ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

Ph.D. THESIS

Enis ARSLAN

LEARNING WORD-VECTOR QUANTIZATION: A STUDY IN MORPHOLOGICAL DISAMBIGUATION OF TURKISH

DEPARTMENT OF COMPUTER ENGINEERING

ADANA-2020

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES

LEARNING WORD-VECTOR QUANTIZATION: A STUDY IN MORPHOLOGICAL DISAMBIGUATION OF TURKISH

Enis ARSLAN

Ph.D. THESIS

DEPARTMENT OF COMPUTER ENGINEERING

We certify that the thesis titled above was reviewed and approved for the award of degree of the Doctor of Philosophy by the board of jury on 17/12/2019.

………………………………..…
Assoc. Prof. Dr. Umut ORHAN
SUPERVISOR

……………..….…………
Prof. Dr. Selma Ayşe ÖZEL
MEMBER

…………..……......
Prof. Dr. Mutlu AVCI
MEMBER

………………...……..…
Prof. Dr. Olcay Taner YILDIZ
MEMBER

……………………..
Asst. Prof. Dr. Ali İNAN
MEMBER

This Ph.D. Thesis is written at the Department of Computer Engineering, Institute of Natural and Applied Sciences of Çukurova University.
Registration Number:

Prof. Dr. Mustafa GÖK
Director
Institute of Natural and Applied Sciences

This study supported by Çukurova University Scientific Research Projects Unit.
Project No: FDK-2015-5155

Note: The usage of the presented specific declarations, tables, figures, and photographs either in this thesis or in any other reference without citation is subject to "The law of Arts and Intellectual Products" number of 5846 of Turkish Republic

ABSTRACT

Ph.D. THESIS

LEARNING WORD-VECTOR QUANTIZATION: A STUDY IN MORPHOLOGICAL DISAMBIGUATION OF TURKISH

Enis ARSLAN

ÇUKUROVA UNIVERSITY
INSTITUTE OF NATURAL AND APPLIED SCIENCES
DEPARTMENT OF COMPUTER ENGINEERING

Supervisor : Assoc. Prof. Dr. Umut ORHAN
Year: 2019, Pages: 105
Jury : Assoc. Prof. Dr. Umut ORHAN
: Prof. Dr. Selma Ayşe ÖZEL
: Prof. Dr. Mutlu AVCI
: Prof. Dr. Olcay Taner YILDIZ
: Assist. Prof. Dr. Ali İNAN

Nowadays, most of the NLP applications are dependent on the accurate morphological analysis of the basic language units: words. Root words, part-of-speech (POS) tags and morphological features are the basic units of a word. Morphologically complex languages like Turkish have rich feature sets. When combined with productive inflectional and derivational morphology, thousands of words can be produced from a root word and this leads to sparsity. Morphological analyzers are the tools that perform the morphological analysis of a word. They can produce multiple parses for a single word where this indicates ambiguity. Disambiguation is the removal process of ambiguity where it is a much complicated task for morphologically complex languages like Turkish. Although high accuracy values are obtained for the studies performed on this task, there is still a challenge. Sparsity and insufficiency of high volume supervised data is the cause of longer running times and accuracy loss. Recent studies for morphological disambiguation are generally presented on neural learning models. To our best knowledge, a disambiguation method which takes the advantage of training of words in a vector-space has not been proposed. Motivated by this shortcoming, in this thesis, we have developed and implemented a vector-space model that solves morphological ambiguity by locating the correct candidates of ambiguous words near to the unambiguous neighbors. The model, named learning word-vector quantization (LWQ), is an adaptation of a well-known learning algorithm, learning vector quantization (LVQ). LWQ outperforms the algorithms presented in the literature for the morphological disambiguation of Turkish.

Key Words: Morphological disambiguation, Complex morphology, Learning vector quantization, Word vector, Ambiguity

ÖZ

DOKTORA TEZİ

SÖZCÜK VEKTÖRÜ NİCELLEŞTİRME ÖĞRENMESİ: TÜRKÇE İÇİN BİÇİMBİRİMSEL BELİRSİZLİK GİDERME ÇALIŞMASI

Enis ARSLAN

ÇUKUROVA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

Danışman : Doç. Dr. Umut ORHAN
Yıl: 2019, Sayfa: 105
Jüri : Doç. Dr. Umut ORHAN
: Prof. Dr. Selma Ayşe ÖZEL
: Prof. Dr. Mutlu AVCI
: Prof. Dr. Olcay Taner YILDIZ
: Dr. Öğr. Üyesi Ali İNAN

NLP uygulamalarının başarısı, dillerin temel birimi olan kelimelerin doğru biçimbirimsel analizine bağlıdır. Kökler, kelime türü etiketleri ve biçimbirimsel özellikler, bir kelimenin temel birimleridir. Türkçe gibi biçimbirimsel olarak karmaşık olan diller zengin özelliklere sahiptir. Türkçe'nin türetimsel olarak üretken yapısı gözönüne alındığında, bir kök kelimeden binlerce kelime üretilebilmekte ve bu durum seyrekleşmeye yol açmaktadır. Biçimbirimsel analizörler, bir kök kelimenin biçimbirim analizini yapan araçlardır. Biçimbirimsel analizörler, tek bir kelime için birden fazla ayrıştırma üretebilir ve bu durum ise belirsizliği göstermektedir. Belirsizlik giderme işlemi, Türkçe gibi morfolojik olarak karmaşık diller için oldukça zor bir işlemdir. Bu problemin giderilmesi için sunulan çalışmalarda yüksek doğruluk değerleri elde edilmiş olmasına rağmen, daha gidilecek yol vardır. Seyreklik ve yüksek miktarda denetimli verinin bulunmuyor olması, daha uzun çalışma sürelerine ve daha düşük doğruluk değerlerine sebep olabilmektedir. Son zamanlarda biçimbirimsel belirsizliklerin giderilmesi çalışmaları genellikle sinir öğrenme modelleri ile yapılmaktadır. Bildiğimiz kadarıyla, Türkçe için, kelimelerin vektör uzayında eğitilerek konumlandırılmasıyla biçimbirimsel belirsizliği gideren bir yöntem henüz önerilmemiştir. Bu eksiklikten hareketle, bu tezde, belirsiz kelimenin doğru adaylarını belirsiz olmayan komşuların yanına yerleştirerek biçimbirimsel belirsizliği çözen bir vektör uzay modeli geliştirilmiş ve uygulanmıştır. Sözcük vektörü nicelleştirme öğrenmesi (LWQ) adlı model, iyi bilinen bir öğrenme algoritması olan vektörel nicelleştirme öğrenmesi (LVQ)'nin bir türevidir. LWQ, literatürde sunulan diğer algoritmalara göre daha iyi başarı oranları elde etmektedir.

Anahtar Kelimeler: Biçimbirimsel belirsizlik giderme, Karmaşık biçimbirim, Vektörel nicelleştirme öğrenmesi, Kelime vektörü, Belirsizlik

EXPANDED ABSTRACT

Turkish is an agglutinative language which has a productive character with inflectional and derivational morphology. In agglutinative languages, new word forms can be easily created by using stems and affixes. Words carry semantical and syntactical information in their internal structure. This information is valuable in advanced natural language processing (NLP) applications and must be exposed. Morphological analysis is a pre-processing step for NLP which identifies the grammatical structure of words. Finite state machines (FSM) are generally the main components of a morphological analyzer and their design varies according to the language’s grammar. Due to the large size of dictionaries and morphological features many analysis (parse outputs) can be provided by a morphological analyzer. Although morphological analyzers are improved with additional constraints, multiple parses of a word are often available for a word, which is stated as “ambiguity”.

Especially, nearly half of the running text is ambiguous in Turkish. Inflectional morphology and large feature sets are the main causes of sparsity. Ambiguity is a barrier for further processing of texts in NLP applications and it should be solved. Morphological disambiguators are the tools that singularize the multiple parses provided by a morphological analyzer by selecting the correct parse solution. Morphological disambiguators implement disambiguation by taking into account the context of a target (ambiguous) word in a sentence or text. Nearly all of the studies performed for Turkish have used context information in their models by incorporating the neighboring information. Applying a conceptual window on the target word to detect its corresponding neighbors, is a general approach.

Early studies on morphological disambiguation of Turkish has begun with constraint-based (Oflazer and Kuruöz 1994; Oflazer and Tür, 1997; Daybelge and Cicekli, 2007) approaches and pure statistical methods (Hakkani-Tür et al., 2002). The main drawback of using constraints in disambiguation is difficulty to handle

new or unknown words with the standard rule-sets. Statistical methods collect usage information of the data models (word n-grams) from the corpus and determine the most probable sequence to disambiguate a sentence. In the following years, hybrid (Kutlu and Cicekli, 2013) models have got more popular which achieves higher success by using the constraints with statistical information. Nearly a decade ago, machine learning methods (Sak et al., 2007; Yuret and Türe, 2006) has presented the state-of-the-art results. In recent years, neural network models (Dayanık et al., 2018; Shen et al., 2016; Yildiz et al., 2016) report promising accuracy values in morphological disambiguation. These models have the advantage of efficient computation and flexibility of modelling the words with context information in network layers. Nearly all of these use the same tagged datasets in their training cycles (Yuret and Türe, 2006). Although most of these algorithms are so successful, any implementation of morphological disambiguation with a classification method which applies a supervised reward-punish mechanism in a vector-space has not been proposed, to our best knowledge.

Motivated by this shortcoming in the literature, we propose to develop, a vector-space model named learning word-vector quantization (LWQ), which is an adaptation of the learning vector quantization (LVQ). Although original LVQ algorithm has been successfully applied in some NLP research fields like text classification (Umer and Khiyal, 2007; Visa et al., 2000; Martín-Valdivia et al., 2007; Pilevar et al., 2009), speech recognition (Haldar and Mishra, 2016), language identification (Gunawan et al., 2017), spam detection (Chuan et al., 2005) and multi-word expressions recognition (Diaz-Galiano et al., 2004), it is not implemented on morphological disambiguation, in our knowledge. For this reason, we have adapted this algorithm specifically for this task, by treating it as a classification problem.

In this thesis, first, we have built the necessary tools and materials. These are necessary to improve a morphological analyzer and prepare a tagged dataset with a web tool. Second, we have developed the LWQ classification model and

trained it with this dataset to optimize the exact locations of the words in space. We believe that words without ambiguity should reside in near locations with the correct parse candidate of an ambiguous word. When the locations of the words are fixed at the end of the training phase, a classification test is applied to disambiguate the ambiguous words. To our best knowledge, this is the first implementation of this technique.

The experimental results show that the proposed technique gives promising accuracy values in morphological disambiguation. Although we have used a limited dataset, training with a larger dataset can contribute to success. In the study, accuracy results are presented with various experiments and data consistency is additionally investigated.

GENİŞLETİLMİŞ ÖZET

Türkçe türetimsel ve çekimsel biçimbirimine sahip sondan eklemeli bir dildir. Sondan eklemeli dillerde kökler ve ekler kullanılarak kolaylıkla yeni kelimeler oluşturulabilmektedir. Kelimeler iç yapılarında anlamsal ve sözdizimsel bilgiler içermektedirler. Bu bilgiler doğal dil işleme (DDİ) uygulamalarında değerli olmaları sebebiyle açığa çıkartılmalıdır. Biçimbirimsel analiz, sözcüklerin gramatik yapılarının tespit edildiği bir DDİ ön işlemidir. Sonlu durum makinaları (SDM) bir biçimbirimsel analizcinin temel yapıtaşlarıdır ve tasarımları uygulandığı dillerin dilbilgisine göre değişiklik göstermektedir. Bir biçimbirimsel analizci sözlüklerinin büyük olması ve özellik sayısının fazlalığı sebebiyle birçok analiz çıktısı oluşturabilmektedir. Her ne kadar biçimbirimsel analizciler birçok kısıtlayıcı kurallar kullanılarak geliştirilmiş olsalar da, bir kelimenin birden fazla analizi sözkonusu olabilmektedir ve bu duruma "belirsizlik" adı verilmektedir.

Neredeyse Türkçe olarak kullanılan metinlerin yarısı belirsizlik içermektedir. Çekimsel biçimbirim ve fazla özellik içeren özellik kümeleri seyrekliğin temel sebeplerinden biridir. Belirsizlik ileri DDİ uygulamalarına geçebilmek için bir engeldir ve çözülmesi gerekmektedir. Biçimbirimsel belirsizlik gidericiler bir biçimbirimsel analizcinin sunduğu birden fazla çözüm içinden doğru çözümün bulunması için kullanılan araçlardır. Biçimbirimsel belirsizlik gidericiler bir cümledeki ya da metindeki bir sözcüğün bağlamını gözönüne alarak hedef (belirsiz) kelimenin belirsizliğini gidermektedirler. Türkçe için bu konuda yapılan çalışmaların neredeyse hepsi tasarımlarında komşuluk ilişkilerinin kullanıldığı bağlam bilgisinden faydalanmaktadırlar. Komşularının tespiti için hedef kelimeye kavramsal bir pencere uygulanması genel bir yaklaşımdır.

Türkçe'nin biçimbirimsel belirsizlik gidermesi üzerinde ilk yapılan çalışmalar kural tabanlı (Oflazer ve Kuruöz 1994; Oflazer ve Tür, 1997; Daybelge ve Cicekli, 2007) ve istatistiksel (Hakkani-Tür vd., 2002) olarak sunulmuştur. Belirsizlik gidermede kural tabanlı yaklaşımları kullanmaktaki temel sorun, yeni

veya bilinmeyen kelimelerin standart kural tablolarıyla tespitinin zor olmasıdır. İstatistiksel yöntemler veri modellerinin (n-gramlar) derlemlerdeki kullanım bilgilerini toplayarak, cümledeki belirsizliği gidermek için en olası dizilimi belirlemektedir. Takip eden yıllarda, kural tabanlı ve istatistiksel yöntemleri kullanarak yüksek başarılar sağlayan hibrid (Kutlu ve Cicekli, 2013) modeller önem kazanmaya başlamıştır. Yaklaşık on yıl kadar önce sunulan makine öğrenmesi yöntemleri (Sak vd., 2007; Yuret ve Türe, 2006) en yüksek başarı değerlerini sunmuştur. Yakın zamanlarda belirsizlik giderme için sunulan sinir ağ modelleri (Dayanık vd., 2018; Shen vd., 2016; Yildiz vd., 2016) umut verici doğruluk değerleri sunmuşlardır. Bu modeller, sinir ağlarında kelimelerin bağlam bilgisiyle beraber verimli bir şekilde modellenmesine olanak sağlamaktadırlar. Bu modellerin neredeyse hepsi aynı etiketli verisetini (Yuret ve Türe, 2006) kullanmaktadırlar. Bu çalışmaların çoğu oldukça başarılı sonuçlar sunmuş olmalarına rağmen, bildiğimiz kadarıyla, kelime uzayında denetimli olarak ödül-ceza uygulaması yapan bir belirsizlik giderme işlemi yapılmamıştır.

Literatürde bulunan bu eksiklik göz önüne alınarak, bu çalışmada, vektör nicelleştirme öğrenmesi (LVQ) algoritmasından esinlenilerek, sözcük vektörü nicelleştirme öğrenmesi (LWQ) modeli geliştirilmiştir. LVQ algoritması, her ne kadar metin sınıflama (Umer ve Khiyal, 2007; Visa vd., 2000; Martín-Valdivia vd., 2007; Pilevar vd., 2009), ses tanıma (Haldar ve Mishra, 2016), dil tanıma (Gunawan vd., 2017), spam tespiti (Chuan et al., 2005) ve çoklu kelime tanıma (Diaz-Galiano vd., 2004) gibi bazı DDİ araştırma alanlarında uygulanmış olsa da bildiğimiz kadarıyla biçimbirimsel belirsizlik gidermede bir uygulaması bulunmamaktadır. Bu sebeple, bu algoritma, belirsizlik giderme problemi için bir sınıflandırma bakış açısıyla değiştirilmiş ve yeni bir model olarak sunulmuştur.

Bu tezde, öncelikle gerekli araçlar ve malzemeler hazırlanmıştır. Bunlar, bir biçimbirimsel analizcinin üzerinde geliştirme yapılması ve bir web aracı yardımıyla etiketli veri hazırlanması için gerekmektedir. İkinci olarak, LWQ adlı sınıflama modeli geliştirilerek veriseti ile eğitim yapılmış ve uzayda kelimelerin

gerçek konumları optimize edilmeye çalışılmıştır. Belirsizlik içermeyen kelimeler belirsiz kelimenin doğru adayının çevresinde bulunmaktadır. Eğitim sonunda kelimelerin uzaysal konumlarının sabit hale gelmesiyle belirsiz kelimelere bir sınıflandırma testi uygulanmakta ve belirsizlik giderilmektedir. Bildiğimiz kadarıyla, bu tekniğin ilk uygulaması bu çalışmada sunulmaktadır.

Deneysel sonuçlar, çalışmada önerilen tekniğin biçimbirimsel belirsizlik gidermede umut verici doğruluk değerleri sağladığını göstermektedir. Sınırlı bir veri kümesi kullanılmış olmasına rağmen daha yüksek veri ile eğitim yapılması başarıyı arttırabilecektir. Bu çalışmada başarı değerleri farklı deneyler ile sunulmuş ve veri tutarlılığı ayrıca araştırılmıştır.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Assoc. Prof. Dr. Umut ORHAN for his supervision, encouragement, patience, motivations and useful suggestions for the successful completion of this work.

I would like to thank members of the Ph.D. thesis jury, Prof. Dr. Selma Ayşe ÖZEL, Prof. Dr. Mutlu AVCI, Prof. Dr. Olcay Taner YILDIZ, and Assist. Prof. Dr. Ali İNAN for their valuable help, support and suggestions.

I would like to express my sincere appreciation to TUBİTAK for their financial support with the project number 215E256 in the process of my Ph.D. education.

I am also thankful to Dr. Çağatay Neftali TÜLÜ and Erhan TURAN for their support and motivation.

Finally, I would like to thank my dear brother Prof. Dr. Niyazi Arslan for encouraging me in the Ph.D. study and his personal assistance, also to other members of my family who have supported me throughout the entire thesis period, both by their love and encouragement.

CONTENTS **PAGE**

LIST OF TABLES **PAGE**

LIST OF FIGURES **PAGE**

LIST OF ABBREVIATIONS

AVF	: Affix Validation Function
CNN	: Convolutional Neural Network
COC	: Consistency of Classification
CRF	: Conditional Random Fields
DB	: Derivational Boundary
FSM	: Finite State Machine
GPA	: Greedy Prepend Algorithm
HMM	: Hidden Markov Model
IG	: Inflectional Group
LDF	: Lemma Discovery Function
LSTM	: Long Short Term Memory
LVQ	: Learning Vector Quantization
LWQ	: Learning Word-Vector Quantization
MD	: Morphological Disambiguator
MF	: Morphological Feature
NER	: Named Entity Recognition
NLP	: Natural Language Processing
NN	: Neural Network
OOV	: Out-of-Vocabulary
POS	: Part of Speech
SMT	: Statistical Machine Translation
TLA	: Turkish Language Association
UW	: Unknown Word
WSD	: Word Sense Disambiguation

1. INTRODUCTION

Turkish is a morphologically complex, agglutinative, free word order language. It is possible to derive infinite number of words by using a root and any number of morphemes (Hakkani-Tür et al., 2002). For advanced Natural Language Processing (NLP) applications, it is an essential task to morphologically analyze the words in texts (Dayanık et al., 2018). Morphological analysis is composed of a root, part-of-speech (POS) and morphological tags (i.e. morphemes).

Morphological analyzers and part-of-speech (POS) taggers are the main tools that analyze a word morphologically. In terms of NLP, morphological ambiguity is the case that, when there exist multiple analyses as the outputs of these tools. Morphological ambiguities should be solved (i.e disambiguated) in order to perform further applications in NLP. Morphological disambiguation is the process of finding the correct parse when a word has multiple parses within the context to which it is bound (Dayanık et al., 2018).

Morphological disambiguation can be achieved in two ways. In some cases, it is necessary to determine the POS tag for a root word or lemma. This process is called POS Tagging and it is helpful to determine the correct POS tag for a lemma provided by a lemmatizer tool. The other case is, to solve ambiguities given as output of a morphological analyzer. Morphological disambiguator tools have been developed to solve this problem. These tools, with various methods and perspectives, can detect the appropriate analysis, regarding the situation of the ambiguous word in the relevant context. POS tagging and morphological disambiguators are often used together because their output is somewhat the same. For example, the output of the POS tagging process is “lemma + POS tag”, while the output of the morphological analyzer is “lemma + POS + MF (Morphological Features”. The usage areas of the outputs may show similarities or differences. Conceptually, POS tagging and morphological disambiguation are acceptable at the same level in an NLP pipeline (Yildiz et al. 2016). But, for morphologically complex languages POS tagging may not be enough and morphological disambiguation is performed directly on the output of a morphological analyzer.

Morphological disambiguation studies have come up with various approaches. However, in recent studies, neural network based machine learning methods are preferred, which encode context information on word embeddings, instead of directly using a morphological analyzer output (Dayanık et al., 2018; Heigold et al., 2017; Yıldız et al. 2016). As a result of this preference, higher accuracy values are reported when compared to statistical and rule-based methods.

Context information can be implemented locally or globally on the disambiguation process. Local application is provided by the addition of the local features of the ambiguous word by using a conceptual window. This is useful in enriching the information for a target word by using the local features of the neighboring words. This kind of window can be applied in one side (left or right) or two sides (both right and left) of the target word depending on the design. In the global sense, it is the use of words that are semantically similar, which co-occur in the same sentence or text.

In literature, many studies presented for morphological disambiguation of Turkish benefit from windowing. In Yuret and Türe (2006) they use all the information of the words which reside in a five-word window. In Görgün and Yıldız (2011) and Yildiz et al. (2016), a three-word window is used which includes the two neighboring words to the left of the target word. In Dayanık et al., (2018)'s study one neighbor word from the left and right of the target word are used in encoding to the neural model.

Morphological disambiguation of Turkish with classification techniques in vector-space has not been studied yet, in our best knowledge. Motivated by this shortcoming, in this study, we propose a classification method to solve the morphological ambiguity problem of the Turkish language. This method is an adaptation of the well-known machine learning algorithm called Learning Vector Quantization (LVQ) (Kohonen, 1984). LVQ is a competitive supervised neural network, which is generally used for classification tasks. Inspired from the LVQ algorithm we have developed LWQ (by using a similar acronym) method and applied to morphological disambiguation problem. Basically, LWQ takes advantage of a vector-space to obtain the optimum locations of word-vectors by using a reward-punish mechanism. As mentioned before, words without ambiguity are helpful in determining the correct candidate of an ambiguous word (a word with ambiguity). In the inspiration of this idea, word-vectors are trained to assure that, unambiguous words are located near to the correct lemma candidate of an ambiguous word in the space. This requires a supervised tagged corpus where training datasets are prepared specially and word-vectors are obtained from the vocabularies of these datasets. Each data line in a dataset represents a single occurrence of an ambiguous word with its immediate unambiguous neighbors that are all in a pre-defined window. Lemma candidates of the ambiguous word that are in the data line are obtained from the morphological analyzer (Yıldız et al., 2019). All these candidates can be thought of a class and they are variable in the count because there is no limit in the distinct parses provided by a morphological analyzer. Original LVQ algorithm enables to design a model which is not limited to two classes. This advantage of LVQ is an inspiration used in the design of LWQ.

LVQ is a three-layer neural network that is based on Kohonen Self Organizing maps and it is supervised. Basically, these are input, competitive (Kohonen) and output layers. Kohonen layer includes the codebook vectors where each of them represents a cluster in the space. The Euclidian distance of each input sample to each code vector is calculated and the nearest codebook vector wins the competition test. The winner codebook vector is always compared with the desired output and the weights are modified accordingly. LVQ is advantageous than the other classification methods in requiring fewer training examples, the ability to handle boundary values and being faster (Umer and Khiyal, 2007). In the literature there are several successful implementations of LVQ methods on NLP tasks such as text classification (Umer and Khiyal, 2007; Visa et al., 2000; Martín-Valdivia et al., 2007; Pilevar et al., 2009), speech recognition (Haldar and Mishra, 2016), language identification (Gunawan et al., 2017), spam detection (Chuan et al., 2005) and multi-word expressions recognition (Diaz-Galiano et al., 2004).

Our study is different from the original LVQ algorithm as follows. First of all, we do not use a neural network model although the logic is nearly the same. Lemma candidates of all ambiguous words in a dataset are represented as similar to codebook vectors of LVQ. Vector-space positions of the ambiguous lemma candidates are arranged according to the supervised knowledge in each data line of the dataset. Each data line represents a correct ambiguous lemma candidate together with the other candidates and their immediate neighboring. When training begins, a calculation is made for each ambiguous lemma candidate independently with the neighbor words (unambiguous words). The calculation with the lowest distance value is the selection of the LWQ system. It is compared with the correct lemma candidate (supervised), if they are consistent to be same, nothing is done. Otherwise, word-vector of the correct lemma candidate is approximated to the neighbor words while other candidates are departed in opposite directions. This iteration goes on till the convergence occurs. To evaluate our results, we have used the “accuracy” parameter in order to comply with the other studies implemented for Turkish morphological disambiguation in the literature.

