



**INTER-FACULTY MASTER PROGRAM on  
COMPLEX SYSTEMS and NETWORKS**

**SCHOOL of MATHEMATICS**

**SCHOOL of BIOLOGY**

**SCHOOL of GEOLOGY**

**SCHOOL of ECONOMICS**

**ARISTOTLE UNIVERSITY of THESSALONIKI**



**Master Thesis**

Machine learning algorithms for big data

Αλγόριθμοι Μάθησης για μεγάλης κλίμακας δεδομένα

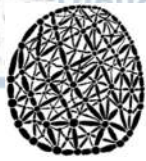
**Elisavet Tsolakidou**

**SUPERVISOR:** Charalampos Bratsas, Laboratory Teaching Staff, AUTH

**CO-SUPERVISOR:** Nikolaos Farmakis, Associate Professor, AUTH

**Thessaloniki, December 2018**





**ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ στα  
ΠΟΛΥΠΛΟΚΑ ΣΥΣΤΗΜΑΤΑ και ΔΙΚΤΥΑ**



**ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ**

**ΤΜΗΜΑ ΒΙΟΛΟΓΙΑΣ**

**ΤΜΗΜΑ ΓΕΩΛΟΓΙΑΣ**

**ΤΜΗΜΑ ΟΙΚΟΝΟΜΙΚΩΝ ΕΠΙΣΤΗΜΩΝ**

**ΑΡΙΣΤΟΤΕΛΕΙΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΟΝΙΚΗΣ**

**ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**Αλγόριθμοι Μάθησης για μεγάλης κλίμακας δεδομένα**

**Ελισάβετ Τσολακίδου**

**ΕΠΙΒΛΕΠΩΝ:** Χαράλαμπος Μπράτσας, ΕΔΙΠ Α.Π.Θ.

**ΣΥΝΕΠΙΒΛΕΠΩΝ:** Νικόλαος Φαρμάκης, Α. Καθηγητής Α.Π.Θ.

Εγκρίθηκε από την Τριμελή Εξεταστική Επιτροπή την 21η Δεκεμβρίου 2018.

.....	.....	.....
N. Φαρμάκης	X. Μπράτσας	I. Αντωνίου
A. Καθηγητής	ΕΔΙΠ Α.Π.Θ.	Καθηγητής Α.Π.Θ
Α.Π.Θ..		

**Θεσσαλονίκη , Δεκέμβριος 2018**



Ελισάβετ Τσολακίδου

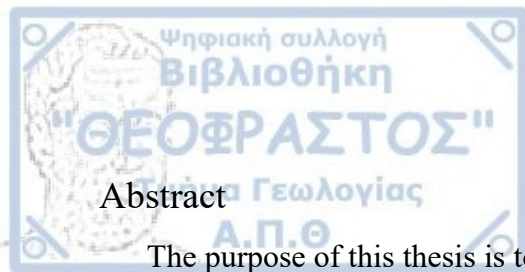
Διπλωματούχος Μαθηματικός Α.Π.Θ.

Copyright © Ελισάβετ Τσολακίδου, 2018

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευτεί ότι εκφράζουν τις επίσημες θέσεις του Α.Π.Θ.



## Abstract

The purpose of this thesis is to present both a comprehensive study and a practical example of methods and tools used for word and document embeddings. Embeddings or vector representations are the necessary first step before natural language data are fed into any neural network for processing. Having vectors whose parameters capture real world properties of the corresponding words or documents has been known to be pivotal for the success of natural language processing tasks such as classification, summarization and translation amongst others. Following the detailed presentation of the most popular methods; experiments and their implementations are conducted using Python programming language in the Greek language and their results are discussed.

## Keywords

Machine Learning, Deep Learning, Word Embeddings, Document Embeddings, Python, Gensim, Greek Language





## Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι να παρουσιαστούν οι μέθοδοι και τα εργαλεία που χρησιμοποιούνται για τη διανυσματική αναπαράσταση λέξεων και εγγράφων. Η διανυσματική αναπαράσταση των λέξεων αποτελεί απαραίτητο πρώτο βήμα για την τροφοδότηση δεδομένων φυσικής γλώσσας σε οποιοδήποτε νευρωνικό δίκτυο με σκοπό την επεξεργασία των λέξεων και την εξαγωγή μοντέλων. Η αναπαράσταση των λέξεων ως διανύσματα των οποίων οι παράμετροι σκιαγραφούν την πληθώρα των ιδιοτήτων τους έχει παρατηρηθεί ότι είναι καθοριστική για την επιτυχία διεργασιών επεξεργασίας φυσικής γλώσσας όπως είναι η κατηγοριοποίηση (classification), περίληψη (summarization) και μετάφραση (translation) κειμένων, μεταξύ άλλων. Αρχικά παρουσιάζονται οι πιο γνωστές τεχνικές που χρησιμοποιούνται για την αναπαράσταση λέξεων και εγγράφων και στην συνέχεια μέσω της γλώσσας προγραμματισμού Python μελετάται η αναπαράσταση λέξεων και εγγράφων στην ελληνική γλώσσα.

## Λέξεις-Κλειδιά

Μηχανική μάθηση, διανυσματική αναπαράσταση λέξεων, διανυσματική αναπαράσταση εγγράφων, Python, Gensim, Ελληνική γλώσσα

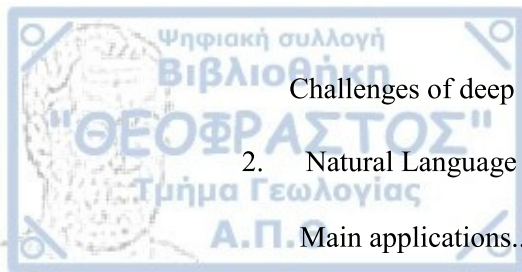






## Table of Contents

Abstract .....	3
Keywords .....	3
Περίληψη .....	5
Λέξεις-Κλειδιά .....	5
Table of Contents .....	6
Synopsis .....	9
Acknowledgements.....	13
1. Machine Learning & Deep Learning Background.....	15
Machine learning.....	15
A brief history of machine learning .....	16
Key concepts & Types of learning.....	16
Machine learning tasks categorization .....	18
Machine learning popular applications .....	19
Machine learning challenges and limitations.....	19
Machine learning algorithms .....	20
Deep learning .....	21
History.....	22
Evolution of architectures .....	22
Artificial neural networks .....	22
Deep learning networks.....	23
Autoencoders.....	23
Convolutional deep neural networks.....	24
Recurrent neural networks .....	25
Deep Belief Networks .....	25
Main applications of deep learning networks .....	25

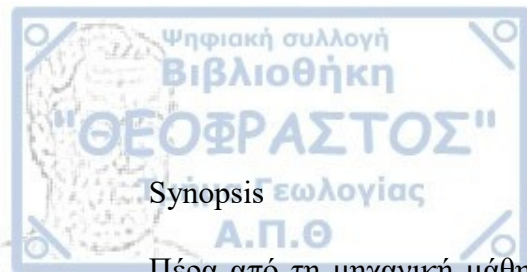


Challenges of deep learning networks .....	26
2. Natural Language Processing – Background .....	29
Main applications.....	29
Text & Document classification.....	29
Machine translation.....	30
Native Language Identification.....	31
Text similarity.....	32
Foundation/ Key concepts.....	33
Preprocessing and parsing.....	33
Word senses .....	34
Word embeddings .....	34
Word representations in vector space .....	35
Information retrieval with TF-IDF.....	40
Applications of TF-IDF .....	42
Additional variations of TF-IDF .....	43
BM 25 .....	44
3. Word embedding algorithms.....	47
Word2vec .....	47
Motivation / Rationale for word2vec .....	48
Word2vec Operation .....	48
Architecture.....	49
Word2Vec example applications .....	52
Extensions/ Variations of word2vec .....	54
GloVe .....	55
FastText.....	57
WordRank .....	59
4. Document embedding algorithms .....	61



Doc2vec.....	61
Paragraph Vector: A Distributed Memory model (PV-DM) .....	62
Paragraph Vector without word ordering: Distributed bag of words (PV-DBOW).....	63
5. Python toolkits and libraries for natural language processing .....	65
Natural Language Toolkit (NLTK).....	65
Pattern for Python .....	67
LibShortText .....	68
Gensim .....	69
Core concepts in Gensim .....	70
6. A practical application of Gensim doc2vec for similarity estimation between Wikipedia articles.....	73
Application goals and Problem definition .....	73
Stakeholders .....	73
Methodology .....	73
Requirements.....	74
Design .....	74
Implementation .....	76
Obtaining the source documents .....	76
Applying word2vec in the articles of Greek Wikipedia/ Word analogy queries.....	77
Modeling with doc2vec and performing document similarity queries .....	79
Results-Evaluation of models .....	81
Results-Evaluation of Word2vec algorithm.....	81
Results of Doc2vec- Document similarity queries.....	83
7. Conclusion.....	87
References .....	89





Πέρα από τη μηχανική μάθηση, η οποία έχει εδραιωθεί και χρησιμοποιείται σε πάρα πολλούς τομείς, τα τελευταία χρόνια το ερευνητικό ενδιαφέρον έχει επικεντρωθεί στις τεχνικές βαθιάς μάθησης (deep learning) οι οποίες υπόσχονται λύσεις σε προβλήματα μάθησης χωρίς επίβλεψη κατά κύριο λόγο, ανάλογες με αυτές που παρέχει ο ανθρώπινος εγκέφαλος. Παρατηρείται ακόμη, ο συνδυασμός τεχνικών μηχανικής μάθησης και βαθιάς μάθησης.

Η παρούσα εργασία επικεντρώνεται σε τεχνικές μάθησης που αφορούν σε προβλήματα επεξεργασίας φυσικής γλώσσας και πιο συγκεκριμένα στη διανυσματική αναπαράσταση λέξεων και εγγράφων. Η διανυσματική αναπαράσταση λέξεων και εγγράφων θέτει νέες βάσεις για την πραγματοποίηση τεχνικών ανάλυσης συναισθημάτων, όπου γίνεται αντιληπτός ο θετικός ή αρνητικός χαρακτήρας ενός κειμένου, μηχανικής μετάφρασης όπου είναι δυνατή η αυτόματη μετάφραση κειμένου, της αναγνώρισης φωνής, κατανόησης κειμένου φυσικής γλώσσας και παραγωγής φυσικής γλώσσας.

Μια λέξη (ή ένα έγγραφο) μπορεί να αναπαρασταθεί με τον πιο απλό τρόπο από ένα διάνυσμα με βάση την παρουσία της σε ένα κείμενο (ή την παρουσία του σε μια συλλογή εγγράφων αντίστοιχα). Για παράδειγμα αν το κείμενο αποτελείται μόνο από την πρόταση “Η μηχανική μάθηση αποσκοπεί στην πρόβλεψη.” κάθε λέξη θα μπορούσε να αναπαρασταθεί ως εξής:

$H = (1, 0, 0, 0, 0, 0)$

μηχανική =  $(0, 1, 0, 0, 0, 0)$

μάθηση =  $(0, 0, 1, 0, 0, 0)$

αποσκοπεί =  $(0, 0, 0, 1, 0, 0)$

στην =  $(0, 0, 0, 0, 1, 0)$

πρόβλεψη =  $(0, 0, 0, 0, 0, 1)$

όπου η ύπαρξη της λέξης στην πρόταση υποδηλώνεται από την ύπαρξη μονάδας στο διάνυσμα και η θέση της μονάδας είναι αντίστοιχη της θέσης της λέξης στην πρόταση. Η αναπαράσταση αυτή της λέξης ωστόσο παρότι δίνει στον ερευνητή την δυνατότητα

να την εισάγει στον υπολογιστή και εν συνεχεία να την χρησιμοποιήσει σε διάφορους αλγορίθμους, υστερεί στο να αποτυπώσει τις σημασιολογικές σχέσεις που υπάρχουν μεταξύ των λέξεων και που ο ανθρώπινος εγκέφαλος μπορεί εύκολα να αναγνωρίσει.

Για τον λόγο αυτό, διάφοροι τρόποι έχουν προταθεί στην προσπάθεια για βελτίωση της αναπαράστασης των λέξεων και κατ' επέκταση των εγγράφων. Οι ανωτέρω αλγόριθμοι χωρίζονται σε δύο βασικές κατηγορίες:

1. Αλγόριθμοι συχνότητας που αφορούν την συχνότητα με την οποία εντοπίζεται μια λέξη με τις γειτονικές της σε ένα κείμενο.
2. Αλγόριθμοι πρόβλεψης που προσπαθούν να προβλέψουν μία λέξη γνωρίζοντας τις γειτονικές λέξεις.

Ένας αλγόριθμος που αντιπροσωπεύει την πρώτη κατηγορία είναι ο Tf-Idf (Term Frequency-Inverse Document Frequency) ο οποίος χρησιμοποιεί έναν συνδυασμό της συχνότητας με την οποία μία λέξη εμφανίζεται σε ένα κείμενο και της συχνότητας με την οποία η λέξη εμφανίζεται σε ένα σύνολο εγγράφων και εκτιμά πόσο σημαντικές είναι οι λέξεις σε κάθε κείμενο. Έτσι πολύ συχνές λέξεις σε ένα κείμενο που εμφανίζουν εξίσου μεγάλη συχνότητα σε ένα σύνολο εγγράφων αποκλείονται μέσω του αλγορίθμου όπως για παράδειγμα τα άρθρα καθώς λέξεις οι οποίες εμφανίζονται σε μεγάλη συχνότητα σε ένα κείμενο αλλά με μικρή συχνότητα σε ένα σύνολο εγγράφων εκτιμούνται ως σημαντικές και λαμβάνουν μεγαλύτερη τιμή.

Αξίζει να σημειωθεί ότι έχουν προταθεί διάφορες τροποποιήσεις του συγκεκριμένου αλγορίθμου, οι οποίες παρουσιάζονται αναλυτικά στο 2ο κεφάλαιο της εργασίας αυτής.

Η δεύτερη κατηγορία αλγορίθμων αποτέλεσε μεγάλη εξέλιξη της πρώτης και βασίζεται στην παραδοχή ότι λέξεις που εμφανίζονται κοντά βρίσκονται κοντά σημασιολογικά. Ο αλγόριθμος που εισήγαγε την κατηγορία αυτή ονομάζεται word2vec και χρησιμοποιεί ένα νευρωνικό δίκτυο για την παραγωγή διανυσμάτων λέξεων. Ο αλγόριθμος αυτός αποτελείται από δύο τεχνικές:

- *bag of words*, όπου υπολογίζεται η πιθανότητα μιας λέξης δοσμένων των γειτονικών της λέξεων



- skip gram, όπου υπολογίζονται οι πιθανότητες των γειτονικών λέξεων δοσμένης μία λέξης.

Έτσι, οι διανυσματικές παραστάσεις των λέξεων μας επιτρέπουν να συγκρίνουμε διάφορες λέξεις σημασιολογικά. Για παράδειγμα, η εύρεση αναλογιών ανάμεσα σε λέξεις είναι μία λειτουργία που παρουσιάζει ιδιαίτερο ενδιαφέρον. Ο τρόπος με τον οποίο επιτυγχάνεται, περιλαμβάνει τον υπολογισμό αλγεβρικών πράξεων ανάμεσα στα διανύσματα των λέξεων. Με δεδομένες τρεις λέξεις και την εκτέλεση αλγεβρικών πράξεων ανάμεσα στα διανύσματα τους παράγεται ένα νέο διάνυσμα και με βάση αυτό ο αλγόριθμος υπολογίζει την πιο κοντινή λέξη. Ακόμη, μέσα από τη σύγκριση των διανυσμάτων, υπάρχει η δυνατότητα να εντοπιστεί η λιγότερο όμοια λέξη δεδομένης μιας λίστας λέξεων. Επέκταση των ανωτέρω αποτελούν οι αλγόριθμοι για διανυσματική αναπαράσταση εγγράφων. Με την βοήθεια των αλγορίθμων αυτών μπορούμε πλέον να υπολογίσουμε την ομοιότητα εγγράφων καθώς και να προχωρήσουμε σε κατηγοριοποίηση αυτών με βάση τη σημασιολογική αναπαράστασή τους.

Στα κεφάλαια που ακολουθούν, αρχικά γίνεται μια παρουσίαση της ιστορίας και των βασικών τεχνικών μηχανικής μάθησης και βαθιάς μάθησης και παρατίθεται το υπόβαθρο που απαιτείται να έχει ο αναγνώστης για τη συνέχεια (Κεφάλαιο 1 & 2). Κατόπιν παρουσιάζονται οι σύγχρονες μέθοδοι για τον υπολογισμό διανυσματικών αναπαραστάσεων, αρχικά λέξεων και κατόπιν ολόκληρων εγγράφων (Κεφάλαιο 3 & 4). Στη συνέχεια, γίνεται μια εκτενής αναφορά στη γλώσσα προγραμματισμού Python η οποία περιέχει υλοποιήσεις για όλες αυτές τις μεθόδους, με έμφαση στο πακέτο Gensim που συγκεντρώνει τις περισσότερες (Κεφάλαιο 5). Τέλος, δίνεται ένα πρακτικό παράδειγμα εφαρμογής των μεθόδων στο κείμενο της Ελληνικής ελεύθερης εγκυκλοπαίδειας Wikipedia, αρχικά για την εύρεση διανυσμάτων λέξεων και στη συνέχεια για τη σύγκριση άρθρων μεταξύ τους (Κεφάλαιο 6).







### Acknowledgements

I would like to express my gratitude to my supervisor Charalampos Bratsas, co-supervisor Nikolaos Farmakis and professor Ioannis Antoniou for the useful comments, remarks and engagement through the learning process of my master's degree and this master thesis. Finally, I must express my very profound gratitude to my family, friends and classmates for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.



## 1. Machine Learning & Deep Learning Background

### Machine learning

What if a problem could be solved not by listing the required commands and procedures but by instructing a computer to learn how to solve it? Is this even feasible and if so, is this artificial intelligence? These are some of the questions posed when one attempts to define machine learning.

As the term suggests, this scientific field is related to machines (i.e., computers) progressively improving their performance by learning without being explicitly told how (i.e. programmed) but by observing shifts in their performance (i.e., by gaining experience). This very broad description has been defined more formally by Tom M. Mitchell, a professor at the Carnegie Mellon University, in his book entitled “Machine Learning” where he stated the following:

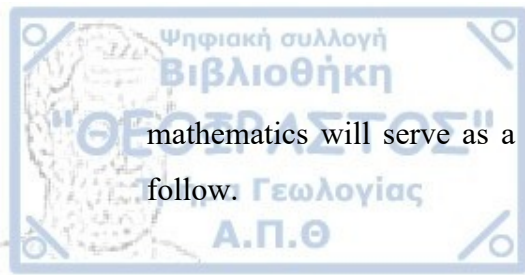
“A computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$ .”

This formal definition cites the key components/ concepts of machine learning which are:

- There is a set of tasks  $T$  that the computer/ software must perform.
- The outcome of the software execution can be evaluated (measurable performance –  $P$ ). E.g. if the software reaches a decision, it could be right or wrong.
- There is a dataset with potential for teaching/ learning ( $E$ ).

