CSI5386

Natural Language Processing

Assignment 1

Corpus analysis and word embeddings

Submitted By:

Nehal Aggarwal 300092092

Ashish Verma 300059114

Part 1: Corpus processing: tokenization, and word counting

Problem definition: Implement a word tokenizer for Twitter messages that splits the text of the messages into tokens and separates punctuation marks and other symbols from the words.

Pre-Tokenization steps

Pre-processing of dataset

Tools Used: NLTK, TweetTokenizer, Regular expression package python (re), Contractions, NLTK.tokenize.RegexpTokenizer, NLTK.corpus.Stopwords.

We analyzed corpus of 48401 twitter messages and below are our observations. All the examples below correspond to individual functionality. All the methods were then sequentially applied to get complete clean tweet.

Important: TweetTokenizer separates hashtags into 2 tokens. For e.g. #jobs will separated as #, jobs. Below results are based on this tokenization process since some meaningful words can be extracted from hashtags e.g. *jobs* is a valid meaningful word. Later, # symbol was removed.

1. **Twitter handles**: Twitter handles were removed using NLTK TweetTokenizer library. In this parameter to remove twitter handles was set to true.

Original Tweet	Cleaned tweet
@samanthaprabu sam i knw u r a cricket fan r u watching any of the world cup matches	sam i knw u r a cricket fan r u watching any of the world cup matches

2. **URLs**: There were URLs which should be removed

Original Tweet	Cleaned tweet
Save BBC World Service from Savage Cuts http://www.petitionbuzz.com/petitions/savews	Save BBC World Service from Savage Cuts

3. **HTML tags**: There were tweets with HTML tags from which text should be extracted

Original Tweet	Cleaned tweet
#1 China Wholesale Store Clothing/Shoes/Handbags/MP4/DVDs/ Cell Phones/Watches/Electronics.	#1 China Wholesale Store Clothing/Shoes/Handbags/MP4/DVDs/ Cell Phones/Watches/Electronics.

4. **Contractions**: There were contractions in the text which were expanded and replaced.

Original Tweet	Cleaned tweet
"I'd wear any of my private attire for the world to see.'' BORNTHISWAYFRIDAY "	"I would wear any of my private attire for the world to see.'' BORNTHISWAYFRIDAY "

5. **Non-uniformity with the case**: All the tweets were moved to lower-case, inorder to simplify more conditions related to removal of retweet keyword since it can appear in both RT and rt manner.

Original Tweet	Cleaned tweet
"I'd wear any of my private attire for the world to see.'' BORNTHISWAYFRIDAY "	"i would wear any of my private attire for the world to see.'' bornthiswayfriday"

6. **Re-Tweet Keyword**: All the rt or RT keywords were removed from tweets after converting them to lowercase.

Original Tweet	Cleaned tweet
RT @YourMansHouse: I Just Wanna Cuddle In Bed & Fall Asleep :(Wheres My Superman? < Saving the world	@YourMansHouse: I Just Wanna Cuddle In Bed & Fall Asleep :(Wheres My Superman? < Saving the world

7. **Hashtags**: Tweets have trend to be accompanied by set of hashtags, which makes no sense.

Original Tweet	Cleaned tweet
#computers #gadgets BBC iPlayer app landing on UK iPads this Thursday, US in June http://dlvr.it/G345v #slingnews	BBC iPlayer app landing on UK iPads this Thursday, US in June http://dlvr.it/G345v

8. **Digits**: Some of the tweets have numbers which were removed to have English text.

Original Tweet	Cleaned tweet
SEPTA BUS No service on Rtes 1, 5, 7, 8, 12, 19, 22, 23, 24, 25, 26, 30, 31, 32, 33, 35, 38, 39, 40, 44, 47, 47m, 48, 53, 55, 57, 59, 61, 62	SEPTA BUS No service on Rtes ,

9. **Punctuation marks:** Some of the tweets have unnecessary punctuation marks, which was removed.

Original Tweet	Cleaned tweet

this @brainfeeder show on BBC right now is KILLING!! http://www.bbc.co.uk/iplayer/console/bbc_radio_one definitely don't wanna miss this	this @brainfeeder show on BBC right now is KILLING http://www.bbc.co.uk/iplayer/console/bbc_radio_one definitely don't wanna miss this

10. **Underscore sign:** Some of the underscore were not removed after analyzing cleaned tweets. Another method was written to underscore signs.