Since the proposed method can work on labelled data, there is a need for low noise and highly accurate dataset. There is a semi-automatically disambiguated dataset (about 1 million words) released in Turkish (Yuret and Türe, 2006) and most of the studies for Turkish (Sak et al. 2007; Shen et al., 2016; Yildiz et al. 2016) use this dataset. While using this dataset, some of the preceding studies presented their work by applying various methods to reduce noise or preparing more reliable test sets. Because of the low reliability of the presented datasets for Turkish, we have prepared a manually disambiguated data by the help of a tagging tool and Expert.

1.1. The Aims and Objectives of This Thesis

Morphological disambiguation is a challenging NLP problem that is more laborious and hard for morphologically complex languages like Turkish, Hungarian and Czech due to the richness of the tagsets and their inflectional character. When the case is Turkish, theoretically infinite number of words can be derived from the stems or lemmas in a dictionary. Generally, statistics of the local features of a word are enriched with the context information and used in the disambiguation models. This requires high volumes of training data and special data models for Turkish. Although recent disambiguation methods (Sak et al., 2017; Shen et al., 2016; Dayanık et al., 2018; Görgün and Yıldız, 2011) are quite successful, using word-space in classification for morphological disambiguation is not well-studied. Therefore, the aim of this study is to develop a classification technique named Learning word-vector quantization (LWQ), which is an adaptation of the well-known supervised machine learning algorithm, Learning vector quantization (LVQ). To achieve the best results, we aimed to take advantage of the LVQ algorithm which requires less training data, flexible in class numbers and function with a reward-punish mechanism.

LWQ is similar to LVQ in optimizing the locations of word prototypes as input vectors but differs in some way. Training in LWQ is achieved by using the immediate neighboring of the ambiguous words (target words) which do not have ambiguity. This is not a new idea (Viterbi, 1967), where some of the successful studies (Hakkani-Tür, 2002; Sak et al., 2017) have established their base models on scoring parse sequences with disambiguated text which with words without ambiguity. Also, we have used the reward-punish mechanism in arranging the word prototypes in space. We believe that using this approach in morphological disambiguation contributes to the classification accuracy.

1.2. Our Contribution

Studies covered in this thesis aim to solve the morphological ambiguity problem of Turkish, which is a challenging task needed to focus on for further NLP studies. Thinking of the parse candidates of a word provided by a morphological analyzer as classes and their immediate neighbouring (which do not have ambiguity) as input words, we have developed a classification technique which selects the correct candidate of an ambiguous word. We name it Learning word-vector quantization (LWQ), in the inspiration of the Learning vector quantization (LVQ) algorithm. Basic idea is to locate all words, as prototypes, in the vector space and train the system by considering the relationships of the parse candidates of an ambiguous word and their immediate neighbours. LWQ inspires the reward-punish mechanism LVQ uses. In our knowledge, there is not any study in the literature, which uses the main ideas of LVQ in classification to solve morphological ambiguity.

1.3. Thesis Organization

This thesis is organized as follows:

In Section 2, a literature overview of studies on morphological disambiguation of Turkish is provided. Methodologies are explained in detail and accuracy values are given.

In Section 3, brief information is given for the materials and tools used in the study. Datasets presented for Turkish morphological disambiguation are reviewed with statistics. Also, the dataset prepared for this study is given. At the end of the section, detailed information is given about the existing and developed tools used for the preparation of the dataset.

In Section 4, the proposed LWQ algorithm is given in detail. Multiword tokenizer that is integrated to the morphological analyzer is explained. Out-of-vocabulary (OOV) discovery framework is presented.

In Section 5, the experimental results of the presented algorithm are given and they are compared with the similar methods in the literature.

Finally, in the Conclusion section, the advantages and disadvantages of the proposed methodology are discussed following the future outlook.

2. RELATED WORKS

In this section, research on morphological disambiguation presented for Turkish is summarized. Before the overview, it is better to focus on the morphological ambiguity problem of Turkish.

Turkish is a morphologically complex, productive and inflectional language. For a natural language, word-parts (i.e. Morphemes) are the smallest meaningful units which can be a stem, a prefix or suffix. Morphologically complex languages carry syntactical and semantical relations in morphemes. To expose this information, a word should be analyzed by using a morphological analyzer. Analyzers are generally based on finite state machines (FSM) and many constraints special to the language. Morphological analyzers present all possible solutions of a word as parse lists, by applying all possible rules without considering the context.

Due to the very large tag set of Turkish, theoretically, infinite number of words can be derived (Hakkani-Tür et al., 2002) and this leads to sparseness. Multiple parse solutions of a word indicate ambiguity and nearly half of the words in the running text in Turkish are ambiguous (Yuret and Türe, 2006). Morphological ambiguity should be solved by morphological disambiguation, for further NLP applications such as syntax and semantic parsing, word sense disambiguation, text-to-speech recognition, machine translation, spell checking, dependency parsing, text summarization, semantic role labeling, topic modeling and named entity recognition (NER). Some well-known examples of an ambiguous word (a word with ambiguity) are given in Table 2.1 and Table 2.2.

Table 2.1 Morphological analysis of the word “masalı” (Dayanık et al., 2018)

Word	Analysis output
masalı	masal+Noun+A3sg+Pnon+Acc masalı yaz. (write the tale.)
	masal+Noun+A3sg+P3sg+Nom babamın masalı (my father’s tale)
	masa+Noun+A3sg+Pnon+NomˆDB+Adj+With mavi masalı oda (room with blue table)

In Table 2.1, the first two analyses have the same roots and POS tags as “masal” and “NOUN”, respectively. But they are different in being accusative or nominative and this tag difference causes a change in the meaning of the word. The third analysis has a different root as “masa” and analysis has a derivational boundary (DB) which means that a new word can be derived from the root with a different affix (or different tag sequence) (+With) and this new word is an adjective.

Table 2.2. Morphological analysis of the word “dolar” (Yildiz et al., 2016)

Word	Analysis output
dolar	dolar +Noun +3sg +Pnon +Nominative on milyon dolar borcu var (has a debt of ten million dollars)
	dola +Verb +Positive +Aorist +3sg ayağına dolar (she wraps to her foot)
	dol +Verb +Positive +Aorist +3sg bardak dolar (cup fills)
	do +Noun +3pl +Pnon +Nominative Multiple müzik notası (musical note)

In Table 2.2, analysis of the foreign word “dolar” can be seen. There are four different roots as “dolar”, “dola”, “dol”, “do”. It is a good point to imply the second and third analyses where they have the same tag sequence as “Verb +Positive +Aorist +3sg”, a new word with a different meaning can be derived by using the roots “dola” and “dol”. There is no limitation in analyze count of a morphological analyzer, where derivation starts with all the possible words and corresponding POS tags listed in the dictionary, and FSMs increase the analysis count and sparsity by considering all the possible paths in automatas.

As mentioned before, when the morphological analyzer outputs a single parse, the term is not ambiguous and when there is more than one candidate root word or lemma in multiple parses, it can be said that the target word is ambiguous. In this study "unambiguous word" term will be used for the words without ambiguity and the "ambiguous word" term will be used for the ambiguous ones. As an example, morphological analyzer generates one candidate lemma as "arabacı + NOUN" for the word "arabacım". On the other hand, two candidate lemmas can be produced for the ambiguous word "kalemi" as "kale + NOUN" and "kalem + NOUN". Morphological analyzers or lemmatizers can only produce outputs including roots, POS tags, morphological features (tags) without considering the context of the target word. In order to resolve this kind of ambiguities, it is essential to incorporate neighboring words of the target word in a text. Therefore, if we consider each word in the text as a word-vector and each word-vector as a point in space, dividing all points into two groups as "ambiguous" and "unambiguous" may be useful in the disambiguation analysis.

2.1. Morphological Disambiguation of Turkish

Morphological disambiguation studies on agglutinative languages are generally examined in four categories as rule-based, statistical, hybrid and machine learning. Recent studies have been introduced on deep learning methods and competitive results have been obtained when compared to the previous studies.

Previous studies for the Turkish language are based on rule-based (Oflazer and Kuruöz 1994; Oflazer and Tür, 1997; Daybelge and Cicekli, 2007) and statistical methods (Tür et al., 2002). Sometimes statistical methods can be used together with machine-learning methods being called hybrid methods (Kutlu and Cicekli, 2013). In the study, they have reported an accuracy rate of 93.4% in morphological disambiguation using statistical data and handcrafted rules on a dataset they created.

In the later studies, morphological disambiguation success is improved by using machine learning methods and better results are obtained according to the rule-based and statistical methods. In morphological disambiguation, the results reported in Sak et al.'s (2007) study is considered state-of-the-art for Turkish. In the study, a multilayer perceptron method, which votes n-gram models (Hakkani-Tür et al., 2002) is presented with a success rate of 96.8%. When the same train and test sets are used, this study overperformed with the value of 95.93% when compared with the studies of Hakkani-Tür et al. (2002) and Yuret and Türe (2006).

Early work (Oflazer and Kuruöz, 1994; Oflazer and Tür, 1996; Oflazer and Tür, 1997, Daybelge and Cicekli, 2007) on disambiguation is frequently presented as rule-based approaches. These work rely on hand-crafted rules which generally suffer from generality problem. Oflazer and Kuruöz, (1994)'s study is the earliest study which uses constraint grammar approach by checking a target word's agreement with the syntactic and positional restrictions. Although this study achieved reasonable results, the constraints were hand-crafted and further improvement was impossible. In another study (Oflazer and Tür, 1996), they have proposed a constraint-based approach that is capable to learn rules and statistics from the corpus in an unsupervised way. The hand-crafted rules are language independent. The process begins by applying the standard set of choose-delete hand-crafted rules to the untagged corpus. While this procedure provides a level of decrease in ambiguity, additionally, a learning mechanism works to induce more choose-delete rules specific to the target language. Following this procedure, the disambiguation procedure is as: the standard set of rules is applied to the same untagged corpus, some parses are deleted by using context statistics and finally, the newly learned rules are applied. In the study, they report a 96 to 97% accuracy in disambiguation. Voting constraints (Oflazer and Tür, 1997) is another rule-based approach where each rule has several constraints and a vote point. Voting is achieved by giving high points to the most specific rules which have a high number of constraints, the constraints with a high number of features, which have reference to specific features. Rules are applied independently to all parses in the tokens of a sentence. If a parse matches, all the constraints of a rule and the vote value is incremented. Disambiguation is performed by selecting the highest-scoring parse by considering the lowest scoring in a calculation.

An example of a hybrid approach (Kutlu and Cicekli, 2013) incorporates statistics and constraints to their model in order to solve morphological ambiguity. Their system includes two stages as training and disambiguation. Training corpus is used to compose word and suffix tables which hold the most likely parses of the words and suffixes according to their frequency values. All words in the training corpus are pre-tagged by using these tables. A modified Brill tagger learns the disambiguation rules from the corpus with many iterations by applying ten-fold cross-validation. Disambiguation is achieved by using the statistics in word and suffix tables, hand-crafted rules, rules learned by Brill tagger and heuristics. Accuracy rate obtained by using this approach is given as 93.4%. When the final IG of the word is considered, it reaches to 94.1%.

In statistical approaches, a probabilistic model is trained by using labeled or unlabeled data and this model is used to tag a new text. The most well-known study (Hakkani-Tür et al., 2002) for Turkish, which has a statistical approach presents trigram models by splitting the words into inflectional groups (IG) to handle data sparseness problem. In the study, they have developed and tested four models. Hidden Markov models (HMM) technique is used to model the morphological parses in a sentence by maximizing the posterior probability to estimate the variables. The first model assumes that the root of a word is dependent on the roots of the previous two words and at the same time the IG of the word depends on the final IGs of the previous two words. The second model is the same as the first model, except that, the last IG of the target word is dependent on the previous IG of the same word. The third model has the same assumptions with the second one, except that, the IG of a target word is independent of the last IGs of the previous words. The last model is a Naïve Bayes model which assumes that the previous two words' IGs are independent of each other. In the training phase, two types of probabilities are calculated independently from the training data as root and IG probabilities. Two trigram models are used to estimate the root and IG probabilities, and in runtime, a combination of these models and test data is used to estimate the best sequence by using the Viterbi algorithm (Viterbi, 1967). They have acquired the best accuracy value for the first model (93.95%) and the worst value for the Naïve Bayes model as 88.85%. When errors of semantic features are ignored, the accuracy increases to 95.07%.

Yuret and Türe (2006) propose a machine learning method, which combines the rule-based methods with statistics. They have composed 126 distinct decision lists for every 126 morphological features. Training subsets are selected from 1 million training instances which consist of each morphological feature at least once in their corresponding parse. Following that, these sets are divided into two groups (positive and negative) where one group includes the morphological feature in the correct parse and the other would not. Training starts with an empty decision list and a default rule. Greedy Prepend Algorithm (GPA) applies a window of size four by centering the target word. Although they have tried bigger window sizes, there was no significant improvement in accuracy. Rule patterns are discovered for each target word for each morphological feature. The rules are ordered in a way that the decision list is ordered from the most specific rule to the most general. GPA prepends each rule with the maximum gain to the top of the lists. Gain is defined by the correct classification ratio while the rules prepend to the lists. Convergence happens when there is no rule to be added to the lists. Disambiguation is achieved by the product of decision list prediction possibilities of each tag in the parse. This methodology is advantageous to handle unknown words as being free of a dictionary. They have overcome data sparseness problem by composing a single list for each morphological feature instead of processing many tags. Accuracy reported in this study is given as 95.82%.

Another study (Sak et al, 2007), proposes a method, a variant of the Perceptron algorithm which is a well-known machine learning method. They model an ambiguous word into morphemic units, which consist of the roots and the morphosyntactic tags. Their statistical baseline model is the same as in (Hakkani-Tür et al., 2002). They have created various templates in this way, as the features to be used in the perceptron algorithm. Perceptron learns weights for each instance by estimating a parameter vector. Here, the input variables are the set of sentences where the outputs are the parse sequences. For each input instance, the algorithm finds the highest scoring candidate. If it is not the correct parse candidate it updates the parameter vector by taking the difference of the correct candidate with the highest scored one and increasing the parameter for the correct candidate. They have split the dataset as training, development, and test where the training set is used for parameter estimation and development test is used for feature extraction. In the testing phase, they have used the same corpus used in (Yuret and Türe, 2006) and they have achieved an accuracy value of 96.45% by using the same manually disambiguated test sets in (Hakkani-Tür et al., 2002; Yuret and Türe, 2006).

In Görgün and Yıldız (2011)'s study, morphological disambiguation problem is defined as a multi-class classification problem. It is aimed to detect the correct parse from N candidates of a word by ignoring the root words. Each distinct parse is a class. The input set is composed of the feature sets of the two previous neighbors and the correct class of the target word. Each feature set has 126 morphological features. The training set includes 1 million disambiguated tokens with 50,673 sentences. Experiments are conducted for many classification algorithms. They have achieved an accuracy value of 95.61% with the J48 tree classifier slightly better than the baseline trigram model (Hakkani-Tür et al., 2002).

As an example of neural models, Yıldız et al. (2016) proposed a general-purpose morphological disambiguation method and presented a Convolutional neural network (CNN) in deep learning architecture that can be used for morphologically rich languages. The neural network consists of two input layers and one output layer. The first input layer includes word representations as embeddings of roots and features. The second input layer incorporates the context knowledge obtained from an n-word window, into the first layer embeddings. The output layer, which is a softmax layer, calculates a classification score. In the training set, the correct parse sequences in a three-word sized window are labeled as positive whereas others are as negative. Stochastic gradient descent is used as the training algorithm and AdaGrad for optimization. Because the network learns as three-word windows, in inference time, all the calculations are made on the neural network for three-word sequences. Morphological disambiguation is achieved by using the Viterbi algorithm to select the best parse from the sentence sequences. They have used the same dataset with Yuret and Türe (2006) and manually tagged 20K of the tokens in the dataset to mitigate noise originating from semi-supervised data. In this study, 85.18% accuracy rate was achieved for Turkish which is higher than Sak et al. (2017) and Yuret and Türe (2006) as, 82.13% and 83.31%, respectively. The accuracy rates of these two studies appear to be low when compared to their own studies, although they have used the same training set. This is because they were tested with the test set prepared specifically for the study of Yıldız et al. (2016).

A recently introduced study (Dayanık et al., 2018), MorphNet, combines morphological analysis with a disambiguation model by using a sequence-to-sequence recurrent neural network. Long Short Term Memory (LSTM) encoders are used to encode three different embeddings. The first one (word encoder) produces character-based embeddings by modeling the root of the target word as character sequences and the tags as single units. The second encoder (word encoder) encodes the previous and next neighbors of the target word by using bi-directional LSTM to incorporate the context information to the model. The third LSTM encoder (output encoder) is used to produce the embeddings by using the analysis of the previous words, ignoring their roots. Decoder has two hidden LSTM layers where the first layer includes the context information and the second layer encodes the word and output embeddings. The decoder uses its hidden layer and three encoders to learn and predict the correct output for the target word. In the study, a new dataset named TrMor2018 is presented which is 97%+ accurate. They state that the previously presented datasets have low accuracy because of the noise. They have tested their morphological disambiguation model with TrMor2018 and with the previous datasets. The highest accuracy is acquired with the TrMor2018 dataset as 98.30% and the accuracy of the test with TrMor2006 was 96.86%.

Shen et al. (2016), have introduced a bi-LSTM model which produces two kinds of embeddings. The first embedding type is produced by using two different bi-LSTMs to separately embed the root and the morphemes. These two embeddings are concatenated with tanh function to produce the final embedding. Another embedding type is context embedding, it can be produced in two different methods. The first method (local) depends on the left and right neighbors of the target word and the second method (global) uses all words in the sequence to embed the context information. Finally, a softmax is used to combine the word embeddings and context embeddings. Test results for the local context model and global context model are 96.90% and 97.24%, respectively, which is slightly higher than the state-of-the-art (Sak et al., 2007, 96.80%).

2.2. Learning Vector Quantization (LVQ) and NLP

LVQ algorithm has been successfully applied in some NLP research fields like text classification (Umer and Khiyal, 2007; Visa et al., 2000; Martín-Valdivia et al., 2007; Pilevar et al., 2009), speech recognition (Haldar and Mishra, 2016), language identification (Gunawan et al., 2017), spam detection (Chuan et al., 2005) and multi-word expressions recognition (Diaz-Galiano et al., 2004).

Martín-Valdivia et al. (2007) have applied LVQ to two tasks as text categorization and word sense disambiguation (WSD). They have used REUTERS-21578 dataset for text categorization and SENSEVAL-3 corpus for WSD. The Kohonen layer they use doesn’t have hidden units but the network has one input and one output layer. When they modelled LVQ for text categorization, inputs are represented as terms in the documents and outputs of the network are categories. They have composed of different networks for each context. The model they have used for WSD has outputs as word senses. Umer and Khiyal (2007) use five different variants of LVQ to classify the texts and researched the best performing one. The tests between the LVQ variants are evaluated as 10-fold cross-validation. Each LVQ variant provide similar results and their training times are near. But the variant named OLVQ1 gives the best result. Performance of OLVQ1 is measured with different classification algorithms in 5-fold cross-validation and OLVQ1 outperforms the k-NN, C4 and Naive Bayes algorithms in both classification accuracy and running time. In (Pilevar et al., 2009), they have used LVQ for text classification purpose. They have used a text collection of 1050 story news as dataset and evaluated their tests for LVQ1, OLVQ1, LVQ 2.1, LVQ3 and OLVQ3 LVQ variants. OLVQ3 was compared with KNN and SVM algorithms and they have reported that, LVQ outperforms these two classification algorithms in both classification and time performance.

LVQ also has research in speech-recognition. In (Mantysalo et al., 1992) they have used LVQ method for Finnish speaker-dependent speech-recognition. In the study, they have aimed to measure the effect of high dimensions of context vectors which represent the phoneme set of Finnish. The context vectors were obtained by concatenating and averaging features in a time-domain. 99% of accuracy value was obtained when short-time feature vectors were used. Also they have realized that using high-dimensions as codebook vectors has positive effect on accuracy. Haldar and Mishra (2016) have used LVQ for multi-lingual speech recognition and language identification of English and Indian languages. They have acquired a recognition rate between 88% and 90%. Gunawan et al. (2017) presented a study for language identification of Arabic, Malay and Thai languages. LVQ was preferred for its low complexity and computational costs and trained with spectral frequencies. They have reported the recognition rate as 73.8%. Chuan et al. (2005) have used LVQ algorithm for identifying spam emails. In the study, LVQ algorithm provides a recall rate between 93.58% and 96.86% with 1500 times training. LVQ outperforms Bayes-based approach. Diaz-Galiano et al. (2004) used LVQ for multiword expression recognition and trained LVQ with CLEF 2001 database. They have acquired high precision values for information retrieval task.

3. MATERIALS

In this study, a morphological disambiguation framework is prepared which follows steps from data preparation to training and testing. This framework is a combination of several different tasks and materials. The block diagram of this structure is given in Figure 3.1.

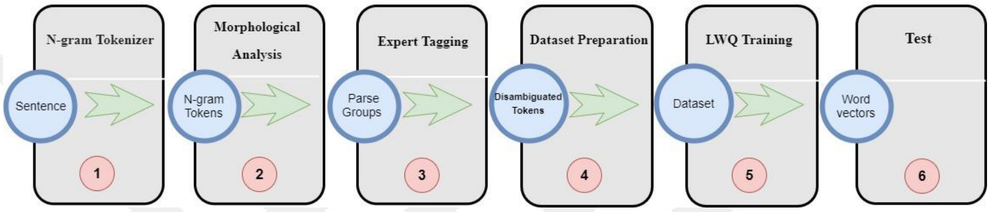


Figure 3.1. The block diagram of the system model of the study

The morphological analyzer (Yıldız et al. 2019) is the main tool used in the diagram shown in Figure 3.1. This analyzer uses a standard tokenizer which treats each word as a single token. However, this kind of analysis ignores the compound words or other multi-words in the sentences and this leads to more ambiguity by producing redundant parses. A better solution to this problem is to develop and integrate a "multi-word tokenizer". In this way, words in a sentence with spaces between them can be recognized as multi-word tokens. Moreover, the morphological analyzer dictionary (especially in terms of non-existing n-grams, compound words) is updated using the dictionary data of "Güncel Sözlük" published by the Turkish Language Association (TLA) to decrease out-of-vocabulary (OOV) rate.

Steps shown in Figure 3.1 begin with tokenization. We have developed a multi-word tokenizer and integrated it into a morphological analyzer. A sample sentence is given below:

"Öğrenci kalemi masasından alarak koşar adım tahtaya doğru yürüdü."

After the processing of the multi-word tokenizer the tokens in the sentence will be as:

"Öğrenci" "kalemi" "masasından" "alarak" "koşar adım" "tahtaya" "doğru" "yürüdü"

The tokens are analyzed in the morphological analyzer and parse outputs are grouped in the output as "lemma + POS". After this process the parse groups will be as:

"öğrenci+NOUN" ("kalem+NOUN","kale+NOUN") "masa+NOUN" "al+VERB" "koşar adım+ADV" "tahta+NOUN" "doğru+ADV" "yürü+VERB"

These parse groups are stored in a NoSQL database in JSON format, maintaining their relations in the sentences and texts. The main purpose of using a morphological analyzer in this framework is to define which word is ambiguous and identify its lemma candidates. The two steps are implemented as a batch job and it can be thought of as a pre-processing step before tagging. In the third step, these pre-analyzed sentences are introduced to an Expert with the help of a tagger tool (which is developed in this study). After Expert tagging the disambiguated sentence will be as:

"öğrenci+NOUN" "kalem+NOUN" "masa+NOUN" "al+VERB" "koşar adım+ADV" "tahta+NOUN" "doğru+ADV" "yürü+VERB"

Expert freely tags the tokens listed by the program. The expert can change the lemma of the token if it is incorrectly lemmatized by the morphological analyzer. When an ambiguous word token is considered, Expert lemmatizes it with the correct POS tag and lemma, in this way disambiguates. All the actions of the

Expert are updated on JSON structures retrieved from the database. In the fourth step, a special format of the dataset is prepared for each text, including a line for each ambiguous word. In the fifth step, in order to solve ambiguities, this new dataset is used in Learning Word-vector Quantization (LWQ) algorithm, to compose word-vectors and train. Finally, the test accuracy of the LWQ method is measured and interpreted by using a disambiguated dataset.

Details of the materials used in the framework are given in the following sections. LWQ is developed in the inspiration of a well-known algorithm LVQ. Because of that, details about the original LVQ algorithm and its variants in the literature are also given below.

3.1. LVQ (Learning Vector Quantization) Algorithm

LVQ algorithm is a reinforcement learning algorithm which is developed by Kohonen (Kohonen, 1984) to solve the classification problems. There are variations for the LVQ algorithm. These are LVQ1 (Kohonen, 1984), LVQ2 (Kohonen, 1990a; Kohonen 1990c), LVQ3 (Kohonen, 1990b), OLVQ1 (Kohonen, 1992), LVQ Algorithm with Penalization Mechanism (De Sieno, 1988) and LVQ-X (Öztemel, 1992) Algorithm.

3.1.1. LVQ1 Algorithm (Kohonen, 1984)

The main purpose of the LVQ algorithm is to represent a set of input vectors with a set of reference vectors. Learning process is used to decide which input vector belongs to which reference vector class. Output values are decided in a “winner takes all” strategy.

Learning process is implemented on an LVQ network with 3 layers. These are:

- **Input Layer:** The samples used for training exist in this layer. No processing occurs here.
- **Kohonen Layer:** This is the intermediate layer which keeps the processing elements. These elements are represented by reference vectors which consist of the weight values obtained by mapping of the input vectors in the Input Layer
- **Output Layer:** This layer is used to identify the class of the input.

