CARLETON UNIVERSITY

SCHOOL OF
MATHEMATICS AND STATISTICS

HONOURS PROJECT

TITLE: Agglomerative Hierarchical Clustering of Queries using Part of Speech Tags

AUTHOR: Fatima Almalki

SUPERVISORS: Dr. Ahmed El – Roby,
Dr. Yiqiang Q. Zhao

DATE: April 7, 2020

Abstract

Natural Language Processing (NLP) is the interdisciplinary field that is concerned with the interface between human language, which can manifest in the form of speech or text, and computers. Being able to classify sentences and queries can prove to be very important for later supervised learning classification tasks. In this paper, we classify 10,000 queries based on their part of speech tags through agglomerative hierarchical clustering.

Contents

Chapter 1

Introduction and background knowledge

Natural Language Processing (NLP) is the interdisciplinary field that is concerned with the interface between human language, which can manifest in the form of speech or text, and computers. Therefore, by logical extension, the fields that constitute NLP are Linguistics and Computer Science. The field of Linguistics contributes the knowledge of how language is structured, including its syntax and meaning, whereas the field of Computer Science applies this knowledge to computer programs so that they can make sense of the given language [9]. This knowledge base is provided for computer programs to use and is constantly updated based on patterns that the program detects and feedback that it receives. Therefore, most NLP programs are built on concepts of artificial intelligence, and many of them use machine learning and deep learning techniques in order to improve their understanding of language [6].

1.1 Applications and Importance

Applications that use NLP are used widely and daily. Many, such as spell checkers, autocomplete, voice text messaging, and spam filters, exist for the purpose of making tasks more convenient and in some cases, even more efficient. Other NLP applications are revolutionizing the way that the disability accommodation gap [4] is being filled. For example, SignAll is an application that uses NLP to convert sign language into text, thus helping deaf individuals and people who do not know sign language communicate. Another example is the Livox App, which has enabled large amounts of people with various types of learning and speech impairments due to diseases such as cerebral palsy to communicate with others [5]. Besides helping people in their everyday lives, NLP is important in the field of big data. With NLP, large quantities of data can be analyzed to draw insights that can be used by businesses to improve their services. Businesses may also use NLP products to provide fast and readily available customer service using bots that can communicate with people. Furthermore, NLP provides a form of numeric structuring to highly unstructured data by resolving ambiguity in language [2].

1.2 Corpora and Tokens

A **corpus** (plural corpora) or text corpus is a large and structured set of texts. For the purpose of research and analysis, these corpora are often annotated with their corresponding part of speech tags (verb, noun, adjective, etc..). Some corpora are also multilingual, offering side by side comparisons of different languages [15]. Popular examples in English include *The Google Books Ngram Corpus*, as well as the *Corpus of Contemporary American English*. There are also more specific examples of corpora, such as *Stanford's IMDB Movie Review Sentiment Classification* or *Flickr 30K* (a collection of 30 thousand described images from flickr) [1].

A **token** is an atomic element in a sentence. This can include words, numbers, acronyms, punctuation, etc.. [15] For example, the sentence:

Carleton University has a population of over 31000 students!

Has tokens the following tokens:

Carleton	**has**	**population**	**over**	**students**
University	**a**	**of**	**31000**	**!**

1.3 Word Embeddings

Word embedding is a set of feature learning techniques where words are mapped to vectors of real numbers. It allows us to capture the context of a word in a document, as well as semantic similarity, and relation to other words. Since machine learning models take vectors as inputs (as opposed to text), word embeddings can also be used to predict Part of Speech (POS) Tags, as discussed in the next section. The word embedding representation is able to reveal many hidden relationships between words, as well as preserves distances between words [10].

For example,

vector("Man") - vector("Woman") is similar to vector("King") - vector("Queen").

This means that the vectors for each word can be added and subtracted, so the difference between the vector for "Man" and "King" added to "Queen" is approximately equal to the vector for "Woman".

