# Word to Vec report

Itai Danielli 023047087

Date: 13/07/2019.

## Term Frequency Inverse Document Frequency

The first step of using Word to vector conversion was done within Git Crawler application which I developed for searching git. The project is designed to find repositories and then classify them using a set of words we may find as characterizing certain properties such as quality security and so on.

The repositories data is collected in to MySql database, the data being collected is the repositories Readme files, issues and comment, and topics.

The classifier I chose to use for this type of data is: Term frequency Inverse Document Frequency. This classifier enables classification of short text using small number of words.

The following table shows a test of searching hello, world, how, are, you. And the repository most compatible with these word is “jest”.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Repo nmanes/Search Term** | **hello** | **world** | **how** | **are** | **you** |
| **jest** | 32.10 | 43.95 | 358.72 | 1031.94 | 780.93 |
| **grumphp** | 0.54 | 0.00 | 21.14 | 85.06 | 65.27 |
| **qualityworks-full-stack-testing** | 0.00 | 0.00 | 0.49 | 0.00 | 0.27 |
| **source** | 0.00 | 0.00 | 0.24 | 0.00 | 0.07 |
| **ios-factor.com** | 0.00 | 0.00 | 0.00 | 0.97 | 0.54 |
| **FingerprintIdentify** | 0.00 | 0.00 | 0.00 | 0.24 | 0.13 |
| **validator.js** | 0.00 | 0.00 | 0.00 | 0.00 | 0.00 |

The project code can be found on my GitHub account in the following location:

<https://github.com/Danielli-Itai/GitCrawlerPy>

# Word embedding Word2Vec Basic:

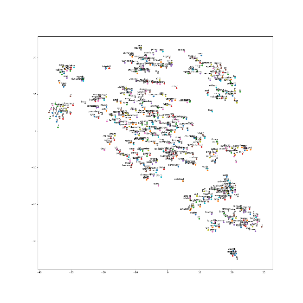
The next step was to run a word to vector embedding algorithm using tensor flow. For this purpose, I chose to use a tutorial provided by tensor flow team, the tutorial is available at:

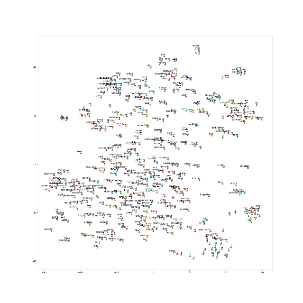
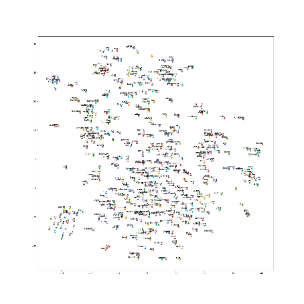
[**https://www.tensorflow.org/tutorials/representation/word2vec**](https://www.tensorflow.org/tutorials/representation/word2vec)

I took the basic implementation as a starting point and re designed the software in order to make it more usable for experimenting. All the algorithm parameters are stored in a single configuration class backed up by an .ini file containing the initial parameters values.

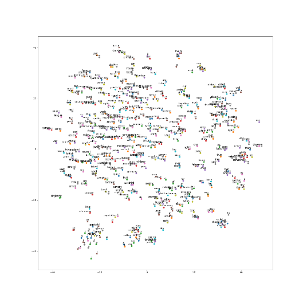
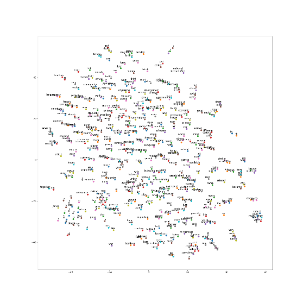
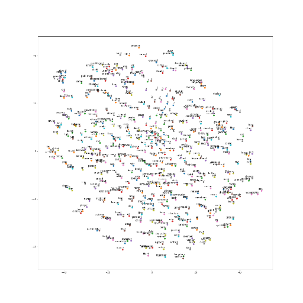
The processing results are all stored in the Summery folder containing the the training log file and the final diagram showing the words vector space down sized in to tow dimensional space. For my experiment I ran the algorithm using embedding layer of sizes 8,16,…256.

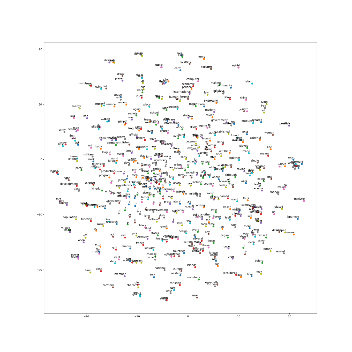
The following graphs were found:

Embedding vector size 8 Embedding vector size 16: Embedding vector size 24:



Embedding vector size 32: Embedding vector size 64: Embedding vector size 128





Embedding vector size 248:

This evolution shows how the size of the embedding vector defines how words are distributed within the vector space allowing similar words become closer while different words are not clustered due to lack of space granularity.

# Word embedding Word2Vec Advanced:

The next step of word to vector embedding is checking the influences of embedding vector size, on the analogy test accuracy. The tutorial advanced example includes analogy testing, but it also requires integration of c code library with the Tensorflow-framework. For this task I installed a virtual Linux machine then built the library. I redesigned the source code to enable running it multiple times with different options.

In order to prevent the effect of randomity on the results, every run is initialized with the same seed.

Database:

Words used for training: 17,005,207

Vocabulary size: 71290

Number of questions: 17827

## Running Window size 1:

I ran a loop of tests with the embedding vector sizes in multiple of 8 which means 8,16, …. 248 and measured the loss and + analogy test accuracy.

From the following table I conclude that there is correlation between the loss and the analogy. The optimal embedding size is around 8\*8=64.

Based on the results I concluded that the window size may have an impact on the low performance in the analogy test accuracy. And so I decided to raise the window size to 5 but reduce the number of iterations because the time to run each test at this size is quite long.

## Running Window size 5:

On this run I reduced the number of iterations to reduce the time required for running the algorithm and used the following embedding vector sizes: 8, 16, 24, 32, 40, 48, 56, 64, 128, 192.

From the following results we can conclude that the window size contributes to the analogy results alongside the window size.