Text Technologies – Assignment 2 – Report s0837795

I. Word overlap retrieval algorithm

The world overlap algorithm is a very basic retrieval algorithm. The algorithm simply returns an integer which represents the intersection of the set of words in the document and the set of words in the query. For example, if the query is "cat dog horse" and the document is "cat horse tram carriage", it would return 2.

The average non-interpolated precision of this algorithm for our test cases is: 0.1871

II. Tf.Idf weighted sum retrieval algorithm

The Tf.Idf weighted sum is a more sophisticated ranking formula. The similarity of two documents (or a document and a query) is computed by the sum:

$$\sum_{w} tfw,Q \cdot \frac{tfw,D}{tfw,Q + \frac{k|D|}{avg|D|}} \cdot \log \frac{|C|}{dfw}$$

Where k = 2, and tfw,X is the term frequency of the word w in document X.

This algorithm has an average non-interpolated precision over our test cases of: 0.3388

III. IvanRank-13 algorithm

For this algorithm, I decided to stick with the Tf.Idf weighted sum formula, as it has shown excellent performance in the previous task, considering the fact that the document data is quite polluted with spelling errors and words in a wide range of languages. In order to clean up the data, I tried a number of approaches and combined the most successful in the final IvanRank-13 algorithm.

First, I decided to clean up the documents in the corpus by implementing a Bayesian spelling correction algorithm, as described by Peter Norvig in his (appropriately titled) essay "How to Write a Spelling Corrector" [1]. For this task I used a large English language corpus and looked to calculate statistically probable spelling corrections, two edits away from each word that did not exist in the natural language corpus. After two and a half hours of processing time, the combination of a spell corrector and Tf.Idf resulted in average non-interpolated precision of 0.5474- an improvement of over 0.20 over plain Tf.Idf.

Due to the substantial processing time of the Bayesian spelling correction algorithm, I had the motivation (and the free time) to implement the SoundEx phonetic algorithm, as described in Lab 2 of this course [2]. This combination resulted in average non-interpolated precision of 0.5869 – an improvement over the spelling correction algorithm. In addition to that, this preprocessing step took roughly 10 seconds, which is a massive improvement in efficiency.

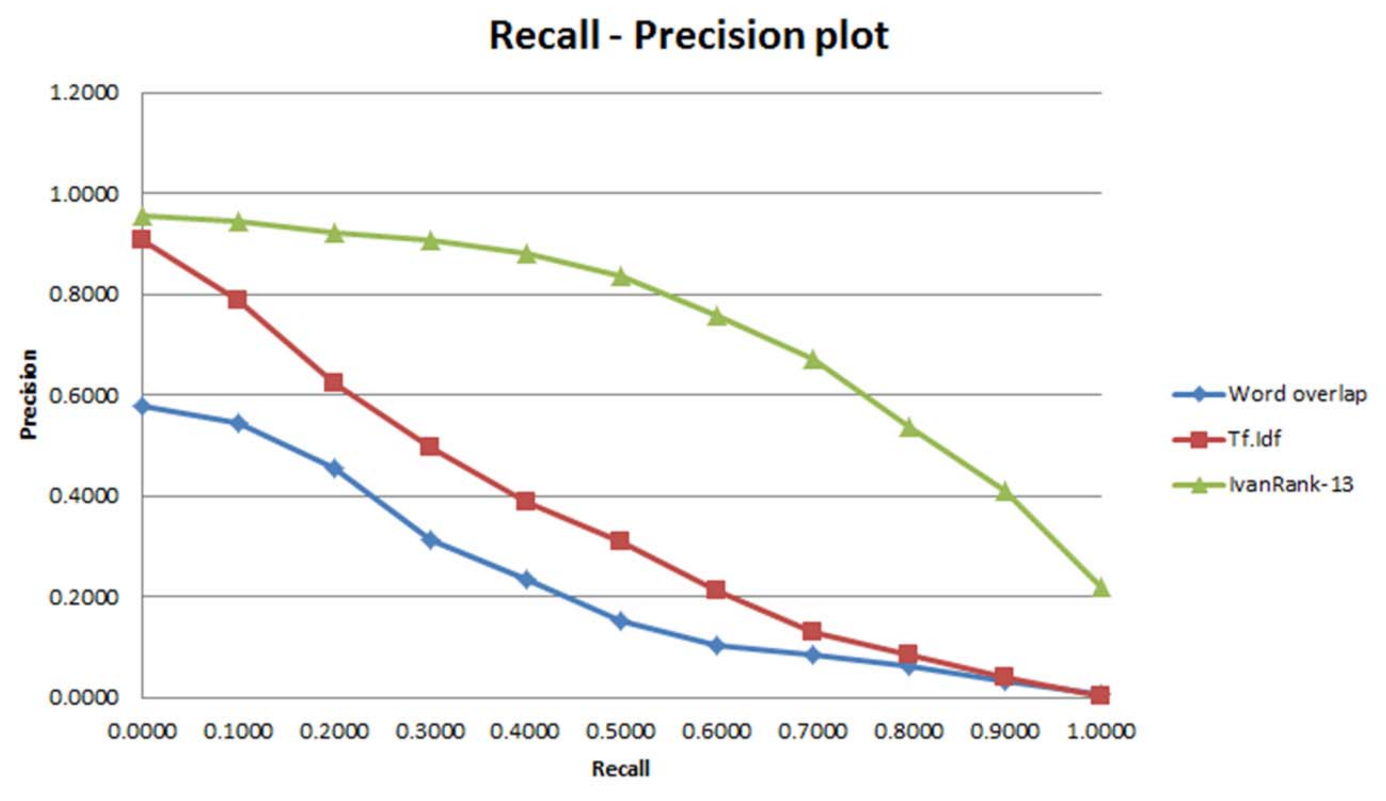
I attempted to improve on the SoundEx + Tf.Idf combination by applying a stemming algorithm, as described in the paper "An algorithm for suffix stripping" by Martin Porter [3]. On the current dataset, this resulted in only a minor improvement from 0.5869 to 0.5892. The result could be easily explained by the fact that our corpus of documents in polluted with hundreds of words in non-English languages, for which stemming would be counter ineffective.

Because of that, I decided to clean up the documents in the corpus by translating foreign language words into English. I attempted to translate unknown words by using the Python library for Google Translate [4], however due to restrictions placed by Google in the Google Translate API, I could only make a small number of calls within a certain time period, insufficient to translate all foreign words in the document. In addition to that, I had to place substantial delays between each request. Because of these restrictions, I decided to abandon this approach and compromise. I wrote a function that contains a dictionary of the 100 most frequent words in the corpus and their translation into English. This hackTranslate() function helped improve the average precision from 0.5869 to 0.7353.

Overall, the final algorithm uses combines the Tf.Idf weighted sum with SoundEx and translation capabilities. With an average non-interpolated precision of 0.7353, it is a substantial improvement to plain Tf.Idf.

References:

1. Peter Norvig, How to Write a Spelling Corrector, URL: http://norvig.com/spell-correct.html
2. Philipp Petrenz, TTS: LAB 2, URL: http://www.inf.ed.ac.uk/teaching/courses/tts/labs/lab2.html
3. Martin Porter, An algorithm for suffix stripping, Program, Vol. 14, no. 3, pp 130-137
4. Peteris Krumins, Python Library for Google Translate, http://www.catonmat.net/blog/python-library-for-google-translate/



Recall	Word overlap	Tf.Idf	IvanRank-13
0.0000	0.5768	0.9071	0.9540
0.1000	0.5466	0.7892	0.9448
0.2000	0.4564	0.6240	0.9203
0.3000	0.3147	0.4960	0.9078
0.4000	0.2364	0.3860	0.8813
0.5000	0.1522	0.3091	0.8348
0.6000	0.1055	0.2122	0.7571
0.7000	0.0851	0.1302	0.6735
0.8000	0.0628	0.0840	0.5367
0.9000	0.0321	0.0389	0.4107
1.0000	0.0058	0.0042	0.2214