Exploring Gensim implementation of Word2Vec on Harry Potter books

Mirna Baksa
FIB - Barcelona School of Informatics
Polytechnic University of Catalonia
Email: mirna.baksa@est.fib.upc.edu

Abstract *- In this report we use Gensim, the python implementation of Word2Vec on the dataset consisting of text of all seven Harry Potter books. The focus will be to show diferrent functionalities offered by Word2Vec and to try to visualise the dataset using Word2Vec combined with t-SNE.*

1 Introduction

Word2Vec was introduced in 2013. by a group of researches from Google. The researchers published their implementation in C, while the Python implementation called Gensim used in this report was published soon after.

Word2Vec essentially is a group of models used to produce word embeddings. These models are shallow two-layer neural networks which are trained to reconstruct linguistic context of words. The idea behing Word2Vec is that two words sharing similar contexts also share a similar meaning and consequently, a similar vector representation. With this, Word2Vec can be used to see and analyse the relationships between words in a dataset or to use the vector representation of those words as input for other deep learning applications such as clustering, classification etc.

The focus of this report will be to explore the functionalities of Word2Vec, using its Python implementation called Gensim. We will try to see what results we get on the Harry Potter dataset with Word2Vec and later on try to visualise the dataset using the combination of Word2Vec with t-SNE.

2 Dataset

The dataset used is the text of all the Harry Potter books, 7 in total 1. I chose these books because I very much enjoy reading them and given so, I am very familiar with the storyline and character personalities - which will be useful in the analysis of the text.

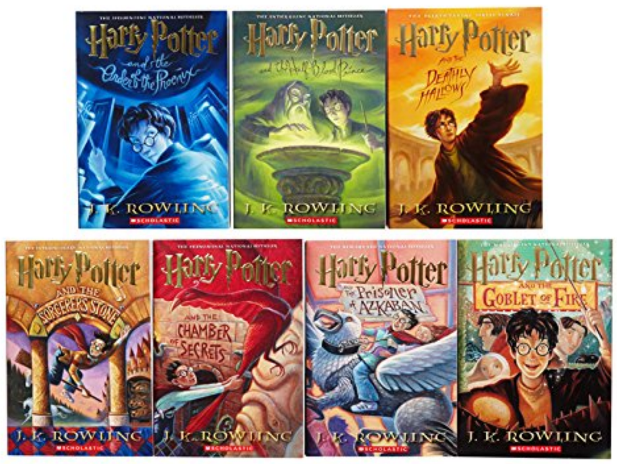

Fig. 1: Harry Potter books.

2.1 Preprocessing

The raw text of the books was found online. As input Gensim requires a list of sequential sentences, where a sentence is a list of words. For this, some basic preprocessing of the raw book data had to be done.

First, using a basic text reader (Notepad++), all the line breaks in all the .txt files containing the books were replaced with spaces - this way we have just the sentences without any gaps in the text which will be easier to process.

Next, even though there are many smarter ways to do so, for the sake of simplicity the text of the books was simply read from their files and appended to a variable.

The text was then split into sentences by dots, question and exclamation marks. Those sentences are then cleaned - split into words, only ASCII characters kept, all the letters converted into lowercase and english stopwords removed. Stopwords are commonly used words in a language, e.g. for English some of the stopwords would be: *the, a, I, it, what, that, but* etc. Those are the words that are used to form sentences, but do not carry any special meaning context-wise. The list of stopwords used in this report was taken from the link given in references [1]. The list is quite extensive, but so is our vocabulary - the 7 books contain over 1 million words in total. In our analysis we mostly want to see characters, magical creatures, magical objects etc., while commonly used verbs or adjectives will not have much meaning.

After these few steps, we have a structure that is ready to use with Gensim.