In a practical example, a machine learning application could be requested to identify if a particular animal is found inside a set of pictures ( $T$ ). In contrast to a more procedural approach where the software will be given the features of the animal as input, in a machine learning approach, the input ( $E$ ) could be a set of pictures where the animal is found. After the software makes each decision (found/ not found) it can be informed of its success rate ( $P$ ) and thereby use this information in future decisions.

The following sections are an overview of the history and the evolution of machine learning, its practical applications, the types of problems it can solve and its relation to deep learning. This high-level introduction of basic concepts, without



mathematics will serve as a theoretical basis for the more technical sections that will follow.

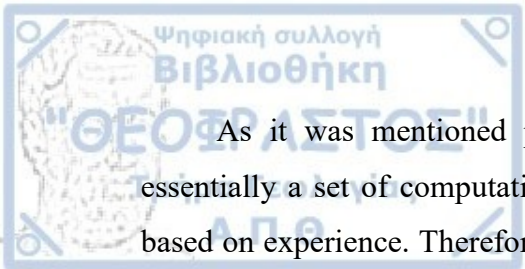
### A brief history of machine learning

The term machine learning was coined in 1959 by Arthur Samuel, a pioneer in the field of computer gaming and artificial intelligence in his paper entitled “Some studies in machine learning using the game of checkers” [33] where he discussed an application of such techniques in the context of a popular board game. In this seminal paper, Samuel essentially attempts to get the computer to improve in checkers by learning in a way that is similar to the way humans learn (i.e. not only from the data at hand). This constitutes his work a more literal machine learning application in which a machine learns using both the rules of the game and the feedback on its performance, much like we do in real life. Indeed, in its first steps and up until machine learning was nearly abandoned in favor of expert systems and other relevant tools, it was thought of as being very closely related to the computer science field of artificial intelligence.

Machine learning resurfaced in the 1990s when its goals shifted from the more or less theoretical pursuit of artificial intelligence to providing solutions for practical problems whose complexity did not allow for satisfactory procedural/ conventional solutions. A decisive contributing factor for this paradigm shift was the abundance of digital data, easily circulated via the newly born Internet. In the new context, machine learning is more closely related to data mining than artificial intelligence and borrows concepts from statistics and probability theory with prediction being a key aspect. The main difference between the two fields lies in the kind of knowledge that is utilized. In machine learning, the goal is to utilize previously acquired knowledge for problem solving where as in data mining the goal is to discover new knowledge hidden in the data.

### Key concepts & Types of learning

In this section, the broadest categorization of machine learning tasks is presented, focusing on the way the actual learning takes place. Before presenting the various learning modes, we will take a look at some basic terminology that will be used throughout this thesis.



As it was mentioned previously machine learning in each current format is essentially a set of computational methods that solve problems/ improve performance based on experience. Therefore, we are dealing with data driven tasks and use statistics, probability theory and optimization to learn from them. In a machine learning algorithm/ application, an **example** is an instance of the available data, also referred to as an item. This **item** has a set of features/ attributes which are of interest to the given problem. A vector format is often selected for the representation of these features as it is suitable for high speed computations. Items are also assigned **labels** which are either indicative of categories or of actual data values.

**Labeled data** (i.e. data whose value or categorization is known) are typically used for training whereas data whose labels are known but hidden are used as test data to check the machine performance. The set of training data may become available to the learner in **batches** or **one at a time** (on-line). Queries on the data labels (output) can be either **active** (the learner can explicit request the label for a given point) or **passive** where the learner receives a set of labeled points. In all cases, the learner is required to provide predictions for point labels. The fundamental distinction between the types of learning is based on the availability/ use of labeled or unlabeled data.

In **supervised learning**, much like in a classroom, there is a teacher entity that feeds the computer with sample inputs and their desired outputs. The computer needs to come up with a general rule that maps inputs to outputs as fast and as accurately as possible. In this mode of learning labeled data are utilized to come up with predictions on unseen points.

In **unsupervised learning**, no labeled data is available, i.e., the learner cannot utilize examples of correct outputs to infer rules. Therefore, the machine is typically requested to uncover (hidden) relations in its input data or discover patterns in the input data structure.

**Semi-supervised learning** falls between the two types mentioned above. In this scenario, both labeled and unlabeled data are available and are used as a basis for predictions on unseen points. The volume of unlabeled data is typically significantly larger. **Active learning** is a special case of semi supervised learning in which the labeled data used for training can be selected by the machine itself. By optimizing the selection of objects whose labels are unveiled the machine can acquire more usable labels while sticking to a prespecified budget.

Lastly, **reinforcement learning** is based on the notions of rewards and punishments provided as feedback by the environment in response to machine actions. Unlike supervised learning which has a set of current input/ output pairs and rules to be discovered, the machine is required to discover new knowledge (explore the environment) while exploiting existing knowledge.

### Machine learning tasks categorization

In broad terms, machine learning tasks or problems suitable for machine learning solutions can be classified in the following categories:

**Classification:** input items are thought of as belonging to a specific set of classes/ categories (often but not always to two classes) and the machine learning application is asked to assign a category to each item. Each item can belong to a single class or multiple classes. A popular example is classifying an e-mail as spam or not spam.

**Regression:** in this type of tasks the algorithm needs to predict the actual values of items as accurately as possible. A real-life example is predicting the values of stocks based on past data.

**Ranking:** in such applications items need to be order based on a given criterion. For example, on an e-commerce site, related products need to be presented to the visitor browsing a particular product in an order that maximizes the likelihood of additional purchases.

**Clustering:** in this category of tasks the machine is requested to partition data into similar or homogenous regions based on prespecified criteria. Typically, this involves large data sets that would be very difficult to manipulate without some form of grouping.

**Dimensionality reduction:** in such applications, complex data are reduced to simpler representations by preserving only some of their properties (dimensions). This lower-dimensional space is easier to navigate.

## Machine learning popular applications

In this section, popular applications of machine learning are presented in general categories. The applications related to natural language processing will be presented in more detail following the chapter on deep learning.

**Text:** this category refers to applications related to the analysis of text-based input. Popular examples include document classification and spam detection.

**Language:** this typically involves natural language processing tasks in which a computer is required to analyze natural language inputs. Early and popular examples include machine translation, parsing and morphological analysis.

**Speech:** this class of applications is related to parsing speech inputs. Popular examples include speech recognition and automatic synthesis.

**Image:** machine learning applications have long been used for image tagging/ annotation, face recognition/ pattern matching, character recognition (in print or handwritten) and others.

**Gaming:** this can refer both to improving gaming tactics/ performance in classical board games but also to more advanced computer games.

**Automation:** this refers generally to unassisted control of machines typically in a dynamic environment. A popular example is self-driving cars that are becoming more and more independent, but it could also refer to the control of unmanned aerial vehicles such as drones or robots.

Other types of more specialized application focus on inferring knowledge for medical diagnoses or assessing the probability of intrusion in a network.

## Machine learning challenges and limitations

Most limitations related to machine learning are primarily linked with the quality and quantity of training data. When the volume of data available for training (and therefore also testing) is small, it may be difficult or even impossible for the algorithm to find patterns and infer a relationship between the input and output. In this direction,



incorrectly posed goals/ questions or unsuitable algorithms may also prove detrimental to the accuracy of the solution.

Apart from lack of data, another very serious issue that degrades the outcome is bias inherent in the data that the algorithm will almost certainly pick up on and perpetuate to unseen cases output. In order to get accurate output predictions for specific inputs, these have to be present in the training data. In a highly publicized example of image recognition software failure, Google's image tagging software would tag black people as gorillas. This was reportedly due to the lack of good quality training photos with people of color and it was not the only mix between species that would occur. What is interesting is the way the company opted to "fix" the issue which was by removing all pictures of gorillas from the training set, thus preferring no categorization to offensive mis-categorization [38].

Accidents involving self-driving vehicles are also a very unfortunate example of failure with devastating consequences.

### Machine learning algorithms

The number of machine learning algorithms available is so large that it does not make much sense to present specific algorithms. Instead, an overview of the algorithm categories will be presented where categorization is based on the way the algorithm tackles the problem and represents data. It must be noted that an algorithm may belong to more than one categories and that not all identified categories will be included in this thesis.

**Statistics-based algorithms/ Regression Algorithms:** algorithms of this category seek to model the relationship between input and output using statistical tools such as regression (linear/ stepwise/ logistic).

**Decision Tree Algorithms:** this type of algorithms construct a decision model that predicts the value of the output variable based on the actual values/ attributes of multiple inputs. The input data is organized in the branches of the tree and the output values are represented by the leaves. Speed and accuracy are often observed in this category of methods, thus making them a popular choice.

**Bayesian algorithms:** algorithms in this category apply Bayes' theorem to infer the probabilities of output values. Data is typically mapped using a direct acyclic graph



in which the presence of an edge connecting two nodes represents a conditional dependency.

**Clustering algorithms:** clustering based machine learning algorithms, like the category of problems with the same name, aim at grouping the data in sets with the maximum features in common.

**Rule-based learning algorithms:** instead of identifying a model that will make an accurate prediction of the output for a given input, algorithms in this category aim at constructing a set of rules that describe the entire knowledge of the system. A popular subgroup in this category are association rule learning algorithms.

**Artificial neural networks:** artificial neural networks are data structures inspired by neurons found in human and animal brains and algorithms using such structures attempt to solve problems like a human brain would. In an artificial neural network connections between neurons function like the synapses of the brain, i.e. they receive and transmit signals from/ to other connected neurons. Using such tools, complex relationships between inputs and outputs can be modeled, patterns in data can be discovered, and unknown joint probability distributions between inputs can be observed.

**Deep learning:** deep learning methods and algorithms are an evolution of artificial neural networks made feasible by the size of the available data and the cheap price of processing power. In such methods, the algorithm focuses on learning data representations rather than a specific task. More complex neural networks are constructed, with each level offering a slightly more abstract and composite representation of the data.

Deep learning techniques and algorithms essentially form a separate discipline and are the topic of this thesis. As such, they are discussed separately in the following chapter.

## Deep learning

As discussed in the previous section, deep learning is a subsector of machine learning. This section presents the history and main techniques/ applications of the field.

The main feature that distinguishes deep learning methods from traditional machine learning methods is the focus on data representations which are now the learning goal rather than specific tasks. By using multiple levels of non-linear abstract information processing, deep learning algorithms facilitate feature learning, representation, classification and pattern recognition [1807.08169.pdf]. Each processing and extraction layer uses the output from the previous layer as input.

## History

The term Deep Learning (DL) first appears on a conference paper by Rita Dechter in 1986 [Learning\_While\_Searching\_in\_Constraint] and around 2000 it was first used in the context of artificial neural networks. Since then, the popularity of deep learning has exploded along with the number of applications in various industry sectors and associated research. Advances in hardware and in particular in the processing power of graphical processing units (GPUs) have improved the speed/ performance of DL networks by many factors thus enabling their use in a variety of popular applications. More recent developments and demonstrations of effective use of DL techniques in areas such as image recognition and bioinformatics are thought of as paving the way for a deep learning revolution [7].

## Evolution of architectures

### Artificial neural networks

As it was previously mentioned, artificial neural networks (ANN) draw terminology and inspiration from the biological neural networks found in animal brains and can process complex data inputs without necessarily being task driven. While the first generation of ANNs used simple neural layers that were limited to simple computations, the advent of the mechanism of backpropagation enabled a much powerful second generation.

Backpropagation allows the weights of neurons to be recalculated according to error rates (gradient of the loss function), thus allowing for multiple hidden layers that use feedback (backwards propagated correction information) to readjust their output. Backpropagation, along with other techniques that surpassed its limitations significantly

improved ANNs and enabled them to be designed in various ways and for various purposes.

### Deep learning networks

Deep Learning (DL) networks are neural networks characterized by a significant number of hidden layers which are used for input feature extraction and calculations. Variations of methods/ tools/ data structures in this category include auto-encoders, convolutional deep neural networks and recurrent neural networks, among others.

In DL networks, learning can be supervised, unsupervised or semi-supervised and even reinforcement learning is applicable.

### Autoencoders

An autoencoder is a special type of artificial neural network that can be used for learning efficient encodings by reconstructing its own inputs instead of predicting some target value  $Y$  given inputs  $X$ .

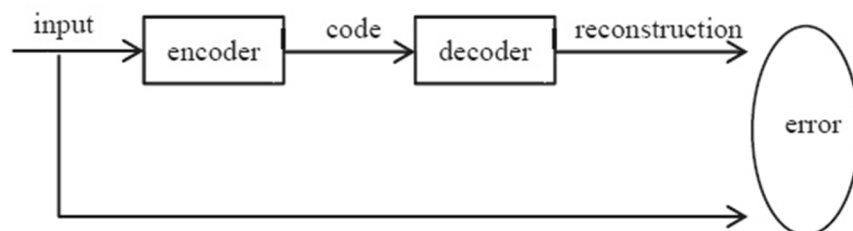


Figure 1: Basic process of an autoencoder [14]

Figure 1 illustrates the operation of an autoencoder where the corresponding code is the learned feature and optimization is minimizing the input reconstruction error. A single layer is generally not able to effectively capture the features of raw data, hence, in deep learning networks the principle of autoencoders is extended to enable deep autoencoders. In an autoencoder, a compressed form of the input (dimensionality reduction) is decoded to reconstruct the input. In a deep autoencoder, lower hidden layers are used for encoding and higher ones for decoding, with error back-propagation

is used for training and multiple hidden layers connecting input and output. This is a form of unsupervised learning network with a multilayer feed forward artificial neural network whose purpose is to reconstruct its own inputs.

### Convolutional deep neural networks

Convolutional neural networks (CNNs) are based on four pillars:

- Connections are local between neurons of adjacent layers.
- There are pooling layers, i.e. layers in which outputs from multiple neurons are combined to a single one.
- Weights of features are shared across all neurons of the same layer.
- Multiple hidden layers exist between input and output layers.

In a CNN, there are four separate types of layers: convolutional layers, pooling layers, fully connected layers and normalization layers. Convolutional layers detect local conjunctions from features use convolutions instead of matrix multiplications. Convolutional neural networks were designed to be suitable for vision related applications such as image recognition and video analysis.

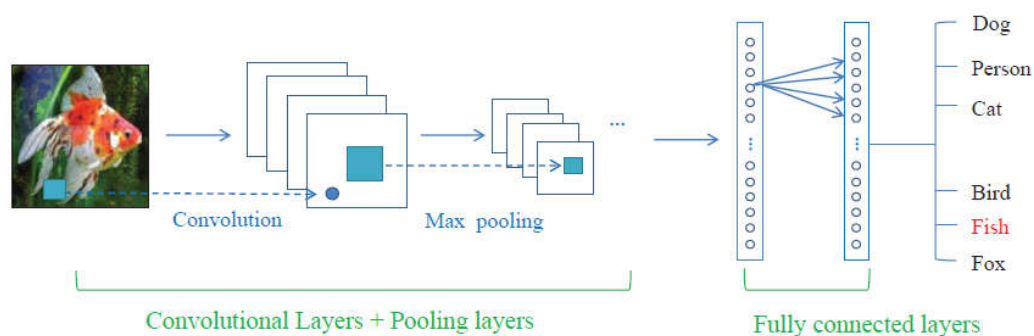


Figure 2: Layer-by-layer architecture of a convolutional neural network (practical application: image classification) [Guo et al., 2016]

Figure 2 depicts the layer-by-layer architecture of a convolutional neural network, designed for image classification. The network is trained in two stages, a forward and a backward one [14]. In each layer of the forward stage, the input image is represented with the current weight and bias parameters. Based on the loss cost computed with the

ground truth labels, the backward stages computes the gradients of each parameter and the results are then used to update the parameters and feed them to the next forward computation. Learning continues after a sufficient number of iterations of the forward and backward stages is complete.

### Recurrent neural networks

Recurrent Neural Networks (RNNs) are better suited for applications involving sequential inputs such as speech and text because they are designed to process sequences of inputs by using the internal memory. Their main distinguishing feature are recurrent hidden units that can be considered as very deep feedforward network with same weights when unfolded in time [1807.08169].

### Deep Belief Networks

A deep belief network (DBN) is a probabilistic generative model which provides a joint probability distribution over observable data and labels [14]. DBNs are formed by “stacking” Restricted Boltzmann Machines, i.e. neural networks that can learn a probability distribution over their inputs. In a DBN, the deep network is first initialized with an efficient layer-by-layer greedy learning strategy. The computed weights are then fine-tuned jointly with the desired outputs. This approach resolves the issue with selecting initial parameters which potentially lead to poor local optima and does not require labeled data for training. This, however, is a computationally expensive task that may involve training several Restricted Boltzmann Machines.

### Main applications of deep learning networks

Deep learning network applications can be summarized in the phrase signal processing where both the terms signal and processing are properly extended [8]. In DL applications, a signal can be audio/ speech, image/ video, but also text/ language and document/ information. Accordingly, processing is not only limited to traditional applications such as coding, analysis, and recognition but also includes interpretation/ understanding, retrieval/ mining, and user interface/ recreation.

The main groups of applications for deep learning networks are as follows:

1. **Speech & audio:** includes applications such as speech recognition (e.g., speaker and language) and speech synthesis, music signal processing and music information retrieval.
2. **Image & video:** includes applications such as image classification (discovery/labelling of main theme for each image and determining a set of additional labels based on probabilities), object detection (detecting the presence of a given class and estimating the position of the instance), image retrieval (discovery of visually similar images or images containing the same object), human pose estimation (recognizing people in images, detecting and describing human body parts and their spatial configuration) and many others.
3. **Natural language processing:** includes applications such as text recognition and semantic parsing, machine translation, automatic text summarization, automatic paraphrasing, information retrieval, sentiment analysis and many more. These applications will be presented in more detail in the following chapter.
4. **Bioinformatics:** the power of deep learning networks has been harvested in many diverse fields related to bioinformatics such as protein structure prediction, gene patterns associations with functions, biomolecular target prediction in drug design and synthesis and others.

### Challenges of deep learning networks

The previous paragraphs focused on the incredible properties and applications of deep learning networks. This paragraph will highlight, in brief, some challenges faced by deep learning networks that either limit their applicability or have delayed their advent.

**Computation time/ hardware requirements:** deep learning network based solutions were only made feasible following dramatically increased chip processing abilities and particularly Graphics Processing Units (GPUs) improvements and significantly lowered cost, as these units are the most suitable ones for matrix and vector computations.

**Initial parameterization:** refers to the difficulty in determining optimal values of initial training parameters, e.g., size (number of layers and number of units per layer), learning

rate, and weights. Exhaustive search solutions for optimal solutions may not be feasible due to the computational cost which results in unrealistic durations.

**Overfitting/ underfitting:** the overfitting problem, which is often observed in models with millions of parameters such as deep belief networks refers to the discovery of rare and weak dependencies in training data. This issue can be effectively addressed by generative pretraining steps such as regularization or data augmentation [34].

