December 16, 2022

1 Part 1

1.1 summary statistics

Various thresholds:

- Words that occur at least 100 times.
- Features that occur at least 75 times.

1. **Co-occurrence type 1**:
    b. **Words number**: 22330.
    c. **Features number**: 227704.
    d. **Max non-zeros features**: 227308.
    e. **Min non-zeros features**: 2.
    f. **Avg non-zeros features**: 4164.33.
2. **Co-occurrence type 2**:
    b. **Words number**: 20034.
    c. **Features number**: 48265.
    d. **Features number**: 40548.
    e. **Features number**: 22.
    f. **Features number**: 1375.59.
3. **Co-occurrence type 3**:
    b. **Words number**: 22330.
    c. **Features number**: 102310.
    d. **Features number**: 95834.
    e. **Features number**: 5.
    f. **Features number**: 1144.45.

1.2 top20words

Top 20 most similar words are in the document "top20.xlsx".

in the sentence matrix (co-occurrence1), We can see prominent trend for a topic relation. (E.g. "horse" is close to : "wagon","race","engine". Another example is the word "fox" that the words "oats" and "badger" are related to it). While, in the window matrix (co-occurrence2), We can see prominent trend for a semantic relation. (E.g. "horse" is close to : "beast","animal", etc). In addition, there are words that usually could have come with the same neighbouring words as the target word (E.g. for "horse" it is "dog").

In the positional matrix (co-occurrence3), it is clearly notice that the words are has the syntactic parsing relation and close meaning.(E.g. "rifle" is close to : "machine gun","gun","weapon","canon" , etc), They mostly could be replaced with it's most similar words (The previous trend intensified).

1.3 top20features

Top 20 most similar words are in the document "top20_features.xlsx".

In the co-occurrences matrix 1, there are many appearances of specific type of the word (proper-noun). E.g. for "car" there are "mondeo", super-mini" ,etc. for "hotel" there are "mirkosta", "floridian" , etc. This phenometa is repetative over all the words. Probably because they are the topic of the sentence, as in Wikipedia the name of the topic is very repetitive. In the co-occurrences matrix 2, there are many words that are in the same class as the word and could help decribing it (As in "bus" there are "Eged", "double-decker" ,"conductor", etc.). These strengthen our intuition regarding the results in the previous section. In the co-occurrences matrix 3, we see word that come in close proximity to the target word (E.g. for target word "drink" there are the words: "poisonous", "canned", "alcoholic".

Another note, sadly the closest context word in this Hebrew corpus for "automobile" is "bomb" as in "bomb-car".

1.4 Manual judgments

Manual annotations are in document "manual_annot.xlsx".

Relation	co-occurrence1	co-occurrence2	co-occurrence3
Topic	0.91	0.84	0.51
Semantic	0.57	0.63	0.49

Since the table matrices look on a smaller context (sentence$\longrightarrow$window$\longrightarrow$position). it is make sense that the topic relation is decreasing, In addition, because the topic relation is including the semantic relation, it is always has higher score than the semantic. In the co-occurrence1 it make sense to have high topic relation and low semantic because all the sentence is included. In co-occurrence2 we can notice decrease in topic relation and increase in semantic relation, which

make sense due that the context in narrow down to two words on each side. This gives numeric value to the intuitions from previous sections.

Furthermore, if we look closely we can see in see it the top values for the for example for the Hebrew name positional matrix (co-occurrence3) we notice in the top similar words for "piano". Musical instruments like : "cello" and "violin". Meaning that the syntactic properties were learnt, in the same context we can change the word piano with another musical instrument. Although the MAP value is low since "celo" for example, is not in the semantic class or related to 'piano".

1.5 Implementation description

Brief description of the code implementation:

Q1 The estimation of the PMI values.

A1 Please see Code. 1 for the code implementations. The input to the function is *ccw* - is the co-occurrence word matrix. *wc* - word counts. *fc* - feature counts.
For each of the entry in the ccw, which is an order pair of: $\#_{word,feature} = (word, feature)$. Get the count for that word: $\#_{word} = (word, *)$ and the count for that feature: $\#_{feature} = (*, feature)$. Given the total amount of ordered pairs: $\#_{total} = (*, *)$. Each entry in the "pmi matrix" follows the Eq. 1.

Q2 The efficient algorithm for computing all similarities for a target word.

A2 The algorithm for the efficient calculation follows the scheme in Fig. 1. Meaning each word has list of none zero features and each feature has a list of the words it appear in.

More precisely, in Code. 2. The cosine similarity is consisting from two parts: *nominator* and *denominator*.

The numerator - for each features of the required word, pmi1, The next word that has that feature is fetched from dictionary of feature→words, pmi2. The pmi1 × pmi2 is added to the relevant array cell representing the word.

The denominator - The L_2 norm for each word is calculated - $\sqrt{\sum PMI^2}$. The requested and examined word norms are multiplied.

The cosine similarity between the requested word and each other word is just the nominator divided by the denominator, which is basically the expression: $\frac{\sum PMI_1 \times PMI_2}{\sqrt{\sum PMI_1^2} \times \sqrt{\sum PMI_2^2}}$.

```
def calc_pmi(ccw, wc, fc):
    # sanity checks
    total_freq = sanity_checks(ccw, wc, fc)
    pmi_matrix = defaultdict()
```

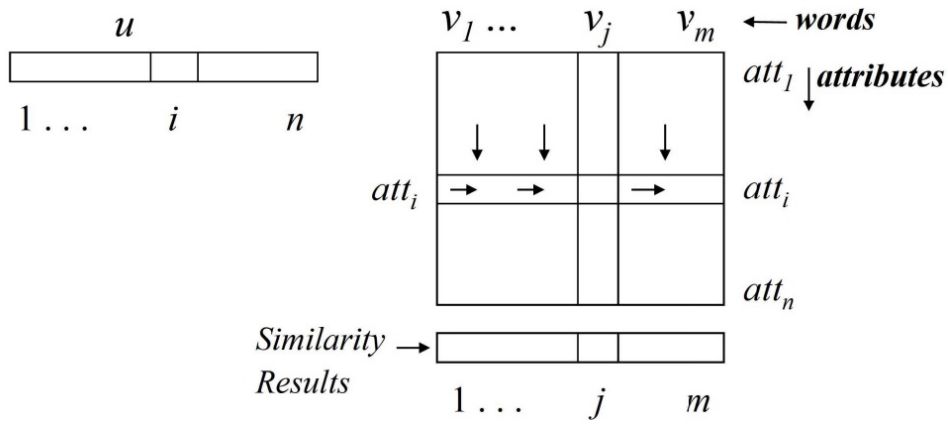

Figure 1: Efficient similarity calculation scheme. Where the attributes are the words specified in each case of the co-occurrences matrix type.

```
    for word_num, features in ccw.items():
        pmi_matrix[word_num] = {}
        freq_word = wc[word_num]

        for feature_num, freq in features.items():
            freq_feature = fc[feature_num]
            pmi_matrix[word_num][feature_num] = np.log2((freq / (
freq_word * freq_feature)) * total_freq)

    return pmi_matrix
```

Code 1: PMI calculation code

$$log_2(\underbrace{\frac{p(word,feature)}{p(word)\times p(features)}}_{\textbf{Class version}}) = log_2(\frac{\frac{\#_{word,feature}}{\#_{total}}}{\frac{\#_{word}}{\#_{total}}\times\frac{\#_{feature}}{\#_{total}}}) = log_2(\underbrace{\frac{\#_{word,feature}\times\#_{total}}{\#_{word}\times\#_{feature}}}_{\textbf{Code 1 version}}) \quad (1)$$

```python
def top_similarities(pmi_matrix, ccf, selected_word_text, words_to_number, top_sim=20):
    def clac_numerator(pmi_matrix, ccf, filtered_number_mapping, selected_word):
        numerator = np.zeros(len(pmi_matrix))
        req_word = pmi_matrix[selected_word]

        for word_feature, pmi1 in req_word.items():
            for other_word, _ in ccf[word_feature].items():
                pmi2 = pmi_matrix[other_word][word_feature]
                numerator[filtered_number_mapping[other_word]] += pmi1 * pmi2

        return numerator

    def clac_denominator(pmi_matrix, filtered_number_mapping):
        denominator = np.zeros(len(pmi_matrix))

        for word_feature_num, features in pmi_matrix.items():
            for feature, pmi in features.items():
                denominator[filtered_number_mapping[word_feature_num]] += pmi ** 2

        denominator = np.sqrt(denominator)

        return denominator

    filtered_number_mapping = {num : i for i, num in enumerate(pmi_matrix)}
    filtered_number_mapping_fliped = {i :num for i, num in enumerate(pmi_matrix)}

    selected_word = words_to_number[selected_word_text]
    denominator = clac_denominator(pmi_matrix, filtered_number_mapping)
    numerator = clac_numerator(pmi_matrix, ccf, filtered_number_mapping, selected_word)

    denominator_selected_word = denominator[filtered_number_mapping[selected_word]]
    for i, _ in enumerate(denominator):
        denominator[i] *= denominator_selected_word

    cos_similarity = numerator / denominator
    cos_similarity[filtered_number_mapping[selected_word]] = 0
    most_similar = np.argsort(cos_similarity)[-top_sim:][::-1]
    most_similar_in_org_index = [filtered_number_mapping_fliped[i] for i in  most_similar]
    return most_similar_in_org_index
```

Code 2: Efficient all word similarity calculation

2 Part 2

2.1 Polysemous Words

Definitions:

- **Group 1**: Polysemous words (words with at least two different meanings) such that the top-10 neighbors of each word reflect **both** word meanings.
- **Group 2**: Polysemous words such that the top-10 neighbors of each word reflect only a **single** meaning.

(*) All dictionary definitions below are taken from "merriam-webster online dictionary" [1]

Q1 Which three polysemous words belong in the Group 1, and what are their neighbors?

A1 (a) **Mouse** - possibilities:

i. **Definition:** Any of numerous small rodents.
Neighbors: `mice`.

ii. **Definition:** A small mobile manual device that controls movement of the cursor.
Neighbors: `cordless_laser`, `keyboard_arrow_keys`.

(b) **Nickel** - possibilities:

i. **Definition:** To plate with nickel.
Neighbors: `nickeling`, textttnickled.

ii. **Definition:** The U.S. 5-cent piece regularly containing 25 percent nickel and 75 percent copper.
Neighbors: `dime`.

(c) **Earth** - possibilities:

i. **Definition:** The fragmental material composing part of the surface of the globe.
Neighbors: `Martian_surface`.

ii. **Definition:** The planet on which we live that is third in order from the sun.
Neighbors: `planet`, `cosmos`.

Q2 Which three polysemous words belong in the Group 2, what are the possible senses for each word, and which of the senses was reflected in the top-10 neighbors?

A2 (a) **Light** - possibilities:

i. **Definition:** The sensation aroused by stimulation of the visual receptors.

Neighbors: `yellowish_glow`, `illumination`, `sensitive_photoreceptor_cells`, `illuminate`.

ii. **Definition:** To set and leave temporarily.

(b) **Park** - possibilities:

i. **Definition:** An enclosed piece of ground stocked with game and held by royal prescription or grant.

Neighbors: `skate_park`, `campground`, `skateboard_park`.

ii. **Definition:** To bring (a vehicle) to a stop and keep standing at the edge of a public way.

(c) **Ceiling** - possibilities:

i. **Definition:** The overhead inside lining of a room.

Neighbors: `acoustical_tile`, `domed_ceiling`, `vaulted_ceiling`, `ceiling_tiles`, `decorative_lightings`.

ii. **Definition:** An upper usually prescribed limit.

Q3 Can you explain why the Group 2 words neighbors reflect only one sense?

A3 A possible explanation is that for the second word sense, there is greater variation of the neighbouring words. Meaning, they can fit to a multiple contexts so to get many occurrences of specific neighboring wordform with the same query word is not common.

2.2 Synonyms and Antonyms

Find a triplet of words (w1, w2, w3) such that all of the following conditions hold:

1. w1 and w2 are synonyms or almost synonyms.
2. w1 and w3 are antonyms.
3. sim(w1,w2) ¡ sim(w1, w3)

Q1 Write your triplet.

A1 w1="warm", w2="warmth", w3="chilly".

Q2 Can you explain this behavior in which the antonyms are more similar than the synonyms?

A2 Because the embeddings are taking by context, synonyms and antonyms exists in a similar context, therefore they are close close to each other. E.g. in the sentence "Today the weather is ___" , the word: "Warm" and it's antonym "Chilly" is equally possible. Furthermore, the synonym of "Warm" - "Warmth" is not appropriate in this context.

2.3 Dimensionality Reduction

Question:

Take the first 5000 words in the vocabulary. From these, keep only words that end either with "ed" or with "ing" (you should be left with 708 words). From the resulting list of 708 words, create a matrix with 708 rows, where each row is a 300-dim vector for one word, and reduce the dimensionality of the matrix to 2-d (the result is a matrix with 708 rows and 2 columns).

Plot the resulting 2-d vectors (you can use the scatter plot in the matplotlib package) where each vector/word is a dot in a scatter plot, according to its coordinates. Color points that correspond to words that end with "ed" in blue, and points that correspond to words that end with "ing" in green. Submit the resulting plot.

Also submit a discussion of it (are the colors separated? why or why not? is the information included in the model? etc).

Answer:

As seen in Fig.2, there is a *rough* desperation between words ending with "ing" and words ending with "ed". By *rough* I mean: there are words that seems to be words that ending with "ed" which are close to words that ending with "ing" (E.g. in the area around (0,0) in figure).

Since the variance explained by the first component is $\sim 4.2\%$ and the variance explained by the second component is $\sim 3.4\%$ than this figure showing only about $\sim 7.6\%$ of the variance! Therefore, it is hard to give better than "rough" estimation.

Note, if for example it would be needed for a downstream task t build a classifier that recognized words that ending with "ed" from words that ending with "ing". It would be possible to project given word (ending with "ing"/"ed") to 2-dimensions and to check to witch cluster mean it is closer.

References

[1] *merriam-webster online dictionary*. URL: `https://www.merriam-webster.com/dictionary`.

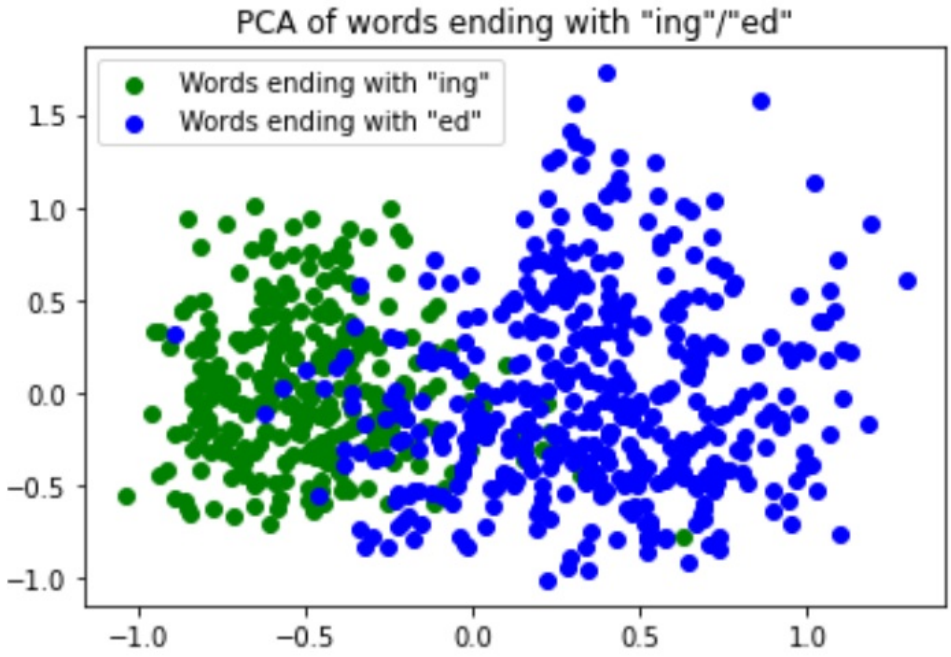

Figure 2: PCA of words ending with "ing"/"ed"