The topology of the LVQ network is shown in Figure 3.2.

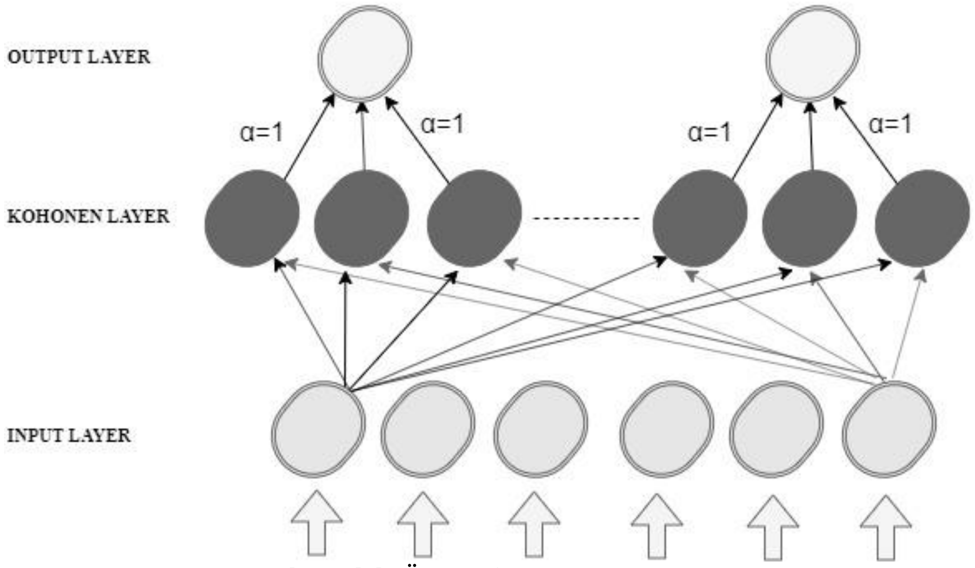


Figure 3.2. LVQ network model (Öztemel, 2012)

As seen in Figure 3.2, all processing elements of the input layer are connected to the processing elements of the Kohonen layer. On the other hand, some of the elements of the Kohonen layer are connected to a single element in the Output layer. Each element of the Kohonen layer belongs to one class. The weight values (α) between the Kohonen layer and the Output layer are binary values (0,1) and cannot be changed. In each iteration of the training processing, elements in the

Kohonen layer race each other. The procedure for the LVQ network is as (Öztemel, 1992):

1. Define the samples
2. Define the network topology
3. Define the network parameters, i.e. learning rate
4. Define the initial weight values
5. Take a sample from the input set and direct to the network
6. Find the winning processing element
7. Change the weight values
8. Repeat steps (5-7) until all the samples are classified correctly

The performance of an LVQ network is dependent on the count of the reference vectors, starting values and the adjustment of the learning rate. These are all defined by experience and there are no rules. Learning rate should descend till zero. Learning rule of the LVQ network is called Kohonen learning rule. Basically, it depends on the racing of the elements in the Kohonen layer. The race starts with the calculation of the weight values between the input elements and racing elements by using Euclidian distance as in (3.1).

$$dist_i = |V_i - X| = \sqrt{\sum_j \left(V_{ij} - x_j\right)^2} \tag{3.1}$$

Where *Vij* and *x_j* represent the weight vector of the reference vector and the input vector for the value *j* respectively. This distance value (dist) is calculated for all the reference vectors. The lowest value for *dist* represents the nearest reference vector to the input vector. This reference vector for the processing element is the winner and only its weight values are recalculated in the recent iteration. The recalculation rule is as follows:

- Winner processing elements belong to the correct class. When this happens the weight values are approximated to the input vector as in the following formula 3.2.

$$V_y = V_e + \lambda(X - V_e) \qquad (3.2)$$

- Winner processing elements belong to the incorrect class. When this happens the weight values are approximated to the input vector as in Equation 3.3.

$$V_y = V_e - \lambda(X - V_e) \qquad (3.3)$$

For both formulas in Eq. (3.2) and Eq. (3.3) λ is the learning rate, V_e is the current value of the reference vector, V_y is the next value of the reference vector and X is the input vector. As seen in Fig 3.3 nearest reference vector to the input vector X is V_1.

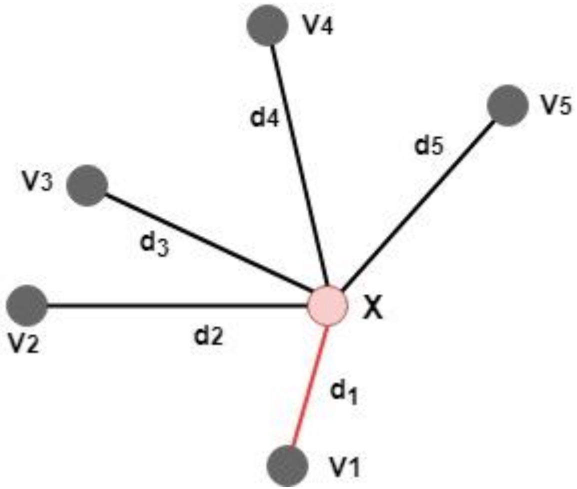


Figure 3.3. Nearest vector to the input vector (Öztemel, 2012)

Fig 3.4 shows the steps for the approximation of vectors *X* and V*1*. As the learning rate decreases by the following iterations, it converges to 0. This situation is represented in Fig. 3.4.

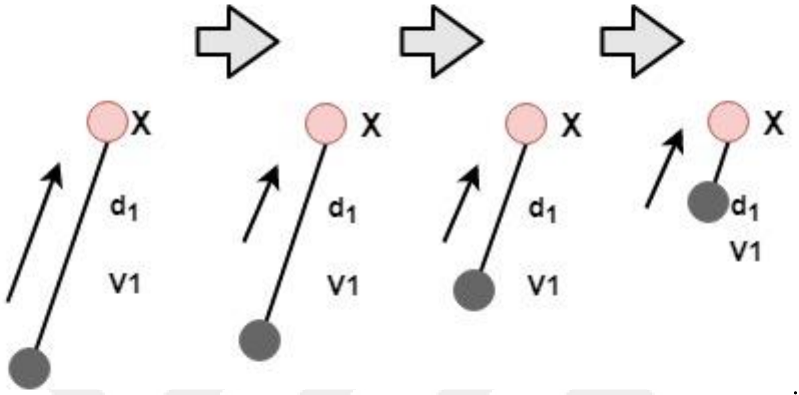


Figure 3.4. Approximation of the input vector to the reference vector (Öztemel, 2012)

When this parameter gets the 0 value, the network overtrains and forgets what it learns. Also sometimes the same reference vectors over wins. There are also problems with boundary values. These disadvantages for the standard LVQ algorithm (LVQ1) can be overcome by different adaptations of the algorithm as described in the following section.

3.1.2. LVQ2 Algorithm (Kohonen, 1990a; Kohonen 1990c)

LVQ2 algorithm was developed to prevent the incorrect classification which occurs in the boundaries of the classes. The idea is to update the weight values of the two vectors which are between the input vectors. There are two conditions to be met. When we call V_1 and V_2 as the nearest vectors to the input vector *X;*

1. V_1 is the nearest vector V_2 is the second nearest vector and V_1 belongs to the incorrect class and V_2 belongs to the correct class.

2. *X* input vector resides between V_1 and V_2 in a window *w*

When these two conditions are met, the new values of V_1 and V_2 are V_{1y} and V_{2y} respectively. The formulations for these calculations are shown in formulas 3.4 and 3.5.

$$V_{1y} = V_{1e} - \lambda(X - V_{1e}) \tag{3.4}$$

$$V_{2y} = V_{2e} + \lambda(X - V_{2e}) \tag{3.5}$$

where λ is the learning rate, V_{1e} and V_{2e} are the current values of the reference vectors. To guarantee the X input vector to be between the reference vectors, the condition in Formula 3.6 should be met.

$$Min\left(\frac{d_1}{d_2}, \frac{d_2}{d_1}\right) > s \text{ where } s = \frac{(1-w)}{(1+w)} \tag{3.6}$$

where d_1 is the distance of the vector V_1 and d_2 is the distance of the vector V_2 to the input vector *X* and *w* is the window length. An example of LVQ2, which shows the behavior of the reference vectors is shown in Fig 3.5.

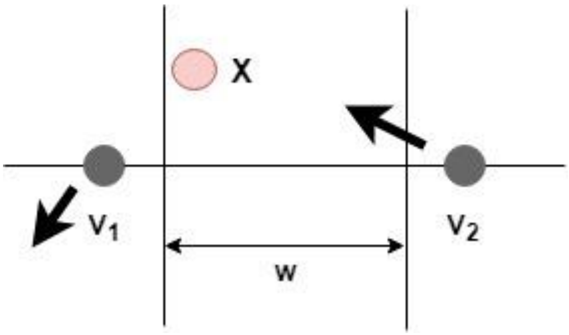


Figure 3.5. Schematic view of LVQ2 algorithm (Öztemel, 2012)

In Fig 3.5, it can be seen that vector *V₁* is going further and vector *V₂* is getting closer to input vector *X*.

3.1.3. LVQ3 Algorithm (Kohonen, 1990b)

In the LVQ2 algorithm, vector V_1 differentially changes in an uncontrolled way. To keep this vector approximate to the correct class distribution, the LVQ3 algorithm is proposed. Additional to the conditions in LVQ2, when the closest reference vectors are in the same class, the formula for the calculation should be as in Equation 3.7.

$$V_{ky} = V_{ke} + \varepsilon(X - V_{ke})\ for\ k \in \ \{1,2\} \tag{3.7}$$

where ε parameter is constant and a value between 0.1 and 0.5 is proposed. This parameter gets smaller when the window gets narrower. LVQ provides the optimal place of the vector V_1.

3.1.4. LVQ Algorithm with Penalization Mechanism (De Sieno, 1988)

In the standard LVQ algorithm, some of the weight vectors win frequently and this leads to an unbalanced network. When this happens, some of the weight vectors are unable to be a reference vector for the input. Penalization mechanism proposed by De Sieno (1988) solves this problem by penalizing the frequent winner vector and prevents it to win recurrently. This mechanism runs by adding a *b* value to the distance (*d*) of this vector with the input vector. *b* value is determined by how many times the winner wins. The amount to be added to the *d* for the i^{th} processing unit is determined by the Equation in 3.8.

$$b_i = C(p_i + \frac{1}{N}) \tag{3.8}$$

$$p^{y}{}_{i} = p^{e}{}_{i} + B(y_i - p^{e}{}_{i}) \tag{3.9}$$

where, in Equation 3.8, C is a constant determined by the designer. N represents the processing element count in the Kohonen layer. And p_i is the possibility of the i^{th} element to win the race. The initial value of p_i is $1/N$ and it is updated according to the rule in 3.9.

In Equation 3.9, p^y is the new possibility, p^e is the recent possibility value in order to win the race. B is a constant value and y_i is the output value and calculated as in 3.10.

$$y_i = \begin{cases} 1 & If\ process\ element\ i\ wins \\ 0 & otherwise \end{cases} \tag{3.10}$$

In each iteration, b value is calculated for each processing element and added to the distance between this element and input reference vector as in 3.11.

$$d^{y}{}_{i} = d^{e}{}_{i} + b_i \tag{3.11}$$

where d^e is the current distance, d^y is the new value of the weight vector calculated according to the winning possibility. Race within the Kohonen layer occurs with the new distance values. Thus frequent winner is penalized and the other elements have the chance to win.

3.1.5. LVQ-X Algorithm (Öztemel, 1992)

LVQ-X algorithm is another variant of the LVQ algorithm where it changes the value of the two weight vectors in each iteration. LVQ2 also changes two boundary vectors but this happens seldom. This algorithm is developed by Öztemel (1992) and provides shorter learning time and increases generality. This is achieved by determining two winning vectors in each iteration. These are:

- Global winner: Represents the nearest process element to the input vector.
- Local winner: Represents the process element which belongs to the correct class and nearest to the input vector.

$$V_y = V_e - \lambda(X - V_e) \tag{3.12}$$

$$V_y = V_e + \lambda(X - V_e) \tag{3.13}$$

When the global winner vector is incorrect class it gets further from the input vector as in 3.12. At the same time, the local winner vector gets closer to the input vector as in 3.13.

3.2. Dataset

In this section, brief information is given about the datasets used for the morphological disambiguation of Turkish. Also, detailed information is presented about the dataset prepared for this study. At the end of the section, details are given about the materials which are developed and used for the preparation of the dataset. Before going into the subsections, a summary of the training datasets used in morphological disambiguation of Turkish are given in Table 3.1

Table 3.1. Training datasets used in morphological disambiguation of Turkish

Study	Training Dataset	Dataset Details
Yuret and Türe (2006)	TrMor2006 Yuret and Türe (2006)	1 million words from semi-automatically disambiguated Turkish news text
Sak et al. (2007)		
Görgün and Yıldız (2011)		
Yildiz et al. (2016)		
Shen et al. (2016)		
Dayanık et al. (2018)	TrMor2018 Dayanık et al. (2018)	460,663 words from semi-automatically disambiguated Turkish news text

TrMor2006 dataset is extracted from 2,386 documents which include 50,716 sentences. On the other hand, TrMor2018 dataset is extracted from 390 documents which consist of 34,673 sentences. In our knowledge, most of the studies have used the TrMor2006 dataset and most of the time both as training and test datasets. Recently, TrMor2018 is presented and used in the same study (Dayanık et al., 2018) for morphological disambiguation.

3.2.1. Statistics of the Datasets for Morphological Disambiguation of Turkish

In the literature, various datasets for morphological disambiguation of the Turkish language are presented. The well-known datasets are TrMor2006 (Yuret and Türe, 2006), TrMor2016 (Yildiz et al. 2016) and TrMor2018 (Dayanık et al., 2018) are produced and introduced as semi-supervised. TrMor2016 has the same train set as TrMor2006. But it includes a different test set. Statistics related to these datasets are given in Table 3.2.

Table 3.2. Datasets prepared for morphological disambiguation of Turkish

(# of tokens)	**Ambiguous**		**Unambiguous**		**Total**	
	Train	**Test**	**Train**	**Test**	**Train**	**Test**
TrMor2006	398,290	379	439,234	483	837,524	862
TrMor2016	398,290	9,460	439,234	9,802	837,524	19,262
TrMor2018	215,024	21,477	243,866	25,166	458,890	46,643

In Table 3.2, disambiguated Turkish datasets given in Table 3.1 are given in more detail with ambiguity situation and the amounts used for training or test. There is no data consistency of information regarding the datasets listed in Table 3.2. For the recently presented TrMor2018 (Dayanık et al., 2018) dataset, 2,090 sentences and 28,909 words are selected from the previous version (TrMor2006). The data noise level was then measured and reported as 3%. This information contributes to measurability in the studies to be carried out by using this dataset.

Since TrMor2018 has not been presented during our study, and since the previous dataset (Yuret and Türe, 2006) was prepared in a semi-supervised way with reliability concerns, we have decided to create our dataset to obtain more reliable results. Expert tagging on Turkish texts is performed to guarantee data consistency in the production of this dataset. Details about the statistics and preparation steps are given in the following sections.

3.2.2. Statistics of the Dataset Prepared for This Study

The statistics related to the dataset prepared and used in this study are shown in Table 3.3.

Table 3.3. Statistics of the supervised dataset prepared for this study (# of tokens)

(Train)	**Ambiguous**	**Unambiguous**	**Total**
Dataset	14,806	21,721	36,527

As seen in Table 3.3, the disambiguated dataset prepared for this study includes 36,527 tokens and the token ambiguity rate is 40.5%. We have manually collected Turkish news and novel texts from the Web, in order to ensure that the dataset has a balance in context. News texts refer to the information on a certain event in the related day. 216 news texts and 3 novel texts are collected. The total number of sentences for the whole dataset is 5,723. 3,149 of them are news texts and the remaining part is from novels.

3.2.3. Dataset Preparation

Dataset preparation steps include the text pre-processing and morphological analysis steps. All texts are pre-processed and cleaned as described below:

i. Split texts to sentences with Turkish NLTK Punkt (Kiss and Strunk, 2006) function.
ii. Remove sentences with less than 20 characters (for example dialogues in novel texts)
iii. Remove punctuation marks and numbers.
iv. Trim white spaces between the words in the sentences and at the start and end of the sentence.
v. Insert all sentences into a MongoDB NoSQL database with their corresponding texts.

The next step in the data preparation phase is, to use the morphological analyzer to parse pre-processed sentences for determining the ambiguity status of a word. The morphological analyzer (Yıldız et al., 2019) processes all sentences in the MongoDB database and saves the outputs in the same database collections. This is a pre-condition to be completed before Expert tagging. The parsing steps are described below:

i. Select a sentence from the MongoDB database and tokenize it with the multi-word tokenizer in morphological analyzer.
ii. Process each token with the morphological analyzer and define its status (ambiguous, unambiguous or OOV). If the word is ambiguous save its correct lemma as “dummy” and save its candidates in the database. If the word is unambiguous save its lemma and POS tag (All data structures are in JSON format).
iii. Go to Step i

The morphological analyzer output is in inflectional group (IG) structure, including the morphological tags and standard POS labels used for Turkish. In the scope of this study, the morphological analyzer output is configured to group parse lists as “lemma + POS”. There are three kinds of parse outputs provided by the morphological analyzer. If there is only one distinct parse output (“lemma + POS”) this indicates there is no ambiguity for the corresponding token. If there are more than one parse outputs, there is ambiguity for the token. In this study, for the first case, the “unambiguous word” term and for the second case “ambiguous word” term will be used respectively. In case, no parse occurs, the morphological analyzer is unable to find a lemma for the token (out-of-vocabulary) and this is out of the scope of the study. In this case, the relevant token is ignored.

A JSON sample which represents a sentence processed by the morphological analyzer is given below:

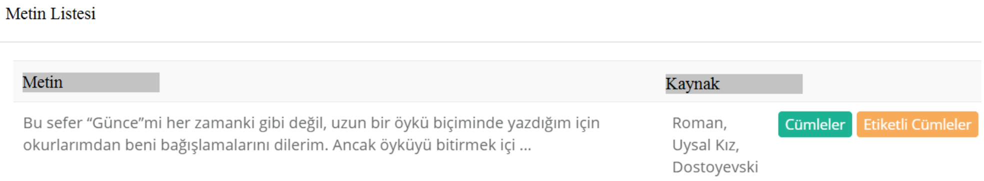
```
{
    "_id" : ObjectId("5cb2422a29101b3c3468a589"),
    "Cumle" : "bir grup milletvekilinin İngilterenin ABden
ayrılmasını önlemeye dönük yasa bir oy farkla onaylanmıştı",
    "IsTagged" : true,
    "Text_ID" : ObjectId("5cb23d87f30c8c3d380f60b4"),
    "Tokens" : [
        {
            "UserId" : ObjectId("5c028e2c9ccc334b60414db0"),
            "TaggedTokens" : [
                {
                    "token" : "bir",
                    "lemma" : "bir",
                    "tag" : "ADJ",
                    "flag" : "1#bir+DET#bir+NOUN"
                },
                {
                    "token" : "grup",
                    "lemma" : "grup",
                    "tag" : "NOUN",
                    "flag" : "0#grup+NOUN"
                },
                {
                    "token" : "milletvekilinin",
                    "lemma" : "milletvekili",
                    "tag" : "NOUN",
                    "flag" : "0#milletvekil+NOUN"
                },
{
    "token" : "ingilterenin",
    "lemma" : "ingiltere",
    "tag" : "NOUN",
    "flag" : "0#ingiltere+NOUN"
},
{
    "token" : "abden",
    "lemma" : "ab",
    "tag" : "NOUN",
    "flag" : "0#ab+NOUN"
},

{
    "token" : "ayrılmasını",
    "lemma" : "ayrıl",
    "tag" : "VERB",
    "flag" : "1#ayrıl+VERB#ayrılma+NOUN#ayır+VERB"
},
{
    "token" : "önlemeye",
    "lemma" : "önle",
    "tag" : "VERB",
    "flag" : "0#önle+VERB"
},
{
    "token" : "dönük",
    "lemma" : "dönük",
    "tag" : "ADJ",
    "flag" : "0#dönük+NOUN"
},
{
    "token" : "yasa",
    "lemma" : "yasa",
    "tag" : "NOUN",
    "flag" : "0#yasa+NOUN"
},
{
    "token" : "bir",
    "lemma" : "bir",
    "tag" : "PUNC",
    "flag" : "1#bir+DET#bir+NOUN"
},
{
    "token" : "oy",
    "lemma" : "oy",
                "tag" : "NOUN",
                "flag" : "1#oy+NOUN#oy+VERB"
            },
            {
                "token" : "farkla",
                "lemma" : "fark",
                "tag" : "NOUN",
                "flag" : "0#fark+NOUN"
            },
            {
                "token" : "onaylanmıştı",
                "lemma" : "onayla",
                "tag" : "VERB",
                "flag" : "1#onay+NOUN#onayla+VERB"
            }
        ]
    }
      ]
```

This JSON tagged sentence example is taken from the MongoDB database. The "token" field is the output of the tokenizer which is a function of the morphological analyzer. The "lemma", "tag" and "flag" fields include the parts of the parse lists. When the "flag" field starts with "0", this indicates there is only one lemma candidate for the token and token is unambiguous. If the "flag" field starts with "1", the following part of the field includes the lemma candidates of an ambiguous word. Sometimes the "flag" field starts with "2". This represents the case where the token is unknown due to being an OOV. It is impossible for the morphological analyzer to select the correct candidate for ambiguous words from the parse options. Because of this, in the beginning, "dummy" is written for the "lemma" and "tag" fields. During tagging, these fields are filled by the Expert. For example, the last token of the sentence is "onaylanmıştı". Morphological analyzer parse produces two lemma candidates as "onay + NOUN" and "onayla + VERB". The initial values of "lemma" and "tag" fields are "dummy". After tagging, these fields are filled by the correct candidate as "onayla" and "VERB" as seen in the JSON example.

3.2.4. Tagging Application

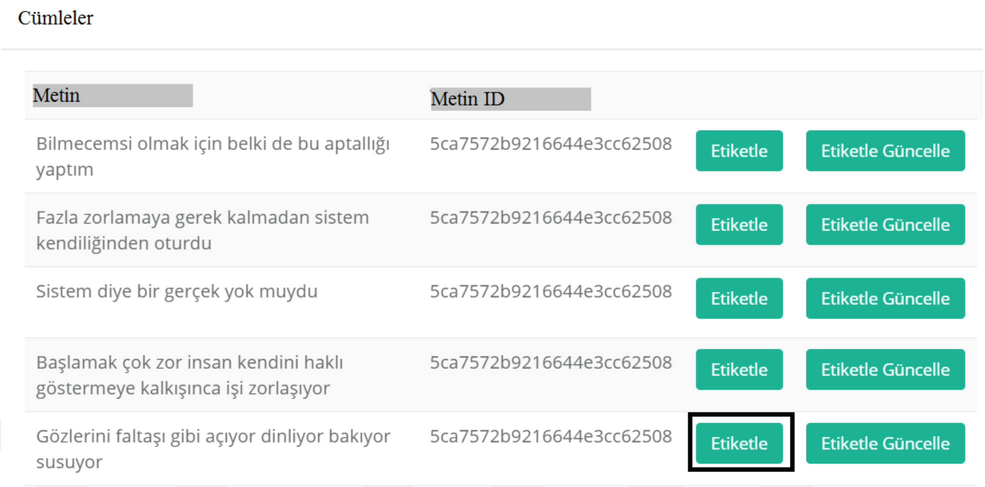
The main purpose of the expert tagging process is to select the correct lemmas of ambiguous words with human knowledge. Analyzing the condition of the ambiguous word in the sentence helps in selecting the correct lemma from the candidates. In the beginning, the data preparation phase should be completed in order to start the tagging process. The tagging process is achieved by using a tagger tool to label each ambiguous token. For this purpose, we have prepared a web-based tagger tool in .NET C#. The tagging tool presents the texts and their corresponding sentences in the MongoDB database to the Expert. The Expert selects the correct lemma candidates and POS tags of the ambiguous words and these selections are saved to the database. An example screen of the tagger tool is shown in Figure 3.6.


Metin Listesi

Metin	Kaynak
Bu sefer “Günce”mi her zamanki gibi değil, uzun bir öykü biçiminde yazdığım için okurlarımdan beni bağışlamalarını dilerim. Ancak öyküyü bitirmek içi ...	Roman, Uysal Kız, Dostoyevski

Figure 3.6. “Texts” screen of the tagger tool

Figure 3.6 shows a screen for the texts stored in the MongoDB database. “Cümleler” button lists all the corresponding sentences of the text. “Etiketli Cümleler” lists the sentences of this text which are tagged by the Expert. Figure 3.7 shows the Sentences screen when the “Cümleler” button is clicked.