**Lack of theoretical explanation/ convergence proof:** this criticism concerns some more complex deep learning architectures where the exact nature of learning and the probability of convergence and the time it will occur are unclear and the networks more closely resemble empirical solutions or black boxes.





## 2. Natural Language Processing – Background

The term refers to any task of automated processing of natural language (written and spoken) such as machine translation, automatic summarization, paraphrasing and many others. Natural language processing (NLP) techniques can, for instance, be used for practical applications such as opinion mining and trend detection based on information available of the web. The entire World Wide Web can be thought of as a large collection of linguistic information that can be search, processed and classified. This approach referred to as the “Web as Corpus” has inherent caveats such as the limited or non-existent semantic structure and metadata. Over the last decade, deep learning has become the core of modern NLP and has practically replaced rule based and statistical methods, especially for language understanding.

### Main applications

This section summarizes key tasks in natural language processing, emphasizing those related to machine learning/ deep learning techniques.

#### Text & Document classification

As the number of documents available online and the size of each document constantly increase, properly classifying them becomes more difficult but also more imperative. Text classification is the process of identifying the category in which a document belongs (selected from a specified set of categories). Very often, text classification is seen as a supervised learning task in which labeled documents are given as input to the classifier in order for it to accurately identify the categories of new documents. It is obvious that the volume of training data impacts the precision of the process. Typically, the classification problem assumes categorical values for the labels, though it is also possible to use continuous values as labels [1].

The problem of text classification finds applications in a wide variety of domains and such algorithms are at the heart of many software systems that process text data at scale. It is commonly used in areas as follows:

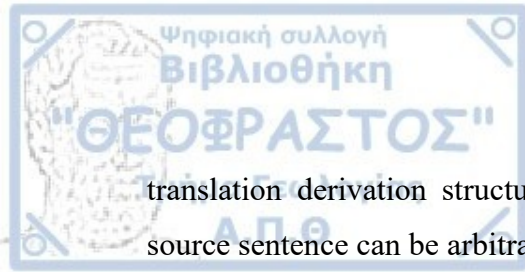


- *Email classification and spam filtering*: email software uses text classification to determine whether incoming mail will be put in the user inbox or filtered into the spam folder. Similarly, discussion forums use text classification to spot comments that need to be flagged as inappropriate, abusive, or commercial.
- *News filtering*: online news services deal with a large volume of articles created daily. The sheer volume makes manual organization very hard. Therefore, automated classification methods can be very useful for news categorization in web portals.
- *Opinion Mining/ Sentiment Analysis*: Customer reviews or opinions are often short text documents which can be mined to determine useful information such as whether the reviewer is positively or negatively inclined and even his emotional state.
- *Document Organization and Retrieval*: This refers to large digital libraries of documents, web collections, scientific literature, or even social feeds.

Techniques for classification that have been proposed in literature include decision trees, rule-based classifiers, state vector machines, neural networks, Bayesian and others. An important issue in text classification is feature selection. This refers to determining the features which are most relevant to the classification process which is very important because some of the words are much more likely to be correlated to the class distribution than others.

### Machine translation

Machine translation as an NLP application refers to finding the most probable target language sentence for the source language sentence, i.e., the sentence that shares the most similar meaning. Essentially, machine translation is a sequence-to-sequence prediction task [43]. Statistical models dominate the machine translation community but they face severe difficulties in obtaining accurate word alignments, in determining the optimum translation for a given source phrase because a source phrase can have many translations, and different contexts lead to different translations, and in predicting the



translation derivation structure because phrase partition and phrase reordering for a source sentence can be arbitrary.

Overall, with statistical machine translation, it is difficult to learn a good language model and this is where deep learning models and networks come into play.

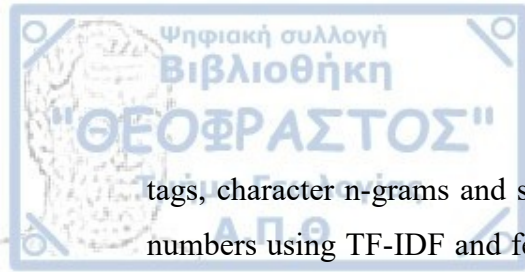
### Native Language Identification

When someone speaks in a given language it is easy for the listeners to identify whether he is a native speaker of that language or even identify the speaker's native language based on the accent. Native Language Identification (NLI) is the task of identifying the native language of authors of texts written in a (potentially) foreign language. NLI is modeled as a text classification task with labels corresponding to native languages. The basis of NLI is the assumption that one's mother tongue influences the way they acquire and produce second languages (Second Language Acquisition – SLA) and that traits easily identifiable in speech production should be identifiable in written texts as well.

The motivation for NLI is twofold. First, there is a linguistic motivation related to the interference between languages learnt and the degree of difficulty based on their similarities and secondly the task has a practical relevance and can be integrated to a number of computational applications. Interesting practical applications include forensic linguistics and in particular authorship profiling which is the process of discovering and asserting information about the writer of a given text, such as age, gender and native language.

The authors of [11] design and develop an NLI system based on linear classifiers which uses TF-IDF weighting for terms. They split the task in three stages. In the first stage, the exact mode in which training and development data will be used is determined. In the second stage, the features that will be extracted are selected and in the third stage the machine learning algorithms that will be applied are chosen along with their parameters considering the time and memory restrictions.

The dataset selected for training and testing consisted of around 12000 essays (300-400 words) written by authors of 11 different native languages whose English was evaluated (by humans) in 3 different levels. Several options were considered for features such as word unigrams, bigrams or n-grams present in essays, part-of-speech



tags, character n-grams and spelling errors. All features, were mapped into normalized numbers using TF-IDF and features that occurred in less than 5% or more than 50% of the essays were removed. For Term Frequency, a logarithmic relationship was chosen (sublinear TF scaling). Differences in essay lengths, were counterbalanced by normalizing each feature vector. After normalization, the resulting essay feature vectors were fed into classifiers. Three types of linear classifiers were used, i.e., linear support vector machines, logistic regression and perceptrons.

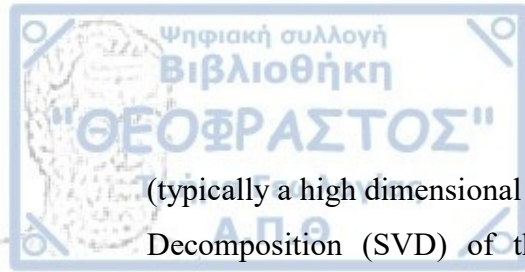
It was noted that for several languages, the features that were most active and separated native languages included names of countries or languages. These were labeled stop-words and removed from the corpus using TF-IDF. Authors reported a success rate ranging between 95% and 72% with the confusion matrix suggesting that languages of geographically closer languages were more often mistaken for one another.

#### Text similarity

Estimating the similarity between two texts of arbitrary length (not necessarily of similar lengths, e.g., a search query and a document) is defined as computing a metric of the semantic distance between the two texts that reflects their actual relatedness. Although it can be thought of as a standalone natural language processing application it is most commonly considered part of other application such as paraphrasing, plagiarism detection, summarization, translation and, of course, indexing & classification, among others.

Determining similarity is a complex and fundamental issue as evidenced by issues in trying to match queries with documents (e.g., in search engines results retrieval). Users want to retrieve conceptually similar content even when they do not use the exact words as the documents and the words they use may not even exist in relevant document. Conversely, because words have multiple meanings, the match of a term in a document does not guarantee that it is of interest to the user.

Latent Semantic Analysis (LSA) [Deerwester et al., 1990] was proposed in an effort to overcome these problems by mapping documents and terms into a representation in the space referred to latent semantic space. This is accomplished by starting with the vector space representations of documents based on term frequencies



(typically a high dimensional vector) and applying a mapping based on a Singular Value Decomposition (SVD) of the corresponding term/document matrix resulting in a reduced space representation. The underlying premise is that document with terms that frequently co-occur will be represented similarly in this space even if the terms are not in the exact same words. This is a form of noise reduction and LSA can detect synonyms as well as words that refer. The corresponding similarity measure is referred to as Latent Semantic Index (LSI).

In [Hoffmann, 2017] a novel approach to LSA and factor analysis is presented that unlike the original LSA has a solid statistical foundation since it applies standard techniques from statistics for questions like model fitting, model combination. Probabilistic Latent Semantic Analysis (PLSA) defines a proper generative model of the data and can deal with polysemous words and distinguish between different meanings/ usages of the same words.

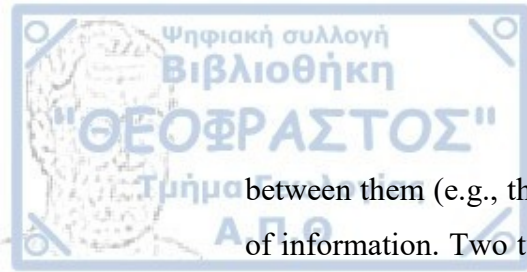
### Foundation/ Key concepts

This section briefly covers the foundations needed in order for the rest of the work to follow. Basic notions of natural language processing are presented and terminology used throughout the thesis.

### Preprocessing and parsing

This paragraph summarizes a set of processing tasks that need to be performed before the “native text” is handed over for computational handling as they transform the original sequence of characters to a cleaner form. Preprocessing typically encompasses the following tasks:

- *Tokenization*: this is typically the first step in a natural language processing solution and it refers to splitting the text into meaningful character sequences/ self-contained semantic units, e.g. words or sentences. A naïve tokenization solution involves removing punctuation and splitting the text by blank spaces.
- *Normalization*: this involves removing morphological variations from words such as capitalization, plural number or tenses, in order to grasp similarities



between them (e.g., the same word in singular and plural), obviously with a loss of information. Two types of techniques are used, stemming and lemmatization.

In the former, language specific patterns are recognized, using for example the rules for converting words from singular to plural or verb tenses. This technique is simple, fast and applicable for large volumes of text. Lemmatization involves using a dictionary (such as WordNet that is both a dictionary and a thesaurus) to extract the roots of common words. This approach can be more accurate compared to stemming, but it is more resource intensive and dictionaries may be incomplete for certain languages. The two methods can complement each other and they are often used in conjunction.

- *Parsing*: this involves a group of functions that are used after term isolation and document cleanup, i.e., after normalization and parsing, which facilitate working in higher abstraction layers. Typically, parsing includes morphological and syntactical analysis of tokens in order to identify their role within sentences (e.g. noun, verb, adjective or object-verb-subject), which is referred to as Part-of-Speech (POS) tagging.

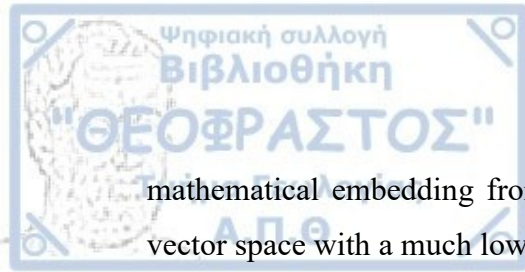
### Word senses

A word sense is the meaning of a word. As several words have multiple meanings when used in different contexts (polysemy) and words can often have the same meaning (synonyms), there is not a 1-to-1 mapping between words and senses. Word-sense disambiguation is the process of identifying the particular meaning of a word based on the way it is used in a sentence and its context. Part of speech tagging is the first step in the disambiguation process. A more advanced task is Named Entity Recognition (NER) which involves identifying and tagging among others, people's names, organizations and geographical locations within the text.

### Word embeddings

Word embedding is the collective name for a set of language modeling and feature learning techniques in natural language processing (NLP) where words or phrases from the vocabulary are mapped to vectors of real numbers. Conceptually it involves a





mathematical embedding from a space with one dimension per word to a continuous vector space with a much lower dimension.

Word and phrase embeddings, when used as the underlying input representation, have been shown to boost the performance in NLP tasks such as syntactic parsing and sentiment analysis.

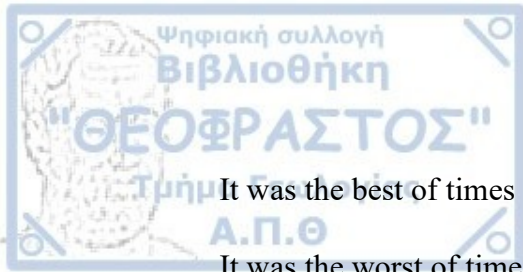
#### Word representations in vector space

As it was previously mentioned, machine learning algorithms cannot work with raw text directly but can work with vectors of numbers. If these vectors are derived from textual data in a mode that captures various linguistic properties of the text, the process is called feature extraction or feature encoding. The following sections present popular approaches. It must be noted that all these approaches are in line with the distributional hypothesis as portrayed by Harris in 1954 “*Words that occur in similar contexts tend to have similar meanings*” [16] and later on by J.R. Firth in 1957 “*You shall know a word by the company it keeps*” [10].

#### *Bag-of-words*

A popular and simple method of feature extraction with text data is the bag-of-words (BoW) model of text. As the name suggests, this model treats documents like bags of words, i.e. as containers where the order of items does not matter. Bags are essentially sets that are allowed to have more than one instances of the same item, meaning that a word may be found in the bag (document) multiple times. This is referred to as multiplicity and it is maintained in this model. The idea behind BoW is that documents are similar if they have similar content and that we can learn something about the meaning of the document from its content. A bag-of-words implementation can be simple or complex depending on decisions regarding the design of the vocabulary of known words (tokens) and the scoring system for known words [5].

The mode in which this model represents individual documents and the entire corpus is best illustrated via an example [9]. Let’s consider the following corpus where each sentence represents a separate document:



It was the best of times

It was the worst of times

It was the age of wisdom

It was the age of foolishness

The distinct words that appear in the corpus are:

'It', 'was', 'the', 'best', 'of', 'times', 'worst', 'age', 'wisdom', 'foolishness'

In order to map documents to vectors, we count the frequencies for all terms (even those that are not present). These vectors can then be fed into machine learning algorithms. For instance, the first document ("It was the best of times") has the following frequencies for each of the 10 unique words.

"it" = 1

"was" = 1

"the" = 1

"best" = 1

"of" = 1

"times" = 1

"worst" = 0

"age" = 0

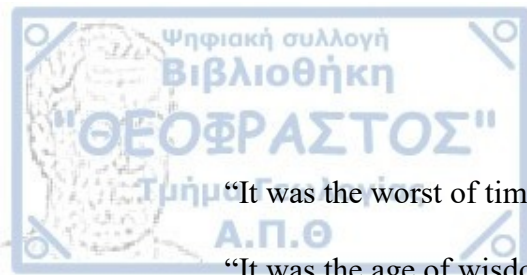
"wisdom" = 0

"foolishness" = 0

Therefore, the vector corresponding to this document is [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]

Similarly, the remaining documents will be:





“It was the worst of times” = [1, 1, 1, 0, 1, 1, 1, 0, 0, 0]

“It was the age of wisdom” = [1, 1, 1, 0, 1, 0, 0, 1, 1, 0]

“It was the age of foolishness” = [1, 1, 1, 0, 1, 0, 0, 1, 0, 1]

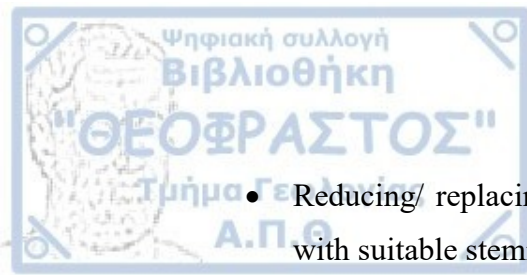
The process of converting natural language text into numbers is called vectorization in machine learning and different ways to convert text into vectors have been proposed (presented in detail in the sections that follow). Indicative approaches include:

- Considering the number of times each word appears in a document.
- Considering the frequency that each word appears in a document relative to all the words in the document.

As the vocabulary size increases, so does the vector representation of documents since the length of the document vector is equal to the number of known words. For a large corpus this could amount to thousands or millions of words whose positions must be tracked. If certain words are relatively rare (i.e. few documents contain them), this results in a vector with lots of frequencies equal to 0, namely a sparse vector. Sparse vectors take up memory and computational resources when modeling while not actually containing useful information. The size of the vocabulary is a serious challenge for modeling algorithms and text cleaning techniques need to be applied to reduce it in a bag-of-words model.

Simple text cleaning techniques that can be used to reduce the size of the vocabulary include:

- Case-insensitivity (case is ignored)
- Ignoring punctuation
- Removing stop words (i.e. too frequent words that don't contain actual information, like articles)
- Correcting spelling errors



- Reducing/ replacing words to/ with their stem (e.g. “play” from “playing”) with suitable stemming algorithms.

### *n-grams and skip-grams*

In the bag-of-words model, each word or token is referred to as a “gram” and the model can be extended to consider more than single words. For example, creating a vocabulary of two-word pairs (consecutive words) would result in the following pairs for the first document “It was the best of times”:

“it was”

“was the”

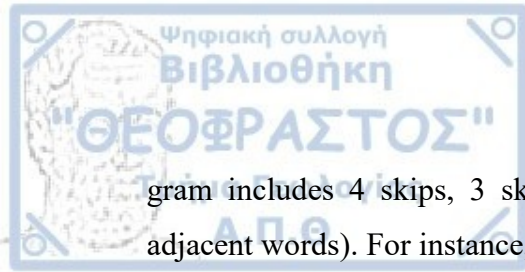
“the best”

“best of”

“of times”

These tokens are called bigrams and the concept can be extended to trigrams and generally n-grams. Using multiple consecutive words as tokens both changes the size of the vocabulary and allows the bag-of-words to capture a little bit more meaning from the document. N-gram models can be used to calculate probabilities for words based on the words already encountered. When using n-grams for language modeling, it is assumed that each word depends only on the last n-1 words which means that the model is considered a good approximation of the true underlying language. This principal is summed up in the phrase “language is its own best model” by some scientists [15] and it implies that sufficient data can be gathered to depict typical (or atypical) language use accurately. In the effort to solve the data sparsity problem presented in the previous section, researchers have proposed the concept of skip-grams.

Skip-grams [15] are a technique in which n-grams are formed (for various values of n) but in addition to tracking adjacent sequences of words, tokens are allowed to be “skipped”. Skip-grams for a certain skip distance k allow a total of k or fewer words to be omitted when constructing the n-gram which means that, for example, a 4-skip-n-



gram includes 4 skips, 3 skips, 2 skips, 1 skip, and 0 skips (typical n-grams with adjacent words). For instance, for the sentence

“Insurgents killed in ongoing fighting.”

the following sets of skip-grams can be constructed for  $k=2$ :

*2-skip-bi-grams* = {insurgents killed, insurgents in, insurgents ongoing, killed in, killed ongoing, killed fighting, in ongoing, in fighting, ongoing fighting}

*2-skip-tri-grams* = {insurgents killed in, insurgents killed ongoing, insurgents killed fighting, insurgents in ongoing, insurgents in fighting, insurgents ongoing fighting, killed in ongoing, killed in fighting, killed ongoing fighting, in ongoing fighting}.

