

# CSI 5388 Assignment 1

Understanding tokenization and word embedding

## Group Details:

Aaditya Suri (300252128)

Ranjan Goyal

Paritosh Pal Singh

## Problem Statement and Introduction

- Implement a word tokenizer that splits texts into tokens and separates punctuation marks and other symbols from the words.
- What is Tokenization ?**
  - Tokenization is the process of breaking up the original raw text into component pieces which are known as tokens.
  - Tokenization is usually the initial step for further NLP operations like stemming, lemmatization, text mining, text classification, sentiment analysis, language translation, chatbot creation, etc.
- What are Tokens ?**

Tokens are broken pieces of the original text that are produced after tokenization. Tokens are the basic building blocks of text - everything that helps us understand the meaning of the text is derived from tokens and the relationship to one another. For example, the character is a token in a word, a word is a token in a sentence, and a sentence is a token in a paragraph.

### Tokenization



### Top 13 Tokens



## Assignment 1 a, Corpus Output

### Token output for first 10 posts in corpus

- [``', 'Yeah', ',', 'I', 'get', 'that', 'argument', '.', 'At', 'this', 'point', ',', 'I', "'d", 'prefer', 'is', 'she', 'lived', 'in', 'NC', 'as', 'well', '.', ""'],
- ['The', 'blazers', 'and', 'Mavericks', '(', 'The', 'wests', '5', 'and', '6', 'seed', ')', 'did', 'not', 'even', 'carry', 'a', 'good', 'enough', 'record', 'to', 'make', 'the', 'playoffs', 'in', 'the', 'east', 'last', 'year', '.'],
- ['They', "'re", 'favored', 'to', 'win', '.'],
- ['deadass', 'do', "n't", 'kill', 'my', 'buzz'],
- ['Yep', 'can', 'confirm', 'I', 'saw', 'the', 'tool', 'they', 'use', 'for', 'that', '.', 'It', 'was', 'made', 'by', 'our', 'boy', 'EASports\_MUT'],
- ['do', 'you', 'find', 'ariana', 'grande', 'sexy', '?'],
- ['What', "'s", 'your', 'weird', 'or', 'unsettling', 'Trick', 'or', 'Treat', 'story', '?'],
- ['Probably', 'Sephiroth', '.', 'I', 'refuse', 'to', 'taint', 'his', 'greatness', 'with', '\*', 'my', '\*', 'weak', 'builds', '.', 'He', 'should', 'equip', '\*', 'ONLY', '\*', 'the', 'best', 'TMs', '.'],
- [``', 'What', 'to', 'upgrade', '?', 'I', 'have', '\$', '500', 'to', 'spend', '(', 'mainly', 'because', 'it', "'s", 'my', 'birthday', 'on', 'the', '31st', ')', 'and', 'I', "m", 'not', 'really', 'sure', 'what', 'to', 'upgrade', '.', 'I', 'do', "n't", 'have', 'to', 'spend', 'all', '\$', '500', ',', 'I', 'could', 'spend', 'as', 'little', 'or', 'all', 'of', 'it', 'if', 'I', 'want', '.', 'Specs', 'are', ':', 'CPU', ':', 'i5-6600K', 'Cooler', ':', 'CM', '212', 'EVO', 'RAM', ':', '8GB', 'Corsair', 'Vengeance', 'LPX', '2400MHz', 'GPU', ':', 'EVGA', 'GTX', '1070', 'SC', 'Case', ':', 'Corsair', 'Spec', 'Alpha', 'PSU', ':', 'EVGA', 'P2', '650w', 'Storage', ':', 'One', '480', 'GB', 'Corsair', 'Force', 'LE', 'Monitor', ':', 'BenQ', 'XL2411Z', 'Keyboard', ':', 'Razer', 'Blackwidow', 'Tournament', 'Edition', 'What', 'I', "m", 'thinking', 'of', 'right', 'now', 'is', ':', '-Get', 'a', '1TB', 'WD', 'Blue', '-Buy', 'another', '8GB', 'of', 'RAM', '-Replace', 'my', 'Razer', 'Blackwidow', 'with', 'a', 'keyboard', 'with', 'actual', 'Cherry', 'MX', 'switches', '-Replace', 'my', 'case', '(', 'Does', "n't", 'really', 'offer', 'features', 'for', 'good', 'cable', 'management', ')', '-Save', 'up', 'for', 'a', 'Volta', 'Card', "'"]],
- [``', 'Probably', 'count', 'Kanye', 'out', 'Since', 'the', 'rest', 'of', 'his', 'tour', 'is', 'canceled', ',', 'it', 'does', "n't", 'seem', 'in', 'the', 'cards', '.', 'But', 'he', "'s", 'also', 'so', 'unpredictable', ',', 'it', 'could', 'change', '.', 'Thankfully', 'I', 'saw', 'him', 'before', 'he', 'started', 'canceling', 'shows', ',', 'going', 'on', 'long', 'rants', 'and', 'showing', 'up', 'late', '.', 'But', 'I', "'d", 'love', 'to', 'see', 'him', 'again', ',', 'especially', 'at', 'Lolla', '.', 'Thoughts', '?', ""']

## Assignment 1 – Part A: Assessing the corpus without exclusions

### Number of Tokens

**2,915,666**

Total number of Tokens obtained from corpus **without removing** punctuations or stop words

### Number of Types

**106,200**

Total number of unique Tokens obtained from corpus **without removing** punctuations or stop words

### Type/Token Ratio

**0.036423**

Number of types ( unique tokens) divided by total number of tokens in the corpus

### Token Appeared Only Once

**57,847**

Tokens which are occurring only one in the corpus

## Assignment 1 – Part A: Top 20 tokens as per frequency

Rank

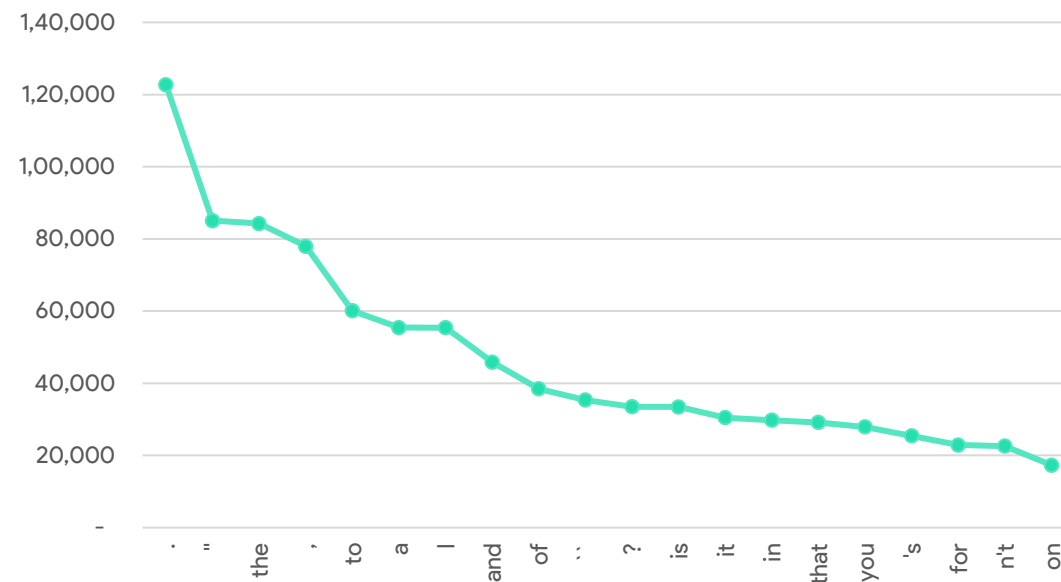
Word

Frequency

1	.	1,22,683
2	"	85,054
3	the	84,270
4	,	77,973
5	to	60,086
6	a	55,442
7	I	55,385
8	and	45,873
9	of	38,445
10	``	35,345

11	?	33,474
12	is	33,456
13	it	30,482
14	in	29,783
15	that	29,155
16	you	27,879
17	's	25,406
18	for	22,883
19	n't	22,598
20	on	17,296

Frequency



## Assignment 1 – Part A: Assessing the corpus removing punctuations

Number of Words

**2,353,399**

Total number of Tokens obtained  
from corpus **removing  
punctuations only**

Type/Token Ratio

**0.0364264**

Number of types (unique tokens)  
divided by total number of tokens  
in the corpus **w/o punctuations**

