EVALUATION OF COSINE SIMILARITY FEATURE RESULTS WITH DIFFERENT EXPERIMENTAL SETUPS FOR NAMED ENTITY RECOGNITION ON TWEETS

(VARLIK İSMİ TANIMLAMA ÜZERİNE KOSİNÜS BENZERLİĞİ ÖZELLİĞİNİN FARKLI ÖRNEKLEMLERDE DEĞERLENDİRİLMESİ)

by

Onur Büyüktopaç, B.S.

Thesis

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

in the

GRADUATE SCHOOL OF SCIENCE AND ENGINEERING

of

GALATASARAY UNIVERSITY

Supervisor: Prof. Dr. Tankut ACARMAN

June 2019

This is to certify that the thesis entitled

EVALUATION OF COSINE SIMILARITY FEATURE RESULTS WITH DIFFERENT EXPERIMENTAL SETUPS FOR NAMED ENTITY RECOGNITION ON TWEETS

prepared by **Onur BÜYÜKTOPAÇ** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering** at **Galatasaray University** is approved by the

Examining Committee:

Prof. Dr. Tankut ACARMAN (Supervisor)
Department of Computer Engineering
Galatasaray University ----------

Assist. Prof. Dr. Murat Akın
Department of Computer Engineering
Galatasaray University ----------

Assist. Prof. Dr. Cemal Okan Şakar
Department of Computer Engineering
Bahçeşehir University ----------

Date: ----------

ACKNOWLEDGEMENTS

I would like to express my deepest appreciation to Prof. Dr. Tankut Acarman for his patient guidance, enthusiastic encouragement, and useful critiques during the planning and development of this thesis.

I am particularly grateful for the assistance given by Mete Taşpınar. I am very thankful to him for sharing his knowledge about his research. I would also like to thank Dr. Ömer Farukhan Güneş co-founder of the company Oxtractor for sharing their dataset.

Finally, I wish to thank my wife Burcu Kantarcı Büyüktopaç for her support and encouragement throughout my study.

June 2019
Onur BÜYÜKTOPAÇ

TABLE OF CONTENTS

LIST OF SYMBOLS

F **:** F-Measure

FN **:** False Negative

FP **:** False Negative

ML **:** Machine Learning

NE **:** Named Entity

NEEL **:** Named Entity rEcognition and Linking

NER **:** Named Entity Recognition

NLP **:** Natural Language Processing

P **:** Precision

POS **:** Part-of-Speech

R **:** Recall

TN **:** True Negative

TP **:** True Positive

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Today, social media is a huge part of our world and it continues to grow exponentially. Enormous content is being created with these platforms and it draws the attention of people for personal and professional levels. However, extracting meaningful information from this volume of content with human capabilities is not possible. Machine learning approaches are used to discover knowledge with the help of computer power.

Natural Language Processing (NLP) is a branch of artificial intelligence which is focused on interacting humans and computers using the natural language. By the aid of machine learning, NLP can achieve tasks from text such as tokenization, classification, sentiment analysis, Named Entity Recognition (NER). These tasks produce successful results for well-structured texts like newspapers, articles, and books. But, working with unstructured texts from social media is still challenging. These types of texts contain emoticons, abbreviations, grammar mistakes, and code-switching making data unpredictable and dirty.

Twitter is one of the most popular microblog among social media platforms. It provides texts which are publicly posted and contains topic-specific opinions. It is a valuable source for collecting data. On the other hand, the content is unstructured because of character limitation and casual writing.

In this study, we present a NER system and we evaluate baseline classifiers for unstructured texts. We develop a cosine similarity feature, and we evaluate and test each classifier subject to different combinations of features including cosine similarity. Our experimental results show that the presented system is reached at 0.74 level in precision, 0.68 in recall and 0.67 in F1 (micro average), respectively for Named Entity rEcognition and Linking (NEEL) 2016 Challenge dataset. The corpus is created from Twitter.

In addition, we evaluate our system using 2 different datasets with different label distribution and types. One dataset is generated by a startup company called Oxtractor. It has 3 label types; “Person”, “Organization”, and “Location”. Also, we present dataset which is labeled manually from specific topics of tweets. It has 7 types of the label; Person”, “Thing”, “Organization”, “Location”, “Product”, “Event”, and “Character”. We check the prediction results and compare classifiers along with feature sets. Logistic regression, SVM, and Random forest are producing the highest results with cosine similarity feature. The results obtained with different feature sets show that supportive features for cosine similarly do not impact the results significantly. The diversity of named entity is distinctive when working with cosine similarity feature.

Finally, we compare prediction results with different testing/training split ratios for the manually labeled dataset from 90/10 to 50/50. The cosine similarity feature does not affect the split ratio remarkably.

Keywords: Named entity recognition, Information Extraction, Twitter, Word embedding, Classification, Machine learning, Cosine Similarity.

ÖZET

Sosyal medya günlük hayatımızın hızla büyüyen bir parçası olmuştur. Sosyal medya kullanımının artışı ile birlikte her geçen gün muazzam büyüklükte içerik oluşmakta ve bu içerik hem araştırmacıların hem de iş modeli geliştiricilerin dikkatini çekmektedir. Bu boyuttaki veri ile çalışmak ve anlamlı sonuçlar elde etmek için bilgisayarların işlem gücüne ihtiyaç duyulmaktadır. Bu noktada da makine öğrenme yaklaşımları geliştirilerek problemlere çözüm üretilmesi hedeflenmektedir.

Doğal Dil İşleme, yapay zeka uygulamalarının bir alt kategorisidir ve bilgisayar ile insan arasındaki etkileşimi dil üzerinden çözmeye odaklanır. Doğal Dil İşleme'de, Makine öğrenmesi uygulamalarının yardımıyla, metinleri parçalara ayırma, sınıflandırma, duygu analizi yapma, varlık ismi tanımlama gibi işlemler yapılabilmektedir. Gazete, makale, kitap gibi düzgün yapıdaki metinlerde bu çalışmalar başarılı sonuçlar verirken sosyal medyadan elde edilen içerikleri işlemek farklı zorlukları da beraberinde getirmektedir. Bu tarz metinler içerisinde pek çok dil bilgisi hatası, kısaltma, "emoji" ve çoklu dil kullanımı bulundurması sebebiyle öngörülemez.

Twitter en çok kullanılan mikro blog sosyal medya platformlarından biridir. Kişisel metin paylaşımlarının yanı sıra, belli bir konuda ve başlık altında da içerik paylaşımları yapılabilmektedir. Bu yönüyle Twitter değerli ve ilgi çekici bir veri kaynağı haline gelmiştir. Buna karşın karakter kısıtlaması, gündelik dil kullanımı ve "emoji" kullanımı gibi sebeplerden ötürü "tweet" verileri yapısal olarak karmaşıktırlar.

Bu çalışmada, temel sınıflandırma algoritmaları kullanılarak mikro blog verisi üzerinde varlık ismi tanımlama sistemi geliştirilmektedir. Kosinüs benzerliği özelliğini geliştirerek, tüm temel sınıflandırma algoritmaları üzerinde farklı özellik kümeleri ile birlikte uygulanmaktadır. Çalışmalarımızın sonuçları 0,74 hassasiyet, 0,68 duyarlılık ve

0,67 F1 skoru ile Named Entity rEcognition and Linking (NEEL) 2016 Challenge veri kümesine uygulanarak alınmıştır.

İlaveten, çalışmamızı farklı dağılımlarda ve özelliklerdeki 2 veri kümesi üzerinde genişlettik. Birinci veri kümemiz Oxtractor isimli bir start-up firmasına aittir. Veri kümesi "Kişi", "Organizasyon" ve "Konum" bilgi etiketlerini içermektedir. İkinci veri kümesi ise Twitter belli başlıklarda konular üzerinden etiketlediğimiz kendi setimizdir. Bu veri kümesi içerisinde "Kişi", "Varlık", "Organizasyon", "Konum", "Ürün", "Etkinlik" ve "Karakter" gibi 7 bilgi etiketi bulunmaktadır. Elde ettiğimiz çoklu sınıf tahminleme sonuçlarını karşılaştırdığımızda "Logistic regression", "SVM" ve "Random forest" sınıflandırma algoritmalarının yaklaşımımızda en yüksek sonuçları ürettiğini gözlemledik. Farklı özellik kombinasyonlarındaki sonuçlar incelendiğinde ise yardımcı özelliklerin kosinüs benzerliği özelliğinin sonuçlarına kayda değer bir katkısı olmadığı gözlendi. Varlık isim kümesinin çeşitliliği kosinüs benzerliği özelliği için ayırıcı bir faktör olarak görünmektedir.

Son olarak, veri kümelerini 90/10'dan 50/50'ye kadar değişen oranlarda öğrenme/test etme bölümlerine ayırdığımızda kosinüs benzerliği özelliği kullanılan çalışmaların sonuçlarında dikkate değer farkların oluşmadığı gözlemlenmiştir.

1 INTRODUCTION

The growing amount of data stored and shared in social media enables powerful tools to extract information and to discover features. Natural Language Processing (NLP) studies are gaining importance in this context. NLP is the field of study focusing on interacting human language and computer. Understanding natural language as a machine is a general purpose and it can be divided into tasks such as summarization, question answering, translation, and Named Entity Recognition (NER). NER is a task of identifying and categorizing textual contents such as person, thing, organization, location, product, event, and character. If the sentence "Lemmy Kilmister founded Motörhead in 1975." is identified by applying NER task, the sentence would be labeled according to pre-defined classes as:

[Lemmy Kilmister]$_{\text{Person}}$ **founded [Motörhead]**$_{\text{Organization}}$ **in [1975]**$_{\text{Year}}$**.**

While traditional hand-made rule-based NER approaches produce successful results when working with well-structured texts, the prediction scores are much lower on unstructured microblog texts like Twitter. In general, these types of texts contain emoticons, abbreviations, grammar mistakes, and code-switching making data unpredictable and dirty, i.e. difficult to be interpreted by a machine. Machine Learning (ML) approaches are applied to extract features from large scale observational data complicated by unstructured environments to improve the NER methodology. Currently, NER studies are focusing on Deep Learning (DL). However, feature engineering is still important since feature-inferring neural network models outperform state-of-the-art applications according to [1]. In this study, we present an improved approach using syntactic, semantic and domain-specific features while augmenting the data worked on. We evaluate the performance of the feature by altering corpus, feature combinations, and classifiers at each time. We investigate 3 corpora which consist of complete (NEEL) 2016

Challenge dataset with 4369 unique tweets and 7 NER types, dataset published by a startup company Oxtractor in England with 4608 unique tweets and 3 NER types, and a new dataset manually labeled during the study with 3310 unique tweets and 7 NER types.

7 baseline classifiers and 9 features including cosine similarity are evaluated and tested in our study. We have implemented all classifiers with feature combinations over all datasets and we compared the results. We observed that cosine similarity improves F1 statistical metric value and always outperforms other features which are used during the evaluation process.

2 LITERATURE REVIEW

NER has been introduced at the Sixth Message Understanding Conference Sundheim in 1996. NER recognizes entity names such as people, organizations, place names, temporal expressions, and numerical expressions. The first study on NER was carried out by Grishman and Sundheim in 1996 [2]. Early NER research was focussed on handcraft rules, lexicons, orthographic features, and ontologies. Then, neural network NER systems have been presented along with minimal feature engineering which leads to domain-independent systems without lexicons or orthology requirement.

Early researches are based on word-based features ("bag of words"). Bag of Words (BoW) depends on the text describing the occurrence of words within a document. The model focus on only occurrences of known words in the document, but not the location in the document. And, it neither covers the wealth of word knowledge. To overcome these limitations new approaches are proposed with using common-sense and domain-specific knowledge to enrich the BoW [3].

Adapting NER to microblogs has been a challenging task. The classical NER approaches for structured texts were applied to tweets in [4]. An SVM-based classifier for classifying person, location, and organization assured the statistical metric values such as 0.74 precision, 0.49 recall, and 0.59 F1 scores.

Twitter is one of the most successful microblogging services. Researches over data gathered from Twitter are not only limited to the content of tweets. They also consider user networks and profile classifications. For this kind of improved analysis, domain-specific linguistic features should be determined and constructed. As an example of user classification, observable information such as the user behavior, network structure and

the linguistic content of the user's Twitter feed are considered to classify users in 3 categories [5].

Even the domain-specific features implementation techniques serve simplicity and robustness, they are limited in many tasks. To have significant results, they should be supported by new architecture models. Vector representation of words is designed depending on neural network based language models. However, some techniques for measuring the quality of the resulting vector representations do not cover multiple degrees of similarity. word2vec model offers to represent words considering multiple degrees of similarity [6].

This study is an extension of [7] while focusing on the most representative feature set analysis and the improvement of the classifier's performance. The most effective feature in the study [7] is a word embedding based cosine similarity measure. For cosine similarity, each labeled word represented as a vector using the precomputed word2vec model and averaging all the vectors belonging to a particular label type. As a word2vec model, the study used a corpus of 400 million tweets [8]. Despite the large word2vec corpus, some of the words cannot be represented as a vector. These words are extracted from the dataset. The approach applied to NEEL 2016 dataset [9] with logistic regression classifier and achieved 0.71 precision, 0.56 recall, and 0.58 F1 score.

The study [7] was selected some of the researches from the NEEL 2016 in order to compare results. The other approaches from the challenge mentioned in [10], [11], and [12].

In [10] a feature-based system combining existing NER systems and domain-specific Part-Of-Speech (POS) tagger is presented. The main idea of the work is to recognize entities and their types from Twitter microposts and link them to another corresponding dataset. There are 4 main steps followed in this work; mention detection, mention type classification, mention linking, and NIL clustering. They develop a hybrid system by using Stanford Named Entity Recognizer and ARK Twitter Part of Speech Tagger approaches and run 3 test scenarios by using various classifiers with different feature sets.

Candidate name generation and classical NER systems such as Stanford NER, MITIE, twitter_nlp, and TwitIE are used in [11]. An adapted Kanopy system for the Twitter domain is implemented in [12]. Both studies are focussed on the solution of the problem of adapting traditional natural language processing to microposts. The approaches consist of having two pipelines where on the one pipeline the linked entity mentions are processed.

3 MACHINE LEARNING

In this section, we describe the state-of-the-art machine learning methods, features, and classifiers used in our study. We first explain the machine learning concept. In the following, we represent the features which are used with cosine similarity. Finally, we elaborate on the classifiers.

3.1 State-of-the-art

3.1.1 Definition of Machine Learning

ML is a method that automates rational decisions. Instead of explicitly programming, machine learning can recognize meaningful patterns from given data with the training process and adapt them to the current problem. Pattern recognition can be achieved by two main techniques; supervised and unsupervised learning.

With supervised learning, the system has both the input variables (X) and the output variable (y). Each input is labeled with the desired output with a mapping function. The formula (1) can describe the supervised learning.

$$y = f(X) \tag{1}$$

Unsupervised learning, however, only knows the input variable. The goal of the system is developing and organizing the data and finding the underlying structure of it.

Machine learning can be used for two types of prediction; classification and regression. Classification predicts discrete numbers. Output variables are often called label or class.

Regression estimates the continuous quantity. Output values can be an integer or a floating number.

There are various domains using machine learning. Recommendation, recognition, route prediction is the most used solutions. Google offers, Netflix video recommendation, Spotify daily mixes, Amazon related products are the recommendation examples. Siri and Cortana are voice recognition solutions. Facebook uses DeepFace algorithm to tag people from photos. Microsoft Kinect human pose recognition algorithm runs with random forest. Google Maps and Uber calculate the fastest routes and arrival time with machine learning. Fraud detection that PayPal uses is also an ML solution.

3.1.2 Named Entity Recognition

NER is a machine learning application area which is used for identifying pre-defined nouns (person, location, organization, etc.) in a given text.

NER is an important area in ML since it is a useful tool for many real-world solutions such as classifying print media and social media contexts, content recommendation, efficient search algorithm, customer support, chatbots.

NER systems for structured texts is a well-studied domain, and they can perform near-human results particularly in English [13]. However, prediction scores of informal and noisy texts such as social media posts, are much lower than formal texts.

3.2 Features

In [7], the authors have presented an approach aiming to identify different classes of named entities in short and noisy texts, mainly tweets, with simple but fast and effective supervised machine learning approach by using word embedding features. The study [7] has evaluated 6 features for the feature vector which were, “hashtag”, “at’”, “capital letter”, “all capital”, “part-of-speech (POS) tagger”, “the similarity to the class centroid (cosine similarity)”.

In addition to the features, we include 3 additional features. These are, “next word POS tagger”, “previous word POS tagger”, and “position”. Each feature represented by a numeric or Boolean value in the feature vector. The feature details are explained below. The abbreviations of the features are noted in parenthesis at titles.

Hashtag (#). The feature searches whether any word of a named entity is hashtagged, written with “#” symbol. The hashtag symbol is used on Twitter to index and highlight topics and keywords. Hashtagged keyword does not contain any space or punctuation and can be included anywhere in a tweet. The feature returns a Boolean value.

At (@). The feature has the same process with the hashtag but it seeks ‘at’ sign, written as “@” symbol, instead of “#”. At sign is used for addressing another Twitter user. Twitter usernames can only contain alphanumeric characters and underscore which guarantees mentioned user belongs in NE or not. It returns a Boolean value.

Capital letter (title). The feature checks whether all words in named entity start with an upper-case letter and the rest of them are lower-case. Symbols and numbers are ignored. It returns a Boolean value.

All capital (all_capital). The feature checks whether all letters of the named entity are upper-case. Symbols and numbers are also ignored. It returns a Boolean value.

Part-of-Speech (POS) Tagger (pos). The feature assigns a part-of-speech tag to each word of NE such as “noun”, “verb”, “adjective”. We use Stanford POS tagger [14] for this task. We tag each word of NE separately and check whether all words have the same tag. If they all have the same POS tag, then NE is assigned with the tag. Otherwise, the NE is labeled as “mixed POS”. To apply the results to the feature vector, we mapped the tags with numeric values.

Next word POS Tagger (next_pos). The feature assigns a POS tag the following word after NE. If NE is the last word then the feature returns “0”.

Previous word POS Tagger (prev_pos). The feature assigns a POS tag the previous word of NE. If NE is the first word then feature returns "0".

Position ratio (position). The feature presents the position of NE in a tweet. It splits tweet by space and indexes them starting from 1. The first word of NE is accepted as an index of NE. Comparing indexes from different tweets is inconsistent because the index is depended on tweet length. Hence, we normalized the index and define a relative position as shown in (2).

$$position\ ratio = \frac{index\ of\ the\ first\ word\ of\ NE}{Total\ word\ count} \tag{2}$$

The similarity to the class centroid (cos). The feature computes the cosine similarity between NE's vector and the centroid vector of each NER type. All words present in NE's are defined as a vector using the precomputed word2vec model, the corpus of 400 million tweets [8]. Each NE is represented by a single vector computed by the weighted average of all word vectors within.

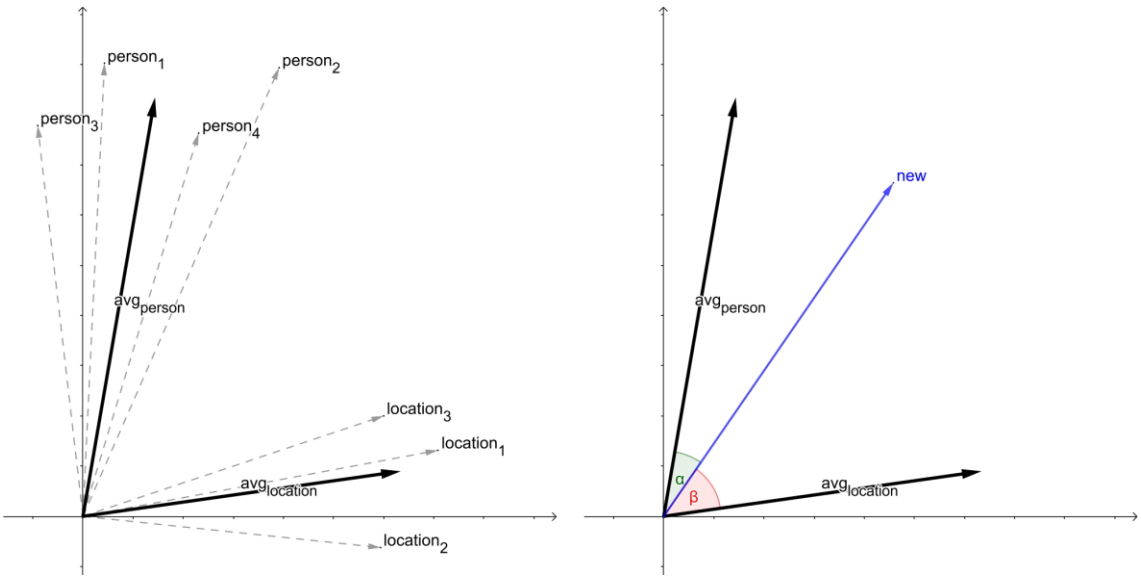

Figure 3.1: an example of the "cos" feature

An example of "cos" feature is represented in Figure 3.1, NE vectors are grouped by NER types to calculate the average vectors of "Person" and "Location". Each NE has a distance

degree between each average vector. For this example, new NE has 2 hidden features as a part of the "cos" feature; one for "avg_{person}" and one for "$avg_{location}$". $X_1 = \cos(\alpha)$ and $X_2 = \cos(\beta)$ are calculated while preparing the feature vector.

In this study, we use 7 NER types. Therefore, similarity to class centroid consists of 7 hidden features. Each hidden feature returns a numeric value.

3.3 Classifiers

We use 6 supervised baseline classification algorithms;

- Logistic Regression,
- Support Vector Machines (SVM) with "RBF" and "linear" kernels,
- k-Nearest Neighbors (k-NN), k is 5,
- Naive Bayes (NB) with Gaussian distribution,
- Decision Tree with "Gini index" and
- Random Forest with 100 estimator trees as parameters.

3.3.1 Logistic Regression

Logistic regression is a statistical method for analyzing datasets. The method generates a coefficient for each feature as an independent variable and predicts the probability of belonging of the feature to a single NER type which is a dependent variable [15].

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_i x_i \quad (3)$$

In the simplified version of logistic regression, shown in (3), "*y*" is the dependent variable, "*β*" is the coefficient of the independent variable, "*x*" is the independent variable, and "*i*" is the number of independent variables.

Logistic regression provides a useful means for modeling the dependence of response variable on features. However, it requires a large size of samples. Also, independent

variables should not be tightly correlated, and an extensive set of features can cause overfitting problem.

3.3.2 Support Vector Machines

SVM is a supervised machine learning algorithm aiming to define the maximum separation between two classes [16]. The idea is finding the NER type instances which are the most likely to the other type, to draw a boundary in-between. These instances are called support vectors. The hyperplane which leaves the maximum margin from the support vectors is named as the decision boundary.

Simple SVM can be applied to linearly separable data. In order to use SVM on non-separable models, a new dimension (deterministic feature) is included in the model to make the data separable. The method is known as “kernel trick.” In our study, we use two different kernel options which are “RBF” and “linear.”

SVM can deal with a large number of features but mapping a higher dimensional space causes highly intensive computation.

3.3.3 K-Nearest Neighbor

k-NN is a simple instance based, lazy-learning algorithm. The algorithm assumes that there is a proximity between the subjects of the same type within a dataset. The prediction of the type of input is determined by calculating the average of the closest k training instances [17]. Selecting k is crucial since it affects the predictions directly. For noisy datasets, k should be high enough to eliminate noise. For small type instance sets, k should be low to prevent different class instances. We use k as 5.

Although the algorithm does not require calculation and generalization for training, it has to store the whole train set. As a result, it needs more storage then computation power.

3.3.4 Decision Tree

Decision tree learning is a logic-based tree algorithm. In order to construct a decision tree, based on feature values, instances are split and sorted accordingly. Each node in the tree represents a feature, and each branch represents a split condition [18].

There are two critical points for an optimal decision tree; the selection order of features and the quality of dividing values. Decision tree uses split criterion heuristics with the intention of the well-constructed tree. We choose “Gini index” for split criterion as our features produce continuous values, and they can be split into several conditions.

3.3.5 Random Forest

Random forest is an ensemble of n decision trees. Each tree uses a random sub-sample of a dataset. Complete forest votes for the prediction and average of them used as a final prediction [19]. A decision tree is heavily dependent on data distribution. Only one tree can cause an over-fitting problem. Using multiple decision trees with random samples improves accuracy and controls over-fitting. We use 100 estimator trees in our experiments.

3.3.6 Naïve Bayes

Naïve Bayes is a learning technique based on Bayes’ theorem with the independence of predictors [20]. It is called “naïve” because classifier assumes that all the features are independent. Naïve Bayes calculates the likelihood of an input belonging to each type to find out the highest probability.

As the assumption is not correct for most of the feature sets, it generally produces less accurate results than sophisticated classifiers. Despite the low accuracy rate, the primary advantage of the algorithm is short computational time for training.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \quad (4)$$

General Bayes formula is given in formula (4) where, *P (A|B)* indicates the posterior probability, *P (B|A)* does the likelihood, *P (A)* does prior probability, and *P (B)* does the marginal likelihood. We use Naïve Bayes classifier with Gaussian likelihood mean in our study.

4 NER EVALUATION

4.1 Datasets

We consume Named Entities (NE) from 3 different corpora with different layouts for the study. For unifying the layouts, we create a common format. The common format is tab-separated and it stores the “tweet id”, “tweet text”, “NE starting index”, “NE ending index”, “NER type” and “NE” information as follows:

```
[tweet_ID] [text]  [NE_starting_index]  [ne_ending_index]  [NER_type]
[NE]
```

One tweet can contain multiple NEs but all NEs are stored individually.

4.1.1 NEEL 2016 Challenge

For benchmarking purposes, we use a publicly available dataset provided by NEEL 2016 Challenge [9]. The source contains only labeled word information and tweet id. The template of the source file can be described as follows:

```
[tweet_ID] [word_start_index] [word_end_index]   [word_DBpedia_link]
[confidence_score] [NER_type]
```

For instance:

```
674869443671941120  93 101    http://dbpedia.org/resource/Egyptians
1  Thing
```

Since some features require the content of tweet to calculate its score, we should merge the information above with the tweet. Finding tweets from the tweet ID is a challenging

task which is also mentioned in [7] because either some tweets were already deleted or they are private. Therefore, we cannot retrieve tweets from Twitter. Instead, we have found all tweets from an open source project from GitHub called Webpack Bundle Analyzer [21].

The collected tweets are harmonized and merged with the NEEL dataset and stored in a format where each sample contains the following information: tweet identifier, tweet, start index of the word, end index of the word, NER type and the word. The resulting form is illustrated as follows:

```
674869443671941120  RT @EntheosShines: Just As Some Parents Have A
Favorite Child, Obama Has Favorites (sign at  Egyptian Airport)
@chirofrenzy @PatVPeters htt…|  93 101    ThingEgyptian
```

The corpus contains 4369 unique tweets with 9687 labeled words. It consists of 7 different labels which are “Person”, “Thing”, “Organization”, “Location”, “Product”, “Event”, and “Character”. The split ratio is 0.10 which is constituted by training dataset of 4073 unique tweets and 8665 labeled words and testing dataset of 296 unique tweets and 1022 labeled words.

4.1.2 Oxtractor

The second corpus is provided by a startup company called Oxtractor focussing on social data. The corpus holds several information fields about the tweet including text, id, retweet count, user profile information, media information, language. However, we only focus on the text, id, entities, language, tokens, and annotation offsets as JSON format. We have simplified the properties for our study and the data structure of a tweet has become the following structure:

```
{
   "text": "Ukraine's pro-Russia rebels hand over Malaysia Airlines
           #MH17's black boxes http://t.co/sWs4wDau3m
           http://t.co/9GyZCurIkM",
   "id": 491401326845510000,
   "entities": ["B-loc", "O", "O", "B-loc", "O", "O", "O", "B-org",
                "I-org", "O", "O", "O", "O", "O"],
   "lang": "en",
   "tokens": ["Ukraine", "'s","pro-", "Russia", "rebels", "hand",
              "over", "Malaysia", "Airlines", "#MH17's", "black",
```

```
              "boxes", "http://t.co/sWs4wDau3m",
              "http://t.co/9GyZCurIkM"],
   "annotation_offsets": [[0,7], [7,9], [10,14], [14,20], [21,27],
                          [28,32], [33,37], [38,46], [47,55],
                          [56,61], [64,69], [70,75], [76,98],
                          [99,121]]
}
```

As it is seen in the sample, BIO encoding was applied for tokenizing. There are four labeled words and three named entities. “B” key represents the beginning of a named entity, and “I” key represents inside of a named entity. We have applied the tab-separated format; then the example transforms into three separate samples for our model.

```
491401326845510000 Ukraine's pro-Russia rebels hand over Malaysia
Airlines #MH17's black boxes http://t.co/sWs4wDau3m
http://t.co/9GyZCurIkM  0  7  Location   Ukraine

491401326845510000 Ukraine's pro-Russia rebels hand over Malaysia
Airlines #MH17's black boxes http://t.co/sWs4wDau3m
http://t.co/9GyZCurIkM  14 20 Location   Russia

491401326845510000 Ukraine's pro-Russia rebels hand over Malaysia
Airlines #MH17's black boxes http://t.co/sWs4wDau3m
http://t.co/9GyZCurIkM  38 55 Organization Malaysia Airlines
```

The corpus has only 3 label types; “Person”, “Organization”, and “Location”. It contains 4608 unique tweets with 8264 labeled words. We have split the data into training and testing by the same ratio used in the previous dataset. Finally, the training dataset consists of 4056 unique tweets with 7395 labeled words and the testing dataset is constituted by 552 unique tweets with 869 labeled words.

4.1.3 Manually labeled

The third corpus has been created manually during the period of 4 months starting July 6th, 2018 and ending October 21st, 2018. We have collected English tweets posted in 2018 from Twitter Search API[1]. Keywords cover multiple memorable events mentioned in this period including

- “World Cup 2018”,
- “U.S. China trade war”,

[1] https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets.html

- “Death of Anthony Bourdain”,
- “Wacken 2018”,
- “Syrian refugee”,
- “Climate Change”,
- “Marvel”,
- “Avengers”.

While the API provides much information about tweet itself, we only consider “id”, “text”, and “lang” properties. The basic version of the data is given below:


```
{
   "statuses":[
      {
         "id": 1014639601313636352,
         "text": "4 of 5 stars to Kitchen Confidential by Anthony
Bourdain                          https://t.co/6wpo4qZHIG",
         "lang": "en"
      }
   ]
}
```


We exclude retweets by a script and we enrich the data with tokenized sentence information:


```
{
   "id": 1014639601313636400,
   "text": "4 of 5 stars to Kitchen Confidential by Anthony Bourdain
https://t.co/6wpo4qZHIG",
   "tokens": ["4", "of", "5", "stars", "to", "Kitchen",
"Confidential", "by",                    "Anthony", "Bourdain",
"https://t.co/6wpo4qZHIG"]
}
```


Furthermore, we filter the auto-posts from such as news channels and YouTube manually. Then we label the tweets according to 7 label types which are “Person”, “Thing”, “Organization”, “Location”, “Product”, “Event”, and “Character”. Labeling process has been handled manually by using a custom developed tool named Twitter Tagger. The user interface of the tool is presented in Figure 4.1.

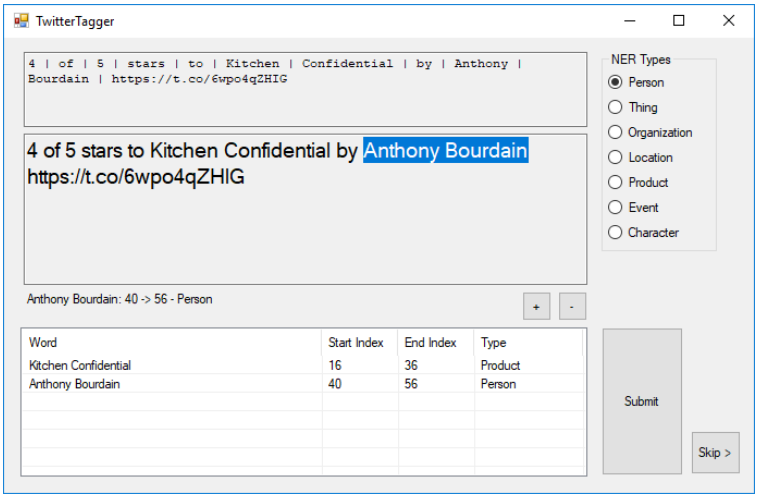

Figure 4.1: Twitter Tagger tool interface

After labeling, the raw data is transformed into a tab-separated format which is compatible with our model:

```
1014639601313636400  4 of 5 stars to Kitchen Confidential by Anthony
Bourdain https://t.co/6wpo4qZHIG  16 36 Product Kitchen Confidential

1014639601313636400  4 of 5 stars to Kitchen Confidential by Anthony
Bourdain https://t.co/6wpo4qZHIG  40 56 Person  Anthony Bourdain
```

There are 3310 unique tweets and 8339 labeled words in the last dataset. We keep the split ratio close to the other datasets. Training dataset contains 3048 unique tweets and 7505 labeled words, and the testing dataset contains 262 unique tweets and 834 labeled words.

In addition, we split the manual labeled corpus by 80/20, 70/30, 60/40, and 50/50 train and test datasets for further evaluation results. The samplings are randomly selected. The distribution of word occurrences is shown in Table 4.1.

Table 4.1: Word occurrences for each split ratio of the manual labeled corpus.

	80/20		70/30		60/40		50/50	
	Training	**Testing**	**Training**	**Testing**	**Training**	**Testing**	**Training**	**Testing**
Person	2932	772	2577	1127	2203	1501	1846	1858
Location	915	231	813	333	710	436	568	578
Organization	1060	290	925	425	776	574	643	707
Product	416	87	348	155	285	218	237	266
Event	704	145	618	231	532	317	456	393
Thing	584	125	506	203	445	264	377	332
Character	61	17	48	30	53	25	41	37
TOTAL	6672	1667	5835	2504	5004	3335	4168	4171

The labeled word occurrences for each dataset are represented in Table 4.2.

Table 4.2: Type occurrences in training and testing datasets.

	NEEL 2016 Challenge		Oxtractor		Manually labeled	
	Training	**Testing**	**Training**	**Testing**	**Training**	**Testing**
Person	2845 (32.83%)	337 (32.97%)	3124 (42.24%)	531 (61.10%)	3338 (44.48%)	366 (43.88%)
Location	1868 (21.56%)	43 (4.21%)	2120 (28.67%)	122 (14.04%)	1023 (13.63%)	123 (14.75%)
Organization	1641 (18.94%)	158 (15.46%)	2151 (29.09%)	216 (24.86%)	1196 (15.94%)	154 (18.47%)
Product	1196 (13.80%)	354 (34.64%)	-	-	458 (6.10%)	45 (5.40%)
Event	482 (5.56%)	24 (2.35%)	-	-	779 (10.38%)	70 (8.39%)
Thing	570 (6.58%)	49 (4.79%)	-	-	644 (8.58%)	65 (7.79%)
Character	63 (0.73%)	57 (5.58%)	-	-	67 (0.89%)	11 (1.32%)
TOTAL	8665 (100%)	1022 (100%)	7395 (100%)	869 (100%)	7505 (100%)	834 (100%)

4.2 Performance Measures

To evaluate the relevance of the results, we choose Precision, Recall, and F1-score. We also use a confusion matrix to visualize a specific outcome.

Before explaining the statistical metric values, we should define true positive, true negative, false positive and false negative. These parameters are the basics of the measurement calculations.

True positive (TP) is the result when the predicted and the actual values are both positive.

True negative (TN) is the result when the predicted and the actual values are both negative.

False positive (FP) is the result when the prediction is positive, but the actual value is negative.

False negative (FN) is the result when the prediction is negative, but actual is positive.
For a multiclass NER problem, the parameters can be defined for a selected class "C" as follows:

- All instances of "C" which are predicted as "C" are TP.
- All instances of "non-C" classes which are predicted as "non-C" classes are TN.
- All instances of "non-C" classes which are predicted as "C" are FP.
- All instances of "C" which are predicted as "non-C" classes are FN.

4.2.1 Confusion Matrix

Confusion matrix also known as error matrix is a table layout to describe the performance of a classification model with comparing actual and predicted results. A simple matrix layout is given in Table 4.3.

Table 4.3: Layout of the confusion matrix

	Predicted Positive	**Predicted Negative**
Actual Positive	TP	FP
Actual Negative	FN	TN

4.2.2 Precision

Precision (P) is the ratio of correct positive results to total positive predictions, as given in (5). It is a good measure to determine when the cost of FP is high.

$$P = \frac{TP}{TP + FP} \tag{5}$$

4.2.3 Recall

Recall (R) is the ratio of correct positive results to total actual positive values, as given in (6). When FN is important for the results, recall is the considered measure.

$$R = \frac{TP}{TP + FN} \tag{6}$$

4.2.4 F1 Score

F1 Score (F1) is the harmonic mean of precision and recall, as given in (7). When we focus on TP and TN values, F1 Score might give a better result with balancing between precision and recall [22].

$$F1 = 2 \times \frac{P \times R}{P + R} \tag{7}$$

The notion of the measurements can be applied only to binary classification problems. Since our classification is multiclass, we practice “micro” and “macro” averaging approaches. Macro averaging ignores the class-based results and calculates metrics by counting values globally. And, micro averaging calculates scores for each class and finds the unweighted mean.

4.3 Approach

Let us describe our approach in this section in terms of steps followed for each dataset. To compare cosine similarity efficiency for each experimental setup, we split all 3 corpora to training/testing datasets with a ratio of approximate 0.9 respectively. We run each corpus with 7 different classifiers with all the combinations of the 9 features. Our experiment environment consists of 10731 test results as described in (8).

$$\begin{aligned} &Corpus\ count\ \times Classifier\ count\ \times\ Feature\ Subsets \\ &= 3 \times 7 \times (2^9 - 1) = 10731 \end{aligned} \tag{8}$$

We consider micro-averaged F1score of all the results to balance between precision and recall. We sort feature subsets by F1 scores for each classifier and average of classifier results in each corpus. Finally, we compare the effects of absence and presence of cosine similarity on F1 scores and investigated the highest and lowest supportive features of cosine similarity.

4.4 Implementation

We implement our algorithm using Python 3.6.3 programming language with the following open source libraries:

- "gensim" [23] for executing word embedding through "word2vec".
- "NumPy" [24] and "pandas" [25] for handling array calculations.
- "scikit-learn" [26] for feature scaling and classifying trained and tested results.
- "nltk" [27] for tokenizing the tweets and the Part-of-Speech (POS) tagging. It provides an adaptor for Stanford POS Tagger, which is written in Java.

In addition, we code Twitter Tagger as a Windows Application with .NET Framework 4.5 (C#.) We inspect results and draw charts using Notepad++ and Microsoft Excel.

Our implementation strategy consists of two steps; building the model and evaluating the model.

4.4.1 Building the model

The stage is dedicated to collecting raw data from various sources, enriching the raw data with additional information and creating the common format toward the evaluation stage. As mentioned in 4.1, each corpus has a different layout with different information hence, we apply separate approaches for each of them. Figure 4.2 shows the overall flow of building each dataset from their corpora.

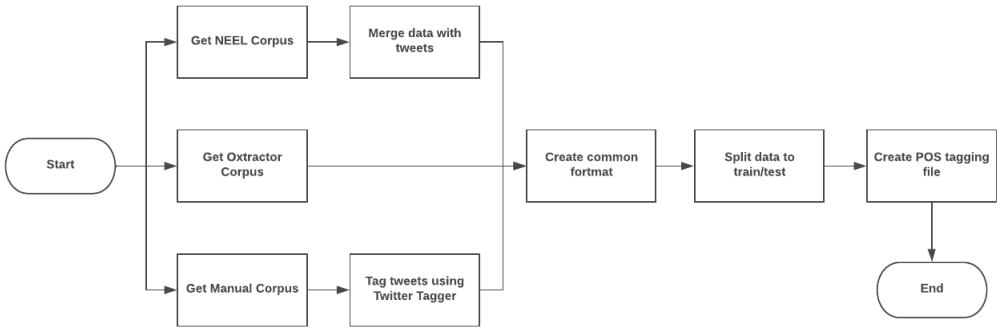

Figure 4.2: the flowchart of the building stage

NEER 2016 files are stored as tab-separated format and tweet texts are not included in the file. The texts are collected from another source [21] which is also tab-separated. We split source files by tab character "\t" line by line, obtained the tweet id, got the matched id from [21], and merged tweet text with original files to create the common format.

Oxtractor holds every information that our model requires. The source file is stored as JSON. We parse the JSON objects, remove the unnecessary properties, and convert BIO format to the common format.

Manual corpus consumed tweets through Twitter Search API with search key "`q`" and language key "`lang`" parameters. We filter tweets starting with "RT" to prevent retweets and keep only "`status.id`" and "`status.text`" fields from responses. We tokenize the text using "`word_tokenize`" and "`CoreNLPParser`" combined from "`nltk`" library [27] and add to the response. Each tweet is stored as a single lined separate JSON object in a file for the labeling tool Twitter Tagger.

Twitter Tagger uses the input file as a stack, it reads the first line, parses the JSON object and sends the tweet to the interface for tagging. The user marks the word/words from the interface, selects the NER type then, adds to list, and finally submits the list. After the submission, each entity is written to the output file as a line compatible with the common format. The user can also choose to skip a tweet if it is not qualified for tagging. Submitted or skipped tweets are removed from the input file and return to the beginning until the input file is empty.

Our model also requires POS tagging for tweets, and tagging is a time-consuming process. To reduce evaluation time we tag the tweets at this step and stored them as a JSON file.

The final task of the stage is splitting the corpora to training and testing datasets. NEEL 2016 corpus was presented as pre-divided training and testing datasets. We implement a splitting script for Oxtractor and Manual corpora. By giving a testing NE ratio, the script selects tweets randomly and creates training and testing files. It also divides the POS tagging file accordingly.

4.4.2 Evaluating the model

Prediction and scoring are calculated in this stage. First, we import the "`word2vec`" model using "`gensim`" library [23], then we import datasets and convert to python dictionaries. We calculate the centroid class vectors from NEs for the "cos" feature. After we define the feature set, we calculate "independent variables" (X) from the feature set and define "dependent variable" (y) from NE for training and testing. As the distance-based classifiers need normalized values to define more accurate distances, we apply feature scaling method to X. Next, we run classification algorithms from "`scikit-learn`" library [26] and score the predictions. Finally, all the results are printed to output as "csv" friendly in a detailed format. The flowchart of the implementation is represented in Figure 4.3. During the process, we also use "`NumPy`" [24] and "`pandas`" [25] libraries for handling the array operations.

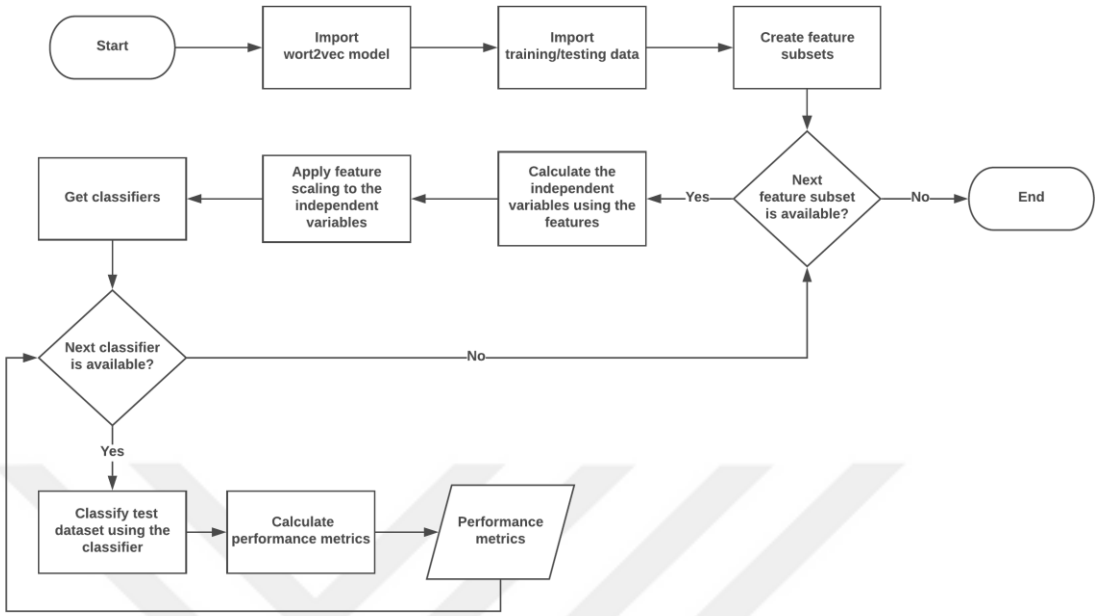

Figure 4.3: the flowchart of the evaluation stage

Due to code optimization made on the previous work [7], calculating all mentioned classifiers for a feature set takes approximately 10 seconds at Intel Core i7 @ 2.30GHz CPU and 8 GB RAM. This performance improvement allows us to run all possible feature set combinations and obtain more detailed outcomes.

5 EVALUATION RESULTS

In this section, we evaluate the results of the combination of 9 different features with 3 different datasets using different split ratios under the baseline classifiers. First, we compare our measurement metrics with previous works [7], [10], [11], and [12]. Secondly, we focus on feature combinations with cosine similarity feature and demonstrate the efficiency of it with different corpora and try to find contributive features of cosine similarity according to average F1 scores. Thirdly, we test and elaborate on the classifier based F1 scores. Finally, we examine the impact of the training/testing spit ratio using manual corpus.

5.1 Related Work Comparison

Evaluations of the combinations in the previous study [7] show that there are more representative feature sets for NEEL 2016 dataset. The SVM classifier with RBF kernel using the feature "**title**", "**position**", "**next_pos**", and "**cos**" achieves the highest statistical metric values; the precision of 0.74, recall of 0.68 and an F1 micro average of 0.67. The classifier predicts 691 true label types correctly from 921 ground truth label type.

Table 5.1: Confusion matrix of SVM classifier with RBF kernel with “cos”, “all capital” and “#” features for 7 label types

		Prediction						
		Person	**Org.**	**Location**	**Thing**	**Product**	**Event**	**Character**
Actual	**Person**	**321**	7	1	4	4	0	0
	Organization	69	**61**	2	18	7	1	0
	Location	7	3	**29**	2	2	0	0
	Thing	5	4	0	**38**	2	0	0
	Product	134	8	3	4	**203**	2	0
	Event	10	1	0	1	0	**12**	0
	Character	19	2	0	0	9	0	**27**

The label “Person” carries the highest portion of the labeled data. According to the confusion matrix of the experiment, given in Table 5.1, the majority of the misclassified NEs are labeled as “Person” even though most of the actual “Person” NEs are labeled correctly. It causes high on recall but low on precision. In contrast, the “Character” label has no false positive value but half of the actual values is falsely labeled. “Location” has the highest F1 score with 0.74 and “Organization” has the lowest F1 score with 0.5. Table 5.2 represents the details of the classification results for each feature.

Table 5.2: The classification report of the features

	precision	**recall**	**F1 score**
Person	0.57	0.95	0.71
Organization	0.71	0.39	0.5
Location	0.83	0.67	0.74
Thing	0.57	0.78	0.66
Product	0.89	0.57	0.7
Event	0.8	0.5	0.62
Character	1	0.47	0.64

In Table 5.3 the precision, recall, and F1 statistical metric values are benchmarked by using the dataset provided by NEEL 2016 Challenge [9]. The methods in [10] [11] and [12] presented during the NEEL 2016 workshop and the method presented by [7] evaluates these three methods while testing the same dataset.

Table 5.3: The performance of our approach is compared with other studies

Study	Precision	Recall	F1 micro avg.
SVM with RBF, 3 features + Cosine Similarity	**0.74**	**0.68**	**0.676**
A feature-based approach performing Stanford NER, [10]	0.729	0.626	0.674
Logistic Regression, 5 features + Cosine Similarity, [7]	0.71	0.56	0.58
TwitIE (CRF Model), [12]	0.435	0.459	0.447
Stanford NER, MITIE, twitter_nlp and TwitIE, [11]	0.587	0.287	0.386

5.2 Performance Metrics Based On Features

The average scores of all experiments in terms of datasets are represented in Figure 5.1. All datasets draw the same pattern wherein the absence of cosine similarity a dramatic drop is observed. There are 3 parameters that we may examine: the count of label types, the volumes and the label distribution of training/testing datasets.

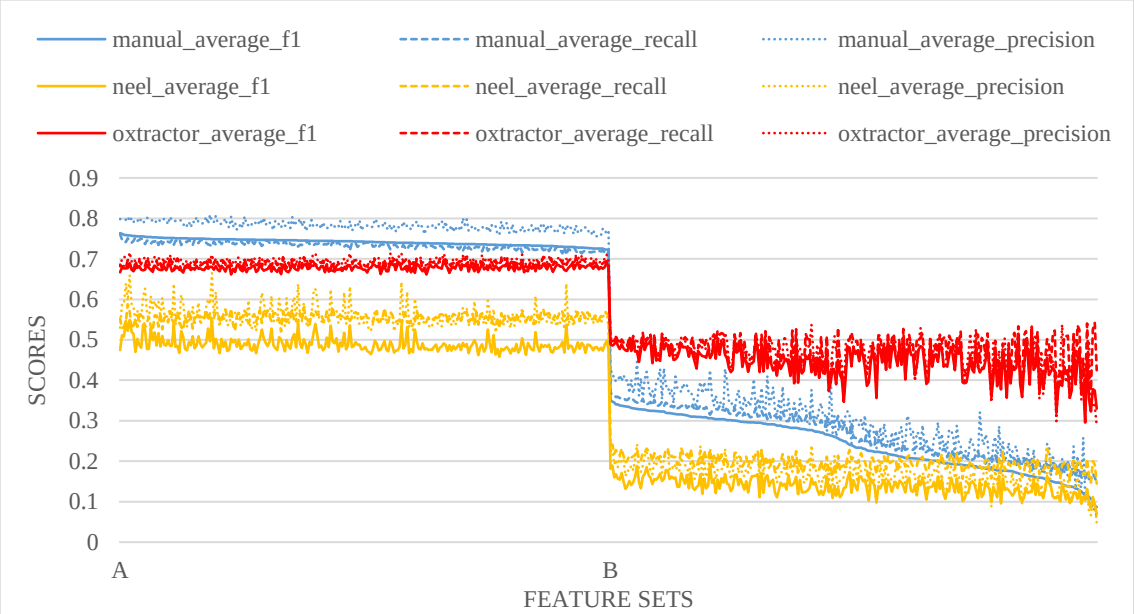

Figure 5.1: the average F1, recall and precision scores of each dataset. “A” represents the features with cosine similarity “B” represents the features without cosine similarity. Feature sets are ordered by manual dataset F1 scores.

When we concentrate on comparing dataset results, the most successful results are achieved by Manual dataset even though higher prediction scores are expected from Oxtractor as it has 3 label types. However, without cosine similarity, although the F1 scores decrease, Oxtractor results give the highest scores as expected. Cosine similarity

feature averages the named entity vectors by their label types. When the diversity of NE reduces, average vectors become more accurate. As the manual dataset focuses on specific topics, clusters of NEs are smaller than the other datasets using this study.

In terms of the volume of the datasets, we have noted that there is not a notable difference between the datasets.

The other distinguishing parameter is the label distribution of training/testing datasets. It is seen in Table 4.2 that Manual dataset has the most similar distribution of training/testing datasets while the most variable distribution of training/testing datasets of NEEL 2016. We have come to the conclusion that cosine similarity might be applied more efficiently to evenly distribute training/testing datasets.

Our final task is finding supportive features with cosine similarity. We analyze the average F1 scores in the absence and presence of the features along with cosine similarity as seen in Table 5.4. “next_pos” and “position” features decrease scores for all datasets. The other features have positive and negative effects according to the dataset. However, the impacts are negligible compared to the cosine similarity itself.

Table 5.4: the F1 scores of the presence and the absence of each feature with “cos” feature.

	NEEL 2016			**Oxtractor**			**Manual**		
	Presence	**Absence**	**Ratio**	**Presence**	**Absence**	**Ratio**	**Presence**	**Absence**	**Ratio**
#	0.523	0.546	-4.21%	0.755	0.748	0.94%	0.737	0.747	-1.34%
@	0.532	0.537	-0.93%	0.750	0.753	-0.40%	0.742	0.742	0.00%
title	0.535	0.533	0.38%	0.750	0.753	-0.40%	0.740	0.743	-0.40%
all_capital	0.537	0.532	0.94%	0.753	0.751	0.27%	0.742	0.741	0.13%
pos	0.531	0.538	-1.30%	0.750	0.753	-0.40%	0.744	0.739	0.68%
next_pos	0.533	0.536	-0.56%	0.750	0.753	-0.40%	0.739	0.744	-0.67%
prev_pos	0.537	0.532	0.94%	0.750	0.753	-0.40%	0.741	0.742	-0.13%
position	0.531	0.538	-1.30%	0.749	0.754	-0.66%	0.741	0.742	-0.13%

5.3 Performance Metrics of the Classifiers

Since the average scores are an acceptable level, to further elaborate the performance of multi-class NER prediction, we analyze F1 score patterns of each classifier for each

corpus. In Figure 5.2, Figure 5.3, and Figure 5.4 for each classifier using the datasets, the responses of the F1 metric value with respect to the feature suite constituted by the combination of 8 features and the cosine similarity feature are compared. Each stand-alone feature is combined with the cosine similarity and then, the suite is varied by combining these features and augmenting with the cosine similarity. Overall, there are 8 features and the cosine similarity, which gives 511 combinations. The F1 metric values for the presence of the cosine similarity are plotted under the area denoted by A. Then, the F1 metric values for the absence of the cosine similarity are plotted under the area B. The results show that the behaviors of the classifiers do not change dramatically when the corpus is changed. All classifiers produce higher scores when the feature "cos" is included in the feature set and there is a significant drop at scores when "cos" is excluded. The only exception of the pattern is Gaussian NB at Oxtractor dataset. It evaluates unpredictable scores within the range of 0.563 and 0.252.

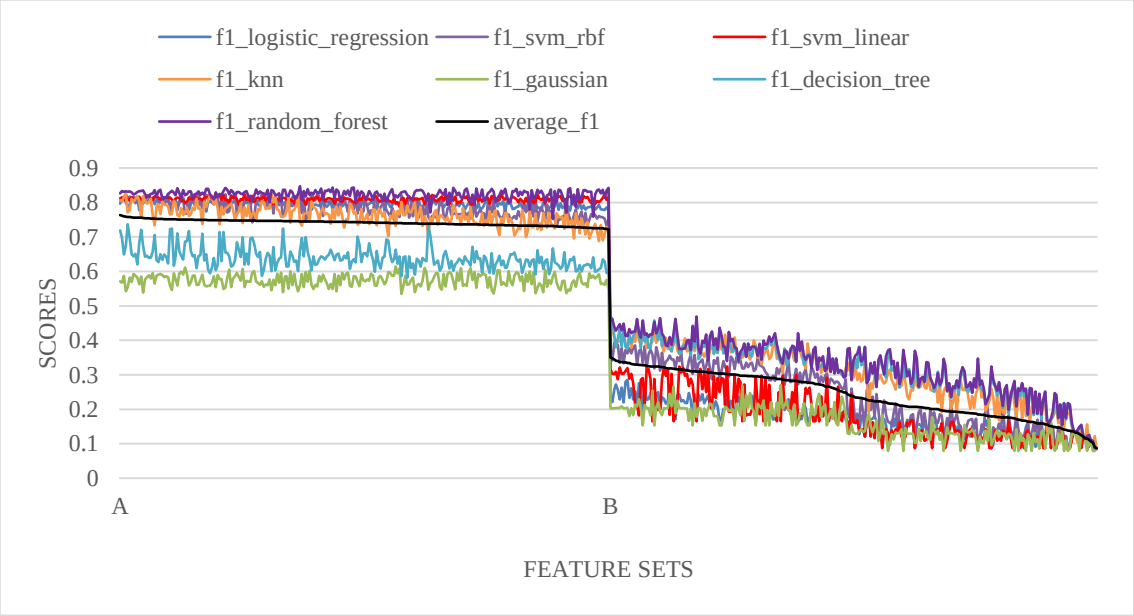

Figure 5.2: the manual dataset F1 scores of each classifier. "A" represents the features with cosine similarity "B" represents the features without cosine similarity. Classifiers are ordered by average F1 scores.

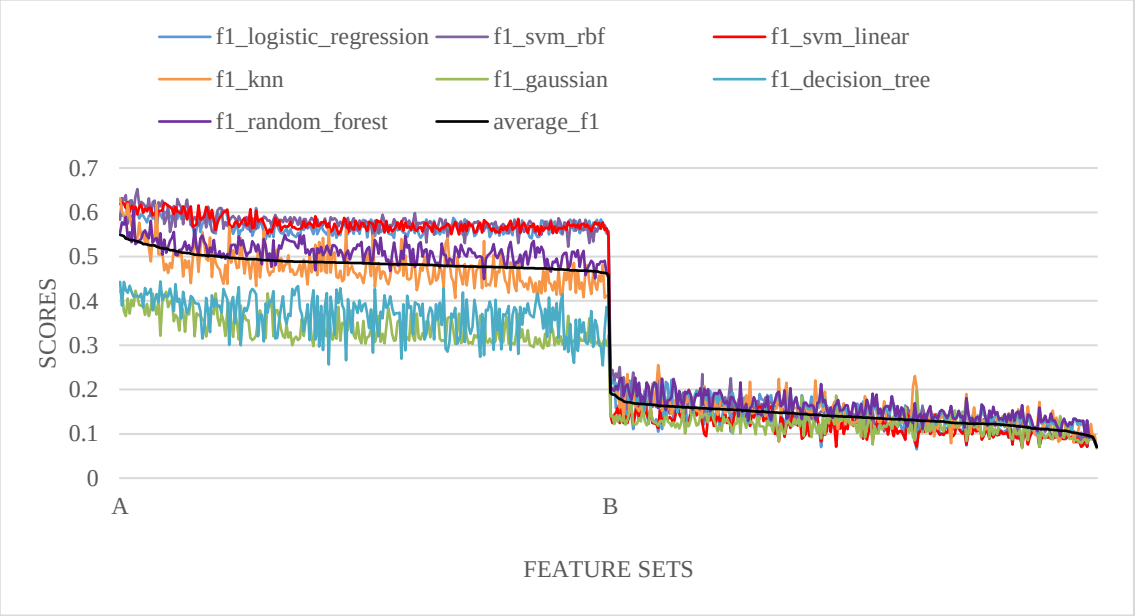

Figure 5.3: NEEL 2016 dataset F1 scores of each classifier. “A” represents the features with cosine similarity “B” represents the features without cosine similarity. Classifiers are ordered by average F1 scores.

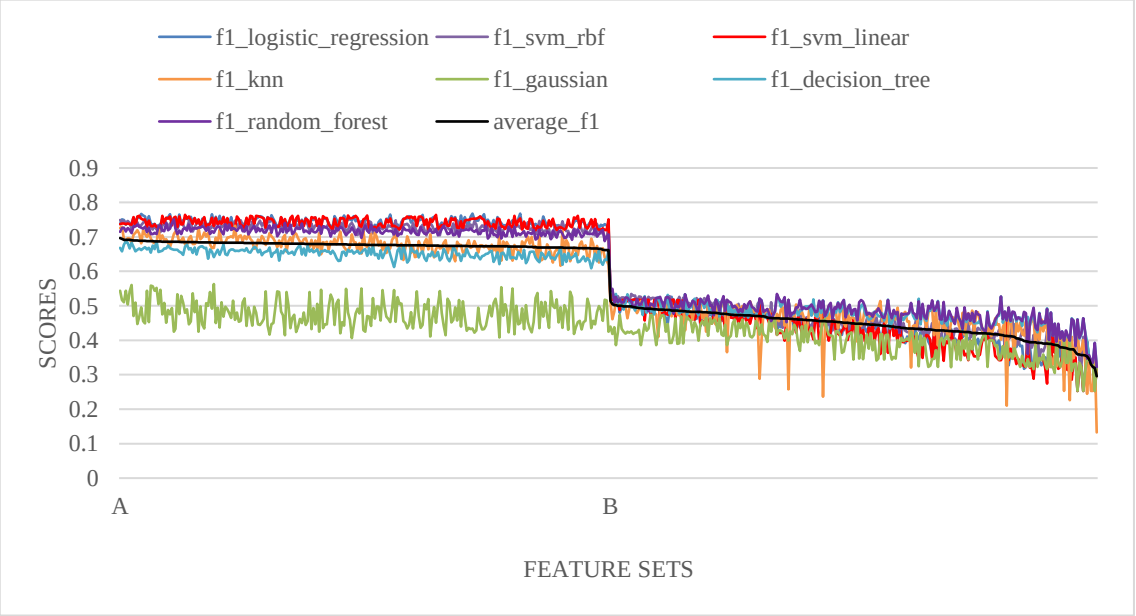

Figure 5.4: Oxtractor dataset F1 scores of each classifier. “A” represents the features with cosine similarity “B” represents the features without cosine similarity. Classifiers are ordered by average F1 scores.

The “cos” feature has a significant impact on the classifier performance in comparison with respect to the suite of other 8 features. In Table 5.5, the F1 statistical metric value

achieved by classifiers with and without (denoted by ‘w/o’) the presence of the “cos” feature in the feature suite are compared. The highest F1 metric values assured by 3 classifiers using the feature suite including the “cos” feature are logistic regression, SVM and random forest. The ranking varies among 3 classifiers depending on the particular dataset. The performance of logistic regression and SVM decreases dramatically when the “cos” feature is excluded. The most successful 3 classifiers using the combination of features excluding the “cos” feature are random forest, decision tree, and k-NN.

Table 5.5: with cos, without cos and average F1 scores for each dataset.

	Manual			**NEEL 2016**			**Oxtractor**		
	w/ cos avg.	**w/o cos avg.**	**avg.**	**w/ cos avg.**	**w/o cos avg.**	**avg.**	**w/ cos avg.**	**w/o cos avg.**	**avg.**
Logistic Regression	0.795	0.172	0.484	0.569	0.129	0.349	0.744	0.431	0.588
SVM with RBF	0.785	0.238	0.512	0.578	0.143	0.361	0.729	0.451	0.590
SVM with Linear	0.810	0.181	0.496	0.576	0.117	0.347	0.742	0.421	0.582
k-NN	0.764	0.297	0.531	0.477	0.146	0.312	0.680	0.450	0.565
Gaussian NB	0.574	0.158	0.367	0.341	0.120	0.231	0.478	0.396	0.437
Decision Tree	0.640	0.314	0.477	0.370	0.153	0.262	0.653	0.475	0.564
Random Forest	0.824	0.323	0.574	0.512	0.157	0.335	0.717	0.481	0.599

Additionally, we note the highest F1 scores during the experiments shown as in Table 5.6. As the average scores in Figure 5.1, the highest result belongs to the Manual dataset followed by Oxtractor and NEEL 2016. The scores are produced by different classifiers. All three feature sets contain the features “**title**”, and “**cos**” however as we already mention in section 5.2 the features except “cos” have not a significant impact on evaluation results individually.

Table 5.6: the highest F1 scores for each dataset.

	Score	Classifier	Feature set
Manual	0.847	Random Forest	@, title, all_capital, next_pos, prev_pos, cos
NEEL 2016	0.652	SVM with RBF	title, position, next_pos, and cos
Oxtractor	0.767	Logistic Regression	@, title, pos, position, prev_pos, and cos

5.4 Performance Metrics Based On The Training and Testing Splitting Ratio

Our final task is to evaluate the scores when training versus testing split ratio is changed. We choose manual corpus for the experiment to investigate its behavior under different samplings since it is generated in the wild during this study. Whereas samplings are random as mentioned in section 4.1, NER type occurrence rates are similar, see for instance Table 4.2.

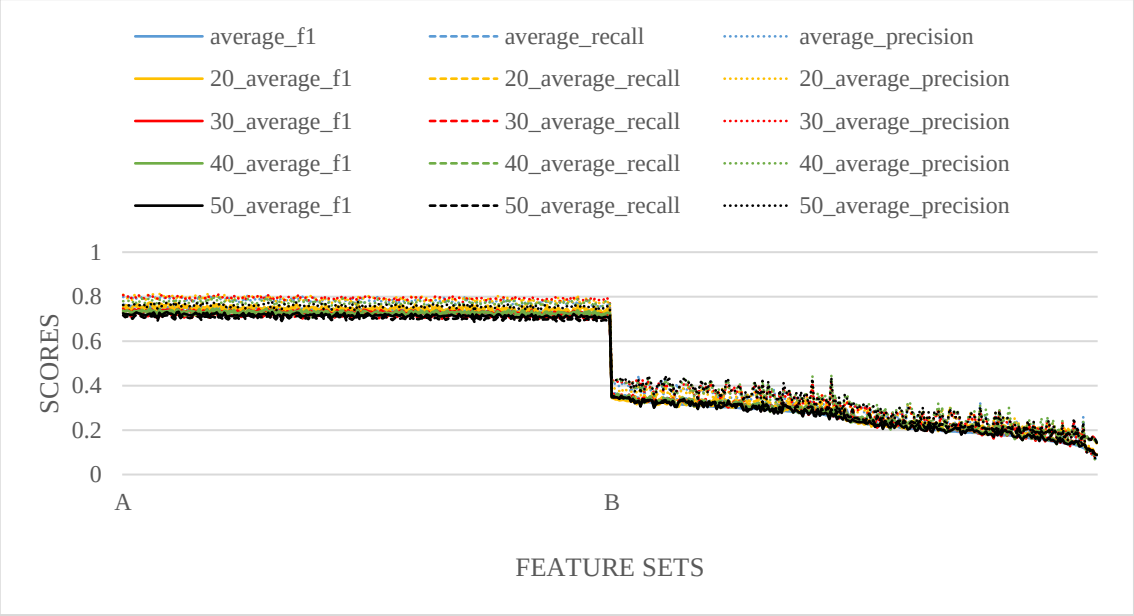

Figure 5.5: the average F1, recall and precision scores of each split ratio of the manual dataset. “A” represents the features with cosine similarity “B” represents the features without cosine similarity.

The average F1 results with “cos” vary between 0.697 and 0.774. Though the highest and the lowest points of the results are similar in Figure 5.5. However when we average F1 scores with “cos”, 80/20 ratio is the optimum sampling for the corpus as seen in Figure 5.6.

Figure 5.6: the average F1 scores with "cos" for each sampling

Random Forest classifier reaches the highest F1 scores independent of the sampling sizes. The highest F1 score is achieved with 0.859 by using 80/20 splitting ratio. We also inspect that using smaller splitting ratio causes relatively low scores as seen in Table 5.7.

Table 5.7: the best scores of the manual corpus according to the split ratio

splitting ratio	classifier	precision	recall	f1-score
90/10	Random Forest	0.882	0.819	0.847
80/20	Random Forest	0.903	0.824	0.859
70/30	Random Forest	0.862	0.784	0.816
60/40	Random Forest	0.860	0.799	0.826
50/50	Random Forest	0.848	0.767	0.802

For better understanding, the results given in Table 5.7 is visualized in Figure 5.7. Each splitting ratio is represented by 3 metrics; precision, recall, and F1 score respectively.

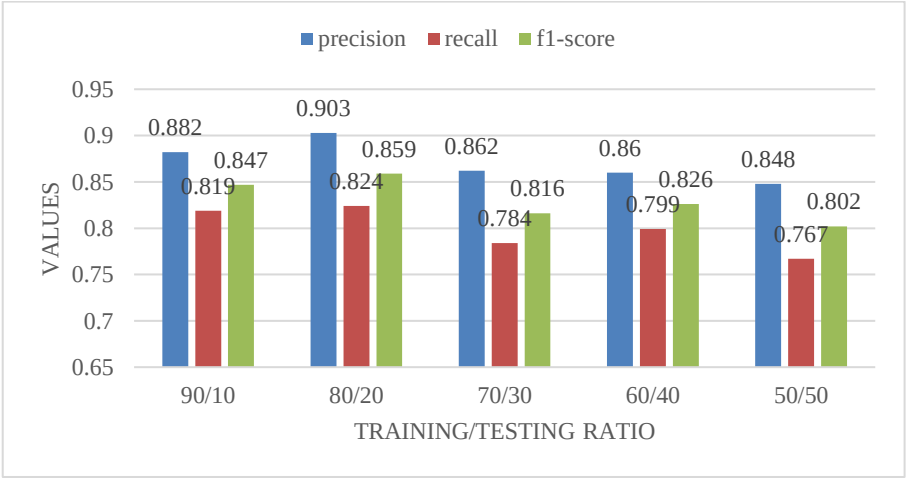

Figure 5.7: the highest precision, recall and F1 metric values provided by Random Forest for each training/testing splitting ratio

6 CONCLUSION

6.1 Objective Completion

Because of the unstructured nature of the tweets, supervised classification features, which are effective on more structured texts like newspapers or articles, do not have an impact on the classifiers. To discover the more representative features, we investigate and analyze syntactic, semantic and domain-specific features on different corpora. First, we gathered all tweets of NEEL 2016 Challenge used in [7] to work on. Second, we reach to another tweet corpus from a start-up focussing on NLP on social media called Oxtractor. Finally, we create our own corpus manually from Twitter in the wild.

The study [7] already states that “cos” is an effective feature for predicting NEs from tweet data. To examine the performance of the feature we add three new supportive features to the study and execute all combinations of features overall selected classifiers for each dataset. We use the confusion matrix, precision, recall, and F1 score to evaluate results.

Through applying all possible feature sets to each classifier, we may find the best fitting combination for NEEL 2016 Challenge dataset. SVM with RBF kernel achieved 0.67 F1 scores with “**title**”, “**position**”, “**next_pos**”, and “**cos**” features.

Focusing on “cos” shows that, neither corpus volume nor NER type count is critical for the performance of the feature. We expect that Oxtractor corpus should achieve the best scores since it contains 3 NER types. However, manual corpus with 7 NER types has the best scores with “cos”. It leads us to the point that results of working on datasets which are focusing on narrowed topics are more successful with “cos”. Despite the fact that

“cos” is the most significant feature for all experiments, we may not find any specific supportive feature for “cos” since there is not any common feature among the feature sets which carries higher FI scores.

We should also analyze the behaviors of classifiers when “cos” is included in the feature sets. Besides the fact that the positive impact of “cos” is observable over all classifiers, the outcomes of Logistic regression, SVM (with 2 different kernels) and random forest are dramatically increased. As random forest, decision tree, and k-NN reach higher scores with feature sets non-including “cos”, the overall results highlight that random forest is significantly adaptive for both cases.

In addition, the split ratio of training/testing datasets does not affect outcomes significantly on manual corpus nevertheless, dividing corpus by 80/20 gives slightly better results among the other experiments.

6.2 Future Work

Looking forward, further attempts could be built by feature-inferring neural network models approach with using the “cos” feature in order to challenge the state-of-the-art “cos” implementation performance on our new presented corpus. We believe that deep learning application with “cos” feature can improve our results.

REFERENCES

[1] V. Yadav and S. Bethard, "A Survey on Recent Advances in Named Entity Recognition from Deep Learning models," in *COLING*, 2018.

[2] R. Grishman and B. Sundheim, "Message Understanding Conference-6: A Brief History," in *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, Stroudsburg, Association for Computational Linguistics, 1996, pp. 466-471.

[3] E. Gabrilovich and S. Markovitch, "Feature generation for text categorization using world knowledge.," in *IJCAI*, Edinburgh, 2005.

[4] B. W. Locke, "Named entity recognition: adapting to microblogging," in *Computer Science Undergraduate Contributions*, 2009.

[5] M. Pennacchiotti and A. M. Popescu, "A machine learning approach to twitter user classification," in *Fifth International AAAI Conference on Weblogs and Social Media*, Barcelona, 2011.

[6] M. Tomas, C. Kai, C. Greg and D. Jeffrey, "Efficient estimation of word representations in vector space," in *ICLR Workshop*, Scottsdale, 2013.

[7] M. Taşpınar, M. C. Ganiz and T. Acarman, "A Feature Based Simple Machine Learning Approach with Word Embeddings to Named Entity Recognition on Tweets," in *Natural Language Processing and Information Systems*, Liège, 2017.

[8] G. Fréderic, V. Baptist, D. N. Wesley and V. d. W. Rik, "Named Entity Recognition for Twitter Microposts using Distributed Word Representations," in *Proceedings of the Workshop on Noisy User-generated Text*, Beijing, 2015.

[9] G. Rizzo, M. v. Erp, J. Plu and R. Troncy, "Making Sense of Microposts (#Microposts2016) Named Entity rEcognition and Linking (NEEL) Challenge," in

6th Workshop on 'Making Sense of Microposts' co-located with the 25th International World Wide Web Conference (WWW 2016), Montréal, 2016.

[10] S. Ghosh, P. Maitra and D. Das, "Feature Based Approach to Named Entity Recognition and Linking for Tweets," in *#Microposts*, 2016.

[11] K. Greenfield, R. S. Caceres, M. Coury, K. Geyer, Y. Gwon, J. Matterer, A. Mensch, C. S. Sahin and O. Simek, "A Reverse Approach to Named Entity Extraction and Linking in Microposts," in *#Microposts*, 2016.

[12] P. Torres-Tramón, H. Hromic, B. Walsh, B. R. Heravi and C. Hayes, "Kanopy4Tweets: Entity Extraction and Linking for Twitter," in *#Microposts*, 2016.

[13] W. J. Black, F. Rinaldi and D. Mowatt, "FACILE: Description of the NE System Used for MUC-7," in *Seventh Message Understanding Conference (MUC-7): Proceedings of a Conference Held in Fairfax, Virginia, April 29 - May 1, 1998*, Virginia, 1998.

[14] K. Toutanova, D. Klein, C. D. Manning and Y. Singer, "Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network," in *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, Edmonton, 2003.

[15] J. A. Nelder and R. W. M. Wedderburn, "Generalized Linear Models," *Journal of the Royal Statistical Society: Series A (General)*, vol. 135, no. 3, pp. 370-384, 1972.

[16] V. N. Vapnik, The Nature of Statistical Learning Theory, Berlin, Heidelberg: Springer-Verlag, 1995.

[17] M.-L. Zhang and Z.-H. Zhou, "ML-KNN: A lazy learning approach to multi-label learning," in *Pattern Recognition, Volume 40, Issue 7*, Nanjing, Elsevier, 2007, pp. 2038-2048.

[18] J. R. Quinlan, "Induction of Decision Trees," *Mach. Learn.*, vol. 1, no. 1, pp. 81-106, 1986.

[19] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[20] D. K. Lewis, "Naive (Bayes) at forty: The independence assumption in information retrieval," in *Machine Learning: ECML-98*, Berlin, Heidelberg, Springer Berlin Heidelberg, 1998, pp. 4-15.

[21] Y. Grunin and V. Laakso, "Github," 2016. [Online]. Available: https://github.com/webpack-contrib/webpack-bundle-analyzer.

[22] N. Chinchor, "MUC-4 Evaluation Metrics," in *Proceedings of the 4th Conference on Message Understanding*, Stroudsburg, Association for Computational Linguistics, 1992, pp. 22-29.

[23] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," in *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks*, Valletta, University of Malta, 2010, pp. 46-50.

[24] T. E. Oliphant, Guide to NumPy, CreateSpace Independent Publishing Platform, 2015.

[25] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot and É. Duchesnay, "Scikit-learn: Machine Learning in Python," *J. Mach. Learn. Res.*, Vols. 12, 2/1/2011, pp. 2825-2830, 2011.

[27] S. Bird, E. Klein and E. Loper, Natural Language Processing with Python, O'Reilly Media, 2009.

APPENDICES

Software Source Code.

```
# main.py

from Session import Session
from ner.constants import Constants
from ner.ner_experiment import NerExperiment
import sys
import warnings

if not sys.warnoptions:
  warnings.simplefilter("ignore")

features = [
  Constants.FEATURE_HASHTAG,
  Constants.FEATURE_AT,
  Constants.FEATURE_TITLE,
  Constants.FEATURE_ALL_CAPITAL,
  Constants.FEATURE_POS,
  Constants.FEATURE_POSITION,
  Constants.FEATURE_NEXT_POS,
  Constants.FEATURE_PREV_POS,
  Constants.FEATURE_COSINE
]

train_path = 'source/ner/manual_neel_train.txt'
test_path = 'source/ner/manual_neel_test.txt'

data_train = []
data_test = []

with open(train_path, encoding="utf8") as file:
  for row in file:
    data_train.append(row)

with open(test_path, encoding="utf8") as file:
  for row in file:
    data_test.append(row)

experiment = NerExperiment([])
session = Session(features, data_train, data_test, type(experiment))
session.run(Session.OutputType.csv_friendly, True)
```

```
# session.py

import itertools
from enum import Enum

from sklearn import metrics, svm
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
```

```
import Experiment


class Session:
  class OutputType(Enum):
    detailed = 0
    csv_friendly = 1

  LOGISTIC_REGRESSION = 'logistic_regression'
  SVM_RBF = 'svm_rbf'
  SVM_LINEAR = 'svm_linear'
  KNN = 'knn'
  GAUSSIAN = 'gaussian'
  DECISION_TREE = 'decision_tree'
  RANDOM_FOREST = 'random_forest'

  def __init__(self, feature_set: list, train_data: list, test_data: list,
               experiment: Experiment):
    self.experiment = experiment
    self.train_data = train_data
    self.test_data = test_data
    self.feature_set = feature_set

    self.output = []
    self.classifier_set = [self.LOGISTIC_REGRESSION,
                           self.SVM_RBF,
                           self.SVM_LINEAR,
                           self.KNN,
                           self.GAUSSIAN,
                           self.DECISION_TREE,
                           self.RANDOM_FOREST]

  def run(self, output_type: OutputType, create_all_combinations=False):
    if create_all_combinations:
      all_combinations = itertools.chain(
        *[itertools.combinations(self.feature_set, i + 1)
          for i, _ in enumerate(self.feature_set)])
      feature_sets = list(all_combinations)
    else:
      feature_sets = [self.feature_set]

    complete_percentage = -1
    for index, feature_set in enumerate(feature_sets):
      if complete_percentage < int(index / len(feature_sets) * 100):
        complete_percentage = int(index / len(feature_sets) * 100)
        print('0% [{}{}] 100% - {}%'.format('=' * complete_percentage,
                                            '.' * (100 -
complete_percentage),
                                            complete_percentage))

      self.__add_to_output('feature_set', feature_set, True)

      experiment = self.experiment(feature_set)

      scalar = StandardScaler()

      X_train, y_train = experiment.run(self.train_data, True)
      X_train = scalar.fit_transform(X_train)

      X_test, y_test = experiment.run(self.test_data, False)
      X_test = scalar.transform(X_test)

      for classifier_name in self.classifier_set:
        y_prediction = self.__predict_by_classifier(classifier_name,
X_train,
                                                    y_train, X_test)
        self.__calculate_score_and_to_output(y_test, y_prediction,
classifier_name)

    if output_type == Session.OutputType.detailed:
      self.__print_detailed_output()
    elif output_type == Session.OutputType.csv_friendly:
      self.__print_csv_friendly_output()
    self.output = []
```

```
  def __predict_by_classifier(self, classifier_name, X_train, y_train,
X_test):
    if classifier_name == self.LOGISTIC_REGRESSION:
      return self.__run_logistic_regression(X_train, y_train, X_test)

    if classifier_name == self.SVM_RBF:
      return self.__run_svc_rbf(X_train, y_train, X_test)

    if classifier_name == self.SVM_LINEAR:
      return self.__run_svc_linear(X_train, y_train, X_test)

    if classifier_name == self.KNN:
      return self.__run_k_neighbors(X_train, y_train, X_test)

    if classifier_name == self.GAUSSIAN:
      return self.__run_gaussian(X_train, y_train, X_test)

    if classifier_name == self.DECISION_TREE:
      return self.__run_decision_tree(X_train, y_train, X_test)

    if classifier_name == self.RANDOM_FOREST:
      return self.__run_random_forest(X_train, y_train, X_test)

    return None

  def __run_logistic_regression(self, X_train, y_train, X_test):
    classifier = LogisticRegression(random_state=0)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_svc_rbf(self, X_train, y_train, X_test):
    classifier = svm.SVC(kernel='rbf', random_state=0)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_svc_linear(self, X_train, y_train, X_test):
    classifier = svm.SVC(kernel='linear', random_state=0)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_k_neighbors(self, X_train, y_train, X_test):
    classifier = KNeighborsClassifier(p=2)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_gaussian(self, X_train, y_train, X_test):
    classifier = GaussianNB()
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_decision_tree(self, X_train, y_train, X_test):
    classifier = DecisionTreeClassifier(random_state=0)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __run_random_forest(self, X_train, y_train, X_test):
    classifier = RandomForestClassifier(n_estimators=100, random_state=0)
    classifier.fit(X_train, y_train)
    return classifier.predict(X_test)

  def __calculate_score_and_to_output(self, y_test, y_prediction,
classifier):
    self.__add_to_output('never_predicted_' + classifier,
                         set(y_test) - set(y_prediction))
    self.__add_to_output('confusion_matrix_' + classifier,
                         metrics.confusion_matrix(y_test, y_prediction))
    self.__add_to_output('classification_report_' + classifier,
                         metrics.classification_report(y_test,
y_prediction))
    self.__add_to_output('f1_macro_' + classifier,
                         metrics.f1_score(y_test, y_prediction,
average='macro'))
    self.__add_to_output('f1_micro_' + classifier,
```

```
                        metrics.f1_score(y_test, y_prediction,
average='micro'))
    self.__add_to_output('recall_' + classifier,
                        metrics.recall_score(y_test, y_prediction,
average='macro'))
    self.__add_to_output('precision_' + classifier,
                        metrics.precision_score(y_test, y_prediction,
                                                average='macro'))

  def __add_to_output(self, key, value, is_new_element=False):
    if is_new_element or len(self.output) == 0:
      self.output.append({})

    self.output[-1][key] = value

  def __print_detailed_output(self):
    last_feature_set = []

    for element in self.output:
      if last_feature_set != element['feature_set']:
        print('+++ Features: {} +++'.format(element['feature_set']))
        last_feature_set = element['feature_set']

      for classifier in self.classifier_set:
        print('++ {} ++'.format(classifier))
        print('+ Confusion Matrix +')
        print(element['confusion_matrix_' + classifier])
        print('+ Classification Report +')
        print(element['classification_report_' + classifier])
        print('+ F1 Micro Average +')
        print(element['f1_micro_' + classifier])
        print('+ F1 Macro Average +')
        print(element['f1_macro_' + classifier])
        print('+ No Predicted Labels +')
        print(element['never_predicted_' + classifier])

  def __print_csv_friendly_output(self):
    line = ''
    for classifier in self.classifier_set:
      line += '\tf1_{}'.format(classifier)
    for classifier in self.classifier_set:
      line += '\trecall_{}'.format(classifier)
    for classifier in self.classifier_set:
      line += '\tprecision_{}'.format(classifier)

    print(line)

    line = ''
    for element in self.output:
      line += '{}'.format(element['feature_set'])
      for classifier in self.classifier_set:
        line += '\t{}'.format(element['f1_macro_' + classifier])
      for classifier in self.classifier_set:
        line += '\t{}'.format(element['recall_' + classifier])
      for classifier in self.classifier_set:
        line += '\t{}'.format(element['precision_' + classifier])

    print(line)
```

```
# experiment.py

class Experiment:

  def __init__(self, features: list):
    self.features = features

  def run(self, data: list, is_train: bool):
    pass
```

```
# ner_experiment.py
```

```
import json
import gensim
import pandas as pd
from numpy.core.multiarray import array

from Experiment import Experiment
from ner.constants import Constants
from ner.features import Features
from ner.calculations import ner_to_numeric
from ner.utils import get_micropost_data_with_model


class NerExperiment(Experiment):
  word2vec_path = "ner/source/word2vec_twitter_model.bin"
  pos_test_path = "ner/source/POS_manual_neel_test.json"
  pos_train_path = "ner/source/POS_manual_neel_train.json"

  word2vec = None
  pos_train = None
  pos_test = None
  data_train = None
  data_test = None
  average_train = None
  average_test = None

  def __init__(self, features: list):
    if NerExperiment.word2vec is None:
      NerExperiment.word2vec =
gensim.models.KeyedVectors.load_word2vec_format(
        NerExperiment.word2vec_path,
        binary=True,
        unicode_errors='ignore')

    if NerExperiment.pos_train is None:
      NerExperiment.pos_train = json.loads(
        open(NerExperiment.pos_train_path, encoding='utf8').read())

    if NerExperiment.pos_test is None:
      NerExperiment.pos_test = json.loads(
        open(NerExperiment.pos_test_path, encoding='utf8').read())

    super(NerExperiment, self).__init__(features)

  def run(self, data: list, is_train: bool):
    if is_train and NerExperiment.data_train is None:
      NerExperiment.data_train, NerExperiment.average_train = \
          get_micropost_data_with_model(
        data,
        NerExperiment.word2vec)

    if not is_train and NerExperiment.data_test is None:
      NerExperiment.data_test, NerExperiment.average_test = \
          get_micropost_data_with_model(
        data,
        NerExperiment.word2vec)

    if is_train:
      words = NerExperiment.data_train
      pos = NerExperiment.pos_train
      averages = NerExperiment.average_train
    else:
      words = NerExperiment.data_test
      pos = NerExperiment.pos_test
      averages = NerExperiment.average_test

    y = []
    X = {}

    for f in self.features:
      if f == Constants.FEATURE_COSINE:
        X[Constants.AVG_PERSON] = []
        X[Constants.AVG_ORGANIZATION] = []
        X[Constants.AVG_LOCATION] = []
        '''7 NER'''
```

```
        X[Constants.AVG_THING] = []
        X[Constants.AVG_PRODUCT] = []
        X[Constants.AVG_EVENT] = []
        X[Constants.AVG_CHARACTER] = []
      else:
        X[f] = []

    features = Features(words)
    for i in range(len(words)):
      y.append(ner_to_numeric(words[i]['NER']))

      for f in self.features:
        if f == Constants.FEATURE_HASHTAG:
          X = features.get_hashtag_feature(X, i)
        if f == Constants.FEATURE_AT:
          X = features.get_at_feature(X, i)
        if f == Constants.FEATURE_TITLE:
          X = features.get_title_feature(X, i)
        if f == Constants.FEATURE_ALL_CAPITAL:
          X = features.get_all_capital_feature(X, i)
        if f == Constants.FEATURE_POS:
          X = features.get_pos_feature(X, i, pos)
        if f == Constants.FEATURE_POSITION:
          X = features.get_position_feature(X, i)
        if f == Constants.FEATURE_COSINE:
          X = features.get_cosine_similarity_features(X, i, averages,

NerExperiment.word2vec)
        if f == Constants.FEATURE_NEXT_POS:
          X = features.get_next_pos_feature(X, i, pos)
        if f == Constants.FEATURE_PREV_POS:
          X = features.get_prev_pos_feature(X, i, pos)

    X = pd.DataFrame(X)
    y = array(y)

    return X, y
```

```
# calculations.py

def score_to_numeric(score):
  """For Stanford POS tagger String to Numeric value"""

  score_dic = {
    'CC': 1, 'CD': 2, 'DT': 3, 'EX': 4, 'FW': 5, 'IN': 6, 'JJ': 7, 'JJR': 8,
    'JJS': 9, 'LS': 10, 'MD': 11, 'NN': 12, 'NNS': 13, 'NNP': 14, 'NNPS':
15,
    'PDT': 16, 'POS': 17, 'PRP': 18, 'PRP$': 19, 'RB': 20, 'RBR': 21, 'RBS':
22,
    'RP': 23, 'SYM': 24, 'TO': 25, 'UH': 26, 'VB': 27, 'VBD': 28, 'VBG': 29,
    'VBN': 30, 'VBP': 31, 'VBZ': 32, 'WDT': 33, 'WP': 34, 'WP$': 35, 'WRB':
36,
    '.': 37, ':': 38, ',': 39, '``': 40, '#': 41, '$': 42, '"': 43, '(': 44,
')': 45
  }

  return score_dic.get(score, 46)


def ner_to_numeric(ner):
  """For NER Types String to Numeric value"""

  ner_dic = {
    'Person': 1,
    'Organization': 2,
    'Location': 3,
    'Thing': 4,
    'Product': 5,
    'Event': 6,
    'Character': 7
  }
```

```
  result = ner_dic.get(ner, 0)

  if result == 0:
    print(ner)

  return result
```

```
# constants.py

class Constants:
  FEATURE_HASHTAG = 'Hashtag'
  FEATURE_AT = 'At'
  FEATURE_TITLE = 'Title'
  FEATURE_ALL_CAPITAL = 'All Capital'
  FEATURE_POS = 'POS Tagger'
  FEATURE_POSITION = 'Position'
  FEATURE_NEXT_POS = 'Next POS Tagger'
  FEATURE_PREV_POS = 'Prev POS Tagger'
  FEATURE_COSINE = 'Cosine'

  AVG_PERSON = 'Avg Person'
  AVG_ORGANIZATION = 'Avg Organization'
  AVG_LOCATION = 'Avg Location'
  AVG_THING = 'Avg Thing'
  AVG_PRODUCT = 'Avg Product'
  AVG_EVENT = 'Avg Event'
  AVG_CHARACTER = 'Cos Character'
```

```
# features.py

import numpy as np
from scipy import spatial

from ner.constants import Constants
from ner.calculations import score_to_numeric


class Features:
  def __init__(self, words):
    self.words = words

  def get_hashtag_feature(self, X, index):
    dic = self.words[index]

    X[Constants.FEATURE_HASHTAG].append(
      '#' in dic['word'] or dic['tweet'][dic['start'] - 1] == '#')

    return X

  def get_at_feature(self, X, index):
    dic = self.words[index]

    X[Constants.FEATURE_AT].append(
      '@' in dic['word'] or dic['tweet'][dic['start'] - 1] == '@')

    return X

  def get_title_feature(self, X, index):
    word = self.words[index]['word']

    X[Constants.FEATURE_TITLE].append(word.istitle())

    return X

  def get_all_capital_feature(self, X, index):
    word = self.words[index]['word']

    X[Constants.FEATURE_ALL_CAPITAL].append(word.isupper())

    return X
```

```
  def get_pos_feature(self, X, index, pos):
    dic = self.words[index]

    tags = pos.get('{}_{}_{}'.format(dic['id'], dic['start'], dic['word']),
[])

    if not tags:
      X[Constants.FEATURE_POS].append(0)
    else:
      # 47 for mixed POS tags
      tag = tags[0][1]
      is_same = True

      for item in tags:
        is_same = tag == item[1]
        if not is_same:
          break

      if is_same:
        X[Constants.FEATURE_POS].append(score_to_numeric(tag))
      else:
        X[Constants.FEATURE_POS].append(47)

    return X

  def get_prev_pos_feature(self, X, index, pos):
    dic = self.words[index]
    tags = pos.get('{}_{}_{}'.format(dic['id'], dic['start'] - 1,
dic['word']), [])

    if not tags:
      X[Constants.FEATURE_PREV_POS].append(0)
    else:
      X[Constants.FEATURE_PREV_POS].append(score_to_numeric(tags[1]))

    return X

  def get_next_pos_feature(self, X, index, pos):
    dic = self.words[index]
    tags = pos.get('{}_{}_{}'.format(dic['id'], dic['start'] + 1,
dic['word']), [])

    if not tags:
      X[Constants.FEATURE_NEXT_POS].append(0)
    else:
      X[Constants.FEATURE_NEXT_POS].append(score_to_numeric(tags[1]))

    return X

  def get_position_feature(self, X, index):
    dic = self.words[index]
    word_before = dic['tweet'][: dic['start']]

    X[Constants.FEATURE_POSITION].append(
      (word_before.count(' ') + 1) / (dic['tweet'].count(' ') + 1))

    return X

  def get_cosine_similarity_features(self, X, index, averages, model):
    dic = self.words[index]
    value_organization = []
    value_person = []
    value_location = []
    value_thing = []
    value_product = []
    value_event = []
    value_character = []

    for ner_word_split in dic['word'].split(' '):
      cleaned_word = ner_word_split.replace('@', '')
      cleaned_word = cleaned_word.replace('#', '')

      if cleaned_word in model.vocab:
        value_organization.append(model[cleaned_word])
```

```
        value_person.append(model[cleaned_word])
        value_location.append(model[cleaned_word])
        value_thing.append(model[cleaned_word])
        value_product.append(model[cleaned_word])
        value_event.append(model[cleaned_word])
        value_character.append(model[cleaned_word])

    value_organization = np.array(value_organization)
    value_person = np.array(value_person)
    value_location = np.array(value_location)
    value_thing = np.array(value_thing)
    value_product = np.array(value_product)
    value_event = np.array(value_event)
    value_character = np.array(value_character)

    if len(value_organization) == 0:
      X[Constants.AVG_PERSON].append(0)
      X[Constants.AVG_ORGANIZATION].append(0)
      X[Constants.AVG_LOCATION].append(0)
      '''7 NER'''
      X[Constants.AVG_THING].append(0)
      X[Constants.AVG_PRODUCT].append(0)
      X[Constants.AVG_EVENT].append(0)
      X[Constants.AVG_CHARACTER].append(0)
    else:
      X[Constants.AVG_PERSON].append(
        1 - spatial.distance.cosine(averages['person'],
                                    np.average(value_person, axis=0)))
      X[Constants.AVG_ORGANIZATION].append(
        1 - spatial.distance.cosine(averages['organization'],
                                    np.average(value_organization, axis=0)))
      X[Constants.AVG_LOCATION].append(
        1 - spatial.distance.cosine(averages['location'],
                                    np.average(value_location, axis=0)))
      '''7 NER'''
      X[Constants.AVG_THING].append(
        1 - spatial.distance.cosine(averages['thing'],
                                    np.average(value_thing, axis=0)))
      X[Constants.AVG_PRODUCT].append(
        1 - spatial.distance.cosine(averages['product'],
                                    np.average(value_product, axis=0)))
      X[Constants.AVG_EVENT].append(
        1 - spatial.distance.cosine(averages['event'],
                                    np.average(value_event, axis=0)))
      X[Constants.AVG_CHARACTER].append(
        1 - spatial.distance.cosine(averages['character'],
                                    np.average(value_character, axis=0)))

    return X
```

```
# utils.py

import numpy as np


def get_micropost_data_with_model(data, word2vec_model):
  words = []

  value_organization = []
  value_person = []
  value_location = []
  value_thing = []
  value_product = []
  value_event = []
  value_character = []

  '''
  line[0] -> tweetId
  line[1] -> tweet
  line[2] -> word start
  line[3] -> word end
  line[4] -> NER Type
```

```
  line[5] -> retweet count
  line[6] -> favorite count
  line[7] -> word
  '''
  index_1 = 0
  index_2 = 0
  for row in data:
    line = row.rstrip().split('\t')
    index_1 += 1
    if len(line) != 8:
      continue

    index_2 += 1
    words.append(
      {'id': line[0], 'tweet': line[1], 'start': int(line[2]), 'end':
int(line[3]),
       'word': line[7],
       'NER': line[4]})

    for ner_word_split in line[7].split(' '):
      word = ner_word_split.replace('@', '')
      word = word.replace('#', '')

      if word in word2vec_model.vocab:
        if line[4] == 'Organization':
          value_organization.append(word2vec_model[word])
        elif line[4] == 'Person':
          value_person.append(word2vec_model[word])
        elif line[4] == 'Location':
          value_location.append(word2vec_model[word])
        elif line[4] == 'Thing':
          value_thing.append(word2vec_model[word])
        elif line[4] == 'Product':
          value_product.append(word2vec_model[word])
        elif line[4] == 'Event':
          value_event.append(word2vec_model[word])
        elif line[4] == 'Character':
          value_character.append(word2vec_model[word])

  value_organization = np.array(value_organization)
  value_person = np.array(value_person)
  value_location = np.array(value_location)
  value_thing = np.array(value_thing)
  value_product = np.array(value_product)
  value_event = np.array(value_event)
  value_character = np.array(value_character)

  averages = {
    'organization': np.average(value_organization, axis=0),
    'person': np.average(value_person, axis=0),
    'location': np.average(value_location, axis=0),
    'thing': np.average(value_thing, axis=0),
    'product': np.average(value_product, axis=0),
    'event': np.average(value_event, axis=0),
    'character': np.average(value_character, axis=0)
  }

  return words, averages
```

BIOGRAPHICAL SKETCH

Onur Büyüktopaç is a software architect at Yemeksepeti in Turkey. He received his BSc degree at Computer Engineering Department of Galatasaray University. In his graduation project, he has worked on strategy building on game theory. He is currently pursuing his MSc degree in Science and Engineering Faculty of the same university and focusing on NLP and machine learning algorithms.