The importance and versatility of n-gram models is illustrated by the fact that Google has created a tool that allows users to track the use of a particular phrase in books through time. The corpus is large and contains books scanned from public libraries in several languages (Greek unfortunately is not included). Google Books Ngram Viewer, which is available in <https://books.google.com/ngrams> will output a graph that represents the use of a particular phrase in books through time. An example for the phrases machine learning, deep learning and natural language processing for a date range from 1978-2008 (the latest available year) is shown in Figure 3. Case insensitivity and smoothing options are also available.

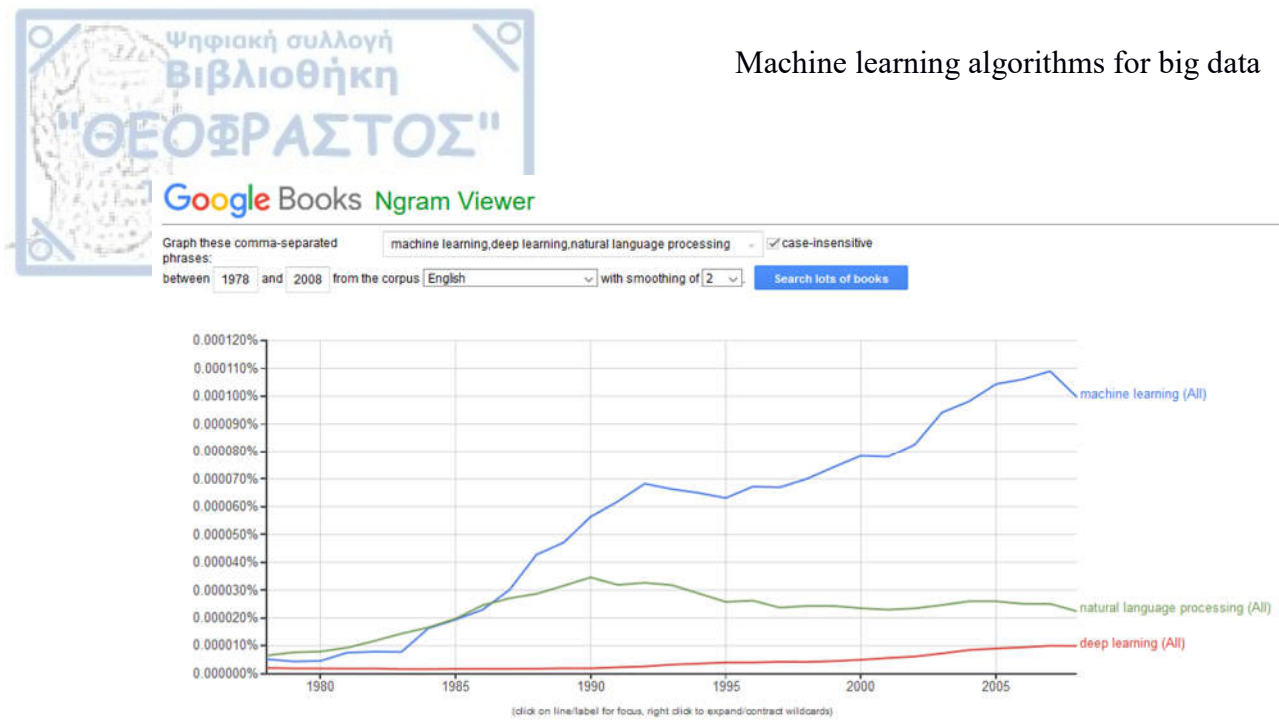
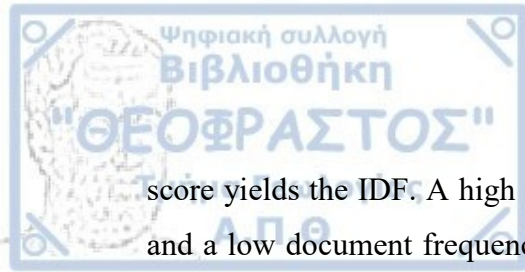


Figure 3: Output of Google's Ngram viewer for the phrases machine learning, deep learning, natural language processing

## Information retrieval with TF-IDF

Information retrieval techniques described in this section are primarily used for document representation and classification. The goal is to provide a simplified representation of documents while preserving their features.

Term Frequency-Inverse Document Frequency or TF-IDF is a numerical statistic used very often in information retrieval applications to estimate the importance of a term in a document, a collection or an entire corpus. The idea behind TF-IDF is simple and straightforward and relies on the two factors included in its name. The combination of these two factors tends to correspond to the way human minds tend to evaluate search relevance [os connections bm25]. Term Frequency (TF) is a value that represents how often a given word appears in a document. Words appearing many times are considered important for the document. However, a very high frequency of the word in the entire corpus means that it is a common word for the given topic therefore its score should be penalized. It must be noted that a word can be rare in general but frequent in a collection of documents for a given topic. For instance, the word “atom” is relatively rare generally but very common in a corpus discussing physics. In TF-IDF, this is adjusted using the second component, i.e. Inverse Document Frequency (IDF). Document Frequency is obtained by counting the number of documents that contain a term and computing a ratio of the total number of documents divided by this value. Inverting this



score yields the IDF. A high TF-IDF score/ weight is reached by a high term frequency and a low document frequency of the term in the entire corpus. Hence, common terms tend to have low weights and are filtered out. In this direction, TF-IDF is often used for stop-words filtering in various subject fields and for various applications such as text summarization and classification.

TF-IDF measures the relative concentration of a term in a given set of documents/articles. If a word is common in a given item but relatively rare elsewhere then the score should and will be high, i.e. the document is very relevant to the search term. Inversely, if a word occurs few times in one document and many times in other documents the TF-IDF score will be relatively low.

Document length is an additional measure that should be taken into account. A term occurring twice in a 400 page book does not mean that the book is about the term while a word contained twice in a short post implies that the post is indeed about the word. This additional bias is “fieldNorms” and favors significantly shorter documents matching a term over longer ones. Term concentration in the shorter document is thought to be an important weighing factor on the relevance of the document with the term and thus should be scored higher.

In the simplest implementation of TF-IDF the weight of a term that occurs in a document is simply proportional to the term frequency, i.e. the number of occurrences of the term in the document. The IDF component is a bit more tricky. The requirement is not to overestimate the importance of documents containing the common words more frequently and downplay the weight of terms with very high frequencies. Karen Spärck Jones in her 1972 paper entitled “A statistical interpretation of term specificity and its application in retrieval” [36] elaborated on the concepts of specificity and exhaustivity and introduced the key concept of Inverse Document Frequency which is now pivotal in term weighting. Spärck Jones argued that specificity should be interpreted as a statistical and not a semantic property of index terms. Then, the exhaustivity of a document description is the number of terms it contains, and the specificity of a term is the number of documents to which it pertains. Thus, the specificity of a term can be quantified as an inverse function of the number of documents in which it occurs. In the simplest implementation, the IDF for a given term is computed by dividing the total

number of documents by the number of documents containing the term and then scaling it logarithmically. Finally TF-IDF is the product of the two statistics.

$$TF - IDF = TF \times IDF$$

Various formulae have been proposed both for computing the TF and the IDF component. These are summarized in the tables that follow:

Frequency Weighting Scheme	Term Frequency Values
Binary, i.e. term found or not in the document	0, 1
Raw count, i.e. number of times term $t$ found in document $d$	$f_{t,d}$
Term frequency, i.e. document length $D$ taken into account	$\frac{f_{t,d}}{D}$ or $\frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$
Log normalization	$\log(1 + f_{t,d})$
Double normalization with factor $K$ . This scheme is a remedy for the bias against longer documents.	$K + (1 - K) \frac{f_{t,d}}{\max\{t' \in d\} f_{t',d}}$

Table 1: Variants of Term Frequency (TF) computation

Weighting Scheme	IDF formula
Basic formula (number of documents that contain the term over the total number of documents – logarithmically scaled)	$\log\left(\frac{ D }{ \{d \in D \mid t \in d\} }\right)$
Unary	$1(\text{if the term is found})$
Inverse document frequency (problematic if term not found in any documents)	$\log\left(\frac{ D }{n_t}\right) = -\log$
Inverse document frequency (smooth)	$\log\left(1 + \frac{ D }{n_t}\right)$
Inverse document frequency (max)	$\log\left(\frac{\max\{t' \in d\} n_{t'}}{1 + n_t}\right)$

Table 2: Variants of Inverse Document Frequency (IDF) computation

### Applications of TF-IDF

In [39] the authors propose a document classification scheme using TF-IDF and a naive Bayes classifier to tag unstructured data either as true or false, where the two Boolean values may correspond to safe/ dangerous, spam/ not spam, etc. The authors emphasize

the fact that the classifier must be able to work fast with a large volume of data collected at a tremendous rate. The proposed classifier works in two phases (training and testing) with the training phase split further in stages. The first stage is the morphological analysis of the input document during which linguistic units are identified while the second one consists of using TF-IDF to extract features used as input in the Bayes classifier which progressively computes conditional probabilities of the document belonging to each class. The classifier which is implemented using Python libraries is trained with two sets of sample data (one for each category) in a supervised learning mode

#### Additional variations of TF-IDF

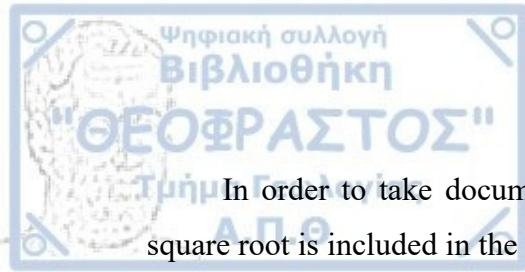
Variations of the basic idea of TF-IDF focus on addressing shortcomings related to how the values/scores it provides relate to human intuition of relevance. For instance, a term occurring two times more in a document does not mean that this document is twice more relevant to this term or a term appearing in two times more documents does not mean that its importance is half the importance of another term found half the times.

Apache Lucene, the free and open-source information retrieval software library supported by the Apache foundation (<http://lucene.apache.org/>) addressed these issues by modifying the basic scoring formula both in terms of the term frequency and the inverse document frequency components. Specifically, instead of the actual term frequency, its square root is used in the formula. This means that, for example, a document with 16 matches is roughly twice as relevant as a document with 4. The IDF component is also modified to reflect the fact that the score of a term appearing in 100 documents should not be 10 times more than the score of one appearing in 1000. The new formula for the IDF is [37]:

$$IDF = \log \left( \frac{numDocs}{docFreq + 1} + 1 \right)$$

In this formula numDocs is the total number of documents in the corpus. The logarithmic weighing means that the IDF component grows more slowly, for instance a term found in only 4 documents is roughly twice as special as a term found in 64.





In order to take document length into account in the scoring, the inverse of its square root is included in the final formula as *fieldNorms*, i.e. the final score is

$$\log \left( \frac{\text{numDocs}}{\text{docFreq} + 1} + 1 \right) \times \sqrt{TF} \times \frac{1}{\sqrt{\text{docLength}}}$$

## BM 25

BM 25, with BM standing for Best Matching is a variation/ improvement on TF-IDF which focuses on assessing the relevance of a document to a query and is widely used for results ranking in search engines. It is often referred to as Okapi BM 25 as it was developed in the context of the Okapi information retrieval system at London's City University in the 1980s and 1990s [31]. BM 25 combined previous variants BM 11 and BM 15 into a single weighting function. In this function, the IDF component is preserved while the TF component is redefined and based on two new parameters ( $k_1$  and  $b$ ). The formula for the relevance (score) of a document  $D$  for a query  $Q$  (that contains  $n$  keywords, labeled  $q_i$  with  $i$  ranging from 1 to  $n$ ) is as follows:

$$\text{score}(D, Q) = \sum_{i=1}^n \text{IDF}(q_i) \cdot \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left( 1 - b + b \cdot \frac{|D|}{\text{avgD}} \right)}$$

Where  $\text{IDF}(q_i)$  is equal to

$$\log \frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}$$

where  $n(q_i)$  is the number of documents that contain the term  $q_i$ .

Suggested values for  $k_1$  range between 1.2 and 2.0 and for  $b$  is equal to 0.75.

Additional extensions of BM25 focus on addressing specific deficiencies or enhancing semantics. BM25+ was proposed to resolve an issue with the lower bound on the term frequency component [22]. Authors observed and proved that the process of



normalization by document length is not properly lower-bounded and as a result, very long documents tend to be overly penalized. They proposed a solution based on an additional parameter  $\delta$  whose value must satisfy the following formula:

$$\delta \geq \frac{k_1}{k_1 + 2}$$

The new parameter  $\delta$  is added in the TF component in the scoring formula which becomes:

$$score(D, Q) = \sum_{i=1}^n IDF(q_i) \cdot \left( \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1 \cdot \left(1 - b + b \cdot \frac{|D|}{avgD}\right)} + \delta \right)$$

BM25F differs from other approaches in the sense that it does not consider the document as a single body of text, unstructured and undifferentiated. The lack of structure is not compatible with most search systems which assume at least some minimal structure in documents [42]. BM25F considers a single flat stream structure, common to all documents, i.e., that the text of each document is split between a global set of labelled streams, for example, a title-abstract-body structure. The ranking function is then applied separately to each stream, and the results are linearly combined (with stream weights) to yield the final document score [32]. Intuitively, the presence of a keyword in the document title will be given a higher weight and will thus contribute to a higher score and the document that contains the keyword in its title will be considered more important.





### 3. Word embedding algorithms

This section presents algorithms that given a text of variable length as input, provide vector representations for the words it contains.

#### Word2vec

Word2vec is a tool for computing continuous distributed representations of words that was created by a team of researchers led by Tomas Mikolov at Google in 2013 [26] and is distributed as open source software with an Apache License(<https://www.apache.org>).

Word2vec is essentially a group of related models that provides an efficient implementation of the continuous bag-of-words and skip-gram model architectures to compute distributed vector representations of words from very large data sets. Following this transformation, the vector representations can be fed into many natural language processing applications such as text classification or machine translation.

The input of word2vec is a large corpus of text (in the range of more than 1 billion words with millions of distinct words) and produces a vector space with each word corresponding to a vector positioned in such a way that words that share common contexts in the corpus are close to one another.

In their seminal 2013 paper the researchers from Google report that their models result in high quality vector representations. The quality is tested by feeding the output into a word similarity task, and the results are compared to previously best performing techniques based on different types of neural networks. Significant improvements are observed both in accuracy and computational cost with a state-of-the-art performance on a test set used for measuring syntactic and semantic word similarities.

This section presents the motivation and rationale behind word2vec, its mode of operation and its output. Furthermore, example applications and variations that have been proposed in literature are also included.

### Motivation / Rationale for word2vec

The main assumption that word2Vec is based on is that words with similar contexts have similar meaning. Up to then, most NLP systems and techniques treated words as atomic units disregarding the notion of similarity between them. This approach was justified

since simple models trained on large volumes of data outperformed complex systems trained on smaller data sets. The researchers behind word2vec initially observed that in the vector representations, similar words not only tend to be close to each other, but also words can have multiple degrees of similarity but were then surprised to discover that similarity of word representations goes beyond simple syntactic regularities. In a well-known example of simple algebraic operations on word vectors, they showed that

$$\text{vector}(\text{"King"}) - \text{vector}(\text{"Man"}) + \text{vector}(\text{"Woman"})$$

results in a vector that is closest to vector("Queen").

The learned vectors explicitly encode many linguistic regularities and patterns and many of these patterns can be represented as linear translations, something that originally surprised researchers. In fact, using word2vec, both syntactic and semantic regularities can be learned with high accuracy that depends on the dimensionality of word vectors and on the volume of training data. It must be noted that although many different models for estimating continuous representations of words had been proposed, such as Latent Semantic Analysis and Latent Dirichlet Allocation, the creators of word2vec focused on distributed representations of words learned by neural networks, as they were superior in preserving linear regularities among words and computationally affordable for large data sets.

Nevertheless, the question of why it works has baffled researchers and several efforts have focused on making the intuition more precise [13].

### Word2vec Operation

The tool first constructs a vocabulary from the training text data and then learns vector representation of words. The simplest way to explore the learned representations is to

use the distance tool to find the closest words for a given one. For example, the closest, i.e., most similar words to france are shown in the following table along with their distances (<https://code.google.com/archive/p/word2vec/>).

Word	Cosine distance
Spain	0.678515
Belgium	0.665923
netherlands	0.652428
Italy	0.633130
switzerland	0.622323
luxembourg	0.610033
Portugal	0.577154
Russia	0.571507
germany	0.563291
catalonia	0.534176

### Architecture

For all the models investigated in the context of word2vec, the complexity was defined as the number of parameters that had to be computed in order to complete its training according to the following formula:

$$O = E \times T \times Q$$

where E is number of the training epochs (passes), T is the number of the words in the training set and Q depends on the particular model architecture. Common choices for E include values between 3 and 50 and for T values approaching one billion.

Previously proposed model architectures that were considered in word2vec were the probabilistic feedforward neural network language model (NNLM) which consists of input, projection, hidden and output layers and the Recurrent Neural Network Language Model (RNNLM) which had been proposed to overcome certain limitations of NNLM and only has input, hidden and output layer. In NNLM, the hidden layer (of size H) is used to compute probability distribution over all the words in the vocabulary, resulting in an output layer with dimensionality V. The complexity is dominated by the dimensionality of the projection layer (typically between 500 and 2000) and that of the

hidden layer. The complexity in RNNLM is dominated by the square of the size of the hidden layer.

The new log-linear models that were proposed in the context of word2vec in order to reduce computational complexity by partly sacrificing precision of data representation could be trained on much more data efficiently. These were:

**Continuous Bag-of-Words (CBOW) Model:** this is essentially a feedforward NNLM, where the non-linear hidden layer has been removed and the projection layer is shared for all words (whereas previously only the projection matrix was shared). This architecture is a type of bag-of-words model as the order of words in the history does not influence the projection. Not only words from recent history but also from the future are used. For example, a log-linear classifier for a given word could consider the four previous and the 4 next words. The computational complexity  $Q$  of the model is:

$$Q = N \times D + D \times \log_2 V,$$

where  $V$  is the size of the vocabulary,  $N$  is the size of the context window and  $N \times D$  is the size of the projection layer.

Unlike standard bag-of-words model, this model uses continuous distributed representation of the context, hence its name. CBOW predicts the current word based on the context and its architecture is shown in Figure 4.

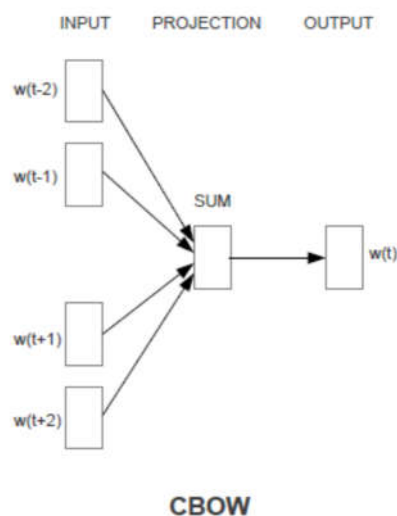


Figure 4: CBOW model architecture [20]

**Continuous Skip-gram Model:** this is a similar architecture to CBOW but instead of predicting the current word based on the context, it tries to predict surrounding words given the current word. Each word encountered is fed as input to a log-linear classifier with a continuous projection layer, and words within a certain range before and after the current word are predicted. The size of the range is a tradeoff between the quality of the resulting vectors and the computational complexity. Distant words are typically assigned lowered weights and are sampled less often given that they are generally less related to the current one.

The training complexity of this architecture is:

$$Q = C \times (D + D \times \log_2 V),$$

where  $C$  is the maximum distance of the words considered. For example, for  $C=5$ , for each training word, a random number  $R$  between 1 and 5 is selected and  $R$  past and  $R$  future words are used as correct labels. This results in  $2 \times R$  classifications and outputs. A typical value for  $C$  is 10. The architecture for the continuous skip-gram model is shown in Figure 5 where it is evident that it predicts surrounding words based on the current word.

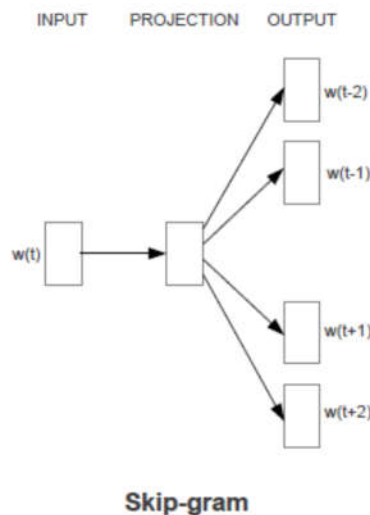
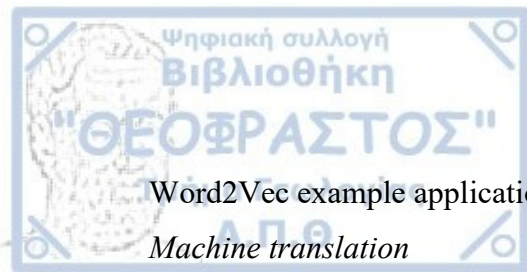


Figure 5: Continuous skip-gram model architecture in word2vec (Mikolov et al., 2013)



The models proposed in the context of word2vec can be used to automate the process of generating dictionaries and phrase tables which are fundamental in machine translation, thus complementing mainstream techniques that rely primarily on raw word counts used in statistical machine translation. The authors of [24] propose achieving this by learning a linear projection between vector spaces that represent the two languages.

Figure 6 illustrates the basic principle behind the idea. Vectors for two groups of related words (numbers and animals) are visualized for the two languages that will be translated (English and Spanish). As it is evident in the figure, concepts have similar geometric arrangements in both languages due to the fact that they are grounded in the real world. This similarity is the key reason why the proposed method works well. Typically, morphological features such as edit distance between word spellings are used to improve performance in translations between related languages (such as the two in the example, English and Spanish). The method based on word2vec can be used for translation between languages that are substantially different (for example, English and Chinese).

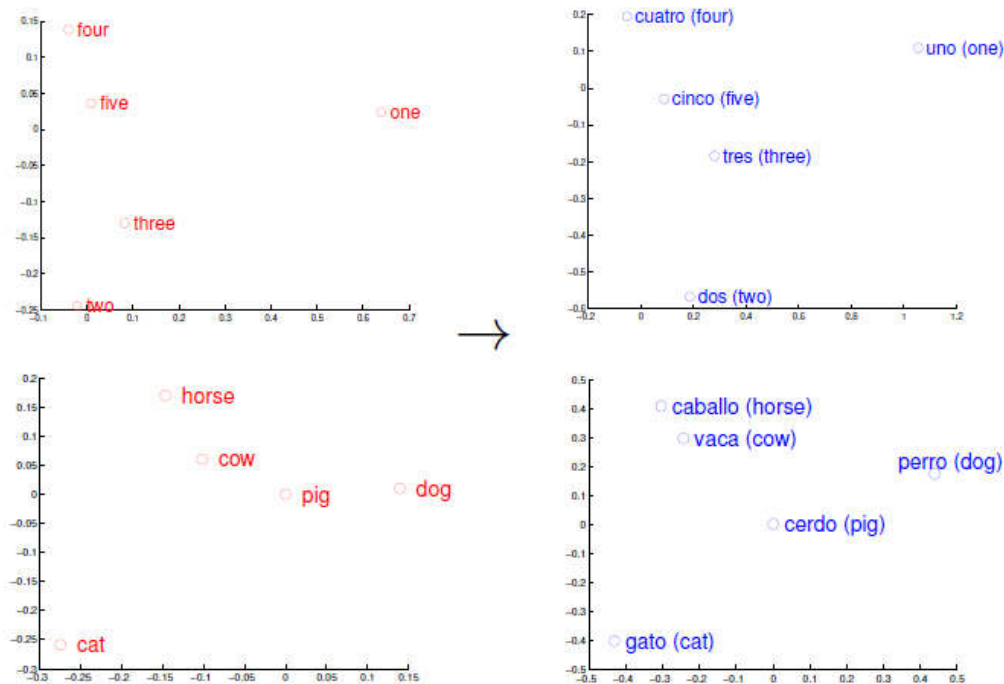


Figure 6: Distributed word representations of numbers and animals in English and Spanish [21]





To obtain the representations as they are shown in Figure 6, the vectors in each language are projected down to two dimensions using principal component analysis, and then manually rotated to accentuate similarities. The similar geometric arrangements suggest that it is possible to learn an accurate linear mapping from one space to another. To put this to action, first, monolingual models of languages are built using large amounts of text. Next, a small bilingual dictionary is used to learn a linear projection between the languages. During the test phase, any word present in the single language corpora can be translated by projecting its vector representation from the source language space to the target language space. The translated word is selected as the most similar word vector in the target language space.

### *Text classification*

Word2vec brings extra semantic features that can help in text classification which is becoming more difficult as the volume of online documents increases [21]. Classification is traditionally based on document representation using information retrieval techniques, for example continuous bag-of-words or tf-idf, that provide a simplified representation of documents through various features. CBOW disregards grammar and word order but keeps multiplicity while tf-idf reflects the importance of a word to a particular document in a collection of documents or corpus.

On the contrary, word2vec is unable to distinguish the importance of each word within the document being classified and treats each word equally. This makes it difficult to extract which words hold higher value over others. The authors of [21] combined word2vec with tf-idf to get the best of both methods in a classification task.

Mathematically speaking, the first step was getting a vector representation using word2vec. Following that, they applied weights using tf-idf weighting with word2vec and then concatenated tf-idf with word2vec weighted by tf-idf. The concatenation operation was in fact vector merging. By adding weights to each word corresponding to its frequency within the document in word2vec, they created weighted sums of word vectors. Stop words were omitted to improve accuracy and skip-gram was used for higher semantic accuracy (at the cost of time efficiency).



Reported results show that the combination of word2vec weighted by tf-idf without stop words and tf-idf without stop words can outperform either word2vec weighted by tf-idf without stop words or tf-idf with or without stop words. Although, the difference in performance is relatively marginal, the performance is consistent (i.e., the combination is reliable) and as the number of different categories increases, the difference in scores becomes even smaller which means that the addition of categories does not offer new information.

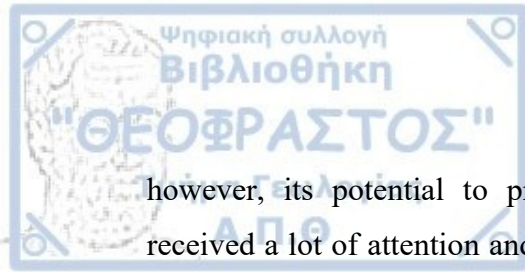
#### Extensions/ Variations of word2vec

The research team behind word2vec published a later paper where they presented several extensions (particularly to the continuous skip-gram model) that improve both the quality of the vectors and the training speed [24]. The first two extensions are essentially additional parameters that are related to subsampling and rare-word pruning. Words appearing fewer than *min-count* times are not taken into account neither as words nor as contexts. Conversely, words appearing more frequently than *sample* times are down-sampled and removed from the text before generating the contexts. This increases the effective window size for some words as more distant words are actually considered and these words are indeed meaningful and not, for example, stop words. This subsampling not only improves the accuracy of representations of less frequent words but also speeds up the process by several orders of magnitude.

Another important extension is a simplified variant of Noise Contrastive Estimation (NCE) for training the Skip-gram model that results in faster training and better vector representations for frequent words, compared to more complex hierarchical Softmax that was used in the original proposal.

#### *Paragraph vector*

This extension was proposed by the creator of the original word2vec and another researcher in a 2014 paper [20] in order to counteract drawbacks of bag-of-words methods such as their disregard for word order and their little sense of semantics. Paragraph vector provides continuous distributed vector representations for variable-length pieces of texts (such as sentences and paragraphs). Since it was proposed,



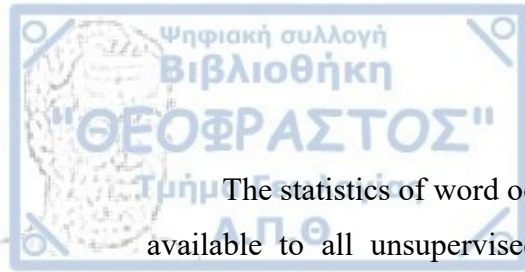
however, its potential to provide vector representations for entire documents has received a lot of attention and it is mostly referred to as doc2vec. Doc2vec is presented in detail in the following chapter entitled Document embedding algorithms.

## GloVe

GloVe is an acronym that stands for Global Vectors for Word Representation. GloVe is a model for obtaining vector representations for words that features a different take on the task compared to word2vec. It is developed as an open-source project at Stanford University [29].

The creators of GloVe distinguish between two model families for learning word vectors: global matrix factorization methods, such as latent semantic analysis and local (shallow) context window methods, such as the skip-gram model proposed in the context of word2vec. Global matrix factorization methods utilize low-rank approximations to decompose large matrices that capture statistical information (varying by application) about a corpus. For example, matrices of type “term-term” could be used where the rows and columns correspond to words and the entries correspond to the number of times a given word occurs in the context of another word. The main problem with such methods is that the most frequent words contribute a disproportionate amount to the similarity measure because of their frequent co-occurrence despite the fact that this does not necessarily mean much about their semantic relatedness. As a result, methods of this category perform relatively poorly on word analogy tasks, indicating a sub-optimal vector space structure, despite the fact that they efficiently leverage statistical information.

Shallow window-based methods are another approach in which representation learning is accomplished by making predictions within local context windows. Methods like skip-gram and continuous bag-of-words have the capacity to learn linguistic patterns as linear relationships between the word vectors and this is demonstrated through evaluation on a word analogy task. Scanning local context windows across the entire corpus fails to take advantage of the vast amount of repetition in the data and these methods do not operate directly on the co-occurrence statistics of the entire corpus.



The statistics of word occurrences in a corpus is the primary source of information available to all unsupervised methods for learning word representations and these methods try to generate word vectors that represent the meaning of these statistics. The GloVe model directly captures the global corpus statistics.

GloVe uses a matrix  $X$  of word-word co-occurrence counts, whose elements  $X_{ij}$  tabulate the number of times word  $j$  occurs in the context of word  $i$ . The sum of the elements in a row  $i$  is noted by  $X_i$  and is equal to the number of times any word appears in the context of word  $i$ . The ratio  $X_{ij}/X_i$  denoted by  $P_{ij}$  is the probability that word  $j$  appears in the context of word  $i$ . In GloVe, the starting point for word vector learning is computing the ratios of co-occurrence probabilities rather than the probabilities themselves. This is better illustrated with an example.

Consider two words  $i$  and  $j$  that exhibit a particular aspect of interest, e.g., take  $i = \text{ice}$  and  $j = \text{steam}$  in a corpus related to physics. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words,  $k$ . For words related to ice but not steam (for example the word solid), the ratio  $P_{ik}/P_{jk}$  is expected to be large while for words related to steam but not ice (say  $k = \text{gas}$ ) the ratio should be small. Similarly, for words either related to both ice and steam (e.g., water) or to neither of them (e.g., fashion) the ratio will be close to one. Compared to the raw probabilities, the ratio is better able to distinguish relevant words (solid and gas) from irrelevant words (water and fashion). The target optimization problem in GloVe is formalized with the following equation:

$$F(w_i, w_j, \mathbf{w}_k) = \frac{P_{ik}}{P_{jk}}$$

where  $w$  are word vectors and  $\mathbf{w}$  are separate context word vectors.

The function  $F$  should be applied to the word vectors and its output should approximate the probability ratio. Analysis of the properties of the optimization function and the characteristics of the vector space formulate the end equation. The result is a new global logbilinear regression model.

$$w_i^T \mathbf{w}_k + b_i + \mathbf{b}_k = \log(1 + X_{ik})$$



The weighting function is pivotal in the equation and should not overweight rare co-occurrences and very frequent ones. The selected function was:

$$f(x) = \begin{cases} (x/x_{max})^a & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

Optimal values for parameters were set to  $x_{max} = 100$  and  $a=3/4$ .

Results from experiments [29] indicate that GloVe and word2vec perform similarly and this is justified by the fact that they are essentially optimizing the same objective, i.e., they share a common base assumption that words with similar contexts have similar meanings. Despite the fact that word2vec does not explicitly utilize global statistics, its mode of operation by sequentially scanning the corpus does implicitly capture them. Regarding the computational complexity, GloVe scales on vocabulary size  $V$  because training is based on the co-occurrence matrix with contains all word pairs. Therefore, a simple upper bound to complexity would be  $O(V^2)$ . This is very practical as the vocabulary size does not grow with the size of the corpus.

### FastText

FastText is an open-source, free, lightweight library for learning text representations created and maintained by researchers in Facebook's AI Research (FAIR) lab [3]. The inspiration for fastText was the observation that previous techniques represent each word by a distinct vector without parameter sharing which is a serious limitation for morphologically rich languages, such as Finnish which has, for example, fifteen inflected cases for nouns. This means that many word forms may occur rarely or not at all in the training corpus and thus learning good word representations is hard. FastText uses character level information to improve vector representations which is beneficial for morphologically rich languages as many word formation follow rules.

The original paper on fastText [4] proposed an extension of the continuous skip gram model in which character n-grams are used and words are represented as the sum of n-gram vectors. This approach with subword information is shown to have good performance for nine languages of different morphologies. fastText differs from previous efforts on morphological word representations as it does not rely on the

morphological decomposition of words. Specifically, fastText tries to take into account the internal structure of words which the continuous skip gram model ignores by using a different scoring function  $s$ . Each word  $w$  is now represented as a bag of character  $n$ -gram. Special boundary symbols  $<$  and  $>$  indicate the beginning and end of words, thus allowing the distinction between prefixes and suffixes and other character sequences. For example, if we consider the word “where” and  $n=3$ , the resulting character trigrams will be:

$< whe, her, ere >$

and the special sequence (entire word) will be

$<where >$

In practice, all the  $n$ -grams for  $n$  greater or equal to 3 and smaller or equal to 6 (word length) are extracted. It must be noted that the sequence  $<her >$  as found in the word “where” is considered different from the sequence  $<her >$  as found in the word “her”.

Given a word  $w$  and a dictionary of  $n$ -grams with size  $G$ , the set of  $n$ -grams appearing in  $w$  is denoted by:

$$G_w \subset \{1, \dots, G\}$$

Each  $n$ -gram  $g$  is associated with a vector representation  $z_g$  and each word is represented by the sum of the vector representation of its  $n$ -grams. Thus, the scoring function becomes:

$$s(w, c) = \sum_{g \in G_w} z_g^T v_c$$

In this simple model, representations can be shared across words, thus allowing to learn reliable representations for rare words. Model memory requirements are upper bound using a hashing function that maps  $n$ -grams to integers in the range 1 to  $K$ . Ultimately, a word is represented by its index in the word dictionary and the set of hashed  $n$ -grams it contains.

Regarding the performance of fastText, in several tasks, it is on par with methods inspired by deep learning, while being much faster. For classification problems in



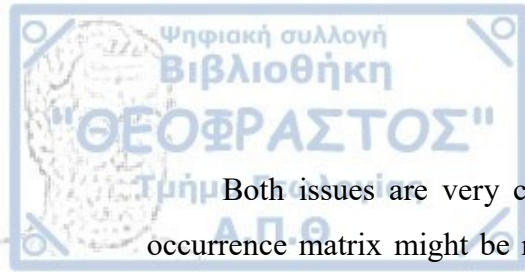
particular, fastText can be trained on more than one billion words in less than ten minutes using a standard multicore CPU, and it can classify half a million sentences among 312K classes in less than a minute [18].

The computational complexity of FastText is effectively the same as the complexity of the skip-gram variant of word2vec as the learning procedure is effectively the same. The added cost of fastText is the cost of splitting each word into its components, fetching their corresponding embedding vectors and compose them into the final word embedding. This is only a linear increase in cost so the complexity class remains the same. Regarding the memory complexity of fastText which is expected to be quite high, extensions to the library have been published that specifically address the issue for classifiers [19]. This is achieved by applying discriminative pruning which aims to keep only important features in the trained model, and by performing quantization of the weight matrices and hashing of the dictionary.

## WordRank

The researchers behind WordRank [18] take a different approach compared to the methods described in the previous sections, in the sense that they do not consider word-context co-occurrence counts as the basis for word embeddings. In WordRank the word embedding task is approached from a different perspective by formulating it as a ranking problem. That is, given a word  $w$ , the aim is to output an ordered list of context words such that words that co-occur with  $w$  appear at the top of the list. In other words, the importance does not lie in the particular scores but rather in the order between the context words.

Casting word embedding as ranking has two distinctive advantages. First, the method is discriminative rather than generative, so, instead of modeling the (potentially normalized) co-occurrence count directly, the aim is to only model the relative order of its values in each row. This fits naturally to popular word embedding tasks such as word similarity and analogy since instead of the likelihood of each word, we are interested in finding the most relevant words in a given context. The second advantage is the inherent robustness of the method to noise.



Both issues are very critical in the domain of word embeddings since the co-occurrence matrix might be noisy due to grammatical errors or unconventional use of language. This is particularly important in smaller document corpora collected from diverse sources. Additionally, WordRank enables sorting out the few most relevant words from very large vocabularies and thus works like a kind of attention mechanism. Experiments show that with 17 million tokens WordRank performs almost as well as existing methods using 7.2 billion tokens on a popular word similarity benchmark [17].



#### 4. Document embedding algorithms

Extending on the notion of word embeddings, document embeddings provide numerical vector representations for texts of variable length, even entire documents. These representations are then fed as input in methods/ algorithms for classification, similarity queries and other natural language processing tasks. This section presents the extension of word2vec for documents, appropriately named doc2ve.