Top 3 Most frequent Words

Word

Frequency

**the**

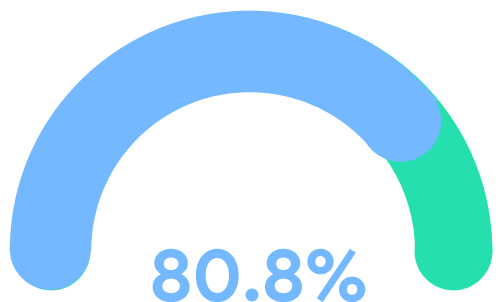
**84,270**

**to**

**60,086**

**a**

**55,442**



of the tokens actual  
words, **19.2%** of the  
tokens are punctuations



Increase in type to  
token ratio after  
removing punctuations

## Assignment 1 – Part A: Top 20 words as per frequency

Rank

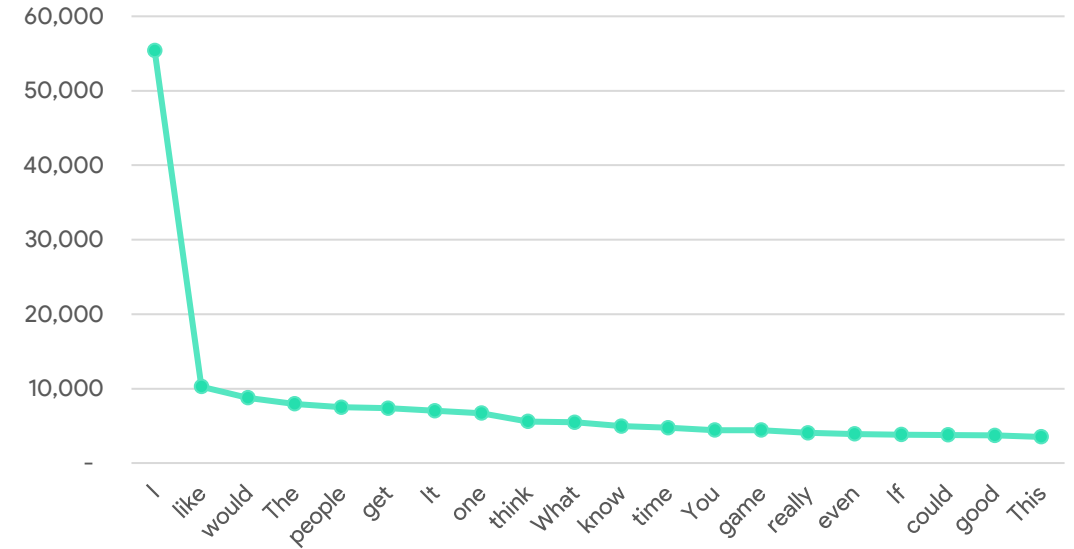
Word

Frequency

1	the	84270
2	to	60086
3	a	55442
4	I	55385
5	and	45873
6	of	38445
7	is	33456
8	it	30482
9	in	29783
10	that	29155

11	you	27879
12	for	22883
13	on	17296
14	be	15277
15	have	15196
16	with	15118
17	this	14875
18	do	14630
19	was	14205
20	are	14132

Frequency

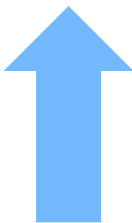


## Assignment 1 – Part A: Assessing the corpus removing punctuations and stop words

Type/Token Ratio

**0.0620838**

Number of types (unique tokens) divided by total number of tokens in the corpus **w/o punctuations and stop words**

 **70%**

Increase in type to token ratio after removing punctuations and stop words

### Top 3 Most frequent Words

Word	Frequency
I	55,385
like	10,275
would	8,768

### Top 3 Most frequent Bigrams

Word	Frequency
I, think	2,810
F*ck, F*ck	1,998
I, know	1,710



## Assignment 1 – Part A: Top 20 words as per frequency

Rank

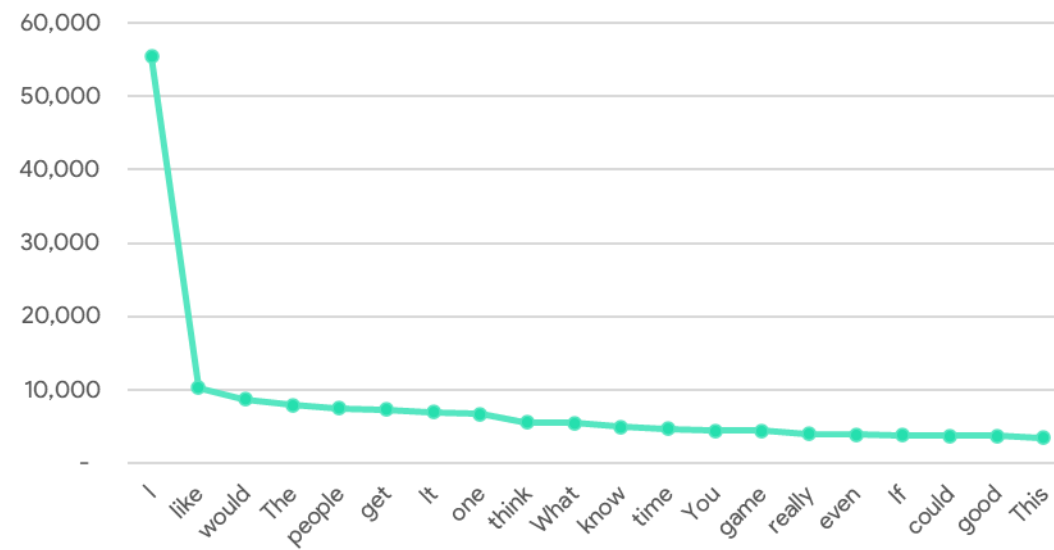
Word

Frequency

1	I	55385
2	like	10275
3	would	8768
4	The	7956
5	people	7513
6	get	7370
7	It	7031
8	one	6706
9	think	5592
10	What	5474

11	know	4973
12	time	4753
13	You	4431
14	game	4426
15	really	4060
16	even	3919
17	If	3832
18	could	3781
19	good	3723
20	This	3525

Frequency



## Assignment 1 – Part A: Top 20 bigrams as per frequency

Rank

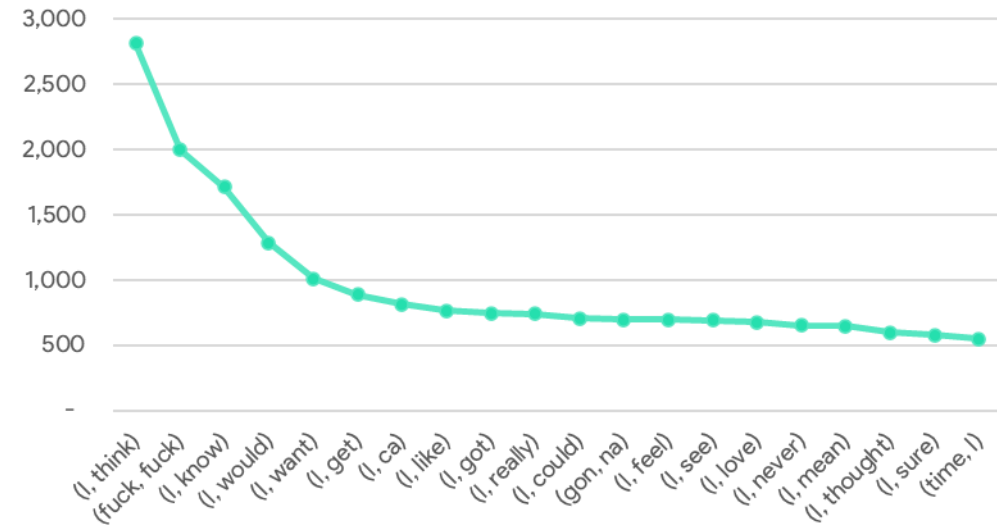
Word

Frequency

1	(I, think)	2,810
2	(fuck, fuck)	1,998
3	(I, know)	1,710
4	(I, would)	1,286
5	(I, want)	1,009
6	(I, get)	888
7	(I, ca)	816
8	(I, like)	765
9	(I, got)	746
10	(I, really)	739

11	(I, could)	705
12	(gon, na)	699
13	(I, feel)	695
14	(I, see)	694
15	(I, love)	679
16	(I, never)	654
17	(I, mean)	649
18	(I, thought)	601
19	(I, sure)	579
20	(time, I)	550

Frequency



## Assignment 1 – Part B: Word Embedding

- Using 5 pre-trained word embeddings calculate their performance over different tasks : Similarity and Analogy on different datasets

- **What is Word embedding ?**

In natural language processing (NLP), word embedding is a term used for the representation of words for text analysis, typically in the form of a real-valued vector that encodes the meaning of the word such that the words that are closer in the vector space are expected to be similar in meaning

- **Datasets and Models used in this assignment ?**

We are using datasets from 2 different contexts

- Similarity

Mturk

MEN

WS353

Rubenstein &  
Goodenough

Rare Words

SimLex999

TR9856

- Analogy

Google Analogy

MSR

## Assignment 1 – Part B: Word Embedding models Used

### Skip Gram and Word2Vec

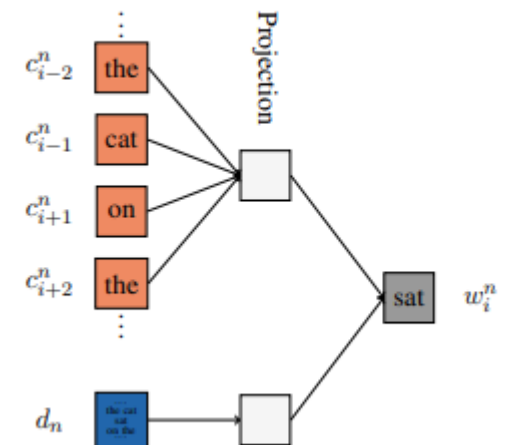
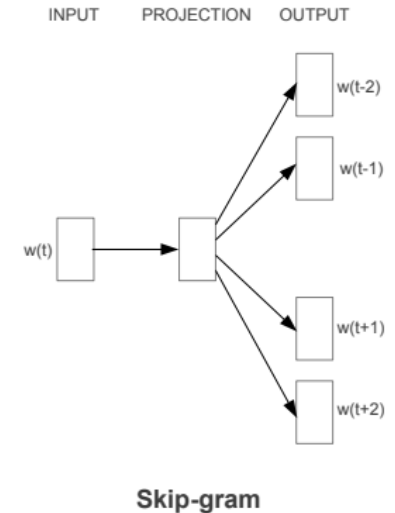
Skip-gram is one of the unsupervised learning techniques used to find the most related words for a given word. Skip-gram is used to predict the context word for a given target word. It's reverse of CBOW algorithm. Here, target word is input while context words are output. As there is more than one context word to be predicted which makes this problem difficult. First step of working of skip grams is conversion of word to vectors.

The **word2vec** tool takes a text corpus as input and produces the word vectors as output. It first constructs a vocabulary from the training text data and then learns vector representation of words. The resulting word vector file can be used as features in many natural language processing and machine learning applications.

### Parallel Document Context (PDC)

In this model, a target word is predicted by its surrounding context, as well as the document it occurs in. The former prediction task captures the paradigmatic relations, since words with similar context will tend to have similar representations. While the latter prediction task models the syntagmatic relations, since words co-occur in the same document will tend to have similar representations.

The model can be viewed as an extension of CBOW model (Mikolov et al., 2013a), by adding an extra document branch. Since both the context and document are parallel in predicting the target word, we call this model the Parallel Document Context (PDC) model.



## Assignment 1 – Part B: Word Embedding models Used

### Hierarchical Document Context

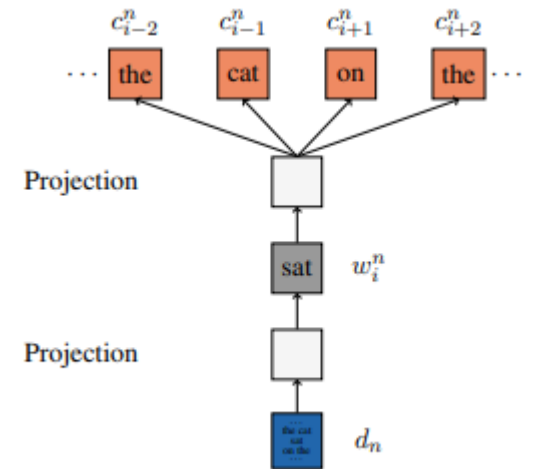
In this model the document is used to predict a target word, and the target word is further used to predict its surrounding context words. Since the prediction is conducted in a hierarchical manner, we name this model the Hierarchical Document Context (HDC) model. Similar as the PDC model, the syntagmatic relation in HDC is modeled by the document-word prediction layer and the word-context prediction layer models the paradigmatic relation.

### LexVec

A method for generating distributed word representations that uses low-rank, weighted factorization of the Positive Point-wise Mutual Information matrix via stochastic gradient descent, employing a weighting scheme that assigns heavier penalties for errors on frequent cooccurrences while still accounting for negative co-occurrence. Evaluation on word similarity and analogy tasks shows that LexVec matches and often outperforms state-of-the-art methods on many of these tasks.

### ConceptNet Numberbatch

Conceptnet Numberbatch is a set of semantic vectors: it associates words and phrases in a variety of languages with lists of 600 numbers, representing the gist of what they mean. Some of the information that these vectors represent comes from ConceptNet, a semantic network of knowledge about word meanings. ConceptNet is collected from a combination of expert-created resources, crowdsourcing, and games with a purpose.



## Assignment 1 – Part B: Word Embedding models Parameters

Skip Gram and Word2Vec

Parallel Document Context

Hierarchical Document Context

ConceptNet Numberbatch

**Clean words: True** – The model only keeps alphanumeric characters and "\_", "-", using this parameter slightly improved our performance for **Skip gram** but had little to no impact for **PDC and HDC**

**Lower:** True converted string to lowercase

LexVec

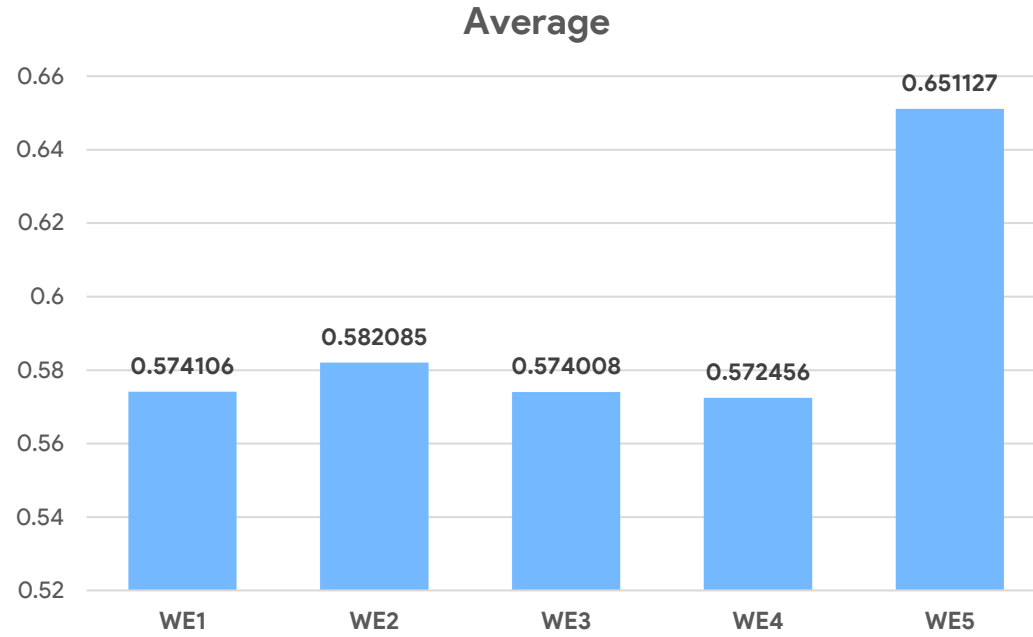
**Commoncrawl-w** – The model is pretrained using dataset extracted from commoncrawl data repository.

## Assignment 1 – Part B: Word Embedding Results: Similarity

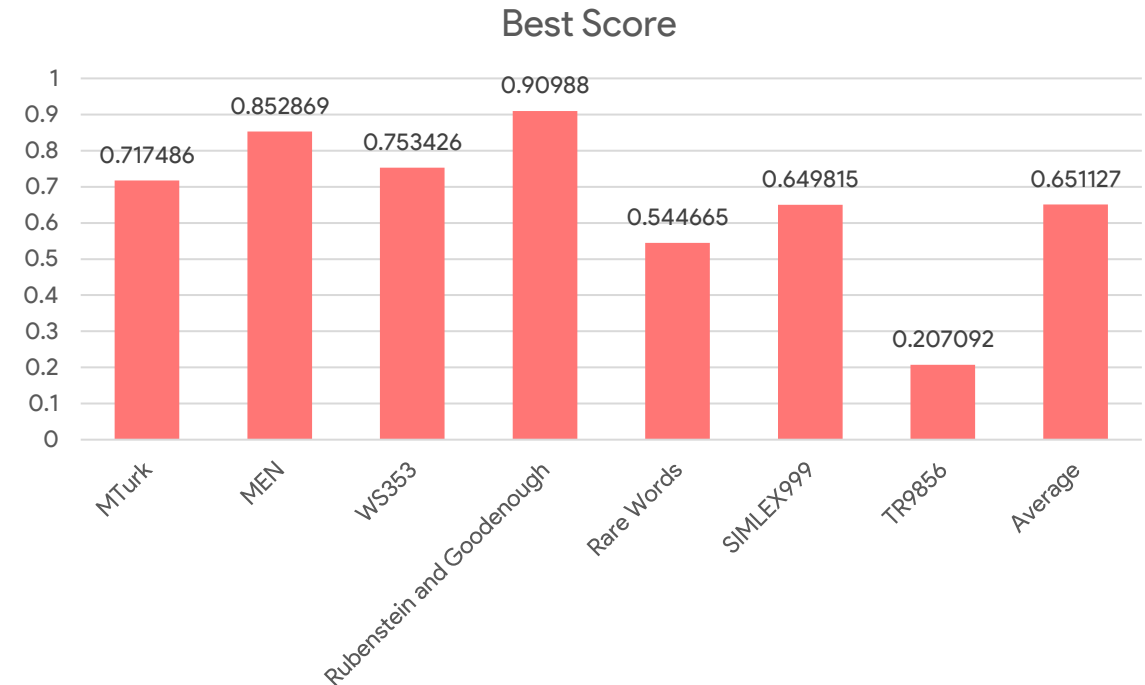
The ***similarity\_score*** is a metric of how semantically related *Word1* and *Word2* are. The score was generated by human annotators. The data pairs were generated using an algorithm as described in the paper by Radinsky et. al (2011). Each pair of words was evaluated by 10 people on a scale of 1-5 for enhanced accuracy

	WE1	WE2	WE3	WE4	WE5	Best Score
MTurk	0.681332	0.672333	0.657670	0.711555	0.717486	0.717486
MEN	0.758531	0.772648	0.760335	0.809187	0.852869	0.852869
WS353	0.700017	0.733431	0.716873	0.692889	0.753426	0.753426
Rubenstein and Goodenough	0.760783	0.790069	0.805805	0.764542	0.909880	0.909880
Rare Words	0.497105	0.472393	0.463447	0.489417	0.544665	0.544665
SIMLEX999	0.441966	0.426882	0.406832	0.419321	0.649815	0.649815
TR9856	0.179013	0.206839	0.207092	0.120280	0.129749	0.207092
Average	0.574106	0.582085	0.574008	0.572456	0.651127	0.651127

## Assignment 1 – Part B: Similarity Score Winner for average and Best score



While comparing the average score across all the datasets for particular word embedding model (for all 5). We observe that WE5 i.e. **concept numberbatch** performs the best overall with average score of **0.651** across all the dataset. Ensuring that with varying data/corpus the algorithm/model still works better than other.



While comparing the best score across all the word embedding algorithm on a particular dataset model (for all 8). We observe that Rubenstein and Goodenough dataset for WE5 i.e. **concept numberbatch** performs the best overall with best score of **0.90988**. Indicating that with for this data/corpus and the algorithm/model we can generate the best similarity score for the words



## Assignment 1 – Part B: Analogy Score Winner for average and Best score

Besides finding similar words, we can also apply word vectors to word analogy tasks. For example, “man”:“woman”::“son”:“daughter” is the form of a word analogy: “man” is to “woman” as “son” is to “daughter”. Specifically, the word analogy completion task can be defined as: for a word analogy  $a:b::c:d$  given the first three words  $aa$ ,  $bb$  and  $cc$ , find  $dd$ . Denote the vector of word  $w$  by  $vec(w)$ . To complete the analogy, we will find the word whose vector is most similar to the result of  $vec(c)+vec(b)-vec(a)$ .

	WE1	WE2	WE3	WE4	WE5	Best Score
Google_analogy	0.401862	0.747595	0.731273	0.710397	0.379298	0.747595
MSR	0.711875	0.596375	0.564375	0.601125	0.537000	0.711875
Average	0.556869	0.671985	0.647824	0.655761	0.458149	0.671985

## Assignment 1 – Part B: Analogy Score Winner for average and Best score

As compared to the similarity task results, the results for the analogy task had much smaller difference in results.

- Apart from WE1(skipgram), all the word embedding algorithms seem to perform better for google news over MSR dataset.
- On an average WE2(PDC) has the best average score as it edges out in google analogy dataset (which is also the best score across all datasets and WE algorithms).
- Apart from this the best performing model for MSR is WE1 (skipgram) with a score of over 0.71

