Aviv shisman, 206558157
Itay Hassid, 209127596

NLP - Exercise 3 - Report results

1)

a. I employed 2 thresholds:

The first is the **minimum occurrences of a word**, in order to insert a word to the dictionary of words we check for co-occurrence, the word need to appear in the corpus more than (or equal to) the minimum threshold we set. In this case, we've been asked to set the threshold to 100.

The second is **minimum occurrences of a feature**, for every word, if its attribute appear less than the minimum thresholds, we don't add it to the word's attribute list/set. In this case, we've chosen to set this threshold to 30.

b. Number of words which passed the minimum threshold (occurrences of a word): 8486

We filtered function words because they harm the result and extend drastically running time.

c.

For the 'by window' approach 25,417,183 features were considered in the calculations(past the min occurrences thresholds)

For the 'by dependency' approach 7,301,116 features were considered in the calculations(past the min occurrences thresholds)

For the 'by sentence' approach 104,021,066 features were considered in the calculations(past the min occurrences thresholds)

All of this make sense because as we expected the by sentence approach which counts all the sentence as features(and in later stages we will use the features threshold to know if to count them or not) is the highest, after that they by window approach because it counts all the words in the window as features and the window has more features the the by dependency approach.

The by dependency approach counts less features because it only takes the few related by the dependency tree.

2) added a file named ‘WordLists - part 1 (Second Order) ‘ with the tables requested

Conclusions:

1. When we take the “by sentence” approach we will often get words that are unrelated, thats happens because we count unrelated words that appear in the sentence, in the PMI Vector for example: “Itay took a sandwich to school” and “Itay took a gun to school” if we will take the “by sentence” approach gun and sandwich have the same features and when we will compare their PMI Vector the result will most likely be positive = **that they are similar** but they are not. (**Cont’ in next page)**

More examples of unrelated words:
‘Car’ and ‘fuselage’ (a plane part) ←-- look at the tables
‘Gun’ and ‘U-boat’(German submarine)

A positive side for this approach is that you most likely won’t miss words that are related to the target
(In other methods you might miss some of them - we discussed about it in lecture 6)

2. When we take the “by dependency” approach we will often get words that are from the same semantic field and we assume they have the same grammar role in the sentence, for example, we got that horse is similar to motorcycle and that’s because they both appeared in the same way → “ride a horse”/ “ride a motorcycle”. This method can also help us understand connections between words, for example: cbs, cbc, nbc, cnn are similar words to fox because they all tv channels so all of them can be written in the same grammatical role → “watch fox/cnn new” or fox/nbc channel”.

3. When we take the “by window” approach we will most of the time get only words that related to the topic, of course it depends on the size of the window (a large window will have the same effect as taking the all sentence) but when we will take small window we will count only the most relevant features(words) that are usually connected to the word.

Cont next page

The downside of this method is that we can lose important features → chance of losing a related words , For example:
-‘Barrel’ is a part of the gun but its not appears in this method words and it is appears in the ‘by sentence’ Approach.
-‘Diesel’ not appears in ‘by window’ words list...

3) added a file named “att - part 1 (First Order)” with the tables requested.

Considering the target word, we would see many co-appearances of the words from the first-order table with the target word.
But we won’t see many co-appearances of words from the second-order table with the target word as the first order one.
Another thing we can say is that the words from the first-order list less similar than the second-order to the target words. The most significant difference can seen in the “by dependency” approach.

4)
We’ve calculated the Average Precision Evaluation for every type for the words **car** and **piano**.
We set N variable in the AP function to be 20. It’s should be the number of all relevant documents (words in our case), but we can’t know how many exist in such tremendous corpus. It’s also possible to set N to be the number of all relevant words found from all the co-occurrences types but then AP would never be 1 because the numerator smaller (or equal) than to 20 and the denominator will be around 35-40 so setting N to 20 seems like the best choice for us.
My decision if a word is similar or not is to check if the words from the list are associated with the target words, so we don’t count only totally unrelated words.

	By sentence	By dependency	By window
car	15.428/20 = 0.7714	11.755/20 = 0.5878	14.4288/20 = 0.7214
piano	18.897/20 = 0.9449	17.624/20 = 0.8812	15.875/20 = 0.7938
MAP	0.8582	0.7345	0.7576

5) a.
For the PMI Vector calculations you may want to see my “CalcWordVecPMI” (Also its very recommended to look at the detailed comments i wrote). In short we calculate the PMI Vec for each word (for later use cosine to compare to the other words) Notice that in the lecture we used log(freq()) instead.
To calculate the PMI i used 3 help dictionaries that help me calculate the pmi value, recall that in the lecture we saw how to calc the propabilities , so these dictionaries serve a similar purpose. Also notice i used the smoother version that uses the magic number ‘0.75’, we learned in word2Vec that the hyper parameters were a big reason for the increase in results, for that reason we used a similar version in the pmi calculations.

b. For this section you may want to look at these functions: “compareVec”, “calcCosine”, ”cosineHelp” i also recommend you read my comments to understand the code better(they are really detailed) .
The first function is the main one which does the compare of the pmi vectors to then output a value of how much the vectors are similar → how much the words are similar.
This function return a dictionary of the cosine values of a specific word with all the other words(this dictionary will be stored in a larger one - see code for more).

For the matrix multiplication (we used dictionaries not an actual matrix) we went over all the features in the PMI Vector and multiplied each one of them in all the other words with the same feature(we have a dictionary for each feature -> all the words that has it)
This is implemented in the “calcCosine” function in the pi*qi line.
We then update the value in the Cosine dictionary of the word, and later the value will be divided by the length of the vectors like in the formula(using the “cosineHelp” function) .

6) in part 2 of the report