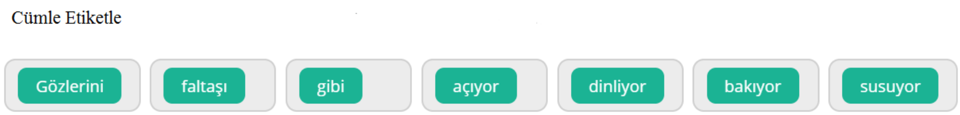


Cümleler

Metin	Metin ID		
Bilmecemsi olmak için belki de bu aptallığı yaptım	5ca7572b9216644e3cc62508	Etiketle	Etiketle Güncelle
Fazla zorlamaya gerek kalmadan sistem kendiliğinden oturdu	5ca7572b9216644e3cc62508	Etiketle	Etiketle Güncelle
Sistem diye bir gerçek yok muydu	5ca7572b9216644e3cc62508	Etiketle	Etiketle Güncelle
Başlamak çok zor insan kendini haklı göstermeye kalkışınca işi zorlaşıyor	5ca7572b9216644e3cc62508	Etiketle	Etiketle Güncelle
Gözlerini faltaşı gibi açıyor dinliyor bakıyor susuyor	5ca7572b9216644e3cc62508	Etiketle	Etiketle Güncelle

Figure 3.7. “Sentences” screen of the tagger tool

In Figure 3.7, tagging starts by clicking the “Etiketle” button. By clicking the “Etiketle Güncelle” button Expert can update the lemma and tag information of the tokens in a pre-tagged sentence. Figure 3.8 shows “tokens” view before tagging.



Cümle Etiketle

Figure 3.8. “Tokens” screen of the tagger tool

The database stores the tokens as single units provided by the morphological analyzer. In some cases, the lemma of a compound word may not available in the dictionary, so tokens are shown as separate tokens on the screen. For example, the “faltaşı”, “gibi” and “aç” tokens are a part of a phrase and seen in Figure 3.8 will be shown to Expert as a single unit if it exists in the dictionary of the morphological analyzer as “faltaşı gibi aç” which is the lemma of the inflected word “faltaşı gibi açıyor”. However, since the compound verb “faltaşı gibi açmak” is not present in the dictionary, the Expert is expected to combine these tokens on the screen. The merging of the units by the expert is shown in Figure 3.9.

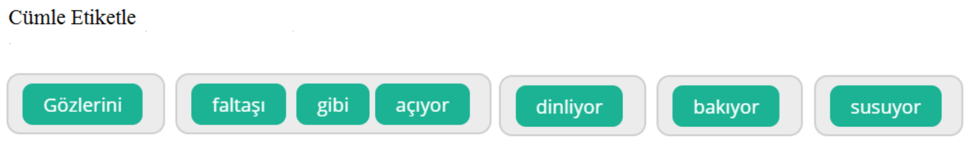
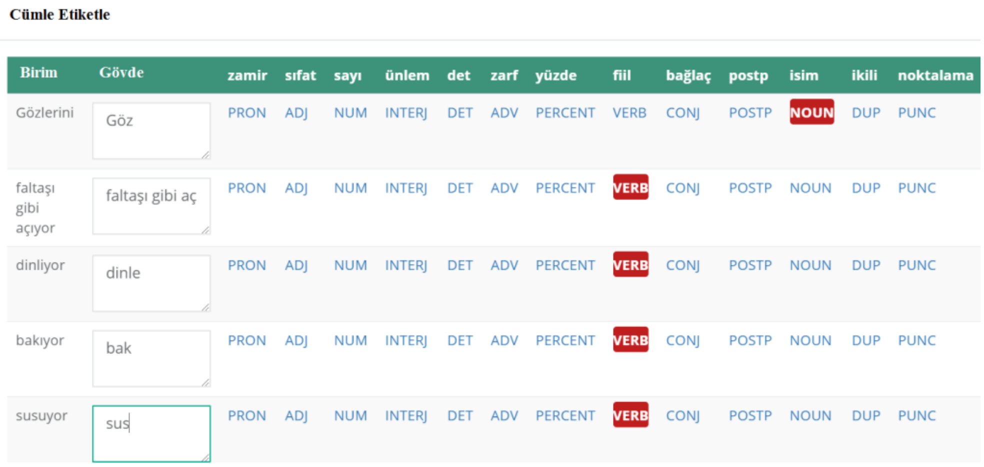


Figure 3.9. Merging tokens in the tagger tool

The tokens shown in Figure 3.8 can be combined any time to form new compound words. These compound words can be added to the morphological analyzer dictionary after Expert control. Thus, the dictionary can be enriched. Following the tokenization step, lemmatization and POS tagging begin as shown in Figure 3.10.



Cümle Etiketle

Birim	Gövde	zamir	sıfat	sayı	ünlem	det	zarf	yüzde	fiil	bağlaç	postp	isim	ikili	noktalama
Gözlerini	Göz	PRON	ADJ	NUM	INTERJ	DET	ADV	PERCENT	VERB	CONJ	POSTP	NOUN	DUP	PUNC
faltaşı gibi açıyor	faltaşı gibi aç	PRON	ADJ	NUM	INTERJ	DET	ADV	PERCENT	VERB	CONJ	POSTP	NOUN	DUP	PUNC
dinliyor	dinle	PRON	ADJ	NUM	INTERJ	DET	ADV	PERCENT	VERB	CONJ	POSTP	NOUN	DUP	PUNC
bakıyor	bak	PRON	ADJ	NUM	INTERJ	DET	ADV	PERCENT	VERB	CONJ	POSTP	NOUN	DUP	PUNC
susuyor	sus	PRON	ADJ	NUM	INTERJ	DET	ADV	PERCENT	VERB	CONJ	POSTP	NOUN	DUP	PUNC

Figure 3.10. Tagger tool lemmatization and POS tagging screen

In Figure 3.10, the lemma and POS tag are selected by the Expert. The Expert does not know about the lemma candidates of a word stored in the database. The POS tag types listed here are exactly the same as the POS tag types used by the morphological analyzer to provide consistency.

3.3. Morphological Analyzer (Yıldız et al., 2019)

A morphological analyzer, which was recently presented by Yıldız et al. (2019), is used in this study. When the codes of morphological analyzer were examined in detail, it was observed that the Finite State Machines (FSM) presented for Turkish by Oflazer (1994) was modified and used in the morphological analyzer. Additional gates are added to the verbal FSMs, specifically to match certain flags in the dictionary. Also, it can be seen that a detailed dictionary has been prepared for the morphological analyzer Yıldız et al. (2019). Accuracy tests are performed for morphological analyzer as its original version and the results are presented in Table 3.4.

Table 3.4. Accuracy test results of the 'Morphological Analyzer'

Word count	28,863
Correctly lemmatized with the exact result	22,640
Incorrect lemmatization	95
Ambiguous words	6,128
Lemmatization Accuracy	99.6% *
*When ambiguity is ignored	

In Table 3.4 we provide the accuracy results of the morphological analyzer tested with a supervised dataset. The words used for lemmatization are provided from a dataset presented by Tahiroğlu B.T. (2014). This dataset is in <word, lemma> format. where "word" is any word and "lemma" denotes its lemma defined by the Expert who prepares the dataset. In Table 3.4, "Correctly lemmatized with exact result" shows the results when the "word" in <word, lemma> is given as input and the output of the morphological analyzer exactly matches with the "lemma" in the tuple. Ambiguity is ignored with removing the

6,128 words from the 28,863 total. In this case, 22,640 unambiguous words are correctly lemmatized and lemmatization accuracy was 99.6%.

The running time performance of the morphological analyzer is also measured and the results are given in Table 3.5 as its original version. The dataset used for the test is acquired from (Sezer T, 2017) which consists of Turkish news data.

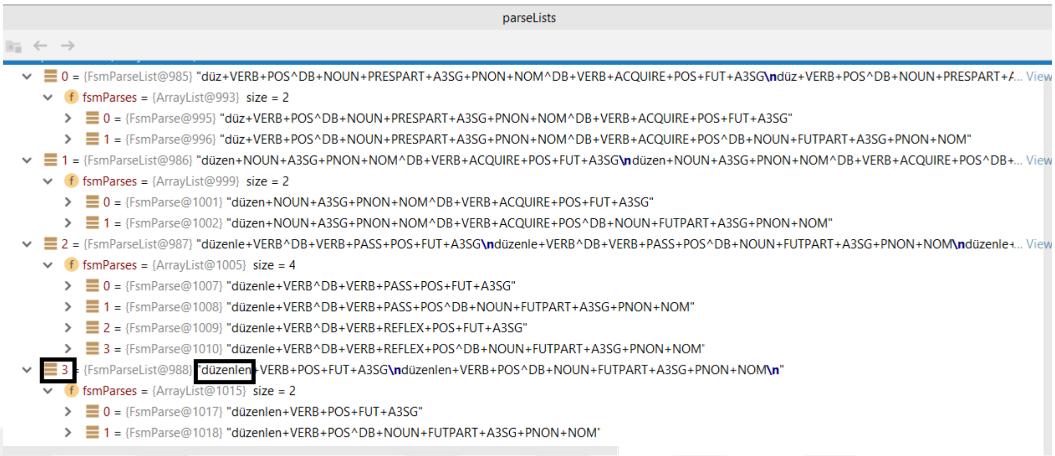
Table 3.5. Time performance of the 'Morphological Analyzer'

Token count	22,785,894
Sentence count	1,552,495
Total parse time	125 seconds

As seen in Table 3.5, the morphological analyzer can parse nearly 22 million tokens approximately in 2 minutes.

We have realized that compound words are missed in the dictionary used by the morphological analyzer. Because of that, units (words) of a compound word are treated as separate tokens in tokenization and this contributes to overall ambiguity. In order to prevent this kind of ambiguity, 29,829 compound words in the TLA dictionary are added to the morphological analyzer dictionary and analyzer is used in this way.

The FSMs embedded in the morphological analyzer include derivational affixes with the inflectional affixes in their designs. Like many other morphological analyzers, while composing the parse outputs, they use both types of affixes together. Because of that, more parse candidates are produced by including the root forms of the lemmas in the same parse. This is sometimes one of the causes of ambiguity originated by the morphological analyzer. An example of this case is given in Figure 3.11.



```
0 = {FsmParseList@985} "düz+VERB+POS^DB+NOUN+PRESPART+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS+FUT+A3SG\ndüz+VERB+POS^DB+NOUN+PRESPART+A... View
    fsmParses = {ArrayList@993} size = 2
        0 = {FsmParse@995} "düz+VERB+POS^DB+NOUN+PRESPART+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS+FUT+A3SG"
        1 = {FsmParse@996} "düz+VERB+POS^DB+NOUN+PRESPART+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM"
1 = {FsmParseList@986} "düzen+NOUN+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS+FUT+A3SG\ndüzen+NOUN+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS^DB+... View
    fsmParses = {ArrayList@999} size = 2
        0 = {FsmParse@1001} "düzen+NOUN+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS+FUT+A3SG"
        1 = {FsmParse@1002} "düzen+NOUN+A3SG+PNON+NOM^DB+VERB+ACQUIRE+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM"
2 = {FsmParseList@987} "düzenle+VERB^DB+VERB+PASS+POS+FUT+A3SG\ndüzenle+VERB^DB+VERB+PASS+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM\ndüzenle+... View
    fsmParses = {ArrayList@1005} size = 4
        0 = {FsmParse@1007} "düzenle+VERB^DB+VERB+PASS+POS+FUT+A3SG"
        1 = {FsmParse@1008} "düzenle+VERB^DB+VERB+PASS+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM"
        2 = {FsmParse@1009} "düzenle+VERB^DB+VERB+REFLEX+POS+FUT+A3SG"
        3 = {FsmParse@1010} "düzenle+VERB^DB+VERB+REFLEX+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM'
3 = {FsmParseList@988} "düzenlen+VERB+POS+FUT+A3SG\ndüzenlen+VERB+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM\n"
    fsmParses = {ArrayList@1015} size = 2
        0 = {FsmParse@1017} "düzenlen+VERB+POS+FUT+A3SG"
        1 = {FsmParse@1018} "düzenlen+VERB+POS^DB+NOUN+FUTPART+A3SG+PNON+NOM'
```

Figure 3.11. Parse output for the word "düzenlenecek"

Figure 3.11 shows the analysis made by morphological analyzer for the word “düzenlenecek”. As a result of the fact that FSMs processed the “-n” derivational suffix with passivity there can be seen many lemma root words like “düzenle”, “düzen” and “düz” in the parse outputs; which causes ambiguity. The parse output should be one instead of many and should contain the correct lemma as “düzenlen” with the corresponding POS tag (VERB).

4. METHODS

In this section, we give the details of the proposed classification algorithm, Learning Word-vector quantization (LWQ), which is an adaptation of the well-known classification algorithm Learning Vector Quantization (LVQ). Also, the multi-word tokenizer we that we have developed is explained with an example. Finally, our approach to the identification of OOV words in texts is given.

4.1. Proposed Morphological Disambiguation Method – LWQ

We have developed LWQ, in inspiration of the LVQ method, to solve the morphological disambiguation problem. The relations of word-vectors that belong to different classes are modeled by adjusting their positions in a vector space with a reward-or-penalize strategy.

The data model is prepared by dividing a sentence or text into segments by applying a window of width *w* to an ambiguous word. The idea is, to collect *w/2* unambiguous neighboring words from both the right and left sides of the target word (ambiguous word). Lemmas of each unambiguous word that enter the window compose “motion pairs” with the lemma candidates of the ambiguous word. Figure 4.1 illustrates how a window with a size of 4 is applied to an example sentence.

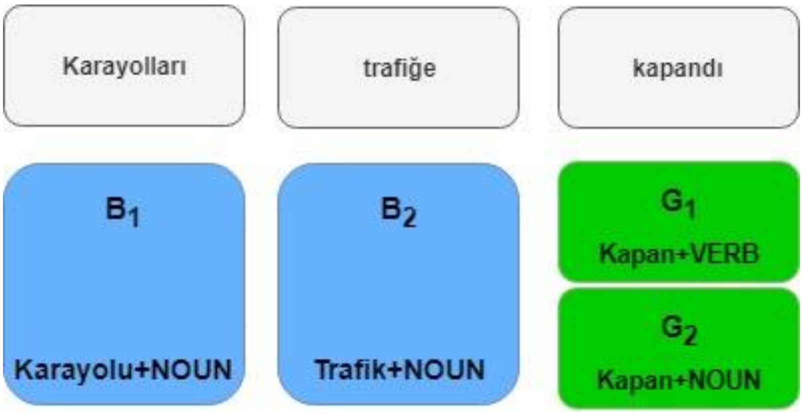


Figure 4.1. A sample sentence for a window-size=4

In Figure 4.1, B_1 and B_2 represent unambiguous words within the window, and G_1 and G_2 represent the lemma candidates of the ambiguous word. The application of a window in the study is applied by selecting an equal number of unambiguous words from the right and left of the target ambiguous word. Since each text in the corpus is evaluated separately (not all sentences of the texts are consecutively linked), the window implementation in the last sentence of each text is performed as in Figure 4.1.

Once the dataset is prepared, two distinct vocabularies for the ambiguous and unambiguous word lemmas are created. Word-elements in these vocabularies are randomly located in a vector-space. Vector size is defined by the user. After that, the training steps begin with a distance-based classification test. This test is performed separately for each row in the dataset. In the dataset row, the ambiguous word's lemma candidate (G_i), whose sum of the distances to w unambiguous words (B_j) is closest, wins the test. Mathematically, $\underset{1 \le i \le m}{\operatorname{argmin}}(\sum_{j=1}^{w} \|G_i - B_j\|)$ calculates all the candidates' distances for each ambiguous word. If the winner is the correct candidate, training passes to the next row of the dataset without updating. Otherwise, the candidate who is required to win is subject to approximation and other candidates are subject to departure in vector space. All the possible relationships that can be established for the example sentence given above can be modeled on a complete bipartite graph when unambiguous word lemmas are represented as one set and ambiguous word lemmas as the other set. The elements of each set only establish relations with the elements of the other group as (B_1, G_1), (B_1, G_2), (B_2, G_1), (B_2, G_2), in the form of "motion pairs". The distance-based classification tests for each ambiguous candidate are shown as "motion pairs" as in Figure 4.2.

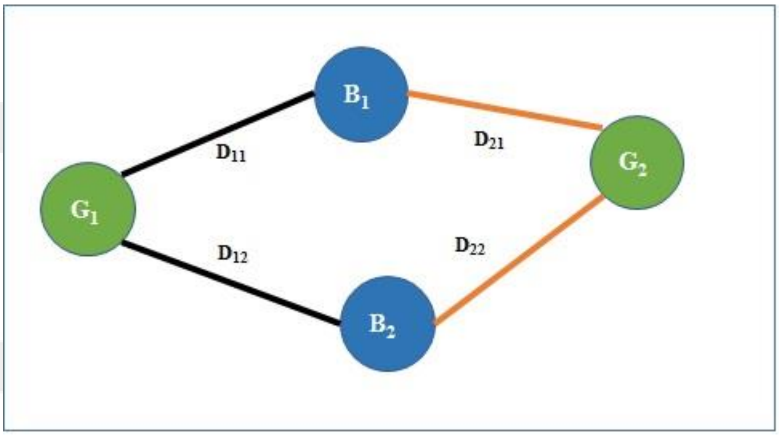


Figure 4.2. Distance-based classification test for a sample sentence

In Figure 4.2, blue circles represent the unambiguous (B_1 and B_2) and green circles represent the ambiguous lemma candidates of a word as (G_1 and G_2). According to the distance-based classification test, the candidate G_1 is more distant than the G_2 candidate ($D_{11}+D_{12}>D_{21}+D_{22}$). Therefore, G_2 (kapan+NOUN) is chosen as the solution to the ambiguity. However, according to the example sentence, for the ambiguous word "kapandı", the correct lemma candidate must be G_1 (kapan + VERB). Reward-or-Penalize step is applied to related word-vectors to solve this error. Accordingly, the LWQ algorithm tends to move each word-vectors of the (B_1, G_1) ve (B_2, G_1) pairs closer and (B_1, G_2) ve (B_2, G_2) pairs further. In terms of ease of representation, in the study, approximating "motion pairs" and departing “motion pairs” are shown as (B_1, G_1, +1), (B_1, G_2, -1) respectively. Figure 4.3 shows the reward mechanism for the sample sentence.

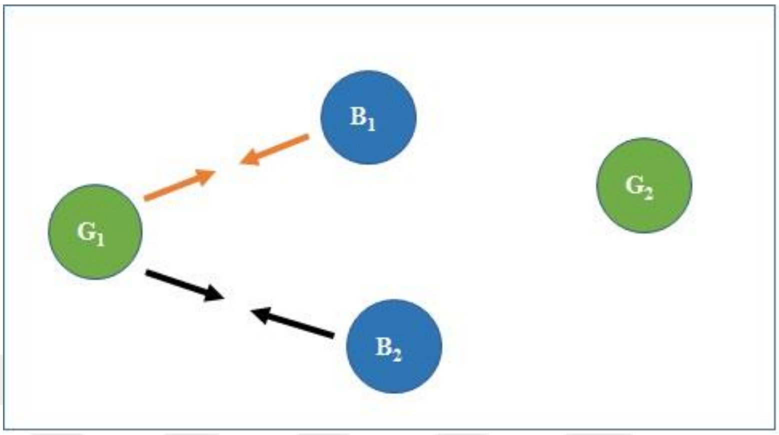


Figure 4.3. Rewarding steps in training for a sample sentence

The rewarding phase shown in Figure 4.3 depicts the approximation of the correct lemma candidate (G_1) (selected by the Expert) to the unambiguous lemmas (B_1 and B_2). This amount of approximation is determined by the learning ratio (η) and gradually reduced until convergence occurs. The mathematical representation of this rewarding movement is represented as (4.1).

$$
\begin{aligned}
B_1(t+1) &= B_1(t) - \eta\big(B_1(t) - G_1(t)\big) \\
G_1(t+1) &= G_1(t) + \eta(B_1(t) - G_1(t))
\end{aligned}
\tag{4.1}
$$

Similarly, the LWQ also has a penalization phase, and this step is shown in Figure 4.4 for the example sentence.

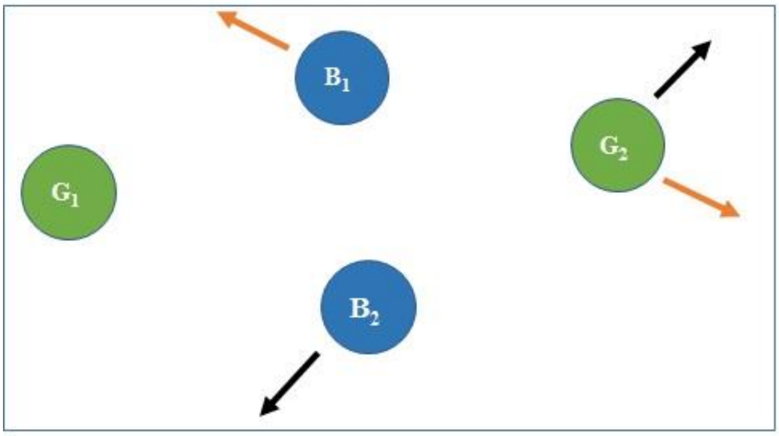


Figure 4.4. Penalization steps for the training in a sample sentence

In the penalization phase shown in Figure 4.4, lemma candidates (*G_2*) which are not the correct lemma candidate and unambiguous word lemmas (*B_1* and *B_2*) are departed from each other. Process calculations are shown in (4.2), which are obtained by changing the direction of the signs in (4.1).

$$B_1(t+1) = B_1(t) + \eta\big(B_1(t) - G_2(t)\big)$$
$$G_2(t+1) = G_2(t) - \eta(B_1(t) - G_2(t)) \tag{4.2}$$

Using directional distances between motion pairs, both vectors take a step toward each other in η (eta) value. Convergence control is performed with the script given below.

```
if $S$(t) ≤ $S$(t-1) {
        cnt = cnt + 1;
        eta = eta / 2;
        if cnt > 5 break;
    }
```

S(t) is the last obtained success (correct classification ratio) in training, S(t-1) is the previous one. The lack of improvement in training achievement in convergence control is compensated by dividing the learning rate into two. If this repeats five times, training is terminated.

When all “motion pairs” in the prepared training datasets are listed separately, it is observed that some pairs consistently approximate or depart. Suppose that, the unambiguous word $\mathrm{B_X}$ and the ambiguous word lemma candidate G_Y should pass together on two different rows of the train set. In the first row, G_Y is the correct candidate and in the second row, it is the incorrect candidate. Accordingly, one of the “motion pairs” prepared for the first row of the train dataset will be (B_X, G_Y, +1) and for the second row, it will be (B_X, G_Y, -1). Considering the characteristic training logic of the LWQ algorithm, it can be said that these two motion pairs will create instability. Because they neutralize each other, these type of “motion pairs” are called “ineffective motion pairs” in this study. These pairs are excluded from training datasets by considering that they could not contribute to training. For this purpose, a simple training dataset is prepared by selecting *w*=2 since a maximum number of “motion pairs” from the training dataset can be obtained. “Ineffective motion pairs” are determined by using this dataset. The lemmas constituting these pairs are collected from the “motion pair” data structure and saved in a list called “ineffective lemma list”. $\mathrm{L_F}$ term is used for “ineffective lemma list”. Then, when training datasets are prepared, these ineffective lemmas are used to filter unambiguous words’ lemmas. Thus, it is ensured that no ineffective lemmas are present which behave as stop-list in LWQ training.

The processing steps of the proposed system (data preparation, LWQ training) are given below.

1. Request window width (w) as input.

2. Select two unambiguous words around the target ambiguous word for the given window width (by ignoring the unambiguous words in the filter list (L_F)). Add them into the dataset as a new row. The added row will be in (B_1,B_2,...B_w : G_1,G_2,...,G_m). Here, the terms B_i are unambiguous word lemmas (lemma+POS) entering the window, and G_i are the lemma candidates (lemma+POS) of the ambiguous word produced by the morphological analyzer. The value of m may vary in each row, depending on the number of candidates of each ambiguous word.
3. Prepare a vocabulary $V = V_U \cup V_C$, which includes all unambiguous words' lemmas and candidates in the dataset.
4. Request vector size (N) as input.
5. Divide the dataset into two as train and test. Define a random initial point in the N dimension for each lemma in V.
6. For each row in the train set, calculate $\underset{1 \le i \le m}{\operatorname{argmin}}\left(\sum_{j=1}^{w} \left\| G_i - B_j \right\|\right)$ and select the winning candidate. If it is not correct, update the word vectors positions.
7. If convergence is not achieved, go to Step 6, otherwise, stop.

Considering that each lemma constituting a "motion pair" is represented by a vector, each "motion pair" can be considered as an equation. With these equations, vectors behave like two magnets that attract or repel each other. As the width of the window sliding over the sentences increases, the number of properties in the dataset and hence the number of "motion pairs" (equations) increases. In statistical studies, it is known that as the number of equations increases for a given number of variables, the probability of solving the problem increases. Therefore, defining a wider window can mean getting more equations and finding more reasonable solutions. Nevertheless, from the training datasets, for an absolute solution, a sufficient count of equations may not be obtained. This is exactly like trying to solve a system of two unknowns with one equation. In this way, the problem of "insufficient equations" can cause the system to propose different results each time as inconsistent outputs. The only thing that can be done is to increase the number of equations, both to ensure consistency and to improve accuracy. In fact, it can be said that there is a need for equations (relationship definition) which provides all possible word relations between V_B and V_G that can be established as a complete bipartite graph. Although we have generated complete bi-partite relationships on each row of the train set, it is not possible to achieve this for the entire vocabulary. Therefore, it would be more accurate to focus on the determination rather than ensuring consistency.

