**Recommendation System for Wikipedia Articles**

**Raksha Kaverappa**

Northeastern University

[Kaverappa.r@husky.neu.edu](mailto:Kaverappa.r@husky.neu.edu)

***Abstract***

This project aims at addressing the solution for entity recommendation of Wikipedia articles. Modern recommendation Systems use a combination of collaborative filtering techniques and content filtering techniques to recommend items based on user preferences. We completely focus our problem on building recommendations based on content analysis. This method we use is called Neural network embedding. We used supervised learning techniques to train the model to place similar items close to one another in an embedding space. We find this method to produce promising results and recommend similar articles based on search query.

**Introduction**

The popularity of Wikipedia articles has sky rocketed since its creation. With over 5 million English language articles present on Wikipedia, this poses a great opportunity to work with this large unstructured database for research and exploration. We use this database to build a recommendation system for the Wikipedia articles. We formulate our problem as a non-empty set of articles Li are extracted in any order. Our challenge here is generate similarity between these articles that generalizes to allowing our recommendation engine to accurately suggest articles based on the query article.

The added challenge here is working with a massive dataset comprising of over 5 million articles.

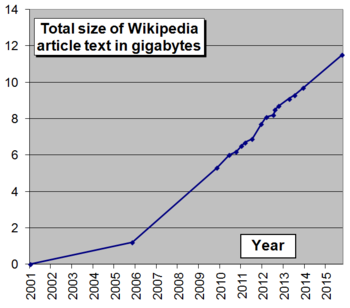
The scope of this project is limited to recommending books using all the Wikipedia articles on books. Our recommendation system is built on the idea that books which link to one another must be placed closer to each other in the embedding space. We will be using Neural Network embedding to carry out this process. We

represent this similarity on the embedding space using TSNE module in python. We learn how to use supervised learning and deep learning techniques to build an efficient Recommendation Engine for Wikipedia articles.

**Preliminary steps:**

***Data Preprocessing***

The size of Wikipedia articles has only exponentially increased since its creation. To build any machine learning model for such a massive amount of data, we must first identify what information we need and what we would like to extract.



As our focus is to just extract articles based on books, we will do that by using some of the functionalities which Wikipedia articles possess.

The first step is to retrieve the data from the wiki dumps. We do that using BeautifulSoup library in python. We decide not to make individual requests to each of the Wikipedia pages because for 2 reasons:

1. The process is time consuming and the data would require a lot of formatting and preprocessing
2. Since the size of the data is too large, we would very quickly run into data rate limits and we would be taxing the wiki servers!

Therefore, we access the Wikipedia dumps through Wikimedia (A dump is a periodic snapshot to a database). We can view all the available versions and chose the one we want to proceed with.

We parse through the dump folder and download the partitioned page articles. We do this so that we can extract the contents from each folder in parallel. The files downloaded comprise of a total of 15 GB of compressed data. We get a total of 60 GB of data when uncompressed. Since that would require a lot of computing power and disk space, we will work with the compressed partitioned files to extract the content we need.

An interesting feature about Wikipedia articles is that each of them comprise of what is called an “Infobox” and they look something like this:



An infobox is Wikipedia is consistently formatted table which is present in articles with a common subject. So, if we are looking for information about a person, their infobox will be under Infobox: Person. Since we are only interested in books, we will query the articles with a condition that they have Infobox: Book and we will extract all the information we need from these book articles. Since this consists of a total file size of 15 GB, we will run the process for each file in parallel so that the computing time is reduced.

The processing time for each file individually took about 5 minutes per 100 MB file. Therefore, running it linearly would take days to process as we have 55 partitions each of size 350 MB approximately. Therefore, we use parallel processing extract out information. After about 15 hours and 3 processors running parallelly, I had successfully extracted the information needed into 55 JSON files and the total file size was 50 MB. That’s a dramatic decrease in the size!

We merge all the JSON files into one single JSON and use this as our input dataset for the neural network embedding.