Original Tweet	Cleaned tweet
This girls breathe is kicking more than a soccer team -___-	This girls breathe is kicking more than a soccer team -

11. **Symbols:** Some of tweets have non-English characters, and symbols which were removed.

Original Tweet	Cleaned tweet
Why are so many people ashamed to admit your in love? Love is a good thing so express it! ❤	Why are so many people ashamed to admit your in love? Love is a good thing so express it!
4gameのキャサリン推しがすごい。同日発売のtwo worldsは公式更新すらネタにしないのにアフロ女とか正直どうでも…	4game two worlds

All the above methods were logically and sequentially applied to each tweet to get a clean and valid corpus for analysis. Some of the examples of clean tweets are below:

1. the game is over the world big game so over too
2. i just want to cuddle in bed fall asleep where is my superman saving the world
3. bbc news sri lankan adventures to lure tourists
4. it is official people in the future live in ikea showrooms

For **Part A, B, C and D** below punctuation, stopwords, and other symbols (which include numerals, hashtags, non-English characters) were not removed since, they were required to be removed for **Part E, F and G.**

<u>Part A</u>: Submit a file microblog2011_tokenized.txt with the tokenizer's output for the whole corpus. Include in your report the output for the first 20 sentences in the corpus.

After cleaning the corpus, the whole corpus was tokenized. Each line in **microblog2011_tokenized.txt** correspond to one token. Tokens for first 20 messages are below.

Tweet 1	Tweet 2	Tweet 3	Tweet 4	Tweet 5	Tweet 6	Tweet 7
save	a	rethink	'	latest	cdt	territory
bbc	lot	group	zombie	:	presents	manager
world	of	positive	'	:	alice	:
service	people	in	fund	top	in	location
from	always	outlook	manager	world	wonderland	:
savage	make	:	phoenix	releases	-	calgary
cuts	fun	technology	appoints		catonsville	,
	about	staffing	new		dinner	alberta
	the	specialist	ceo		has	,
	end	the	:		posted	canada
	of	rethink	phoenix		'	job
	the	group	buys		cdt	category
	world	expects	up		presents	:
	but	revenues	funds		alice	bu
	the	to	that		in	...
	question	be	have		wonderland	#
	is	"	been		'	jobs
	..	marg	closed		to	
	``	...	to		the	
	are		new		...	
	you		business			
	ready		and			
	for		...			
	it					
	?					
	..					

Tweet 8	Tweet 9	Tweet 10	Tweet 11	Tweet 12	Tweet 13
i	bbc	manchester	,	fitness	david
cud	news	city	if	first	cook
murder	-	council	you	to	!
sum1	today	details	are	float	has
today	-	saving	interested	but	the
n	free	cuts	in	is	mostest
not	school	plan	professional	not	beautiful

even flinch I am tht fukin angry today	funding plans ' lack transparency ' - …	: ... depressing . apparently we ’ re 4th most deprived & top 5 hardest hit	global translation services	the full service model dead ?	smile in the world !

Tweet 14	Tweet 15	Tweet 16	Tweet 17	Tweet 18	Tweet 19
piss off . cnt stand lick asses	beware the blue meanies : # cuts # thebluemeanies	como perde os dentes no world of warcraft - via alisson	how exciting ! : hello ! what is happening in your world ? we are all gearing up for # valentines with bouquets flying out	I would very much appreciate it if people would stop broadcasting asking me to add people on bbm .	sam i knw you r a cricket fan r you watching any of the world cup matches

			the door .		

Tweet 20
john baer : who did not see this coming ? : to those who know ed and midge rendell - heck , to the philly world at la ...

PART B: How many tokens did you find in the corpus? How many types (unique tokens) did you have? What is the type/token ratio for the corpus? The type/token ratio is defined as the number of types divided by the number of tokens.