##### Doc2vec

Despite their popularity, bag-of-words features have two major weaknesses: they disregard the order of words and thus different sentences can have exactly the same representation, as long as they contain the same words. Furthermore, they have very little sense about the semantics of words or more formally the distances between words. As a result, for example, the words “powerful,” “strong” and “Paris” are equally distant despite the fact that semantically, “powerful” should be closer to “strong” than “Paris.”

Following the success of word embedding methods such as word2vec, researchers have pursued extensions to go beyond word level to phrase-level or sentence-level representations. Example of simple approaches were using a weighted average of all the words in a document or combining the word vectors in an order given by a sentence parse tree using matrix-vector operations. Both simple approaches mentioned have weaknesses. The first one loses the word order much like the standard bag-of-words models do while the second one that relies on parsing works only for sentences.

To counteract these drawbacks, the authors of [20] propose Paragraph Vector, an unsupervised framework that learns continuous distributed vector representations for variable-length pieces of texts (from single phrase/sentences to entire documents). The original name they suggested (Paragraph Vector) illustrates the variability in text length. However, the method has since become known as Doc2vec, in the sense that it can provide vector embeddings for entire documents. Unlike some of the previous approaches, it is general, applicable to texts of any length and does not require task-specific tuning of the word weighting function or parse trees. Two separate models are proposed, in a mode analogous to word2vec, Distributed Memory Model of Paragraph Vectors (PV-DM) and Paragraph Vector with Distributed Bag of Words (PV-DBOW).

### Paragraph Vector: A Distributed Memory model (PV-DM)

This approach for learning paragraph vector is based on the one for learning word vectors. More specifically, much like word vectors are asked to contribute to a prediction task about the next word in a sentence, the paragraph vectors are asked to contribute to the prediction task of the next word given many contexts sampled from the paragraph. The prediction learning task is the reason why word vectors eventually capture semantics despite the fact that they are randomly initialized. Accordingly, paragraphs and words are mapped to unique vectors (columns in matrix D and in matrix W, respectively).

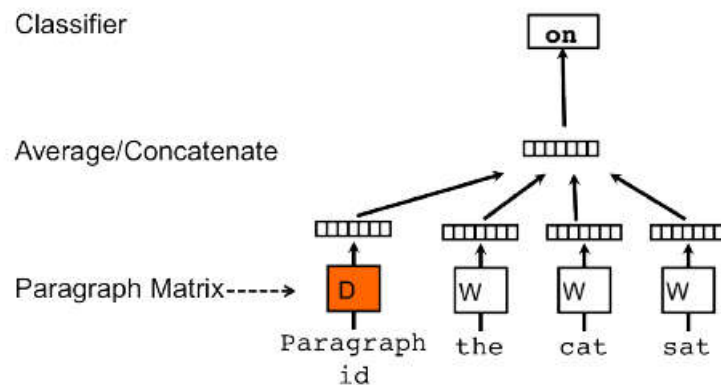


Figure 7: The operation of PV-DM. The word vectors along with the paragraph vector are used to predict the following word [Le & Mikolov, 2014].

The resulting paragraph vector representation is trained to be useful for predicting the following words in a paragraph by concatenating it with several word vectors from the same paragraph (Figure 7). Both word vectors and paragraph vectors are trained using stochastic gradient descent and backpropagation. Paragraph vectors are unique for each paragraph but shared across all contexts generated from the same paragraph while word vectors are shared, i.e., the vector for “powerful” is the same for all paragraphs. Contexts are fixed-length and sampled from a sliding window over the paragraph.

The operation of the algorithm that generates the vectors can be summed up in two stages: the first is training to get word vectors W, softmax weights and parameters and paragraph vectors D on already seen paragraphs and the second one is the inference stage where vectors for unseen paragraphs are computed by adding more columns to the

matrix  $D$  while holding all else fixed. For this purpose, a standard classifier is used (e.g., logistic regression).

Paragraph Vector without word ordering: Distributed bag of words (PV-DBOW)

PV-DM concatenates the paragraph vector with the word vectors in order to predict the next word in a text window. In contrast, Distributed Bag of Words (DBOW) ignores the context words in the input, but forces the model to predict words randomly sampled from the paragraph in the output. The operation of the model is illustrated in Figure 8. At each iteration of stochastic gradient descent, a random word is sampled from the also randomly selected text window and a classification task is formed given the Paragraph Vector.

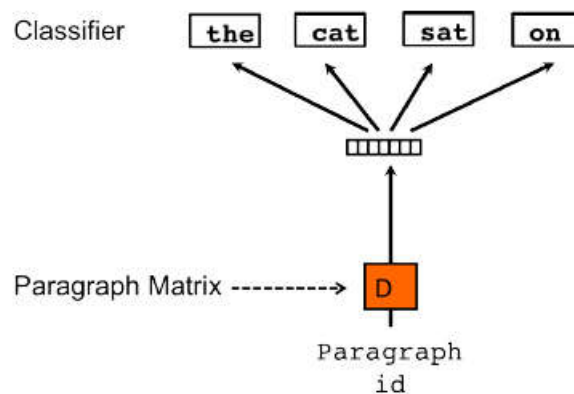


Figure 8: The operation of PV-DBOW. The paragraph vector is trained to predict the words in a small window [Le & Mikolov, 2014]

This model is conceptually simple and similar to the Skip-gram model in word vectors. Compared to PV-DM it stores less data and while it can work well alone, the combination of both paragraph vectors (one learned with PV-DM and one with PV-DBOW) is usually more consistent across many tasks.

Paragraph vectors generated with the methods described in the previous two sections have several advantages. Firstly, they do not require labeled data and thus can work well for tasks that do not have enough such data. Most importantly, they address the weaknesses of bag-of-words models. Paragraph vectors retain the semantics of the

words (i.e., in this space, “powerful” is closer to “strong” than to “Paris”) and they consider the order of words (albeit in a small context) in the same way that an n-gram model with a large n would do. Compared to a theoretical bag-of-n-grams model, Le and Mikolov [20] note that their paragraph vectors are superior in the sense that a bag of n-grams model would create a very high-dimensional representation that tends to generalize poorly.

## 5. Python toolkits and libraries for natural language processing

Python is a very popular programming language and natural language processing is one of the primary ways it is used. This section provides an overview of several well-known and used toolkits and libraries written in Python that relate to natural language processing.

### Natural Language Toolkit (NLTK)

The Natural Language Toolkit [27], also referred to as NLTK is a platform for building programs that work with human language data, written in Python. NLTK is a suite of libraries for symbolic and statistical natural language processing that was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania. It is free and open source software distributed under the Apache License and hosted on Github. NLTK is suitable for students and professionals alike and is available for Windows, Mac OS X, and Linux. It is also accompanied by a free book written by NLTK creators [2], which introduces newcomers to natural language processing with Python.

According to the creators of NLTK, they chose Python as the programming language for implementation because of its simplicity, its syntax transparency and its abilities in string handling. Python combines multiple programming paradigms and has a shallow learning curve. Its standard library is very extensive and includes powerful tools for graphical programming, numerical processing, and web connectivity. NLTK contains the following components:

1. **Code:** libraries/ modules for all functions required in natural language processing (50,000 lines of code). Popular functions include corpus readers, tokenizers, stemmers, taggers, parsers, semantic interpretation, clusterers, evaluation metrics, etc.
2. **Corpora:** more than 30 annotated data sets widely used in NLP.
3. **Documentation:** a 400-page book, articles, reviews, API documentation.

Basic functions of NLTK include:



- `nltk.word_tokenize()`: outputs a list of strings/tokens appearing in the argument text.
- `nltk.pos_tag()`: outputs a list of word/ part of speech tuples.
- `nltk.corpus.stopwords.words('english')`: outputs a list of stop words for the English language.

NLTK also includes dictionaries and a thesaurus, directly accessible via the command line and can output word definitions, synonyms, antonyms and sample usages. It also includes functions that estimate if two words are related.

Apart from working with words, NLTK can analyze and visualize sentence structure. With the corresponding modules, NLTK provides answers to the following questions:

1. How can we use a formal grammar to describe the structure of an unlimited set of sentences?
2. How do we represent the structure of sentences using syntax trees?
3. How do parsers analyze a sentence and automatically build a syntax tree?

Indeed, systematic aspects of meaning are much easier to capture once the structure of sentences has been identified. Parse trees automatically generated by NLTK are an excellent tool for sentence structure visualization and ambiguity management. A well-known example is the analysis of the sentence:

While hunting in Africa, I shot an elephant in my pajamas.

By using NLTK and defining a simple grammar, the sentence can be analyzed in its parts.

```
>>> groucho_grammar = nltk.CFG.fromstring("""
... S -> NP VP
... PP -> P NP
... NP -> Det N | Det N PP | 'I'
... VP -> V NP | VP PP
... Det -> 'an' | 'my'
... N -> 'elephant' | 'pajamas'
... V -> 'shot'
... P -> 'in'
... """)
```

Based on this grammar, the sentence can be analyzed in one of two ways, depending on whether the prepositional phrase “in my pajamas” describes the elephant or the shooting event.



```
>>> grouche_grammar = nltk.CFG.fromstring("""
... S -> NP VP
... PP -> P NP
... NP -> Det N | Det N PP | 'I'
... VP -> V NP | VP PP
... Det -> 'an' | 'my'
... N -> 'elephant' | 'pajamas'
... V -> 'shot'
... P -> 'in'
... """)
```

The corresponding parse trees are shown in Figure 9 as they are generated by NLTK.

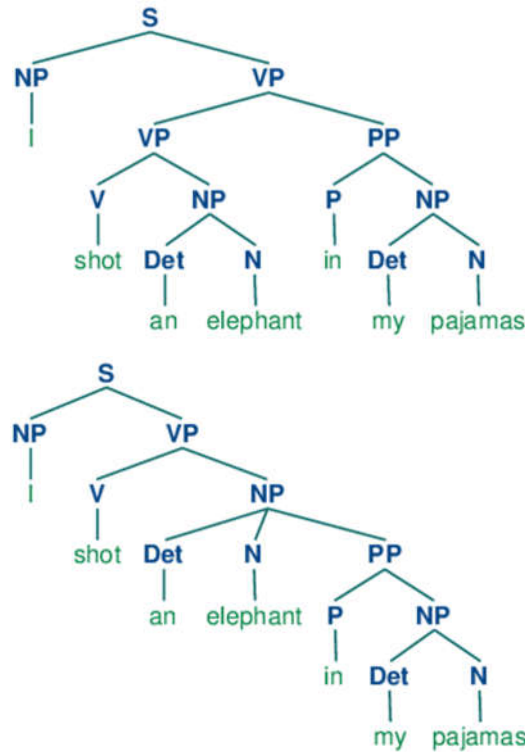


Figure 9: Parse trees generated by NLTK for the two interpretations of the sentence [Bird et al., 2009]

### Pattern for Python

Pattern is a Python package for web mining, natural language processing, machine learning and network analysis which offers a collection of tools commonly used in applications that harness the Web. Pattern is free and open source software licensed under BSD and is organized in separate modules/ packages that can be chained. Pattern is written in pure Python, mainly for readability. The main packages in Pattern are [35]:

- **pattern.web**: this package includes tools for web data mining, i.e. tools for downloading content and using web services such as the ones offered by search engines and Wikipedia. It also includes an HTML parser, a parser for PDF documents, a web crawler, and a webmail interface.





- `pattern.en`: this package is essentially a fast, regular expressions-based shallow parser for English that can identify sentence constituents such as verbs, nouns and adjectives. The original implementation included a parser for the Dutch language and developers can add support for other languages.
- `pattern.search`: this module includes an N-gram pattern matching algorithm for objects of the Sentence class. Search queries can include a mixture of words, phrases, part-of-speech-tags, taxonomy terms (e.g., pet = dog, cat or goldfish) and operators (such as +, \*, ()) to extract relevant information.
- `pattern.vector`: this module includes the tools that compute TF-IDF, distance metrics and perform dimension reduction. It also includes a hierarchical and a k-means clustering algorithm, some simple classifiers and tools for feature selection and K-fold cross validation.
- `pattern.graph`: this module supports graph data structures useful for example for modeling semantic networks. The module has algorithms for shortest path computation, subgraph partitioning, eigenvector centrality and betweenness centrality.
- `pattern.metrics`: this module supports descriptive statistics functions, such as functions for accuracy, precision and recall.
- `pattern.db`: this module includes the tools for working with CSV files and SQLITE/ MYSQL databases.

### LibShortText

Short texts include titles, questions, sentences and short messages. The approaches to classification and analysis must consider the special properties related to their small length for instance the fact that words in them are most likely distinct. Existing procedures may need to be altered to apply in shorter texts which are generally easier for investigation and experimentation. The authors of [39] developed an open source tool licensed under BSD called LibShortText whose main features are:

1. It is optimized for short texts i.e. it is more efficient for large-scale short-text classification compared to traditional tools.





2. The default options are selected to guarantee the best performance for user applications.
3. Error analysis is performed via an interactive tool at each text level detail.

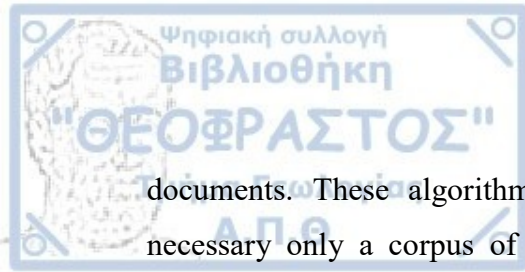
LibShortText is written in Python for simplicity and extensibility with portions in C/C++ for speed/efficiency. The workflow in a LibShortText application follows three steps, each corresponding to an included library.

1. *libshorttext.converter*: the bag-of-word model is used to generate features. Short texts can be pre-processed by tokenization and optionally by stemming and removing stop-word. The library both unigram and bigram features.
2. *libshorttext.classifier*: after the user chooses how features will be represented (options include binary, word count or TF-IDF), the library generates sparse feature vectors and calls a linear-classification package for training/testing. Multi-class classification is also supported.
3. *libshorttext.analyser*: this is the interactive tool used to conduct error analysis in both the overall performance level and the level of analysis of each feature of a short text.

## Gensim

Gensim [30] is a free and open source vector space modeling and topic modeling toolkit implemented in Python. It was created in 2009 and it is distributed under the GNU LGPLv2.1 license. Since its creation, it has become a reference point both for researchers in related fields and for companies and is also used in commercial products. It is supported by the company RaRe Technologies and it is hosted on GitHub.

In the words of its creator, Radim Řehůřek, Gensim is “the most robust, efficient and hassle-free piece of software to realize unsupervised semantic modeling from plain text”. The algorithms in Gensim, such as Word2Vec, FastText, Latent Semantic Analysis, Latent Dirichlet Allocation, etc, automatically discover the semantic structure of documents by examining statistical co-occurrence patterns within a corpus of training



documents. These algorithms are unsupervised, which means no human input is necessary only a corpus of plain text documents is required. After these statistical patterns have been discovered, any plain text document of any length can be succinctly expressed in the new, semantic representation and queried for topical similarity against words, phrases or documents [30].

The basic features of Gensim are:

- *Memory independence*: the entire corpus used in training does not need to reside in the computer's RAM all at any one time. Therefore, large corpora (such as web-like ones) are an option for training.
- *Memory sharing*: models that have completed their training can be saved to disk and loaded back for experiments and multiple processes can share the same data.
- Efficient implementations for several popular vector space algorithms.
- Input/ output wrappers and readers from several popular data formats.
- Fast similarity queries for documents in their semantic representation.

Gensim was designed to be straightforward to use and easy to learn for developers, with an impressive API that is great for prototyping. Additionally, because Gensim operates in a streaming fashion, one document at a time, the size of the corpus is not a hindering factor.

### Core concepts in Gensim

This section summarizes some of the core concepts in Gensim which are required to follow the details of code implementation presented in the next chapter.

#### *Corpus*

A corpus in Gensim is a collection of digital documents. Corpora serve as input for model training and models utilize the training corpus to initialize their internal parameters. No human intervention is required (e.g. annotations or tagging by hand

which are very costly and cannot be performed in bulk in reasonable time frames). Indeed, Gensim focuses solely on unsupervised models.

Corpora in Gensim also serve as documents to organize. After training, a topic model can be used to extract topics from new documents (not already contained in the training corpus).

### *Vector space model*

As it was discussed in previous sections, in a Vector Space Model, each document is represented by an array of features. For example, a single feature may be thought of as a question-answer pair:

*How many times does a specific word appear in the document? Zero.*

*How many paragraphs does the document consist of? Two.*

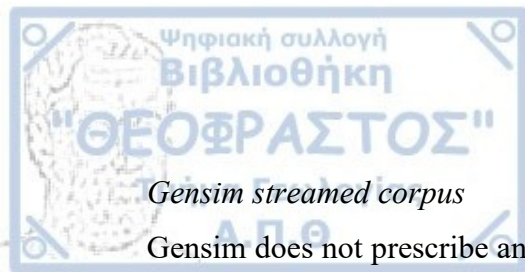
*How many fonts does the document use? Five.*

The question is usually represented only by its integer id (such as 1, 2 and 3 here), so that the representation of this document becomes a series of pairs like (1, 0.0), (2, 2.0), (3, 5.0).

This sequence of answers can be thought of as a vector (in this case a 3-dimensional dense vector) and are the same for all documents. As a result, vector similarity can be interpreted as document similarity. The selection of questions and the degree to which they correlate with real world similarity is therefore critical.

### *Gensim sparse vector*

For space saving reasons, Gensim does not store vector elements that are equal to zero. Each vector element is 2-tuple of (feature\_id, feature\_value) and all missing values are resolved to zero so documents are potentially represented by sparse vectors.

*Gensim streamed corpus*

Gensim does not prescribe any specific corpus format. A corpus is simply a sequence of sparse vectors, as described above and any object that when iterated over, successively yields such sparse bag-of-word vectors is acceptable in Gensim. This flexibility enables users to create their own corpus classes that stream vectors directly from disk even on the fly.

*Model, Transformation*

In Gensim the term model is used to refer to the code and associated data (parameters) required to transform one document representation to another. As discussed above, documents in Gensim correspond to vectors so a model is essentially a transformation from one vector space to another. The parameters of this transformation are learned from the training corpus and data computed based on these parameters (i.e., the trained model) can be written to disk and then reloaded and reused either for further training or new transformations.

## 6. A practical application of Gensim doc2vec for similarity estimation between Wikipedia articles

### Application goals and Problem definition

This section describes a practical example of using the word2vec and doc2vec implementation in order to model data, produce analogy when given a set of words and find the less relevant one between a bunch of words. The objective is also to be able to execute similarity queries among the articles of Greek Wikipedia and obtain a list of article rankings based on similarity indices.

### Stakeholders

This functionality can help scientists or interested parties in word/document similarities in the Greek language. Word and document embeddings, as discussed in previous chapters, can be also used as input in various machine learning algorithms for automatic summarization, machine translation, sentiment analysis, speech recognition etc. This functionality, consequently, can help scientists or interested parties in exploring the methods mentioned above in the Greek language.