Both the problem of "insufficient equations" described above and the fact that word vectors start training in random positions raises concerns about the consistency of classification (COC). In order to evaluate this situation, the most commonly used approach in the literature is to examine the outcomes by repeating the training at different times. The proposed LWQ algorithm is repeated 100 times in all experiments, on each training dataset and the variability of the classified outputs by the algorithm is analyzed. Accordingly, LWQ estimates some ambiguities in the train dataset consistent-correct (always correct), only a small part consistent- incorrect (sometimes correct and sometimes incorrect), and the rest as inconsistent (sometimes correct, sometimes incorrect). The effect of the window-size and the size of the Euclidian space defined for the word-vectors to the COC and accuracy are presented and interpreted below.

LWQ algorithm shows some similarities with the original LVQ algorithm. Both use supervised datasets. But they also differ in some way. At first, the models are different. LWQ uses unambiguous word lemmas as inputs and ambiguous word lemma candidates are prototypes. Given a data row, each prototype belongs to a single class as "correct" and "incorrect". Vocabularies of the input and prototype words are coded as vectors and represented in the same space with random locations. LVQ algorithm is a pure NN (Neural Network) where, there are mainly three layers as input, Kohonen and output. Kohonen layer consists of prototypes which are represented as reference vectors. Reference vectors are a subset of input vectors and their classes are assigned randomly, not as supervised in LWQ. Only input vectors' classes are known. In each iteration, a Euclidian distance calculation is done for each input vector to all the reference vectors in the model independently. But in LWQ, training is limited to a row (which represents an ambiguous word) regarding its candidates and the unambiguous word lemmas. In LVQ, all reference vectors are used in Euclidian distance calculation of an input vector, but in LWQ, only the ones in the same line, dependently. Porotypes of LVQ use a weight matrix where LWQ does not benefit from a weight matrix.

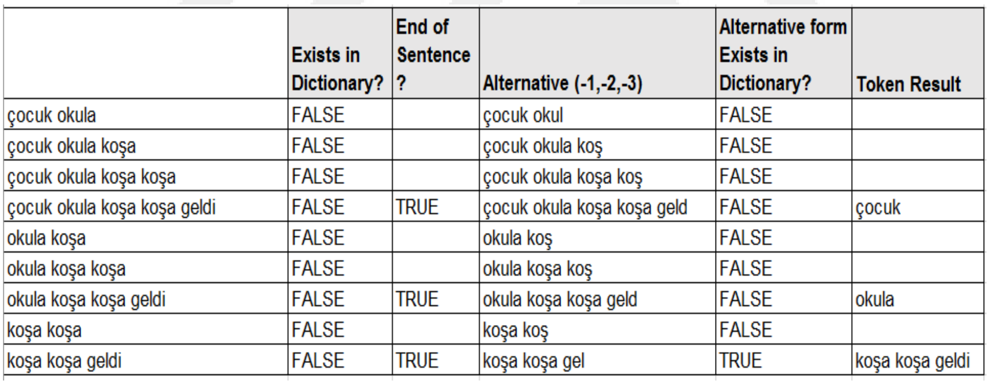
These two algorithms differ also in method. LVQ aims to assign a class to an input vector by using the information of the reference vectors. But LWQ calculates the optimum locations of the input and reference vectors by using the class information. Reward steps are similar in logic, but implementation is different. In LWQ, limited to a data row, each input vector's distance to the prototypes (i.e. candidates of ambiguous word) are calculated separately. If the prototype with the correct class is more near than the other prototypes (in the same row) to the input vectors (in the same row) no action is taken, otherwise, the correct prototype and these input vectors approximate (reward) to each other while the other prototypes and same input vectors depart (punish) each other. In LVQ the nearest prototype vectors are calculated for a given input vector and the nearest prototypes are departed or approximated according to being in the different class or not.

4.2. Multi-word Tokenizer

As mentioned before, all the texts in the dataset are split into sentences and each sentence is tokenized. Original codes of morphological analyzer include a basic tokenizer that tokenizes sentences word by word in its normal operation.

Compound words can exist in sentences in their simple or inflected forms. In order to identify these multiple word strings, we have developed a tokenizer function and replaced it with the original one.

This tokenizer iteratively examines each word and the following words in the sentence and detects the possible multi-words according to their presence in the dictionary. When the detected word group exists as a compound word in the dictionary, it is defined as a token. The tokenization process continues until the end of the sentence iteratively by assigning the last word of the recently detected word group as the start point. The possibility that a word group is an inflected form of a compound word, is analyzed by searching similar words in the dictionary by subtracting two and three letters from the end. Tokenizer function execution steps for an example sentence is presented in Figure 3.12.



	Exists in Dictionary?	End of Sentence ?	Alternative (-1,-2,-3)	Alternative form Exists in Dictionary?	Token Result
çocuk okula	FALSE		çocuk okul	FALSE	
çocuk okula koşa	FALSE		çocuk okula koş	FALSE	
çocuk okula koşa koşa	FALSE		çocuk okula koşa koş	FALSE	
çocuk okula koşa koşa geldi	FALSE	TRUE	çocuk okula koşa koşa geld	FALSE	çocuk
okula koşa	FALSE		okula koş	FALSE	
okula koşa koşa	FALSE		okula koşa koş	FALSE	
okula koşa koşa geldi	FALSE	TRUE	okula koşa koşa geld	FALSE	okula
koşa koşa	FALSE		koşa koş	FALSE	
koşa koşa geldi	FALSE	TRUE	koşa koşa gel	TRUE	koşa koşa geldi

Figure 4.5 Execution steps of the tokenizer function in an example sentence

4.3. OOV Discovery

Morphological analyzers are mostly dependent on their static dictionaries. OOV words cause ineffective operation of the morphological analyzer by causing many unknown words. In this section, we share the details of a system (Arslan and Orhan, 2019) that can discover OOV words in a semantical graph. This system is also capable of discovering collocations.

This graph system is composed of four phases and is shown in Figure 4.6.

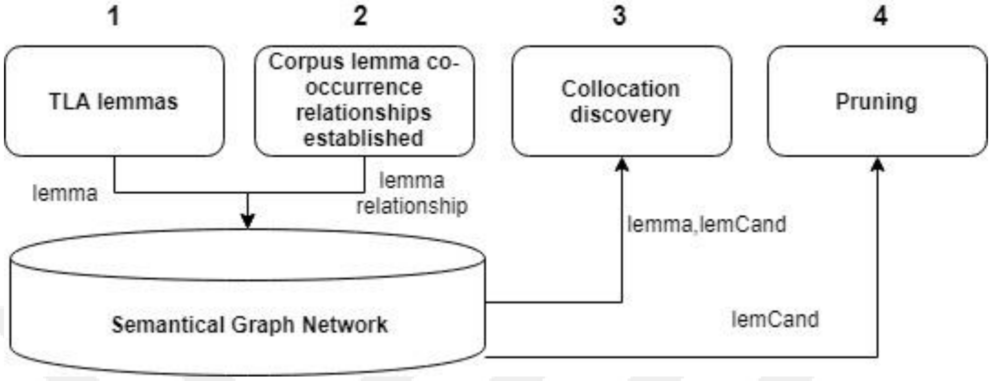


Figure 4.6 OOV discovery system schema

In the first phase, lemmas of the TLA dictionary (from Güncel Sözlük) are added as nodes. In the second phase sentences which are parts of news texts (Tahiroğlu B.T., 2014) are tokenized and lemmatized (details will be given later). When lemmas are obtained, co-occurrence information is created as graph relations for each of them. In the third phase, collocations are discovered. In the fourth phase, discovered lemma candidates are pruned by deleting the unnecessary ones.

As a dataset, 50,000 sentences are used which were retrieved from Cukurova University Turkoloji Corpus (Tahiroğlu B.T., 2014). Sentences are processed and for every 1,000 sentence, collocation discovery and candidate pruning were applied.

In the beginning, all the TLA lemmas are added as nodes to a graph database (Neo4J). Statistics of the TLA lemmas are given in Table 4.1.

Table 4.1. Statistics of TLA lemmas for semantic graph

	Verb	Noun	Total
Lemma count	12,706	66,658	79,364
Lemma count with double POS tags			535

Since there are some lemmas with both Noun and Verb POS tags, distinct nodes are created for each and each may have different graph relations.

Tokenization simply tokenizes each word in the sentence regarding the spaces between them. But also tokenization is capable to detect compound words in the sentence. Each token phrase is searched in the graph to be a TLA lemma compound with the same name. Possible compound TLA candidates are retrieved as candidates and checked if the surface form (token phrase in the sentence) can be an inflected form of these candidates with affix validation function (AVF) function (Arslan and Orhan, 2017). If one candidate is possible, and it has n words, tokenizer goes to the $(n+1)^{th}$ word in the sentence.

For example, for the sentence: “Çocuk okula koşa koşa gitti” the tokenizer will detect the TLA lemma “koşa koşa git” as Verb from the graph, and the tokens will be as: “Çocuk”, “okula”, “koşa koşa gitti”.

After tokenization, Lemmatization starts. It searches for all similar forms of the token in the graph. If there is one, node frequency increases by 1. Otherwise, lemma candidates (similar words) of the token are retrieved from the graph and they are checked with AVF function. When this function returns true, the detected (checked) candidates’ node frequencies are incremented by 1. A node will be created for the token (i.e. inflected word) with “Word” tag and it establishes relations (MORPH) with the lemmas. If no candidate returns, this is an unknown word (UW) and lemma discovery function (LDF) runs.

UW’s are the input to the LDF function. LDF simply removes a character from the UW and behaves each substring as a possible lemma. All possible lemma candidates are collected in this way. These candidates are validated with AVF function. If the validation returns true, the substring is a new node with the "LemCand" label. The newly created LemCand nodes are connected to the UW node with MORPH relation type and the node and relation frequencies are increased by 1. The pseudo-code of the LDF function is as follows:

```
N = LEN(X) - 1
$X_{cand}$ = LEFT(X, N)
WHILE (N >1)
checkVal = AVF ($X_{cand}$, X, “noun”)
IF (checkVal)
CREATE Node {name:$X_{cand}$, postag:“noun”}
checkVal = AVF ($X_{cand}$, X, “verb”)
IF (checkVal)
CREATE Node {name:$X_{cand}$, postag:“verb”}
N = N – 1
$X_{cand}$ = LEFT($X_{cand}$,N)
END WHILE
```

where N is the number of characters of the UW, X_{cand} is the substring to be checked. Function named "LEFT" takes the left N characters from the X string as a new substring. Two nodes are created with Noun and Verb POS tags, some of them will be eliminated in the pruning phase.

When all possible lemmas and lemma candidates (LemCands) in a sentence are detected, a n-to-n neighboring relation is established within each other with the "COOCCUR" label. If this relation exists before it is incremented, otherwise, a 1 value is given for the relation.

A summary of the created nodes and relationships in the semantic graph are given in Table 4.2

Table 4.2. Node and relation labels summarized

Label	Type	Description
Word	Node	Inflected word
Lemma	Node	Lemma originates from TLA Dictionary
LemCand	Node	Lemma candidate discovered by LDF
VerLemCand	Node	Verified Lemma candidates selected after pruning
CollCand	Node	Collocation candidates discovered
COOCCUR	Relation	Relations established between Lemmas and LemCands
MORPH	Relation	Relations established between Lemmas (LemCands, VerLemCands) and Inflected Word

Collocation discovery is achieved by using the frequency statistics of the node pairs with a simple formula:

```
IF (R.neighFreq/R.relFreq)>0.86 AND (r.neighFreq>10)
THEN CONCATENATE n,m
```

where neighFreq denotes the frequency of token lemmas that cooccur together. relFreq denotes the general frequency value to be neighbor in any distance. The threshold values of 0.86 and 10 are defined empirically. The pairs consistent with the formula are concatenated as a new word (collocation) and a node created with the label “CollCand”.

Statistics of the semantic graph when all sentences in the dataset are processed is given in Table 4.3.

Table 4.3. Statistics of the semantic graph

Description	Label	Count
Lemma node	Lemma	79,364
Lemma candidate node	LemCand	47,979
Verified Lemma candidate node	VerLemCand	4,312

Nodes with Lemcand labels are mostly meaningless. But of course, there are real discoveries. After the processing of each 1000 sentence, a pruning job starts and cleans the unnecessary LemCands. The remaining ones change the label as “VerLemCand” as being verified. This job is also responsible to execute the collocation discovery. Some examples of the collocation discovery are given in Table 4.4.

Table 4.4. Some examples of the discovered collocations

Discovered word	Frequency value
başta olmak	93
konu olmak	59
daha fazla	26
üzerinde bulunmak	22
maç oynamak	21
bayram günü	21
diye düşünmek	17

The lemma discovery flowchart is given in Figure 4.7.

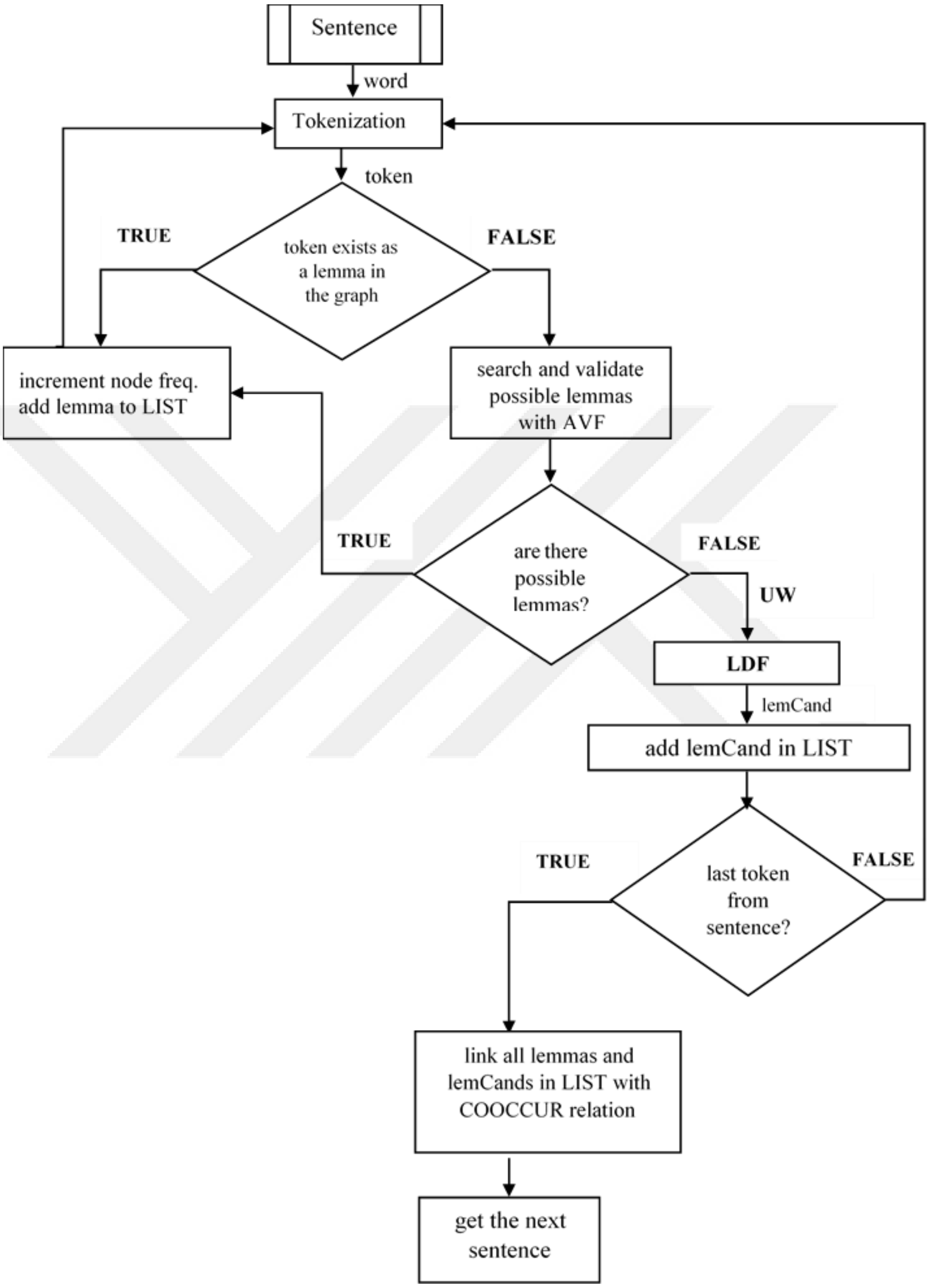

Sentence
word
Tokenization
token
TRUE
token exists as a lemma in the graph
FALSE
increment node freq. add lemma to LIST
search and validate possible lemmas with AVF
TRUE
are there possible lemmas?
FALSE
UW
LDF
lemCand
add lemCand in LIST
TRUE
last token from sentence?
FALSE
link all lemmas and lemCands in LIST with COOCCUR relation
get the next sentence

Figure 4.7 OOV discovery process flowchart

5. RESULTS AND DISCUSSIONS

In this section, experimental results are explained in detail. Comparison results are given with the related studies performed for the morphological disambiguation of Turkish.

The experiments were run on a PC which has a Windows 10 Pro operating system, 64 GB of RAM, Intel Core i7-8700 3.2 GHz double-core processor. The processing load is handled by 2 GeForce GTX 1080 GPU units.

5.1. Performance Metrics for the Proposed Method

In this study, accuracy parameter is used to measure the success of the LWQ classification because it is the well-known and most used metric in morphological disambiguation literature.

$$Accuracy = \frac{C_{amb}}{N} \tag{5.1}$$

where *accuracy* denotes the correct prediction ratio, C_{amb} represents the correctly predicted (by LWQ) ambiguous word count (one data line consists of only one ambiguous word) and *N* is the data line count in a dataset.

5.2. Experimental Results for the Proposed Method

This section includes k-fold validity tests and the results obtained by examining the window-size (*w*) and vector space size on accuracy and consistency of classification (COC). Also, the causes of inconsistency in classification are analyzed.

5.2.1. k-fold Validity Tests

In order to measure test accuracy as accuracy, and to measure validity, the k-fold approach was used. Here, various experiments have been performed for the values of *k* between 2 and 10, where this interval is commonly used in the literature. In a study (Altintas et al., 2005), which implements windowing, they have acquired the highest accuracy value with window-size 14. For this reason, we have selected window-size as 14. Our study has similarities with Glove (Pennington et al., 2014) and they have used the word-vector dimension as 300 for the best results. In this inspiration, we have chosen a close value as 200, for the word-vector dimension. accuracy and processing time values for different k-fold values regarding the parameters as the window-size (*w*) 14 and word-vector dimension (*N*) 200 are presented in Table 5.1.

Table 5.1. K-fold validity test results for different *k* values (*w*=14 and *N*=200)

	2	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Accuracy **(%)**	85.14	85.27	85.60	85.02	85.64	85.34	85.01	84.96	85.55
Time (min)	103	157	191	236	241	293	325	358	361

According to Table 5.1, it has been determined that different *k* values affect accuracy at a minor level. For *k* values 2 and 3, although accuracy values are close, the system spends 1.5 times more processing time. Similarly, when the algorithm runs for *k*=6, which provides the best accuracy value, it spends nearly 2 times processing time when compared to *k*=2. It can be seen that; different k values affect success at a minor level with low running time. When a large number of experiments are planned it is considerable to select the optimum *k* value as 2. Because of this, *k*=2 value is used in the tests as 2-fold cross-validation where 50% of the dataset is used as train and other 50% as test datasets.

After the determination of the validity requirement and accuracy definition requirements, it is better to optimize the parameters which affect the results of the study. In this consideration, the effects of two important parameters (window-size (*w)* and word-vector dimension (*N*)), which could directly affect the results, are examined in the following experiments.

5.2.2. Effects of Window-Size and Word-Vector Dimension on Accuracy and COC

This section includes experiments to measure accuracy for different word-vector dimension values, on different train datasets which are prepared by changing the window-size. The preparation of the train datasets requires changing the window-size by using the windowing method, while the adjustment of the word-vector dimension requires the retraining of the LWQ algorithm. Table 5.2 shows the results of some experiments conducted on different values of these two parameters. The experiments are repeated 100 times to measure the consistency (COC) of the accuracy. Training success is between 97% and 100%.

Table 5.2. The effects of window-size and word-vector dimension on accuracy (%)

Word-vector dimension (N)															
	2			20			200			2000			5000		
(w)	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**
2	66.1	0.9	84.0	74.3	5.8	96.1	75.9	9.7	97.4	79.0	22.3	97.9	80.8	30.1	97.9
6	73.7	4.3	97.6	79.2	14.5	98.0	81.4	28.8	97.5	84.8	46.0	97.1	85.8	53.0	96.7
10	75.1	6.3	97.8	80.7	20.3	98.2	83.6	37.5	97.7	86.3	54.5	96.7	86.9	60.6	96.3
14	75.6	7.3	97.7	81.5	24.0	98.0	84.6	44.1	97.5	86.8	59.0	96.4	87.1	62.9	96.2
18	76.1	8.1	97.6	82.1	26.5	98.1	85.1	47.0	97.3	86.6	59.8	96.3	86.9	63.2	95.9
22	75.9	8.2	97.5	82.1	28.8	97.9	85.1	48.0	97.1	86.5	59.6	96.1	86.6	62.8	95.8
26	75.0	8.3	97.0	82.1	30.0	97.4	84.9	47.7	96.8	85.9	58.5	95.9	86.0	60.6	95.6

The terms used in Table 5.2 are as follows: S_1 is the average value of the accuracy when the training is applied for 100 times (**average accuracy)**. S_2 is the average value of the ambiguous terms, which are classified as always correct or always incorrect in all 100 training cycles (**COC ratio**). S_3 is the average value of the correctly classified terms in consistent data rows for all of the training cycles (**consistent accuracy**). According to Table 5.2, both average accuracy (S_1) and COC ratio (S_2) values increase for the parameters *N* and *w* in direct proportion. However, the rate of increase in accuracy is decreased, when the window-size exceeds 14. Likewise, when the word-vector dimension is greater than 200, there are little increases in the values of S_1, S_2, and S_3. Interestingly, S$_3$ isn't too much affected by the change in parameters *N* and *w*, except for *w*=2. In experiments, word-vector dimension starts by the value *N*=2 and increments with a factor of 10. Because of the limitation of the GPU-card memory, a maximum dimension of 5000 is used.

When test results are considered, standard deviation values are lowest and very near for window-size values 10 and 14 for N=2000 as 0.247 and 0.250, respectively. For the same windows-size values when the test is applied for n=200, the values are 0.27 and 0.31 and for N=5000, the values are 0.21 and 0.26, respectively. Standard deviation values are very high for low window-size (w=26) as 1.42 and values change between 0.68 and 1.42 for all dimensions.

In order to measure the effect of the “ineffective lemma pairs” in accuracy, filtered datasets are also trained by LWQ. accuracy values provided by these filtered datasets are presented in Table 5.3.

Table 5.3. The effects of window-size and word-vector dimension on accuracy when filtered with ineffective lemma pairs (%)

Word-vector dimension (N)															
	2			**20**			**200**			**2000**			**5000**		
(w)	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3	S_1	S_2	S_3
2	66.3	0.7	88.5	74.5	5.3	97.1	76.1	9.3	98.3	79.3	22.5	98.3	81.1	31.8	98.2
6	74.1	4.0	97.5	79.3	13.6	98,2	81.8	28.4	97.9	85.2	47.8	97.4	86.3	55.2	97.0
10	75.5	6.5	97.9	80.9	19.4	97.9	83.9	37.6	98.0	86.7	57.3	97.0	87.2	61.9	96.6
14	76.1	7.4	98.1	81.6	23.7	98.3	84.8	44.0	97.8	87.0	60.3	96.6	87.3	64.6	96.3
18	76.5	8.5	98.0	82.0	26.3	98.3	85.3	47.1	97.5	87.0	61.8	96.3	87.2	65.2	96.0
22	76.6	9.1	98.0	82.3	28.3	98.0	85.4	48.8	97.2	86.8	62.3	96.1	87.0	65.2	95.8
26	76.7	9.2	98.0	82.6	30.1	97.6	85.5	49.6	97.2	86.7	62.1	96.1	86.8	65.1	95.9

When the experiment results in Table 5.3 are compared with Table 5.2, it can be seen that accuracy is increased in [3..5]% with the usage of the datasets, which discards the “ineffective lemma pairs” (filtering). All remaining values are in accordance with each other. While window-size values *w*=10 (Yuret and Türe, 2006; İlgen et al., 2013) and *w*=14 (Altintas et al.,2005) provide the best accuracy values for Turkish, this complies with the experiment results given in Table 5.2 and 5.3.

Standard deviation values for the tests with filtered data are lowest and very near for window-size values 10 and 14 for N=2000 as 0.22 and 0.25, respectively. For the same windows-size values when the test is applied for n=200, the values are 0.29 and 0.28 and for N=5000, the values are 0.23 and 0.25, respectively. Standard deviation values are very high for low window-size (w=2) as 1.32 and values change between 0.66 and 1.32 for all dimensions.

In another analysis, it was identified that intersection values of the vocabulary list of the words (“lemma + POS”) used in the input (unambiguous word lemmas-V_B) and the output (ambiguous word lemmas- V_G) parts of the datasets were very high and they have included the filtered “ineffective lemmas”.

Table 5.4 presents the numbers of elements in intersection and union of the vocabularies (V_G and V_B) (used in the datasets of experiments given in Table 5.2 and Table 5.3) for different window widths.

Table 5.4. Some relations of the vocabularies for different window-size values

	Window-size (w)						
	2	**6**	**10**	**14**	**18**	**22**	**26**
$V_G \cap V_B$	989	1,146	1,170	1,166	1,158	1,142	1,123
$V_G \cup V_B$	7,398	8,699	8,953	8,984	8,962	8,890	8,820
$V_G \cap V_B$ *	930	1,052	1,059	1,047	1,032	1,010	977
$V_G \cup V_B$ *	7,641	8,795	8,946	8,919	8,839	8,750	8,610

*: datasets filtered with the "ineffective lemmas"

Table 5.4 shows that using "ineffective lemmas" in a filter, has decreased the intersections and unions. When the experiments shown in Tables 5.2 and 5.3 are considered together with Table 5.4, it can be stated that using this filter increases accuracy by decreasing the intersection of vocabularies. This supports the highlighted idea at the beginning of the study: "the vocabulary of the dataset can be modeled by splitting it into two vocabularies of ambiguous and unambiguous words lemmas". To assure this claim, in another experiment, intersection of the vocabularies (in other words, words exist in the dataset in both ambiguous and unambiguous form) of ambiguous and unambiguous words ($V_B \cap V_G$) is used as another filter (intersection filter). By using this filter, the relationship between the input and output parts of the dataset represents a bipartite graph. It can be said that only unambiguous words can be used to detect the correct candidate for the ambiguous word. The experiment results obtained by using the datasets prepared by using the defined filters are given in Table 5.5.

Table 5.5. Effect of different filters on COC and accuracy for window-size 14 (%)

Word-vector dimension (N)															
	2			**20**			**200**			**2000**			**5000**		
(w)	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**	**S_1**	**S_2**	**S_3**
N	75.6	7.3	97.7	81.5	24.0	98.0	84.6	44.1	97.5	86.8	59.0	96.4	87.1	62.9	96.2
F-1	76.1	7.4	98.1	81.6	23.7	98.3	84.8	44.0	97.8	87.0	60.3	96.6	87.3	64.6	96.3
F-2	76.1	7.1	97.9	81.4	20.8	98.7	85.0	40.8	98.4	87.6	60.0	97.2	88.0	65.2	96.9

N: Normal, F-1: Filter-1, F-2: Filter-2

In Table 5.5, the first line (Normal) represents the experiment results for accuracy and COC, when no filter is applied, the second line, when “ineffective lemma” filter (Filter-1) is applied and the third line, when “intersection filter” (Filter-2) is applied to the datasets. According to Table 5.5, the highest accuracy values are obtained when the dataset is prepared as $V_G \cap V_B = \emptyset$ (Filter-2). With these results, it was determined that, LWQ system is negatively affected by the use of unambiguous forms of lemmas in the input parts of the datasets, which of them can sometimes be ambiguous (intersects), For example, morphological analyzer produces “ev+NOUN” as the lemma form of the inflected word “evde”. Although, another inflected form of its lemma enables the derivation of the inflected word “evini” which is ambiguous. The morphological analyzer can produce the lemma candidates as “ev+NOUN” and “evin+NOUN” for the word “evini”. When “intersection filter” is applied to the datasets, such lemmas like “ev+NOUN” will not occur in both the input and output sides of the dataset at the same time. Limitation on the intersection of vocabularies of both ambiguous and unambiguous words with this filter makes a positive impact on the accuracy. But for the short texts which consist of a little count of unambiguous lemmas, this filter may not be applicable.

LWQ system’s full training plan is based on using the approximating and departing equations to locate the word-vectors. Word-vectors are semi-trained when only approximating or departing equations are used for the location positioning. For this reason, types of equations are divided into two: 1) EQ+ for approximating, 2) EQ- for departing. In the study, the word-vectors trained only with EQ+ or EQ- type equations are considered to be semi-trained. Word-vectors start training in random starting points, very distant from each other, with high dimension sizes as 2000 or 5000. When high dimensions are considered, only EQ- type equations can be sufficient and these vectors in the training behave as EQ+ type. This can be the interpretation of why a dataset with a high “insufficient equation” problem can achieve the highest accuracy in high dimensions. But the opposite (EQ+ type word-vectors trained in the low-space size with EQ- type equations only) of this idea isn’t always true. Word-vectors trained in the low dimension size may cause undesirable approximations and EQ+ type training cannot be simulated. Following these claims, it is necessary to examine the sentences to determine the causes of inconsistency and incorrect-consistent results.

5.2.3. Causes of Inconsistency and Incorrect-Consistency

In this experiment, consistent-incorrect (incorrectly classified in all tests) and inconsistently (sometimes correctly sometimes incorrectly classified) classified data lines obtained for *w*=14 and *N*=5000 parameters (which achieve the highest accuracy value in the preceding experiments) are examined by visually. Statistics of data lines used in this analysis are given in Table 5.6.

Table 5.6 Statistics of the dataset for inconsistent and consistent-incorrect lines

Class		**Count**	**Ratio (%)**
Consistent-Incorrect	Noise	137	56
	Normal	108	44
Inconsistent	Noise	2,551	57
	Normal	1,928	43

As seen in Table 5.6, analysis is applied in two groups: “Noise” and “Normal” for consistent-incorrect and inconsistent classifications. “Noise” labeled data lines consist of incorrect lemmatization performed by the morphological analyzer or incorrect tagging applied by the Expert. There can be two reasons for the incorrect lemmatization of the morphological analyzer. First reason is the outdated dictionary that morphological analyzer uses. This can lead to too many unnecessary (i.e redundant) lemma candidates in the parse outputs. The second reason is the parse lists provided by the morphological analyzer which include the roots and their corresponding lemma forms at the same time. This situation increases ambiguity. It can be frequently seen in the noun-to-noun and noun-to-adverb derivations. For the first case, to give an example, “hava + NOUN” and “havalı + NOUN” parse candidates both have the same root and “hava” root is affixed by “-lı” derivational suffix. The morphological analyzer does not have any functionality to prevent the listing of the root when its lemma form exists as “havalı + NOUN” and “hava + NOUN” is also listed as a parse option. An example of the noun-to-adverb derivation is: “sabah + NOUN” and its adverb form can be given as “sabahleyin+NOUN”. They can both listed in the same parse output. For the “Noise” group, expert tagging errors can also be seen in the analysis outputs caused by fast data entry.

In Table 5.6, the “Normal” group consists of the data lines in which morphological analyzer lists the correct parse candidates as lemmas and the expert tags the correct lemma with the correct POS tag. The consistent-incorrect data lines in the “Normal” group represent the words with the same lemma and different POS tags. Correctly tagging of these kinds of words is sometimes very hard for even a native speaker of Turkish. For example, the word morphological analyzer lists “ilginç + NOUN” ve “ilginç + ADJ” lemma candidates for the word “ilginçti”. On the other hand, the “insufficient equation” problem is the main cause of inconsistent data lines in the “Normal” group.

5.2.4. The Effect of “Insufficient Equation” on Accuracy and COC

In this experiment, the “insufficient equation” problem is analyzed, which is the main cause of inconsistency. Our observation is that the candidates which can be sometimes EQ+ or EQ- are consistent. Therefore, two different vocabulary lists are prepared, which consist of the correct and incorrect candidates in data lines calculated as their intersection. Common candidate count value is nearly 17% (which move two-way as EQ+ or EQ-) where the remaining 83% (37% are EQ+, 46% are EQ-) are one-way. Some of the candidates which are in one-way equations also can be consistent. Thus we can introduce a hypothesize as “consistency can be guaranteed by candidates trained with a sufficient number of equations”. In the study, these kinds of candidates are mentioned as “fine-trained” candidates. To define these candidates and their positive effect on COC values, two parameters are created: the number of equations required to define a fine-trained candidate (#E_{ft}), and the number of fine-trained candidates in the same equation (#C_{ft}). Table 5.7 gives the analysis results for the effects of four parameters (N, w, #E_{ft}, and #C_{ft}) on COC.

Table 5.7. Effects of fine-trained candidates on CoC with parameters window width and word-vector dimension (%)

	w	2			10			14			18		
N	E_{FT} / C_{FT}	1	10	100	1	10	100	1	10	100	1	10	100
2	0	0.7	0.8	0.5	0	0	2.2	0	0	3	0	0	3.9
	1	0.5	0.6	0.7	0.6	1.6	7.3	0.6	2.2	8.5	1.2	2.4	8.3
	2	0.7	0.6	1.9	6	8	23.5	7.2	9.8	27.4	8	10.9	30.9
	3	0.6	0.5	0.8	11.7	16.3	84.4	12.1	17.1	79.2	14.7	20.9	87.9
	4	0.5	0.4	0	1.8	2.3	2.7	3.4	4.4	4.6	4.9	7	7.9
20	0	0.7	0.8	1.6	0.4	0.8	13.1	0	0.8	17.8	0	1.1	20.5
	1	1.7	1.8	4.4	2.3	4.6	18.9	2.9	5.9	23.3	3.6	8.5	27.6
	2	5	6.4	22.1	19.8	26.4	51.7	24.8	33.8	53.6	28	38	54.7
	3	9	12	42	25	34.3	100	29.4	40.8	100	31.7	44.1	100
	4	1.6	1.8	2.6	18.8	25	58.9	20.6	27.5	63	22.2	30.2	65.3
200	0	0.7	0.8	4.2	3.4	5.1	31.5	3.5	8.2	38	2.2	10.5	41.5
	1	1.2	2.1	4.2	8.7	15.2	39.9	12	23.2	48.1	10.54	29.7	51.5
	2	8.6	11.3	33	40.8	52.4	67.5	48.2	61.4	71.8	52.3	65.5	73.8
	3	16.6	22	72.9	40.8	53.7	100	46	60	100	48.4	62.1	100
	4	2.9	3.5	6.1	29.5	38.1	80.4	34.5	36.7	86.1	38.3	46.1	90.1
2000	0	0.7	0.8	15.1	5.2	15	51.1	6.2	21.5	55.6	6.8	27.5	57.9
	1	3.8	4.1	16.1	18.9	37.5	63.6	23.4	44.9	62.9	24.1	49.1	63.1
	2	23.9	32.0	68.7	62.3	76.5	86.2	65.7	78.8	83.9	67.9	79.1	81.6
	3	29.5	39.2	80	59	71.7	100	62.1	73.9	100	62.8	73.9	100
	4	24.6	31	93.9	49.2	57.1	100	50.9	58.5	100	50.6	55.4	100

In Table 5.7, the optimum windows-sizes (w=10 and w=14) are used with a minimum (w=2) and maximum (w=18) values. It can be seen that the window-size is ineffective on COC values for window-sizes bigger than 10. This is also valid for low word-vector dimensions. But when high-number of equations (like $\#E_{ft}$>100) are used with high-dimension values (i.e. N=2000), fine-trained candidate count ($\#C_{ft}$) is proportional to COC values. This relation supports the idea that one-way candidates can be fine-trained in high-dimensions. But, sometimes, in some data-lines, fine-trained candidates may not exist and this leads to very low COC as 0.5% (and sometimes 0). This situation shows the positive effect of fine-trained candidates on consistency.

When we examine the counts of data rows involved in training with E_{ft} and C_{ft} parameters we have seen that, for $\#C_{ft}$=0 and $\#C_{ft}$=1 data row count increases with the increase of E_{ft} (1,10 and 100). Surprisingly, this is opposite for $\#C_{ft}$=2 and $\#C_{ft}$=3 and $\#C_{ft}$=4. Although data row counts decrease for these C_{ft} values, COC values are higher when compared to $\#C_{ft}$=0 and $\#C_{ft}$=1. Another case is, when the values of C_{ft}=2 and $\#C_{ft}$=3 or C_{ft}=4, for window-sizes 10,14 and 18, the data row counts decrease but COC values increase. This shows that fine-trained candidate counts which are high quality for classification can be obtained in window-sizes bigger than 10 even there are lower count of data rows. Also for high dimensions (such as 2000) low C_{ft} values as 0 and 1 acquire higher COC values even when E_{ft} gets higher, but their COC values cannot exceed 57.9% and 63.6%, respectively.

Generally, low COC values can be seen for even high-dimensions provided with a high number of equations ($\#E_{ft}$>100). But it is important to have a higher number of equations that are greater than 100 ($\#E_{ft}$>100) to define a fine-trained candidate.

As a result, when high word-vector dimension, optimum window-size and a high number of equations are used, high COC can be obtained with data lines that have 3 or 4 fine-trained candidates.

5.2.5. Effect of Dataset Size on Classification Accuracy and Consistency

Due to the nature of languages, one ambiguous word can co-occur randomly with an unambiguous word in a sentence. It can be claimed that the “insufficient equation” problem can be overcome by enlarging the dataset. In order to test this hypothesis, the existing dataset is decreased randomly to simulate the dataset expansion. In this experiment, 5 subsets of data are created (including the main dataset) by using the “intersection filter” with a window-size 14. The train-test cycle is applied for 100 repetitions and the accuracy and COC are examined. The results are given in Table 5.8.

Table 5.8. The effects of the change in the amount of data on accuracy and CoC (%)

No	Corpus (#Sentences)	Vocabulary (#Words)	$\text{accuracy}_{\text{avg}}$	CoC_{ft}
1	2,401	6,069	76.8	20.0
2	4,802	7,181	81.5	37.9
3	7,203	7,787	84.4	47.9
4	9,604	8,300	84.9	51.9
5	12,009	8,656	85.0	69.0

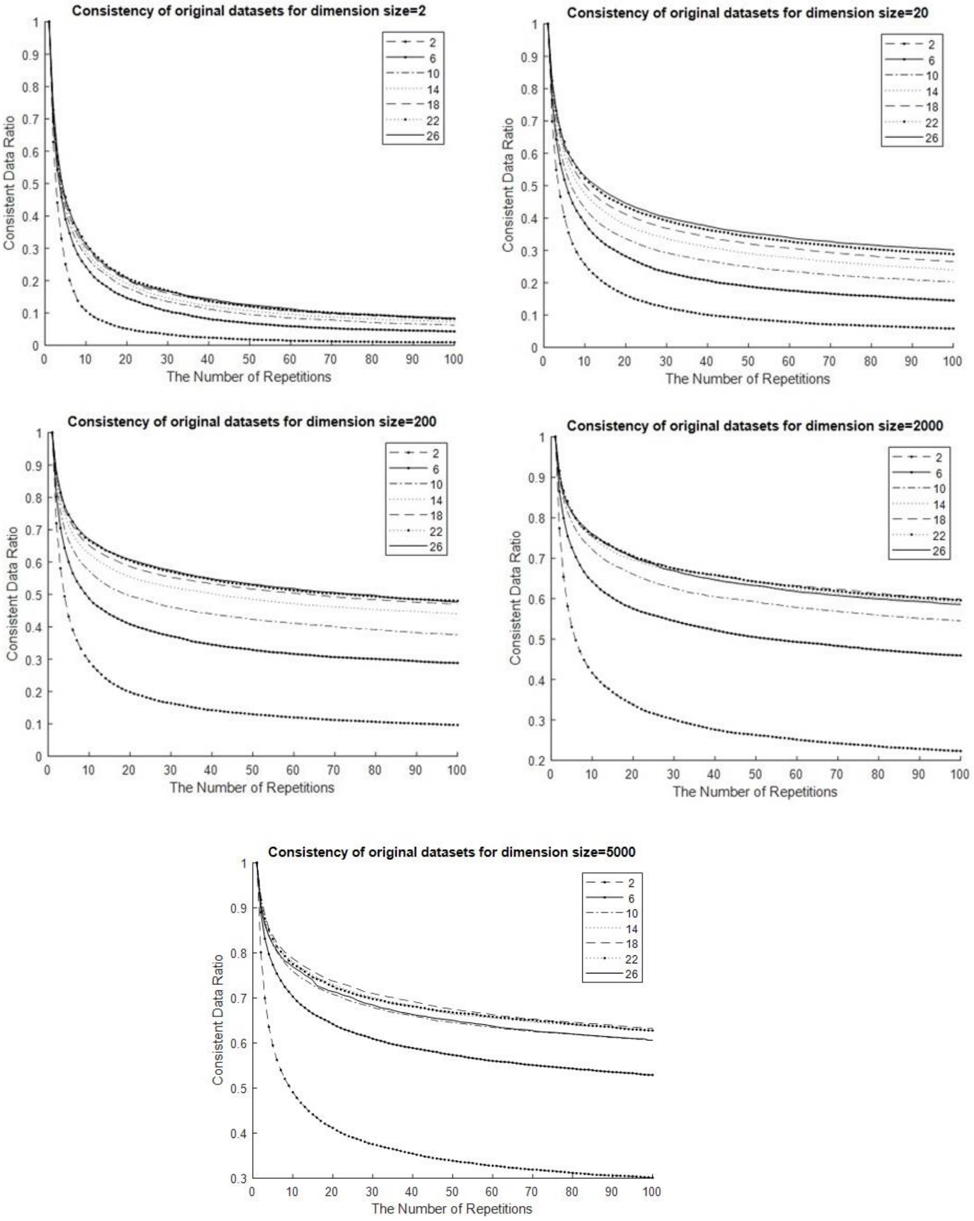
Dataset versions given in Table 5.8 are obtained by randomly subtracting sentences from the novel and news texts to save the balance in the count and these are given in ascending order as word counts. For training and tests, N=200 word-vector dimension and w=14 are used for their representative power on classification. Although it is hard to find common accuracy and COC values, $\#E_{\text{ft}}>34$ and $\#C_{\text{ft}}=2$ values are defined as optimums by searching $\#E_{\text{ft}}$ in [1..100] and $\#C_{\text{ft}}$ in [1..4].

According to Table 5.8, it can be seen that when sentence count increases, COC_{ft} (which represents COC calculated only with fine-trained candidates) increases in accordance with the vocabulary. On the other hand, the increase in $accuracy_{avg}$ (S_3) decreases. For this reason, it can be expected that, when the tagged dataset (more novel and more news text) is enlarged until full consistency is provided, higher accuracy values can be obtained.

5.2.6. Consistency Analysis for all Window-Size Values and Dimensions

In this section, the effect of parameters “window-size” and “word-vector dimension” on COC (S_2) are given in graphical representations.

Figure 5.1, shows the analysis results for the effect of the word-vector dimension on COC (S_2) considering all window-size values used in the experiments.


Consistency of original datasets for dimension size=2

Consistency of original datasets for dimension size=20

Consistency of original datasets for dimension size=200

Consistency of original datasets for dimension size=2000

Consistency of original datasets for dimension size=5000

Figure 5.1. COC analysis for different vector- dimensions on the same window-size values

As seen in Figure 5.1, COC values are low for the lower window-sizes. When word-vector sizes increase gradually, consistency value increases for all window-size values. When the word-vector dimension is 2, the system acquires the worst COC results for all windows. The sharpest increase in COC is obtained by the experiments applied for the dimension range [20..200]. The COC value increase goes by the dimension range [200..2000]. Although the increase continues with the dimension value range [2000..5000], it can be seen that convergence occurs in this range.

Figure 5.2 shows the similar results given in Figure 5.1, differently when the dataset is filtered with the “intersection filter”.

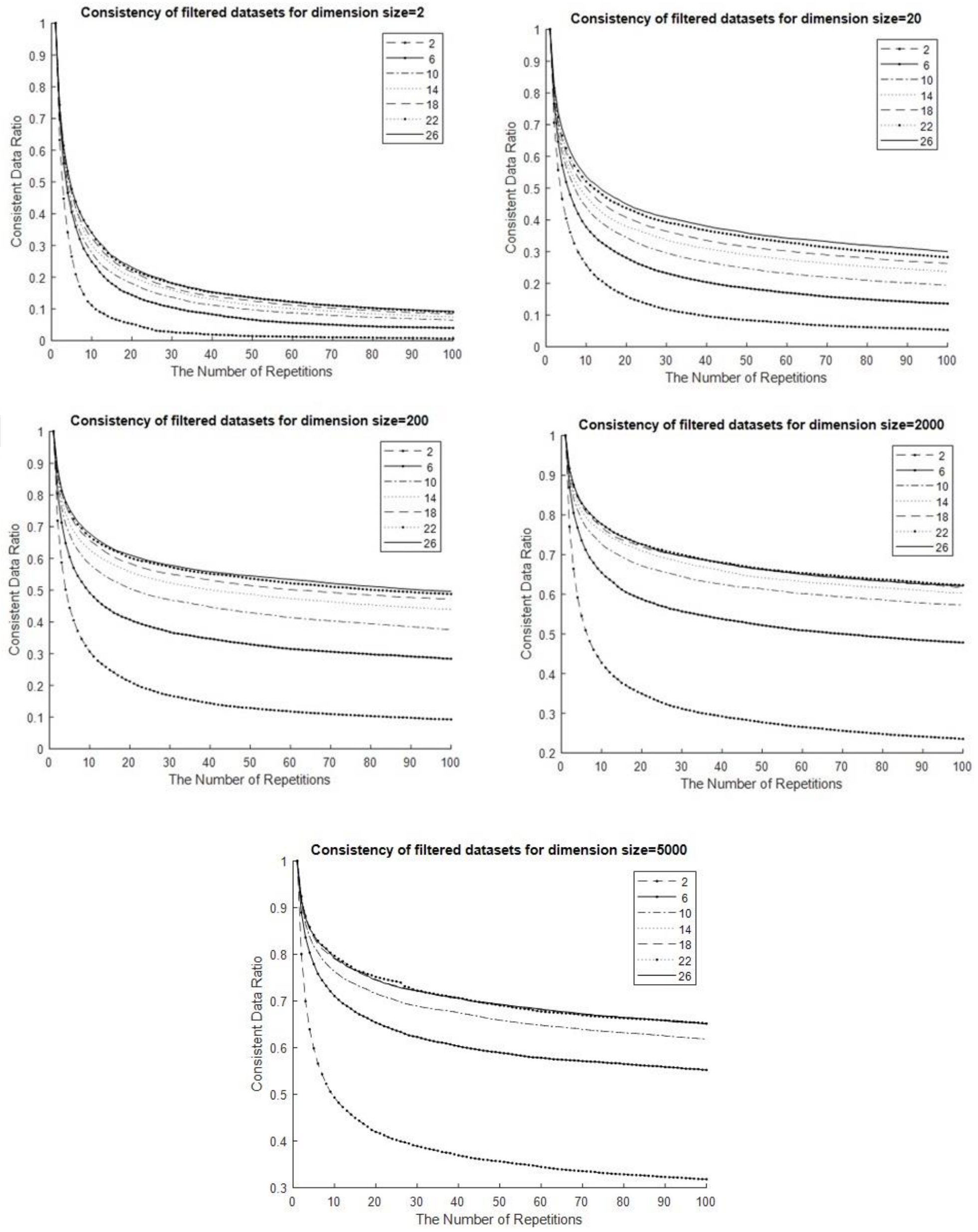
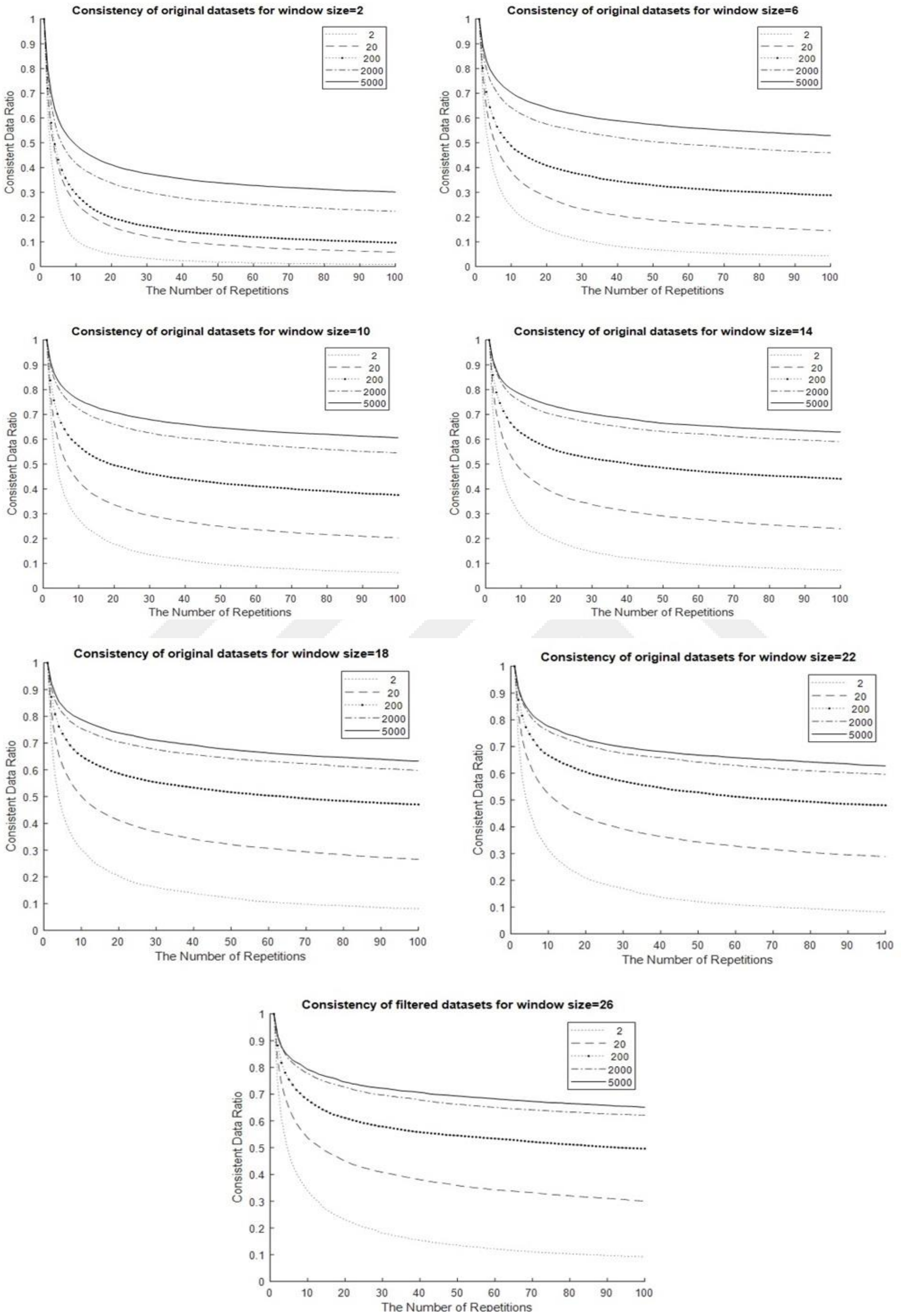


Figure 5.2. COC analysis for different word-vector dimensions on the same window-size values with “intersection filter”

In Figure 5.2, comments made for the Figure 5.1 are valid, except, filtered data provides little increase for all word-vector dimension values on different window-size values. Results are given in Figure 5.3 when the experiments are repeated for each window-size values for all word dimensions through [2..5000].



