Introduction

The distributional semantics hypothesis states that words that occur in similar contexts share a similar meaning. Our chosen method relies heavily on this assumption. At a high-level, our algorithm learns vector representations of words, and when presented with a question and its comments, transforms both the question and the comments into this vector representation, and ranks the comments according to their similarity to the question. We used two separate models to generate the vector representations: latent semantic indexing (LSI), and word embeddings (WE). We used the cosine similarity measure to compare vector representations of each question and its comments. We also tried combining model predictions in the hope that each model may learn something slightly different, and the combined output would offer an improvement over any of the single models. Our model also includes some hand-crafted features that were derived by inspecting the training datasets.

Method

Before constructing each of the models, the questions and related comments were scraped from the training sets, with each question and its related comments treated as a single document. A term-document matrix was created in order the construct the LSI model. The matrix was constructed using the term frequency–inverse document frequency (tf-idf) statistic. Word embeddings were generated using the word2vec software, which is based on the methods described in [1]. Both embedding learning methods mentioned in [1] (CBOW and skip-gram) were tried. Pre-generated word embeddings were also used for comparison: one set created by the authors of [1] that was trained on a large corpus of google news articles, and a second set trained on tweets [2]. We hypothesised that these pre-trained embeddings would offer superior performance on account of their being trained on much larger corpuses. We also presumed the second set of pre-trained embeddings ([2]) would offer superior performance to the first set, on account of their being trained on more colloquial language (tweets) than the first, which would be more similar to the examples our system is likely to encounter. Combined model ranking scores were combined according to:

$$S = \alpha s_1 + (1 - \alpha) s_2 \qquad (1),$$

where S represents the combined score, s_1 and s_2 represent the scores output by the individual models, and α is a weighting factor.

Model hyper-parameters, such as the reduced dimensionality of the term-topic space in LSI, and the dimensionality of the word embeddings, were determined empirically using a separate, unseen set of questions and comments (the development set). Once final LSI and word embedding models were constructed, they were combined, and their relative contributions to the combined model score (represented by the parameter α) were fine-tuned using the development set. Specifically, configurations with which a higher MAP (mean average precision) score was achieved on the development set (when evaluated by the “ev.py” script) were favoured over those configurations for which a lower MAP score was achieved.

In addition to using these unsupervised feature learning methods, we manually inspected the training sets and hand-crafted some features.

Results

Model comparison

The optimum word embedding dimensionality was determined to be 200. Learning the embeddings with the 'skip-gram' training scheme, as opposed to the 'CBOW' scheme produced superior embeddings. The embeddings trained on our training sets outperformed both sets of pre-trained embeddings, and the twitter embeddings outperformed the google news embeddings. The optimal number of LSI dimensions was 170. The word embedding model outperformed the LSI model. The first column of Table 1 shows the MAP scores obtained using the two models. Combining the models didn't offer an improvement over the word embedding model, but did offer an improvement over the LSI model, because the word embedding model outperforms the LSI model, and mixing the LSI rank scores with the word embedding rank scores produced scores that were more similar to the word embedding scores. Therefore the optimal α parameter in Equation 1 was determined to be 0.

Hand-crafted features

It was noticed that users often replied to their own questions, and that when they did so, their comments were disproportionately rated as "PotentiallyUseful" or "Bad" (hereinafter referred to as potentially useful, and bad, respectively) - explicitly, comments made by users other than the user that posed the question were rated as "Good" (hereinafter referred to as good) three times as frequently as comments made by the user that asked the question. The correlation between comment length and answer quality was investigated, and it was found that comments rated as good were generally longer than comments rated as either potentially useful or bad, and potentially useful comments were also longer on average than bad comments. We incorporated the first fact into our algorithm by automatically setting the ranking score of any comments written by the user that asked a given question to 0; this led to a MAP-score improvement. The second point was incorporated by including a comment length term in the score function as follows:

$$S = \beta s_{sim} + \frac{(1-\beta)L}{Lmax} \qquad (2),$$

where s_{sim} is the similarity score, L is the comment length, $Lmax$ is the maximum comment length of all the comments to be evaluated (across all of the questions), and β is a weighting factor. This feature also improved the MAP-score. In addition, applying the comment length adjustment after the asker-comment penalty, instead of the other way around, led to a small MAP-score improvement. This reflects the occurrences of questions in which the asker posts more than one comment, one or more of which is good and is longer in length than their other comments, which aren't labelled as good. These results are displayed in Table 1. The optimal value of the β parameter in Equation 2 was determined to be 0.045.

Model	Standard	ACP	ACP+CLAI	ACP+CLAII
LSI	0.5251	0.5706	0.5738	0.5755
Word Embeddings	0.5602	0.5960	0.6031	0.6042

Table 1: Summary of MAP scores achieved with different models/configurations on the development set. Standard refers to a model with no additional features, ACP refers to the asker-comment penalty, ACP+CLAI refers to the combination of the asker-comment penalty and the comment-length adjustment, with the comment-length adjustment applied before the asker-comment penalty, and CLAII refers to same thing, but with the comment-length adjustment applied after the asker-comment penalty.

Discussion

The relative performance of the two models can be attributed to their complexity. The LSI model is very basic, and relies solely on word co-occurrence across documents, whereas the heavy lifting in the embeddings model is done by a neural network, which is much more complex, and can therefore represent more complex hypotheses. The performance difference between the two sets of pre-trained embeddings could be due to the more colloquial style of the twitter training corpus, which is more similar to the style used in our datasets (than the style of the google news corpus). The performance difference between our embeddings and the pre-trained embeddings may be due to terms that appear in our training and test sets but do not feature in the pre-trained embeddings. When this occurred, we simply ignored the missing word's contribution to the overall comment vector (additive combination of each word in the comment). In addition, the contextual meaning of some words in our training set may be diluted in the pre-trained embeddings, on account of the shear size of data the pre-trained embeddings were trained on. An interesting extension would be to try to fine-tune the twitter embeddings on our training set, and see if this leads to improved performance.

References

[1] - T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient Estimation of Word Representations in Vector Space, arXiv:1301.3781.
[2] - Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation