Text-Based Information Retrieval

Semantic Embeddings in Information Retrieval
Programming Assignment 2014-2015

Kallirroi Dogani

April 26, 2015

1 Solving Analogies using Semantic Embeddings

1.1 Important Remarks

We implemented three analogy models, all described in [1].

1. *3CosAdd*
   This is the basic model proposed in [2]. Having the analogy

   $$\mathrm{a : b = c : ?}$$

   we are looking for the word with vector closest to the vector $c - a + b$, which is translated as:

   $$\arg\max_{d \in V}(\cos(d, c - a + b))$$

2. *PairDirection*
   This model was also used in [2]. It is similar with the 3CossAdd model, but it conserves the direction of transformation.

   $$\arg\max_{d \in V}(\cos(d - c, b - a))$$

3. *3CosMul*
   This model was proposed in [1].

   $$\arg\max_{d \in V} \frac{\cos(d,c)\cos(d,b)}{\cos(d,a)+\epsilon}$$

   where $\epsilon = 0.001$ is used to prevent division by zero. This model also requires all similarities be non-negative, so, as the authors did, we transformed cosine similarities to [0,1] using $(x + 1)/2$ before calculating.

1.2 Implementation Details

The programming language we used is Java. You will find a README file in the main folder with instructions how to run the program using different parameters. Below we describe shortly the classes we used:

- `AnalogySolver.java`: Contains the `main` method, handles the parameters, reads the vectors, and executes the selected analogy model.
- `WordEmbedding.java`: Represents a specific word and its vector representation.
- `RepresentationModel.java`: Holds all the word embeddings in an array list and also maps each word to its vector.

- `CosineSimilarity.java`: Contains a static method for computine the cosine similarity between two vectors.
- `AnalogyModel.java`: An interface for the analogy models.
- `CosAdd.java`: The *3CosAdd* analogy model.
- `PairDirection.java`: The *PairDirection* analogy model.
- `CosMul.java`: The *3CosMul* analogy model.
- `AnalogyModelEvaluation.java`: Computes the recall.

More details about the implementation you can find in the comments of the code.

1.3 Experiments and Results

Table 1 shows the recall values of the *3CosAdd* analogy model for each semantic category, but also the average of all categories[1]. Bold values indicate the highest recall value for each semantic category. We observe that increasing the dimensionality of the vectors, we have better results. Higher dimensions contain more information for each word. The highest values are for $d = 300$ and mostly for Word2Vec representation model. One possible reason is that in this dataset there are 3000000 words, whereas in the GloVe dataset we have 400000 words.

Having the recall values for each category is very important, because we can see that for some categories recall is very high (e.g. *capital-common-countries*), but for others is quiet low (e.g. currency). So, the average recall (e.g. 0.466 for $d = 50$) is not representative for all the categories.

Tables 2 and 3 shows the recall values for the models *PairDirection* and *3CosMul* respectively. We observe that the recall values of *3CosMul* are close to those of *3CosAdd*, so we cannot say that one of them is much better than the other. However, the performance of *PairDirection* model is clearly lower than those of the other two models. Finally, we see again that increasing the dimension results to higher recall values.

For all the analogy models, one common error was that the predicted word was one of the three words in the analogy. For example, the predicted word for the analogy *write : writes = work : ?* was the word *writes*. This is an error we can avoid. In case the first word (*write*) is different than the third word (*work*), then the predicted word should be also different from the second word (*writes*). Making these checks, we increased the recall more than 10%. For example, for $d = 50$ in the *3CosAdd* model recall was increased from 0.3166 to 0.4644.

Table 4 shows other typicall errors. We observe that the predicted word, even if it is wrong, it is not totally irrelevant with the correct word. For example, in the first analogy the predicted word is *Germany*, whereas the correct word

[1]You can find all these outputs for the three models in the folder `/Results`. We deliver these files, because they have long execution times, so it would not be easy for you to run them all again.

	GloVe				Word2Vec
Category	50d	100d	200d	300d	300d
capital-common-countries	0.7984	0.9407	0.9525	**0.9545**	0.8320
capital-world	0.6876	0.6542	0.9420	**0.9597**	0.7913
currency	0.0866	0.1454	0.1662	0.1616	**0.3510**
family	0.7015	0.8003	0.8359	**0.8774**	0.8478
city-in-state	0.1584	0.3019	0.5070	0.6043	**0.7089**
gram1-adjective-to-adverb	0.1360	0.2358	0.25	0.2247	**0.2893**
gram2-opposite	0.0923	0.2019	0.2364	0.2684	**0.4298**
gram3-comparative	0.5375	0.7905	0.8566	0.8723	**0.9099**
gram4-superlative	0.2834	0.5490	0.6755	0.7076	0.8770
gram5-present-participle	0.4166	0.6884	0.6751	0.6950	**0.7850**
gram6-nationality-adjective	0.8630	0.8855	0.9243	**0.9255**	0.8993
gram7-past-tense	0.3596	0.5339	0.5858	0.5961	**0.6692**
gram8-plural	0.5900	0.7162	0.7717	0.7785	**0.9031**
gram9-plural-verbs	0.1793	0.5839	0.5954	0.5850	**0.6816**
Total	0.4644	0.6270	0.6934	0.7156	**0.7379**

Table 1: Recall values for the analogy model *3CosAdd* using vectors of different dimensionality.

	GloVe				Word2Vec
Category	50d	100d	200d	300d	300d
capital-common-countries	0.4881	0.7272	0.6699	0.6956	**0.5612**
capital-world	0.3320	0.4969	**0.5369**	0.5307	0.4199
currency	0.0612	0.0900	0.0981	0.0842	**0.2378**
family	0.4308	0.4920	0.4703	0.4644	**0.5217**
city-in-state	0.1179	0.1564	0.1901	0.2180	**0.3056**
gram1-adjective-to-adverb	0.0312	**0.0564**	0.0262	0.0332	0.0362
gram2-opposite	0.0073	0.0332	0.0357	0.0418	**0.0812**
gram3-comparative	0.0563	0.2447	0.3190	0.3633	**0.5653**
gram4-superlative	0.0490	0.1247	0.1408	0.1524	**0.4055**
gram5-present-participle	0.0340	0.1524	0.0956	0.0871	**0.2670**
gram6-nationality-adjective	0.7148	0.8011	0.8536	0.8317	**0.8430**
gram7-past-tense	0.0269	0.0833	0.0814	0.0858	**0.1903**
gram8-plural	0.1276	0.1478	0.1201	0.1111	**0.2042**
gram9-plural-verbs	0.0471	0.1954	0.1816	0.1850	**0.2505**
Total	0.2000	0.2976	0.3125	0.3165	**0.3650**

Table 2: Recall values for the analogy model *PairDirection* using vectors of different dimensionality.

	GloVe				Word2Vec
Category	50d	100d	200d	300d	300d
capital-common-countries	0.7075	0.9229	0.9446	**0.9545**	0.8517
capital-world	0.5369	0.8390	0.9341	**0.9566**	0.8057
currency	0.0311	0.1397	0.1824	0.1986	**0.3579**
family	0.6284	0.7786	0.8517	**0.8735**	0.8458
city-in-state	0.0912	0.2638	0.4665	0.5755	**0.7146**
gram1-adjective-to-adverb	0.0997	0.2288	0.2530	0.2560	**0.3195**
gram2-opposite	0.0492	0.1576	0.1958	0.2463	**0.4273**
gram3-comparative	0.4136	0.7484	0.8378	0.8701	**0.9076**
gram4-superlative	0.1711	0.5062	0.6907	0.7691	**0.9188**
gram5-present-participle	0.2945	0.6515	0.6704	0.7035	**0.8087**
gram6-nationality-adjective	0.8267	0.8424	0.9074	**0.9174**	0.8936
gram7-past-tense	0.3115	0.5282	0.6032	0.6243	**0.7115**
gram8-plural	0.4602	0.6666	0.7642	0.7935	**0.9061**
gram9-plural-verbs	0.2586	0.5735	0.6517	0.6942	**0.7390**
Total	0.3681	0.5932	0.6871	0.7248	**0.7533**

Table 3: Recall values for the analogy model *3CosMul* using vectors of different dimensionality.

is *Switzerland*. But they are both countries. Similar case is the third analogy, where the predicted word is *daughter* and the correct one is *sister*. Both words are about females.

