injure , die
Movement Events	transport-person , transport-artifact
Business Events	start-org, merge-org, declare-bankruptcy, end-org
Conflict Events	attack, demonstrate
Contact Events	meet, correspondence, broadcast, contact
Personnel Events	start-position, end-position, nominate, elect
Transaction Events	transfer-ownership, transfer-money, transaction
Justice Events	arrest-jail, release-parole, trial-hearing, charge-indict, sue, convict, sentence, fine, execute, extradite, acquit, appeal, pardon
Manufacture	artifact

Another way to classify event nuggets is their realis value. It shows whether the event actually happened. It can be of three types as stated by Mitamura and Hovy [5].

- ACTUAL : Event nuggets whose events actually took place.
- GENERIC : Event nuggets without a specific time and/or place.
- OTHER : Event nuggets of failed events, future events, conditional statements, and all other non-generic variations.

One more way we would like to attempt to classify event nuggets is to check whether the event is a main event. Main event is the event that is the reason why the news is written. It will be a binary classification of an event nugget to figuring out if the event nugget is the main event.

1.2. Problem Description

With the terminology having been introduced, we can now talk about what our research is about. The problem that we would like to tackle is the detection of event nuggets in Turkish news texts and classification of event mentions within news text depending for their event type, the realis value and whether they are the main event inside that news text.

There are two main tasks and two other side tasks that we would like to tackle. The first task is detecting the event nuggets in a news text. Let us look at some challenges that we will face in this task.

There may be only one event nugget in a document, as well as multiple event nuggets that refer to the same event. Also it is possible to have different event nuggets in the same sentence that are mentions of different events. As mentioned, event nuggets can be multiword, also between the words of the event nugget there may be words that are not part of the event nugget.

Example 1.2.1. Ünlü sanatçı *dünyaya** bu eski evde *geldi**.
Translation: The famous artist *was** *born** in this old house.

Due to the already explained nature of event nuggets, we would like to consider this as a sequence classification problem. We will talk more about this in Section 3.2.

The second task is to classify the detected event nuggets into event nugget type classes. There are 38 event types that are given in Table 1.1. Adding the event type "None" to the event type set, we have 39 classes that we can classify an event nugget

into, however the “None” type is not used in our experiments for nuggets since we use sequence classification first to detect event nuggets. The “None” type is used for other tokens inside the text that are not event nuggets. Naturally, an already identified event nugget cannot have “None” as its event type. These are the two main tasks that we would like to handle. We also have two side tasks as we mentioned. Now let us take a look at them.

Our first side task is to identify the realis value of an event. It can have 3 different values, namely Actual, Generic and Other as explained in Section 1.1. The second side task is to detect whether an event mention is the main event of the news text file. Main event is the reason why the news text is written therefore only one event can be the main event inside the news text whereas there can be more than one event mentions referring to the same event. This means that there can be more than one event nugget that belongs to the main event inside a news text.

The research in this thesis is based on Turkish texts, but may expand to other languages over time depending on the different methods to be used. We are planning to exploit language dependent features as well as language independent features. Since Turkish is a morphology rich language, we believe that by making use of morphological features we can achieve significant results.

The way we would like to attempt to solve this problem is through trying different machine learning techniques, namely logistic regression, random forest and extreme gradient boosting, as well as using a combination of different features including Turkish specific dependency related and morphological features.

We believe that by developing a new method and retrieving higher results for accuracy in detecting and classifying event nuggets, we will provide more room for research for more complex tasks such as fake news detection. Also identifying similar news and grouping them can be achieved with higher accuracy.

2. BACKGROUND

Until now, we have done a brief introduction to the problem. In order to fully understand the nature of the study and the methods that we make use of, some background information should be introduced. We will first focus on what has been done in the scientific world for event nugget detection and event type classification. Secondly, we will give a brief information of the characteristics of the Turkish language.Then, we will provide information on some of our highlighted techniques.

2.1. Related Work

There have been many attempts to tackle the problem of information extraction from texts. There are different approaches in information extraction such as rule-based approaches and machine learning techniques as well as hybrid approaches [7].

The event nugget detection problem is one of those information extraction problems that have been tried to be solved. Due to the fact that rule based approaches require a lot of hand crafted features, the recent research gave focus to machine learning techniques for information extraction as well as detection of event nuggets [8].

Hsi *et al.* used multilingual data sources to detect event nuggets [3]. They attempted to develop a cross-lingual event extraction technique that made use of both language dependent and language independent features. They used a logistic regression classifier. They made use of features like current word/lemma and bigrams of the current word/lemma with words/lemmas within a fixed context window, part-of-speech tag for the current word, bigrams of part-of-speech tags for the current word and words within a fixed context window, word embedding vector for the current word, universal part-of-speech tags, and dependent/governor information from dependency parsing. The event triggers were found using these features and consecutive event triggers and words were merged to determine the nuggets if they existed in the phrase list. They discovered that using Chinese text increased the f-score of less frequent classes, whereas

it decreased the f-score of more frequent classes. The results were actually improved in macroaverage, but worse in microaverage.

Sammons *et al.* developed a supervised model for event nugget detection with some lexical and semantic features [9]. They used Support Vector Machines (SVMs) with features such as lexical features, features from a parser, Named Entity Recognition (NER), Semantic Role Labelling (SRL), entity co-reference and WordNet, and other semantic features from Explicit Semantic Analysis (ESA). They used ACE2005 as development and test data set. They had 34 event subtypes. The highest score with ACE-KBP trained data was achieved with resampling and subsampling. For REALIS, the same classifier was used. They achieved an f-score of 49.42 for event type detection and an f-score of 42.87 for REALIS classification. Their scores for both type and REALIS are reported as type+realis=precision:49.88 recall:28.5 f-score:36.28.

Previously in the literature, hand designed features were used (using NLP toolkits) and these features are language dependent, since they require linguistic knowledge in that language. Ghaeini *et al.* developed a system for event nugget detection using For- ward Backward Recurrent Neural Networks [10]. Convolutional Neural Networks (CNN) was applied before but for single token event detection which required fixed size window. Using Recurrent Neural Networks (RNN) removed the problem of windows being fixed size and enabled the system to detect nuggets consisting of more than one word. As data set to train their system, they used ACE2005 and Rich ERE 2015. To the best of our knowledge, this is the first time in literature that RNN is used for the event nugget detection task.

Hou *et al.* also used machine learning techniques to tackle the task of the detection of the events, event types, subtypes and REALIS using distributional semantics and neural embedding techniques [11]. They have two contributions in event extraction: Extracting semantic features using semantic role labeling and predicting events, event types and subtypes using neural networks. They made use of semantic role labeling. Semantic scores were assigned for phrase type, grammatical function, position, voice and FrameNet&VerbNet features. Each word was represented as a vector of 5

elements, elements being conditional probabilities. The skip-gram model was used to learn distributed vectors. They achieved micro-averaged 100% precision with 55.16 % recall (f1-score:71.10) and macro-averaged 98.51% precision with 63.7 % recall (f1-score: 77.37). The low recall score is given to be due to the weak handling of unseen data.

Satyapanich *et al.* also have taken up the event nugget detection task [12]. They addressed detecting 18 event types and REALIS. They designed two systems. The first system makes use of CNN. Each word is represented as a vector and each sentence is represented as a matrix with a window size of 10 words. Pre-trained word vectors were trained on Google News data set (making it a total 300 dimensional vectors for 3 million words and phrases). Preprocessing includes exclusion of stop words and editing for window sizing. If a sentence has a trigger word, it was processed for selection. For selection, conditional probabilities were used. Every word W has 3 values: the similarity value (max similarity of the word with the trigger words in corpus), P(Event — W), P(Non-event — W). Only one word was selected to be the trigger word of the sentence. The second system makes use of features extracted from the training data. Features include similarity values, part of speech tags, frequency of each named entity type found in the same sentence, dependency function of the target word. To find realis value, they made use of verb tenses, four named entity types surrounding the verb, number of habitual occurence word in the sentence, number of negation and conditional words, number of modality words. The training data is ACE2005 corpus, DEFT Rich ERE English training annotation v2, DEFT Rich ERE English training annotation R2 v2, DEFT Rich ERE Chinese and English parallel annotation v2. The best result for the first system is f1-score of 34.14 and for the second system f1-score of 35,24. The low performance is said to be due to imbalance in the training data and the small size of the training data.