Consistency of original datasets for window size=2

Consistency of original datasets for window size=6

Consistency of original datasets for window size=10

Consistency of original datasets for window size=14

Consistency of original datasets for window size=18

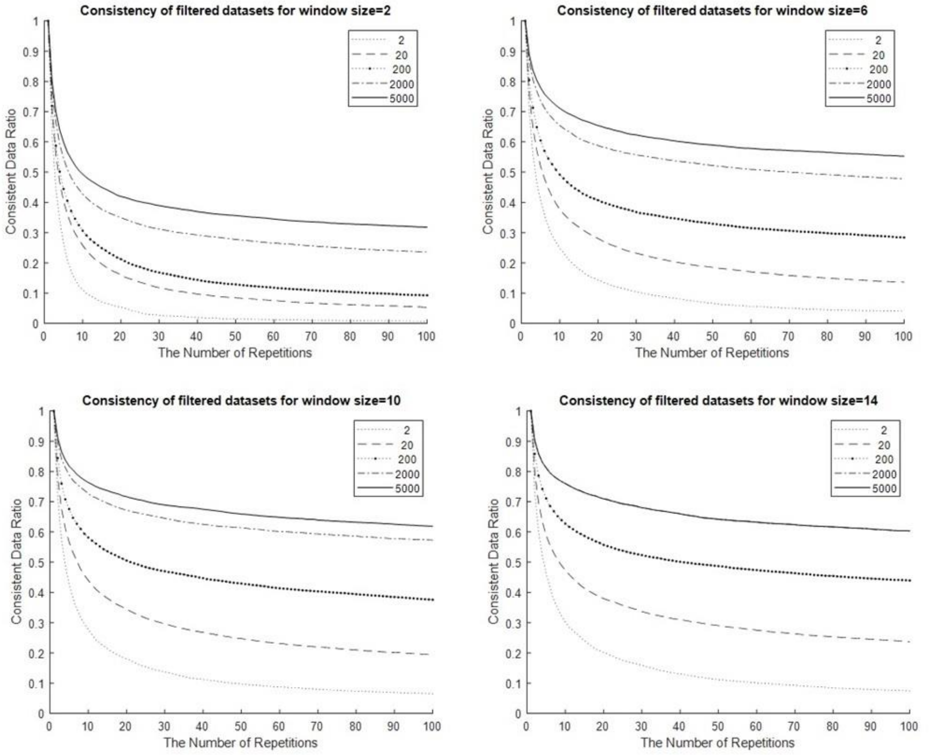
Consistency of original datasets for window size=22

Consistency of filtered datasets for window size=26

Figure 5.3. COC analysis for different window-size values on the same word-vector dimensions

In Figure 5.3, for all word-vector dimensions, COC increases gradually till the window-size is 18. The COC value increase is sharp from *w*=2 to *w*=6. And from *w*=6 to *w*=18 increase is in a fixed ratio. From window-size *w*=18 to window-size values *w*=22 and *w*=26, consistency slightly increases.

When the datasets prepared and used for the tests in Figure 5.3 are filtered with “intersection filter”, results for the same experiments are given in Figure 5.4.



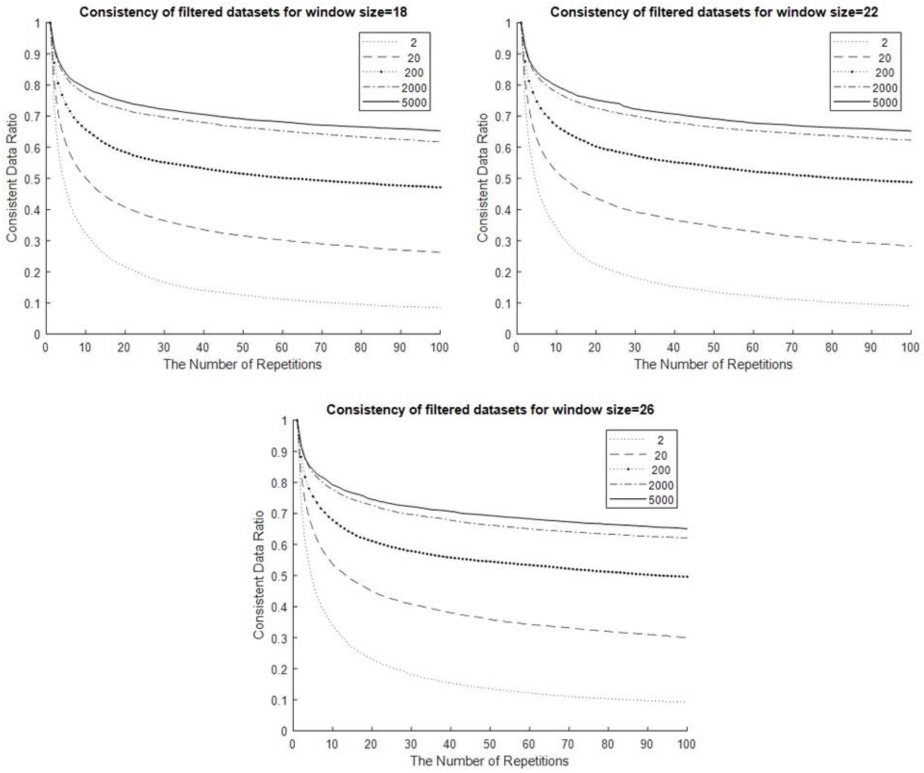


Figure 5.4. COC analysis for different window-size values on the same vector dimensions with "intersection filter"

In Figure 5.4, filtered datasets provide better results, when compared to the results given in Figure 5.3. COC values are near for window sizes 18, 22 and 26 but there are little achievements.

In order to visualize the effect of filtered datasets with "intersection filter" on the system, experiments are performed for all combinations of window-size and word-vector dimensions. In Figure 5.5, for all word-vector dimensions, results are given when the filtered and original datasets are prepared for window-size 2.

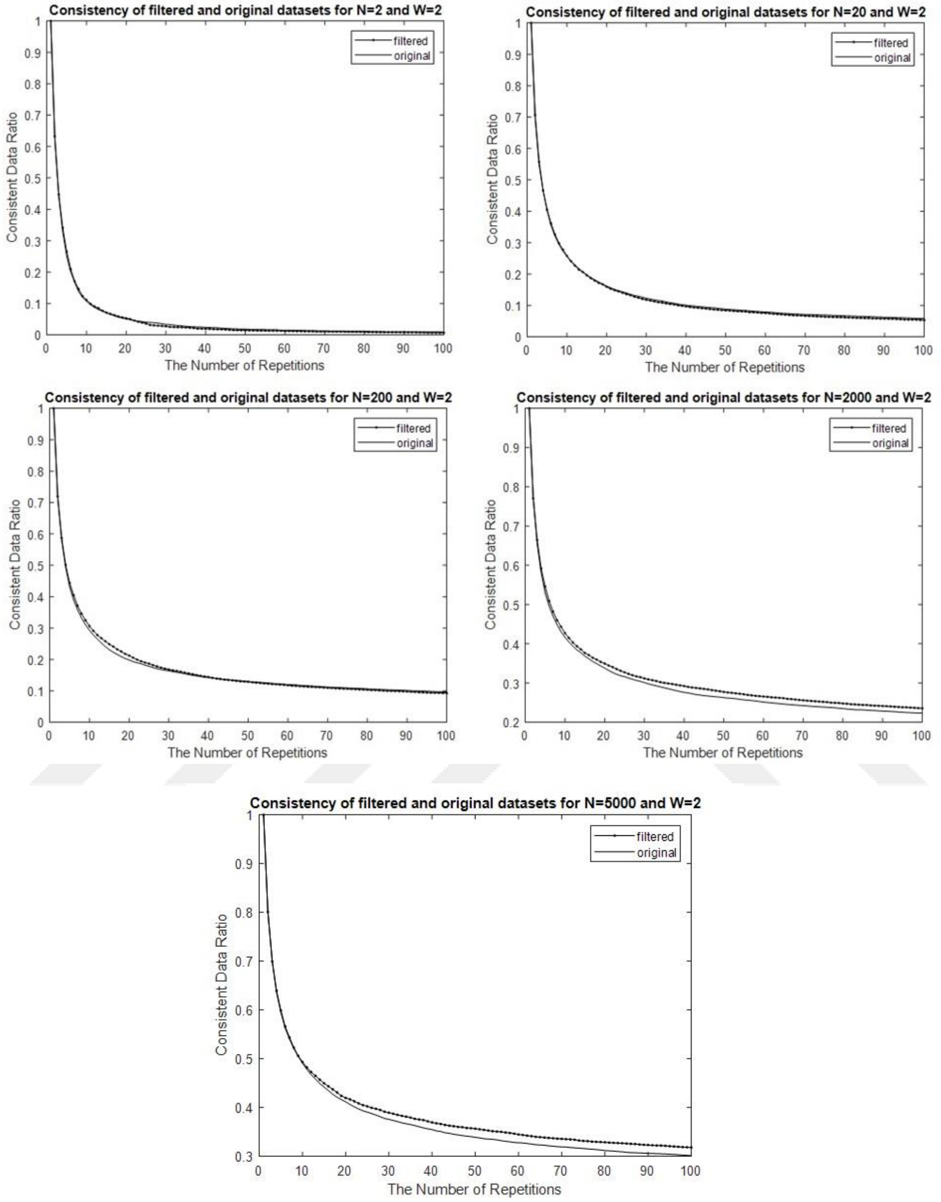


Figure 5.5. COC analysis for window-size=2 on different word-vector dimension values with original and filtered datasets

In Figure 5.5, although there is little increase in COC values for the dimensions *N*=2, *N*=20, and *N*=200, these cannot be distinguished in graphics.

However, the increase in COC with the usage of the filter can be seen easily for the dimensions *N*=200 and *N*=5000.

In Figure 5.6, for all word-vector dimensions, results are given, when the filtered and original datasets are prepared for window-size = 6.

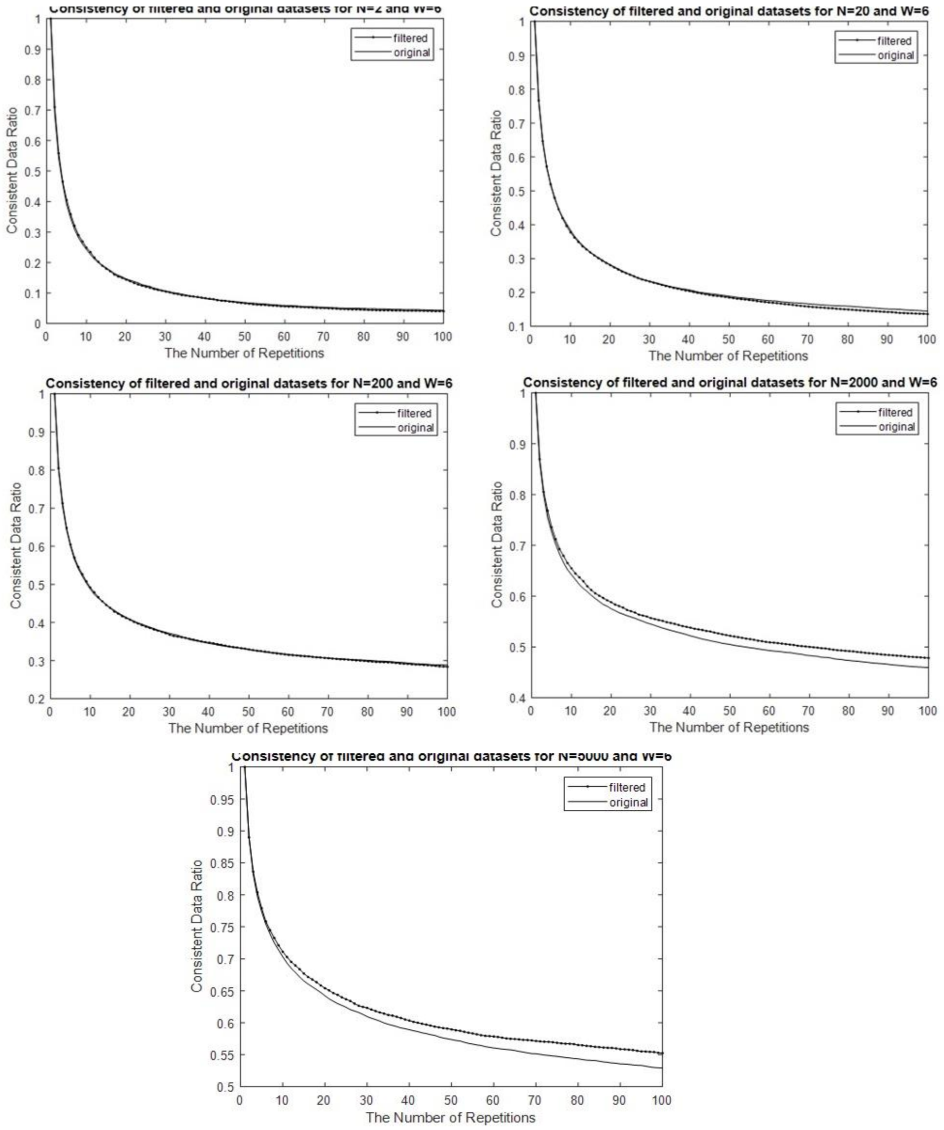


Figure 5.6. COC analysis for window-size=6 on different word-vector dimensions with original and filtered datasets

In Figure 5.6, although there is little increase in COC values for the dimensions *N*=2, *N*=20, and *N*=200, these again cannot be distinguished in graphics. For *N*=20 the increase in the last repetitions can be seen. However, the increase in COC with the usage of the filter can be seen easily for the dimensions *N*=200 and *N*=5000.

In Figure 5.7, for all word-vector dimensions, results are given when the filtered and original datasets are prepared for window-size = 10.

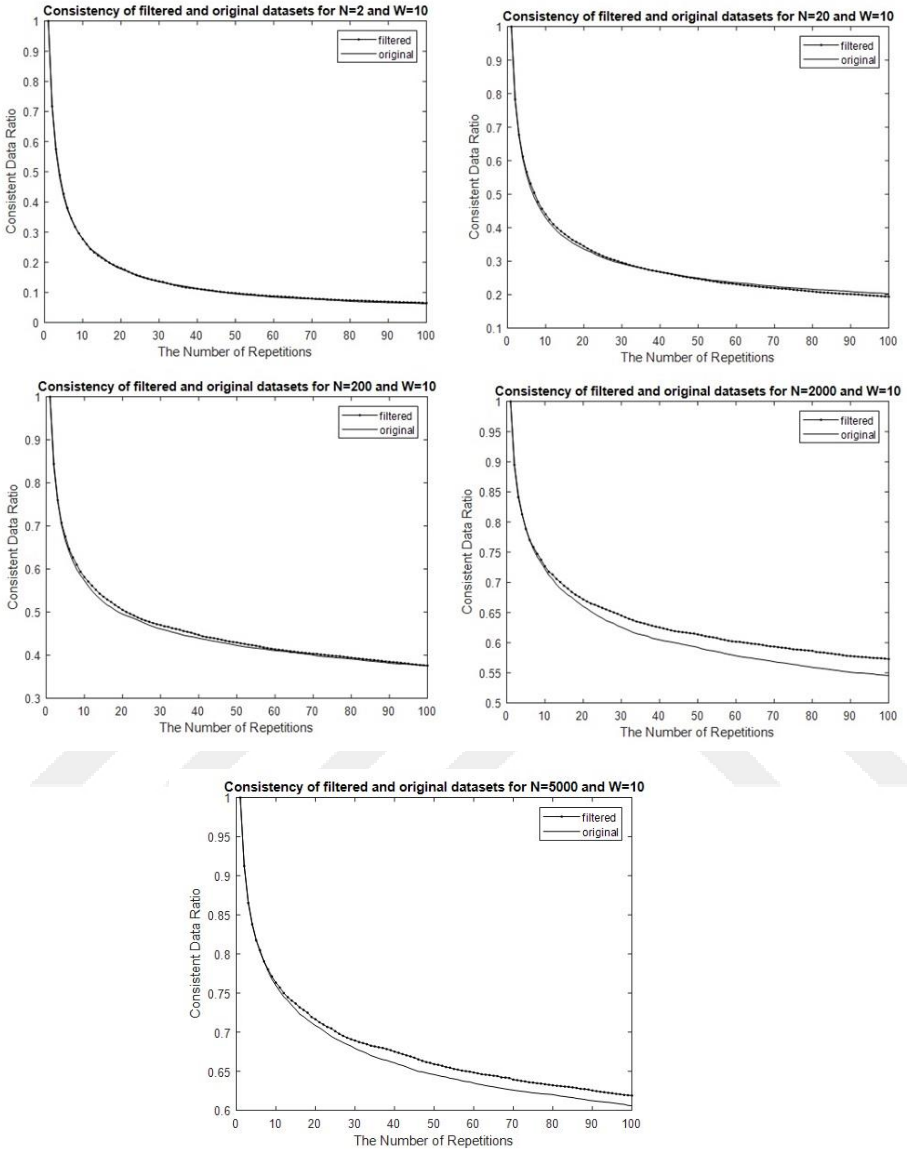


Figure 5.7. COC analysis for window-size=10 on different word-vector dimensions with original and filtered datasets

In Figure 5.7, for *N*=20 and *N*=200, an increase in consistency is more evident when compared to the previous window-size values. The increase in COC, with the usage of the filter, can be seen easily for the dimensions *N*=200 and *N*=5000.

In Figure 5.8, for all word-vector dimensions, COC results are given when the filtered and original datasets are prepared for window-size = 14.

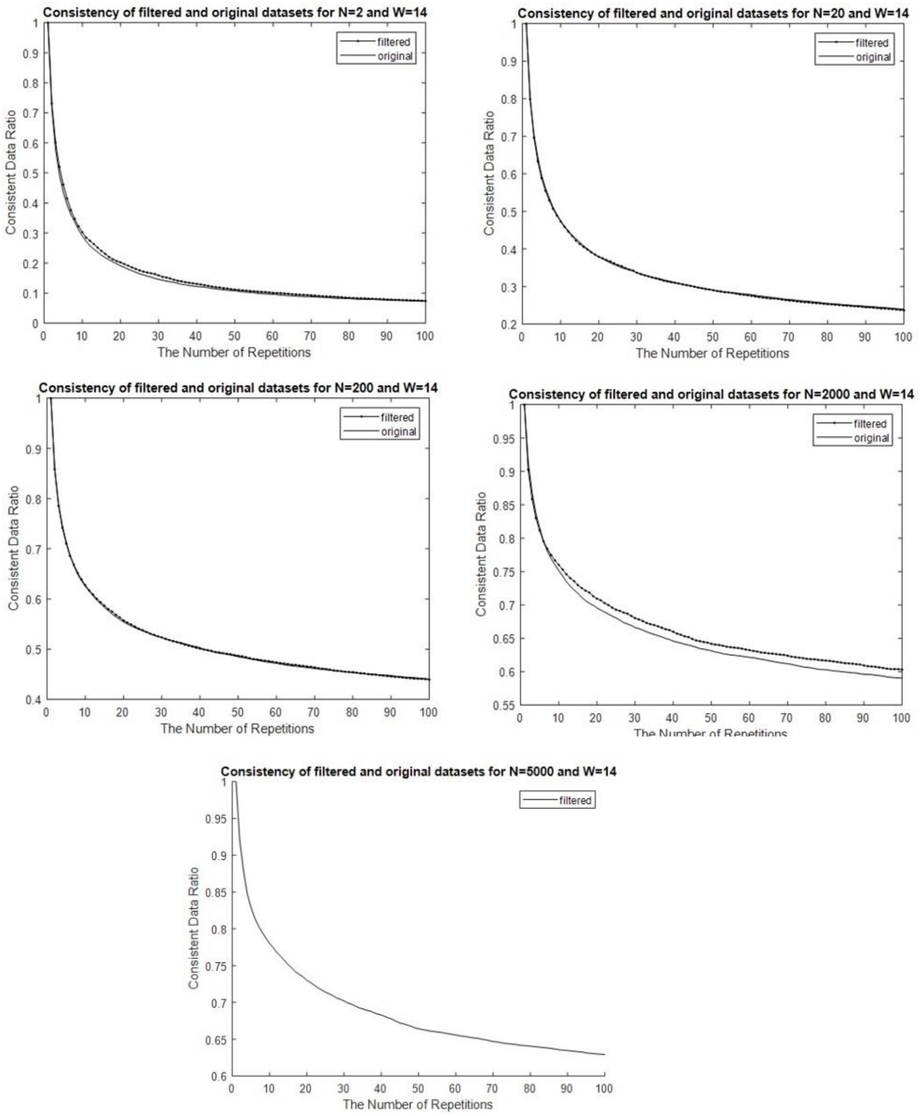


Figure 5.8. COC analysis for window-size=14 on different word-vector dimensions with original and filtered datasets

In Figure 5.8, for *N*=2, in the first repetitions, the increase in COC is evident. Dimensions *N*=20 and *N*=200 are not too much effective for this window-size to increase COC. Surprisingly, the increase in COC, for the highest dimension size *N*=5000 is very little.

In Figure 5.9, for all word-vector dimensions, results are given when the filtered and original datasets are prepared for window-size = 18.

