ALEXANDRU IOAN CUZA UNIVERSITY OF IAŞI

FACULTY OF COMPUTER SCIENCE

MASTER THESIS

Word2Vec for Romanian Language

Proposed by

Colban Mihai

Session: *July, 2017*

Scientific coordinator

Associate Professor, PhD, Iftene Adrian

ALEXANDRU IOAN CUZA UNIVERSITY OF IAŞI

FACULTY OF COMPUTER SCIENCE

Word2Vec for Romanian Language

Colban Mihai

Session: *July, 2017*

Scientific coordinator

Associate Professor, PhD, Iftene Adrian

Abstract

The Word2Vec model and application by Tomas Mikolov (together with his colleagues at Google) have attracted a great amount of attention in the recent years. The vector representations of words learned by word2vec models have been shown to carry semantic meanings and are useful in various natural language processing tasks.

In this thesis we are using a large Romanian text corpus gathered from a Wikipedia database dump, training it using Word2Vec and analysing the output results.

DECLARAȚIE PRIVIND ORIGINALITATE ŞI RESPECTAREA DREPTURILOR DE AUTOR

Prin prezenta declar că Lucrarea de disertație cu titlul *“Word2Vec for Romanian Language”* este scrisă de mine şi nu a mai fost prezentată niciodată la o altă facultate sau instituție de învățământ superior din țară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului: toate fragmentele de text reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi dețin referința precisă a sursei:

- reformularea în cuvinte proprii a textelor scrise de către alți autori deține referința precisă;
- codul sursă, imagini etc. preluate din proiecte open-source sau alte surse sunt utilizate cu respectarea drepturilor de autor şi dețin referințe precise;
- rezumarea ideilor altor autori precizează referința precisă la textul original.

03 iulie 2017

Absolvent *Colban Mihai*

(semnătura în original)

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de disertație cu titlul *“Word2Vec for Romanian Language”*, codul sursă al programelor şi celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea Alexandru Ioan Cuza Iaşi să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de disertație.

Iaşi, *03 iulie 2017*

Absolvent *Colban Mihai*

(semnătura în original)

Contents

Abstract **3**

1. Introduction **7**

1.1 Word2Vec 7

1.2 Continuous Bag of Word (CBOW) 9

1.3 N-gram model 13

1.4 Skip-grams 14

1.5 In this thesis 18

2. Technologies used **20**

3. Implementation details **21**

3.1 Architecture 21

3.2 Preprocessing 21

3.2.1 Text extraction and cleaning 21

3.2.2 Text Normalization 24

3.2.3 Stopword Removal 25

3.2.4 Stemming 26

3.3 Python implementation 29

3.3.1 Data trainer 29

3.3.2 Plot generator 30

3.3.2.1 PCA 31

3.3.2.2 TSNE 33

3.4 .Net implementation 35

4. Metrics **38**

4.1 Word2phrase 38

4.2 Word2vec 38

4.3 Cython 39

5. Study Cases 40

6. Conclusion 44

6.1 Future work 44

7. Bibliography 45

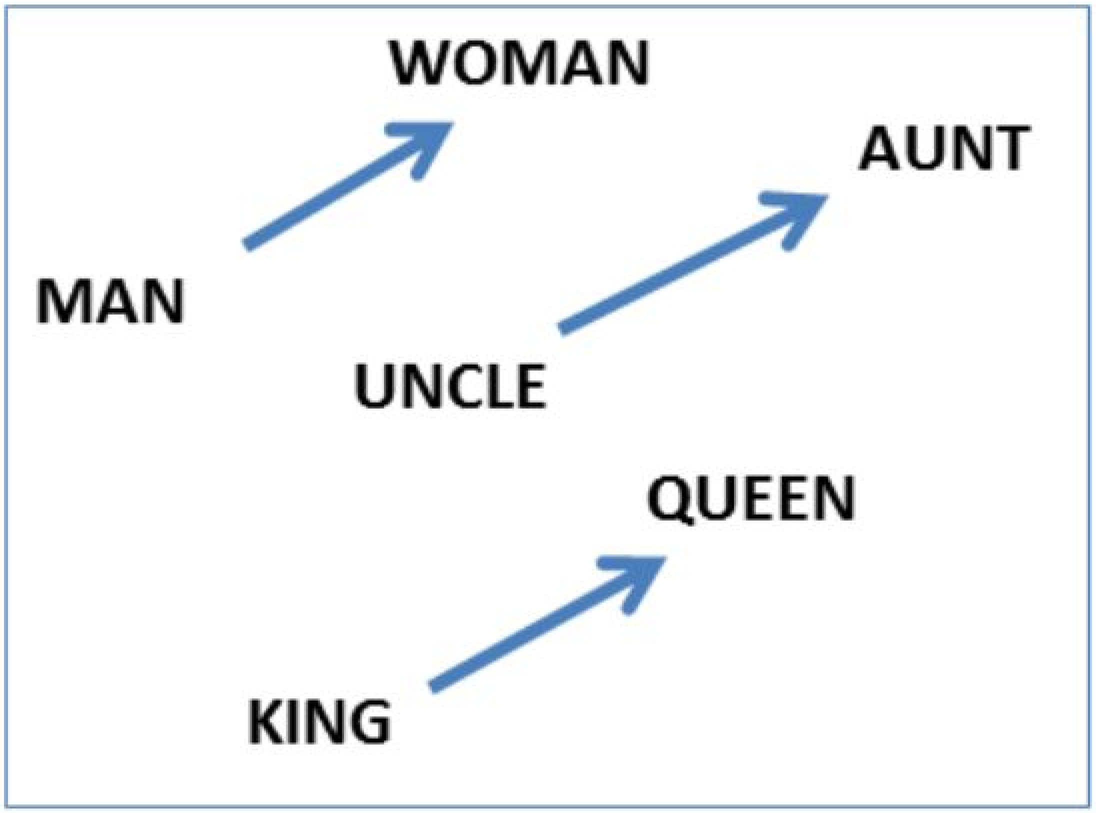
1. Introduction

1.1 Word2Vec

The word meaning and relationships between words are encoded spatially. There are two main learning algorithms in word2vec:

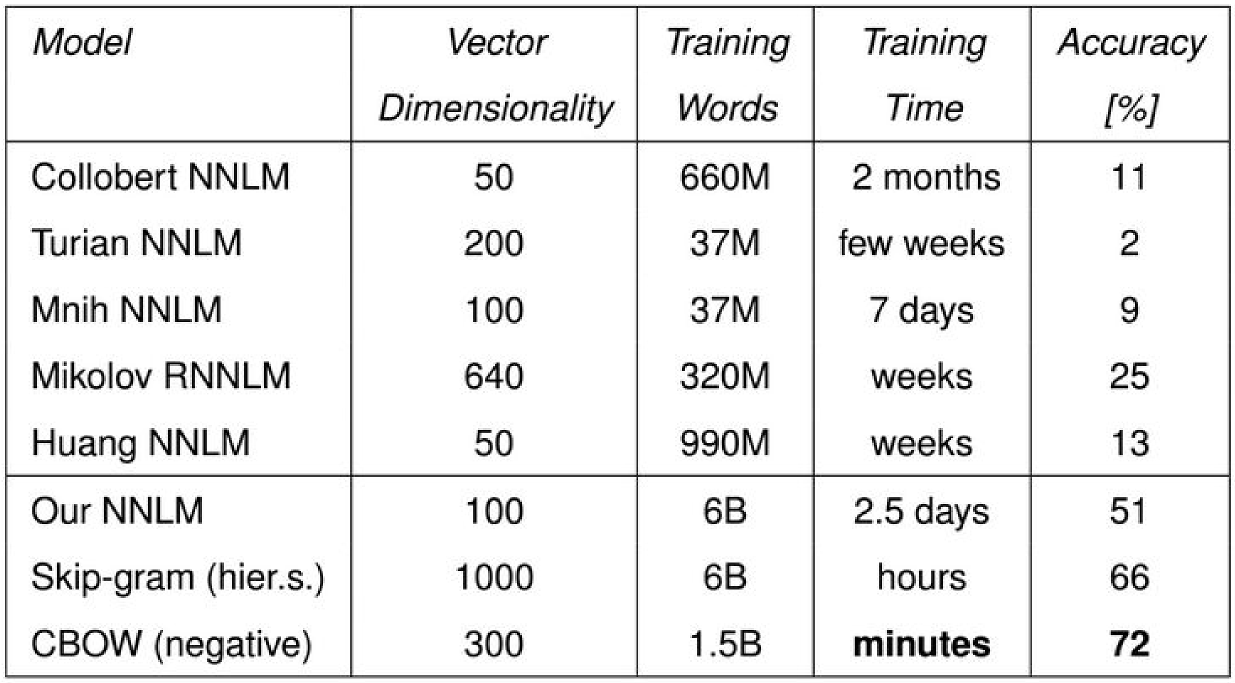
- Continuous bag-of word model which works well on syntactic and training is faster;
- Continuous skip-gram model which works well on semantic but the training is slower.

These algorithms will be presented in the next subchapters of this.



[Mikolov et al., NAACL HLT, 2013]

The basic idea is that semantic vectors (such as the ones provided by Word2Vec) should preserve most of the relevant information about a text while having relatively low dimensionality which allows better machine learning treatment than straight one-hot encoding of words. Another advantage of topic models is that they are unsupervised so they can help when labeled data is scarce. Say you only have one thousand manually classified blog posts but a million unlabeled ones. A high quality topic model can be trained on the full set of one million. If you can use topic modeling-derived features in your classification, you will be benefitting from your entire collection of texts, not just the labeled ones. Another advantage of Word2Vec is the training time, this can be seen in the following image.



Model	*Vector Dimensionality*	*Training Words*	*Training Time*	*Accuracy [%]*
Collobert NNLM	50	660M	2 months	11
Turian NNLM	200	37M	few weeks	2
Mnih NNLM	100	37M	7 days	9
Mikolov RNNLM	640	320M	weeks	25
Huang NNLM	50	990M	weeks	13
Our NNLM	100	6B	2.5 days	51
Skip-gram (hier.s.)	1000	6B	hours	66
CBOW (negative)	300	1.5B	**minutes**	**72**

[Mikolov et al., NAACL HLT, 2013]

1.2 Continuous Bag of Word (CBOW)

Continuous Bag of Words model is a simplifying representation used in natural language processing and information retrieval. In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model has also been used for computer vision. [1]

The bag-of-words model is commonly used in methods of document classification where the frequency of each word is used as a feature for training a classifier. The main ideea is that using context words, we can predict the center word.

```
Probability(“Max ( ? ) to watch birds.” → “likes” )
```

The goal of this model is to train a weight-matrix (**W**) that can satisfy the below formula:

$$argmax_W \{Minimize(|\, time - softmax(pr(time \,|\, "Max", "to", "watch", "birds"))\,| \,;\, W)\}$$

Loss-function (using cross-entropy method)

$$E = -\log p(W_t | W_{t-C} \ldots W_{t+C})$$

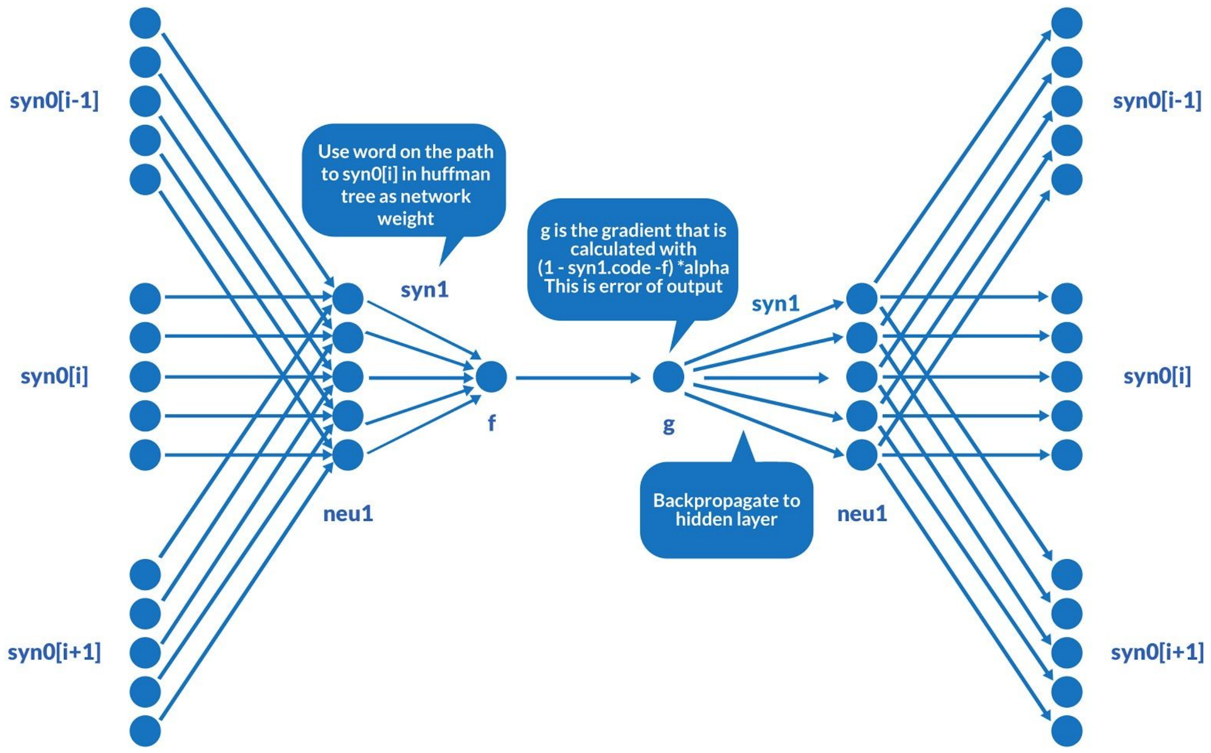
The following example models a text document using bag-of-words. Here are two simple text inputs:

```
    (1) Max likes to watch birds. Mary likes birds too.


    (2) Max also likes to watch football games.
```

Based on these two text documents, a list is constructed as follows:

```
[
    "Max",
    "Likes",
    "To",
    "Watch",
    "Birds",
    "Also",
    "Football",
    "Games",
    "Mary",
    "Too"
]
```


syn0[i-1]
syn0[i]
syn0[i+1]
Use word on the path to syn0[i] in huffman tree as network weight
syn1
neu1
f
g is the gradient that is calculated with (1 - syn1.code -f) *alpha This is error of output
g
Backpropagate to hidden layer
syn1
neu1
syn0[i-1]
syn0[i]
syn0[i+1]

In practice, the Bag-of-words model is mainly used as a tool of feature generation. After transforming the text into a "bag of words", we can calculate various measures to characterize the text. The most common type of characteristics, or features calculated from the Bag-of-words model is term frequency, namely, the number of times a term appears in the text. For the example above, we can construct the following two lists to record the term frequencies of all the distinct words:

```
(1) [1, 2, 1, 1, 2, 0, 0, 0, 1, 1]

(2) [1, 1, 1, 1, 0, 1, 1, 1, 0, 0]
```

Each entry of the lists refers to count of the corresponding entry in the list (this is also the histogram representation). For example, in the first list (which represents document 1), the first two entries are “1, 2”. The first entry corresponds to the word “Max” which is the first word in the list, and its value is “1” because “Max” appears in the first document 1 time. Similarly, the second entry corresponds to the word “likes” which is the second word in the list, and its value is “2” because "likes" appears in the first document 2 times. This list (or vector) representation does not preserve the order of the words in the original sentences, which is just the main feature of the Bag-of-words model. This kind of representation has several successful applications, for example email filtering.

However, term frequencies are not necessarily the best representation for the text. Common words like “the”, “a”, “to” are almost always the terms with highest frequency in the text. Thus, having a high raw count does not necessarily mean that the corresponding word is more important. To address this problem, one of the most popular ways to “normalize” the term frequencies is to weight a term by the inverse of document frequency, or tf–idf. Additionally, for the specific purpose of classification, supervised alternatives have been developed that take into account the class label of a document. Lastly, binary (presence/absence or 1/0) weighting is used in place of frequencies for some problems.

1.3 N-gram model

Bag-of-word model is an orderless document representation — only the counts of words mattered. For instance, in the above example “Max likes to watch birds. Mary likes birds too”, the bag-of-words representation will not reveal the fact that a person’s name is always followed by the verb “likes” in this text. As an alternative, the n-gram model can be used to store this spatial information within the text. Applying to the same example above, a **bigram** model will parse the text into following units and store the term frequency of each unit as before.

```
[
    "Max likes",
    "likes to",
    "to watch",
    "watch birds",
    "Mary likes",
    "likes birds",
    "birds too",
]
```

Conceptually, we can view bag-of-word model as a special case of the n-gram model, with $n = 1$.

A common alternative to the use of dictionaries is the hashing trick, where words are directly mapped to indices with a hashing function. By mapping words to indices directly with a hash function, no memory is required to store a dictionary. Hash collisions are typically dealt with by using freed-up memory to increase the number of hash buckets. In practice, hashing greatly simplifies the implementation of bag-of-words models and improves their scalability.

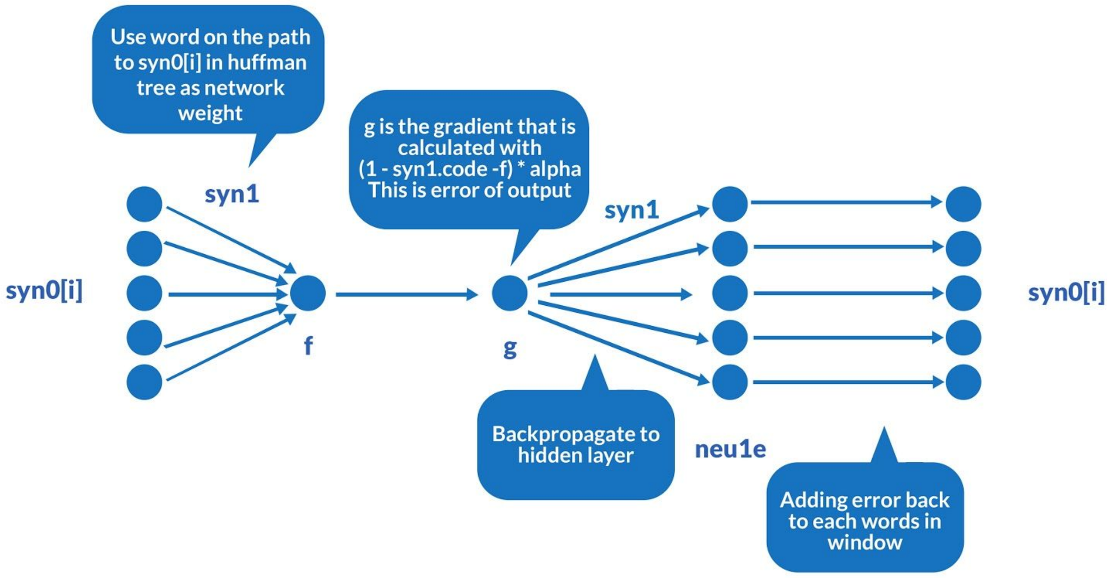
1.4 Skip-grams

In the field of computational linguistics, in particular language modeling, skip-grams are a generalization of n-grams in which the components (typically words) need not be consecutive in the text under consideration, but may leave gaps that are skipped over. They provide one way of overcoming the data sparsity problem found with conventional n-gram analysis.

We define k-skip-n-grams for a sentence $w_1 \dots w_m$ to be the set:

$$\{w_{i_1}, w_{i_2}, \dots w_{i_n} \mid \sum_{j=1}^{n} i_j - i_{j-1} < k\}$$

Skip-grams reported for a certain kip distance *k* allow a total of *k* or less skips to construct the n-gram. As such, "5-skip-n-gram" results include 5 skips, 4 skips, 3 skips, 2 skips, 1 skip and 0 skips (usually n-grams are formed from adjacent words).


Use word on the path to syn0[i] in huffman tree as network weight
g is the gradient that is calculated with (1 - syn1.code -f) * alpha This is error of output
syn1
syn1
syn0[i]
f
g
syn0[i]
Backpropagate to hidden layer
neu1e
Adding error back to each words in window

Here is an actual sentence example showing 2-skip-bigrams and trigrams compared to standard bigrams and trigrams consisting of adjacent words for the sentence:

```
    Max likes to watch birds.
```

Bigrams:

```
 [
     "Max likes",
     "likes to",
     "to watch",
     "watch birds"
]
```

2-skip-bigrams:

```
[
     "Max likes",
     "Max to",
     "Max watch",
     "likes to",
     "likes watch",
     "likes birds",
     "to watch",
     "to birds",
     "watch birds"
]
```

Trigrams:

```
[
     "Max likes to",
     "likes to watch",
     "to watch birds"
]
```

2-skip-trigrams

```
 [
     "Max likes to",
     "Max likes watch",
     "Max likes birds",
     "Max to watch",
     "Max to birds",
     "Max watch birds",
     "likes  to watch",
     "likes to birds",
     "likes watch birds",
     "to watch birds"
]
```

In this example, over three times as many 2-skip-tri-grams were produced than adjacent tri-grams and this trend continues the more skips that are allowed. A typical sentence of ten words, for example, will produce 8 trigrams, but 80 4-skip-tri-grams. Sentences that are 20 words long have 18 tri-grams and 230 4-skip-tri-grams.

Bigrams					
Sentence Length	**Bi-grams**	**1-skip**	**2-skip**	**3-skip**	**4-skip**

5	4	7	9	10	10
10	9	17	24	30	35
15	14	29	30	50	60
20	19	37	54	70	85

Trigrams					
Sentence Length	**Bi-grams**	**1-skip**	**2-skip**	**3-skip**	**4-skip**
5	3	7	10	10	10
10	8	22	40	60	80
15	13	37	70	110	155
20	18	53	100	160	230

Number of n-grams vs. number of k-skip n-grams produced

For an n word sentence, the formula for the number of trigrams with exactly k skips is given by the following formula:

$$(n - (k+2))(k+1),\ for\ n > k + 3$$

But, we use k-skip gram to mean k skips or less for an n word sentence, which can be written as:

$$n \sum_{i=i}^{k+1} i - \sum_{i=i}^{k+1} i(i + 1), \, for \; n > k+2$$

$$= \frac{(k+1)(k+2)}{6}(3n - 2k - 6)$$

1.5 In this thesis

The basic idea behind this thesis is to test the word embedding that are generated by Word2Vec for a large text corpus that is in Romanian language. Although there exist a couple of implementations of the Word2Vec for different languages (especially for different types of english documents) we could not find any for the Romanian language.

2. Technologies used

We intend to write a python script, using Word2Vec library, for processing the text corpus from the Wikipedia database dump and we will use a C# application to read the trained data and show the most relevant word matches.

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse.

For running the python scripts we used the Anaconda distribution. Anaconda is a freemium open source distribution of the Python and R programming languages for large-scale data processing, predictive analytics, and scientific computing, that aims to simplify package management and deployment.

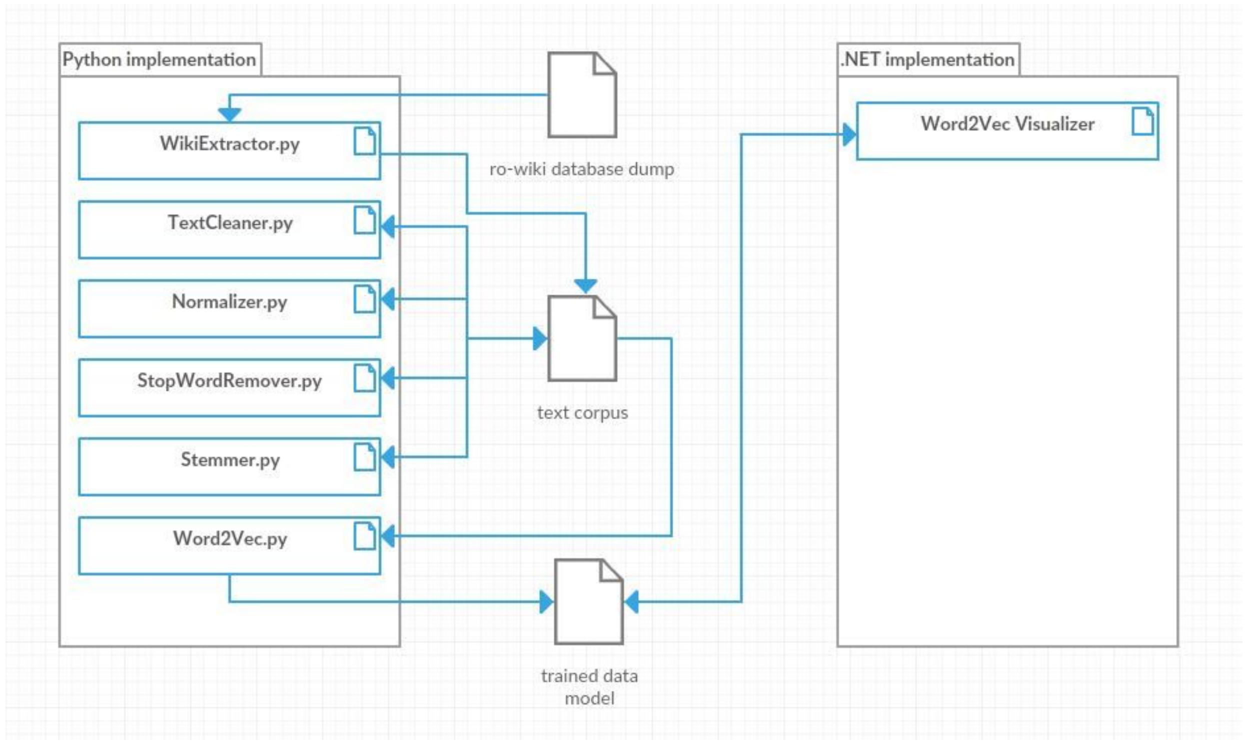
C# is a hybrid of C and C++, it is a Microsoft programming language developed to compete with Sun's Java language. C# is an object-oriented programming language used with XML-based Web services on the .NET platform and designed for improving productivity in the development of Web applications.

For the developing purposes we are using Visual Studio 2017, an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as web sites, web apps, web services and mobile apps. For the python script we are using the Python Development Tools for Visual Studio, with the Python 2.7.13 release.

3. Implementation details

3.1 Architecture

We implemented multiple python scripts to serve to our purpose. To obtain our trained data we had to perform multiple text processing tasks.



3.2 Preprocessing

3.2.1 Text extraction and cleaning

The Wikipedia database dump that we are using is in XML (eXtensible Markup Language) format, so our first task is to extract just the plain-text and ignore the rest of the information provided in the file. The XML itself contains complete, raw text of every revision, so in particular the full history files can be extremely large; en.wikipedia.org's January 2010 dump is about 5.87e12 bytes (5.34 TiB) raw. The Wikipedia maintainers provide, each month, an XML *dump* of all documents in the database: it consists of a single XML file containing the whole encyclopedia, that can be used for various kinds of analysis, such as statistics, service lists, etc.

```
{{Infocaseta Arie protejată
|nume= Cheile Şuşarei
|categorie_iucn= IV
|foto= Sasca.jpg
|descriere_foto=
|amplasare= [[Fişier:Actual Caras-Severin county
CoA.png|20px]][[Judeţul Caraş-Severin]] <br>{{ROM}}
|oraş_apropiat= [[Oraviţa]]
|hartă= România
|descriere_hartă= Localizarea rezervaţiei pe harta ţării
|lat_d= 44
|lat_m= 51
|lat_s= 22
|lat_NS= N
|long_d= 21
|long_m= 44
|long_s= 23
|long_EV= E
|ref_coordonate=
|înfiinţat= [[1982]], declarat în [[2000]]
|suprafaţă_ha= 246
|lungime_km= 3
|}}
'''Cheile Şuşarei''' (cunoscute şi sub numele de ''Cheile Valea
Morii'') alcătuiesc o arie protejată de interes naţional ce
corespunde categoriei a IV [[IUCN]] (rezervaţie naturală de tip
mixt), situată în [[Banat]], pe teritoriul judeţuli [[Judeţul
Caraş-Severin|Caraş-Severin]]<ref>[http://www.protectedplanet.net
/sites/183539 ProtectedPlanet.net - Cheile Şuşarei - delimitarea
ariei protejate], accesat la 10 martie 2012</ref>.
```