Liu *et al.* also studied the detection of event nuggets [13]. They developed two systems. Both systems make use of a conditional random field (CRF) system with different selected features. The first system uses a vanilla average perceptron CRF model with multi-type mentions belonging to new classes. The second system

uses passive aggressive CRF model and makes use of 5 best decoding to cope with multi-type mentions. The features used for first system are Part-of-Speech, lemma, named entity tag of words in the 2-word window, Brown clusters, WordNet synonym and derivations of the trigger word, closest named entity type, whether surrounding words match selected WordNet senses, dependency features, and semantic role related features. SVM is used. The best score retrived for 5 fold validation is 72.66 as f-score. Second system is treated as a sequence prediction task. They optimized k-best viterbi parses during CRF training. Their overall score for event type detection is given as 46.67 (f1-score). Realis detection f1-score is 34.88.

Nyugen *et al.* tackle the event nugget detection, REALIS classification and coreference tasks [14]. They used neural networks so that the features are learned automatically rather than doing feature engineering. As preprocessing they made use of sentence detection and tokenization with the OpenNLP toolkit and dependency parsing with the CoreNLP toolkit. They consider event nuggets as single tokens and as a classification problem. They made use of 3 vectors concatenated, word embeddings, position embeddings and dependency vector. They used Non-consecutive Convolutional Neural Networks (NCNN) for nugget detection. Although they tried bidirectional recurrent neural networks (BRNN), since they got worse results they continued with non-consecutive convolutional neural networks(NCNN). For REALIS classification, they use the same machine learning techniques with different features. The data for training is EN2015 training data, DEFT Rich ERE English training annotation, half of the evaluation data for EN2015. They achieved f1-scores of 67.77 for plain, 59.74 for type detection, 53.82 for REALIS detection, which were higher than the TAC 2015 winner system's scores.

Lu and Ng used multiple 1-nearest neighbor models for event nugget detection and event nugget classification [15]. They use five 1-nearest neighbor models with features derived from dependency tree parser and named entity recognition tagger. They achieved an f-score of 50.37 on [16] for event nugget detection and classification by classifying nuggets into 18 event subtypes instead of 38. The f-score for realis identification is reported to be 40.91.

Jiang *et al.* employed a combination of bidirectional LSTM and CRF model for nugget detection and classification for 18 subtypes [17]. For realis classification they used SVM model. They used word embeddings, lemma, dependency type, tense, synset, grammar, NER nearby, token position, sentence position, trigger word dictionary and WordNet index as features and made use of sequence classification to detect event nuggets. They achieved 68.06 f-score for nugget detection, 56.92 f-score for nugget type classification, 48.75 f-score for realis classification.

To the best of our knowledge, there does not exist any prior work on event nugget detection for Turkish.

2.2. Turkish Language Characteristics

Turkish language is the language that is spoken in Turkey, Cyprus and abroad by the Turkish diaspora. It is one of the Turkic languages and sometimes it is called Turkey Turkish. It uses the Turkish Alphabet which is the Latin Alphabet with added letters to correspond to the sounds that do not exist in Latin Alphabet.

Turkish language does not have grammatical gender or word classes. It is an inflected language and usually follows Subject-Object-Verb (SOV) word order, though the word order is flexible since inflections help to determine the roles of the words.

Turkish has extensive agglutination and vowel harmony which makes morphological analysis more useful and necessary. In theory, using agglutination, you can make a word consisting of infinite number of letters, but possibly after adding too many suffixes the word will be incomprehensible even if it is syntactically acceptable.

There are six word cases (nominative, accusative, genitive, dative, locative, ablative) and no definite articles in Turkish. There are two types of suffixes, derivational suffixes and inflectional suffixes. Derivational suffixes change the meaning of the word whereas inflectional suffixes change the function or case of the word. Verb conjugations and plural suffixes are also of this type. This rich morphology requires extra analysis

before we can make NLP analysis on a Turkish sentence.

2.3. Related Techniques

In this section we provide a brief summary about the techniques that we used in this thesis. We have used a variety of different techniques and models. We are going to discuss the most important ones that are critical for our research. These techniques include Smote oversampling, logistic regression, random forest classifier, extreme gradient boosting and word embeddings.

2.3.1. Smote Oversampling

We have a highly unbalanced data. Some event types do not exist in the data set, some are just a few, even though some event types have more than 500 instances. This create problems in classification as some models may tend to favor the majority class for classification so some measure should be taken. Working with unbalanced data, to solve issues in the data level, we can either downsample the classes with higher instances than a threshold or we can apply oversampling for minority classes.

Smote stands for Synthetic Minority Oversampling Technique, so what Smote does is that it synthesizes new samples from the minority classes by creating a feature space from its N nearest neighbor and randomly picking features within the feature space, leaving majority classes untouched. Since it makes use of N nearest neighbor method, the number of data instances in the minority class should be given greater than N. This method is shown to work well with imbalanced data. In our results as well, using oversampling almost always improved our classifications.

2.3.2. Logistic Regression

Logistic regression is one of the most prevalent classification methods. It is considered simple yet powerful. Although it actually is a binary classification method, multiclass classification can be realized using a One versus the Rest (OvR) method. It

performs especially well on linearly separable classes.

Firstly net input is calculated via linear combination of weights and sample features. Then net input is evaluated by the activation function. The output of the activation function is used to determine the class labels. Minimizing the cost function helps us to determine the weights more accurately.

Logistic regression has a sigmoid activation function.

$$g(z) = \frac{1}{1 + exp(-z)}$$

It makes use of a cost function J given below. x_i, y_i, m and β denote i^{th} feature of a data instance, i^{th} label of a data instance, the number of data instances and best parameters to make to decision boundary respectively.

$$J(\beta) = \frac{1}{m}\sum_{i=1}^{m}[-y_i log(h_\beta(x_i) – (1 – y_i) log(1 - h_\beta(x_i))]$$

The algorithm flow is shown in Figure 2.1.

2.3.3. Random Forest Classifier

Random forest classifier is a kind of ensemble learning method that makes use of decision trees. It is a very common example of ensembling many weak learners to build a strong learner. It became popular due to its scalability, classification performance and ease of use. The random forest classification algorithm is given in Figure 2.3.3.

Random forest classification algorithm grows a given number of decision trees and the key difference is that not all features are evaluated for prediction by each tree

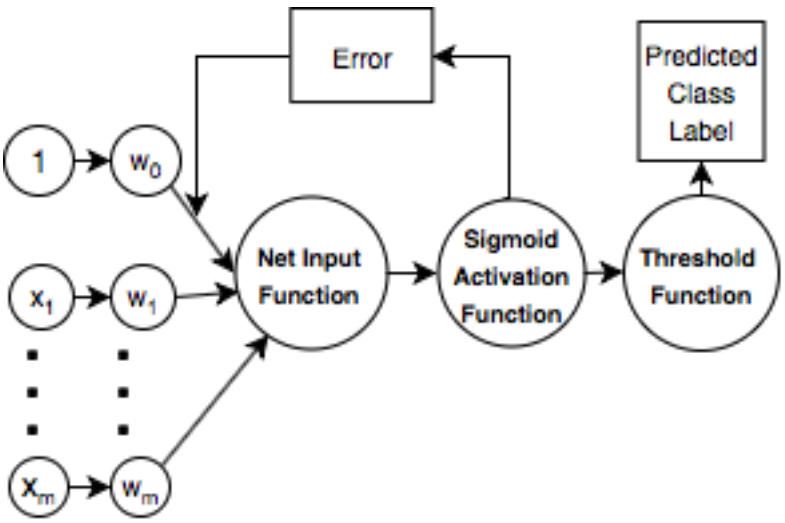

Figure 2.1. Logistic Regression Algorithm Flow [18].

for $k = 1$ to K **do**
  Draw random samples of size n from the training set with replacement;
  Build a decision tree from the sample;
  for Each node in the tree **do**
    Randomly select d features;
    Split the node using the feature with maximum information gain;
  end for
end for
Assign the class label by majority vote from the predictions by each tree

Figure 2.2. Random Forest Classification Algorithm [18].

but the features are selected randomly. This type of ensemble learning makes a robust model and has a better generalization performance than an individual decision tree. Although random forests make use of decision trees, they are not as interpretable as decision trees.

2.3.4. Extreme Gradient Boosting (XGBoost)

Extreme Gradient Boosting is one of the most popular techniques. It is a type of gradient boosted decision trees that can be used for classification, regression and ranking. It proved itself to have higher speed and performance to many data scientists and is used frequently in data science competitions.

As with the random forest classifier, XGBoost also is an ensemble model of weak learners that are decision trees in this case. It is designed be "a more regularized model formalization to control over-fitting, which gives it better performance," as stated by [19].

