Advanced Natural Language Processing

Recognition of Animated Nouns with word2vec

Arkadi Schelling*
University of Potsdam

This essay describes an experiment to use Stanford's NER and Google's word2vec on the German part of the Gutenberg corpus in order to recognize animated nouns. For theoretical reasons the results should not be to good. A rough evaluation is showing a poor precision of 50%.

1. Introduction

> The violence of language consists in its effort to capture the ineffable and, hence, to destroy it, to seize hold of that which must remain elusive for language to operate as a living thing
> – Judith Butler, Excitable Speech: A Politics of the Performative

Judith Butler's concern about the normative power of language resulted in this article. The German language distinguishes between a female and a male form for nearly all nouns that are referring to people. Still, in many cases only the male form is used, whereas females stay unvoiced. This so called generic masculine shows up mainly in plurals that are actually referring to both genders. "In Potsdam beklagen Studenten den Mangel an Wohnheimplätzen." (Spiegel Online)[1] Another preeminant appearance of the generic masculine is when talking about generic persons, e. g. in job offers "C++ Programmierer gesucht!" Due to the social and psychological implications that have been uncovered by scientists in the years after Butler's Gender Trouble, the German legislative introduced measures to promote so-called Gender-fair that is trying to voice both genders. Despite resistance in parts of German society, others start to promote an "inclusive language" that is disclosing other normalization processes in language that might marginalize people – like "Let's go", though that might be deemed difficult for disabled people in the group. The main idea that led to this paper was to develop a NLP tool, that can automatically translate from non-inclusive language into inclusive language. The first task to this would be the recognization of generic masculine forms in a text, with the recognition of nouns referring to people as the very first step. Even though easier suggestions to solve the problem exist, for pedagogical reasons the following approach was chosen.

For this article a German corpus was processed by Named Entity Recognition, giving a distinct tag to persons. All these named entities with a person tag were substituted by one single person token. This person token should show up in surroundings that are typical for persons. I created a word representation from this preprocessed text. Other

* Student, M. Sc. Cognitive Systems, Student-ID 779135. E-mail: arkadi.schelling@gmail.com.

1 `http://www.spiegel.de/unispiegel/studium/in-potsdam-beklagen-studenten-den-mangel-an-wohnheimplaetzen-a-860586.html`

words referring to persons should then be in a close proximity to the person tag. The assumption to this is that the contexts of personal names and other nouns referring to persons are similar. The results of this experiment are refuting this claim.

2. Choice of NLP Tools, Corpus and Proof of Concept

1. The choice of a corpus is essential for good results. Usually the trade-off is between size and quality. In our case between the German Wikipedia and the German Gutenberg corpus. The choice fell on the Gutenberg Corpus.
2. Due to the lack of other easily accessible tools the choice for NER fell on Stanford's NER. The heavy downside is its bad memory allocation, slow speed and poor precision. (See below)
3. The main two tools for word representations currently used are Google's word2vec and Stanford's GloVe. Radim Řehůřek, the not completely unbiased author of word2vec's Python API, compared both programs.[2] Where GloVe is showing clearer theoretical insights, the implementation of word2vec is far superior in its speed and usability. There are no reliable results to proof a substantial difference in performance on NLP tasks. For actually using word2vec and gensim on German credits go to Miguel Cabrera who published a lovely blog post on this topic[3].

The NER shall only give us contexts that are typical for persons. Therefore for our application we do not care for a high recall of named entities, but mainly for precision. A quick proof of concept was done by counting true and false positives to calculate precision of Stanford's NER on the Wikipedia article about Berlin. A sentences like

> Von/O Karl/I-PER Friedrich/I-PER von/I-PER Klöden/I-PER rekonstruierter/O
> Plan/O von/O Berlin/I-LOC und/O Cölln/O zur/O Zeit/O der/O Gründung/O

shows the important feature, that multiple word named entities get one tag per word. Such compound named entities were only counted as one. Even though the article contains long lists of politicians and sportsmen, the tagger reached a rather poor precision of 75.2–81.4%[4]. Also, we have to take into account that all these tagged entities are definite nouns and do not usually appear in contexts, where we could expect a definite but generic noun. Partially this is due to the text, this should be different with novels, that tend to speak longer about one subject. For this reason we chose the German part of the Gutenberg corpus. The downsides of this are:

- The Gutenberg corpus lacks modern written language due to copyright reasons prohibiting the publishing of German books younger than 80 years.

2 `http://radimrehurek.com/2014/12/making-sense-of-word2vec/`
3 `http://mfcabrera.com/research/2013/11/14/word2vec-german.blog.org/`
4 The NER-PER tags were 85 times correct, 21 times incorrect and 7 times incomplete (e.g. Albrecht/I-PER, den/O Bären/O).

- The Gutenberg corpus has about 70M words after the maliciously complicated stripping of metadata. Whereas very good results of word2vec are reported for copora of about 1B words.

3. Named Entity Recognition

NER in German is much more difficult than in other languages. Since other languages mostly distinguish named entities orthographically by capitalization, whereas German capitalizes all nouns. Stanford's NER[5] uses Conditional Random Fields (CRFs) to solve NER for any language. A CRF is trying to solve similar tasks as Hidden Markov Models without the underlying assumption of statistical independence. In other words, the transition functions between the hidden states can vary depending on the input sequence. For parsing a text, this implies to read all instances of a word, to run the algorithm. This global structure of CRFs leads to a very high consumption of memory. In my test runs this was about 100 times more than the input files size. Stanford publishes some answers on how to decrease the amount of memory, but I kept with the simplest method of decreasing the input size. I suspect that a very large text corpus might even decrease the probability to recognize rare names in single texts of a big corpus, in comparison to analyzing these texts all separately.

4. word2vec

A traditional way to represent word meanings is captured by John Rupert Firth's quote "You shall know a word by the company it keeps." This recurs to local meaning representations – like N-grams and bag-of-words. One major drawback of these models is their discreetness and lack of metric structure, which do neither allow optimization techniques like gradient descent nor distance calculations. Because of this it is natural to ask for an embedding from a discreet meaning representation into a complete metric space, if possible with a differentiable structure. The most widely known such spaces are Euclidean spaces and other vector spaces with an inner product[6]. More complex examples are Euclidean manifolds like n-dimensional spheres, hyperbolic planes and projective spaces. First ideas to represent words as vectors came up already in the late 80s by Hinton and Rumelhart. Learning these word embeddings with Neural Networks was started by Yoshua Bengio in the early 2000s.[7]

How does a Feedforward Neural Network language model (FFNNLM) with back-propagation of errors embed an input into a vector space? A FFNNLM is a multilayer collection of perceptrons, in Bengio's article with two layers. For training with back-propagation of errors it uses an evaluation function on the output of the last layer of perceptrons. A simple perceptron has only a discreet output of $\{-1, 0, 1\}$ and cannot be used for gradient descent. To overcome this weekness perceptrons in Neural Networks are frequently modelled by the continuous and differentiable Sigmoid function. Thus each perceptron of the Neural Network can be regarded as an embedding of the input into the open interval $(-1, 1)$, which is differentiably equivalent to $\mathbb{R}$. The collection of

5 `http://nlp.stanford.edu/software/CRF-NER.shtml`
6 So called Hilbert Spaces.
7 Y. Bengio, R. Ducharme, P. Vincent. A neural probabilistic language model. Journal of Ma- chine Learning Research, 3:1137-1155, 2003. A more recent word representation of Bengio can be found here `http://metaoptimize.com/projects/wordreprs/`

all m perceptrons from the first layer of the Neural Network becomes an embedding from the input space into $\mathbb{R}^m$.

The word2vec algorithm was developed by Tomas Mikolov et. al. from Google and released in 2013[8]. It combines a local representations of word meanings with a continuous representation of these word meanings in a metric space. Apart from an efficient multithreading implementation in C, it derrives its speed from its single main difference to Bengio's FFNNLM with backpropagation algorithm. It is throwing away all hidden layers but the first one. It can thus be referred to as a shallow Deep Learning method. The local representation for input and the output are modelled on n-grams of words that stretch into both directions of a word. A 2-gram is thus a word sequence $w_{t-2}w_{t-1}w_tw_{t+1}w_{t+2}$. The algorithm can take the central word w_t as input and tries to predict the surrounding words as the output – this architecture is referred to by "Skip-gram". The second option is the so-called "Bag-of-words" architecture which takes the surrounding words as input and trains with backpropagation by trying to predict the word w_t. Due to the absence of the hidden layer, the word2vec algorithm needs more than 1 billion words for training to reach good results. This is feasible due to its speed up of about 1000 times to a total training speed of 50K–5M words per second.

The resulting embedding into a vector space does not tell how to measure the distance between words, yet. From former similar embeddings it was learned that words with different frequencies but in exactly the same contexts appear on the same rays from the origin, but in different distances. Thus their Euclidean distance is not zero, even though they should be regarded as synonimous. For these case the cosine distance is a more appropriate choice. Even though the embedding of the word2vec algorithm should not depend on the frequency of words, they chose the same metric. This is astonishing, since it means that words from different contexts might end up with zero distance. Even more, the often quoted break through in mapping the algebraic vector space structure to semantics to solve the analogy task "king + woman – man = queen" is not compatible with this synonymy relation. More explicitly, if we have three vectors v, v_1, v_2 with v_1 being linearly aligned but different to v_2 (synonimous), then $v + v_1$ is not linearly aligned to $v + v_2$ (not synonimous). In a nutshell, the geometric construction of this complete metric space for our embedding is the quotient of a vectorspace by the positive scalar multiplication, with the metric defined by the angle between the underlying equivalence classes. This means that the resulting geometry is a $m - 1$ dimensional sphere, with m the dimension of the Neural Network.

5. Data Processing

The creation of a word representation took three steps.

1. Download the German Gutenberg Corpus and concatenate all files into one text file.
2. Pass the text file to Stanford's NER in memory saving chunks and concatenate the output to a NER tagged text.

8 See `https://code.google.com/p/word2vec/` for the code and quick explanations. A scientific article was published on `http://arxiv.org/pdf/1301.3781.pdf`

3. Process the NER tagged text to lower case words without special characters like umlauts and punctuation and run Google's word2vec program on the output.

The scripts to these tasks can be found with a short readme and evaluation on my github page.[9]

6. Playing with the Results and Further Ideas

There are at least two different ways to use the created model for calculating whether a word should belong to a person class or not.

- Calculating the similarities between the person tag and setting a threshold
- Classification by clusters

The first way is working somewhat but not perfectly. We are getting similarities of 0.4, 0.3 and 0.15 for the pronouns "er", "sie", "es". Still, the words "lehrer" and "lehrerin" are giving similarities of 0.13 and 0.089, whereas even the adverb or noun "morgen" is giving a higher similarity of 0.15 to the person tag. The second way is more promising, since it is giving more degrees of freedom. In the evaluation on github I took the person, location and miscellanious tags created by Stanford's NER as clusters. This is giving a recall of 100% but only a precision of 50% on a quickly created test set of 30 words. This result might further be optimized with a bigger test set, that can be used to train a classifier with Machine Learning methods. Overall, both methods did not yet work properly. A better evaluation and classifier could be created with GermaNet. This would allow using the word classes of "Person" and "Beruf" to create a big evaluation set. Probably, this would even account for most of the words we can find in the Gutenberg corpus. An accompanying step would thus be the creation of a bigger corpus. Not forgetting the main task of classifying words that can denote generical masculine nouns, we might still come back to the promising idea of training an HMM on n-grams, similar to PoS-tagging.

9 https://github.com/arksch/NE2vec