### Methodology

To ensure project application goals are met the following methodology shown by the diagram below was utilized. The first step is to identify user requirements, and the second step is to identify the appropriate machine learning algorithms. The following step is to implement the algorithms chosen and finally to present and evaluate the results. This framework allows us to select the right algorithms to accomplish goals set (please see below).

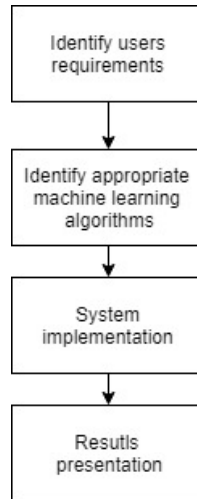


Figure 1. Methodology diagram

## Requirements

This section describes the requirements of the system. The ecosystem should consist of two concrete applications.

1. The first application should be able to identify:
  - Semantic similarities between words.
    - A less semantically relevant word between a bunch of words.
    - Analogies between one word and a tuple (two words semantically connected).
2. The second application should be able to identify:
  - Semantic similarities between documents.
    - The closest semantically related documents within a collection (corpus) given one document as input.

## Design

In the section to follow we discuss the method in which the requirements can be met. Initially, the methodology is decided upon, followed by the structure of the proposed solution. Depicted in the following diagram is the overall structure of the application code.

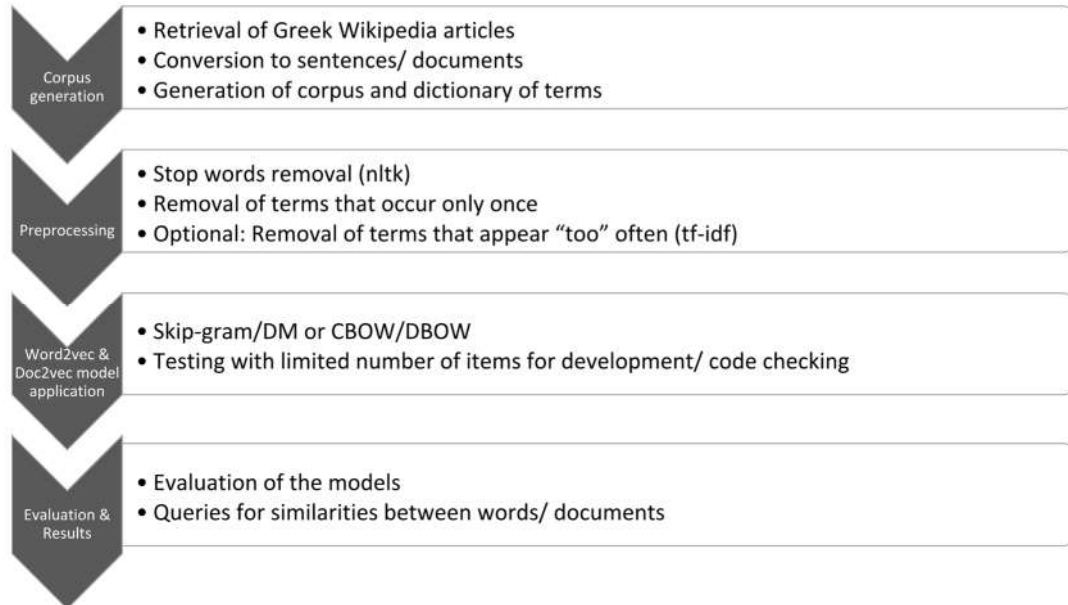
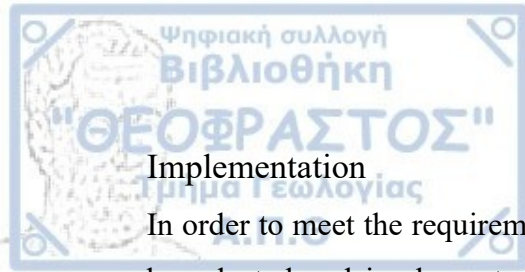


Figure 2: Python application code building blocks

Our initial step is to create the corpus of this application. For this purpose obtaining the source documents and then converting these documents into sentences gives us the opportunity to construct a dictionary. Secondly, preprocessing methods are needed to be applied as they transform the original sequence of characters to a cleaner form. For this purpose we will convert all characters to lowercase. One of the major forms of preprocessing is to filter out useless data. In natural language processing, useless words are referred to as stop words. Therefore, we will remove all stop words and terms that appear only once in each document.

Moving on, we are going to divide our application into two sub applications which both use different algorithms and compute different tasks. The first sub application should utilize the wordToVec algorithm in order to transform a word into a vector. The second application should utilize the docToVec algorithm with the intention of assigning vectors to documents. Both the selected algorithms contain two different architectures which will be applied in each case. We then are going to test them for development/ code checking with a limited number of items.

Our last step is to evaluate the accuracy of our models and present our results revealing similarities between words and documents.



### Implementation

In order to meet the requirements some appropriate machine learning algorithms should be selected and implemented as discussed previously. For the purpose of this thesis, Word2vec and Doc2vec algorithms were selected. We initially executed commands directly on the command line interpreter and then aggregated them into files for batch execution which, in turn allowed for us to observe the results.

Next, we ran all experiments in Python 3.7 (64 bit version) on a Windows 10 machine. Initially we experimented on applying simpler algorithms such as tf-idf and word2vec to the data at hand, to observe the capabilities of Gensim and to incorporate them into our code further on.

The resulting Python code is modular and reusable, in the sense that blocks of code generate intermediate results that can be used for other experiments of the same type or slightly different processing of the article texts.

### Obtaining the source documents

As in many natural language processing applications, we utilized the latest dump of Wikipedia articles. Wikipedia offers a wide variety of downloadable files including articles, list of all articles titles, media metadata, article to article links etc. all widely used in various research projects.

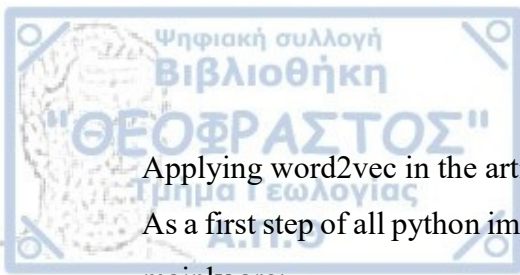
The latest dump of Greek Wikipedia articles can be found in the link below:

<https://dumps.wikimedia.org/elwiki/latest/>

Wikipedia dumps, by definition, are the most recent Wikipedia's platform updates. Therefore this dataset is consistently updated. In this thesis we conducted experiments using the full texts in two concrete dates.

The size of the download file (~310MB) and the number of articles (~145,000) is convenient for experimentation. Gensim can process the compressed bz2 file directly, therefore no deflating is required.





Applying word2vec in the articles of Greek Wikipedia/ Word analogy queries

As a first step of all python implementations specific packages need to be imported. These mainly are:

- *Multiprocessing* that allows the programmer to fully leverage multiple processors on a given machine.
- *wikicorpus* that constructs a corpus from a Wikipedia (or other MediaWiki-based) database dump. Wikicorpus uses *multiprocessing* internally to parallelize the work and process the dump more quickly.
- *Word2vec* that implements the word2vec family of algorithms, using highly optimized C routines, data streaming and Pythonic interfaces.

One of the first things, as mentioned in Chapter 2, required for natural language processing (NLP) tasks is a corpus. In linguistics and NLP, corpus refers to a collection of texts. So next we are going to create the corpus.

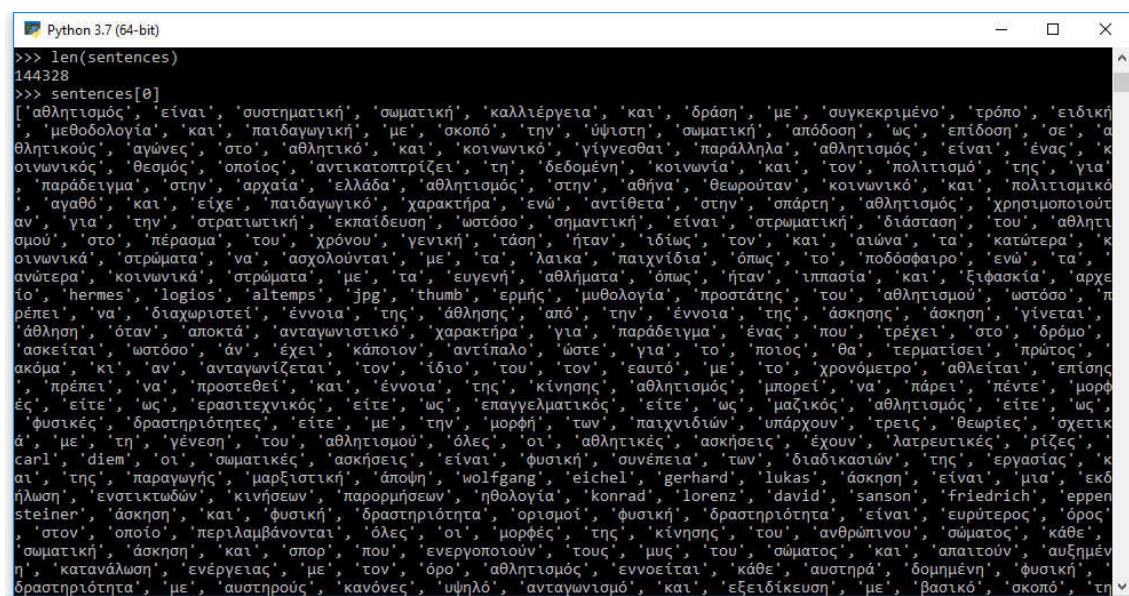
```
#WikiCorpus (wiki = WikiCorpus("D:\code\data\elwiki-latest-pages-articles.xml.bz2",  
lemmatize=False, dictionary={}))
```

Main parameters [30]:

- *fname (str)*: Path to where the Wikipedia dump file is stored.
- *processes (int, optional)*: Number of processes to run, defaults to  $\max(1, \text{number of cpu} - 1)$ .
- *lemmatize (boolean)*: If the parameter lemmatization is set to True, it uses lemmatization instead of simple regexp tokenization. Defaults to True if you have the pattern package installed.
- *dictionary (Dictionary, optional)*: If a dictionary is not provided, Gensim scans the corpus once, to determine its vocabulary, which takes a long time.
- *article\_min\_tokens (int, optional)*: Minimum tokens in article. Article will be ignored if number of tokens is less.
- *token\_min\_len (int, optional)*: Minimal token length.
- *token\_max\_len (int, optional)*: Maximal token length.
- *lower (boolean, optional)*: It converts all text to lower case, if it is set to True.

As the corpus was created we then used `get_texts()` that iterates over the dump, yielding a list of tokens for each article that passed the length and namespace filtering parameters (if any – not used in this example). The length parameter can be used, for example, to exclude short articles (with fewer than 50 words) and the namespace filtering to exclude, e.g. the discussion pages. This function uses multiprocessing internally to parallelize the work and process the dump more quickly. In our experiments, this command took about 5 minutes to execute for a dump with close to 144,000 articles.

The results of the command are shown in Figure 3 where each item in the sentences list is a list of tokens (words) from the given Wikipedia article. The length of the list (i.e. the number of elements) is also shown (144328).



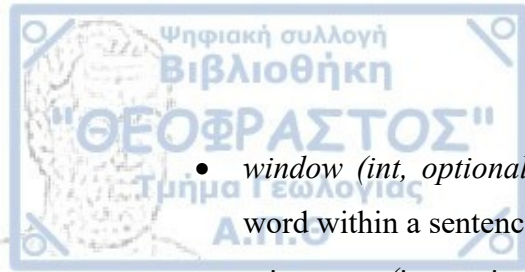
```
Python 3.7 (64-bit)
>>> len(sentences)
144328
>>> sentences[0]
['αθλητισμός', 'είναι', 'συστηματική', 'σωματική', 'καλλιέργεια', 'και', 'δράση', 'με', 'συγκεκριμένο', 'τρόπο', 'ειδική', 'μεθοδολογία', 'και', 'παιδαγωγική', 'με', 'σκοπό', 'την', 'ύψιστη', 'σωματική', 'απόδοση', 'ως', 'επίδοση', 'σε', 'αθλητικούς', 'αγώνες', 'στο', 'αθλητικό', 'και', 'κοινωνικό', 'γίνεσθαι', 'παράλληλα', 'αθλητισμός', 'είναι', 'ένας', 'κοινωνικός', 'θεσμός', 'οποίος', 'αντικατοπτρίζει', 'τη', 'δεδομένη', 'κοινωνία', 'και', 'τον', 'πολιτισμό', 'της', 'για', 'παράδειγμα', 'στην', 'αρχαία', 'ελλάδα', 'αθλητισμός', 'στην', 'αθήνα', 'θεωρούνταν', 'κοινωνικό', 'και', 'πολιτισμικό', 'αγαθό', 'και', 'είχε', 'παιδαγωγικό', 'χαρακτήρα', 'ενώ', 'αντίθετα', 'στην', 'σπάρτη', 'αθλητισμός', 'χρησιμοποιούσαν', 'για', 'την', 'στρατιωτική', 'εκπαίδευση', 'ωστόσο', 'σημαντική', 'είναι', 'στρωματική', 'διάσταση', 'του', 'αθλητισμού', 'στο', 'πέρασμα', 'του', 'χρόνου', 'γενική', 'τάση', 'ήταν', 'ιδίως', 'τον', 'και', 'αιώνα', 'τα', 'κατώτερα', 'κοινωνικά', 'στρώματα', 'να', 'ασχολούνται', 'με', 'τα', 'λαϊκά', 'παιχνίδια', 'όπως', 'το', 'ποδόσφαιρο', 'ενώ', 'τα', 'ανώτερα', 'κοινωνικά', 'στρώματα', 'με', 'τα', 'ευγενή', 'αθλήματα', 'όπως', 'ήταν', 'ιππασία', 'και', 'ξίφασκία', 'αρχαίως', 'hermes', 'logios', 'altempis', 'jrg', 'thumb', 'ερμής', 'μυθολογία', 'προστάτης', 'του', 'αθλητισμού', 'ωστόσο', 'πρέπει', 'να', 'διαχωριστεί', 'έννοια', 'της', 'άθλησης', 'από', 'την', 'έννοια', 'της', 'άσκησης', 'άσκηση', 'γίνεται', 'άθληση', 'όταν', 'αποκτά', 'ανταγωνιστικό', 'χαρακτήρα', 'για', 'παράδειγμα', 'ένας', 'που', 'τρέχει', 'στο', 'δρόμο', 'ασκείται', 'ωστόσο', 'αν', 'έχει', 'κάποιον', 'αντίπαλο', 'ώστε', 'για', 'το', 'ποιος', 'θα', 'τερματίσει', 'πρώτος', 'ακόμα', 'κι', 'αν', 'ανταγωνίζεται', 'τον', 'ίδιο', 'του', 'τον', 'εαυτό', 'με', 'το', 'χρονόμετρο', 'αθλείται', 'επίσης', 'πρέπει', 'να', 'προσεβεί', 'και', 'έννοια', 'της', 'κίνησης', 'αθλητισμός', 'μπορεί', 'να', 'πάρει', 'πέντε', 'μορφές', 'είτε', 'ως', 'ερασιτεχνικός', 'είτε', 'ως', 'επαγγελματικός', 'είτε', 'ως', 'μαζικός', 'αθλητισμός', 'είτε', 'ως', 'φυσικός', 'δραστηριότητες', 'είτε', 'με', 'την', 'μορφή', 'των', 'παιχνιδιών', 'υπάρχουν', 'τρεις', 'θεωρίες', 'σχετικά', 'με', 'τη', 'γένεση', 'του', 'αθλητισμού', 'όλες', 'οι', 'αθλητικές', 'ασκήσεις', 'έχουν', 'λατρευτικές', 'ρίζες', 'carl', 'diem', 'οι', 'σωματικές', 'ασκήσεις', 'είναι', 'φυσική', 'συνέπεια', 'των', 'διαδιδασκίων', 'της', 'εργασίας', 'και', 'της', 'παραγωγής', 'μαρξιστική', 'άποψη', 'wolfgang', 'eichel', 'gerhard', 'lukas', 'άσκηση', 'είναι', 'μια', 'εκδήλωση', 'ενστικτωδών', 'κινήσεων', 'παρορμήσεων', 'ηθολογία', 'konrad', 'lorenz', 'david', 'sanson', 'friedrich', 'erpensteiner', 'άσκηση', 'και', 'φυσική', 'δραστηριότητα', 'ορισμοί', 'φυσική', 'δραστηριότητα', 'είναι', 'ευρύτερος', 'όρος', 'στον', 'οποίο', 'περιλαμβάνονται', 'όλες', 'οι', 'μορφές', 'της', 'κίνησης', 'του', 'ανθρώπινου', 'σώματος', 'κάθε', 'σωματική', 'άσκηση', 'και', 'σπορ', 'που', 'ενεργοποιούν', 'τους', 'μυς', 'του', 'σώματος', 'και', 'απαιτούν', 'αυξημένη', 'κατανάλωση', 'ενέργειας', 'με', 'τον', 'όρο', 'αθλητισμός', 'εννοείται', 'κάθε', 'αυστηρά', 'δομημένη', 'φυσική', 'δραστηριότητα', 'με', 'αυστηρούς', 'κανόνες', 'υψηλό', 'ανταγωνισμό', 'και', 'εξειδίκευση', 'με', 'βασικό', 'σκοπό', 'τη
```

Figure 3: Results of token extraction from Wikipedia articles

## #Word2Vec

We first need to set the parameters for the word2vec modeling (`params = {'size': 200, 'window': 10, 'min_count': 10, 'workers': max(1, multiprocessing.cpu_count() - 1), 'sample': 1E-3,}`) and then train the word2vec model (`Word2Vec(sentences, **params)`). Gensim's implementation of word2vec takes many parameters. The main ones are [26]:

- `size (int, optional)`: Dimensionality of the word vectors.



- *window (int, optional)*: Maximum distance between the current and predicted word within a sentence.
- *min\_count (int, optional)*: Ignores all words with total frequency lower than this.
- *workers (int, optional)*: Use these many worker threads to train the model. This is typically set equal to the number of available cores.
- *sg ({0, 1}, optional)*: Training algorithm: 1 for skip-gram; otherwise CBOW.
- *hs ({0, 1}, optional)*: If 1, hierarchical softmax will be used for model training. If 0, and negative is non-zero, negative sampling will be used.
- *negative (int, optional)*: If > 0, negative sampling will be used, the int for negative specifies how many “noise words” should be drawn (usually between 5-20). If set to 0, no negative sampling is used.
- *max\_vocab\_size (int, optional)*: Limits the RAM during vocabulary building; if there are more unique words than this, then the infrequent ones are pruned.
- *sample (float, optional)*: The threshold for configuring which higher-frequency words are randomly down-sampled, useful range is (0, 1e-5).
- *iter (int, optional)*: Number of iterations (epochs) over the corpus.