**Word Embedding**

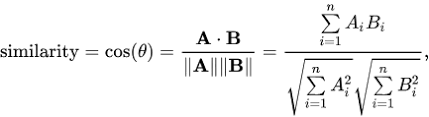
Before we dwell into Neural networks and Model building, let us first understand what word embedding is and why it is important.

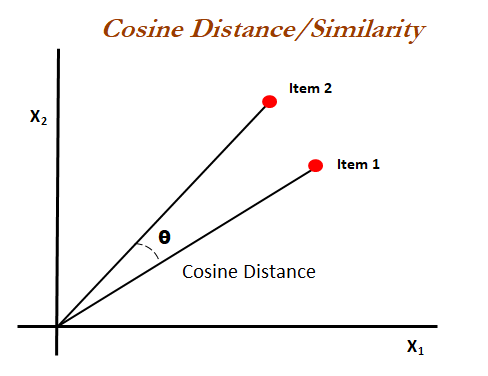
Word embedding is one of the most popular representations of document vocabulary. It is capable of capturing context of a word in a document, semantic and syntactic similarity, relation with other words, etc. They are basically vector representations of a word. But the question arises when we use the term word embedding: How are they generated? And how do they capture the context?

We all know that one way of representing categorical data is using One-hot encoding. One hot encoding has many drawbacks. They result in extremely high dimensional data when the number of categories is very high. They also don’t place similar words closer to one another. In other words, “good” and “great” is as different from each other as “good” and “bad”.

Our objective is to have words with similar context close to one another in spatial positions. To achieve this, we use Word2Vec embeddings

which is one of the most popular techniques used for word embeddings. They use shallow neural networks to do so. We use cosine similarity to measure the distance between two such words. Cosine is 1 when the two vectors are closest to each other i.e., the angle is 0.





In this project, we have used Neural network embedding create word to vector representations of the Wikipedia book articles. In the context of neural networks, embeddings are low dimensional continuous vector representations of discrete variables. They reduce the dimensionality of the categorical variables and meaningfully represent categories in the transformed space.

For our book recommendation system, we will represent each of the 37000 books using a 50-dimensional vector. Since embeddings are learned, books that are closer in context to one another are placed closer together in the embedding space.

**Neural network embedding: *Implementation***

As mentioned before, the book recommendation system is a supervised learning task where in we predict if a given link appears in a page or not. To achieve this, we generate a training set of book titles and links. We assign each pair a label as 1 or -1. We assign the label as 1 if the link appears in the article and we assign it as -1 otherwise. We incrementally generate this training input which is vectorized and fed to the model.

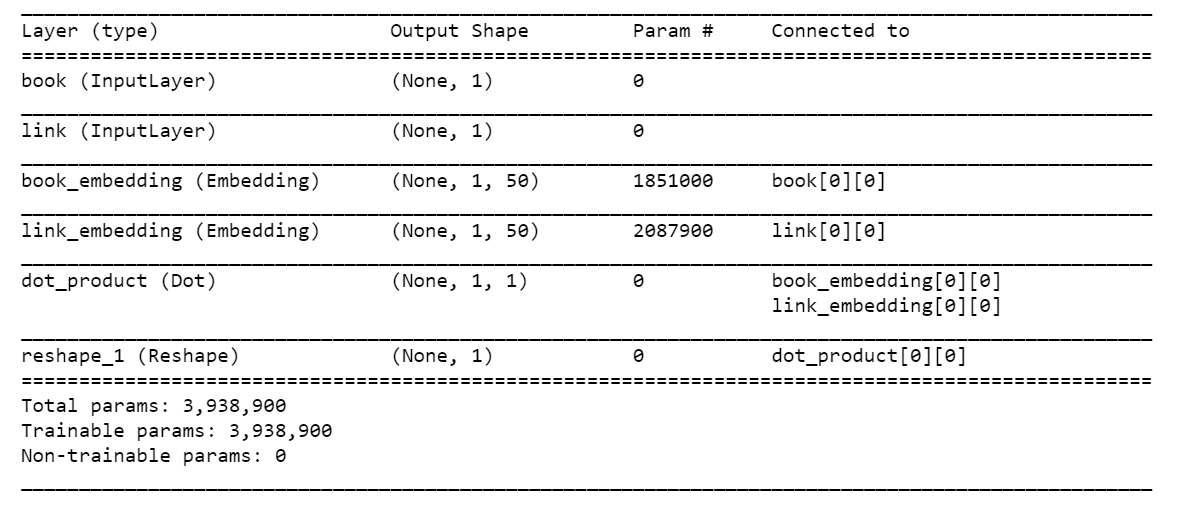
In our network, we use two parallel 50-dimensional layers for book titles and links. We then have a dot product layer that combines the previous two layers for prediction. These embeddings are adjusted during training to minimize the loss. We use Keras to train the model and build embedding layers. We extract the embedding weights which is a representation of books and links as continuous vectors. To compute similarity, we take a query book and find the dot product between its vector and those of all the other books. If our embeddings are normalized, this dot product is the [cosine distance](http://blog.christianperone.com/2013/09/machine-learning-cosine-similarity-for-vector-space-models-part-iii/) between vectors that ranges from -1, most dissimilar, to +1, most similar

***Regression***

In regression, our labels for (book, link) pairs are either -1 or 1. We use mean squared error as the loss function in order to minimize the distance between the prediction and the output. Using the dot product with normalization means that the Dot layer is finding the cosine similarity between the embedding for the book and the link. Using this method for combining the embeddings means we are trying to make the network learn similar embeddings for books that link to similar pages.

***Classification***

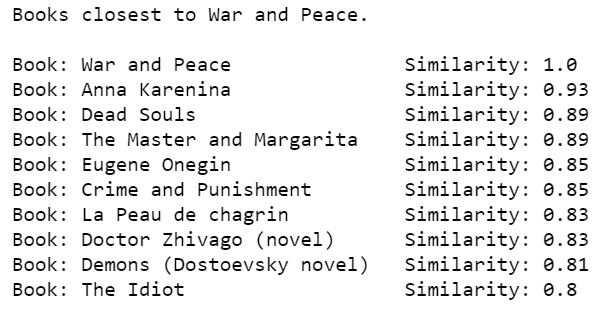
In terms of classification, our labels for (book, link) pairs are either 1 or 0. We also add an extra Dense layer with a sigmoid activation function to squash the outputs between 1 and 0. We use Binary cross entropy as the loss function which measures the error in neural network binary classification problem and thus measure the similarity between two distributions.

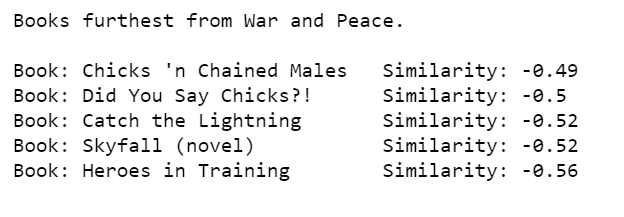


**Results**

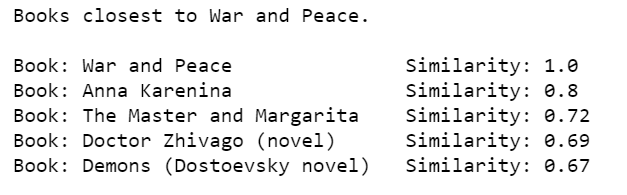
I found both regression and classification to produce similar results.

In case of regression, we found that the most similar books/articles were grouped together in the embedding space.

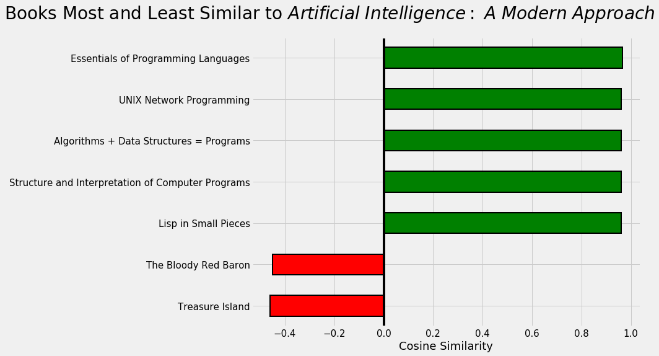




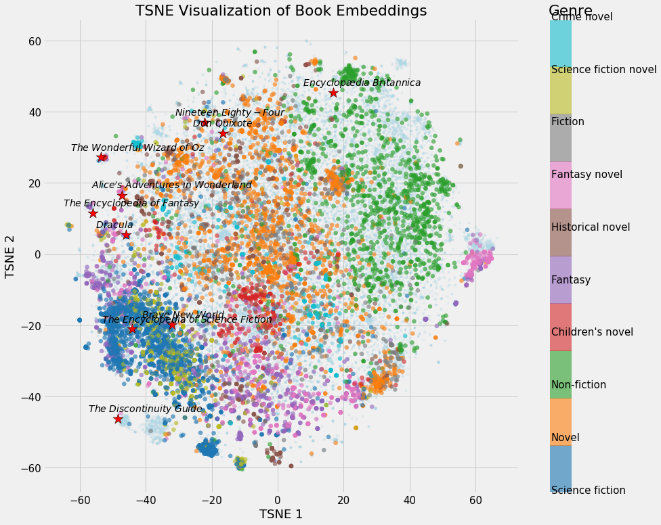
In case of classification, we produced similar results:



We could also plot these results to visualize them:



I used the TSNE module to map all the embeddings in a 2-dimensional space and visualized the clusters as shown below:



**Conclusion**

As we can clearly see from the results, we have successfully explored an efficient way to parse through wiki articles using multiprocessing and process them in such a way to place similar articles close to one another. We have efficiently scaled a massive dataset to extract the information we need for the recommendation system. We explored Neural network embeddings to obtain the cosine similarity between the articles. We can now extend the scope of the project beyond just books.

**Future work**

My initial idea behind this project was to build a graph database for these articles and connect them through the wiki links that are present in the Wikipedia page. The article with the most common wiki links with the query article would be recommended. The idea was to build on this and on the concept of hierarchical clustering to obtain the most similar articles. Wikipedia also provides a clickstream dataset which provides the count of clicks on an article. We could use this with a combination of wiki dumps to provide most similar articles based on the query articles and also take into account the popularity of the article.

**References**

[1] <https://towardsdatascience.com/building-a-recommendation-system-using-neural-network-embeddings-1ef92e5c80c9>

[2] <https://towardsdatascience.com/wikipedia-data-science-working-with-the-worlds-largest-encyclopedia-c08efbac5f5c>

[3] John Rothfels, Brennan Saeta, Emin Topalovic. A recommendation engine for Wikipedia articles based on

constrained training data.

[4] Kulkarni, Swapna, "A Recommendation Engine Using Apache Spark" (2015). Master's Projects. 456.

**Scope**

This project was carried out individually. As explained above, I have explored various approaches to solve the problem of building a recommendation system for Wikipedia. The biggest challenge involved in this project was to retrieve the data from the Wikipedia dumps and to handle the enormous amount of data (60 GB uncompressed). After a lot of research, I found that multiprocessing in python would be the best way to go about this problem. After the input to the model was scaled down to just a few MBs, the processing and model building was easier and faster.

**Context**

The idea behind Neural network embeddings was referenced using the links provided above. Beyond the scope of those links, I have further explored various other parameters that can be used to optimize the model to get better results. I changed the loss function, optimizer and the number of dimensions for a given vector. I also tuned the model for a greater number of epochs and found that 50 epochs were giving optimum results.

To extend the scope of this project, I would be working on building graph databases using Neo4j or Graph Dataframe in spark and build a recommendation on top of that.