Words Embedding Report

Xia Song
PID A53228931

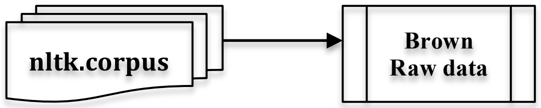
Introduction: For this project, the basic idea is words that tend to appear in similar context are likely to be related. You shall know a word by the company it keeps (Firth, 1957). Standing on this concept, we mainly investigate an embedding of words that is based on co-occurrence statistics.

Data source: Brown corpus is a collection of text samples from a wide range of sources, with a total of over a million words. Our analysis is mainly based on Brown corpus.

The followings are the detailed analysis steps.

1. Description of 100-dimensional embedding

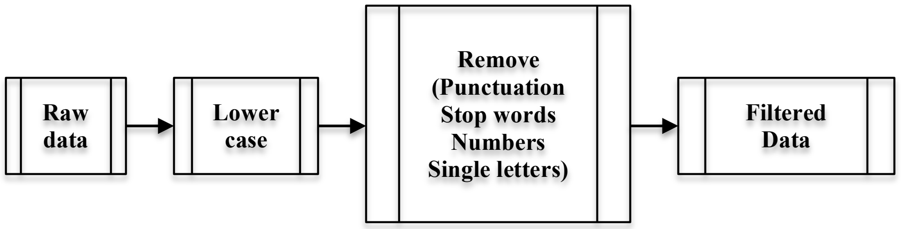
1.1 Download brown corpus data from nltk.corpus (brown.words).



1.2 Import the stop words from nltk.corpus, meanwhile import the punctuation from string. However, the punctuation of string missed some punctuation. Combining the string and extra punctuations appeared in brown.words, I create a new punctuations list (punctuation).

punctuation = ['!', '"', '#', '$', '%', '&', "'", '(', ')', '*', '+', ',', '-', '.', '/', ':', '; ', '<', '=', '>', '?', '@', '[', '\\', ']', '^', '_', '`', '{', '|', '}', '~', '``', "''", '--']
(Note: red punctuations are from string, and black punctuations are appeared in brown.words)

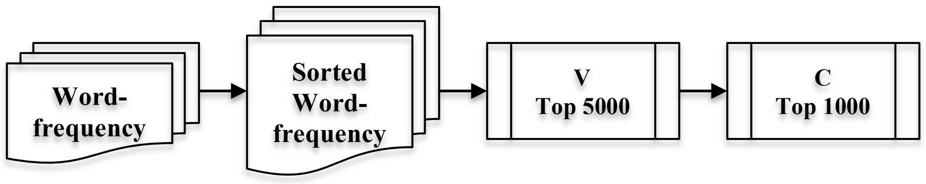
1.3 Data preprocessing and cleaning: lower case each word, removing punctuation (import from string), filtering stop words (import from nltk.corpus), removing numbers and single letters. At last, get the word list of filtered_words. All the following analyses are all based on the word list of filter_words.



1.4 Count each word frequency in filtered data, and form a word-frequency dictionary (words_count).

1.5 Sort the word frequency dictionary and create a new-sorted word list (sorted_words); select the top 5000 most commonly-occurring words from sorted dictionary, and form word list V. In this project the word list of V is used as our target words.

1.6 From word list of V, select the top 1000 most used words and form word list C. In this project the word list of C is used as our context words.



1.7 For each target word of V, find the positions of each occurrence of it in filtered data (positions); for each position forming a window (the window size include 5 words, target word in middle, two words before and two words after: [index-2:index]+filter_words[index+1:index+3] and index in positions), look at the surrounding four words, and count how often these surrounding words appear in context word list C (cword_fre).

1.8 According to the count of each context word in these windows (if appeared twice in one window, count one) and the total window number, construct the probability distribution Pr(c|w) (cword_fre / cword_win) of context words (c) around target word (w). Finally, combine the target word (Word), context word (Cword), context word frequency (Cwordfre), window number (Winum) and conditional probability (Pr_cw) into one data frame cwords.

Head of cwords data frame

	Word	Cword	Cwordfre	Winum	Pr_cw
0	one	major	12	3292	0.003645
1	one	make	33	3292	0.010024
2	one	wanted	8	3292	0.002430
3	one	wait	1	3292	0.000304
4	one	two	86	3292	0.026124

1.9 Meanwhile, for each context word of C, count the occurrence in filtered_words list (Count_c), and construct the overall probability distribution in preprocessed data set Pr(c) (Count_c / n, n is the length of preprocessed word list (filter_words)). Combine context words (Cont_words), context words frequency (Cont_counts), and the overall probability of context word (Pr_c) into one data frame Context_words.

Head of Context_words data frame

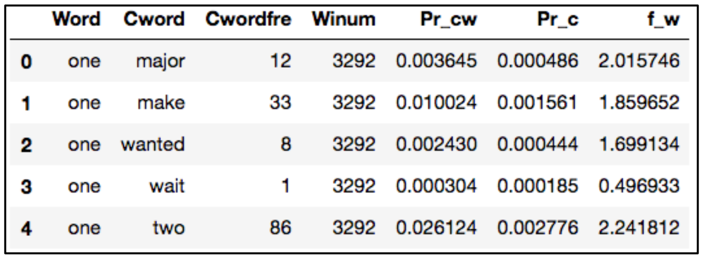
	Cont_counts	Cont_words	Pr_c
0	229	seem	0.000450
1	183	father	0.000360
2	2714	would	0.005336
3	100	places	0.000197
4	312	began	0.000613

1.10 According to the positive point wise mutual information to represent each target word (w) by context-word dimensional vector $\emptyset(w)$ (f_w)

$$\emptyset(w) = \max\left(0, log\frac{\Pr(c|w)}{\Pr(c)}\right)$$

1.11 Combine all the necessary variables into one table (cwords). These variables include target word (Word), context word (Cword), context word frequency (Cwordfre), window number for each target word (Winum), conditional Probabilty of context word appeared when target word occurs (Pr_cw), overall probability of context word in preprocessed data (Pr_c), and mutual information $\emptyset(w)$ (f_w). The following table is the head of combined table.

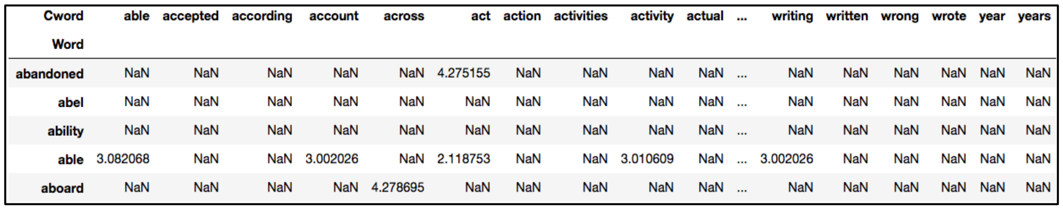
Head of cwords data frame



	Word	Cword	Cwordfre	Winum	Pr_cw	Pr_c	f_w
0	one	major	12	3292	0.003645	0.000486	2.015746
1	one	make	33	3292	0.010024	0.001561	1.859652
2	one	wanted	8	3292	0.002430	0.000444	1.699134
3	one	wait	1	3292	0.000304	0.000185	0.496933
4	one	two	86	3292	0.026124	0.002776	2.241812