### Modeling with doc2vec and performing document similarity queries

In this experiment, we created a corpus of Wikipedia articles and used Gensim’s doc2vec implementation to model them and perform similarity queries based on search terms or between documents within the corpus.

Same as in the previous example, the first step is to generate a corpus from Wikipedia articles. For this example, we first used the entire dump of Wikipedia articles and then included a subset of them in our corpus for the similarity tests in order for the model training to be quick. The results are directly applicable to the entire list of articles, with appropriate training time. The most important commands are as follows:

**#join(sentences[])**

In order for the training tasks to be completed in a logical amount of time we used a subset of our original data by connecting the first 6 documents contained into the

Wikipedia dump (`data = [' '.join(sentences[0]), ' '.join(sentences[1]), ' '.join(sentences[2]), ' '.join(sentences[3]), ' '.join(sentences[4]), ' '.join(sentences[5])]`).

As a result data consists of the text of the first six articles (in Greek). Their titles translate to Sport (Αθλητισμός), Occitan language (Οξιτανική γλώσσα), Eleftherios Venizelos (Ελευθέριος Βενιζέλος), Cyprus (Κύπρος), Geography (Γεωγραφία) and Nicosia (Λευκωσία).

### #Preprocessing

Pre-processing can be referred to as the process of converting data to something a computer can comprehend. One of the major forms of pre-processing is to filter out useless data. In natural language processing, as discussed in Chapter 2, useless words (data) are referred to as stop words. We can remove them easily, by storing a list of words considered to be stop words. NLTK (Natural Language Toolkit) in python has a list of stop words stored in 16 different languages. Unfortunately, it does not contain yet a list in the Greek language. For the purpose of this thesis our stop word list will be created by the user in the Greek language (manual user implementation of list) (`stoplist = set('από με στη στο στα το τα της που του τη για'.split())`).

Continuing, we will convert all characters to lowercase and we will remove words that appear only once (`texts = [[word for word in document.lower().split() if word not in stoplist] ]`).

### #Doc2vec

Doc2vec consists of two models that use different architectures. Gensim's Doc2vec implementation [26] takes many parameters. These mainly are:

- `dm ({1,0}, optional)` – This parameter defines the training algorithm to be implemented. In the case of `dm=1`, distributed memory (PV-DM) is used. Otherwise, distributed bag of words (PV-DBOW) is employed.
- `vector_size (int, optional)` – This corresponds to the dimensionality of the feature vectors.
- `window (int, optional)` – The maximum distance between the current and predicted word within a sentence.



- *min\_count (int, optional)* – Ignores all words with total frequency lower than this.
- *workers (int, optional)* – Use these many worker threads to train the model (=faster training with multicore machines).
- *epochs (int, optional)* – Number of iterations over the corpus.
- *dm\_mean ({1,0}, optional)* – If 0, use the sum of the context word vectors. If 1, use the mean. Only applies when dm is used in non-concatenative mode.
- *dbow\_words ({1,0}, optional)* – If set to 1 trains word-vectors (in skip-gram fashion) simultaneous with DBOW doc-vector training; If 0, only trains doc-vectors (faster).

For the purposes of this thesis both architectures were implemented with the following values selected as optimal.

- PV-DBOW (*Doc2Vec(dm=0, dbow\_words=1, vector\_size=200, window=8, min\_count=19, epochs=10, workers=cores)*)
- PV-DM (*Doc2Vec(dm=1, dm\_mean=1, vector\_size=200, window=8, min\_count=19, epochs=10, workers=cores)*)

## Results-Evaluation of models

This section briefly discusses the evaluation techniques performed and the results obtained from the implementation of the two algorithms.

### Results-Evaluation of Word2vec algorithm

In order to test the trained model, we first ran some classic similarity queries and subsequently used an analogy test, which is a commonly used automated way to evaluate models, or compare algorithms. We set a list of analogy tasks by hand and computed the accuracy of our model (63.9%). To do so, a method that computes cosine similarity between a simple mean of the projection weight vectors of the given words and the vectors for each word in the model and finds the top n most similar words was used. Positive words contribute positively towards the similarity, whereas negative words contribute



negatively. This accuracy level is considered satisfactory given the volume of our input data. Some of the results on analogy tasks are depicted below:

1. ***model.most\_similar(positive=['γυναίκα', 'βασιλιάς'], negative=['άντρας'])***

*βασιλιά  $\approx 0.609$*

*βασιλίτσα  $\approx 0.596$*

*σύζυγο  $\approx 0.533$*

*στέψη  $\approx 0.520$*

*δυναστεία  $\approx 0.511$*

The words reference **Result 1** translate from Greek to English as follows: γυναίκα (woman), βασιλιάς (king), άντρας (man), βασιλίτσα (queen), σύζυγο (wife), στέψη (coronation), δυναστεία (dynasty). It must be mentioned that due to declination the form of the word may change in greek as in the case of βασιλιά which also means king.

2. ***model.most\_similar(positive=['αγόρι', 'παμπάς'], negative=['μαμά'])***

*κορίτσι  $\approx 0.703$*

*παιδάκι  $\approx 0.698$*

*μωρό  $\approx 0.694$*

*κοριτσάκι  $\approx 0.689$*

*αγοράκι  $\approx 0.647$*

The words reference **Result 2** translate from Greek to English as follows: αγόρι (boy), παμπάς (father), μαμά (mother), κορίτσι (girl), παιδάκι (child), μωρό (baby), κοριτσάκι (little girl), αγοράκι (little boy).

Continuing, another common method (*doesn't match*) was used. This function determines which word doesn't match the context of the others. The results are shown below:

- ***model.doesnt\_match('κάπνισμα περπάτημα κολύμπι ποδήλατο'.split())***

*“κάπνισμα”*

The words reference **Result 1** translate from Greek to English as follows: κάπνισμα (smoking), περπάτημα (walking), κολύμπι (swimming), ποδήλατο (riding). As a result smoking was returned.

- ***model.doesnt\_match('πρωινό βραδινό δημοκρατικά μεσημεριανό'.split())***

*“δημοκρατικά”*

The words reference **Result 2** translate from Greek to English as follows: πρωινό (breakfast), βραδινό (dinner), δημητριακά (cereal), μεσημεριανό (lunch). As a result cereal was returned.

1. `model.doesnt_match('πορτοκάλια μήλα φράουλες φασκόμηλο'.split())`  
“φασκόμηλο”

The words reference **Result 3** translate from Greek to English as follows: πορτοκάλια (oranges), μήλα (apples), φράουλες (strawberries), φασκόμηλο (sage). As a result sage was returned.

### Results of Doc2vec- Document similarity queries

In Doc2vec's original paper[16] experiments on several benchmark datasets were presented in order to evaluate the algorithm and so as to demonstrate the advantages of Paragraph Vector. These mainly were sentiment analysis tasks and text classification tasks where Doc2vec has proved to outperform previous related algorithms. Implementing this, however, exceeds the scope of this thesis and can be left for future work.

In the experiments conducted in its original paper, each paragraph vector was a combination of two vectors: one learned by the standard paragraph vector with distributed memory (PV-DM) and one learned by the paragraph vector with distributed bag of words (PVDDBOW). It is discussed that PV-DM alone works well for most tasks (with state-of-art performances), but its combination with PV-DBOW is usually more consistent across many tasks [20].

In our experiment, as discussed in the previous chapter, we conducted both the Distributed Bag of Words version of Paragraph Vector (PV-DBOW) and the Distributed Memory Model of Paragraph Vectors (PV-DM). Both these models produced similar results with some examples shown below. A method (`most_similar`) that computes cosine similarity between a simple mean of the projection weight vectors of the given documents was used. Documents may be specified as vectors, integer indexes of trained document vectors, or if the documents were originally presented with string tags, by the corresponding tags:

1. As a first example using the PV-DM model, we executed the following query:  
`model.docvecs.most_similar(positive=['Τεχνητή νοημοσύνη'])` with the

intention to find the most similar articles to Artificial intelligence within Wikipedia.

('Θεωρητική Πληροφορική', 0.7000911235809326),

('Εμπειρα συστήματα', 0.6822481751441956),

('Ταυτοχρονισμός', 0.6772329211235046),

('Μηχανική όραση', 0.6662312746047974),

('Νευρωνικό δίκτυο', 0.6657699346542358),

('Μηχατρονική', 0.6579104661941528),

('Έλεγχος μοντέλων', 0.6565818786621094),

('CrypTool', 0.6496784090995789),

('Γλώσσα περιγραφής υλικού', 0.648236870765686),

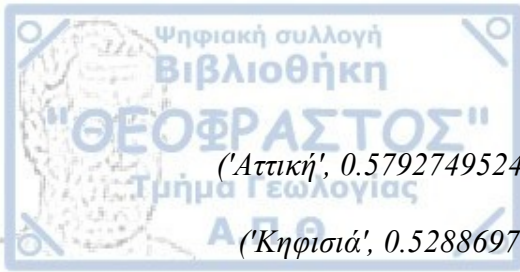
('Μηχανική μάθηση', 0.6471858024597168)]

The words reference **Result 1** translate from Greek to English as follows: "Τεχνητή νοημοσύνη" (Artificial Intelligence), Θεωρητική Πληροφορική (Theoretical computer science), Έμπειρα συστήματα (Expert system), Ταυτοχρονισμός (Concurrency), Μηχανική όραση (Computer vision), Νευρωνικό δίκτυο (Neural Network), Μηχατρονική (Mechatronics), Έλεγχος μοντέλων (Model checking), CrypTool (CrypTool), Γλώσσα περιγραφής υλικού (Hardware description language), Μηχανική μάθηση (Machine Learning) .

Executing the same query using the PV-DBOW model we obtained similar results. Θεωρητική Πληροφορική (Theoretical computer science), Μηχανική μάθηση (Machine Learning), Μηχανική όραση (Computer vision) and Έμπειρα συστήματα (Expert system) were some of the articles obtained in both occasions.

2. As a second example we executed the following query with the intention of finding the most similar articles to Athens in Greece  
(*model.docvecs.most\_similar(positive=["Ελλάδα", "Αθήνα"])*). Again PV-DBOW and PV-DM models returned similar results with the most important ones being the following:





(*Αττική*', 0.5792749524116516)

(*Κηφισιά*', 0.5288697481155396)

(*Αρχαία Αθήνα*', 0.5189750790596008)

(*Προσφυγικό ζήτημα (Μικρασιατική Καταστροφή)*', 0.5028291940689087)

(*Βόρεια προάστια Αθηνών*', 0.5021364092826843)

(*Ελευσίνα Ιπποθοωντίδας*', 0.500443696975708)

(*Ηλιούπολη Αττικής*', 0.4993937611579895)

(*Δυτικά προάστια Αθηνών*', 0.4962661862373352)

The words reference **Result 2** translate from Greek to English as follows: "*Αττική*" (Attiki) which is a neighborhood of Athens, *Κηφισιά* (Kifissia) which is a suburb of Athens, *Αρχαία Αθήνα* (Classical Athens) which corresponds to the city of Athens during the classical period of Ancient Greece, *Προσφυγικό ζήτημα (Μικρασιατική Καταστροφή)* (Greek refugees), *Βόρεια προάστια Αθηνών* (Athens northern suburbs), *Ελευσίνα Ιπποθοωντίδας* (Elefsina), *Ηλιούπολη Αττικής* (Ilioupoli) which is a suburban municipality in the southeastern part of the Athens urban area, *Δυτικά προάστια Αθηνών* (Western suburbs of Athens).





## 7. Conclusion

This thesis has provided a comprehensive review of word and document embedding algorithms, both from a theoretical and a practical perspective. It must be highlighted, that the related research field is not only really active, but numerous exciting and promising areas of application have emerged in the last few years, and will continue to emerge, with every application being trained/ tailored for a number of languages.

The results in performing natural language processing tasks after training are impressive and consistent. This thesis was an attempt to apply word and document embeddings in the Greek language. However, during the length of this paper several limitations were met as the Greek language has a vast and rich vocabulary and the volume of available resources in the Greek language is still quite limited. From research conducted a suggestion for future work would be a combination of word embedding algorithms with knowledge graphs such as Wordnet (that can be found in the greek language). This could enhance the word embeddings produced in this thesis by learning word embeddings that incorporate the semantic information from the resource and lead to even better results.

As the volume of information available online increases and the hardware performance improves and develops, further training and experiments may lead to a variety of natural language processing tasks in the Greek language.





## References

- [1] Aggarwal, C. C., & Zhai, C. (2012). A survey of text classification algorithms. In Mining text data (pp. 163-222). Springer, Boston, MA.
- [2] Bird, S., Klein, E., & Loper, E. (2009). Natural language processing with Python: analyzing text with the natural language toolkit. O'Reilly Media, Inc.
- [3] Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T. (2016). fastText. URL: <https://research.fb.com/fasttext/> . Accessed on: 17/10/18.
- [4] Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016). Enriching word vectors with subword information. arXiv preprint arXiv:1607.04606.
- [5] Brownlee, J. (2017). "A Gentle Introduction to the Bag-of-Words Model". URL: <https://machinelearningmastery.com/gentle-introduction-bag-words-model/>. Accessed on: 17/09/18.
- [6] Dahl, G. E., Yu, D., Deng, L., & Acero, A. (2012). Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. IEEE Transactions on audio, speech, and language processing, 20(1), 30-42.
- [7] Dechter, R. (1986). Learning while searching in constraint-satisfaction-problems. Proceedings of the Fifth AAAI National Conference on Artificial Intelligence, 178-183.
- [8] Deng, L. (2014). A tutorial survey of architectures, algorithms, and applications for deep learning. APSIPA Transactions on Signal and Information Processing, 3.
- [9] D'Souza, J., "An Introduction to Bag-of-Words in NLP". URL: <https://medium.com/greyatom/an-introduction-to-bag-of-words-in-nlp-ac967d43b428>. Accessed on: 9/09/18.
- [10] Firth, J.R. (1957). *"A synopsis of linguistic theory 1930-1955"*. *Studies in Linguistic Analysis*. Oxford: Philological Society: 1-32. Reprinted in F.R. Palmer, ed. (1968). *Selected Papers of J.R. Firth 1952-1959*. London: Longman.
- [11] Gebre, B. G., Zampieri, M., Wittenburg, P., & Heskes, T. (2013). Improving native language identification with tf-idf weighting. In the 8th NAACL Workshop on Innovative Use of NLP for Building Educational Applications (BEA8) (pp. 216-223).
- [12] Giel, A., & Diaz, R. (2016). Document embeddings via recurrent language models. University of Stanford, Tech. Rep.

- [13] Goldberg, Y., & Levy, O. (2014). word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722.
- [14] Guo, Y., Liu, Y., Oerlemans, A., Lao, S., Wu, S., & Lew, M. S. (2016). Deep learning for visual understanding: A review. *Neurocomputing*, 187, 27-48.
- [15] Guthrie, D., Allison, B., Liu, W., Guthrie, L., & Wilks, Y. (2006, May). A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)* (pp. 1-4).
- [16] Harris, Z. (1954). "Distributional structure". *Word* 10 (23), 146–162.
- [17] Ji, S., Yun, H., Yanardag, P., Matsushima, S., & Vishwanathan, S. V. N. (2015). Wordrank: Learning word embeddings via robust ranking. arXiv preprint arXiv:1506.02761.
- [18] Joulin, A., Grave, E., Bojanowski, P., & Mikolov, T. (2016). Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759.
- [19] Joulin, A., Grave, E., Bojanowski, P., Douze, M., Jégou, H., & Mikolov, T. (2016). Fasttext.zip: Compressing text classification models. arXiv preprint arXiv:1612.03651.
- [20] Le, Q., & Mikolov, T. (2014, January). Distributed representations of sentences and documents. In *International Conference on Machine Learning* (pp. 1188-1196).
- [21] Lilleberg, J., Zhu, Y., & Zhang, Y. (2015, July). Support vector machines and word2vec for text classification with semantic features. In *Cognitive Informatics & Cognitive Computing (ICCI\* CC), 2015 IEEE 14th International Conference on* (pp. 136-140). IEEE.
- [22] Lv, Y., & Zhai, C. (2011, October). Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM international conference on Information and knowledge management* (pp. 7-16). ACM.
- [23] Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.
- [24] Mikolov, T., Le, Q. V., & Sutskever, I. (2013). Exploiting similarities among languages for machine translation. arXiv preprint arXiv:1309.4168.
- [25] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* (pp. 3111-3119).
- [26] Minar, M. R., & Naher, J. (2018). Recent Advances in Deep Learning: An Overview. arXiv preprint arXiv:1807.08169.

- [27] NLTK Project. (2019). Natural Language Toolkit. URL: <https://www.nltk.org/>. Accessed on: 13/09/18.
- [28] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., & Vanderplas, J. (2011). Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), 2825-2830.
- [29] Pennington, J., Socher, R., & 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).
- [30] Řehůřek, R. (2018). Gensim: Topic Modelling for Humans. URL: <https://radimrehurek.com/gensim/>. Accessed on: 18/09/18.
- [31] Robertson, S. E., Walker, S., Jones, S., Hancock-Beaulieu, M. M., & Gatford, M. (1995). Okapi at TREC-3. *Nist Special Publication Sp*, 109.
- [32] Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4), 333-389.
- [33] Samuel, A. L. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3), 210-229.
- [34] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural networks*, 61, 85-117.
- [35] Smedt, T. D., & Daelemans, W. (2012). Pattern for python. *Journal of Machine Learning Research*, 13(Jun), 2063-2067.
- [36] Spärck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1), 11-21.
- [37] Turnbull, D. (2015). BM25 The Next Generation of Lucene Relevance. URL: <https://opensourceconnections.com/blog/2015/10/16/bm25-the-next-generation-of-lucene-relevation/>. Accessed on: 21/09/18.
- [38] Vincent, J. (2018). Google 'fixed' its racist algorithm by removing gorillas from its image-labeling tech. URL: <https://www.theverge.com/2018/1/12/16882408/google-racist-gorillas-photo-recognition-algorithm-ai>. Accessed on: 14/09/18.
- [39] Yoo, J. Y., & Yang, D. (2015). Classification scheme of unstructured text document using TF-IDF and naive Bayes classifier. *Advanced Science and Technology Letters*, 111(50), 263-266.

- [40] Young, T., Hazarika, D., Poria, S., & Cambria, E. (2018). Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence magazine*, 13(3), 55-75.
- [41] Yu, H., Ho, C., Juan, Y., & Lin, C. (2013). Libshorttext: A library for short-text classification and analysis. *Rapport interne, Department of Computer Science, National Taiwan University. Software available at <http://www.csie.ntu.edu.tw/~cjlin/libshorttext>.*
- [42] Zaragoza, H., Craswell, N., Taylor, M. J., Saria, S., & Robertson, S. E. (2004, November). Microsoft Cambridge at TREC 13: Web and Hard Tracks. In *TREC* (Vol. 4, pp. 1-1).
- [43] Zhang, J., & Zong, C. (2015). Deep neural networks in machine translation: An overview. *IEEE Intelligent Systems*, 30(5), 16-25.