Text Mining Seminar

GloVe: Global Vectors for Word Representation

Munich, April 11, 2018

Department of Statistics
Ludwig Maximilian University of Munich

Degree course: M.Sc. Statistics

Student:

Daniel Schalk
(11470019)

Advisor:

Prof. Dr. Christian Heumann

Contents

1 Introduction

An important technique within the context of text mining is to map words to vectors. Those word embeddings can be used as features in statistical models to do further analyses. GloVe is a technique to create word embeddings out of a given corpus. A very interesting fact is that we want to learn from text without having labels. Hence, we have an unsupervised task.

Word vectors should be able to represent the human understanding of text in a very basic way. That means that word vectors should be able to display the analogy of different words as well as general semantic questions. How this is achieved shows image 1.1.

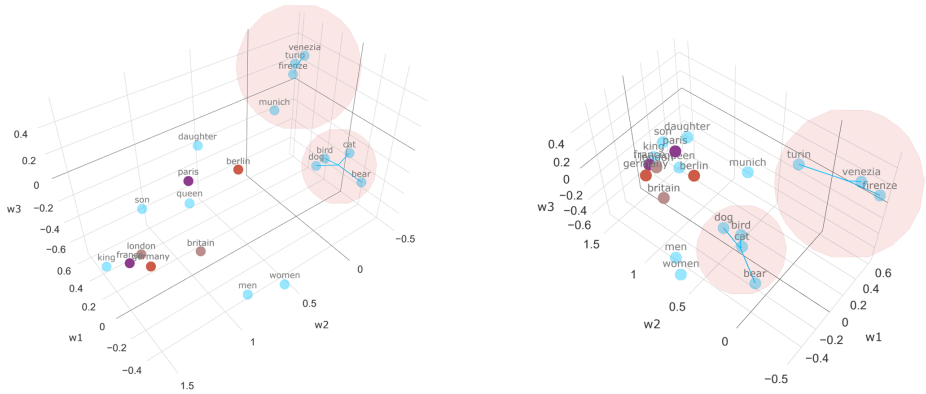

Figure 1.1: Semantic analogies are stored within the difference of word vectors. For instance, the "is capital" context is represented by the difference between "france and paris" or "germany and berlin" which looks very similar. Nevertheless, the "britian and london" difference looks very different. Hence, a high dimension is used for word vectors (e. g. 300 or 500). Additionally, we want that the word vectors build cluster to related words (e. g. italian cities or animals).

In the following we want to discuss some important topics related to GloVe. After deriving the model in section 2 we take a look at how to evaluating given word vectors in section 3 which is quite interesting since we handle an unsupervised task. After that we take a short look at the data and common sources for text corpora in section 4.

Then we know how to evaluate word vectors and have an idea about the data. With that knowledge we may ask how the dimension or different data sources influences the quality of the word vectors. This is discussed shortly in section 5. After that we give an outlook on how GloVe embeddings can be used for further text classification techniques followed from a small conclusion in section 6.

2 GloVe Model

2.1 Word-Word Co-Occurence Matrix

Base of the model is the word-word co-occurrence matrix $X \in \mathbb{R}^{|V| \times |V|}$, where $|V|$ is the number of different words which occurs within a given corpus. In X every entry X_{ij} describes how often word j occurs in context of word i with a given window size. Therefore we have the following properties:

- $X_i = \sum_k X_{ik}$: Number of words which occur in the context of i.
- $P_{ij} = P(j|i) = X_{ij}/X_i$: "Probability" of word j occurs in context of i.
- The ratio P_{ik}/P_{jk} describes if word k is more related to ...
  - ... word i if $P_{ik}/P_{jk} > 1$
  - ... word j if $P_{ik}/P_{jk} < 1$
  - ... or similar related if $P_{ik}/P_{jk} \approx 1$

Example: word-word co-occurenece Matrix

To understand how an entry of the word-word co-occurence matrix is computed we take a look at the following sample corpus:

$$\text{A D C E A D E B A C E D} \;\Rightarrow\; V = \{\text{A B C D E}\},\; |V| = 5$$

Furthermore, we need to specify a window size which indicate how much words we want to look at around a specific word. We choose a window size of 2 (the 2 words of the either side) for this example. With the given corpus and the window size we now compute the counts of how often word D occurs in context of word A (X_{AD}):

$$\begin{aligned}
&\text{A D C E A D E B A C E D} \;\Rightarrow \text{D occurs 1 time}\\
&\text{A D C E A D E B A C E D} \;\Rightarrow \text{D occurs 1 time}\\
&\text{A D C E A D E B A C E D} \;\Rightarrow \text{D occurs 0 times}
\end{aligned}$$

$$\Rightarrow \text{D occurs 2 times in context of A: } X_{AD} = 2$$

Finally, we do this for every word combination of $(i, j) \in V \times V$ and calculate the ratio P_{AD}:

$$X = \begin{array}{c|ccccc} & \text{A} & \text{B} & \text{C} & \text{D} & \text{E} \\ \hline \text{A} & 0 & 1 & 3 & 2 & 4 \\ \text{B} & 1 & 0 & 1 & 1 & 1 \\ \text{C} & 3 & 1 & 0 & 2 & 2 \\ \text{D} & 2 & 1 & 2 & 0 & 4 \\ \text{E} & 4 & 1 & 2 & 4 & 0 \end{array} \quad \Rightarrow \quad P_{\text{AD}} = X_{\text{AD}}/X_{\text{A}} = 2/10$$

Sidenote:

Some packages (like R's `text2vec`) have an option to give a weight vector with as many weights as the window size. It is then common to weight context words closer to the inspected word higher than words which aren't that close.

2.2 The Model

GloVe was introduced by Jeffrey Pennington, Richard Socher and Christopher D.Manning [PSDM14]:

> "*... GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.*"

Remember: We want to create word vectors $w_i \in \mathbb{R}^d$ and want to use the ratio P_{ik}/P_{jk} since the ratio is more appropriate to take words into context.

The idea is to take a function F which takes the three words i, j and k, given by using the ratio, and map F to that ratio:

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

Here w_i, w_j are context vectors while $\tilde{w}_k$ is the context vector. F is unknown at this stage and basically can be any function. But we remember that we want to keep the linear structure of the vector space $\mathbb{R}^d$. So it is important to choose a function F which is able to hold the linear structure. For instance we can choose F to be a neural net. But that wouldn't guarantee that F keeps the linear structure.

We now parameterize every word vector, therefore we have $d \cdot |V|$ parameter to estimate to get word vectors. But how should we estimate those parameters without a specific function F? There are several steps to come up with a solution which fulfilles the desierd behaviour:

1. Reduce number of possible outputs of F by taking $w_i - w_j$ instead of two vectors w_i and w_j:
$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$
2. We want to keep the linear structure between the word vectors and context vector:
$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{P_{ik}}{P_{jk}}$$
3. Keep exchange symmetry. We want be able to exchange word vectors w with context vectors $\tilde{w}$ without getting different results. To obtain this property we apply to tricks:
   a) We restrict F to be a homomorphism between $(\mathbb{R}, f) = (\mathbb{R}, +)$ and $(\mathbb{R}_+, g) = (\mathbb{R}_+, \cdot)$. That means we expect F to fulfill:
$$F(a+b) = \underbrace{F(f(a,b)) = g(F(a), F(b))}_{\text{Definition of homomorphism}} = F(a)F(b)$$

With $a = w_i^T \tilde{w}_k$ and $b = -w_j^T \tilde{w}_k$.
We notice, that the upper equation implies a functional equation which has just one solution:

$$F = \exp_a$$

We choose $a = \exp(1)$, therefore $F = \exp$.

$$F\left((w_i - w_j)^T \tilde{w}_k\right) = \exp\left((w_i - w_j)^T \tilde{w}_k\right) = \frac{\exp(w_i^T \tilde{w}_k)}{\exp(w_j^T \tilde{w}_k)} = \frac{P_{ik}}{P_{jk}}$$

$$\begin{aligned} &\Rightarrow \exp(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i} \\ &\Leftrightarrow w_i^T \tilde{w}_k = \log(X_{ik}) - \log(X_i) \\ &\Leftrightarrow w_i^T \tilde{w}_k + \log(X_i) = \log(X_{ik}) \end{aligned}$$

b) Since $\log(X_i)$ is independent of k we can put this in a bias term. This bias term can be decomposed into a term b_i from w_i and $\tilde{b}_k$ from $\tilde{w}_k$:

$$\Rightarrow \; w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik})$$

This is now an model equation which we can use for training the word vectors. We also note, that we have transformed the unsupervised task into a supervised task which we know how to handle.

Sidenote:

Point 2. looks a bit like the model equation of a generalized linear model (glm) with linear predictor $(x_i - w_j)^T \tilde{w}_k$, response function F and response P_{ik}/P_{jk}. The glm takes a linear structure and transforms the output on an interval of plausible values. This also holds for GloVe by using a homomorphism as F which preserves the structure between the two algebraic structures. Therefore, it is possible to keep the linearity of the used vector space $\mathbb{R}^d$.

Model Equation

A problem appears for $X_{ik} = 0$. This is definitely the case since X is a sparse matrix. We don't want to drop the sparsity due to convenient memory handling. Therefore, we use an additive shift in the logarithm:

$$\log(X_{ik}) \;\rightarrow\; \log(X_{ik} + 1)$$

This maintains the sparsity:

$$X_{ik} = 0 \;\Rightarrow\; \log(X_{ik} + 1) = \log(1) = 0$$

The final model equation which we use for training is

$$w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik} + 1)$$

Empirical Risk

To estimate the word vectors GloVe uses a weighted least squares approach:

$$J = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} f(X_{ij}) \left(w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log(X_{ij} + 1) \right)^2$$

[PSDM14] proposed as weight function (illustrated in figure 2.1):

$$f(x) = \begin{cases} (x/x_{\max})^\alpha, & x < x_{\max} \\ 1, & x \geq x_{\max} \end{cases}$$

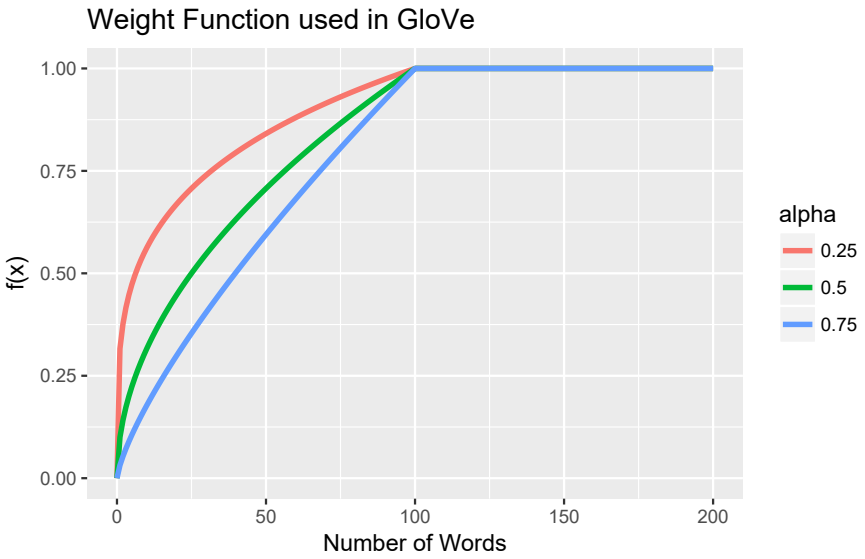

Figure 2.1: Illustration of the weight function $f(x)$ for different α values.

Using this function also introduces two new hyperparameters α and $x_{\max}$. The ordinary way to find good values for α and $x_{\max}$ would be to use a tuning method. The problem now is that the common tuning methods requires that the model can be evaluated. Since we handle an unsupervised task evaluating isn't straightforward.

Algorithm used for Fitting

Basically, GloVe uses a gradient descend technique to minimize the objective J. Nevertheless, using ordinary gradient descend would be way too expensive in practice. A more appropriate gradient based method is used here called adaptive gradient descent (AdaGrad). A short description of the most important properties of AdaGrad transferred to text mining are:

- Individual learning rate in each iteration.
- Adaption of the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameter.
- Well-suited for sparse data, improve robustness of (stochastic) gradient descent.
- For GloVe, infrequent words require much larger updates than frequent ones.

For a more detailled explanation see [Rud16].

3 Evaluating Word Vectors

3.1 Evaluation Metrics

We need to measure the distance between vectors to define similarity between word vectors and therefore analogies between words:

- **Euclidean Distance**:

$$d_{\text{euclid}}(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- **Cosine Distance**

$$d_{\text{cosine}}(x, y) = 1 - \cos(\angle(x, y)) = 1 - \underbrace{\frac{\langle x, y\rangle}{\|x\|_2\|y\|_2}}_{\text{Cosine Similarity}}$$

In general we could use every metric to evaluate the model. But the cosine similarity is more robust in terms of curse of dimensionality.

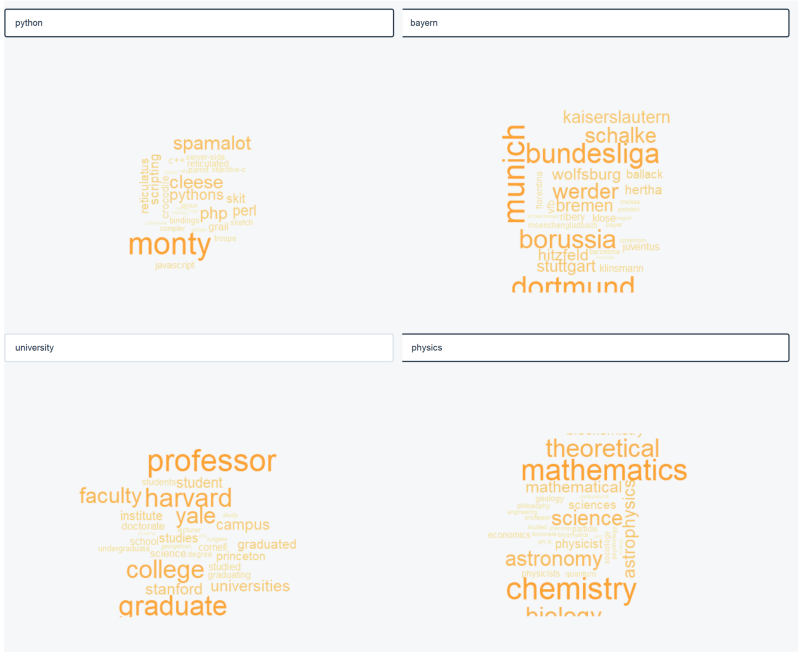

Figure 3.1: Different word clouds for the words python, bayern, university and physics. The bigger the words, the smaller the cosine distance. The images are taken from an shiny application programmed for this seminar using pre trained word vectors from [PSDM14] using the wikipedia dump (see section 4).

3.2 p-Norm in high Dimensions

[AHK01] have shown, that in high dimensions the ratio of the maximal norm divided by the minimal norm of n points $x_1, \ldots, x_n$ which are randomly drawn converges in probability to 1 for increasing dimension d:

$$\operatorname*{p\,lim}_{d\to\infty} \frac{\max_k \|x_k\|_2}{\min_k \|x_k\|_2} = 1$$

This means if we use the euclidean norm, in high dimensions points are almost surely concentrated on the surface of a hyper sphere. The same holds for every p-Norm. Figure 3.2 shows an simulation of the p-Norm in high dimension.

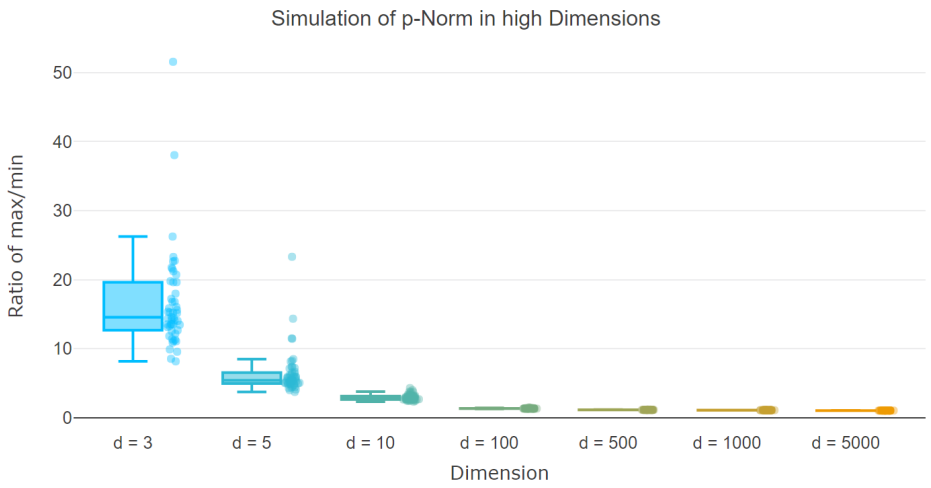

Figure 3.2: Simulating the curse of dimensionality by drawing 100 points of a uniform distribution on a hypercube and calculating the ratio of the longest and the shortest vector. This is repeated 50 times for each of the dimensions 3, 5, 10, 100, 500, 1000 and 5000.

3.3 Semantic and Analogies

One thing we can do now is to ask for semantic analogies between words. Something like:

paris behaves to france like berlin to ?
animal behaves to animals like people to ?
i behaves to j like k to l

Therefore, we have 3 given word vectors w_i, w_j and w_k. To get the desired fourth word l we use the linearity of the word vector space:

$$w_l \approx w_j - w_i + w_k$$

We can explain this linear combination by remembering the introduction where we explained, that the context of words (e. g. is capital) is memorised within the difference of

words. We now add this context to the desired word to obtain the corresponding context word. Furthermore, we obtain $\widehat{l}$ from our model and a given metric $d(w_i, w_j)$ (mostly $d = d_{\text{cosine}}$) by computing:

$$\widehat{l} = \arg\min_{l \in V} d(w_j - w_i + w_k, w_l)$$

3.4 Questions Words File

To evaluate trained word vectors, [MCCD13] provide a word similarity task. This task is given within a question words file which contains about 19544 semantic analogies. The following list shows the first 10 tasks:

```
: capital-common-countries
Athens Greece Baghdad Iraq
Athens Greece Bangkok Thailand
Athens Greece Beijing China
Athens Greece Berlin Germany
Athens Greece Bern Switzerland
Athens Greece Cairo Egypt
Athens Greece Canberra Australia
Athens Greece Hanoi Vietnam
Athens Greece Havana Cuba
Athens Greece Helsinki Finland
```

Basically, this 19544 tasks are separated into 14 categories. Table 3.1 shows all the categories, how much tasks each category has and an example for each category.

3.5 Hyperparamter Tuning

Having such a question words file can now be used to evaluating word embeddings. Therefore, each of the 19544 words are evaluated as explained in section 3.3. If the estimated word matches the fourth word the corresponding line gets a 1, if not than a 0. Finally, to get a score how good our word embedding is we average over all these zeros and ones to obtain the accuracy.

We remember, that creating word embeddings is an unsupervised task. But using a question word file makes it possible to compare different word embeddings and therefore make tuning possible. Nevertheless, this tuning highly depends on the question words file and fits the task which is captured there. If we want to test other properties of the embeddings, then we need another file capturing this property.

[PSDM14] have tuned the model and state out that good values for the hyperparameters are:

- $\alpha = 0.75$
- $x_{\max} = 100$ (does just have a weakly influence on performance)

Category	**Number of Test Lines**	**Example**
capital-common-countries	506	Athens Greece Baghdad Iraq
capital-world	4524	Abuja Nigeria Accra Ghana
currency	866	Algeria dinar Angola kwanza
city-in-state	2467	Chicago Illinois Houston Texas
family	506	boy girl brother sister
gram1-adjective-to-adverb	992	amazing amazingly apparent apparently
gram2-opposite	812	acceptable unacceptable aware unaware
gram3-comparative	1332	bad worse big bigger
gram4-superlative	1122	bad worst big biggest
gram5-present-participle	1056	code coding dance dancing
gram6-nationality-adjective	1599	Albania Albanian Argentina Argentinean
gram7-past-tense	1560	dancing danced decreasing decreased
gram8-plural	1332	banana bananas bird birds
gram9-plural-verbs	870	decrease decreases describe describes

Table 3.1: Examples for questions per category within the question word file.

4 About the Data

4.1 The Language

Be careful with the language. For instance, German has much more rare words and a bigger variety (harder for modelling) than English since more words become more rare. Table 4.1 shows an example of comparing related words in both languages focusing on the term hydrogen.

German term list	**English term list**
Wassermolekuel	hydrogen
Wasserstoff	hydrogen-bonding
Wasserstoffatom	
Wasserstoffbindung	
Wasserstoffbrueckenbildung	
Wasserstoffbrueckenbindung	
Wasserstoffhalogenid	
Wasserstoffverbindung	

Table 4.1: Copied from Grammar & Corpora 2009 [Kon11]. Excerpt of the resulting English and German term lists focusing on the term *hydrogen* (German: *Wasserstoff*).

4.2 Common Sources

To train word embeddings we need a lot of words to train those models. GloVe makes no difference. Common sources have multiple billion words an a total size of multiple gigabytes. To get that huge amount of words the common way is to crawl from the internet. This is mostly followed by a lot of preprocessing such as removing regular expressions (HTML syntax) or filtering stop words (very common words as the or this).

This also imply two question which has been asked:

1. How big should the corpus and the vocabular be to get good word vectors?
2. Does different corpora imply a different quality of the word vectors?

We want to answer the first one now while the second one will be answered in section 5. The most common sources for word embeddings are the following (tokens all words within the corpus while the vocabulary describes the number of unique words $|V|$):

- **Wikipedia Dump + Gigaword 5:** Wikipedia gives access to all articles collected within one XML file (unzipped about 63 GB). [PSDM14] combines this with Gigaword 5, an archive of newswire text data.

  → 6 billion tokens and 400 thousand word vocabulary

- **Common Crawl:** Published by Amazon Web Services through its Public Data Sets program in 2012. The data was crawled from the whole web and contains 2.9 billion web pages and over 240 TB of data.

  → 42 billion tokens and 1.9 million word vocabulary or
  → 820 billion tokens and 2.2 million word vocabulary

- **Twitter:** [PSDM14] crawled 2 billion tweets.

  → 27 billion tokens and 1.2 million word vocabulary

5 Real Word Vectors

5.1 Training Word Vectors

Within the scope of this seminar we have trained own word vectors using GloVe from the `R` package `text2vec` (see [SW17]). The word vectors are trained on the full Wikipedia dump using the first 453.77 (16 %) million words. We were not able to use the full corpus due to memory consumptions. For the training we have used the same setting as [PSDM14]:

- $\alpha = 0.75$
- $x_{\max} = 100$
- Learning rate of 0.05
- Number of iterations $= \begin{cases} 50 & \text{if } d < 300 \\ 100 & \text{otherwise} \end{cases}$.

5.2 Trained Word Vectors

To answer the second question of section 4.2 we now focus on real word embeddings. Therefore we use two different types:

- Own word vectors which are trained as explained above.
- Pre trained word vectors by [PSDM14]. Those word vectors can be downloaded from their homepage `https://nlp.stanford.edu/projects/glove/`

The following two subsections shows tables of own word vectors and pre trained word vectors. Those word vectors are evaluated using the word similarity task described in section 3. Finally, to get a better understanding of the dependency of the corpora and the quality of word vectors we take a look at figure 5.1.

5.2.1 Table of Own Word Vectors

Category	**Number of Test Lines**	**25 d**	**50 d**	**100 d**	**200 d**	**300 d**
capital-common-countries	506	0.3715	0.6739	0.8636	0.8715	0.8854
capital-world	4524	0.1461	0.3333	0.4657	0.4348	0.4726
currency	866	0.0035	0.0127	0.0173	0.0231	0.0150
city-in-state	2467	0.1110	0.2345	0.4191	0.4664	0.5306
family	506	0.3794	0.5474	0.6186	0.6047	0.6344
gram1-adjective-to-adverb	992	0.0554	0.1562	0.2006	0.1925	0.1986
gram2-opposite	812	0.0222	0.0480	0.1195	0.1527	0.1884
gram3-comparative	1332	0.3033	0.4775	0.6329	0.7065	0.7680
gram4-superlative	1122	0.0784	0.1613	0.2843	0.3084	0.3422
gram5-present-participle	1056	0.2377	0.3191	0.4186	0.4934	0.5625
gram6-nationality-adjective	1599	0.5358	0.7199	0.8054	0.8554	0.8698
gram7-past-tense	1560	0.2032	0.3436	0.4609	0.5385	0.5532
gram8-plural	1332	0.3326	0.6051	0.7568	0.8303	0.8116
gram9-plural-verbs	870	0.1793	0.2713	0.4149	0.4977	0.5402
Accuracy		**0.1992**	**0.3469**	**0.4688**	**0.4989**	**0.5300**

Table 5.1: Accuracy of own word vectors using GloVe.

5.2.2 Table of Pre-Trained Word Vectors

Category	Number of Test Lines	Wikipedia + Gigaword 5				Common Crawl	Twitter			
		50 d	100 d	200 d	300 d	300 d	25 d	50 d	100 d	200 d
capital-common-countries	506	0.7984	0.9407	0.9526	0.9545	0.9466	0.1502	0.2688	0.5178	0.6477
capital-world	4524	0.6877	0.8840	0.9421	0.9598	0.9229	0.0969	0.2452	0.5050	0.6796
currency	866	0.0866	0.1455	0.1663	0.1617	0.1663	0.0000	0.0080	0.0120	0.0199
city-in-state	2467	0.1585	0.3020	0.5071	0.6044	0.7815	0.0243	0.0669	0.1767	0.3522
family	506	0.7016	0.8004	0.8360	0.8775	0.9130	0.3063	0.4565	0.5929	0.6660
gram1-adjective-to-adverb	992	0.1361	0.2359	0.2500	0.2248	0.2913	0.0131	0.0343	0.0746	0.1159
gram2-opposite	812	0.0924	0.2020	0.2365	0.2685	0.3374	0.0582	0.0913	0.1865	0.2447
gram3-comparative	1332	0.5375	0.7905	0.8566	0.8724	0.8566	0.1682	0.4429	0.6306	0.7342
gram4-superlative	1122	0.2834	0.5490	0.6756	0.7077	0.8601	0.1542	0.3734	0.5570	0.5998
gram5-present-participle	1056	0.4167	0.6884	0.6752	0.6951	0.8002	0.2074	0.4706	0.6136	0.6553
gram6-nationality-adjective	1599	0.8630	0.8856	0.9243	0.9256	0.8799	0.1427	0.3024	0.4799	0.6160
gram7-past-tense	1560	0.3596	0.5340	0.5859	0.5962	0.4910	0.1737	0.3532	0.4923	0.5051
gram8-plural	1332	0.5901	0.7162	0.7718	0.7785	0.8401	0.2020	0.4932	0.6441	0.7575
gram9-plural-verbs	870	0.3805	0.5839	0.5954	0.5851	0.6425	0.1885	0.4034	0.5460	0.5241
Accuracy		**0.4645**	**0.6271**	**0.6934**	**0.7157**	**0.7472**	**0.1212**	**0.2744**	**0.4358**	**0.5385**

Table 5.2: Accuracy of pre trained word vectors using GloVe.

5.2.3 Accuracy of Word Vectors

As we can see in figure 5.1 the quality of word vectors depends on the size of the corpus but also on the quality. If we try to explain why the pre trained word vectors of the Wikipedia + Common Crawl corpus have a higher accuracy as the word vectors of the twitter corpus we may think of the way the text is created. For instance, the Wikipedia articles should have a higher quality in terms of grammar and linguistic than the tweets.

Another interesting comparison are the twitter word vectors and the own word vectors which are trained on much less data. The own word vectors are good in lower dimension and are overtaken from dimension 200. The difference here is the huge amount of words. The Twitter corpus has about 54 times more words to train the vectors. This difference seems to have especially an impact on higher dimensions.

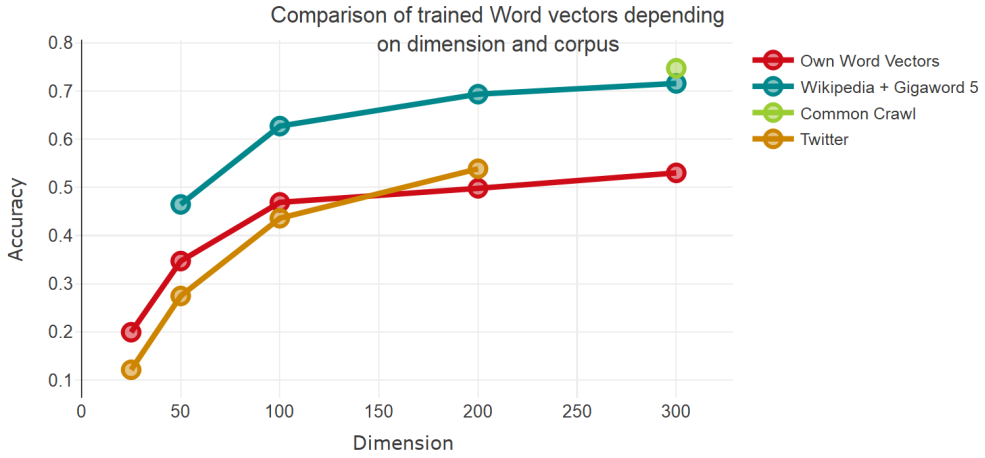

Figure 5.1: Comparison of the overall accuracy of own word vectors and pre trained word vectors for different dimensions.

6 Outlook and Conclusion

6.1 Outlook

An example to use word embeddings for more advanced tasks is text classification. The idea is to map a collection of words given by a text to an image:

1. Imagine the following word vectors obtained by GloVe:

$$\begin{array}{c|ccc} A & 0.1 & 0.2 & 0.1 \\ B & 0.2 & 0.5 & 0.3 \\ C & 0.9 & 0.7 & 0.3 \\ D & 0.4 & 0.8 & 0.1 \end{array} \quad \Rightarrow \quad w_i \in \mathbb{R}^3,\ d = 3$$

   Note that word vectors created by GloVe are organized as rows within the matrix where the rownames displays the vocabulary of available words.

2. Now we have a given text:

   A B A D B C

3. Next, each word in the text is mapped to the corresponding word vector to obtain a matrix for the given text:

A	B	A	D	B	C
↓	↓	↓	↓	↓	↓
0.1	0.2	0.1	0.4	0.2	0.9
0.2	0.5	0.2	0.8	0.5	0.7
0.1	0.3	0.1	0.1	0.3	0.3

4. This matrix can be used to create an image out of the given text by translating the matrix to an image:

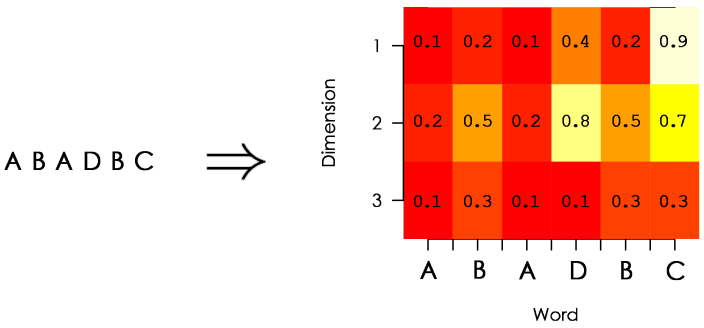

Figure 6.1: Explanation of the mapping between text and image.

The images in figure 6.2 were generated from the introduction of Principia Mathematica by Sir Isaac Newton and Elements of Statistical Learning, as well as from Sonnet 18 by William Shakespear.

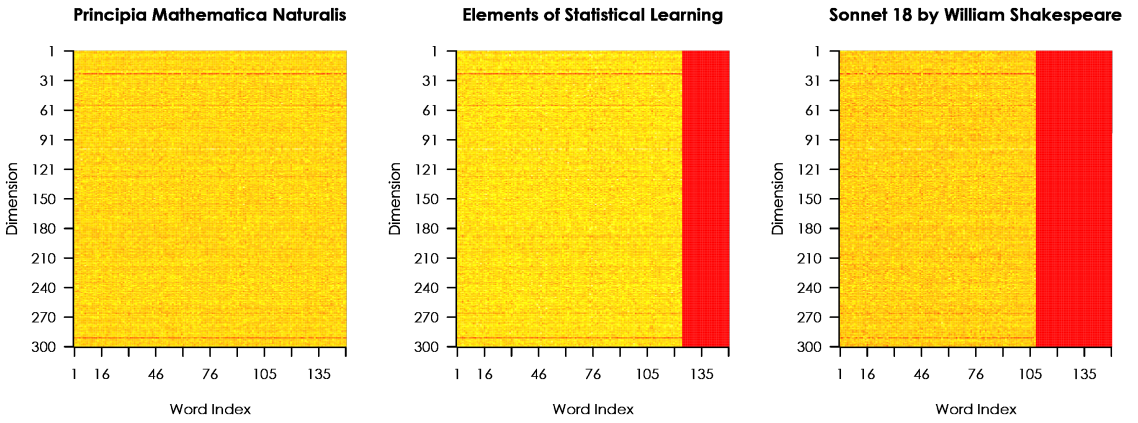

Figure 6.2: Creating images for text using the introduction of Principia Mathematica by Sir Isaac Newton and Elements of Statistical Learning, as well as from Sonnet 18 by William Shakespear. The red bars are missing words filled up to 150 words since image classifier expect a fixed number of pixels.

The next step would be to run an image classifier (e. g. a CNN) on the images and an label. This classifier could find differences by analysing specific patterns or maybe the color intensity.

6.2 Conclusion

- With GloVe it is possible to create meaningful word embeddings.
- [PSDM14] compared several models on the word analogy task, whereas GloVe outperforms the others.
- But accuracy of word analogy task depends not only on the dimension, but also on the corpus.
- Use servers or pcs with huge computing power to train GloVe due to the need of huge corpora an computing time!

List of Figures

List of Tables

Bibliography

[AHK01] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer, 2001.

[Kon11] Marek Konopka. *Grammar & Corpora 2009*, volume 1. BoD–Books on Demand, 2011.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[PSDM14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[Rud16] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[SW17] Dmitriy Selivanov and Qing Wang. *text2vec: Modern Text Mining Framework for R*, 2017. R package version 0.5.0.