Just a brief comment about the process of preprocessing - this is by far not the best nor the most optimal way to do so, especially when reading all the text and appending it to an existing variable. However, all the preprocessing was possible to be done in my machine (a regular laptop) in a reasonable amount of time (a few minutes), so no further optimizations were done for the reading and preprocessing of the text. Of course, if the preprocessing had been taking more time and memory, some steps could have been taken - generators to preprocess .txt files, using the pandas framework to handle data, generators to use while training to save memory etc. Since I had neither time nor memory problems in my laptop, I decided to focus more on the Word2Vec functionalities rather then optimizing the preprocessing part while emphasizing that I am aware of the faults of my process.

2.2 Examples

In table 1 are some examples of sentences before and after preprocessing. From the table we can see that only the most representative words are kept, while the stopwords and interpunction characters are removed.

Table 1: Examples of sentences.

Raw Text	Cleaned Text
Maybe the house in Privet Drive would be so full of letters when they got back that he'd be able to steal one somehow.	['house', 'privet', 'drive', 'full', 'letters', 'steal']
Dotted here and there among the students, the ghosts shone misty silver.	['dotted', 'students', 'ghosts', 'shone', 'misty', 'silver']
The Hogwarts Express was streaking along below them like a scarlet snake.	['hogwarts', 'express', 'streaking', 'scarlet', 'snake']
Rons ears were scarlet; Hermione looked nervous.	['ron', 'ears', 'scarlet', 'hermione', 'looked', 'nervous']
The scar had not pained Harry for nineteen years.	['scar', 'pained', 'harry', 'nineteen', 'years']

2.3 Code structure

Code used in the experiments can be found at a GitHub repository [2].

The code is separated in two files:

1. **preprocess_train.py**
   This file loads and preprocesses the dataset as described in section 2.1. It builds the vocabulary, trains Word2Vec and saves the model to a file.
2. **analyse.py** This file loads the previously saved trained model and runs the various analysis on the model.
3. **visualise.py** This file loads the previously saved trained model and runs the visualisation of data.

3 Experiments

This section will describe the experiments performed on the Harry Potter dataset.

3.1 Training

Just to briefly describe the process of training, let's take a look at listing 1.

```
model = Word2Vec(
    size=300,
    window=5
    min_count=5,
    sample=0.001,
    negative=5,
    workers=cores-1)

model.build_vocab(sentences,
    progress_per=10000)

model.train(sentences,
    total_examples = w2v_model.corpus_count,
    epochs=50)
```

Listing 1: Training.

First, we build a model. Here we can define different parameters of our model:

size - dimensionality of the feature vectors.
window - maximum distance between the current and predicted word within a sentence - *window* words on the left and *window* words on the right of our current target word.
min_count - ignore all words with total frequency lower than this.
sample - threshold for configuring which higher-frequency words are randomly downsampled.
negative - negative sampling will be used, the int for negative specifies how many noise words should be drawn.
workers - use this many worker threads to train the model.

After that, we build a vocabulary using the preprocessed sentences and start the traning. As we can see, the process of training is quite straightforward!

3.2 Exploring the Word2Vec model

A trained Word2Vec model offers some interesting methods to explore 2.

```
print(len(model.wv.vocab))
print(model.wv['hogwarts'])
print(model.wv.index2entity[:10])
```

Listing 2: Word2Vec general queries.

At the beggining we can query the size of the vocabulary which in our case is **3357**.

For every word in the vocabulary we can get the vector representation of the word 2. Since in listing 1 we defined the size to be 300, the vector dimensionality will be 300.

Lastly, we can see the 10 most common words in the dataset - shown in table 2.

Fig. 2: Vector representation of the word 'Hogwarts'.

$$\begin{pmatrix} 0.0767 \\ 0.2209 \\ 0.0311 \\ \dots \\ -0.3623 \\ -0.2907 \\ 0.048 \end{pmatrix}$$

Table 2: 10 most frequent words in the dataset.

word	**frequency**	**word**	**frequency**
harry	18120	hagrid	2037
ron	6283	professor	2015
hermione	5317	well	1971
dumbledore	3344	snape	1819
looked	2337	time	1729

10 most common words are some of the characters - *Harry, Ron, Hermione, Dumbledore, Hagrid and Snape*. The word *professor* also fits well since a lot of the characters are professors in Hogwarts.