Extreme gradient boosting is an optimized version of gradient boosted decision tree algorithms for better performance. It supports parallelization which allows the algorithm to run on parallel on different processors. Tree pruning is done with a depth first, pruning second approach and this improves the computational performance. The algorithm is designed to make use of the hardware resources efficiently. It applies L1 and L2 regularization to prevent overfitting.

2.3.5. Word Embeddings

Word embeddings are representations of words in a vector space with real numbers that were learned via neural networks and dimentionality reduction [20] [21]. It is widely used in machine learning and especially in deep learning.

Normally to encode a word, usually an array of unique encodings for each word is required. This creates a big vector space, and for a new word in the vocabulary, a

new dimension is created. These encoding may suffer from the curse of dimensionality with the increasing vocabulary size. Also usually with this kind of encoding, we end up with very large sparse vectors.

Words embedding are dense vectors with fewer dimensions. The dimension size can be selected and it is usually arbitrary and empirical. Word embedding can capture semantic similarities and allows us to do semantic operations on them. For word embeddings to have meaningful representations, the data should be of reasonably big size.

This new vector space capturing the semantic similarities of words opens door to various semantic operations that can be applied to words. Checking word meaning similarity or synonyms is one common usage. An example of semantic operations with word embedding usage is as follows.

Example 2.3.1. When we query [Mother + Men - Women], the word embedding vector yields = Father

3. DATA

Since there was no annotated data set for event nugget detection, we had to come up with a data set by manual annotation of Turkish News Text. Instead of crawling the web for Turkish news texts, we made use of the data set presented in [22]. The total number of News Text that were annotated are 190. All 190 files were annotated token by token and then added to the same python dictionary, which makes up our data set.

The events and event nuggets in news documents are annotated using the Rich ERE Annotation Guideline v2.5.1 [23].

3.1. Format

The data is annotated manually by a tailor-made annotation tool that we developed for this task. The data set is a python dictionary embedding in a pickle object. Its keys are filenames and value is another python dictionary. This python dictionary includes seven five elements: “annotated”,“file_name”, “tokens”, “sentences”, “file_text”. Annotated is a boolean value, that returns true if the text file with the given file name is annotated. Sentences are a list of tokenized sentences from the news text and can be found with “sentences” attribute. “file_text” has the news text as a string. “tokens” is a list of tokens having all the manually annotated tokens in the order they appear in the news text.

Tokens have four annotated attributes which are sequence type for nugget detection, event type, realis value and main event value. The definition of these annotated attributes and their annotation guidelines are presented in detail in Section 3.2.

3.2. Annotation Guidelines

News files are first tokenized into paragraphs and sentences and then into smaller chunks which are tokens made out of words, punctuations and numbers. Newlines are marked with the token "<newline>". At the end of every news file, the token "<eof>" is added. With all these added tokens, we have 45,271 tokens in total. The data is evaluated token by token. All 45,271 tokens are annotated with four different classes of labels: sequence type, nugget type, main event attribute and realis value.

The tokens are annotated first for their sequence types. BILOU tags are used for Sequence Type Classification. The tags stand for:

- B - "beginning of event"
- I - "inside event"
- L - "last of event"
- O - "outside event"
- U - "unit, uniword event"

If a token is an event trigger (one word), its sequence type value (seq_type) is marked as "U". If the token is not part of an event nugget, it is simply marked as "O". If the token is part of a multiword event nugget, the relevant "B", "I", "L" tags are used. Let's illustrate with an example.

Example 3.2.1. Yeni yavrular dünyaya gözlerini açtılar. Translation: New puppies opened their eyes to the world (were born).

Since *"yeni yavrular"* (newborn puppies) is not an event the tokens were tagged as "O". *"Dünyaya gözlerini açtılar"* is a phrase that means "were born". We cannot separate this phrase any way so that it carries the same meaning, so they are all parts of the same event nugget. Since *"dünyaya"* is the first word of the token, it is tagged as "B" (beginning of event). The second word *"gözlerini"* is part of the event nugget, but it is neither first word, nor last word, hence it is tagged as "I" (inside event). The

third word is *açtılar* tagged as "L" since it is the last word of the event nugget. The "." is not part of the event nugget so it is tagged as "O". The annotations for this sentence can be seen below.

Yeni	yavrular	dünyaya	gözlerini	açtılar	.
O	O	B	I	L	O

After tagging the sequence, we tagged each token for their nugget type. Each token belongs to one and only one of the 39 subtypes given in Table 1.1 (38 subtypes + "Not" type which means the token is not part of an event nugget). This feature is called "nugget_type" in the data set. Now let us see how each event nugget type is annotated.

- Business: It is for business related events. It has four subtypes. Only event nuggets belonging to these four subtypes are annotated as business event.
  (i) Declare-Bankruptcy: If a business declare bankruptcy, this subtype is chosen for annotation.
  (ii) End-Org: If a business closes down for whatever reason, this subtype is chosen for annotation.
  (iii) Merge-Org: If merging different businesses/organization is mentioned this subtype is chosen for annotation.
  (iv) Start-Org: If a business/organization starts, this subtype is chosen for annotation.
- Conflict: There may be many conflict events like political conflicts or war, but since the annotation guidelines [6] has two subtypes, only event nuggets related to these two subtypes are annotated as conflict type.
  (i) Attack: There may be many different meanings behind an attack subtype. It can be two people fighting. It can be domestic violence. There can be a terrorist attack or a military intervention. Sometimes even a mention of war is an attack if an attack is mentioned explicity, whereas declaring war itself without an attack is not considered as an attack. If any of the attack

instances is mentioned this subtype is chosen for annotation.

(ii) Demonstrate: The mention of any type of demonstration is considered to be of demonstrate subtype. It can be going on a strike, marching towards somewhere chanting, other types of protests.

- Contact: It is for communication related events. Any mention of any contact, speech, communication even if it is a sound bite is considered a contact event. It has five subtypes, namely broadcast, communicate, contact, correspondence and meet.

  (i) Broadcast: If the communication is one-way in the sense that if the speaker does not know who exactly is listening, then this subtype is chosen for annotation. Speeches, sound bites, politicians making statements are all of this subtype.

  (ii) Communicate: This type was not used for annotation, because in the latest studies it is not used anymore. For this study to be backwards compatible with other studies, we still included this subtype in our event type definitions.

  (iii) Contact: If the communication does not fall under categories, this subtype is annotated. When we cannot infer the medium or the audience, we choose this category for event nuggets.

  (iv) Correspondence: Event mentions of two-way communications of not-in-person medium are considered to be of correspondence subtype.

  (v) Meet: For the event mention to be of this subtype, contact should be two way and in-person. Mentions of meeting face-to-face also fall under this category.

- Justice: This is the nugget type that has the most number of subtypes with 13 different subtypes. These subtypes are acquit, appeal, arrest-jail, charge-indict, convict, execute, extradite, fine, pardon, release-parole, sentence, sue, trial-hearing. This type is chosen for all justice related event mentions.

  (i) Acquit: This subtype is chosen if a person or an entity is acquitted. Though, there has not been a single occurence of this event mention type in our data set.

(ii) Appeal: Mentions of appeal or requests of appeal call for this event subtype.

(iii) Arrest-Jail: If someone is caught by the police, arrested, taken into custody or sent to jail without explicit mentioning of any recent sentencing, this subtype is chosen for annotation.

(iv) Charge-Indict: If there is a mention or indiction, or someone is charged for a crime, this subtype is annotated.

(v) Convict: Mentions of convictions, past or present, are of this category.

(vi) Execute: If someone is executed, this category is chosen.

(vii) Extradite: If there is a mention of someone being extradited, this subtype is used as the label.

(viii) Fine: Event mentions of paying fines are of this subtype however we do not have event nuggets of this subtype in our data set.

(ix) Pardon: If there is a mention of a pardon, this subtype is chosen. Event nuggets of this subtype do not exist in our data set.

(x) Release-Parole: If somebody is released from jail without mentions of any pardon, or if parole occurs, this subtype is chosen for annotation.

(xi) Sentence: If a mention of any sentencing exists, this subtype is chosen. It can be sentencing for community service, sentencing for execution, sentencing for jail time.

(xii) Sue: If there is a mention of suing, or even a threat of suing, this subtype is chosen for annotation.

(xiii) Trial-Hearing: Any mention of trial or a hearing triggers this event subtype.

- Life: Life event type covers life related event mentions such as birth, death, divorce, injury and marriage.

  (i) Be-Born: With any mention of being born, even if it is casually mentioned like “Istanbul-born writer”, this event subtype is chosen for annotation.

  (ii) Die: Any mention of death call for this event subtype, even if it is the death of a historical person casually mentioned. (The famous writer who died in 1771...)

  (iii) Divorce: This subtype is chosen for annotation for any mention of divorce in an event nugget.

