CS 4740 Project 4

Gi Yoon Han, Rui Liang, Xiaoyun Quan, Saerom Choi

12/2/2016

Kaggle team name: random_name

1 Overview

Our goal in this project is to incorporate word embeddings into project 2 where we tried to tag uncertain words/sentences in the test text data given some training data. In project 2, we compared a few potentially powerful methods in parallel and found the best training model is CRF with multiple features including:

- word itself with all letters lowercased;
- POS tags that are provided in the data;
- first letter capitalized (rest lower case);
- 7 consecutive words in a sequence with the current word in the middle of its neighbour words.

However, the CRF model does encounter sparsity issues due to 1) unbalanced data where we have much less words with uncertainty tags than those without; and 2) none of the features can neutralize this unbalance. Therefore, we added one more feature (via word embeddings) to the CRF model so that we can alleviate this issue to some extent.

2 Model description

We first repeat the data preprocessing as we did in Project 2. We then sepcify the features in CRF model to be the ones as described above plus "the nearest neighbour". The new feature "the nearest neighbour" (TNN) is a binary feature where we consider the nearest neighbour to the word. We start with defining the list of uncertain words: find all uncertain words with a B-CUE or I-CUE tag in our training data, and use `word2vec` to find the nearest neighbour to each of these words, then add all these neighbours to the list of uncertain words. Note that we used BROWN corpus to train `word2vec`, with setting dimensions of vector space to be 100 and the minimum count of words to be 5 (such that all words with count less than 5 will be ignored). For example, if the word is "probably", we start with finding the nearest neighbour to "might" (which is found to be "nor"), where the distances are obtained from `word2vec` (documentation can be found at `https://code.google.com/archive/p/word2vec/`) based on pre-trained vectors trained on part of Brown corpus (as default in `word2vec`); and we update the list of uncertain words by adding that neighouring word to the list. When we encounter the word "nor" in test data, the TNN feature will be set to be 1, which will then be passed to CRF model.

This alleiviates the sparsity to some extent by enriching (doubling, in fact) the list of uncertain words (so that the data is less unbalanced). A snapshot of the expanded list is here (the first column is the uncertain words from training data, the second column is the closest word to it, hence added to the list of uncertain words):

may	must
some	any
people	things
generally	frequently
should	must
many	these

evidence true
thought felt
claims estimates
several three
concretely Santayana

We can see that although some of them are not making sense (e.g. several and three, concretely and Santayana), the rest are not bad: “felt” is added to be uncertain since it’s close to “thought”; “estimates” is added as it’s close to “claims”. Undoutedly “felt” and “estimates” are words with uncertainty, so the results from word embedding is quite reliable.

3 Basic observations

Basic observations to help illustrate the concept of word embeddings using `word2vec`: For example, if we look at the frequently uncertain word “probably”, its 5 nearest neighbours and the corresponding distances to “likely” are:

(‘expected’, 1.784043732343727)
(‘necessary’, 2.159495559122513)
(‘needed’, 2.2028901102493608)
(‘difficult’, 2.2232477543709184)
(‘sure’, 2.320627366889919)

According to `word2vec` trained on data from BROWN corpus, the five words above are closest to “likely” in the vector space. It makes sense that “likely” has similar meanings or will appear in similar context to the five words. What we will do then is to add “expected” to our list of uncertain words. And if we would encouter the word “expected” in the test data, we will pass its TNN feature as 1 to the CRF model.

However, if we look at another frequently uncertain word “maybe”, its 5 nearest neighbours and the corresponding distances to “maybe” are:

(“he’s”, 0.9250226844020926)
(‘worry’, 1.0099421610387924)
(‘careful’, 1.0185852577824932)
(‘wonderful’, 1.0445925988533)
(‘yes’, 1.0511055575976425)

It’s interesting that the closest word to “maybe” is “he’s”, which makes sense that most often we will say “he’s maybe xxxxx....”. But it does not help too much in our problem that “he’s” is not a word with any uncertainty. So although the word embeddings alleviate the sparsity issue, it’s still not flawless.

Another interesting example is considering the word “bank”. The 5 nearest neighbours to “bank”, according to `word2vec`, are:

(‘Mississippi’, 0.6004403350772677)
(‘row’, 0.6392226432068754)
(‘sky’, 0.6401606361300888)
(‘edges’, 0.6512596226050484)
(‘headquarters’, 0.6562710660532266)]

It is seen that the vector space we obtained from `word2vec` is highly dependent on the training dataset. Here we use BROWN corpus to train, but if we use some other database such as GOOGLE-NEWS, we will obtain very different closest neighbours to the words such as “bank” (e.g. financial words, rather than “Mississippi”, might be mapped as the closest to “bank”). So it implies that if the training corpus is only limited to a certain topic (or the corpus is of small size), then the vector space will be biased: the closest words might be relevant to the word under such topic.

4 Experiments and results

To find the best possible model, we conducted 2 experiments with different number of closest words, denoted as k, where $k = 1, 2$. The data are divided into two parts as we did in project 2: first 1067 (90%) of the training text files in `train` folder as training data, the rest 117 (10%) of the training text files as evaluating data.

k	Experiment	F-score
N/A	the previous CRF model in project 2	0.57
1	previous CRF model with extra feature TNN (1 closest word)	0.60
2	previous CRF model with extra feature TNN (2 closest words)	0.58

By running these experiments, we are able to compare the performances with different number of closest words to include in the list of uncertain words. We stopped at $k = 2$ because we saw the performance is getting worse as we add more closest words to the list, which may be due to overfitting. In short, we find it's optimal to include the single word closest to the uncertain words in the training data, where we have F-score as high as 0.60. However, it can be seen that the $k = 2$ case also outperforms our previous model, although it's not as good as $k = 1$ possibly due to overfitting.

5 Sumamry

In conclusion, we conducted two experiments and found the model with $k = 1$ is the best and it improves the performance as F-score for the evaluating data increases to 0.60. However, the predictions obtained from this model for the testing data saw a tiny drop from 0.35721 to 0.32368 after we submitted onto Kaggle. We suspect that this might be due to overfitting where we probably added too many irrelevant (that has nothing to do with indication of uncertainty) to our list of uncertain words. But we do see improvement in our evaluating data.

We have been enjoying working within our team through the entire course of this project (and of course all previous projects), and the workflow has been equally broken down: Saerom and Giyoon implemented `word2vec` part, Rui incorporated the output from `word2vec` into the model, and Xiaoyun analyzed the results and wrote up the report.

6 Instruction on running the code

1. Decompress the code.zip
2. Download Brown corpus at: https://drive.google.com/file/d/0Bw-jfSph7WfSTk4xWmwtMEh5X3c/view
3. Put Brown corpus file in the same folder as word_embedding.py
4. Run word_embedding.py
5. Run crf.py