3.2.1 Similarity

Using the trained model, we can calculate the similarity between any two words.

```
model.wv.similarity("harry", "ron")
```

Listing 3: Similarity example

The way Word2Vec calculates similarity is using the *cosine similarity* measure between two vectors as shown in figure 3. Cosine similarity is a good measure of semantic similarity (better than e.g. Euclidean distance) often used in NLP purposes because it uses the angle between the vectors rather than the magnitude. For example, if we want to measure the similarity of the words "cats" and "dogs" in a corpus even when one of them occurs a lot more then the other, the cosine similarity would still capture this well since it does not depend on the magnitude of the vectors.

Some query examples are shown in the table below 3.

$$similarity(A,B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

Fig. 3: Cosine similarity

Table 3: Similariy

words	similarity
Harry - Ron	0.6303
muggle - wizard	0.4943
Fred - George	0.8388

3.2.2 Most similar words

Very similar to the last section, we can ask our model to find most similar words to a word - using the same measure as in the last section.. By default the method returns 10 most similar words, but we can also specify the number of words we want.

```
model.wv.most_similar(
    positive=['dumbledore'])

model.wv.most_similar(
    positive=[''voldemort], topn=5)
```

Listing 4: Similarity example

Some query examples are shown in tables below 4, 5, 6.

Table 4: Query for word: Dumbledore

word	similarity	word	similarity
headmaster	0.5607	witchcraft	0.4013
severus	0.4657	calmly	0.3964
wizardry	0.4097	connection	0.3838
voldemort	0.4078	scrimgeour	0.3788
albus	0.4076	ability	0.3752

Table 5: Query for word: Expelliarmus

word	similarity	word	similarity
stupefy	0.6763	cried	0.5417
shouted	0.6415	jets	0.5254
yelled	0.5904	stunning	0.5223
screamed	0.5691	dolohov	0.5085
masked	0.5433	amycus	0.5066

Table 6: Query for word: Hogwards

word	similarity	word	similarity
school	0.7745	pupil	0.6407
wizardry	0.7543	expel	0.6230
witchcraft	0.7366	behaviour	0.6142
appointed	0.6633	safer	0.6119
restriction	0.6500	safety	0.5999

A Harry Potter fan will see that these similarities make a lot of sense. For those not fluent in Harry Potter trivia, let's briefly explain.

Albus Dumbledore 4 is the *headmaster* of the Hogwarts School of *Witchcraft* and *Wizardry*. He is very connected to a school teacher called *Severus* Snape and his biggest enemy is Lord *Voldemort*.

Expelliarmus 5 is a magic spell often used in the story, as is *stupefy* (another magic spell). The spells are often *shouted*, *yelled*, *screamed* or *cried* in the heated battles along with *jets* of bright lights sparking from magic wands.

Hogwarts 6 is a *school* of *Witchcraft* and *Wizadry*. *Safety* is a big concern for the *pupils*, so certain *restrictions* are introduced to regulate their *behaviour*. If they do not follow the regulations, they could be *expelled*.

As we can see, the most similar words really make sense and are true to the story.

3.2.3 Odd one out

This feature can be used to detect a word in a list which does not match (or matches the least) with other given words 5.

```
model.wv.doesnt_match(['potter', 'weasley',
'malfoy', 'neville'])
```

Listing 5: Odd-one-out example

Some query examples are shown in the table below. The first row of the table states three negative char-

Table 7: Odd one out.

words	odd one out
Lucius, Harry, Bellatrix, Greyback	Harry
Ron, Harry, Snape, Hermione	Snape
expelliarmus, kedavra, magic, lumos	magic
dementor, werewolf, hermione, goblin	Hermione
snake, Crookshanks, Hedwig, Fang	snake

acters (Lucius, Bellatrix, Greyback) and Harry as a positive character.

The second row gives the trio of friends (Harry, Ron, Hermione) and the algorithm correctly excludes the teacher Snape from the friends.

The third row states three magical spells (expelliarmus, kedavra, lumos) and a general word - magic.