In order to find the computational complexity of each analogy model, we have to examine carefully their formulas. *3CosAdd* model is based on the following formula:

$$\arg\max_{d \in V}(\cos(d, c - a + b))$$

Assuming that the dimension of the vectors is N and there are k possible words in the set V, then the complexity $O(kN)$. PairDirection model is described as follows:

$$\arg\max_{d \in V}(\cos(d - c, b - a))$$

The only difference here is that we have to compute every time the vectors $d - c$ and $b - a$, so we have $k \cdot (N + 2)$ iterations, but the big O complexity is still $O(kN)$. Finally, the formula of the *3CosMul* model is:

$$\arg\max_{d \in V} \frac{\cos(d,c)\cos(d,b)}{\cos(d,a)+\epsilon}$$

In this case we have $3kN$ iterations, so the complexity is again $O(kN)$. These small differences between the models can cause some differences also in the execution time, but their computational complexity is the same.

	Predicted word	Correct Word
Abuja : Nigeria = Bern : ?	Germany	Switzerland
Abuja : Nigeria = Brussels : ?	France	Belgium
boy : girl = brother : ?	daughter	sister
amazing : amazingly = calm : ?	quiet	calm
amazing : amazingly = most : ?	relatively	mostly
young : youngest = weak : ?	weaker	weakest
young : youngest = weird : ?	strangest	weirdest
code : coding = dance : ?	choreography	dancing
code : coding = decrease : ?	decreases	decreasing
Albania : Albanian = Argentina : ?	argentine	Argentinean

Table 4: Typical errors for the analogies.

2 Retrieval Task: From Sentences to Images

2.1 Important Remarks

Information Retrieval Models

We implemented and applied four different IR models:

1. *Unigram Language Model*
   This model is based on the following equation:

   $$P(d|q) \propto P(d) \cdot \Pi_{t \in q}((1-\lambda) \cdot P(t|M_c) + \lambda \cdot P(t|M_d))$$

   where d is a document , q is a query, t the terms contained in this query, $P(t|M_c)$ is probability of occurence of term t in the corpus (collection), $P(t|M_d)$ is probability of occurence of term t in the document d and λ is a parameter in (0,1) which is tuned to optimize performance.

   We assume that $P(d)$ is the same for all the documents (sentences) meaning that a document is not more probable than another document.

2. *Additive Vector-Based Model*
   The vector representation for each sentence is computed as follows:

   $$\overrightarrow{s} = \overrightarrow{w_1} + \overrightarrow{w_2} + ... + \overrightarrow{w_n s}$$

   where $\overrightarrow{w_i}$ is the vector for the word i of the sentence s.

3. *Multiplicative Vector-Based Model*
   This model is also described in [3]. It is similar with the previous model, but now the vector representation for each sentence is computed by the multiplication of the word vectors:

   $$\overrightarrow{s} = \overrightarrow{w_1} \cdot \overrightarrow{w_2} \cdot ... \cdot \overrightarrow{w_n s}$$

4. *Combined Model*
   We compined the unigram language model and the additive vector-based model:

$$score_{comb} = \lambda \cdot score_{unigram} + (1 - \lambda) \cdot score_{embeddings}$$

   First, we normalized the scores to the interval [0,1] using the equation $normalizedValue = (value - min)/max - min$.

Aggregation Heuristics

We implemented and applied the following aggregation heuristics:

1. *Single Aggregation*
   The result of a query is the image associated with the best scoring sentence. The other low-scored sentences for this image are ignored.

2. *Average Aggregation*
   The score of each image is based on the average score of the sentences associated with this image.

Another possible aggregation could be the weighted average. The single aggregation can be seen as weighted average with weight 1 for the best-scoring sentence and weights 0 for the rest sentences of the image. The average aggregation is weighted average with equal weights for all the sentences. The weighted average aggregation is a heuristic between the single aggregation and average aggregation, where the weights are higher for the high scores. For example, assume that the image img has 4 sentences s_i, where s_1 is the best scoring sentence, s_2 the second best scoring sentence etc. The final score for this image could be:

$$score_{img} = 0.4 \cdot score_{s_1} + 0.3 \cdot score_{s_2} + 0.2 \cdot score_{s_3} + 0.1 \cdot score_{s_4}$$

This way we make the high score sentences more important, but we also take into account the rest sentences. The intuition behind this is that if a sentence has a very high score, then it is very probable that this image is the correct one, even if the rest sentences have very low scores. However, we do not ignore totally these low weights.

Evaluation

For the evaluation, we are using the Recall@N and the Mean Reciprocal Rank (MRR). Regarding the Recall@N, the user can specify the number N of top candidates. Recall is inscreased every time the correct image is in the top N images. The default value for N is $N = 1$.

More details of how to run the above models and aggregation techniques you can find in the README file.

2.2 Implementation Details

A short description for the classes we used:

- `ImageSearch.java`: Contains the `main` method, reads the parameters and executes the selected IR model.
- `Sentence.java`: The terms of each sentence.
- `Image.java`: Each image is described by the image id (e.g. 3385593926_d3e9c21170.jpg) and the sentences (but not the sentence used as query).
- `Query.java`: Extends the class `Sentence` and contains also the rankings of each sentence for this query. It applies the selected aggreagation heuristic.
- `Ranking.java`: Contains a specific image and a ranking value for this image.
- `RankingComparator.java`: Provides a `compare` method for comparing rankings based on their values.
- `IRModel.java`: An interface for the IR models.
- `UnigramLanguageModel.java`: Implements the Unigram Language Model as described in 2.1.
- `VectorBasedModel.java`: An abstract class for the vector based models. It reads the file with the vectors.
- `AdditiveVectorBasedModel.java`: Extends the abstract class *VectorBasedModel* and implements the additive vector-based model.
- `MultiplicativeVectorBasedModel.java`: Extends the abstract class *VectorBasedModel* and implements the multiplicative vector-based model.
- `CombinedModel.java`: Compines the unigram language model and the additive vector-based model.
- `IRModelEvaluation.java`: Computes the Recall@N and the Mean Reciprocal Rank (MRR).

More details about the implementation you can find in the comments of the code.

2.3 Experiments and Results

Since we choose randomly a sentence as a query sentence, the results (recall) are not the same every time we run the experiments. So, in order to have more accurate evaluation results, we run 5 executions for each experiment. The results shown in the following tables are the average values of these 5 executions.

	Recall@1	Recall@5	Recall@10	Recall@20	MRR
single aggregation	0.447	0.701	0.779	0.849	0.5564
average aggregation	0.476	0.711	0.801	0.860	0.5810

Table 5: Average values from 5 executions for the Unigram Language Model ($\lambda = 0.5$) using different aggregation heuristics.

	d=50		d=100		d=200		d=300	
	R@1	MRR	R@1	MRR	R@1	MRR	R@1	MRR
single aggregation	0.262	0.357	0.327	0.425	0.38	0.475	0.389	0.499
average aggregation	0.197	0.289	0.255	0.355	0.305	0.413	0.345	0.441

Table 6: Evaluation results (average values from 5 executions) for the *Additive Vector-Based Model* using different aggregation methods and vectors with different dimensions.

Unigram Language Model

Table 5 shows the results for the unigram language model. We observe that the recall is higher as we increase the top N canditates. The value 0.849 for *Recall*@20 means that 849 out of the 1000 queries returned the correct image in top 20 images. Comparing the single aggregation with the average aggregation heuristic, the average aggregation has better results.

Regarding the parameter λ for this model we used $\lambda = 0.5$. We tried also different values for λ, but we did not observe significant difference in the results.

Vector-Based Models

Table 6 shows the results for the additive vector-based model. We observe that we get better results as we increase the dimension of the vectors. However, even the highest value of Recall@1 (0.389 for $d = 300$) is lower than the Recall@1 we had in the unigram language model. So, the unigram language model seems to be better than this model. Another difference is that in this case the single aggregation gives better results than average aggregation.

The main problem with this model is that the order of the words cannot influence the result. Using different syntax (order) could result to different semantics. However, this model cannot capture these semantics.

[3] describes a simple multiplicative vector-based model. We also implemented this model, but as you see in Table 7 the scores are very low. This model has also the same problem with the additive model since it works with 'bag of words' and not with the order of the words.

[3] proposes also other models which are syntax aware. For example, giving weights to each word have different results for different word order ($s = \alpha \cdot word_1 + \beta \cdot word_2$). Another approach is the combination of the additive and the multiplicative model ($s = \alpha \cdot word_1 + \beta \cdot word_2 + \gamma \cdot word_1 \cdot word_2$)

	d=50		d=100		d=200		d=300	
	R@1	MRR	R@1	MRR	R@1	MRR	R@1	MRR
single aggregation	0.014	0.024	0.016	0.032	0.015	0.035	0.007	0.021
average aggregation	0.004	0.014	0.003	0.012	0.009	0.027	0.005	0.013

Table 7: Evaluation results (average values from 5 executions) for the *Multiplicative Vector-Based Model* using different aggregation methods and vectors with different dimensions.

	$\lambda = 0.1$		$\lambda = 0.3$		$\lambda = 0.5$		$\lambda = 0.7$		$\lambda = 0.9$	
	R@1	MRR	R@1	MRR	R@1	MRR	R@1	MRR	R@1	MRR
single aggr.	0.436	0.530	0.454	0.557	0.466	0.568	0.472	0.579	0.455	0.557
average aggr.	0.372	0.447	0.462	0.537	0.469	0.568	0.473	0.581	0.475	0.586

Table 8: Evaluation results (average values from 5 executions) for the *Combined Model* using different aggregation methods and different values of λ (the dimension of the vectors for the additive vector-based model is $d = 50$).

Combined Model

This model compines the unigram language model and the additive vector-based model. In order to show how the parameter λ influences the results we chose the dimension $d = 50$ which gives the lowest values for recall. Using $d = 300$ has similar results with the unigram language model, so the difference was not obvious for the different values of λ.

Table 8 shows the results of the compined model. When the λ is low the score of each image is closer to the score produced by the vector-based model. As a result, the recall and MRR values are not as high as in the unigram language model which performs better ($recall_{unigram} = 0.476$, $recall_{combined} = 0.372$). On the other hand, using large values for λ (e.g $\lambda = 0.9$) gives higher weight to the unigram language model, so the results are very close to this model ($recall_{unigram} = 0.447$, $recall_{combined} = 0.455$ and $recall_{unigram} = 0.476$, $recall_{combined} = 0.475$)

Using the entire Flickr8K dataset

We tried the unigram language model and the additive-vector based model over the entire Flickr8K dataset (TEST + TRAIN + DEV) . As we expected, since the search space is expanded, the number of correct images returned as result was decreased having as result lower values of recall and MRR (Table 9). We observe again that the unigram language model performs better than the vector-based model.

	Unigram Language Model		Additive Vector-Based Model (d=300)	
	R@1	MRR	R@1	MRR
single aggr.	0.259	0.342	0.199	0.285
average aggr.	0.279	0.376	0.196	0.274

Table 9: Evaluation results (average values from 5 executions) running the two models over the entire Flickr8K dataset.

Bibliography

[1] O. Levy and Y. Golderg. Linguistic regularities in sparse and explicit word representations. *Proceedings of the Eighteenth Conference on Computational Language Learning*, pages 171–180, 2014.

[2] T. Mikolov, W. Yih, and G. Zweig. Linguistic regularities in continuous space word representations. *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, 2013.

[3] J. Mitchell and M. Lapata. Vector-based models of semantic composition. *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 236–244, 2008.