(iv) Injure: Any type of injuries regardless of the cause is annotated with this subtype.

(v) Marry: Event nuggets mentioning any marriage event, or mentioning someone being married calls for this event subtype.

- Manufacture: This event type has only one subtype: artifact.
  (i) Artifact: This event subtype is chosen for annotation when there is a mention of production of things. Words like hand-made, made-in-Turkey also triggers this event subtype.
- Movement: Any explicit mention of moving from one place to another, unless it is a protest, is a movement event. It has two subtypes, one for moving objects, one for moving people.
  (i) Transport-Artifact: Any artifact moving itself, like plane flying somewhere or if it moved, like carrying a box, this subtype is chosen for annotation.
  (ii) Transport-Person: Any person moving from one place to another, either voluntarily or by force, this subtype is chosen. If somebody's body is carried, whether the person is alive or dead, this subtype is chosen as well. Labeling with this subtype was sometimes complicated due to the fact that vehicles have mostly drivers, so if the driver is mentioned for the movement, this type is chosen. Otherwise, if the vehicle moves without the explicit mention of the driver, then transport-artifact is chosen for annotation.
- Personnel: This event type has four subtypes concerning the status of personnel. They are elect, end position, nominate and start position.
  (i) Elect: Any mention of someone being elected for a position calls for this subtype.
  (ii) End-Position: If somebody quits his job/ place in office, or is fired, this event subtype is chosen for annotation.
  (iii) Nominate: Someone being nominated for a position calls for this event subtype however we do not have a single instance of event nuggets of this subtype.
  (iv) Start-Position: Any mention of someone starting a new position, either a new job, or being transfered to another department, triggers this event sub-

types.

- Transaction: This event type has three subtypes: transaction, transfer money and transfer ownership. It covers transaction related events.
  (i) Transaction: If the event nugget mentions a transaction, but you cannot infer if it is money or commodity, then this subtype is chosen.
  (ii) Transfer-Money: This event subtype covers many money related events. Paying taxes, transferring money, donating money, paying anything but fines are all of this subtypes.
  (iii) Transfer-Ownership: This event subtype cover all buying and selling, donating things except money, transfering ownership of anything, even if it is a bank account full of money.

The third annotation is the main event attribute. This attribute is binary, i.e. it has values either True or False. The token is annotated as True if it is part of an event nugget that is the main event. Main event is the reason why the news story is written. This feature is called "is_main_event" in the data set. The news text files have only one event that is the main event but multiple event nuggets can refer to the same event so there can be more than one event nugget that is labeled as main event. Labeling this was at times challenging especially if the event is of contact type. News mentioning a politician making statements with words like "he said that" can be a main event but determining which of these words "said, reported, added, emphasized..." constitute the main event is complicated.

Lastly, we annotated the realis value as A, G, O or N. A, G and O represent Actual, Generic and Other respectively as explained before, and N is used for non-nuggets. This feature is called "realis" in our data set. Let us see how it is annotated.

- ACTUAL : Event nuggets whose event mentions clearly state that the event actually took place are labeled as actual as their realis value. These type of events are usually in past tense.

- GENERIC : Event nuggets that do not mention a specific time and/or place. Events that occur generally, or facts like "all people die" are of this category.
- OTHER : Event nuggets of failed events, future events, conditional statements, and all other non-generic variations are of this category. Basically if the event nugget is not actual or generic, this realis value is chosen for the realis value of the event nugget.

3.3. Data Statistics

Here are the statistics about the News Data Set that we have annotated. The total number of News Text that were annotated are 190. New lines are marked with the token "<newline>". At the end of every news file, the token "<eof>" is added. With all these added tokens, we have 45,271 tokens in total.

As stated earlier, BILOU tags are used for Sequence Type Classification. Statistics about the Sequence Types are shown in Table 3.1.

Table 3.1. Data Frequency for Sequence Type.

Sequence Type	**Token Count**
B	621
I	76
L	621
O	42,593
U	1,360
TOTAL	45,271

The second statistics we would like to point out is the number of tokens that are part of main events in Table 3.2. It is also important to emphasize that main events that do not belong to any of the 38 subtypes described before are not marked as main event.

Table 3.2. Data Frequency for Main Events.

Main Event	**Token Count**
True	2,014
False	43,257
TOTAL	45,271

The third statistics is about the Realis values in Table 3.3. The tokens that are not part of event nuggets are shown as Null. In the data set, Null is marked as "N".

Table 3.3. Data Frequency for Realis Value.

Realis Value	**Token Count**
A	1,568
G	107
O	1,003
Null (Non-nugget)	42,593
TOTAL	45,271

The last statistics is about Nugget Types and Subtypes. It is the most important one as it is our main research of interest to classify event nuggets into these types and subtypes.

There are nine different event types for event nuggets as well as 38 different subtypes, that are defined by the Rich ERE Annotation Guideline v2.5.1 [23].

The first type is Business, shown in Table 3.4, which has a total of 54 tokens as parts of event nuggets which are labeled as subtypes of Business event nugget type.

The second type is Conflict, shown in Table 3.5, which has a total of 284 tokens as parts of event nuggets which are labeled as subtypes of Conflict event nugget type.

Table 3.4. Data Frequency for Business Subtypes.

Business	**Total**: 54
Declare-Bankruptcy	3
End-Org	16
Merge-Org	5
Start-Org	30

Table 3.5. Data Frequency for Conflict Subtypes.

Conflict	**Total**: 284
Attack	233
Demonstrate	51

The third type is Contact, shown in Table 3.6, which has a total of 850 tokens as parts of event nuggets which are labeled as subtypes of Contact event nugget type.

Table 3.6. Data Frequency for Contact Subtypes.

Contact	**Total**: 850
Broadcast	704
Communicate	0
Contact	67
Correspondence	18
Meet	61

The fourth type is Justice, shown in Table 3.7, which has a total of 366 tokens as parts of event nuggets which are labeled as subtypes of Justice event nugget type.

The fifth type is Life, shown in Table 3.8, which has a total of 384 tokens as parts of event nuggets which are labeled as subtypes of Life event nugget type.

Table 3.7. Data Frequency for Justice Subtypes.

Justice	**Total**: 366
Acquit	0
Appeal	4
Arrest-Jail	133
Charge-Indict	26
Convict	5
Execute	35
Extradite	6
Fine	0
Pardon	0
Release-Parole	55
Sentence	7
Sue	47
Trial-Hearing	48

Table 3.8. Data Frequency for Life Subtypes.

Life	**Total**: 384
Be-Born	13
Die	210
Divorce	4
Injure	133
Marry	24

The sixth type is Manufacture, shown in Table 3.9, which has a total of 73 tokens as parts of event nuggets which are labeled as subtypes of Manufacture event nugget type. There is only one subtype belonging to this nugget type.

Table 3.9. Data Frequency for Manufacture Subtypes.

Manufacture	**Total**: 73
Artifact	73

The seventh type is Movement, shown in Table 3.10, which has a total of 420 tokens as parts of event nuggets which are labeled as subtypes of Movement event nugget type.

Table 3.10. Data Frequency for Movement Subtypes.

Movement	**Total**: 420
Transport-Artifact	144
Transport-Person	276

The eighth type is Personnel, shown in Table 3.11, which has a total of 97 tokens as parts of event nuggets which are labeled as subtypes of Personnel event nugget type.

Table 3.11. Data Frequency for Personnel Subtypes.

Personnel	**Total**: 97
Elect	1
End-Position	26
Nominate	0
Start-Position	70

The ninth type is Transaction, shown in Table 3.12, which has a total of 150 tokens as parts of event nuggets which are labeled as subtypes of Transaction event nugget type.

The last table is the number of tokens that do not belong to any type, which is marked as the 39th type, namely “Not”. In Table 3.13 non-nugget types and the total

Table 3.12. Data Frequency for Transaction Subtypes.

Transaction	**Total**: 150
Transaction	27
Transfer-Money	51
Transfer-Ownership	72

number of tokens are shown.

Table 3.13. Data Frequency for Non-Nugget Types.

Not (Non-nugget)	42,593
TOTAL (of all event types)	45,271

4. METHODOLOGY AND IMPLEMENTATION

4.1. Preprocessing

We randomly gathered 190 news that all have less than 1000 words per news. With the raw news text, we first separated paragraphs and then sentences. We tokenized sentences into tokens consisting of words, punctuations and numbers. We changed all newline characters with the token "<newline>" and we added "<eof>" at the end of each news file. We added the data of all news files into the same python dictionary, keeping their filename as key. After data annotation was done, the resulting data set is saved as a pickle object so that this data set can later be imported and used.

We then applied the second part of our preprocessing to our data set. We took all the annotated tokens from the data set, and concatenate them to an array. We applied feature engineering onto this data set to create a data frame (another data set) containing all the features. In this preprocessing stage, we engineered all the relevant features such as morphological information, dependency parse tree, and word embeddings as described in Section 4.2. This data frame is used for all our experiments.

4.2. Features

We made use of a combination of different features. We have crafted bigrams, trigrams, fivegrams, wordshape, sentence number, filename, token number, start and end positions inside the next text and a feature called "is title" from the data set. We use the dependency parser of [24] to have more features for the tokens such as dependecy head, child words, child positions, dependency relation label, parent word. We used the morphological analyzer and disambiguity resolver of [25] to obtain more features such as postag and stem. For word embeddings we have used the pretained word embeddings constructed by [26].

Below are the list of features with their explanations. In order to describe them better, we will demonstrate an example of what each feature looks like from an actual data token in the data set.

Consider the sentence below.

Example 4.2.1. Bu gerçekten çok önemli dedi.

Each word is given a unique number as ID depending on their position in the data frame. For the sake of clarity, we would like to mention words with their IDs in the examples. Here is the same sentence with their IDs written below the words.

Bu	gerçekten	çok	önemli	dedi	.
242	243	244	245	246	247

The listed feature names are named in all lowercase letters and with underscores instead of spaces in our data frame.

4.2.1. Numerical Features

Most of our numeric features are integers or list of integers. The word embedding feature is a Pandas DataFrame consisting of a 400 dimensional word vector.

- *Child Positions*: This feature is taken from the dependency parser. It gives us the children positions of the token according to the parse tree of the token's sentence. The positions are relative to the token. (-1 means the first left neighbor, 2 means the second right neighbor.) The token itself (0) is always included in the child positions in order to have at least one element in this feature. This feature has the name "child_positions" in the data frame.

  Example 4.2.2. Token 246 "dedi" has [-2, -1, 0, 1] as child_positions value.

- *Dependency Head*: This feature is also taken from the dependency parser. It is the relative position of the parent of the given token in the dependency tree.

  Example 4.2.3. Token 246 “dedi” has 0 as child_positions value, since it is the head of the dependency tree, it refers to itself.

- *Distance To Root*: This feature is calculated from the dependency parser. It is the distance of the current token to the root token in the parse tree.

  Example 4.2.4. Token 246 “dedi” has 0 as distance_to_root value, since it is the root of the dependency tree, the distance of token 246 is 0 to the root.

- *End*: This feature is the character-wise end position of the token with respect to the news file that it is taken from.

  Example 4.2.5. Token 246 “dedi” has 541 as end value.

- *Raw Dependency Head*: This is the dependency head position exactly as taken from the dependency parser. The number refers to the indexes (starting from 1) of the tokens in a sentence. The root of any sentence has 0 as raw dependency head value. The feature has the name “raw_dependency_head” in the data frame.

  Example 4.2.6. Token 246 “dedi” has 0 as raw_dependency_head value, since it is the root of the sentence.

- *Sentence Id*: This feature is the index of the sentence that the token is from inside the whole data set. The feature name is “sentence_id” in the data frame.

  Example 4.2.7. Token 246 “dedi” has 13 as sentence_id value.

- *Sentence Number*: This feature is the index of the sentence that the token is from inside the news text file. The feature name is "sentence_number" in the data frame.

  Example 4.2.8. Token 246 "dedi" has 2 as sentence_number value, which means the token is from the second sentence inside the news file text it is from.

- *Start*: This feature is the character-wise start position of the token with respect to the news file that it is taken from.

  Example 4.2.9. Token 246 "dedi" has 537 as start value.

- *Token No*: This is the index of the token inside the news text file. It has the name "token_no" in the data frame.

  Example 4.2.10. Token 246 "dedi" has 79 as token_no value.

- *Token No In Sentence*: It is the index of the token in its sentence.

  Example 4.2.11. Token 246 "dedi" has 4 as token_no_in_sentence value, since it is the fourth token in the sentence.

- *Word Embeddings*: This is the pretrained word embeddings vector for the given token.

4.2.2. Textual Features

Our textual features are by nature all categorical features. They can be strings, list of strings or more complicated structures that are made out of strings, like python dictionary.

- *Bigraml*: The token is concatenated with its left neighbor. If there is no left neighbor, "<eof>" is added as a placeholder. This feature has the name "bigraml" in the data frame.

  Example 4.2.12. Token 246 "dedi" has "önemli dedi" as *bigraml* value.

- *Bigramr*: The token is concatenated with its right neighbor. If there is no right neighbor, "<eof>" is added as a placeholder. This feature has the name "bigramr" in the data frame.

  Example 4.2.13. Token 246 "dedi" has "dedi ." as *bigramr* value.

- *Child Words*: This feature has the child words as a list from the dependency tree. It is calculated from the child positions feature. This feature has the name "child_words" in the data frame.

  Example 4.2.14. Token 246 "dedi" has "['Çok', 'önemli', 'dedi', '.']" as *child_words* value.

- *Child Words Str*: This feature has the child words as a string from the dependency tree. It is calculated from the child words feature. This feature has the name "child_words_str" in the data frame.

  Example 4.2.15. Token 246 "dedi" has "çok önemli dedi ." as *child_words_str* value.

- *Dependency Relation Label*: This feature is taken from the dependency parser. It is the dependency relation label of the token from the dependency tree and it has the name "dependency_relation_label" in the data frame.

  Example 4.2.16. Token 246 "dedi" has "root" as *dependency_relation_label* value.

- *Filename*: Filename of the news text that the token is taken from. This feature has the name "filename" in the data frame.

  Example 4.2.17. Token 246 "dedi" has "planet/1008" as *filename* value.

- *Fivegram*: Fivegrams of the token are calculated from the data set via concatenating bigrams of two neighboring tokens. If there are no neighboring tokens, "<eof>" is added as a place holder.

  Example 4.2.18. Token 246 "dedi" has "çok önemli dedi . <newline>" as *fivegram* value.

- *L1neighbor*: This is the first left neighboring token with respect to the token.

  Example 4.2.19. Token 246 "dedi" has "önemli" as *l1neighbor* value.

- *L2neighbor*: This is the second left neighboring token with respect to the token.

  Example 4.2.20. Token 246 "dedi" has "çok" as *l2neighbor* value.

- *Leftwordpostag*: This is the postag of the left neighboring token with respect to the token.

  Example 4.2.21. Token 246 "dedi" has "Unknown" as *leftwordpostag* value. (The morphological analyzer was unable to assign a postag to the left neighboring token.)

- *Parent Word*: This feature is calculated from the position of the dependency head feature. It is the string version of the same dependency head. This feature has the name "parent_word" in the data frame.

Example 4.2.22. Token 246 “dedi” has “dedi” as *parent_word* value.

- *Postag*: Although the name postag is used for this feature, it entails so much more. This is a Turkish-specific feature with all the morphological information we could gather from the word. It is the postag of the word along with all the inflectional suffixes and their functionalities. This feature has the name “postag” in the data frame.

  Example 4.2.23. Token 246 “dedi” has “Verb Pos Past A3sg” as *postag* value.

- *R1neighbor*: This is the first right neighboring token with respect to the token.

  Example 4.2.24. Token 246 “dedi” has “.” as *r1neighbor* value.

- *R2neighbor*: This is the second right neighboring token with respect to the token.

  Example 4.2.25. Token 246 “dedi” has “<newline>” as *r2neighbor* value.

- *Rawtag*: This is the output from the morphological parser.

  Example 4.2.26. Token 246 “dedi” has “de[Verb]+[Pos]+DH[Past]+[A3sg]” as *rawtag* value.

- *Rawtaglist*: This feature is processed from rawtag. It is made into an easy to use format. Stemtype “N” signifies root of the word, stemtype “-” signifies derivational suffixes, “+” signify inflectional suffixes. “Stem” is the lemma or the suffixes in their text form. This feature has the name “rawtaglist” in the data frame.

  Example 4.2.27. Token 246 “dedi” has “[‘postag’: [‘Verb’, ‘Pos’, ‘Past’, ‘A3sg’], ‘stemtype’: ‘N’, ‘stem’: ‘de’, ‘postag’: [‘Past’, ‘A3sg’], ‘stemtype’: ‘+’, ‘stem’:

'DH']" as *rawtaglist* value.

- *Rawword*: This is the token exactly as obtained from the news text file. This feature has the name "rawword" in the data frame.

  Example 4.2.28. Token 246 "dedi" has "dedi" as *rawword* value.