$$King - Man + Queen \approx Woman$$

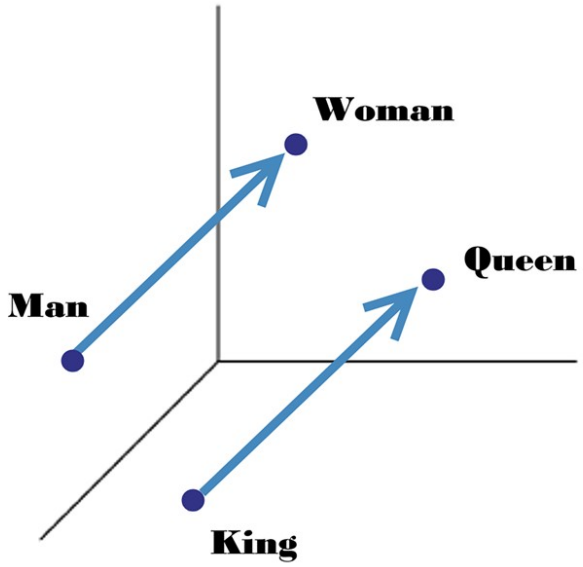

Figure 1.1: Gender vectors. ©Lior Shkiller

1.4 Purpose

Factoid questions are queries that are typically answered with simple facts expressed in short text answers. An example of a factoid question and it's answer is:

Q: What is the smallest planet?

A: Mercury

Categorizing such questions into clusters can prove to be very important for later supervised learning classification tasks. In this paper, we cluster 10, 000 questions using agglomerative hierarchical clustering using the question's part of speech tags and parse tree.

Chapter 2

Part of Speech Tagging

2.1 Introduction to Part of Speech Tagging

A single word may have many different meanings depending on its position in a sentence, as well as its relation to other words. For example,

1. "Don't forget to ***lock*** the door."
2. "I bought a new ***lock*** for the fence."

In this case, the word *lock* is used as a verb in the first sentence and as a noun in the second. A part of speech tagger must therefore be able to identify the part of speech (proper noun, verb, preposition, adjective, etc..) for each token in a sentence, based on its definition and the context in which it was used.

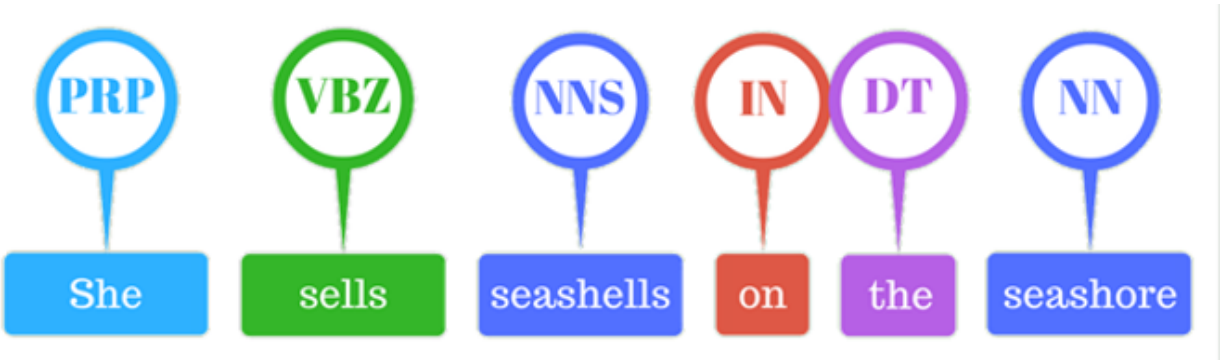

Figure 2.1: Example of the corresponding part of speech tags for each word in the sentence "She sells seashells on the seashore"

For this project, the Python NLP library called **Flair** is used to predict the POS tags for each query [3]. **Flair** is accompanied by a pre-trained model, which was built using the *OneNotes* Corpus, a dataset of labelled 1,745, 000 text data from a range of sources such as telephone conversations, newswire, broadcast news, broadcast conversation and web-blogs. The model uses an artificial recurrent neural network (LTSM) on word embeddings to produce the POS tags. An artificial neural network is a type of learning model, inspired by the biological neural network in animal brains. To recognize patterns using labelled examples, neural networks have layers of weighted *nodes*, for which significance is assigned. Input data is fed through the nodes, and depending on their weight, the effect of the input is either dampened or amplified. As the model takes in more data, the weights shift in order to produce the optimal classification [8].