Example of original Wikipedia article structure

Wikipedia articles are written in the MediaWiki Markup Language which provides a simple notation for formatting text (bolds, italics, underlines, images, tables, etc.). It also allows inserting HTML markup in the documents. We used a Python tool called wikiExtractor for converting our code to a cleaner version.

```
<doc id="960189"
url="https://ro.wikipedia.org/wiki?curid=960189" title="Cheile
Şuşarei">
Cheile Şuşarei

Cheile Şuşarei (cunoscute şi sub numele de "Cheile Valea Morii")
alcătuiesc o arie protejată de interes naţional ce corespunde
categoriei a IV IUCN (rezervaţie naturală de tip mixt), situată
în Banat, pe teritoriul judeţuli Caraş-Severin.
…..

</doc>
```

Example of cleaned code using wikiExtractor

We can see that this tool still keeps some of the XML tags, so we further cleaned the code using a custom python script and Regex and got just the plaintext from our file.

3.2.2 Text Normalization

After a couple of tests we discovered that the text corpus that we gathered from wikipedia needs to go through a normalization process. The romanian alphabet is a modification of the classical Latin alphabet and consists of 31 letters, five of which (Ă, Â, Î, Ș, and Ț) have been modified from their Latin originals for the phonetic requirements of the language.[13]

```
Ă ă — a with breve - for the sound /ə/
Â â — a with circumflex - for the sound /ɨ/
Î î — i with circumflex - for the sound /ɨ/
Ș ș — s with comma - for the sound /ʃ/
Ț ț — t with comma - for the sound /t͡s/
```

Many printed and online texts still incorrectly use cedilla for the letters *Ş ş Ţ* and *ţ* .This state of affairs is due to an initial lack of glyph standardization, compounded by the lack of computer font support for the comma-below variants. Because of this we had to create a *find-and-replace* script to avoid the duplication of some words that are written in two ways. This step also improved a little the accuracy of our results.

3.2.3 Stopword Removal

Stopword removal is the process of filtering out very common words that really do not add any information in terms of the meaning of the sentence or the text. In general for common languages used in NLP we find complex public lists that can be used, but since there is not that much information provided for Romanian language we created a database of stopwords based on data merged from different open sources. The main sources were the python package *stop-words 2015.2.23.1* and Kevin Bougé's (ex Microsoft developer) website free resources.

```
$ pip install stop-words
```

stop-words 2015.2.23.1 installation

```
from stop_words import get_stop_words

stop_words = get_stop_words('ro')
stop_words = get_stop_words('romanian')

from stop_words import safe_get_stop_words

stop_words = safe_get_stop_words('unsupported language')
```

Basic usage of the package

3.2.4 Stemming

Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes.[7]

word	**stem**
abstract	abstract
abstractă	abstract
abstracte	abstract
abstractiza	abstractiz
abstractizare	abstractiz
abstractizat	abstractiz
abstractizăm	abstractiz
abstracto	abstracto
abstracția	abstracț
abstracții	abstracț

The stemming process is split in the following steps:

Step 1. Removal of the plural form for some words

```
   foreach (string word in textCorpus)
       {
           if (word.EndsWith("ul")) word.Remove(word.Length -
2);
           if (word.EndsWith("ului")) word.Remove(word.Length -
4);
            if (word.EndsWith("aua")) word.Remove(word.Length -
2);

            ..........................

        }
```

The following logic is applied here:

Search for the longest among the following suffixes perform the action indicated.

Suffix	**Action**
ul, ului	delete
aua	replace with ***a***
ea, ele, elor	replace with ***e***
ii, iua, iei, iile, iilor, ilor	replace with ***i***
ile	replace with ***i*** if not preceded by ***ab***
atei	replace with ***at***
ație, ația	replace with ***ați***

Step 2. Reduction of combining suffixes

Suffix	**Action**
abilitate, abilitati, abilităi, abilităţi	replace with ***abil***
ibilitate	replace with ***ibil***
ivitate, ivitati, ivităi, ivităţi	replace with ***iv***
icitate, icitati, icităi, icităţi, icator, icatori, iciv, iciva, icive, icivi, icivă, ical, icala, icale, icali, icală	replace with ***ic***
ativ, ativa, ative, ativi, ativă, aţiune, atoare, ator, atori ătoare, ător, ători	replace with ***at***
itiv, itiva, itive, itivi, itivă, iţiune, itoare, itor, itori	replace with ***it***

Step 3. Remove the most common Romanian suffixes

Suffix	**Action**
at, ata, ată, ati, ate, ut, uta, ută, uti, ute, it, ita, ită, iti ite, ic, ica, ice, ici, ică, abil, abila, abile, abili, abilă, ibil, ibila, ibile, ibili, ibilă, oasa, oasă, oase, os, osi, oşi, ant, anta, ante, anti, antă, ator, atori, itate, itati ităi, ităţi, iv, iva, ive, ivi, ivă	delete
iune, iuni	delete if preceded by ***ţ***, and replace the ***ţ*** by ***t***.
ism, isme, ist, ista, iste, isti, istă, işti	replace with ***ist***
ism isme ist ista iste isti istă işti	replace with ***ic***

3.3 Python implementation

On this subchapter we will focus on the two main python scripts that were used, the training script and the script for plot generation.

3.3.1 Data trainer

For training our data we used the word2vec implementation offered by gensim python library, it produces word vectors with deep learning via word2vec's "skip-gram and CBOW models", using either hierarchical softmax or negative sampling.
The first step was to initialize a model and save it to disk so we can further use it on our tests. We started with a smaller chunk of our text corpus and upgraded from there, also increasing the dimensions of our vectors.

```
model = Word2Vec(
Wiki_plain_text,
size=100, #dimensionality of the feature vectors. (50-300)
window=5, #max distance between current and predicted word
min_count=10,#ignore words with total frequency lower than this.
workers=10) #number of threads

model.save(fname) #save a model to disk

....

model=Word2Vec.load(fname) #reload the trained model from disk
```

The word vectors are stored in a KeyedVectors instance in *model.wv*. This separates the read-only word vector lookup operations in KeyedVectors from the training code in Word2Vec. The word vectors can also be instantiated from an existing file on disk in the word2vec C format as a KeyedVectors instance but it is impossible to continue training the vectors loaded from the C format because hidden weights, vocabulary frequency and the binary tree is missing.

```
model.wv[`calculator`]  # numpy vector of a word
output: array([-0.00449447, -0.00310097,  0.02421786, ...],
dtype=float32)
```

```
from gensim.models.keyedvectors import KeyedVectors
word_vectors =
KeyedVectors.load_word2vec_format('/tmp/vectors.txt',
binary=False)
 # C text format
word_vectors =
KeyedVectors.load_word2vec_format('/tmp/vectors.bin',
binary=True)  # C binary format
```

3.3.2 Plot generator

Besides the .Net application for getting the most similar words from the vocabulary, we implemented a python script to generate plots with those words. To achieve this we had to reduce the dimensionality of the vectors of given words.