Number of Tokens in corpus are: 880781

Number of Unique Tokens in corpus are: 61426

Type token ratio of corpus is: 0.069740378141672 i.e. (880781/61426)

PART C: For each token, print the token and its frequency in a file called Tokens.txt (from the most frequent to the least frequent) and include the first 100 lines in your report.

Token	Frequency	Token	Frequency	Token	Frequency	Token	Frequency
:	25578	with	3142	all	1562	how	1123
#	22600	my	3122	|	1532	more	1118
the	21106	that	3035	what	1478	de	1095
.	21032	new	2969	no	1465	union	1054
,	18922	&	2754	now	1448	people	1052
to	14358	news	2566	..	1414	*	1023
...	13444	will	2503	was	1403	he	1022
!	11336	from	2469	so	1393	who	1019
in	10724	this	2441	super	1391	security	1014
-	10668	/	2300	…	1387	love	1003
of	10559	be	2264	an	1370	or	998
a	10524	have	2239	up	1367	airport	998
and	8366	I	2127	25-Jan	1342	today	991
is	7952	am	2119	world	1331		
for	7099	by	2050	via	1326		
i	6520	do	1969	media	1325		
on	6262	egyptian	1887	social	1321		
you	5728	your	1823	white	1312		
?	5704	us	1821	2	1307		
)	5611	me	1819	like	1303		
``	5507	obama	1818	bowl	1286		
(	5054	state	1780	get	1284		
not	4390	s	1778	about	1279		
's	3980	just	1740	but	1270		
at	3932	as	1733	’	1241		
it	3843	we	1727	if	1226		
'	3500	can	1669	$	1222		
egypt	3271	out	1630	they	1199		
are	3152	has	1588	2011	1140		

PART D: How many tokens appeared only once in the corpus?

Number of tokens that appeared only once are: 35822

PART E: From the list of tokens, extract only words, by excluding punctuation and other symbols, including Twitter specific symbols. How many words did you find? List the top 100 most frequent words in your report, with their frequencies. What is the type/token ratio when you use only word tokens (called lexical diversity)?

Number of Tokens in corpus with only words are: 672044

Number of Unique Tokens in corpus with only words are: 48397

Type token ratio of corpus with only words is: 0.07201462999446465

Top 100 most frequent words after completely cleaning the corpus:

Token	Frequency	Token	Frequency	Token	Frequency	Token	Frequency
the	21160	am	2213	via	1327	president	979
to	14389	I	2130	world	1318	release	961
in	10830	by	2062	like	1317	one	957
a	10633	do	1976	media	1317	time	951
of	10575	egyptian	1846	white	1310	law	929
and	8386	me	1836	social	1308	his	922
is	7958	us	1835	get	1290	good	895
for	7111	your	1824	bowl	1289	house	885
i	6572	state	1796	about	1278	video	864
on	6298	just	1748	but	1276	over	863
s	5821	as	1740	if	1228	when	850
you	5797	we	1736	they	1209	show	848
not	4392	obama	1707	how	1129	our	822
at	3957	egypt	1702	more	1122	protests	822
it	3862	can	1672	de	1109	service	810
are	3163	out	1650	people	1053	got	810
with	3143	all	1594	union	1050	go	800
my	3122	has	1588	he	1027	lol	775
that	3036	no	1532	day	1023	after	764
new	2977	what	1480	who	1020	going	748
will	2516	up	1466	or	1000	phone	747
from	2470	now	1458	com	999	there	746
this	2443	super	1415	today	995		
be	2273	so	1407	love	992		
have	2243	was	1404	security	986		
news	2234	an	1386	airport	981		

PART F: From the list of words, exclude stopwords. List the top 100 most frequent words and their frequencies in your report. You can use this list of stopwords (or any other that you consider adequate). Also compute the type/token ratio when you use only word tokens without stopwords (called lexical density)?

Since, there was freedom to use any list of stop-words, we use the one defined in NLTK library. It was pulled and directly used to remove stop-words. It can be found here: https://gist.github.com/sebleier/554280.