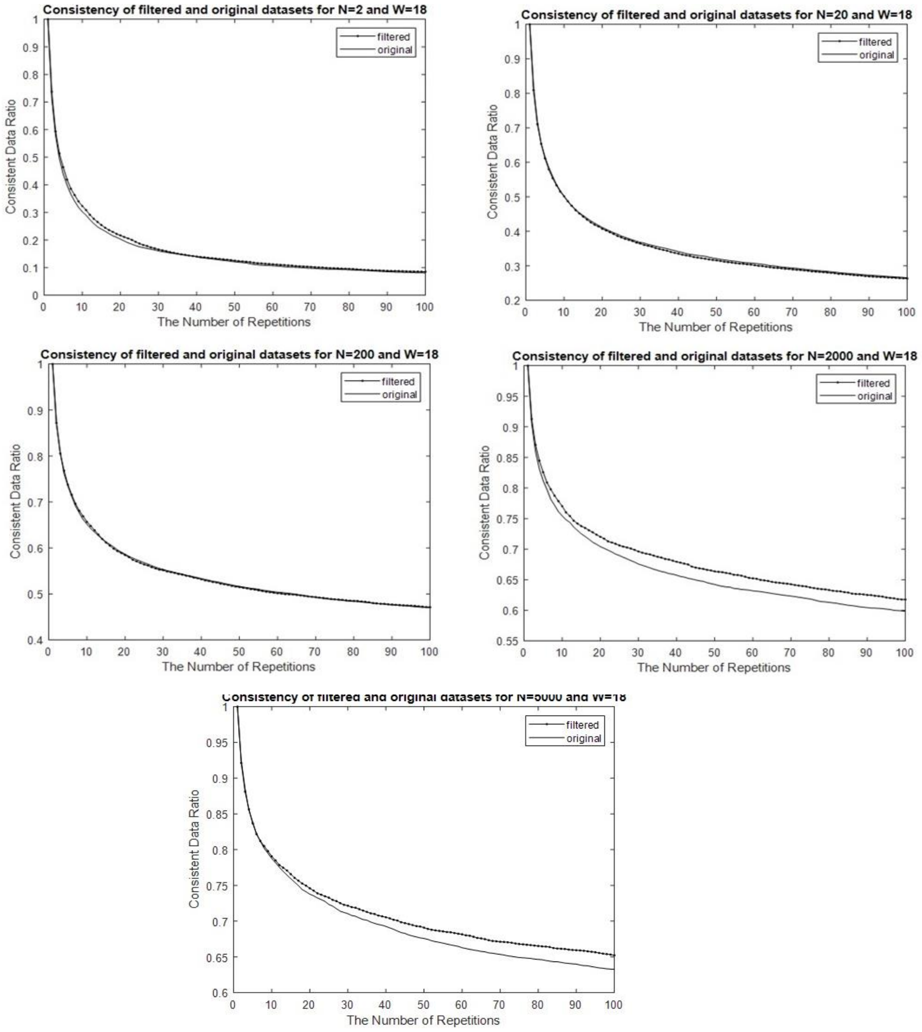
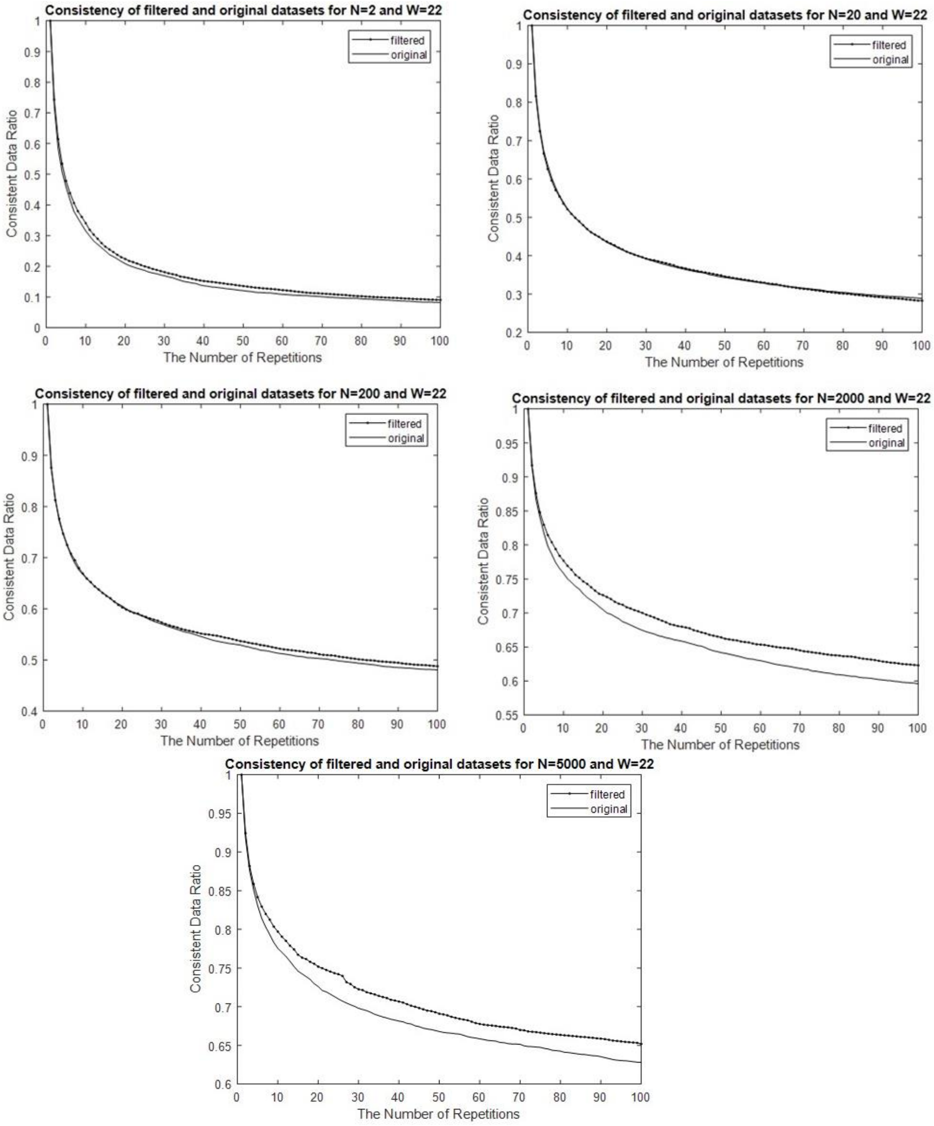


Figure 5.9. COC analysis for window-size=18 on different word-vector dimensions with original and filtered datasets

In Figure 5.9, results are similar to the results given for window-size 14 except, for *N*=5000 the increase in COC is evident. In Figure 5.10, for all word-vector dimensions, results are given when the filtered and original datasets are prepared for window-size = 22.



Consistency of filtered and original datasets for N=2 and W=22

Consistency of filtered and original datasets for N=20 and W=22

Consistency of filtered and original datasets for N=200 and W=22

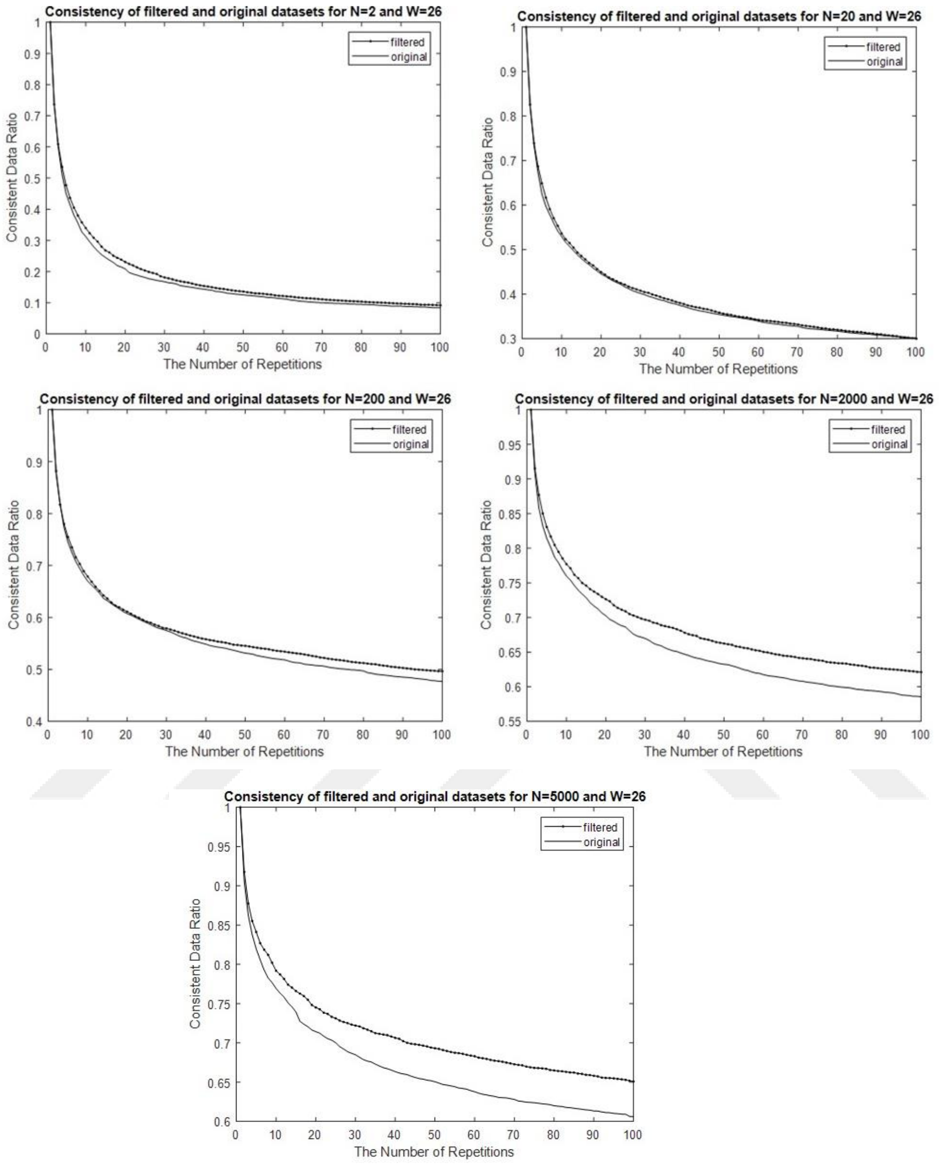
Consistency of filtered and original datasets for N=2000 and W=22

Consistency of filtered and original datasets for N=5000 and W=22

Figure 5.10 COC analysis for window-size=22 on different word-vector dimensions with original and filtered datasets

In Figure 5.10, for the first time, the lowest dimension *N*=2 shows distinctly the increase in COC for *w*=22. For *N*=20 and *w*=22 the filtered data effect is very little. On the other hand, in the last repetitions for *N*=200, the increase is evident and for *N*=2000 and *N*=5000 results are similar to the results given for window-size 18.

In Figure 5.12, for all word-vector dimensions, results are given when the filtered and original datasets are prepared for window-size = 26.



Consistency of filtered and original datasets for N=2 and W=26

Consistency of filtered and original datasets for N=20 and W=26

Consistency of filtered and original datasets for N=200 and W=26

Consistency of filtered and original datasets for N=2000 and W=26

Consistency of filtered and original datasets for N=5000 and W=26

Figure 5.11 COC analysis for window-size=26 on different word-vector dimensions with original and filtered datasets

In Figure 5.11, for *N*=200, *N*=2000 and *N*=5000 the highest gain for filtered datasets can be seen. But for these dimensions, it is clear that convergence slope is sharp.

5.3. Comparison Results with the Other Morphological Disambiguation Methods

When preceding experiments are considered, it can be claimed that it is advantageous to use fine-trained candidates in training cycles. Because of this, the LWQ system is trained with data lines which include the fine-trained candidates. For comparison, accuracy results of this study in morphological disambiguation and the other studies that exist in literature with the highest achievements for Turkish language and similar languages are given in Table 5.9.

Table 5.9. Accuracy values given in related studies for morphological disambiguation

Method	Accuracy (%)	Language
Orosz and Novák (2013)	95.81	Cz
Muller and Schutze (2015)	96.83	Hu
Hakkani-Tür et al. (2002)	95.07	Tr
Yuret and Ture (2006)	95.82	
Sak et al. (2007)	96.80	
Görgün and Yıldız (2011)	96.28	
Kutlu and Cicekli (2013)	93.40	
Yildiz et al. (2016)	96.28	
Shen et al. (2016)	97.24	
Dayanik et al. (2018)	96.86	
This Study	98.40	

Table 5.9 provides the accuracy values obtained in Turkish and morphologically similar languages like Hungarian and Czech. The values provided for Hungarian and Czech are selected from the highest values in the literature.

In early work in morphological disambiguation of Czech, Spoustová et al. (2007) have reported a 95.68% accuracy value by using the statistical methods combined with rule-based methods. Orosz and Novák (2013) present PurePOS which is a stochastic tagger and by using PurePOS they report 96.48% accuracy value. When they have used an input filter accuracy increases to 96.63%. Although this filter limits some input data, as we applied in this study, there does not exist any detailed analysis of its effect on accuracy. In Hungarian, most of the research uses the term “morphological tagging” instead of “morphological disambiguation”. One of these (Muller et al., 2013) provides the highest accuracy value by using the stochastic tagger named MarMot. They use second-order pruned CRFs and obtain a accuracy value of 96.57%. In their later study (Muller and Schutze, 2015), they have outperformed their previous study with a accuracy value of 96.83% by using the output of a morphological analyzer to reduce the lattice. This study also reports 94.48% accuracy value for Czech. In Tkachenko and Sirts's (2018) study, they have proposed a multi-class neural model which again uses MarMot as tagger and they achieve the highest accuracy value for Czech as 95.81%, while 84.12% for Hungarian.

Early studies on Turkish morphological disambiguation are mostly based on rule-based, statistical and hybrid models (Hakkani-Tür et al., 2002; Kutlu and Cicekli, 2013). The most prominent study (Hakkani-Tür et al., 2002) presents different n-gram models and trains these models with statistical information. The highest accuracy value reported in this study is 95.07% which is very near to the recent neural models’ achievements.

According to Table 5.6, it can be seen that the state of the art accuracy values for Turkish are near to similar languages, Hungarian and Czech. For example, Yildiz et al. (2016) reach 96.28%, and Dayanik et al. (2018) obtain 96.86%. Neural models are flexible and successful by presenting various embedding models and designs, but the necessity for large amounts of training data for the model and optimal parameter selection is a hard problem. One challenge in the morphological disambiguation task of Turkish is to obtain fully supervised training datasets. The most successful studies (Yuret and Ture, 2006; Sak et al., 2007; Yildiz et al., 2016) presented for Turkish use semi-supervised datasets without discussing the datasets' quality. According to Table 5.6, the highest value ever reported for Turkish (and also for similar languages) is reported by Shen et al. (2016) as 97.24%. Our study has outperformed this study by 98.40% accuracy value when the "intersection filter" is used in the dataset with training parameters w=14 and N=200. Theoretically, when the noise level given in Table 5.6 is considered, if morphological analyzer and user errors are corrected, this accuracy value can be increased to 99.3% by a simple calculation (98.40% + 56% x 1.60%.).

6. CONCLUSIONS

Morphological disambiguation is an important natural language processing (NLP) task for morphologically complex languages like Turkish. Although many studies present successful results by using various methods, dataset reliability and amount of data needed is a major concern. Using classification techniques in a vector-space for morphological disambiguation has not been studied in the literature, to our best knowledge.

Motivated by this, in this thesis, we have focused on morphological disambiguation task by using a new classification method named Learning Word-vector Quantization (LWQ). LWQ method is an adaptation of Learning Vector Quantization (LVQ) which is a well-known competitive machine learning algorithm. LVQ has been applied in many research fields. Although some of them are mainly on NLP problems, in our best knowledge, it has not been applied to any of the disambiguation tasks. Basically, LWQ locates the words as vectors in Euclidian space and optimizes the word-vectors with many training cycles. This optimization is provided by using a reward-or-penalize mechanism that is inspired by the original LVQ algorithm. In the study, the dataset is prepared from a standpoint that, “an ambiguous word in a text can be disambiguated by considering the neighbor words without ambiguity”. Datasets are prepared in a special format to separate the ambiguous words form unambiguous ones in two distinct vocabularies. Relationships between these vocabularies are used as equations which are used in approximating and departing the positions of the word-vectors.

In the study, many experiments are performed to investigate the effect of the parameters: window-size and vector-space dimension on accuracy and consistency of classification. Also, in an experiment, it has been identified that increasing the dataset size increases the system consistency and classification accuracy by contributing to the count of fully-trained word-vectors. On the other hand, tagging errors are defined and proposals are made for the improvement of the morphological analyzer. Finally, classification accuracy results are compared with similar studies in the literature presented for morphological disambiguation of Turkish.

LWQ method handles the morphological disambiguation problem as a classification problem in order to identify the correct parse candidate of an ambiguous word in a text. This method is directly dependent on the quality of the data which should be tagged correctly by an expert. Also, a larger tagged dataset provides more accurate classification rates by providing more equations for training. It is obvious that accuracy value can be increased in future studies by conducting a significant number of experiments. LWQ method is language-independent when the task is morphological disambiguation. We believe that it can be successful in less sparse languages other than complex languages with a low amount of data. On the other hand, it is applicable to word sense disambiguation (WSD) problem by taking advantage of the Euclidian space by mapping semantic information. It is thought that the LWQ method, introduced as a new method to the literature, will give a different perspective to natural language processing (NLP) studies with different adaptations.

REFERENCES

Altintas, E., Karsligil, E., and Coskun, V., 2005. The effect of windowing in word sense disambiguation. In *International Symposium on Computer and Information Sciences*(pp. 626-635). Springer, Berlin, Heidelberg.

Arslan, E, and Orhan, U., 2017. Using Graphs in Construction of a Lemmatization Model for Turkish, International Mediterranean Science and Engineering Congress, IMSEC.

Arslan, E., and Orhan, U., 2019. Identification of OOV words in Turkish texts. *Gaziosmanpaşa Bilimsel Araştırma Dergisi*, *8*(2), 35-48.

Bohnet, B., McDonald, R., Simoes, G., Andor, D., Pitler, E., and Maynez, J., 2018. Morphosyntactic tagging with a meta-bilstm model over context sensitive token encodings. *arXiv preprint arXiv:1805.08237*.

Brants, T., 2000. TnT: a statistical part-of-speech tagger. In *Proceedings of the sixth conference on Applied natural language processing* (pp. 224-231). Association for Computational Linguistics.

Chuan, Z., Xianliang, L., Mengshu, H., and Xu, Z., 2005. A LVQ-based neural network anti-spam email approach. *ACM SIGOPS Operating Systems Review*, *39*(1), 34-39.

Church, K. W., 1989. A stochastic parts program and noun phrase parser for unrestricted text. In *International Conference on Acoustics, Speech, and Signal Processing*, (pp. 695-698). IEEE.

Dayanık, E., Akyürek, E., and Yuret, D., 2018. Morphnet: A sequence-to-sequence model that combines morphological analysis and disambiguation”. *arXiv preprint arXiv:1805.07946*.

Daybelge, T., and Cicekli, I., 2007. A rule-based morphological disambiguator for Turkish. In *Proceedings of Recent Advances in Natural Language Processing* (pp. 145-149).

DeSieno, D., 1988. Adding a conscience to competitive learning. In *IEEE international conference on neural networks* (Vol. 1, No. 6, pp. 117-124). New York: Institute of Electrical and Electronics Engineers

Diaz-Galiano, M. C., Martin-Valdivia, M. T., Martinez-Santiago, F., and Ureña-López, L. A., 2004. Multi-word expressions recognition with the LVQ algorithm. *Proceedings of Methodologies and Evaluation of Multiword Unit in Real-world Applications, LREC, 2004*.

Ehsani, R., Alper, M. E., Eryigit, G., and Adali, E., 2012. Disambiguating main POS tags for Turkish. In *Proceedings of the 24th Conference on Computational Linguistics and Speech Processing (ROCLING 2012)* (pp. 202-213).

Ezeiza, N., Alegria, I., Arriola, J. M., Urizar, R., and Aduriz, I., 1998. Combining stochastic and rule-based methods for disambiguation in agglutinative languages. In *Proceedings of the 17th international conference on Computational linguistics-Volume 1* (pp. 380-384). Association for Computational Linguistics.

Görgün, O., and Yıldız, O. T., 2011. A novel approach to morphological disambiguation for Turkish. In Computer and Information Sciences II (pp. 77-83). London: Springer.

Görgün, O., and Yıldız, O. T., 2012. Using morphology in English-Turkish statistical machine translation. In 2012 20th Signal Processing and Communications Applications Conference (SIU) (pp. 1-4).

Gunawan, T. S., Kartiwi, M., and Ardzemi, N. H., 2017. Development of Language Identification using Line Spectral Frequencies and Learning Vector Quantization Networks. *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, *9*(3-7), 21-27.

Hakkani-Tür, D. Z., Oflazer, K., and Tür, G., 2002. Statistical morphological disambiguation for agglutinative languages. *Computers and the Humanities*, *36*(4), 381-410.

Hakkani-Tür, D. Z., Saraçlar, M., Tür, G., Oflazer, K., and Yuret, D., 2018. Morphological Disambiguation for Turkish. In *Turkish Natural Language Processing* (pp. 53-67). Springer, Cham.

Halácsy, P., Kornai, A., Oravecz, C., Vikto, T., and Varga, D., 2006. Using a morphological analyzer in high precision POS tagging of Hungarian.

Haldar, R., and Mishra, P. K. (2016). Learning Vector Quantization (LVQ) Neural Network Approach for Multilingual Speech Recognition. *Int. Res. J. Eng. Technol*, *3*, 2863-2869.

Heigold, G., Neumann, G., and van Genabith, J., 2017. An extensive empirical evaluation of character-based morphological tagging for 14 languages. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers* (pp. 505-513).

İlgen, B., Adalı, E., and Tantuğ, A. C., 2013. A comparative study to determine the effective window size of Turkish word sense disambiguation systems. In *Information Sciences and Systems 2013* (pp. 169-176). Springer, Cham.

Kiss, T., and Strunk, J., 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, *32*(4), 485-525.

Kohonen, T., 1984. Self-organization and associative memory, Springer Verlag, New York

Kohonen, T., 1990a. Improved versions of learning vector quantization. In 1990 IJCNN International Joint Conference on Neural Networks (pp. 545-550). IEEE.

Kohonen, T., 1990b. The self-organizing map. Proceedings of the IEEE, 78(9), 1464-1480.

Kohonen, T. 1990c. Statistical pattern recognition revisited. In Advanced neural computers (pp. 137-144). North-Holland.

Kohonen, T. 1992. New developments of learning vector quantization and the self-organizing map. In Symposium on Neural Networks; Alliances and Perspectives in Senri l992 (SYNAPSE'92) Japan. Osaka.

Kutlu, M., and Cicekli, I., 2013. A hybrid morphological disambiguation system for Turkish. In *Proceedings of the Sixth International Joint Conference on Natural Language Processing* (pp. 1230-1236).

Labeau, M., Löser, K., and Allauzen, A., 2015. Non-lexical neural architecture for fine-grained pos tagging. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 232–237.

Lafferty, J., McCallum, A., and Pereira, F. C., 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data.

Laki, L. J., and Orosz, G., 2014. An efficient language independent toolkit for complete morphological disambiguation. In LREC (pp. 1625-1630).

Ma, X, and Hovy, E, 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, volume 1, pages 1064–1074.

Mantysalo, J., Torkkola, K., & Kohonen, T. 1992. LVQ-based speech recognition with high-dimensional context vectors. In *Second International Conference on Spoken Language Processing*.

Martín-Valdivia, M. T., Ureña-López, L. A., and García-Vega, M. (2007). The learning vector quantization algorithm applied to automatic text classification tasks. *Neural Networks*, *20*(6), 748-756.

Muller, T., Schmid, H., and Schutze, H., 2013, Efficient higher-order CRFs for morphological tagging. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing* (pp. 322-332).

Muller, T., and Schutze, H., 2015. Robust morphological tagging with word representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies* (pp. 526-536).

Nguyen, D. Q., Pham, D. D., and Pham, S. B., 2016. A robust transformation-based learning approach using ripple down rules for part-of-speech tagging. AI Communications, 29(3), 409-422.

Oflazer, K., 1994. Two-level description of Turkish morphology. Literary and linguistic computing 9.2, 137-148.

Oflazer, K., and Tür, G., 1996. Combining hand-crafted rules and unsupervised learning in constraint-based morphological disambiguation. *arXiv preprint cmp-lg/9604001*.

Oflazer, K., and Tür, G., 1997. Morphological disambiguation by voting constraints. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics* (pp. 222-229). Association for Computational Linguistics.

Orosz, G., and Novák, A., 2013. PurePos 2.0: a hybrid tool for morphological disambiguation. In *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013* (pp. 539-545).

Öztemel, E., 1992. Integrating expert systems and neural networks for intelligent on-line statistical process control.

Öztemel, E., 2012. Yapay sinir ağları. Papatya Yayıncılık Eğitim, İstanbul, 232s.

Pennington, J., Socher, R., and Manning, C., 2014. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

Pilevar, M. T., Feili, H., and Soltani, M. (2009, September). Classification of Persian textual documents using learning vector quantization. In *2009 International Conference on Natural Language Processing and Knowledge Engineering* (pp. 1-6). IEEE.

Sak, H., Güngör, T., and Saraçlar, M., 2007. Morphological disambiguation of Turkish text with perceptron algorithm. In *International Conference on*

Intelligent Text Processing and Computational Linguistics (pp. 107-118). Springer, Berlin, Heidelberg.

Sezer, T., 2017. TS Corpus Project: An online Turkish Dictionary and TS DIY Corpus. European Journal of Language and Literature, 9(1), 18-24.

Shen, Q., Clothiaux, D., Tagtow, E., Littell, P., and Dyer, C., 2016. The role of context in neural morphological disambiguation. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (pp. 181-191).

Silfverberg, M., Ruokolainen, T., Lindén, K., and Kurimo, M., 2016. FinnPos: an open-source morphological tagging and lemmatization toolkit for Finnish. *Language Resources and Evaluation*, *50*(4), 863-878.

Straková, J., Straka, M., and Hajič, J., 2014. Open-source tools for morphology, lemmatization, POS tagging and named entity recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations* (pp. 13-18).

Tahiroğlu, B.T., 2014. Türkçe Çevrim İçi Haber Metinlerinde Yeni Sözlerin (Neolojizm) Otomatik Çıkarımı. Derlem Dilbilim Uygulamaları, Özkan, B., Tahiroğlu, B. Tahir ve Özkan Ayşe Eda (Ed.), Karahan Kitabevi Yayınları, Adana, ss.1-22.

Tkachenko, A., and Sirts, K., 2018. Modeling composite labels for neural morphological tagging. *arXiv preprint arXiv:1810.08815*.

TLA,http://www.tdk.gov.tr/index.php?option=com_content&id=198:Kisaltmalar, Access Date:2018

Umer, M. F., and Khiyal, M. S. H. (2007). Classification of textual documents using learning vector quantization. *Information Technology Journal*, *6*(1), 154-159.

Visa, A. J., Toivanen, J., Back, B., and Vanharanta, H. (2000, April). Toward text understanding: Classification of text documents by word map. In *Data*

Mining and Knowledge Discovery: Theory, Tools, and Technology II (Vol. 4057, pp. 299-305). International Society for Optics and Photonics.

Viterbi, A., 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm, IEEE transactions on Information Theory, 13(2), 260-269.

Yıldız, O. T., Avar, B., and Ercan, G, 2019. An Open, Extendible, and Fast Turkish Morphological Analyzer.

Yildiz, E., Tirkaz, C., Sahin, H. B., Eren, M. T., and Sonmez, O. O., 2016. A morphology-aware network for morphological disambiguation. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Yuret, D., and Türe, F., 2006. Learning morphological disambiguation rules for Turkish. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics* (pp. 328-334). Association for Computational Linguistics.

CURRICULUM VITAE

Enis ARSLAN was born in Zonguldak, in 1976. He received a Bachelor's degree in Computer Engineering from Yıldız Technical University (YTÜ) in 2000, M.Sc. degree in Computer Engineering from Doğuş University in 2012. He also has an MBA degree from Beykent University. Following the graduation from university as an engineer, he had worked on several job titles as System Analyst, Business Analyst, Senior Database Operation Engineer and Project Manager in different sectors, especially Medical and Telecommunications. During the Ph.D. study at Çukurova University, he had worked as Project Assistant for the Tübitak Project with the number 215E256.