- *Rightwordpostag*: This is the postag of the right neighboring token with respect to the token.

  Example 4.2.29. Token 246 "dedi" has "Punc" as *rightwordpostag* value.

- *Sibling Words*: This is the list of child tokens of the parent word of the given token. The token itself is also included in the list of sibling words, so if the token is the root of the dependency tree, it only has itself as its sibling word value.

  Example 4.2.30. Token 246 "dedi" has "dedi" as *sibling_words* value.

- *Stem*: This feature is the output of the morphological parser. It is the root of the token. It is sometimes called "Lemma" as well however this feature has the name "stem" in the data frame.

  Example 4.2.31. Token 246 "dedi" has "de" as *stem* value.

- *Trigram*: Trigrams are calculated by concatenating the left and right neighboring tokens of the token. This feature has the name "trigram" in the data frame.

  Example 4.2.32. Token 246 "dedi" has "önemli dedi ." as *trigram* value.

- *Unipostag*: This feature is taken from the dependency parser. It is mostly the same as the postag feature, so it is not used in the experiments at all. This

feature has the name “unipostag” in the data frame.

Example 4.2.33. Token 246 “dedi” has “VERB” as *unipostag* value.

- *Word*: We transform the token into all lowercase to obtain the word feature.

  Example 4.2.34. Token 246 “dedi” has “dedi” as *word* value.

- *Wordshape*: This feature is extracted from the rawword feature. Lower case letters are replaced by a lowercase x, uppercase letters are replaced by uppercase X, and the digits are replaced by d. The rest of the characters are left untouched. This feature has the name “wordshape” in the data frame.

  Example 4.2.35. Token 246 “dedi” has “xxxx” as *wordshape* value.

4.2.3. Boolean Features

There is for the time being only one boolean feature: “is_title”. Most features can be casted into a boolean feature (like “is_parentword”), however this is the only boolean feature that we explicitly wanted to make use of.

- *Is Title*: It is the indication that whether the token is in title. The first line of each news text contains a title. This feature has the value True if the token is part of the title, i.e. the first line, False otherwise. Since the main event is sometimes mentioned in the title, we thought this can be an interesting feature to make use of. Only the titles on the first line of the news texts are considered for this feature. Subtitles or other titles inside the text are ignored.

  Example 4.2.36. Word 246 “dedi” has “False” as is_title value since it is not part of the title.

4.3. System Pipeline

First of all, our system takes the annotated data set as input. It does some preprocessing and feature engineering on it as explained in Section 3.3 and converts it to a data frame to work with. Word embeddings and all other features are added in this preprocessing step. We make use of this data frame in our experiments.

Secondly, features from the data frame are filtered depending the experiments. If any cutoff value is set to drop tokens with less than a given number of instances, those tokens are dropped at this step.

Thirdly, we encode our categorical features with one hot encoding. We use label encoder to encode our labels. After feature encoding, we split our data frame into training and test sets. Then, oversampling is applied with Smote.

Now that our training and test sets are all ready to work with, different experiments and fine tuning is done in this step. Grid search cross validation is used for hyperparameter tuning. After fine-tuning, the experiments are run with the best parameters. After the model is trained on the training set, the model is tested with the test set that was split in the beginning. The results are reported in terms of precision, recall and macro-averaged f1 score.

Our pipeline is shown in Figure 4.1.

4.4. Model Description

We tried different machine learning models to find the best-fitting model for the event nugget detection in Turkish. Since there is no previous research for this task in Turkish, we aimed to build a strong baseline. We also developed another model for event type classification. After event nugget classification, we then had two other test tasks: realis classification and main event binary classification.

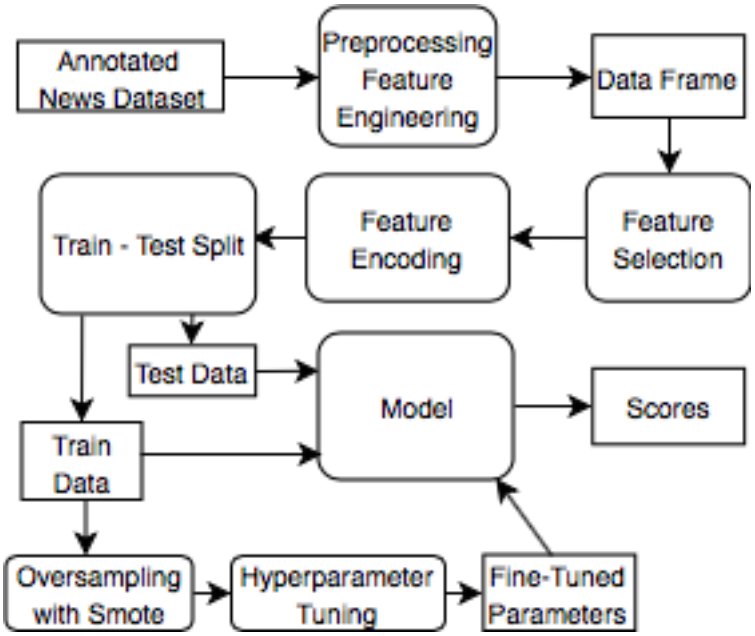

Figure 4.1. Our System Pipeline.

As our pipeline of event nugget detection and classification we used sequence type classification to obtain the nugget spans to detect the event nugget fully with all its words. Then we tried to classify all tokens into different event types, with non-nuggets having the event type "None".

In our sequence classification, we aimed to test different things. First of all we defined different sets of features for different experiments. We started with the basic features and found the best working model on these features. Then we worked on the best performing model for the basic features with adding more features to see the effect of Turkish language specific features. These added features include morphological features as well as features obtained from the dependency parser. We then run a final experiment on all the models with all the features. We made use of different models such as logistic regression, random forest classifier and extreme gradient boosted trees (xgboost). In all our models, we made use of [27] except for xgboost, which was developed in [19].

As described in the Subsection 3.3, the data is highly imbalanced. To counter the effects of this, we made use of oversampling by Smote [28] when applicable. Sometimes by using oversampling, the data size became unmanageable, so oversampling could not be used. Also some experiments were run using only word embeddings with basic features, and some run by only one hot encoded words with basic features. The list of feature sets used in the experiments is shown in Table 4.1.

Table 4.1. Feature Sets for Experiments.

Feature Set Name	**Features**
Basic Features	is_title, sentence_number, wordshape, token_no_in_sentence
Word Features	bigraml, word, l2neighbor, l1neighbor, r2neighbor, r1neighbor,
Word Embedding Features	embedding
Morphological Features	postag, stem, unipostag, rightwordpostag, leftwordpostag
Dependency Features	child_words, dependency_relation_label, distance_to_root, parent_word, sibling_words
All Features	Basic Features + Word Features + Morphological Features + Dependency Features

5. EXPERIMENTS AND RESULTS

We have done numerous experiments with different sets of features and with different models with different parameters. In this section, we will describe the most significant ones and show the results of these experiments. All experiments are done over the same tokens in the dataframe.

In evaluation of all experiments, we calculated scores according to the greedy mapping algorithm given by TAC KBP Scoring Guidelines [29]. We first determined the predictions from the model, then we formed our nugget candidates from the tokens. For all pairs of predicted token labels (system list) and actual test token labels (gold list) we calculated the dice coefficient score (DCS) according to Eq. 5.1. For each pair of dice scores, we find the pair with the maximum dice score and add it to a mapping of dice score and system predicted nugget. After finding all the mapping elements, we then calculated True Positives (TP) by adding dice scores of each element in the mapping.

The span only greedy mapping algorithm that we utilized is described in Figure 5.1. Gold standard mentions are denoted by G and system mentions are with S. T_S and T_G are used to represent the tokens of the mention. Dice scores of all pairs of tokens of nuggets in gold list and system list are calculated and added to a list L before the algorithm starts. DCS returns scores between 0 and 1.