In the fourth row, Hermione is the odd one out from the list of magical creatures (dementor, werewolf, goblin).

The fifth row states names of pets in the books (Crookshanks the cat, Hedwig the owl and Fang the dog), while the snake as an animal is the odd one out.

This works quite well, but of course there are some examples in which the algorithm fails 8.

Table 8: Odd one out failures.

words	odd one out
pound, sickle, knut, galleon	galleon
Potter, Weasley, Malfoy, Neville	Potter

In the first row we have magical money (sickle, knut, galleon) and real money (pounds). The second row states three last names (Potter, Weasley, Malfoy) and a first name Neville.

Of course, it is reasonable to expect some failing instances - the machine can not handle context as well as humans, especially since the words do have a lot of similar meaning throughout the books.

3.2.4 Analogies

Word2Vec can also analyse analogies - what is to **z** as **x** is to **y**?

```
model.wv.most_similar(positive=["draco",
"harry"], negative=["ron"], topn=5)
```

Listing 6: Analogy example.

Some examples are shown in the tables below 9 10.

Table 9: What (or who) is to Draco as Harry is to Ron?

word	certainty
lucius	0.5610
narcissa	0.4613
malfoy	0.4516
sneering	0.3868
severus	0.3668

Even though the results in table 9 are not not exactly right, as Draco Malfoys closest friends in the books are Crabbe and Goyle, the results are not at all bad - Lucius and Narcissa are Draco's parents, Malfoy is his surname and Severus is his closest teachers name.

Table 10: What (or who) is to Arthur as Vernon is to Petunia?

word	certainty
molly	0.4317
works	0.3923
wife	0.3856
sneered	0.3812
eh	0.3765

As for the results in table 10, Vernon and Petunia are a married couple and Harry's aunt and uncle. The first result is Molly, which is actually Arthurs wife (Ron Weasley's parents). Success! Among other results, *wife* also makes sense, while the others (*eh?*) are not as fitting.

All in all, this is an interesting feature of Word2Vec which gives us the possibility to explore relationship among words in a text, draw some conclusions etc.

3.2.5 Visualisation

When working with large amounts of data it is always interesting to visualise. Even though this is not a direct feature of the Word2Vec, using the vector representations of words it is easy to visualise the dataset and relationships among words and with that get a better insight of our data.

The problem in visualising these kinds of problems is the large dimensionality of data. Below we have an example of a vector representation of the word ’harry’ 4. Obviously, this is ony a small part of the vector since the dimensionality of the vector representations was defined to be 300 (as can be seen in listing 1).

Fig. 4: Vector representation of the word ’Harry’.

$$\begin{pmatrix} 0.1586 \\ 0.1142 \\ 0.1752 \\ \ldots \\ 0.1196 \\ 0.4320 \\ 0.1936 \end{pmatrix}$$

The question is - how to visualise vectors of such lengths? For this, we can use a dimensionality reduction technique called t-Distributed Stohastic Neighbour Embedding (t-SNE). The original paper [3] describer the t-SNE as:

t-Distributed stochastic neighbor embedding (t-SNE) minimizes the divergence between two distributions: a distribution that measures pairwise similarities of the input objects and a distribution that measures pairwise similarities of the corresponding low-dimensional points in the embedding.

The idea is essentialy for the algorithm to find the best way of representing the input data with less dimensions by matching both distributions. This can be quite a computationally heavy task and in case of very high dimensional data we may need to apply another dimensionality reduction techniques (e.g. PCA for dense data or TruncatedSVD for sparse data) before using t-SNE.

Using some of the previously mentioned functionalities of Word2Vec combined with the power given to us by t-SNE, let’s try to visualise our data.

First, we can visualise 10 most similar words to the word **Weasley** 5. To understand the figure, the Weasleys are a family consisting of the parents Arthur and Molly (not on the graph) and the children: Ron, Ginny, Charlie, Percy, Bill, George and Fred. Tonks is also a character who is a very close friend to the family and later on marries Bill Weasley. We can see that this visualisation really nicely presents the (almost whole) Weasley family.

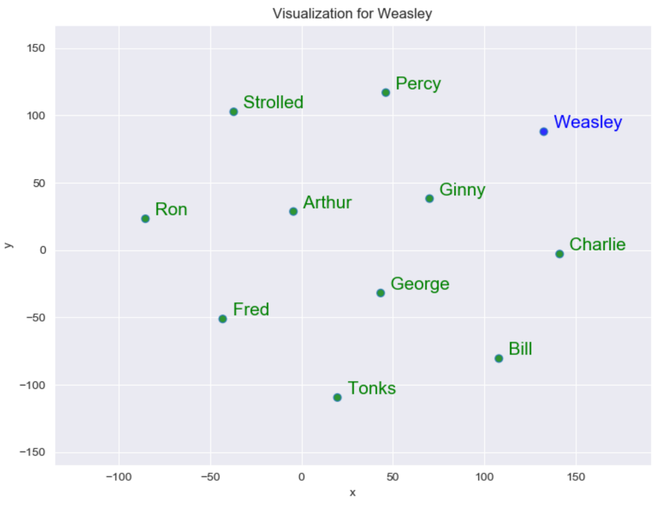

Fig. 5: Weasley

Along with 10 most similar words, we can plot 10 least similar words. Example below is with word **Ginny** 6, another character in the books. The words marked with green dots are most similar, while the red ones are least similar.

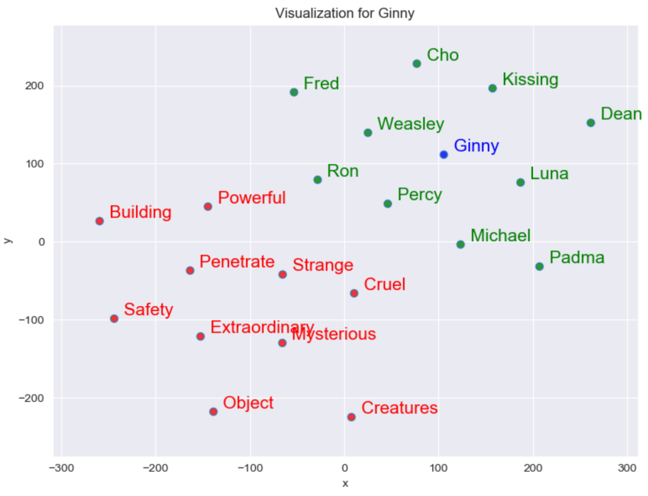

Fig. 6: Ginny

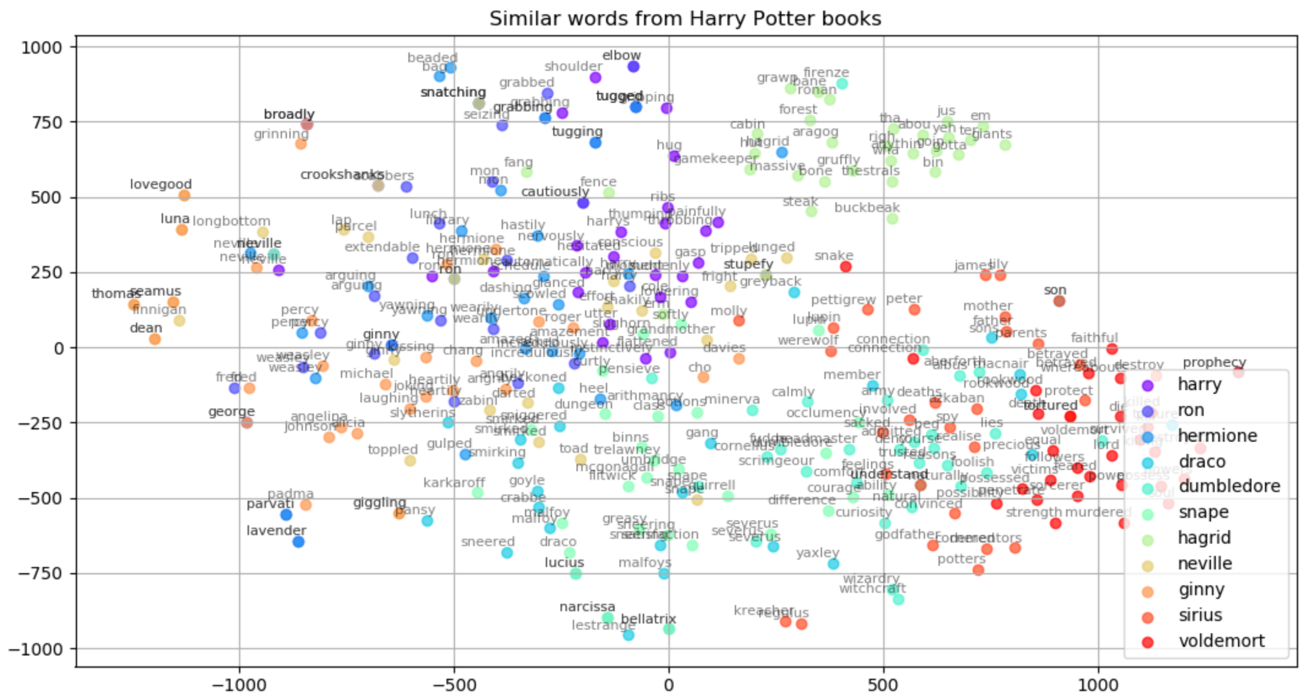

Fig. 7: 30 most similar words to the main characters

Words closest to the word "Ginny" are the names of her brothers (Ron, Fred, Percy) and friends (Luna, Padma, Cho, Michael, Dean). We see that the least-similar words are further from the most-similar cluster.

Next, let's look at the bigger picture and visualise 30 most similar words to the main characters in the books 7.

From this "big picture" we can not draw many smart conclusions since all the characters are very intertwined in the books, but a few details can be seen - Harry, Ron and Hermione as the main characters share a lot of similar words and their words are very close on the graph. Hagrid as a character is more separated from the others since the most contact he has is with the trio (Harry, Ron, Hermione) - we can see that his words are close to theirs, but still separate. The words from the evil Lord Voldemort are more to the edge of the graph, not mixing with the words from characters Ginny and Neville who do not have as much contact with him as Harry or Dumbledore (closer on the graph).

Let's also look at different locations mentioned in the books 8.

From the locations graph 8, we can see that the Durmstrang school is most separate from the others which is logical since it is mentioned only in a separate context during the fourth book when the students from the school come to Hogwarts for a tournament. On the upper left corner we see a mixture of words similar to Hogsmeade (a magical village near Hogwarts) and Burrow (house of family Weasley) - a good part of the story happens in the two locations, but it makes sense for them to be separate from the others since it is mostly an isolated part of the plot. Words that are most dense are words similar to Hogwarts, Ministry (of Magic), Gringotts (magical bank), Azkaban (wizard prison).

If we take a look at some individual words we can see some interesting facts:

- **Burrow** is associated with cars, shopping, London, road - often the characters stayed in the Burrow before the beggining of the school year.
- **Durmstrang** - mystery, Krum (student), Karkaroff (principal), champion, task, Beauxbatons (another magical school) - words linked to the tournament.
- **Ministry** - underage, education, official, artifacts, Sturgis and Podmore (employees).
- **Gringotts** - safest, haunted, goblins (magical creatures working in the bank), Griphook (goblin working in the bank), stolen.

As a Harry Potter fan I find it very interesting to look at these 'big picture' graphs as there are a lot of small details linking the words together and the more you look at the graph the more information it gives.