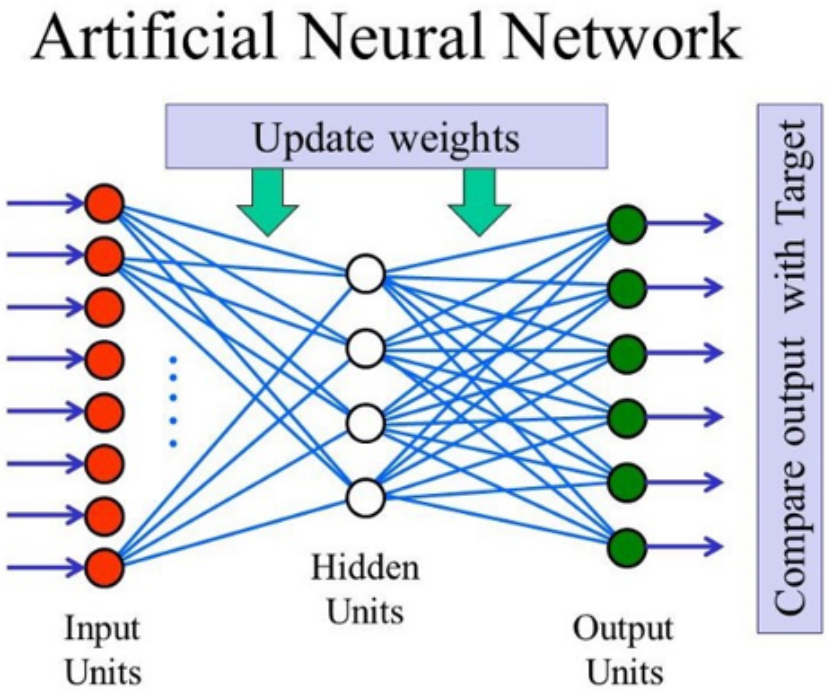

Figure 2.2: Schematic of an artificial neural network ©Edward Tsang

The model used by **Flair** is a type of artificial neural network called a Long Short-Term Memory (LTSM) artificial recurrent neural network. LTSM differs from straight forward neural networks in that it has feedback connections. It can process entire sequences of data (such as an entire sentence) and learn long-term dependencies, as opposed to only single data points (such as a single token) [3] [8]. The POS tagger in **Flair** labels tokens as one of:

PRON : Pronoun	DET : Determiner
NOUN : Noun	ADJ : Adjective
VERB: Verb	PROPN: Proposition
ADP : Adposition	CCONJ : Coordinating conjunction
X: Other	SYM: Symbol
PUNCT : Punctuation	NUM: Number
ADV: Adverb	AUX: Auxiliary
PART : Particle	INTJ: Interjection

2.2 Python Code for Part of Speech Tagging

Listing 2.1: Part of Speech Tagging in Python

```
#Import libaries
from flair.data import Sentence
from flair.models import SequenceTagger

#Create empty dictionary for adding POS tags
POS_dict = {}
#Loop through queries
for indx, query in enumerate(queries_list):
    #Create instance of 'Sentence'
    sentence = Sentence(query)
    #Create a sub dictionary for each query
    sub_dict = {}
    #Predict the sentence
    tagger.predict(sentence)
    #Predict the POS tag
    for token in sentence:
        sub_dict[token.text] = token.get_tag('pos').value
    #Add sub-dictionary to the main dictionary
    POS_dict[indx] = sub_dict
```

Chapter 3

Parse trees

3.1 Introduction to Parse Trees

A parse tree is a representation of the grammatical structure of a sentence. The figure below shows an example of a parse tree. The root of the tree is the sentence, which then branches off into a noun (John) and a verb phrase (hit the ball). The verb phrase is then further broken into a verb (hit) and a noun phrase (the ball). Finally, the noun phrase is broken into a determiner (the) and a noun (ball).

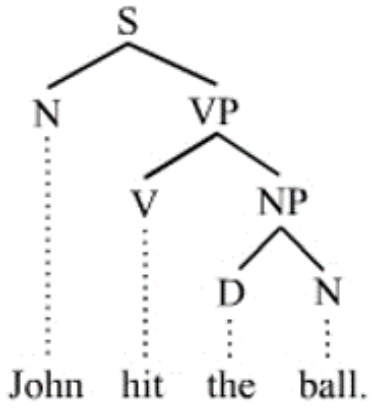

Figure 3.1: Example of a parse tree for the sentence "John hit the ball."

For this project, the Python NLP library **AllenNLP** is used to predict the parse tree for each query [7]. **AllenNLP** uses a constituency based neural parser, as described in the

paper '*A Minimal Span-Based Neural Constituency Parser*' by Stern, Andreas, and Klein [11]. A *span* is an ordered sequence of tokens, and so a constituency tree can be regarded as a collection of labeled spans over a sentence.

3.2 Python Code for Parse Tree Generation

Listing 3.1: Parse tree generation in Python

```
#Create empty dictionary for storing parse trees
Tree_dict = {}
#Loop through the list of queries
for indx, query in enumerate(queries_list):
    #Prediction
    sentence = predictor.predict(sentence = query)
    #Add parse tree to dictionary
    Tree_dict[indx] = sentence['trees']
```

Chapter 4

Agglomerative Heirarchical Clustering

4.1 Jaccard Similarity Measure

In order to cluster the queries based on their part of speech tag, a distance measure must be employed in order to compare which queries are most similar. The proposed solution for this is the Jaccard similarity measure. The Jaccard similarity index is a measure of the size of the intersection between two sets [12]. It is then scaled by dividing by the size of the union of the two sets in order to get the relative distance and have all the values lie between 0 and 1. Values closer to 1 indicate that the sets are similar.

For example, given the 2 sets A and B as follows:

$$A = \{1, 2, 3\} \text{ and } B = \{1, 2, 4, 5\}$$

Then the intersection and union of the two sets would be:

$$A \cap B = \{1, 2, 3, 4, 5\} \text{ and } A \cup B = \{1, 2\}$$

The size of the intersection and union of the sets would therefore be 2 and 5, respectively. The Jaccard index is then calculated as:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{2}{5} = 0.4$$

This similarity measure can then be extended to the POS tags for sentences. For example, take the two sentences A and B:

A = Ottawa is the capital of Canada

B = The cat slept all day

Here, sentence A contains 3 nouns, 1 verb, 1 determiner, and 1 preposition. Sentence B contains 2 nouns, 1 verb, and 2 determiners. Therefore, the intersection between these sentences would be the 2 nouns, 1 verb, and 1 determiner, giving a size of 4, while the union of the sentences would be the 3 nouns, 1 verb, 2 determiners, and 1 preposition, giving a size of 6. The Jaccard similarity index would then be equal to $2 \div 3 \approx 0.6666$.

In order to use the Jaccard index when clustering, the Jaccard *dissimilarity* is used as a measure of *distance*. The Jaccard dissimilarity is essentially the opposite of the Jaccard similarity, and is calculated by subtracting the similarity index from one. On the scale of 0 to 1, a dissimilarity value of 0 means that the sets are identical while a value of 1 means that the sets are not at all similar. In the example above, sentences A and B would have a Jaccard dissimilarity measure of $1 - 0.666 = 0.333$.

4.2 Distance Matrix

Pairwise distances between all of the queries can be represented in a distance matrix. A distance matrix is a symmetric square matrix that contains pairwise distances taken from elements of a set. The diagonal entries of the matrix have a value of 0, because each element has a distance of 0 with itself. A simple example of this is given below.

Sentence Index	1	2	3	4	5	6
1	0	0.666	0.833	0.59	0.35	0.5
2	0.666	0	0.25	0.3	0.3	0.83
3	0.833	0.25	0	0.2	0.666	0.55
4	0.59	0.3	0.2	0	0.55	0.666
5	0.35	0.3	0.666	0.55	0	0.33
6	0.5	0.83	0.55	0.666	0.333	0

Figure 4.1: Example of a distance matrix for 6 sentences using the Jaccard dissimilarity

4.3 Agglomerative Hierarchical Clustering

Clustering is a method of grouping a set of objects in a way that objects in the same group are more similar to each other than those in other groups. Aggolmerative hierarchical clustering is a method that builds a hierarchy of clusters by starting with each observation as a cluster and merging clusters as one moves up the hierarchy [14].

Distances between observations and clusters must be calculated in some way. A common distance measure is the euclidean norm for numeric data. In this project, the Jaccard dissimilarity measure will be used as the distance measure.

There are several criteria that can be used for grouping the most similar clusters. In this project, the complete linkage criteria was used. In complete linkage, the distance between

clusters is chosen to be the maximum distance between all possible pairs in the two groups. This method is also known as farthest neighbour clustering [14].

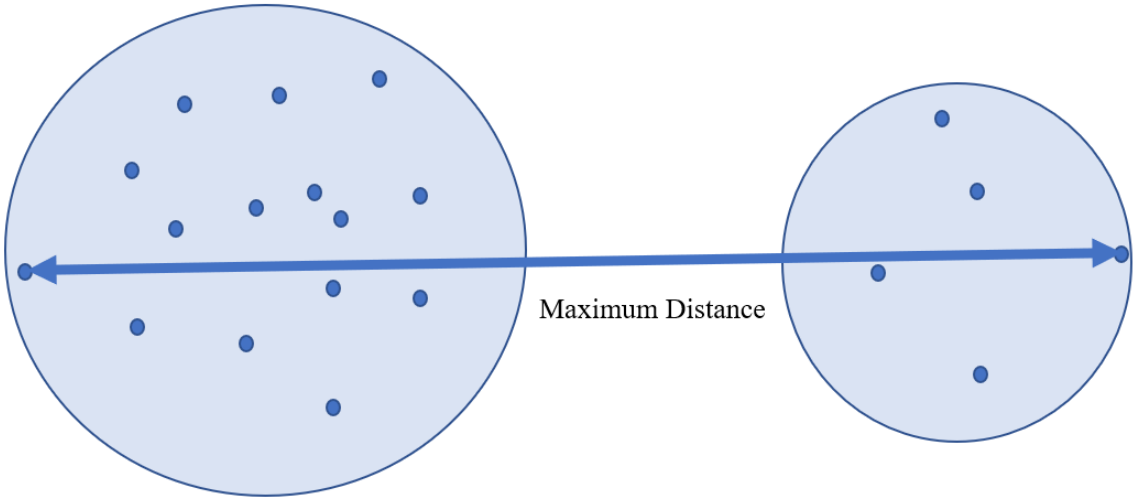

Figure 4.2: Example of cluster distance defined by complete linkage

4.4 Calinski-Harabasz Index

Since hierarchical clustering is a "bottom-up" approach, we need to choose the ideal clustering number. We want to choose an ideal number of K clusters so that the variability between clusters is maximized while the variability within each cluster is minimized. There is a trade-off, since increasing the number of clusters will decrease the intra-cluster variability, however it will unfortunately also decrease the inter- cluster variability [14].

The Calinski-Harabasz (CH) Index is used to provide an ideal number of clusters which will provide an ideal trade-off between a high level of inter-cluster variability and a minimum level of intra-cluster variability [13]. The ideal number of clusters is found by maximizing the CH Index, which is defined as:

$$CH = \frac{SS_B}{SS_w} \times \frac{N-k}{k-1}$$

Where SS_B is the overall inter-cluster variance, SS_W is the overall intra-cluster variance, N is the number of observations, and k is the number of clusters [13].

The CH index was calculated for numbers of clusters, and a value of k = 3 clusters produced the largest CH value of 932.7, as seen in the figure below.

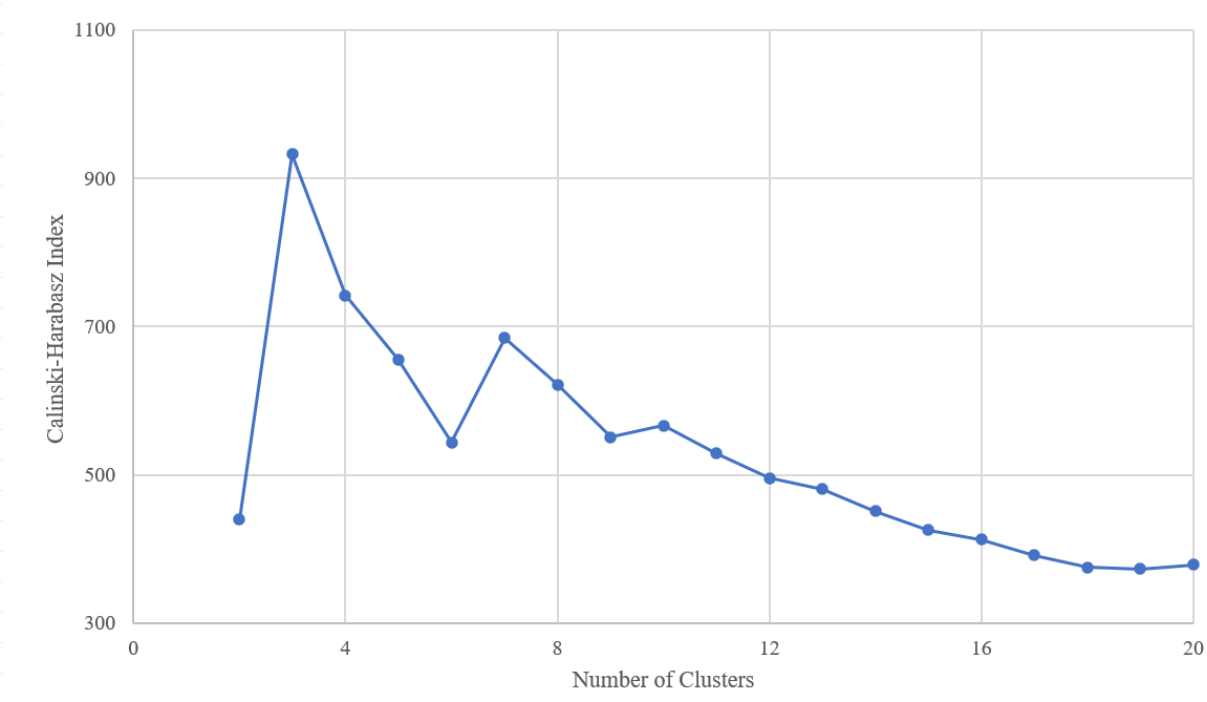

Figure 4.3: A plot of the CH index for various numbers of clusters

4.5 Python Code for Jaccard Dissimilarity and Hierarchical Clustering

Listing 4.1: Preparing Part of Speech Tag Matrix for Jaccard Similarity

```
import pandas as pd

#Extract POS tags from dictionary
pos_complete_list = []
for p_id, p_info in POS_dict.items():
    for key in p_info:
        pos_complete_list.append(p_info[key])

#Remove duplicates
pos_complete_list = list(dict.fromkeys(pos_complete_list))

#Create dataframe with tags as columns and queries as rows
POS_count = pd.DataFrame(queries_list)

for POS in pos_complete_list:
    POS_count[str(POS)] = ""

#Populate dataframe
for indx, info in POS_dict.items():
    temp_list_pos = []
    for key in info:
        temp_list_pos.append(info[key])
    for POS in pos_complete_list:
        POS_count[str(POS)][indx] = temp_list_pos.count(POS)
```

Listing 4.2: Jaccard Dissimilarity Matrix

```
import pandas as pd
from scipy.spatial.distance import squareform
from scipy.spatial.distance import pdist

res = pdist(POS_count.drop(POS_count.columns[0], axis=1), 'jaccard')
distance = pd.DataFrame(squareform(res))
```

Listing 4.3: Hierarchical Clustering and CH index

```
import pandas as pd
from sklearn.cluster import AgglomerativeClustering

distance_matrix = distance_subset.as_matrix()

model = AgglomerativeClustering(affinity='precomputed', n_clusters=2,
    linkage='complete').fit(distance_matrix)

#CH score of model
labels = model.labels_
metrics.calinski_harabasz_score(distance_matrix, labels)

#Finding ideal value for k (num of clusters)
for k in range(2, 21):
    hier_model = AgglomerativeClustering(affinity='precomputed',
                                         n_clusters=k, linkage='complete').
                                            fit(distance_matrix)
    labels = hier_model.labels_
    print (k, metrics.calinski_harabasz_score(distance_matrix, labels))
```

Bibliography

[1] Jason Brownlee. Datasets for Natural Language Processing, 2017.

[2] Admond Lee. Why NLP is important and it'll be the future — our future, 2019.

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

[4] Kevin Banks, Richard Chaykowski, and George Slotsve. The disability accommodation gap in canadian workplaces: What does it mean for law, policy, and an aging population? *SSRN Electronic Journal*, 01 2013.

[5] Bernard Marr. Amazing Examples Of Natural Language Processing (NLP) In Practice, 2019.

[6] Michael J. Garbade. A Simple Introduction to Natural Language Processing, 2018.

[7] Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. Allennlp: A deep semantic natural language processing platform. 2017.

[8] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. PHI Learning, 2009.

[9] Ibrahim Sharaf ElDen. Introduction to Natural Language Processing (NLP), 2019.

[10] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. pages 298–307, 01 2015.

[11] Mitchell Stern, Jacob Andreas, and Dan Klein. A minimal span-based neural constituency parser. pages 818–827, 01 2017.

[12] P. K. Verma, S. Agarwal, and M. A. Khan. Opinion mining considering roman words using jaccard similarity algorithm based on clustering. In *2017 8th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–4, 2017.

[13] Xu Wang and Yusheng Xu. An improved index for clustering validation based on silhouette index and calinski-harabasz index. In *IOP Conference Series: Materials Science and Engineering*, volume 569, page 052024. IOP Publishing, 2019.

[14] Rui Xu and Donald C. Wunsch. *Clustering*. IEEE Press, 2009.

[15] Imad Zeroual and Abdelhak Lakhouaja. Data science in light of natural language processing: An overview. *Procedia Computer Science*, 127:82–91, 01 2018.