Assignment 2

Yumin cao, a1754926

Reading list:
1– Basic analysis of building TF–IDF
2– Basic analysis of using glove
3– Basic analysis of using SVD
4– Building average sentence embedding
5– Building cosine similarity
6– Result and analysis
7– Compare TF–IDF, average sentence embedding and SIF embedding

1. Basic analysis of building TF–IDF

The main idea of TF–IDF is that if a word occurs many times in a sentence, while this word doesn't often occur in other sentences, then this word should be special and important to that sentence.

There are some ways to calculate term frequency(TF) and inverse documents.

Term frequency [edit]

Term frequency, $\mathrm{tf}(t,d)$, is the frequency of term t,

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

where $f_{t,d}$ is the *raw count* of a term in a document, i.e., the number of times that term t occurs in document d. There are various other ways to define term frequency:[5]:128

- the raw count itself: $\mathrm{tf}(t,d) = f_{t,d}$
- Boolean "frequencies": $\mathrm{tf}(t,d) = 1$ if t occurs in d and 0 otherwise;
- term frequency adjusted for document length: $\mathrm{tf}(t,d) = f_{t,d} \div (\text{number of words in d})$
- logarithmically scaled frequency: $\mathrm{tf}(t,d) = \log\,(1 + f_{t,d})$;[6]
- augmented frequency, to prevent a bias towards longer documents, e.g. raw frequency divided by the raw frequency of the most occurring term in the document:

$$\mathrm{tf}(t,d) = 0.5 + 0.5 \cdot \frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$

What we were taught on the class is using the **log(count(w, d))** as term frequency, although this TF method using log to smooth the value, for one document, **it is still better to use scale**. Because if a question is long, for example its' count(word)=5 when the total sequence words = 30, would this be more important than another word/total = 0.2 while total sequence words = 10? Not. So **using the scale of TF** may be a more good choice.

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}},$$

$$\mathrm{idf}(t, D) = \log \frac{N}{|\{d \in D : t \in d\}|}$$

Besides, we were taught on the class to use **N/df** as IDF, however, if a words, like 'Algorithms' only appear once, then its IDF would be too large, when the IDF of word like 'how' would be too small, so, **it is better use log to smooth the value**. After building the TF–IDF table, we inverse it from the doc–word to word–doc structure.

Word	(Documents, TF-IDF)
Inverted	(doc1, 0.04), (doc4,0.02) ...
File	(doc1,0.02) ...
NLP	(doc2, 0.06)
Semantic	(doc3, 0.05), (doc5, 0.05)...
...	...

One thing that is important when using TF–IDF is the **computing time,** I used TF–IDF in assignment 1 but failed with the computing time. This time I learnt the inverse TF–IDF and realised we could only form the word table we need. For example, if a query is 'he is a man', then we could only search these four words instead of forming a total complete table.

And in TF–IDF we need to do the stemming, and remove stopwords to make the result robust, and the stopwords would not affect our calculating while the stemming would help reduce dimension.

2. Glove

Glove is a pre trained word embedding using at least three sets to generate word embedding.

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

After analysing its function, I found that we need to add **<s> and </s>** for each sentence, because '<s> how' and 'how' may have different embedding.

Besides, Glove can handle words like 'better' and 'best', so it is not a good choice to do **stemming** before using glove.

In this assignment, numbers, punctuation, capital letter, html may be something not that important, so I remove them before using glove. And I used twitter.embedding.20d as my file, because it is easy for CPU.

What's more, glove records some most frequency words in real world. We have counted that there are more than 70'000 words in the corpus, but some are wrong words while others may be obscure, so I only keep the first 30'000 words as our **vocabulary size.**

3. SVD

We have do the SVD in other class, however in this assignment, we may occur problem with **RAM memory explosion** because the matrix is very big. However, For the U of SVD, only several of the first columns important, so we could use metrics similar to the target metrics using parameter 'full matrices = False' in np.linalg.svd function.

4. Building average sentence embedding

After using Glove, we could get all the word embedding in corpus, then we could calculate the sentence embedding:

$$Sentence = \frac{1}{n} \sum_{word\ in\ sent}^{n} word$$

This is not hard, however, when embedding our query, we may occur a problem: **Some words not in our embedding corpus**, in this case, my solution is that: since we don't know this word, then it shouldn't have any effect for us to understand the meaning(representation) of the sentence, then we should ignore it. For example, there are 6 words in a sentence, while one word is unknown, then the n should be 5 and we just add those 5 words' embedding.

However, this method has drawbacks, **if a word occur multiple times in a sentence, then the sentence representation would be dominated by that word.** So we need down weight those words. And SIF embedding method makes it up.

5. Building cosine similarity

$$\text{cosine}(\vec{v}, \vec{w}) = \frac{\vec{v} \cdot \vec{w}}{|\vec{v}||\vec{w}|} = \frac{\sum_{i=1}^{N} v_i w_i}{\sqrt{\sum_{i=1}^{N} v_i^2} \sqrt{\sum_{i=1}^{N} w_i^2}}$$

After getting two sentence embedding, we could use cosine similarity to calculate the **cosine angle** to calculate their similarity, where the numerator is the inner product of two sentence embedding.

6. Result

Accuracy	TF-IDF	Average embedding	SIF
Top2	23%	20%	34%
Top5	33%	30%	51%

From the result we can see that TF–IDF has similar result as average embedding, while SIF embedding performs better.

7. Compare TF–IDF, average sentence embedding and SIF embedding
SIF embedding:

Algorithm 1 Sentence Embedding

Input: Word embeddings $\{v_w : w \in \mathcal{V}\}$, a set of sentences $\mathcal{S}$, parameter a and estimated probabilities $\{p(w) : w \in \mathcal{V}\}$ of the words.
Output: Sentence embeddings $\{v_s : s \in \mathcal{S}\}$
1: **for all** sentence s in $\mathcal{S}$ **do**
2: $v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a+p(w)} v_w$
3: **end for**
4: Form a matrix X whose columns are $\{v_s : s \in \mathcal{S}\}$, and let u be its first singular vector
5: **for all** sentence s in $\mathcal{S}$ **do**
6: $v_s \leftarrow v_s - uu^\top v_s$
7: **end for**

This method is simple but wonderful, we could see it calculate the p(w) as word frequency, the weight of a word would becomes small as the p(w) get big, this can be seen as the **down weight method,** if we treat the average sentence embedding as **uniform average**, then this SIF method **considers the weight of each word**. However, if we just use simple sum of word embedding as sentence embedding, the information in words may be **redundant**, so it use the SVD to get the **main component** of sentence vector. For the parameter a, we could find in the article Sanjeev Arora, Yingyu Liang, Tengyu Ma, A SIMPLE BUT TOUGH–TO–BEAT BASELINE FOR SENTENCE EMBEDDINGS, ICLR 2017. **3x(10^{-3} – 10^{-4})** would be a good range for a.

Through the above analysis we could find that down weight method is important for the sentence method, as some high frequency–less important words may mislead our sentence embedding. Although the TF–IDF considers the weight, but it doesn't consider the order as well as the sentence semantics, which makes it performs less good.