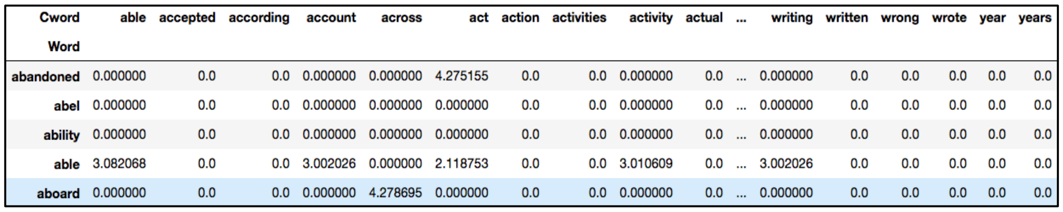
1.12 In order to do the feature reduce analysis, using pd.pivot_table stack Word (target word), Cword (context word), and f_w (mutual information) from cwords dataframe to construct a new data frame (mutal_words). For the new table, Word as index, Cwords as columns, and f_w as value. However, the new table is pretty sparse and includes many NaN values. In order for the following analysis, fill all the NaN values with 0.

Before NaN fill



Cword	able	accepted	according	account	across	act	action	activities	activity	actual	...	writing	written	wrong	wrote	year	years
Word																	
abandoned	NaN	NaN	NaN	NaN	NaN	4.275155	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
abel	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
ability	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN
able	3.082068	NaN	NaN	3.002026	NaN	2.118753	NaN	NaN	3.010609	NaN	...	3.002026	NaN	NaN	NaN	NaN	NaN
aboard	NaN	NaN	NaN	NaN	4.278695	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN

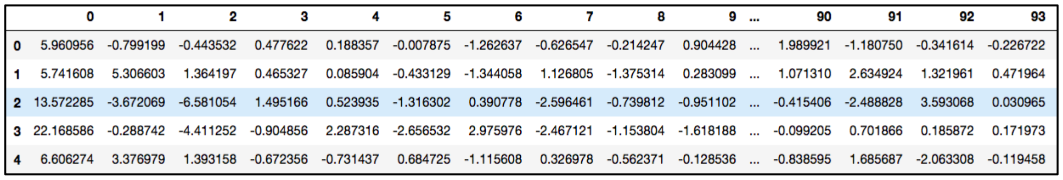
After Nan fill



Cword	able	accepted	according	account	across	act	action	activities	activity	actual	...	writing	written	wrong	wrote	year	years
Word																	
abandoned	0.000000	0.0	0.0	0.000000	0.000000	4.275155	0.0	0.0	0.000000	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.0
abel	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.0
ability	0.000000	0.0	0.0	0.000000	0.000000	0.000000	0.0	0.0	0.000000	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.0
able	3.082068	0.0	0.0	3.002026	0.000000	2.118753	0.0	0.0	3.010609	0.0	...	3.002026	0.0	0.0	0.0	0.0	0.0
aboard	0.000000	0.0	0.0	0.000000	4.278695	0.000000	0.0	0.0	0.000000	0.0	...	0.000000	0.0	0.0	0.0	0.0	0.0

1.13 This step is about the features selection. Features are Cword (1000). The above table shows us that our data are pretty sparse. When we do features selection, for high dimensional data, generally PCA is used to decompose the data. However, for the high dimensional sparse text data set (like our data set), singular Value Decomposition (SVD) is better. Therefore, in this project we use TruncatedSVD to decompose our data. Finally we get the 100-dimension-word representation.

Head of reduced feature dataframe



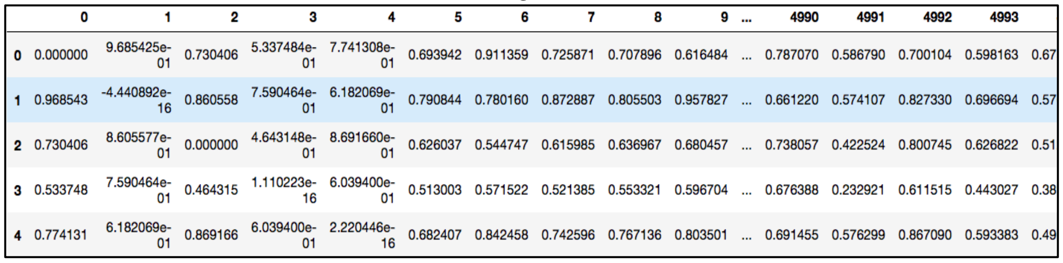
	0	1	2	3	4	5	6	7	8	9	...	90	91	92	93
0	5.960956	-0.799199	-0.443532	0.477622	0.188357	-0.007875	-1.262637	-0.626547	-0.214247	0.904428	...	1.989921	-1.180750	-0.341614	-0.226722
1	5.741608	5.306603	1.364197	0.465327	0.085904	-0.433129	-1.344058	1.126805	-1.375314	0.283099	...	1.071310	2.634924	1.321961	0.471964
2	13.572285	-3.672069	-6.581054	1.495166	0.523935	-1.316302	0.390778	-2.596461	-0.739812	-0.951102	...	-0.415406	-2.488828	3.593068	0.030965
3	22.168586	-0.288742	-4.411252	-0.904856	2.287316	-2.656532	2.975976	-2.467121	-1.153804	-1.618188	...	-0.099205	0.701866	0.185872	0.171973
4	6.606274	3.376979	1.393158	-0.672356	-0.731437	0.684725	-1.115608	0.326978	-0.562371	-0.128536	...	-0.838595	1.685687	-2.063308	-0.119458

2. Nearest neighbor results

2.1 Use package of cosine_similarity of sklearn.metrics.pairwise to calculate the cosine distance of each pair of target word, and get the symmetrical matrix (dist). Transform this matrix into a data frame (dist_df). For this table, each row represents the distance of one target word to all the other target word.

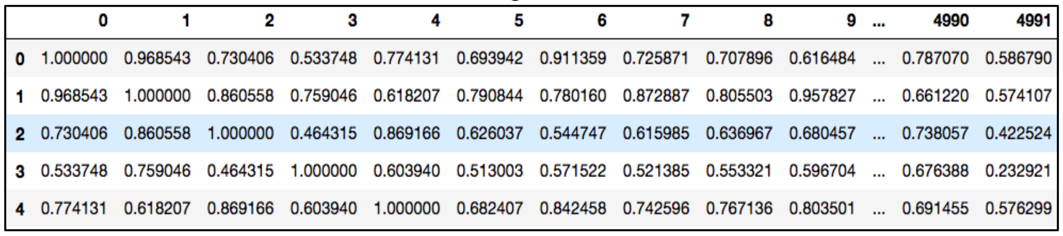
2.2 The purpose of this step is to find the nearest neighbor of picked word. Nearest neighbor means the minimum distance. However, for each word, the distance to itself is minimum (close to zero, in diagonal of matrix). In order to find other nearest neighbor (not the word itself), set the value of diagonal to 1.

Before set diagonal value to 1



	0	1	2	3	4	5	6	7	8	9	...	4990	4991	4992	4993	
0	0.000000	9.685425e-01	0.730406	5.337484e-01	7.741308e-01	0.693942	0.911359	0.725871	0.707896	0.616484	...	0.787070	0.586790	0.700104	0.598163	0.67
1	0.968543	-4.440892e-16	0.860558	7.590464e-01	6.182069e-01	0.790844	0.780160	0.872887	0.805503	0.957827	...	0.661220	0.574107	0.827330	0.696694	0.57
2	0.730406	8.605577e-01	0.000000	4.643148e-01	8.691660e-01	0.626037	0.544747	0.615985	0.636967	0.680457	...	0.738057	0.422524	0.800745	0.626822	0.51
3	0.533748	7.590464e-01	0.464315	1.110223e-16	6.039400e-01	0.513003	0.571522	0.521385	0.553321	0.596704	...	0.676388	0.232921	0.611515	0.443027	0.38
4	0.774131	6.182069e-01	0.869166	6.039400e-01	2.220446e-16	0.682407	0.842458	0.742596	0.767136	0.803501	...	0.691455	0.576299	0.867090	0.593383	0.49

After set diagonal value to 1



	0	1	2	3	4	5	6	7	8	9	...	4990	4991
0	1.000000	0.968543	0.730406	0.533748	0.774131	0.693942	0.911359	0.725871	0.707896	0.616484	...	0.787070	0.586790
1	0.968543	1.000000	0.860558	0.759046	0.618207	0.790844	0.780160	0.872887	0.805503	0.957827	...	0.661220	0.574107
2	0.730406	0.860558	1.000000	0.464315	0.869166	0.626037	0.544747	0.615985	0.636967	0.680457	...	0.738057	0.422524
3	0.533748	0.759046	0.464315	1.000000	0.603940	0.513003	0.571522	0.521385	0.553321	0.596704	...	0.676388	0.232921
4	0.774131	0.618207	0.869166	0.603940	1.000000	0.682407	0.842458	0.742596	0.767136	0.803501	...	0.691455	0.576299

2.3 Pick up 25 meaningful words from V word list

word_list = ['room', 'community', 'america', 'washington', 'english', 'nature', 'equipment', 'summer', 'europe', 'hospital', 'cities', 'leaders', 'sunday', 'communist', 'chicago', 'construction', 'conference', 'food', 'ideas', 'production', 'black', 'issue', 'north', 'friends', 'company']

2.4 Use cosine distance to measure the distance, find the nearest neighbor of each word, finally the nearest neighbor of each word is:

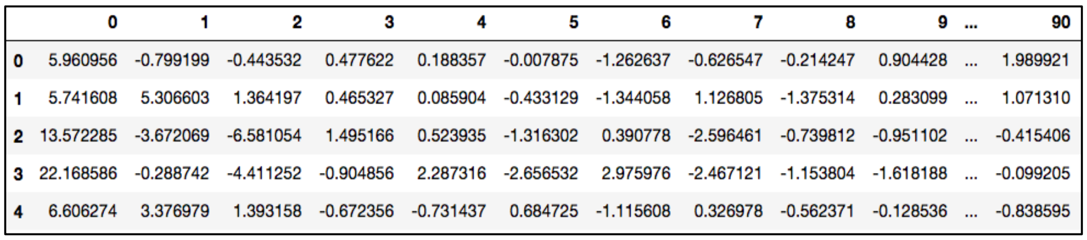
(room ---> door)
(community ---> group)
(america ---> world)
(washington ---> president)
(english ---> great)
(nature ---> human)
(equipment ---> available)
(summer ---> day)
(europe ---> world)

(hospital ---> spent)
(cities ---> towns)
(leaders ---> government)
(sunday ---> evening)
(communist ---> soviet)
(chicago ---> portland)
(construction ---> sales)
(conference ---> meeting)
(food ---> also)
(ideas ---> values)
(production ---> marketing)
(black ---> gray)
(issue ---> power)
(north ---> south)
(friends ---> come)
(company ---> new)

3. Clustering

3.1 After SVD decompose analysis, get the 100 dimension transformed data set (X1). Use k_mean to cluster the target words into 100 clusters. In order to understand the dataset X1, convert it to data frame.

The head of transformed X data frame



	0	1	2	3	4	5	6	7	8	9	...	90
0	5.960956	-0.799199	-0.443532	0.477622	0.188357	-0.007875	-1.262637	-0.626547	-0.214247	0.904428	...	1.989921
1	5.741608	5.306603	1.364197	0.465327	0.085904	-0.433129	-1.344058	1.126805	-1.375314	0.283099	...	1.071310
2	13.572285	-3.672069	-6.581054	1.495166	0.523935	-1.316302	0.390778	-2.596461	-0.739812	-0.951102	...	-0.415406
3	22.168586	-0.288742	-4.411252	-0.904856	2.287316	-2.656532	2.975976	-2.467121	-1.153804	-1.618188	...	-0.099205
4	6.606274	3.376979	1.393158	-0.672356	-0.731437	0.684725	-1.115608	0.326978	-0.562371	-0.128536	...	-0.838595