3.3.2.1 PCA

Principal component analysis (PCA) is a technique used to emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize. For this we used scikit-learn python package for the transformations and matplotlib for the plot generation.

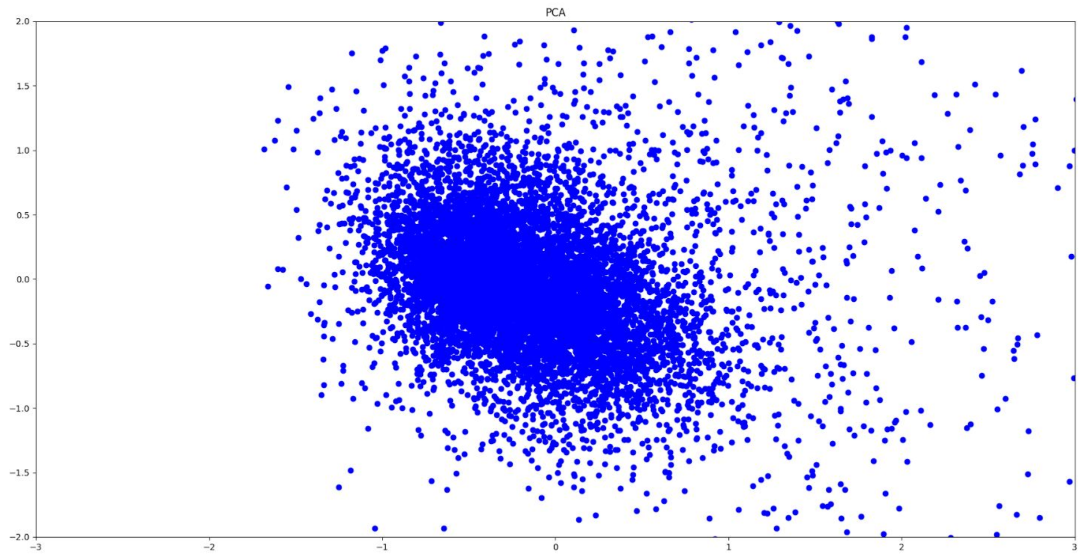
```
conda install scikit-learn
conda install -c conda-forge matplotlib=2.0.2
```

It uses the LAPACK implementation of the full SVD or a randomized truncated SVD by the method of Halko et al. 2009, depending on the shape of the input data and the number of components to extract.
It can also use the scipy.sparse.linalg ARPACK implementation of the truncated SVD.

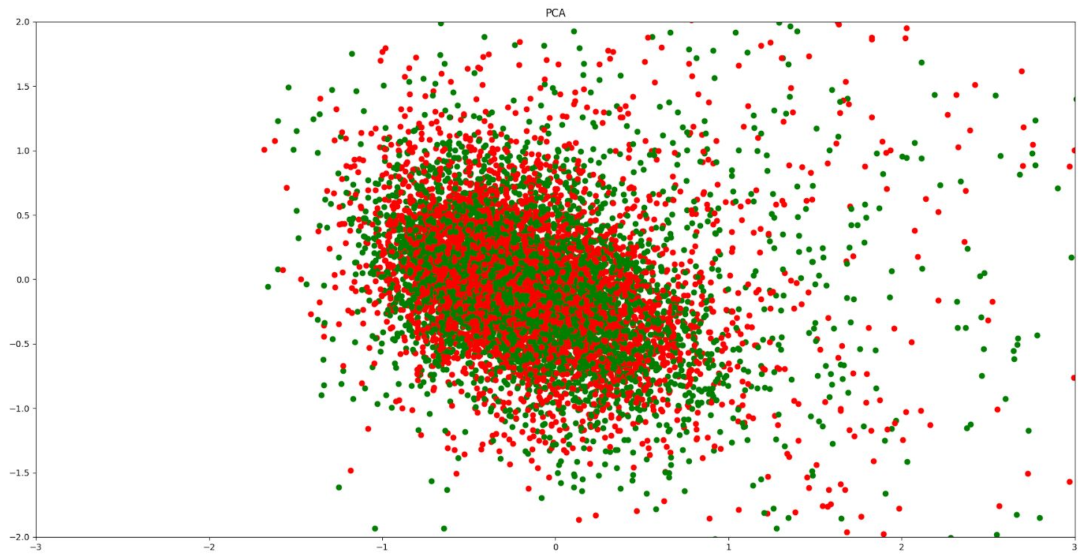
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA

…

vectors = [model[word] for word in words]
pca = PCA(n_components=2, whiten=True)
vectors2d = pca.fit(vectors).transform(vectors)
for point, word in zip(vectors2d , words):
   plt.scatter(point[0], # point coord x
               Point[1], # point coord y
               c='b')    # color
   plt.annotate(
   word,   #node name
   xy = (point[0], point[1]),
   size = "x-large"
   )
plt.show() #show plot
```



[Point distribution for the first 10000 words in vocabulary]



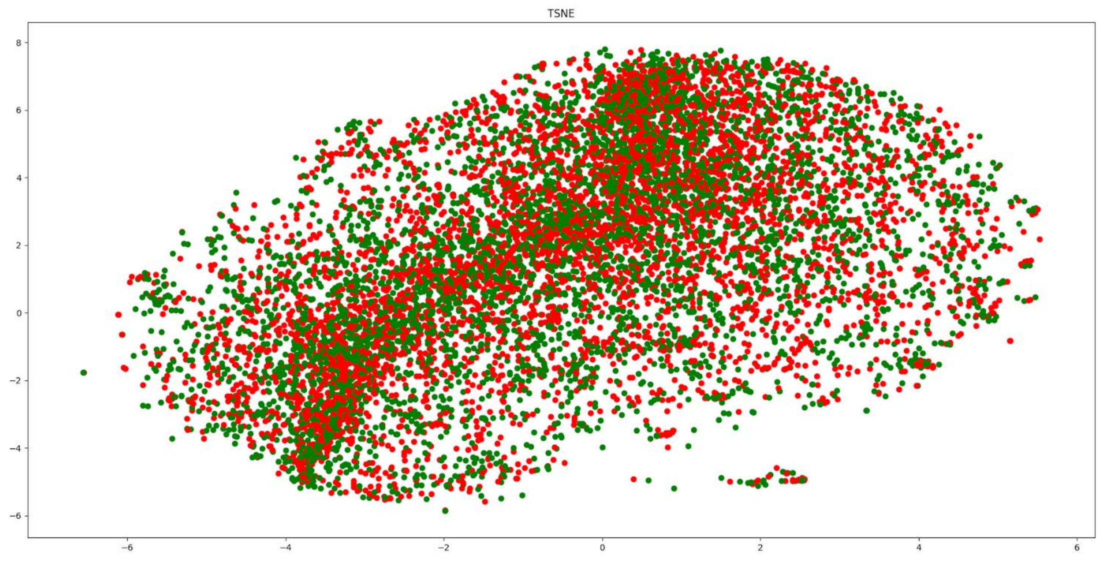
3.3.2.2 TSNE

t-distributed stochastic neighbor embedding (t-SNE) is a machine learning algorithm for dimensionality reduction developed by Geoffrey Hinton and Laurens van der Maaten. It is a nonlinear dimensionality reduction technique that is particularly well-suited for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. Specifically, it models each high-dimensional object by a two- or three-dimensional point in such a way that similar objects are modeled by nearby points and dissimilar objects are modeled by distant points.

t-SNE [18] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.

```
tsne = TSNE
```

```
(
n_components=2, #Dimension of the embedded space.
perplexity=30.0,
early_exaggeration=4.0,
learning_rate=1000.0,
n_iter=1000,
n_iter_without_progress=30,
min_grad_norm=1e-07,
metric='euclidean',
init='random',
verbose=0,
random_state=None,
method='barnes_hut',
angle=0.5
)
vectors2d = tsne.fit_transform(vectors)
```



[TSNE for the first 10000 words in vocabulary]

3.4 .Net implementation

Because we wanted a visual representation of the trained data we created a C# Window Form application and an open source implementation of the Word2Vec library in .NET. From this application parse the trained data in the search of the nearest words for our search.

```
public BestWord[] Search(string intext)
     {
          BestWord[] bestWords = new BestWord[N];
          long[] bi = new long[100];
          float[] vec = new float[max_size];
          string[] st = intext.Split(' ');
          int cn = st.Length;
          long b = -1;

               for (long a = 0; a < cn; a++)
               {

                    for (b = 0; b < Words; b++)
                    {
                    string word = new string(Vocab, (int)
(b*max_w), (int) max_w).Replace("\0", string.Empty);;
                    if (word.Equals(st[a])) break;
                    }

                    if (b == Words) b = -1;
                    bi[a] = b;
                    Console.Write("\nWord: {0}  Position in
vocabulary: {1}\n", st[a], bi[a]);

                    if (b == -1)
                    {
                    Console.Write("Out of dictionary word!\n");
                    break;
                    }

               }
               if (b == -1) return new BestWord[0];

               for (long a = 0; a < Size; a++) vec[a] = 0;
               for (b = 0; b < cn; b++)
               {
                    if (bi[b] == -1) continue;
                    for (long a = 0; a < Size; a++) vec[a] +=
M[a + bi[b] * Size];
               }
               float len = 0;
               for (long a = 0; a < Size; a++) len += vec[a] *
vec[a];
               len = (float)Math.Sqrt(len);
               for (long a = 0; a < Size; a++) vec[a] /= len;
               for (long c = 0; c < Words; c++)
               {
                    long a = 0;
                    for (b = 0; b < cn; b++) if (bi[b] == c) a
= 1;
                    if (a == 1) continue;
                    float dist = 0;
                    for (a = 0; a < Size; a++) dist += vec[a] *
```

```
M[a + c * Size];
                     for (a = 0; a < N; a++)
                     {
                     if (dist > bestWords[a].Distance)
                     {
                          for (long d = N - 1; d > a; d--)
                          {
                               bestWords[d] = bestWords[d - 1];
                          }
                          bestWords[a].Distance = dist;
                          bestWords[a].Word = new string(Vocab,
(int)(max_w * c), (int)max_w).Replace("\0",String.Empty).Trim();
                          break;
                     }
                     }


                }
           return bestWords;
      }
```

The results are stored in a list as a pair of word-distance.

```
public struct BestWord
{
public string Word { get; set; }
public float Distance { get; set;
}
}
```

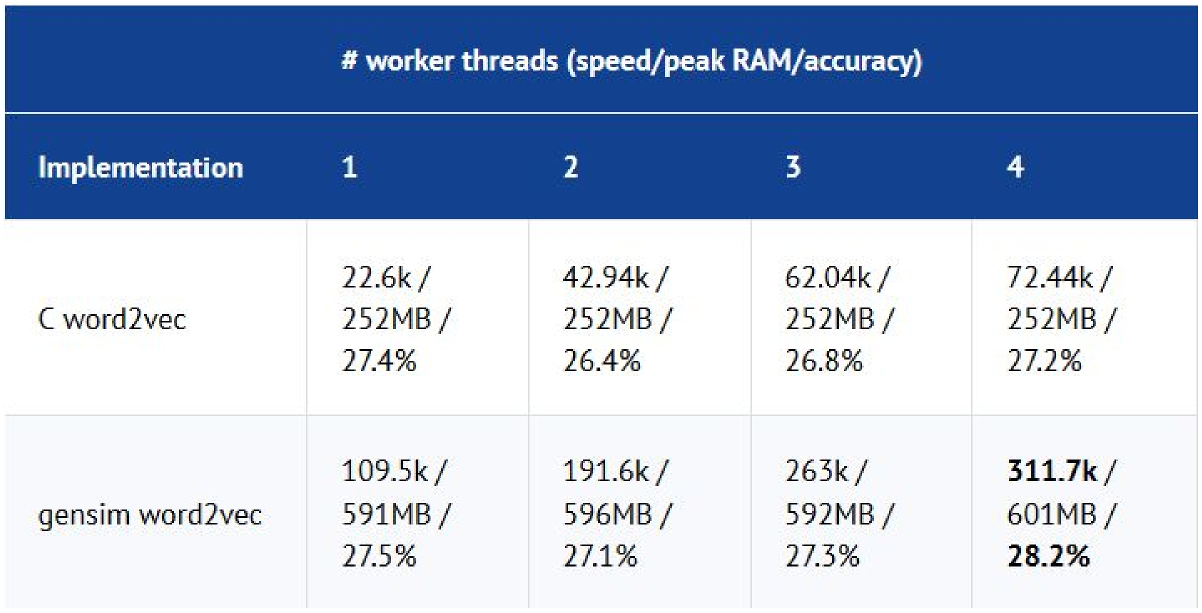
4. Metrics

We start training the data sequentially.

4.1 Word2phrase						
Test number	File Size	Words processed	Vocab. size	Vocab. size unigrams + bigrams	Words in train file	Time using 10 threads
1.	642 MB	82100K	7057K	5267235	82156584	1811 sec.
2.	1540 MB	167000K	2000K (fixed size)	1969376	26693760	~

4.2 Word2vec					
Test number	File Size	Vocab. size	Dimensionality of the feature vectors	Words in train file	Time using 10 threads
1.	418 MB	1050458	50	59060537	1372 sec.
2.	418 MB	1050458	100	59060537	2247 sec.
3.	1540 MB	1000000 (fixed size)	100	17718161	~7-8

				1	hours



	# worker threads (speed/peak RAM/accuracy)			
Implementation	1	2	3	4
C word2vec	22.6k / 252MB / 27.4%	42.94k / 252MB / 26.4%	62.04k / 252MB / 26.8%	72.44k / 252MB / 27.2%
gensim word2vec	109.5k / 591MB / 27.5%	191.6k / 596MB / 27.1%	263k / 592MB / 27.3%	**311.7k** / 601MB / **28.2%**

[Radim Řehůřek -Parallelizing word2vec in Python]

4.3 Cython

The baseline performance of the NumPy code under these conditions is 1.4k words per second. Rewriting the training loop in Cython improves this to **33.3k words/sec**, which is a **24x speedup**.

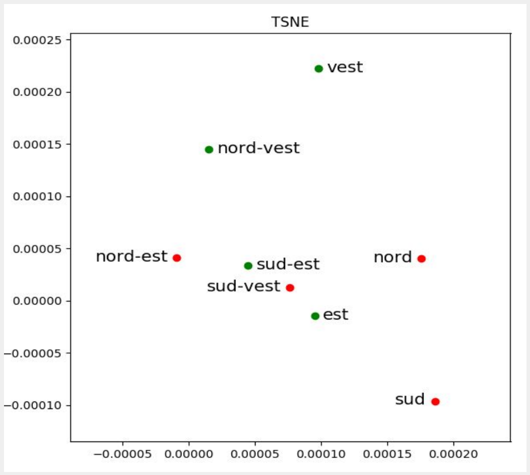
5. Study Cases

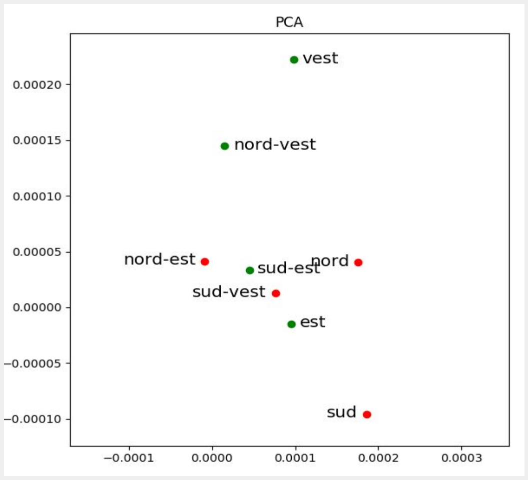
In this section we describe our investigations and experiments on the trained data. In our tests we have discovered that for some words we receive better results compared with others, so we tried to investigate the reason behind this. We tested our data for different fields and we saw a pattern, we got the best results for technical fields like computer science, geography, biology, chemistry etc.

Test 1.

First seven matches for the word **nord**

```
vest 0,8996736
sud 0,897999
est 0,8366427
nord-est 0,7947339
nord-vest 0,7787046
sud-vest 0,7704518
sud-est 0,7697451
```





Test 2.

First twenty matches for the word **calculator**

aplicaţie **0,8392286**

computer **0,8365033**

aparat **0,8364583**

utilizator **0,8331482**

modulul **0,8313626**

logic **0,8278407**

digital **0,8197194**

server **0,8192139**

interfaţă_grafică **0,8173387**

utilizatorii **0,8145707**

sistem_unix **0,813207**

utilizatorul **0,8128749**

sql **0,8069478**

lcd **0,8041156**

fişiere **0,8036943**

```
cablurilor 0,8033258
design 0,8029817
gnu 0,7993091
ecran 0,7953753
ssl 0,7942548
```

We have provided in the above examples the cases in which our model performs at it's best, but also noticed that for some common Romanian words we received not so good results.

```
First ten matches for the word imigrant

spătarul_toma 0,6021668
pâlc_stejari 0,6016639
şcoală_exegeza 0,5917977
datorează_pont 0,5769308
johann_martinelli 0,5730138
1987_conservator 0,5719516
nume_botaş 0,5690339
regele_fracisc 0,561347
biserici_stefăneşti 0,5603685
johann_jakob 0,5581158
```

After a couple of tests we discovered that some words are added into our vocabulary but they lack enough information, for example the word ”*imigrant*” presented above does not exist as an wikipedia article and the mentions of it are scattered across different articles, lacking context, but still having enough occurrences to be saved in the vocabulary.

6. Conclusion

In this thesis we have presented the training process of a text corpus using Word2Vec and also analysed how well this group of related models perform on Romanian language. We have tested data trained with different dimensions for accuracy.

6.1 Future work

We want to expand our python script collection to further improve the preprocessing of the data. Moreover, we intend to gather more data across the internet, from various fields, for training purposes thus being able to further improve our model accuracy.

7. Bibliography

1. **Goldberg, Y.** and **Levy, O.** (2014). *word2vec explained: deriving Mikolov et al.'s negative-sampling word-embedding method*.
2. **Mikolov, T., Chen, K., Corrado, G.,** and **Dean, J**. (2013a). *Efficient estimation of word representations in vector space*
3. **Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S.,** and **Dean, J**. (2013b). *Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems*, pages 3111–3119.
4. **Mnih, A.** and **Hinton, G. E**. (2009). *A scalable hierarchical distributed language model.* In Koller, D., Schuurmans, D., Bengio, Y., and Bottou, L., editors, Advances in Neural Information Processing Systems 21, pages 1081–1088. Curran Associates, Inc.
5. **Morin F.** and **Bengio Y.** (2005). *Hierarchical probabilistic neural network language model*. In AISTATS, volume 5, pages 246–252. Citeseer.
6. *A Closer Look at Skip-gram Modelling* **David Guthrie, Ben Allison, Wei Liu**, **Louise Guthrie**, **Yorick Wilks** - NLP Research Group, Department of Computer Science, University of Sheffield Regent court, 211 Portobello Street, Sheffield, S10 4DP{dguthrie, ben, wei, louise, yorick}@dcs.shef.ac.uk
7. **Christopher D. Manning**, **Prabhakar Raghavan** and **Hinrich Schütze**, *Introduction to Information Retrieval*, Cambridge University Press. 2008. [Online] https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html
8. **Mikolov, Tomas; Yih, Wen-tau; Zweig, Geoffrey** (2013). "*Linguistic Regularities in Continuous Space Word Representations.*". *HLT-NAACL*: pp. 746–751.
9. **Grimes, S.** (2007). *A Brief History of Text Analytics*. [online] B-eye-network.com.
10. **Donna Harman**. *How Effective is Suffixing?* Journal of the American Society for Information Science, 42(1):7-15, 1991.

11. **David Hull**. *Stemming Algorithms -- A Case Study for Detailed Evaluation*. Journal of the American Society for Information Science, 47(1), 1996.
12. **M. Popovic and P. Willett**. *The Effectiveness of Stemming for Natural Language Access to Slovene Textual Data*. Journal of the American Society for Information Science, 43(5):384-390, 1992.
13. **Alf Lombard**, "*Despre folosirea literelor â şi î*", Limba română, 1992, nr. 10, p. 531
14. Wikimedia text format - https://www.mediawiki.org/wiki/Help:Formatting
15. **David Meyer -** *How exactly does word2vec work,* dmm@ { 1-4-5. Net, uoregon.edu, brocade.com, … } July 31, 2016
16. **Gensim topic modelling for humans -** models.word2vec - online documentation - https://radimrehurek.com/gensim/models/word2vec.html
17. **Scikit-learn 0.18** - *sklearn.decomposition.PCA and sklearn.decomposition.TSNE* - online documentation- http://scikit-learn.org/stable/documentation.html
18. **van der Maaten, L.J.P.; Hinton**, **G.E**. *Visualizing High-Dimensional Data Using t-SNE*. Journal of Machine Learning Research 9:2579-2605, 2008.