UCLouvain-EPL

Algorithms in Data Science

Project 2: A Complete Study on Anna Karenina, and Biden-Trump fans' tweets

Teachers:
Vincent Blondel
Jean-Charles Delvenne
Krings Gautier
Course assistant:
Alexey Medvedev

TEAM Members :
Nima Farnoodian - 68372000 - nima.farnoodian@student.uclouvain.be
Charles Rongione - 51841500 - charles.rongione@student.uclouvain.be
Breno Tiburcio - 74042000 - breno.tiburcio@student.uclouvain.be

Abstract

Following our previous project in which we studied the network of Count Leo Tolstoy's novel, Anna Karenina, in this project, we broaden our study on this network by weighing its links with the number of occurrences of their endpoints (Actors) in the paragraphs. We also utilize a node embedding approach called Nod2Vec to embed the nodes of the network in an n-dimensional space. Using this embedding technique and k-means clustering algorithm, we could find a set of clusters resembling the community structure detected in our network by the Louvain Algorithm. The best and worst Jaccard similarity we found are 0.808 and 0.387, respectively.

As the second part of our project, we could train three classifiers that represent reasonable accuracy on the test set. Using the Doc2Vec method, we could train multi-layer neural networks with a 70 percent accuracy on the test set. Also, as the Doc2Vec benefits from a feature for documents, namely tag, we could use Doc2Vec solely to obtain a simple classifier that borrows its concept from Cosine Similarity. This simple approach could achieve a 69.8 percent accuracy on the test set. In the end, we used BERT framework for our classification task, and we could achieve a good accuracy of 84 percent on the test set. It's not an exaggeration to say that BERT has significantly altered the NLP landscape.

1 Anna Karenina: Complete study on Community Structure

In the previous project, we studied Anna Karenina, a novel by the Russian author Count Leo Tolstoy. We created a co-occurrence (unweighted) network such that the characters are linked if they are mentioned in the same paragraph at least once. Here, in this section, we aim to embed the nodes of the network into an n-dimensional space and then find the k clusters using the well-known k-means clustering algorithm in order to compare the detected clusters in the n-dimensional space to the communities detected using the Louvain algorithm. To this end, we first created a weighted version of our network with the hope of a better representation of the community structure of our network for comparison. To build the weighted network, we followed the same approach as we discussed in the previous report, but we just added the links' weights that indicate the number of occurrences of the two characters in the paragraphs. After weighting, the structure of the network did not change significantly. For instance, the degree assortativity increases to 0.234 (only 0.013 difference), Modularity for the detected community set is slightly improved (0.057)— Figure 1. illustrates our weighted network with detected communities, the gender-based assortativity is negligibly impacted, and the list of six high-degree actors remain unchanged as they are the major characters who the story revolves around. Table 1. shows a summary of the network characteristics.

	Value	Description
Avg. (weighted) Degree	285.909	-
Density (unweighted)	0.305	Relatively High
Degree Assortativity (weighted)	-0.234	The appearance of the significant and marginal characters together
Avg. (weighted) Clustering	0.032	-
Diameter (unweighted)	3	Low
Avg. Path Length (unweighted)	1.779	Low
Gender-Assortativity (weighted)	0.025	The network is randomly mixed based on the gender of the actors
Q for Community set (weighted)	0.248	Three communities observed using the Louvain Alg.
Best Jaccard score (NodeEmbedding)	0.808	Found by accident; best score found was around 0.775 on average

Table 1: Network Characteristics

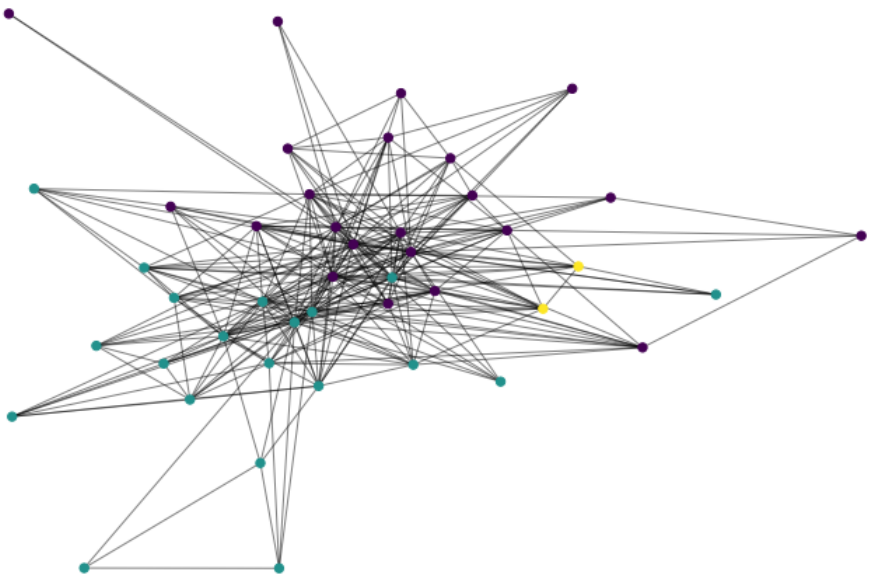

Figure 1: Detected Communities in our weighted network. Each color represents a community.

In what follows, we discuss node embedding using Node2Vec[1] and then compare the obtained clusters with our ground-truth community structure provided by the Louvain Algorithm. To measure the similarity between the two sets of detected clusters, we use Jaccard Similarity and Normalized Mutual Information (NMI) score.

1.1 Node Embedding: Node2Vec

Node2Vec is a framework for learning continuous feature representations for nodes/actors in networks. In this framework, the nodes are mapped into an n-dimensional space of features such that the probability of preserving the network neighborhood of nodes is maximized. In this model, random walks starting at random nodes are performed, and then the steps going to the neighboring nodes finally forms a set of sequences that are deemed as the sentences of a corpus. In the end, the sentences are given to word2vector with n-dimension to obtain the vector representation of the nodes in an n-dimension space. To get such node vectors using this framework, we utilized a python package named Node2Vec that is compatible with Gensim and can be simply tuned by several parameters. The significant parameters for this API are dimension, walk-length, num-walk, window, min-count, and Batch-words. Based on these parameters, different vector representations of nodes for a network is achieved; therefore, as the goal of this task of our project is to find two clustering sets, one of which should be close to our ground-truth and another must be far from it, we should train the Node2Vec model with different parameters to realize these two sets.

[1] A. Grover, J. Leskovec, node2vec: Scalable Feature Learning for Networks, 2016

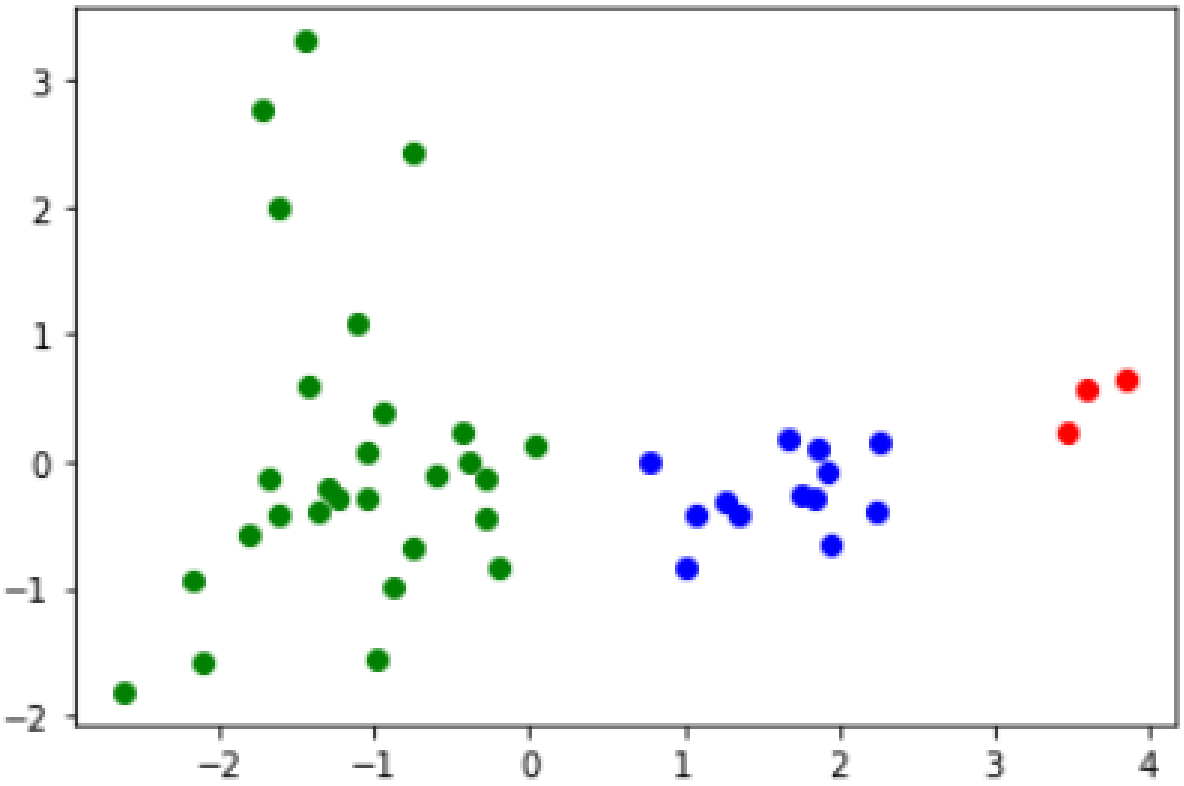

Figure 2: Clusters Detected using k-means, k=3. The figure shows the clusters that could give the highest Jaccard Similarity score.

We first set the parameters of the model as dimension=128, walk-length=30, num-walks=200, window=15, and batch-words=18. Combining this trained model with k-means algorithms with cosine distance accidentally gave us a Jaccard score of 0.808 and NMI of 0.77. Although the result was obtained by chance, this is the best possible score that we could obtain. We ran a GridSearch with different combinations of parameters, but after even 1024 iterations, we could not achieve a Jaccard score exceeding 0.808. Indeed, our GridSearch gave us a Jaccard score of around 0.776 and NMI of 0.714 as the best scores and a Jaccard score of 0.387 and NMI of 0.433 as the worst scores. As in Fiqure 3, we can see that the Jaccard score never become higher than 0.776 and never fell below 0.387. More interestingly, these scores were all found in the first iterations of GridSearch. A point behind these results might be the randomness of k-means clustering. K-means at each time may give different outputs as we observed; therefore, we think that the score of 0.808 was achieved due to the fact k-means could detect the clusters that can more resemble our ground-truth by a specific, but rare, initial points. The detected parameters for our worst model are dimension=128, walk-length=20, num-walks=50, window=40, and batch-words=2. Figure 4 and Figure 5 illustrate the best model clusters (with Jaccard of 0.808) and the worst model clusters. It is worth noting, to plot the clusters, we first used the PCA algorithm to reduce the dimensionality to two and then applied k-means clusters with Euclidean distance for visualization.

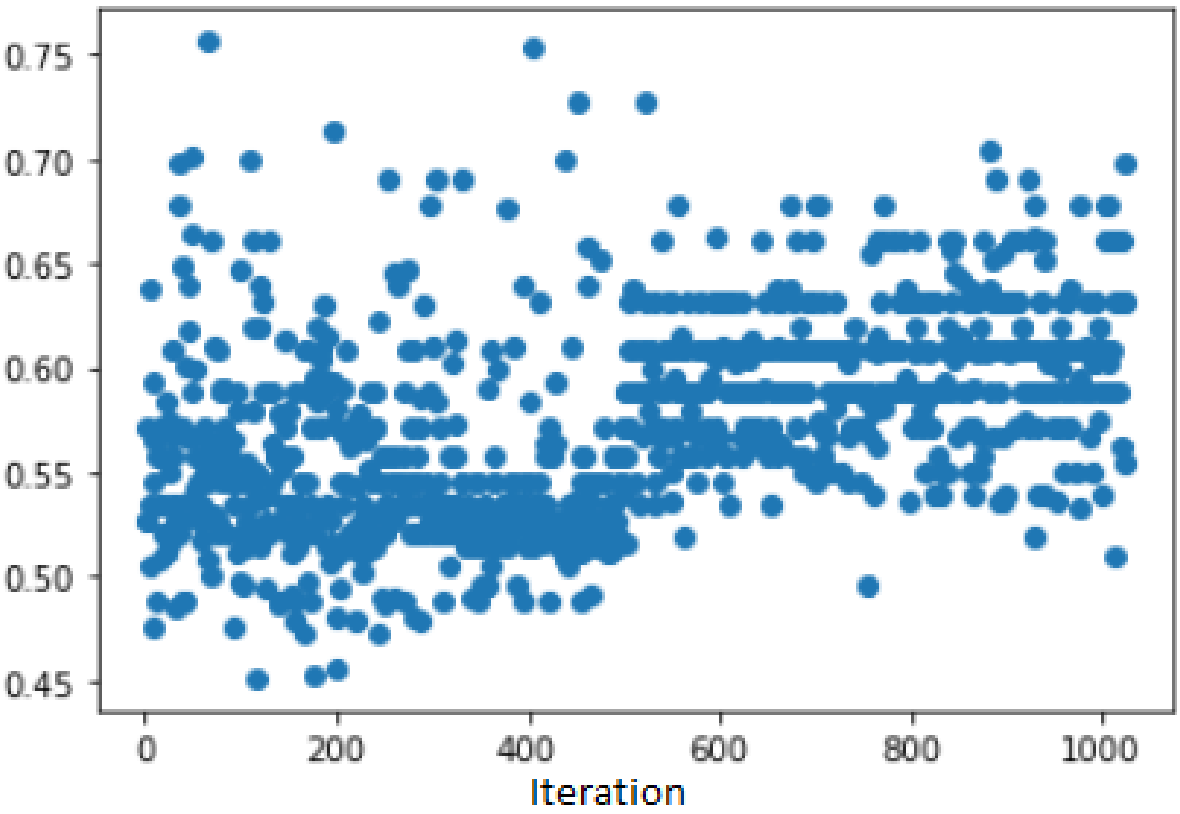

Figure 3: Jaccard Similarity Score for 1024 iterations

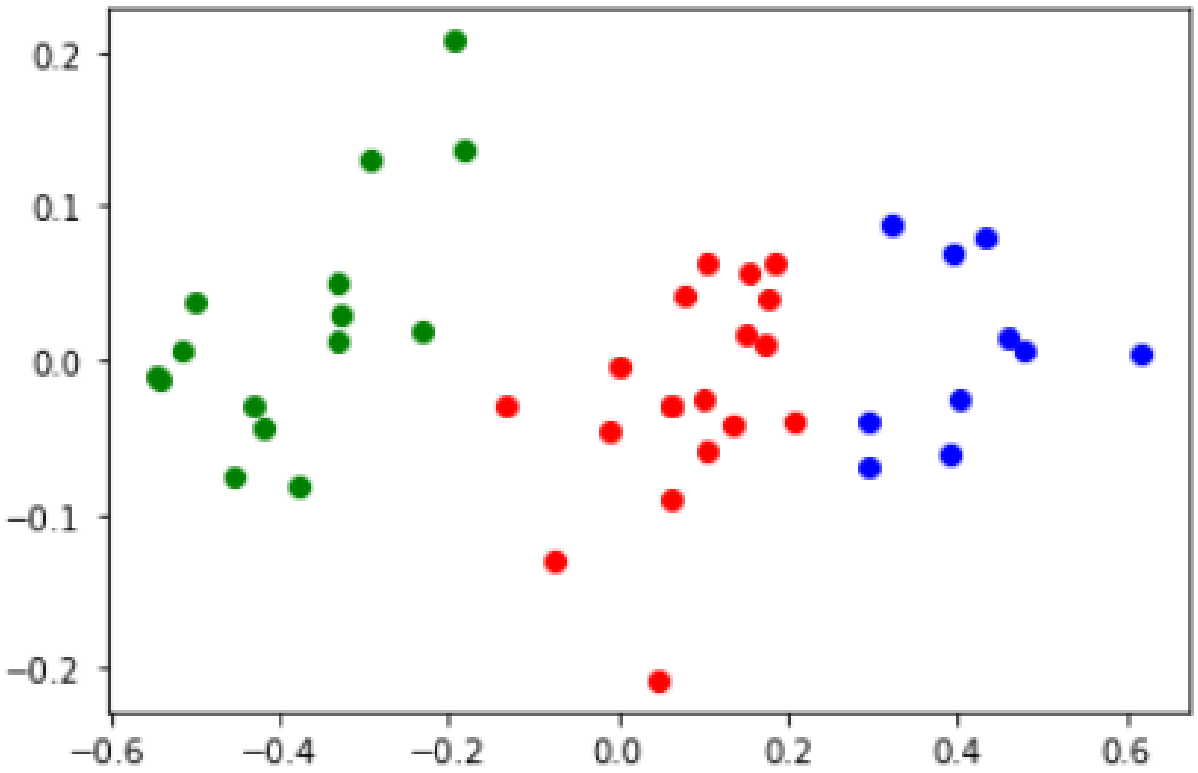

Figure 4: Clusters Detected using k-means, k=3. The figure shows the clusters that could give the worst Jaccard Similarity score. The parameters are dimension=128, walk-length=20, num-walks=50, window=40, and batch-words=2

As final words, we should mention that the network is inherently inseparable because we mainly focused on the major characters of the books while creating the networks, so only a few marginal characters were added to the network. It is trivial that the major characters are frequently mentioned in the paragraph to vitalize the whole story, and this is why we observe a large average degree of 285.909 for a network of 44 nodes. The characters are highly connected, and they resist the separation. Interestingly, the modularity scores for the best embedding model and the worst embedding model are 0.24 and 0.202, which means no significant difference between these two models. However, their outputs are entirely different in terms of community structure.

2 Part2: Text embedding and classification

In this section, we represent our results for the text classification task for President-elect Joe Biden and President Donald Trump fans' tweets. The goal here is to train a good model to guess "in favor of whom a comment or tweet is about". For obtaining vectors of documents and words, we made use of Doc2Vec model and also BERT framework. In what follows, we will explain the approaches above and the Machine Learning algorithms we utilized to achieve this classification task with reasonable accuracy.

2.1 Doc2Vector

Before training the Doc2Vec model, we first cleaned our data sets by removing stop-words, punctuations, Hashtags, URLs, and so on. After clean both the training set and test set, we utilized Doc2Vec of Gensim and trained a model to convert the sentences/tweets of the training set to the vectors with a size of 200. Our learning parameters are window=3, min-count=10, and alpha=0.01. Based on the trained model and the vocabulary built using it, we then obtained the Document vectors of the test set sentences for evaluating our classifiers. In the end, the converted training set was fed into different machin learning classification algorithms to find the model with the best accuracy.

We first used SckitLearn and trained the model with different classifiers, including Decision Tree, Support Vector Machine, Random Forest, Logistic Regression, and even Multi-layer perceptrons. Almost all algorithms either suffered from overfitting or did not act well on the test sets— accuracy less than 0.65. For example, Random Forest could achieve an accuracy of 0.999 on the training set, but it acted poorly on the test set with an accuracy score of 0.595. Even GridSearch with many possibilities and parameters could not provide better accuracy on the test set for neither Random Forest nor Multi-layer perceptrons. Therefore, we came up with using the Neural Network of Keras and using Cosin similarity (as a simple approach, but still useful) for gaining better accuracy on the test data sets. Notice that the Jupyter Notebook related to this section (Doc2Vec) is Biden-Trump w2v.ipynb.

2.1.1 Keras: Neural Network

We used Keras to make a neural network classifier due to it's user friendly interface and fast prototyping. After several experiments on Hyper parameters: Learning rate, loss function, size and number of hidden layers, number of epochs, batchsize and activation functions, the best results were provided by a network with two hidden layers of size 10 and 2, one Relu and two Sigmoïd activation functions and the best loss function was the LogCosh. The best learning rate was 0.005, and the optimum number of epochs and batchsize were 8 and 5 respectively. The accuracy we managed to reach was 70 percent on the test data set.

2.1.2 Cosine Similarity

In training a Doc2Vec model, each sentence can have a tag. In many cases, a random tag is assigned to the sentence. To have a classification model, while training the DocVec, we chose the label of the sentence as its tag. Therefore, a sentence has either tag=1 (for Biden) or tag=0 (for Trump). After training the model, we have the vectors of the sentences and the associated tags that clarify their labels. To predict the label for a new sentence or comment, we just need to compute the new sentence's similarity with the sentences with tag=1 and tag=0. Then, the tag that shows the highest similarity is considered as the label of the new sentence. Thanks to Doc2Vec, this operation is straightforward and can be done only by a few code lines as below. Using this simple strategy, we could achive an accuracy of 69.8 percent on the test set, indicating its good performance compared to the accuracy of the Keras Neural Networks that is 70 percent. The model is simple yet as practical as the Keras Neural Networks.

Given that gmodel is a trained Doc2Vec model, the label can be predicted by the following function:

```
def predict_Similar(gmodel, twitt):
    tokens = twitt.split()

    new_vector = gmodel.infer_vector(tokens)
    sims = gmodel.docvecs.most_similar([new_vector])
```

```
return sims[0][0]
```

2.2 BERT

The treated comments were passed through a Tolkenizer (built on top of Keras Layers and a Vocabulary File). The latter is a map built-in an intelligent fashion correlating words' semantic thoroughly into numbers.

After the tokenizer, each word assumes a unique number. Now, these numerical vectors, which were sentences of words, are ready to fit a model.

Ultimately, we format the tokenized text into data set divided into batched that set the paddings of our future model.

2.3 Our Classifier

Before building the training and the testing set, we added a column with a binary variable to differ with each candidate. After that, we merged the files designated for training and testing, respectively.

We opted for a Convolution Neural Model for our comments classification. These types of models are more commonly used to image classification, but we decided to give a try due to its exciting recognition features.

It consists of three hidden layers, each one with a different number of kernels. After each Kernel scan, a MaxPooling is performed to reduces the data variance.

We have performed a different approach to reach the best possible accuracy. From the data perspective, we used NLKT package to clean stop-words. Although the accuracy has not significantly improved, the data-set became lighter, enabling us with faster training.

From the model parameters, we adjusted the number of batches for each model padding (calibration) as well as model parameters as Drop-out rate, Embedding-Dimension, number of filters, and dense layer size. It's important to mention that the "binary cross-entropy" loss function presented slightly more superior performance than the log cosh loss function.

Also, the DCNN model code is pretty straight-forward and can be easily interpreted from our "BERT.ipynb". The ultimate parameters that we set could achieve an accuracy of 84 percent on the test data set.