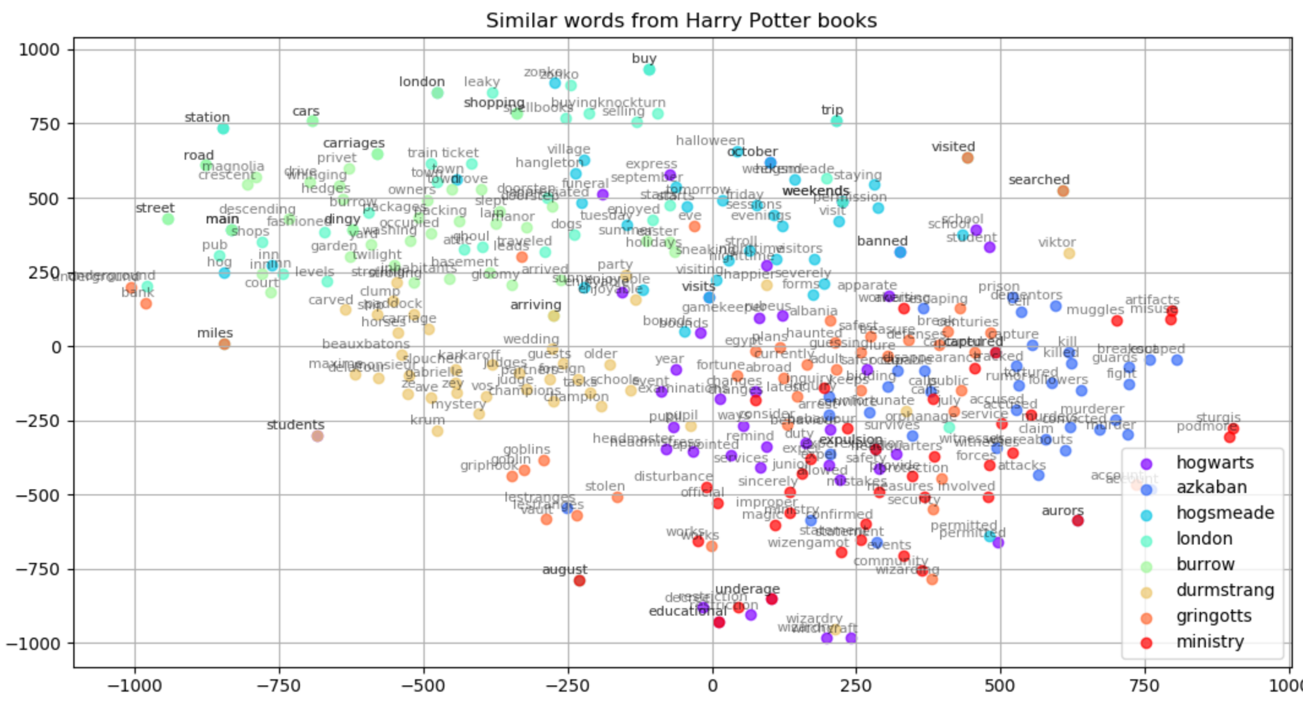

Fig. 8: 30 most similar words to the main locations

4 Extensions

The focus in this report was more on exploring how Word2Vec works and different functionalities it offers, but the true power it has can best be seen when using it in other deep learning applications.

Since Word2Vec is an embedding method, it can be used in various Natural Language Processing applications:

– machine translation
– automatic summarization
– speech recognition
– question answering
– sentiment analysis

Another thing that would be interesting to look at is how different training parameters have an effect on our results:

– bigger/smaller window
– less/more training epochs
– bigger/smaller word vector dimensionality
– bigger/smaller min_count
– etc.

5 Conclusion

This report gave a brief overview of the Word2Vec word embedding method using the Gensim python binding, using the text of Harry Potter books as a dataset.

Word2Vec was used to calculate most frequent words, similarities between words, most similar words to a given word, finding the odd-one-out from a list of words or calculating analogies. It was also shown how a dataset can be visualized using the combination of Word2Vec and t-SNE.

All in all, it was very fun and interesting applying Word2Vec to the Harry Potter books and seing what the math says about the relationships in the books.

References

[1] Stopwords, `https://www.ranks.nl/stopwords`
[2] Code repository on Github, `https://github.com/mirnabaksa/Deep-Learning`
[3] t-SNE original paper, `http://jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf`