Analyzing semantics between translations with Word2Vec

Statistical Natural Language Processing

Hölbling Rene [681432], Vilkki Eetu [428077]

November 20, 2020

1 Introduction

The first idea for the topic of this project was to evaluate the translations of natural language. In detail, we wanted to analyze how well different meanings, expressions and descriptions translate to the three different languages: English, Finnish and German. Not only an automated evaluation system for translations could be developed, but also relations between the vector spaces of the languages could be determined. This could be used to improve machine translation services (e.g. Google translate) even further.

Due to the difficulty of this task and that one of our group members dropped out in the last minute, we focused on this question: does the semantic meaning of words persist between translations within just the English and German models. This is an interesting question, as the semantic meaning of the language used in many works, like books and movies, is intended to articulate many nuances in the story and dialogue. If the translations have different semantics, the language doesn't fully represent the intentions of the original author.

For the data input we use subtitles of a collection of movies. Subtitles in different languages have the advantage that they should be almost direct translations from each other. Therefore, the models should be more comparable to each other, as they should incorporate similar semantics. Another reason we chose subtitles is that they usually represent the actual language that's used in everyday life better than books, for example. The subtitles of forty different movies are used to train our models:

```
Star Wreck: In the Pirkinning
Interstellar
The Shawshank Redemption
Harry Potter and the Deathly Hallows: Part 2
Catch Me If You Can
Hachi: A Dog's Tale
Logan (2017)
Spotlight
Gone Girl
The Wolf of Wall Street
The Avengers
The Hitman's Bodyguard
Deadpool
Kingsman: The Secret Service
```

```
The Martian
Fast Five
The Bourne Ultimatum
Superbad
The Equalizer
The Internship
Birdman
Manchester by the Sea
Adam's apples
Beasts of the Southern Wild
American Psycho
The Theory of Everything
Kill Bill Vol 1
The Big Lebowski
A clockwork orange
Forrest Gump
The Dark Knight
Fight Club
The Silence of the Lambs
Gladiator
Inception
Full Metal Jacket
The Green Mile
Reservoir Dogs
No Country for Old Men
Good Will Hunting
```

These subtitles are downloaded from `https://www.opensubtitles.org/` and use the `SubRip` Text format.

2 Methods

2.1 Preprocessing

To preprocess the SubRip Text of the subtitles a python program was written. The program merely keeps the sentences of the subtitles and deletes all the unnecessary timecodes and other stamps. After the sentences of the subtitles are extracted, we use Natural Language Toolkit (Loper and Bird, 2002) to remove the most common stopwords, as they are unnecessary noise for the model. Last, we get rid of the inflected forms of the words in the sentences, because the same stem of the word could have many different inflected forms. These inflected forms appear in a similar context, due to the relation to the stem, but every inflected word would be treated individually and have their own unique vectorial representation in the models. The solution to this problem is a stemmer for the preprocessed language. For the English model the Porter stemmer is used and for the German model the Snowball stemmer is used. These stemmers remove the common morphological and inflexional endings of words.

2.2 Word2Vec

To determine the semantic meaning of words in the context of the collected movies our approach is to calculate the ten most similar words to a given word. We train Word2Vec models (Mikolov et al., 2013b) for both languages, English and Finnish, with the subtitles from the mentioned movies and use the models to calculate the nearest neighbours of words. Although Word2Vec is an efficient and relatively simple model, it has proved to be quite reliable in calculating nearest neighbours to words and, therefore, suits our needs nicely.

Generally speaking, there are two architectures for Word2Vec models: Continuous Bag-of-Words, which can be abbreviated as CBOW, and Skip-Gram (Mikolov et al., 2013a). According to Mikolov (2013), one of the initial inventors of Word2Vec, Skip-Gram works well with small amounts of training data and also represents well even rare words. As our training set is quite small, we decided to go with Skip-Gram, but CBOW could work as well.

When implementing Word2Vec models, the tuning of the hyperparameters needs to be considered as well. For most of the parameters we are using the default values in Gensim (Řehůřek and Sojka, 2010). The reason is that we had time and manpower constraints that didn't allow for larger optimization of the hyperparameters, and in this research we weren't specifically interested in the performance of the model itself. That said, we tweaked the minimum frequency of words, or min count, number of training epochs and the feature vector dimensionality. We lowered the minimum frequency of a word from 5 to 4, as our small training set size could result in some words occurring more rarely than what's desired. Next, we increased the number of training epochs from 5 to 30 to allow the model to learn the training data more precisely. Last, the square root of the vocabulary length is used for the feature vector dimensionality, as this is a fairly common heuristic (Joglekar, 2015).