Number of Tokens in corpus with only words and excluding stop words are: 450941

Number of Unique Tokens in corpus with only words and excluding stop words are: 48250

Type token ratio of corpus with only words and excluding stop words is: 0.10699847651910117

Top 100 most frequent words after removing stopwords are:

Token	Frequency	Token	Frequency	Token	Frequency	Token	Frequency
new	2977	security	986	free	695	year	580
news	2234	airport	981	dog	690	great	565
I	2130	president	979	back	685	think	564
egyptian	1846	release	961	global	682	make	559
us	1835	one	957	w	670	best	550
state	1796	time	951	want	668	man	547
obama	1707	law	929	taco	668	tv	544
egypt	1702	good	895	bell	666	court	544
super	1415	house	885	cairo	657	crash	527
via	1327	video	864	bbc	649	al	526
world	1318	show	848	protesters	642	twitter	524
like	1317	protests	822	live	638	post	521
media	1317	service	810	return	634	watch	511
white	1310	got	810	rite	629	home	510
social	1308	go	800	need	626	cuts	509
get	1290	lol	775	special	607	budget	508
bowl	1289	going	748	ap	605	health	502
de	1109	phone	747	know	602	weather	498
people	1053	says	743	last	600	say	494
union	1050	would	741	see	599	la	491
day	1023	mubarak	730	first	598	top	490
com	999	job	729	jobs	595	could	485
today	995	police	715	toyota	587	b	481
love	992	energy	710	n	580	th	480

Token	Frequency
government	479
online	477
right	475
week	473

PART G: Compute all the pairs of two consecutive words (bigrams) (excluding stopwords and punctuation). List the most frequent 100 pairs and their frequencies in your report.

Top 100 most frequent Bi-grams are:

Bigram	Frequency	Bigram	Frequency
('super', 'bowl')	1205	('supreme', 'court')	122
('social', 'media')	984	('world', 'news')	118
('state', 'union')	925	('egypt', 'protests')	118
('taco', 'bell')	580	('glenn', 'beck')	118
('white', 'house')	367	('ap', 'ap')	117
('union', 'address')	363	('tahrir', 'square')	110
('new', 'york')	357	('egyptian', 'protesters')	110
('global', 'warming')	315	('birth', 'certificate')	109
('president', 'obama')	269	('middle', 'east')	108
('keith', 'olbermann')	268	('de', 'la')	106
('al', 'jazeera')	256	('last', 'night')	105
('bowl', 'xlv')	210	('egyptian', 'museum')	105
('white', 'stripes')	205	('release', 'date')	105
('world', 'cup')	201	('prime', 'minister')	103
('barack', 'obama')	195	('blog', 'post')	103
('I', 'going')	183	('world', 'service')	102
('bbc', 'news')	182	('phone', 'hacking')	100
('rahm', 'emanuel')	181	('hillary', 'clinton')	99
('health', 'care')	176	('mph', 'kt')	99
('united', 'states')	174	('egyptian', 'protests')	99
('moscow', 'airport')	173	('breaking', 'news')	98
('let', 'us')	170	('year', 'old')	97
('obama', 'state')	161	('president', 'hosni')	93
('press', 'release')	145	('bbc', 'world')	92
('budget', 'cuts')	143	('end', 'date')	92
('customer', 'service')	142	('media', 'marketing')	92
('julian', 'assange')	141	('egyptian', 'president')	91
('president', 'barack')	134	('protests', 'egypt')	88
('hosni', 'mubarak')	129	('msnbc', 'com')	87
('new', 'post')	123	('current', 'tv')	87
('youtube', 'video')	122	('federal', 'judge')	86

Bigram	Frequency
('new', 'blog')	85
('got', 'part')	84
('egyptian', 'police')	83
('n', 'ap')	83
('I', 'would')	80
('union', 'speech')	80
('anti', 'government')	79
('bid', 'end')	77
('new', 'job')	76
('kate', 'middleton')	76
('obama', 'birth')	75
('security', 'forces')	73
('fair', 'f')	73
('weight', 'loss')	70
('airport', 'security')	70
('looks', 'like')	69
('us', 'bid')	69
('secretary', 'state')	69
('feel', 'like')	68
('share', 'friends')	68
('egyptian', 'government')	68
('fox', 'news')	67
('egyptian', 'people')	67
('special', 'olympics')	67
('w', 'others')	66
('egyptian', 'embassy')	66
('international', 'airport')	66
('cowboys', 'stadium')	65
('care', 'law')	65
('gabrielle', 'giffords')	65
('tear', 'gas')	64
('fifa', 'soccer')	63
('louis', 'vuitton')	63
('unemployment', 'rate')	63
('egyptian', 'army')	62
('state', 'tv')	61
('anthony', 'hopkins')	61
('last', 'week')	60

Part 2: Evaluation of word embeddings

Pre-Evaluation

To implement this task first it was required to find relevant word embeddings models and improve the GitHub code as shared. At first there were few challenges due to SSL handshake errors and broken links to download the word embeddings. Below were the steps followed to get the similarity and analogy tasks executable:

1. Unverified context was set in the python code so that it could download over HTTPS connections and doesn't verify certificate. This was required to perform TR9856 similarity task.
2. A new method was written and implemented to load static word embeddings from drive by specifying only the path.

To evaluate word embeddings, below are the different models/algorithms considered for this task:

S.No.	Model
1.	Glove (Global Vectors)
2.	Continuous Bag-of-Words
3.	Skipgram
4.	PDC
5.	LexVec
6.	Concept Net Number Batch
7.	HDC
8.	FastText

Additional models were downloaded from http://vectors.nlpl.eu/repository/ but were later discarded to select best 8 word embeddings.

Mechanism behind each model

1. **Glove (Global Vectors):** All the models are built to predict surrounding words in a sentence. A Glove is an unsupervised learning algorithm to calculate vectors of words. In Glove model first, a co-occurrence matrix is built. The main feature of glove model is that it considers local context while calculating co-occurrence matrix by using fixed window size. After calculating vectors, they are evaluated as ratios of co-occurrence between two words in a context. This also implies that they are strongly connected to meaning.

2. **Continuous Bag-of-Words (CBOW):** Unlike Glove, CBOW is a prediction-based vector. Given a context, it tries to predict the probability of a word. A context can be both single

or group of words. In this one-hot encoded matrix if formed for a given corpus. This matrix is then passed to a neural network of input, hidden and output layer. The weights between input – hidden layer and hidden – output layer is multiplied by input and output is calculated. The error is calculated and back-propagated to re-adjust weights. The vector result is weight between hidden – output layer for the given word. The main advantage of CBOW model is, it requires less memory than Glove model.

3. **SkipGram:** SkipGram is similar to CBOW, the only difference is that it tries to predict context given a word. Which is just opposite of what CBOW. In this more than one target variables are output. In this model errors are calculated for one or more target variables and error vectors are added element wise to have final error vector which is further back-propagated to adjust weights. Weights between input-hidden layer is taken as vector representation of target variable. Its main use is, it can predict two semantics of single words unlike Glove and CBOW.

4. **LexVec:** LexVec is combination of Glove and Skipgram model with negative sampling. The only difference if LexVec is that it factorizes PPMI matrix which essentially co-occurrence matrix calculated for a given word and window of context words. LexVec uses two loss functions Mini-Batch and Stochastic descent. Mini-Batch is executed in same manner as in SkipGram, wherein, weights are updated whenever a pair is observed by window sampling and pairs drawn by negative sampling. It updates weights by assigning heavy penalties for errors on frequent co-occurrences while still accounting for negative co-occurrences.

5. **Concept Net Numberbatch:** ConceptNet is a semantic network of knowledge about word meaning. It is sparse graph modelled using matrix where each cell contains sum of the weights of all edges that connect the two corresponding terms. This model is further extended by combining distributional word embeddings using a method called as retrofitting, which adjusts existing matrix of word embeddings using a knowledge graph. ConceptNet Numberbatch is a hybrid model of ConceptNet5.5, word2vec and Glove word embeddings.

6. **PDC:** In Parallel Document Context (PDC) Model, a target word is predicted by its surrounding context and parallelly predicts in document it occurs. The first task captures paradigmatic relations as words with similar context will have similar representation. The word prediction in document models syntagmatic relations, as words co-occur in same document will have similar representation. Its an extension of CBOW model with added layer for document task. As both context and document are parallel in predicting the word it is called as Parallel Document Context model. Stochastic gradient descent is used for optimization and gradient is calculated via back-propagation algorithm.

7. **HDC:** This Hierarchical Document Context (HDC) is similar to above PDC model, the only difference is the document is used to predict a target word and this target word is further used to predict its surrounding context words. As we can understand prediction is done in hierarchical way, so it is named as Hierarchical Document Context model. The syntagmatic relation is modeled by document-word prediction layer and paradigmatic relation is modeled by word-context prediction layer. In this model also Stochastic gradient descent is used for optimization and gradient is calculated via back-propagation algorithm.

8. **FastText:** FastText tends to learn vector representation of each character n-gram for a given word and not just only the word. Each word is represented as a bag of character n-grams. Eventually, the word embedding is sum of all these character n-gram. This model focusses more on character level embeddings so that it generates embeddings for rare and out of corpus words. FastText exists as learning to both unsupervised representation and supervised text classification. In unsupervised, a skipgram model with negative sampling is used to calculate similarity between fixed target word and contextual word. Here the similarity function is dot-product between two words represented as sum of n-gram vectors. For text classification, each word is sum of its n-grams. A latent text embedding is calculated to predict text label by calculating average over word embeddings.

Word Embeddings

In total we evaluated 12-word embeddings. All these 12 models are listed in Appendix-1. Out of these 12, we chose 8 best and distinct models based on results.

Word Embeddings	Model	Corpus
WE1	Glove	Wikipedia-6B
WE6	Gensim Continuous Bag-of-Words	WIkipedia corpus
WE7	Gensim Continuous Skipgram	Wikipedia Dump of February 2017
WE8	PDC	Wikipedia
WE9	LexVec	Wikipedia newscrawl
WE10	Concept Net Number Batch	English words
WE11	HDC	Wikipedia
WE12	FastText	Wikipedia news

Evaluation Tasks

Two tasks were evaluated against different benchmark datasets. Below are the list of tasks and benchmark dataset used:

1. **Similarity task**: In this tasks we measure cosine similarity (degree of similarity) for two embedding word vector, to predict how similar the two words are.
    a. **Benchmark Datasets used:** MTurk, MEN, WS353, Rubenstein and Goodenough, Rare Words, SimLex999, TR9856
2. **Analogy task:** In this task we try to complete a sentence of form 'w' is to 'x' as 'y' is to '___'. We try to find word 'z' such that its word vector is associated and related to x-w and z-y.
    a. Benchmark Dataset used: MSR WordRep, Google_analogy, MSR, SEMEVAL 2012 Task 2

Evaluation Results

All the word embeddings were evaluated by keeping these parameters constant throughout:

Parameter	Description
normalize=True	To normalize all vector to unit length
lower=False	Case of words were kept as-is.
clean_words=False	To avoid alphanumeric characters and "_", "-"
Dimension (dim): length of vectors	300 (kept it uniform throughout different models)

Similarity tasks results

Below table document results of similarity task across all models.

Tasks	Similarity tasks							Overall Average
Word embeddings	MTurk	MEN	WS353	Rubenstein and Goodenough	Rare Words	SimLex999	TR9856	
WE1	0.633181998	0.737464697	0.543325561	0.769524979	0.36698186	0.370500357	0.096716734	0.502528027
WE6	0.263398794	0.360702193	0.333565455	0.326998942	0.182599091	0.169714158	0.13748553	0.25349488
WE7	0.542700093	0.739921018	0.717734202	0.806810114	0.407218989	0.396398552	0.139981084	0.535823556
WE8	0.672333057	0.772648337	0.73352982	0.790068957	0.472392698	0.426882404	0.207274917	0.582161456
WE9	0.655410546	0.751388395	0.621565099	0.747057734	0.456229758	0.385223435	0.147233309	0.537729754

WE10	0.719718323	0.859637746	0.754611288	0.909879691	0.545442212	0.650524847	0.132788051	0.65322888
WE11	0.657670 13	0.760335161	0.716873204	0.805804771	0.463446735	0.406832371	0.207092411	0.574007826
WE12	0.702549 09	0.790632357	0.733276329	0.846258924	0.513407502	0.449964919	0.157883094	0.599138888

It can be seen from table above that most of the models are above 50% of overall prediction accuracy for similarity tasks. **WE10** leads with 65% accuracy overall for similarity tasks, whereas **WE6** has lowest accuracy of 25%.

Analogy task results

Below table document results of analogy task across all models.

Tasks	**Analogy tasks**					**Overall Average**
Word embeddings	**MSR WordRep**		**Google_analogy**	**MSR**	**SEMEVAL 2012 Task 2**	
	All	**Wordnet**				
WE1	0.29	0.575	0.71735571	0.61425	0.163963	0.446392178
WE6	0.018	0.025	0.127302497	0.0775	0.074303061	0.07427639
WE7	0.152	0.308333	0.321428571	0.5565	0.19394744	0.305969003
WE8	0.25	0.5	0.74759517	0.596375	0.174073631	0.44201095
WE9	0.258	0.516667	0.728305362	0.57375	0.198185	0.439560091
WE10	0.12	0.25	0.381242325	0.539375	0.238101814	0.319679785
WE11	0.254	0.504167	0.731273025	0.564375	0.184511345	0.433539843
WE12	0.264	0.516667	0.59250921	0.813	0.219977793	0.472371751

It can be seen from table above that most of the models are above 50% of overall prediction accuracy for similarity tasks. **WE12** leads with 47% accuracy overall for analogy tasks, whereas **WE6** has lowest accuracy of 7%.

Overall tasks result with average

Below are results of 8 best and distinct models. Results to all 12 models are documented under Appendix-2 for reference. All the numerical result corresponds to **predicted accuracy**.

Tasks	Similarity tasks							Analogy tasks					Overall Average
Word embeddings	MTurk	MEN	WS353	Rubenstein and Goodenough	Rare Words	SimLex999	TR9856	MSR WordRep All	MSR WordRep Wordnet	Google_analogy	MSR	SEMEVAL 2012 Task 2	
WE1	0.6333	0.774	0.5433	0.769524979	0.36698	0.3705	0.09067	0.29	0.575	0.71735571	0.61425	0.163963	0.48211499 1
WE6	0.2633	0.336	0.334	0.326998942	0.18259	0.16971	0.1375	0.018	0.025	0.12730249 7	0.07745	0.074303061	0.18832452
WE7	0.5433	0.774	0.718	0.806810114	0.40721 9	0.39639 9	0.14	0.152	0.30833 3	0.32142857 1	0.5565	0.19394744	0.45224008 2
WE8	0.672	0.777	0.734	0.790068957	0.47239 27	0.40268 82	0.20 73	0.25	0.5	0.74759517	0.59637 5	0.174073631	0.53119763 6
WE9	0.655	0.75	0.622	0.747057734	0.45622 98	0.38523	0.14 72	0.2582	0.51666 7	0.72830536 2	0.57375	0.198185	0.50203169 5
WE10	0.72	0.866	0.755	0.909879691	0.54544 22	0.65052 5	0.1328	0.12	0.25	0.38124232 5	0.53937 5	0.238101814	0.5319383
WE11	0.658	0.776	0.717	0.805804771	0.46344 67	0.40680 32	0.2071	0.254	0.50416 7	0.73127302 5	0.56437 5	0.184511345	0.52292855 9
WE12	0.703	0.79	0.733	0.846258924	0.51340 75	0.44996 5	0.1579	0.264	0.51666 7	0.59250921	0.813	0.219977793	0.55304174 7

From the table above, it can be observed that models **WE12, WE10 and WE8** performed well overall. However, models **WE11 and WE9** are also above 50% of overall accuracy.

Conclusion

Top 3-word embeddings which performed well on overall predicted accuracy are:

S.No.	Word Embeddings
1	WE12: FastText
2	WE10: Concept Net Number batch
3	WE8: PDC

Challenges

1. Due to limited computing capabilities, we were not able to evaluate on MSR WordRep for higher number of pairs. If evaluated for higher number of pairs, average on all datasets could be better. We tried evaluating for higher number of pairs for analogy tasks on cloud but our Jupyter notebook crashed multiple times and it took more than 4 hours to execute per word embedding.

Division of tasks:

All the tasks were performed together and peer reviewed. The only task that differed was to find best word embedding. Later, all the models were compiled and evaluated in the presence of both team members.

Appendix-1

List of all 12 models with parameters used.

Word Embeddings Key	Model	Corpus	Dimension
WE1	Glove	Wikipedia-6B	300
WE2	Gensim continuous Skipgram	Gigaword 5th Edition	300
WE3	Glove	Gigaword 5th Edition	300
WE4	Glove	English Wikipedia Dump of February 2017	300
WE5	Glove	English Wikipedia Dump of February 2017 and Gigaword 5th Edition	300
WE6	Gensim Continuous Bag-of-Words	Wikipedia corpus	400
WE7	Gensim Continuous Skipgram	Wikipedia Dump of February 2017	300
WE8	PDC	Wikipedia	300
WE9	LexVec	Wikipedia news	300
WE10	Concept Net Number batch	English words	300

WE11	HDC	Wikipedia	300
WE12	FastText	Wikipedia news	300

Appendix-2

Results of all 12 models are below. Similar color rows represent that algorithm or model behind generating these word embeddings is same.

Tasks	**Similarity tasks**							**Analogy tasks**				**Overall Average**
Word embeddings	**MTurk**	**MEN**	**WS353**	**Rubenstein and Goodenough**	**Rare Words**	**SimLex999**	**TR9856**	**MSR WordRep All**	**MSR WordRep Wordnet**	**Google_analogy**	**MSR**	**SEMEVAL 2012 Task 2**	
WE1	0.633	0.74	0.543	0.769524979	0.3669819	0.3705	0.09967	0.29	0.575	0.7173571	0.614125	0.163963	0.4821149
WE2	0.617	0.73	0.619	0.703434563	0.2775156	0.408592	0.1273	0.1488	0.283333	0.5328489	0.302875	0.157796	0.4207747
WE3	0.526	0.72	0.524	0.735671125	0.2602366	0.384376	0.08064	0.1322	0.254167	0.49759517	0.21675	0.167859	0.3863215
WE4	0.567	0.74	0.638	0.809607592	0.2881588	0.363224	0.08042	0.1088	0.220833	0.30265043	0.51075	0.1795	0.4173342
WE5	0.544	0.76	0.629	0.780408916	0.3263283	0.368989	0.08009	0.1766	0.341667	0.3194330 74	0.5855	0.163677997	0.4301017
WE6	0.263	0.36	0.334	0.326998942	0.1825991	0.169714	0.1375	0.018	0.025	0.1273024 97	0.0775	0.074303061	0.1883245
WE7	0.543	0.74	0.718	0.806810114	0.4072219	0.396399	0.14	0.152	0.308333	0.3214285 71	0.5565	0.19394744	0.4522400
WE8	0.672	0.77	0.734	0.790068957	0.4723927	0.426882	0.2073	0.25	0.5	0.74759517	0.596375	0.174073631	0.5311976

WE9	0.655	0.75	0.622	0.747057734	0.45622	0.385223	0.1472	0.2258	0.516667	0.72830536	0.57375	0.198185	0.5020316
WE10	0.72	0.86	0.755	0.909879691	0.54544	0.650525	0.1328	0.12	0.25	0.3812423	0.53937	0.238101	0.5319383
WE11	0.658	0.76	0.717	0.805804771	0.46344	0.406832	0.2071	0.254	0.504167	0.7312730	0.56437	0.184511	0.5229285
WE12	0.703	0.79	0.733	0.846258924	0.51340	0.449965	0.1579	0.264	0.516667	0.59250921	0.813	0.219977	0.5530417

References

1. https://www.aclweb.org/anthology/P16-2068.pdf
2. http://vectors.nlpl.eu/repository/#
3. http://ofey.me/projects/wordrep/
4. https://github.com/kudkudak/word-embeddings-benchmarks
5. https://github.com/commonsense/conceptnet-numberbatch
6. https://github.com/alexandres/lexvec
7. https://fasttext.cc/docs/en/english-vectors.html
8. https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
9. https://towardsdatascience.com/beyond-word-embeddings-part-2-word-vectors-nlp-modeling-from-bow-to-bert-4ebd4711d0ec
10. https://towardsdatascience.com/fasttext-under-the-hood-11efc57b2b3