In all the experiments we have reported precision, recall and f1 score, which are respectively calculated using the formulas in Eq 5.2, Eq 5.3 and Eq 5.4.

$$\mathbf{DCS} = \frac{2\,|\mathbf{X} \cap \mathbf{Y}|}{|\mathbf{X} \cup \mathbf{Y}|} \tag{5.1}$$

Input: A list L of scores for all pairs of G,S calculated using Eq. 5.1;

Input: A list GL of gold standard mentions;

$M \leftarrow \emptyset$; $U_s \leftarrow \emptyset$; $U_g \leftarrow \emptyset$; $TP \leftarrow 0$;

while $L \neq \emptyset$ **do**

  $G_m, S_n \leftarrow argmax_{(G,S)\in L} DCS(T_G, T_S)$;

  $L \leftarrow L - \{DCS(T_{G_m}, T_{S_n})\}$;

  if $S_n \notin U_s$ **and** $G_m \notin U_g$ **and** $DCS(T_{G_m}, T_{S_n}) > 0$ **then**

    $M_{G_m} \leftarrow (S_n, DCS(T_{G_m}, T_{S_n}))$;

    $U_s \leftarrow U_s \cup \{S_n\}$;

    $U_g \leftarrow U_g \cup \{G_m\}$;

  end if

end while

for $G \in GL$ **do**

  $(S, DCS) \leftarrow M_G$;

  $TP \leftarrow +DCS$;

end for

Output: TP

Figure 5.1. Span Only Greedy Mapping Algorithm adopted from [29].

$$\mathbf{P} = \frac{TP}{|Systemlist|} \tag{5.2}$$

$$\mathbf{R} = \frac{TP}{|Goldlist|} \tag{5.3}$$

$$\mathbf{F1} = \frac{2PR}{(P+R)} \tag{5.4}$$

First of all, let us talk about our main task, i.e. event nugget detection. As mentioned earlier all event nugget detection experiments (Experiments 1-5) are handled as sequence classification problem. There are five labels for classification, which are "B", "I", "L", "O", "U". It makes the random guess probability 20%. The experiments are mostly run on three different fine-tuned models. These models are logistic regression, random forest classifier and XGBoost. As for metrics, precision, recall and macro-averaged f1 score are reported in the corresponding tables. These scores are calculated on nuggets rather than tokens according to the span only greedy mapping algorithm, so all the tokens in the nugget should be identified correctly to get perfect score.

Experiment 1 is run for detection of event nuggets with basic feature set and word features, with and without oversampling. Sequence classification is used for this experiment. The experiments are run on three different fine-tuned models. These models are logistic regression, random forest classifier and XGBoost. As for metrics, precision, recall and macro-averaged f1 score are reported. The results are shown in Table 5.1. The models that run with oversampling are noted with Smote inside parentheses. Since this experiment is run with the most basic features, we did not get high results. Since the feature set is too basic, the effects of oversampling is not much visible. In Experiment 1 with oversampling, the highest scoring model is Logistic Regression but XGBoost is also close and has higher precision. Without oversampling, random forest classifier has worse results with the basic feature set with word features. It could not predict any nugget correctly, so the score is zero.

Table 5.1. Experiment 1 Results for Event Nugget Identification with Basic + Word Features.

Model Name	**Precision**	**Recall**	**F1score**
Logistic Regression (Smote)	0.62	0.17	0.27
Random Forest Classifier (Smote)	0.08	0.61	0.14
XGBoost (Smote)	0.29	0.21	0.25
Logistic Regression	0.61	0.18	0.28
Random Forest Classifier	0	0	0
XGBoost	0.46	0.12	0.19

Experiment 2 uses basic features and word embeddings with and without oversampling. Oversampled runs are again marked as Smote inside parentheses. The results are shown in Table 5.2. There is a significant increase in f1 scores. The highest scoring model turned out to be XGBoost with this feature set. We see that oversampling has a slight positive effect, so for the sake of simplicity, we will, from now on, report experiment results only over the oversampled training data.

Table 5.2. Experiment 2 Results for Event Nugget Identification with Basic + Embedding Features.

Model Name	**Precision**	**Recall**	**F1score**
Logistic Regression (Smote)	0.14	0.60	0.23
Random Forest Classifier (Smote)	0.20	0.54	0.29
XGBoost (Smote)	0.47	0.43	0.45
Logistic Regression	0.28	0.45	0.35
Random Forest Classifier	0.54	0.33	0.41
XGBoost	0.50	0.39	0.44

Experiment 3 uses different sets of features with Logistic Regression with oversampling. The results are shown in Table 5.3. Some combinations of features are tested

in this experiment to see the effects of features in a given model. Basic feature set combined with word features is tested first. Then we added morphological features to this set and see a significant increase. When we added dependency features to the basic feature set combined with word features, we also see an increase but not as dramatic as with the morphological feature set. Then, when we used the all features set the f1 score has improved and we achieved the best results. Adding word embedding feature to the all features set does not seem to improve the result in logistic regression.

Table 5.3. Experiment 3 Results for Event Nugget Identification with Logistic Regression.

Features Used	**Precision**	**Recall**	**F1score**
Basic + Word	0.14	0.60	0.23
Basic + Word + Morphological	0.55	0.42	0.48
Basic + Word + Dependency	0.71	0.28	0.40
All Features	0.56	0.43	0.49
All Features + Word Embedding	0.57	0.42	0.48

Experiment 4 is done for Random Forest Classifier with different feature combinations with oversampling. The results are shown in Table 5.4. Similar to the Experiment 3, basic feature set combined with word features is tested first. Then we added morphological features to this set and see a significant increase. In both models, logistic regression and random forest classifier, f1 score has doubled when we added the morphological features. After the dependency features were added to the basic feature set combined with word features, we also see an increase but again not as dramatic as with the morphological feature set. Then we used the all features set, and we achieved similar f1 score as with adding only morphological features to the basic word feature set. Surprisingly, adding the word embedding feature to the all features set results in notable decrease in the f1-score.

Experiment 5 is done for XGBoost classifier with different feature combinations with oversampling. The results are shown in Table 5.5. Similar to the Experiment

Table 5.4. Experiment 4 Results for Event Nugget Identification with Random Forest Classifier.

Features Used	**Precision**	**Recall**	**F1score**
Basic + Word	0.08	0.61	0.14
Basic + Word + Morphological	0.21	0.45	0.29
Basic + Word + Dependency	0.14	0.50	0.22
All Features	0.22	0.45	0.29
All Features + Word Embedding	0.15	0.52	0.23

3, basic feature set combined with word features is tested first. Then we added morphological features to this set and see a significant increase. After the dependency features were added to the basic feature set combined with word features, we see a slight increase. Then we used the all features set, and the f1 score has improved but adding embeddings to all features yields the best results in XGBoost.

Table 5.5. Experiment 5 Results for Event Nugget Identification with XGBoost.

Features Used	**Precision**	**Recall**	**F1score**
Basic + Word	0.29	0.21	0.25
Basic + Word + Morphological	0.54	0.35	0.42
Basic + Word + Dependency	0.66	0.17	0.27
All Features	0.56	0.33	0.41
All Features + Word Embedding	0.55	0.40	0.46

As observed in all the experiments up until now, our best performance for event nugget identification is achieved with logistic regression using all features except embeddings. In the rest of the experiments, we first run this model to get the event nuggets, then we run all three models with all feature set and word embeddings to predict the test labels, namely event type, realis value and main event value. We then use attribute augmented greedy mapping algorithm to calculate the scores. It is almost

the same as the previous span only greedy mapping algorithm, the difference is that the mapping elements list is constructed with elements if the predicted label and the real label of the element have the same value. The algorithm is described in Figure 5.2.

Input: A list L of scores for all pair of G,S calculated using Eq. 5.1.
Input: A list GL of gold standard mentions;
$M \leftarrow \emptyset$; $U_s \leftarrow \emptyset$; $U_g \leftarrow \emptyset$; $TP \leftarrow 0$;
while $L \neq \emptyset$ **do**
  $G_m, S_n \leftarrow argmax_{(G,S)\in L} DCS(T_G, T_S)$;
  $L \leftarrow L - \{DCS(T_{G_m}, T_{S_n})\}$;
  if $S_n \notin U_s$ **and** $G_m \notin U_g$ **and** $DCS(T_{G_m}, T_{S_n}) > 0$ **then**
    if $A_{S_n} = A_{G_m}$ **then**
      $M_{G_m} \leftarrow (S_n, DCS(T_{G_m}, T_{S_n}))$;
      $U_s \leftarrow U_s \cup \{S_n\}$;
      $U_g \leftarrow U_g \cup \{G_m\}$;
    end if
  end if
end while
for $G \in GL$ **do**
  $(S, DCS) \leftarrow M_G$;
  $TP \leftarrow +DCS$;
end for
Output: TP

Figure 5.2. Attribute Augmented Greedy Mapping Algorithm adopted from [29].

In Experiment 6 we tackled the event nugget type classification problem. We set a cutoff value of 10, which means event nugget subtypes with less than 10 instances are to be dropped. This leaves us with 27 different subtypes to classify for. In this experiment all features with embeddings are used with oversampling. The results are

shown in Table 5.6. Logistic regression yields the best results.

We also wanted to see how well we classify event main types when the model is trained with event subtypes. The result of this experiment with logistic regression has an f1 score of 0.43 (with a precision of 0.44 and recall of 0.42) which is surprisingly slightly less than that of subtype classification.

Table 5.6. Experiment 6 Results for Event Nugget Type Classification.

Model Name	**Precision**	**Recall**	**F1score**
Logistic Regression	0.50	0.47	0.48
Random Forest Classifier	0.47	0.44	0.45
XGBoost	0.49	0.46	0.47

As our side task, in Experiment 7 we tested realis value classification. In this experiment all features with embeddings are used with oversampling. All three models are used for this experiment. The results are shown in Table 5.7. Logistic regression has the best results.

Table 5.7. Experiment 7 Results for Realis Value Classification.

Model Name	**Precision**	**Recall**	**F1score**
Logistic Regression	0.38	0.36	0.37
Random Forest Classifier	0.35	0.34	0.35
XGBoost	0.33	0.31	0.32

In Experiment 8 we tested main event classification. In this experiment all features are used with oversampling. The results are shown in Table 5.8. Random forest classifier yields the best results.

Table 5.8. Experiment 8 Results for Main Event Classification.

Model Name	**Precision**	**Recall**	**F1score**
Logistic Regression	0.33	0.30	0.32
Random Forest Classifier	0.35	0.33	0.34
XGBoost	0.31	0.28	0.29

6. CONCLUSION

Event nugget detection and type identification in Turkish texts has yet an unlocked potential to be discovered. In this thesis, we have attempted to explore the world of machine learning methods for event nugget detection and nugget type classification in Turkish texts for the first time to set a strong baseline for future research. We have tried different applications of methods and evaluated a diverse range of techniques for the problem. We tested the effects of different types of fea- ture sets to see if Turkish-specific morphological and dependency related features help to improve our results, and we have seen that they were quite effective.

We have also gathered a new manually annotated data set for event nugget detection, nugget type classification, realis value classification and main event detection. The data set is made publicly available. Since there has not ben any data set up until now, this data set will now open door to more research.

6.1. Future Work

New approaches can be taken to further improve the results that we have obtained. More feature engineering can be done, if in the future there are more resources available for Turkish, this may enable us to make full use of better trained word embeddings, and tools to explore morphological and semantic aspects of the Turkish language.

More data collection and annotation would help as well since due to the fact that some event subtypes did not have any instances, we could not run event nugget detection on all the 38 given subtypes. Since we randomly selected the news not to be biased about the news and the events mentioned in the news, imbalances in the event types are natural. We did not want to change the nature of the news, so we did not handpick news to fit better into our research. Also deep learning methods can be applied if a larger data set is annotated.

The data set has been annotated by only one annotator. Annotations by multiple annotators will help us evaluate the cross-annotator agreement, so that a more reliable data set is released.

REFERENCES

1. Yang, B. and T. M. Mitchell, "Joint Extraction of Events and Entities within a Document Context", *CoRR*, Vol. abs/1609.03632, 2016.

2. Mitamura, T., Y. Yamakawa, S. Holm, Z. Song, A. Bies, S. Kulick and S. Strassel, "Event Nugget Annotation: Processes and Issues", *Proceedings of the 3rd Workshop on EVENTS: Definition, Detection, Coreference, and Representation at the NAACL-HLT 2015*, pp. 66–76, 2015.

3. Hsi, A., J. G. Carbonell and Y. Yang, "Modeling Event Extraction via Multilingual Data Sources", *TAC*, 2015.

4. Reimers, N. and I. Gurevych, "Event Nugget Detection, Classification and Coreference Resolution using Deep Neural Networks and Gradient Boosted Decision Trees", *Transfer*, Vol. 551, p. 554, 2015.

5. Mitamura, T. and E. Hovy, *TAC KBP Event Detection and Coreference Tasks*, Tech. rep., v2. 1. Technical report, Carnegie Mellon University, 2016.

6. Consortium, L. D., *Rich ERE Annotation Guidelines Overview V4.2*, 2016, `https://tac.nist.gov/2016/KBP/guidelines/summary_rich_ere_v4.2.pdf/`, accessed in June 2017.

7. Chiticariu, L., Y. Li and F. R. Reiss, "Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems!", *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, 2013.

8. Chiticariu, L., R. Krishnamurthy, Y. Li, F. Reiss and S. Vaithyanathan, "Domain Adaptation of Rule-based Annotators for Named-entity Recognition Tasks", *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, 2010.

9. Sammons, M., H. Peng, Y. Song, S. Upadhyay, C.-T. Tsai, P. Reddy, S. Roy and D. Roth, "Illinois CCG TAC 2015 Event Nugget, Entity Discovery and Linking, and Slot Filler Validation Systems", *TAC*, 2015.

10. Ghaeini, R., X. Fern, L. Huang and P. Tadepalli, "Event Nugget Detection with Forward-Backward Recurrent Neural Networks", *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, 2016.

11. Hou, W.-J. and B. Ceesay, "Distributed Neural Embedding for Event Nugget Extraction", *TAC*, 2015.

12. Satyapanich, T. W., T. Finin *et al.*, "Event nugget detection task: Umbc systems", *Proceedings of the 9th Text Analysis Worksho*, 2016.

13. Liu, Z., J. Araki, D. Dua, T. Mitamura and E. H. Hovy, "CMU-LTI at KBP 2015 Event Track", *TAC*, 2015.

14. Nguyen, T. H., A. Meyers and R. Grishman, "New York University 2016 System for KBP Event Nugget: A Deep Learning Approach", *TAC*, 2016.

15. Lu, J. and V. Ng, "UTD's Event Nugget Detection and Coreference System at KBP 2017", *Proceedings of the 2017 Text Analysis Conference, TAC 2017, Gaithersburg, Maryland, USA, November 13-14, 2017*, 2017.

16. Dataset, T. K. ., *TAC KBP 2017 Data*, 2017, `https://tac.nist.gov/2017/KBP/TACKBP17_Eval_License.pdf/`, accessed in June 2017.

17. Jiang, S., Y. Li, T. Qin, Q. Meng and B. Dong, "SRCB Entity Discovery and Linking (EDL) and Event Nugget Systems for TAC 2017", *Proceedings of the 2017 Text Analysis Conference, TAC 2017, Gaithersburg, Maryland, USA, November 13-14, 2017* [30].

18. Raschka, S. and V. Mirjalili, *Python Machine Learning*, Packt Publishing, 2017.

19. Chen, T. and C. Guestrin, "XGBoost: A Scalable Tree Boosting System", *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, ACM, New York, NY, USA, 2016.

20. Mikolov, T., I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality", C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Q. Weinberger (Editors), *Advances in Neural Information Processing Systems 26*, pp. 3111–3119, Curran Associates, Inc., 2013.

21. Pennington, J., R. Socher and C. D. Manning, "Glove: Global vectors for word representation", *In EMNLP*, 2014.

22. Yıldırım, O. and F. Atık, *Kişisel Gazete*, 2013, `http://www.kemik.yildiz.edu.tr/?id=28`, accessed in June 2019.

23. Zhiyi, S., A. Bies, Strassel, T. Riese, J. Mott, J. Ellis, J. Wright, S. Kulick, N. Ryant and X. Ma, "From Light to Rich ERE: Annotation of Entities, Relations, and Events", *In Proceedings of the 3rd Workshop on EVENTS at the NAACL-HLT 2015*, pp. 89–98, 2015.

24. Türk, U., F. Atmaca, Ş. B. Özateş, B. Öztürk, T. Güngör and A. Özgür, "Improving the Annotations of the Turkish Universal Dependency Treebank", *In Proceedings of Universal Dependencies Workshop, SyntaxFest*, 2019.

25. Sak, H., T. Güngör and M. Saraçlar, "Turkish Language Resources: Morphological Parser, Morphological Disambiguator and Web Corpus", *GoTAL 2008*, Vol. 5221 of *LNCS*, pp. 417–427, Springer, 2008.

26. Koksal, A., *Turkish Word2Vec*, 2018, `https://github.com/akoksal/Turkish-Word2Vec`, accessed in June 2019.

27. Buitinck, L., G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Nic-

ulae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project", *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

28. Chawla, N. V., K. W. Bowyer, L. O. Hall and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique", *Journal of Artificial Intelligence Research*, Vol. 16, pp. 321–357, 2002.

29. Language Technologies Institute, C. M. U., *Event Nugget Detection and Coreference Scoring*, 2015, `http://cairo.lti.cs.cmu.edu/kbp/2015/event/Event-Mention-Detection-scoring-v27.pdf`, accessed in June 2019.

30. *Proceedings of the 2017 Text Analysis Conference, TAC 2017, Gaithersburg, Maryland, USA, November 13-14, 2017*, NIST, 2017.