3 Experiments

To analyze if the semantic meaning of words persists between translations, a handpicked set of only twelve words and their ten nearest neighbours are investigated in both languages. The goal is to compare whether the nearest neighbours to these specific words are similar in both of the models. If they are, it would indicate that the semantic meaning has persisted, at least to some degree, between the translations. Our hypothesis is that the semantic meaning persists between translations.

We chose this qualitative research method instead of a quantitative one, due to time constraints. Words with an unambiguous and straightforward translation, like "Hund - dog", are used, as well as words with multiple possible meanings in both languages, e.g. "Zug - train" are used. Beneath is a list of the the used word pairs.

```
Leben - Life
Sozial - Social
Mächtig - Powerful
Möglichkeit - Possibility
Geld - Money
Höflichkeit - Courtesy
Zug - train
Kraft - force
Hund - dog
Gebäude - building
```

```
Idee - idea
Wort - word
```

4 Results

Below are the results of our experiments. In red you can see the neighbours of a specific word that were given as output from both of the models.

Results of the German model:

```
Leben:
Gründen Kindheit Aussterben Pferd verbringen Lüge verschlafen
Pralinen Stange Rom

Sozial:
Bindung Symphonie nutzen Rückkehr Internet Erziehung Aspekte
Enttäuschung unwichtig Schatzsuche

Mächtig:
ungewiss Täter äußerst Verweis Fans angeschossen selbstverständlich
unkontrollierbar Sonderbare angetrieben

Möglichkeit:
legal Möglicherweise Ausweg eigenartig anstrengend bearbeiten
angetrieben freien Turbo Aggressive

Geld:
Crockett Wieviel Jules’ Heckenschütze Adresse Treehorn Munition
Scagnetti Jackie übergeben

Höflichkeit:
Überreste bleichen Mitchell zustimmen Ken Durchbruch Ms loslassen
mitgenommen Sch

Zug:
wegbringen Collings SWATTeam abwärts Feld bewachen Desi Sabbat
Sturmwarnung dorthin

Kraft:
Raumzeit jeglichen Newtons Vorausgesetzt ausgeht Singularitt
Ausrüstung Rahm Hulk Rückkehr

Hund:
Severus beleidigt Aha eilig erledigt Hoppla Rox angeheuert Hunger
Killer

Gebäude:
Korridor umstellen tausend Stockwerk eingelöst East Kilometer
```

```
Raum Waterloo Prozess

Idee:
Charakter Sauce Schmeckt Therapeuten Macht's Sal Courtney nervös
vorgestellt Gazelle

Wort:
Großvater vorhatten Sch Ashland Googliness besiegen erfunden Chloe
Vorlesung beigebracht
```

Results of the English model:

```
Life:
boring sunlight extinction K shape sentence reality Rome merely happiest

Social:
certificate utility gumbo Hattori Hanzo Taxfree Bond character
flowers discipline

Powerful:
sedative craved tower ORen antimaterial capacity attorney raw
guidance arc

Possibility:
tide catharsis depth Vulgars somehow aspect rations recovery
Blatch condolences

Money:
stolen pee deadbeat cuffs Joe's taxi bums laundering Meghan peed

Courtesy:
extend hardest Sunday Annie o'clock brogues woowoo II Mia Cate

Train:
station shell Phillipa longer HTML Uhhuh chief vigil Rover Del's

Force:
accelerating Committee Chitauri produce tournament legions retreat
Martian highest traitor

Dog:
cat WHOOPING Mmm insane Stewie yoga idiot coward grass fan

Building:
occupied careless Wilkes volunteered sweeps perceive layers
administration Rio Durden

Idea:
```

```
seed nightmare stash totem ridiculous Meghan Grownups Ls
situation destiny

Word:
express clue artist Severus beaucoup certainty wicked charming
husband Slavi
```

5 Discussion

The results in the last section show that there are not a lot of similarities between the ten nearest neighbours of the two languages models. This could have a lot of reasons. Our approach could be just too simple for the comparison of two complex languages models. For example, we did not consider the different word orders of the two languages. The context of the translation, which is very important for ambiguous translations, is not particularly taken into account. Furthermore, we didn't spend enough time tuning the hyperparameters to make the models robust.

Another reason for the unexpected results of our approach could be the small corpus of just forty subtitles. The subtitles of a small set of movies could be too specific to draw conclusions about the semantic meaning of words between translations in general. Additionally, we only compared the neighbours of 12 words, which also might be too few to get reliable results. The data itself should be of good quality for the task, as the subtitles have been generated and revised by human translators. The meanings, descriptions and the messages of all the subtitles in each language should be almost the same. Although the subtitles are downloaded from an amateur website, it shouldn't compromise the quality.

Due to these reasons, it's clear that some of the neighbours in the results aren't accurate, which indicates that the models didn't fully capture the semantics of the words. Our assumption was that it wouldn't matter, as both of the models were trained in a similar manner with similar data, so they should give similar results, but as the models aren't particularly robust, they could give unreliable and therefore uncomparable results.

Although these reasons could explain the results, it's also possible that our hypothesis was wrong. The fundamental differences in the languages themselves could result in the translations having significant semantic discrepancies. It is also possible that the translators didn't take the semantics enough into account.

According to our research, the semantic meaning of words doesn't persist between translations, but it is likely that our approach wasn't thorough enough. Therefore, further quantitative research would be needed to confirm the results.

6 Acknowledgements

We would like to thank Sachin Joglekar's blog (Joglekar, 2015), which gave us a good starting point for the code of the model and explained the main parameters of a word2vec model.

7 Appendix

7.1 Input Data:

Example of the `SubRip` Text format:

```
1
00:02:05,090 --> 00:02:07,960
<i>People always ask me
if I know Tyler Durden.</i>

2
00:02:08,380 --> 00:02:09,880
Three minutes.

3
00:02:09,970 --> 00:02:12,420
This is it. Ground zero.

4
00:02:13,050 --> 00:02:15,090
Want to say a few words for the occasion?
```

7.2 German Model:

Code of the preprocessing:

```
import re
import os

def get_corpus_from_subs(path="subs", file_extension=".srt",
try_to_use_old_corpus=True):
    file_names = os.listdir(path)
    corpus = ''

    if try_to_use_old_corpus and os.path.isfile('corpus.txt'):
        file_w = open('corpus.txt', 'r')
        return file_w.read()

    for file_name in file_names:
        if file_name.endswith(file_extension):
            file = open(path + '/' + file_name, 'r', encoding="ISO-8859-1")
            corpus += process_file(file)

    file_w = open('corpus.txt', 'w')
    file_w.write(corpus)
    return corpus


def process_file(file):
    string = ''

    for line in file:
        if line.find('-->') != -1 or line.replace('\n', '').isdigit()
```

```
        or line == '\n':
            continue
        line = line.replace('<i>', '')
        line = line.replace('</i>', '')
        string += line
    string = string.replace('\n', ' ')
    string = re.sub(r'[^a-zA-ZäööüÄÖÜß.!?\' ]', '', string)
    return string + ' '
```

Code of the model:

```
import nltk
from preprocess import get_corpus_from_subs
import re
import math
from gensim.models import Word2Vec

global_stemmer = nltk.stem.snowball.GermanStemmer(ignore_stopwords=False)

class StemmingHelper(object):
    """
    Class to aid the stemming process - from word to stemmed form,
    and vice versa.
    The 'original' form of a stemmed word will be returned as the
    form in which its been used the most number of times in the text.
    """

    #This reverse lookup will remember the original forms of the stemmed
    #words
    word_lookup = {}

    @classmethod
    def stem(cls, word):
        """
        Stems a word and updates the reverse lookup.
        """

        #Stem the word
        stemmed = global_stemmer.stem(word)

        #Update the word lookup
        if stemmed not in cls.word_lookup:
            cls.word_lookup[stemmed] = {}
        cls.word_lookup[stemmed][word] = (
            cls.word_lookup[stemmed].get(word, 0) + 1)

        return stemmed

    @classmethod
```

```
    def original_form(cls, word):
        """
        Returns original form of a word given the stemmed version,
        as stored in the word lookup.
        """

        if word in cls.word_lookup:
            return max(cls.word_lookup[word].keys(),
                       key=lambda x: cls.word_lookup[word][x])
        else:
            return word

def format_output(model_output):
    words = ''
    for word in model_output:
        words += StemmingHelper.original_form(word[0]) + ' '
    return words


min_count = 4
window = 5

corpus = get_corpus_from_subs()
corpus = [word for word in corpus.split(' ') if word not in
         set(nltk.corpus.stopwords.words('german'))]
sentences = re.compile('[?!.]').split(" ".join(corpus))
sentences = list(filter(None, [sentence.strip() for sentence in sentences]))
sentences = [sentence.split(" ") for sentence in sentences]
sentences_stemmed = list(map(lambda sentence: [StemmingHelper.stem(word)
                    for word in sentence], sentences))
sentences_stemmed = [list(filter(None, sentence)) for sentence
                    in sentences_stemmed]

vocab_list = [item for sublist in sentences_stemmed for item in sublist]
vocab_len = len(set(vocab_list))

size = int(math.sqrt(vocab_len))
model = Word2Vec(sentences_stemmed, min_count=min_count, size=size,
window=window, iter=30, sg=1)

print('\nLeben:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Leben'))))

print('\nsozial:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('sozial'))))

print('\nmächtig:')
```

```
print(format_output(model.wv.most_similar(StemmingHelper.stem('mächtig'))))

print('\nMöglichkeit:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Möglichkeit'))))

print('\nGeld:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Geld'))))

print('\nHöflichkeit:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Höflichkeit'))))

print('\nZug:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Zug'))))

print('\nKraft:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Kraft'))))

print('\nHund:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Hund'))))

print('\nGebäude:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Gebäude'))))

print('\nIdee:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Idee'))))

print('\nWort:')
print(format_output(model.wv.most_similar(StemmingHelper.stem('Wort'))))
```

7.3 English Model:

Code of the preprocessing:

```
import re
import os


def get_corpus_from_subs(path="subs", file_extension=".srt",
try_to_use_old_corpus=True):
    file_names = os.listdir(path)
    corpus = ''

    if try_to_use_old_corpus and os.path.isfile('corpus.txt'):
        file_w = open('corpus.txt', 'r')
        return file_w.read()

    for file_name in file_names:
```

```
        if file_name.endswith(file_extension):
            file = open(path + '/' + file_name, 'r', encoding="latin-1")
            corpus += process_file(file)

    file_w = open('corpus.txt', 'w')
    file_w.write(corpus)
    return corpus


def process_file(file):
    string = ''
    for line in file:
        if line.find("-->") != -1 or line.replace('\n', '').isdigit()
        or line == '\n':
            continue
        line = line.replace('<i>', '')
        line = line.replace('</i>', '')
        string += line
    string = string.replace('\n', ' ')
    string = re.sub(r'[^a-zA-Z.!?\' ]', '', string)
    return string + ' '
```

Code of the model:

```
from preprocess_eetu import get_corpus_from_subs
from gensim.parsing import PorterStemmer
import nltk
import re
import math
from gensim.models import Word2Vec


global_stemmer = PorterStemmer()
min_count = 4
window = 5


class StemmingHelper(object):
    """
    Class to aid the stemming process - from word to stemmed form,
    and vice versa.
    The 'original' form of a stemmed word will be returned as the
    form in which its been used the most number of times in the text.
    """

    # This reverse lookup will remember the original forms of the stemmed
    # words
    word_lookup = {}
```

```
    @classmethod
    def stem(cls, word):
        """
        Stems a word and updates the reverse lookup.
        """

        # Stem the word
        stemmed = global_stemmer.stem(word)

        # Update the word lookup
        if stemmed not in cls.word_lookup:
            cls.word_lookup[stemmed] = {}
        cls.word_lookup[stemmed][word] = (
                cls.word_lookup[stemmed].get(word, 0) + 1)

        return stemmed

    @classmethod
    def original_form(cls, word):
        """
        Returns original form of a word given the stemmed version,
        as stored in the word lookup.
        """

        if word in cls.word_lookup:
            return max(cls.word_lookup[word].keys(),
                       key=lambda x: cls.word_lookup[word][x])
        else:
            return word


def main():
    nltk.download("stopwords")

    corpus = get_corpus_from_subs()
    corpus = [word for word in corpus.split(' ') if word not in
    set(nltk.corpus.stopwords.words('english'))]

    sentences = re.compile('[?!.]').split(" ".join(corpus))
    sentences = list(filter(None, [sentence.strip() for sentence in sentences]))
    sentences = [sentence.split(" ") for sentence in sentences]

    sentences_stemmed = list(map(lambda sentence: [StemmingHelper.stem(word) for
    word in sentence], sentences))
    sentences_stemmed = [list(filter(None, sentence)) for sentence in
    sentences_stemmed]
```

```
    vocab_list = [item for sublist in sentences_stemmed for item in sublist]
    vocab_len = len(set(vocab_list))

    size = int(math.sqrt(vocab_len))

    model = Word2Vec(sentences_stemmed, sg=1, min_count=min_count, size=size,
    window=window, iter=30)

    top_words_count = 10


def format_output(model_output):
    words = ''
    for word in model_output:
        words += StemmingHelper.original_form(word[0]) + ' '
    return words


if __name__ == "__main__":
    # execute only if run as a script
    main()
```

References

Joglekar, S. (2015). Generating a Word2Vec model from a block of Text using Gensim (Python). https://codesachin.wordpress.com/2015/10/09/generating-a-word2vec-model-from-a-block-of-text-using-gensim-python/. [Online; accessed 24-April-2018].

Loper, E. and Bird, S. (2002). NLTK: the natural language toolkit. *CoRR*, cs.CL/0205028.

Mikolov, T. (2013). word2vec-toolkit. https://groups.google.com/d/msg/word2vec-toolkit/NLvYXU99cAM/E5ld8LcDxlAJ. [Online; accessed 24-April-2018].

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013a). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., and Dean, J. (2013b). Distributed representations of words and phrases and their compositionality. *CoRR*, abs/1310.4546.

Řehůřek, R. and Sojka, P. (2010). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, pages 45–50, Valletta, Malta. ELRA. http://is.muni.cz/publication/884893/en.