ID2221 Project Report

Drug Interaction Prediction using Spark and Graph ML Techniques

Blanca Bastardés Climent, Amrita Panesar, Group 13

October 30, 2020

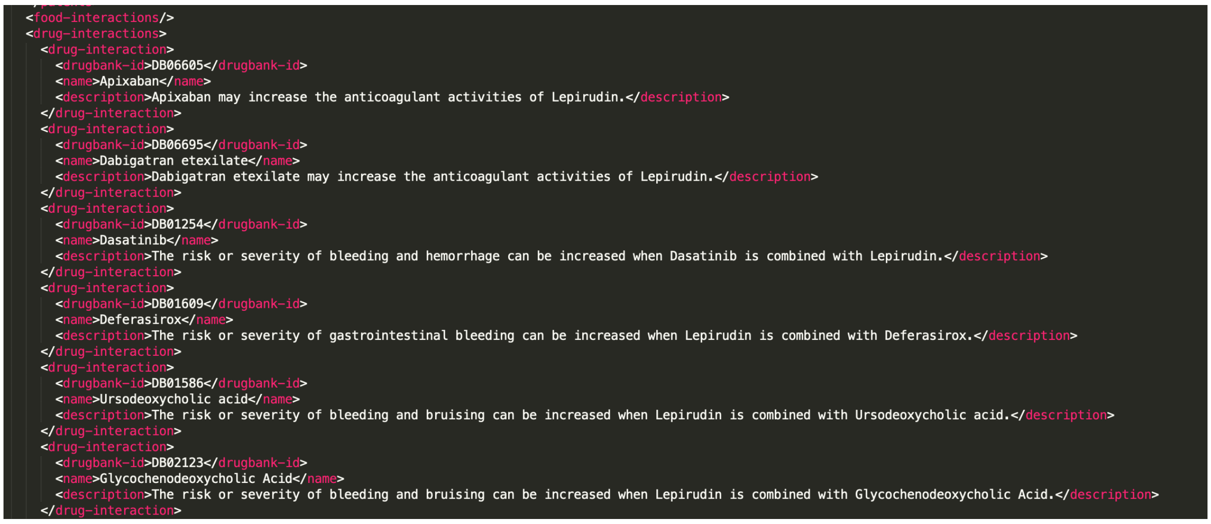
1 Introduction

The aim of this project was to predict drug-drug interactions given prior information of drug-drug interactions, using Graph Machine Learning techniques. A large number of drugs are introduced every year, and it is common to take several different medications at the same time, especially in the elderly/chronically ill. The administration of a combination of drugs can cause a drug interaction, which is when the effect of one drug is modified by another drug and can cause serious, unexpected side-effects for patients. For this reason, studying drug interactions and facilitating their prediction with the incorporation of new drugs is an important problem to address.

We approached this problem using a graph approach. Predicting drug interactions is particularly suited to graph data since each drug can be modelled as a node, and each edge/link as an (unwanted) drug interaction. The task of predicting drug interactions then becomes link prediction in the domain of graphs. In this project, we applied Node2Vec and Graph Convolutional Networks (GCNs) for link prediction on the DrugBank database [1].

2 Dataset

The dataset of choice for this project was the DrugBank database, which contains 9591 drug entries in XML format and is commonly referenced in drug interaction literature. An example screenshot of part of the dataset containing drug interactions is shown below:



```xml
<food-interactions/>
<drug-interactions>
  <drug-interaction>
    <drugbank-id>DB06605</drugbank-id>
    <name>Apixaban</name>
    <description>Apixaban may increase the anticoagulant activities of Lepirudin.</description>
  </drug-interaction>
  <drug-interaction>
    <drugbank-id>DB06695</drugbank-id>
    <name>Dabigatran etexilate</name>
    <description>Dabigatran etexilate may increase the anticoagulant activities of Lepirudin.</description>
  </drug-interaction>
  <drug-interaction>
    <drugbank-id>DB01254</drugbank-id>
    <name>Dasatinib</name>
    <description>The risk or severity of bleeding and hemorrhage can be increased when Dasatinib is combined with Lepirudin.</description>
  </drug-interaction>
  <drug-interaction>
    <drugbank-id>DB01609</drugbank-id>
    <name>Deferasirox</name>
    <description>The risk or severity of gastrointestinal bleeding can be increased when Lepirudin is combined with Deferasirox.</description>
  </drug-interaction>
  <drug-interaction>
    <drugbank-id>DB01586</drugbank-id>
    <name>Ursodeoxycholic acid</name>
    <description>The risk or severity of bleeding and bruising can be increased when Lepirudin is combined with Ursodeoxycholic acid.</description>
  </drug-interaction>
  <drug-interaction>
    <drugbank-id>DB02123</drugbank-id>
    <name>Glycochenodeoxycholic Acid</name>
    <description>The risk or severity of bleeding and bruising can be increased when Lepirudin is combined with Glycochenodeoxycholic Acid.</description>
  </drug-interaction>
```

3 Method

3.1 Parsing the XML Dataset

We extracted the individual drugs and the drug interactions from the XML using PySpark. This resulted in a dataset with drugs and their features, as well as a dataset with each drug interaction as a separate row (i.e. containing drug A's DrugBankID and drug B's DrugBank ID, drug A's name drug B's name). Features regarding group and type, as well as target gene names and enzyme names were also extracted to our intermediate dataset, which we then used to then extract a graph from. For the purpose of our work, we only used the group and type for the graph extraction - further work could be carried out using the target and enzyme gene names as features in the graph. Code for this can be found in *data_processing.ipynb.*

3.2 Graph Extraction

Graph extraction was carried out using Spark to make use of its fast, distributed processing via dataframes. To extract the graph, we obtained all unique drugs (by their DrugBankID, a unique ID given to each drug in the database) and mapped these to unique node IDs. This was to avoid passing extra information about the drug by using the DrugBankID as the node ID, which subsequent graph ML models might learn an unmeaningful representation from regarding the drugs and their interactions. We then joined the drug interaction dataset with these node IDs to obtain a set of edges between drug A and drug B (containing Drug A's node ID and drug B's node ID).

We also extracted features for each drug, including group and type of drug from the XML. 'group' contained a combination of 1 or more of the following: ['withdrawn', 'illicit', 'vet_approved', 'investigational', 'approved', 'experimental', 'nutraceutical']. Since there were only 7 possible values, we decided to create 7 new features with is_withdrawn, is_illicit, etc which took the form of binary variables - 1 if the feature was present in 'group' and 0 otherwise. The same was done for 'type', which contained 1 of [null, 'small molecule', 'biotech'].

3.3 Node2Vec

We created a graph using the StellarGraph [2] library made up of the edges we previously processed, and nodes (with no features). StellarGraph allows for easy creation of a graph from CSV files using the Python Pandas library, hence we used that to do so. We then split the graph into a train, val and test set, and first trained a node embedding model to extract embeddings for each node, and then used this to train a further link prediction model for predicting whether a link exists (meaning an interaction exists) between two nodes. Details of the hyperparameters for the node embedding model and link prediction model can be found in *node2vec-link-prediction.ipynb.*

3.4 GCN

For training the GCN, we again created a StellarGraph and split the graph into train/val/test sets, but this time also passed in node features into the graph (as detailed in the Graph Extraction section). We then created a GCN model and trained and evaluated this on our graph.

4 Results

We present our results from running Node2Vec and a GCN on our graph dataset as follows:

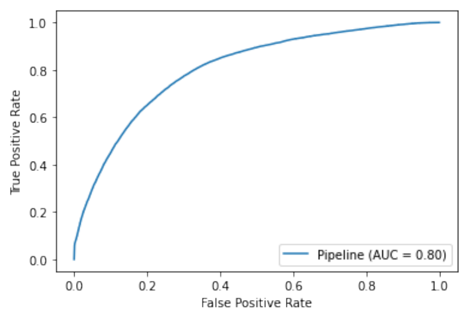
4.1 Node2Vec

After training the node embedding model (with length of the random walk = 80) and the link prediction model for a maximum of 2000 iterations, the following results were obtained using Node2Vec:

Table 1: Results on test set using 'operator_hadamard' (to 4dp)

ROC AUC score	Accuracy score
0.6179	0.5290

Graphs showing the AUC-ROC curve on the test set and PCA visualisation of the link embeddings can be seen in the following Figures 1a-1b.



(a) AUC - ROC curve: When FPR is low, TPR is high.

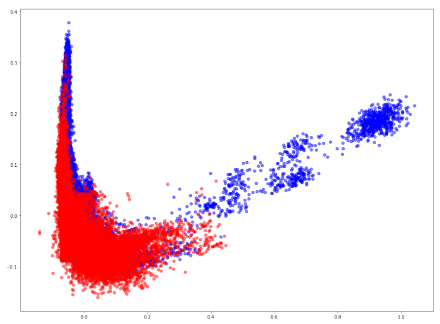

(b) PCA Visualisation on test set. Classes are separated.

Figure 1: Results on test set for Node2Vec.

4.2 GCNs

We trained the GCN for 500 epochs using 'Adam' optimizer with a learning rate of 0.01 and 'binary_accuracy' and 'mean_squared_error' as metrics. Graphs showing the training and validation loss are below:

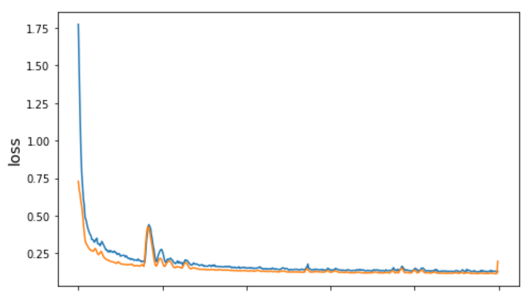


Figure 2: Training/Validation Loss over time

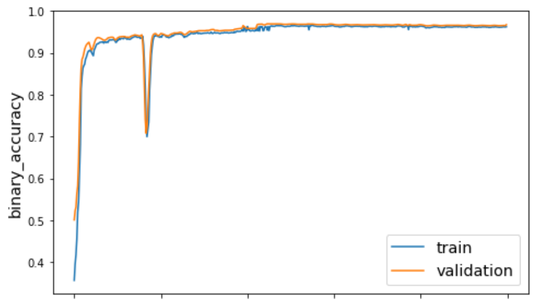


Figure 3: Training/Validation Accuracy over time

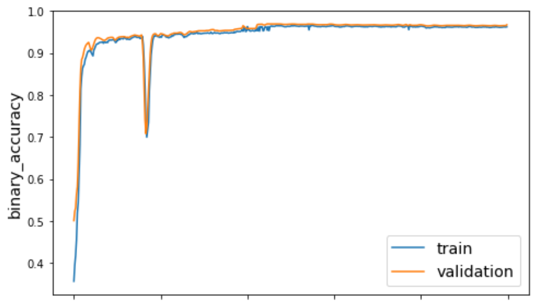


Figure 4: Training/Validation Accuracy over time

After training the model, the test set was used to evaluate it achieving the following results:

Table 2: Results on test set using GCN

Loss	Binary Accuracy	MSE
0.1964	0.9672	0.0537

Performance was much higher using the GCN than Node2Vec model. We reason that this is because of the higher model complexity of the GCN compared to Node2Vec, as well as the fact that we passed in node features to the GCN model, unlike with Node2Vec.

5 How to run the code

We provide the following notebooks and files attached:

- **data_processing.ipynb:** A PySpark notebook containing code for processing the DrugBank XML and producing csv files with the necessary data for further graph extraction.
- **graph_extraction.ipynb:** A Spark notebook containing
- **node2vec-link-prediction.ipynb:** A Python notebook code to run and evaluate Node2Vec on our graph dataset using StellarGraph.

- **gcn-link-prediction.ipynb:** A Python notebook code to run and evaluate GCNs on our graph dataset using StellarGraph.
- **spark-xml_2.12-0.6.0.jar**: We use the Spark Databricks library [3] to read in and parse the XML database. This should be placed inside a folder *spark-jars* inside the same directory as the notebooks.

We used the Docker all-spark notebook [4] to provide an environment to process the data in Spark 2.12 and PySpark. This can be run using the following command:

```
docker run --rm -p 10000:8888 -e JUPYTER_ENABLE_LAB=yes -v
"$PWD":/home/jovyan/work jupyter/all-spark-notebook
```

The node2vec-link-prediction and gcn-link-prediction notebooks were run using Google Colab for access to a GPU.

Results can be replicated by running the *data_processing* notebook, followed by the *graph_extraction* notebook on this extracted data, and then each of the link-prediction notebooks using the extracted graph data.

References

[1] DS Wishart, C Knox, AC Guo, S Shrivastava, M Hassanali, P Stothard, Z Chang, and J Woolsey. Drugbank: a comprehensive resource for in silico drug discovery and exploration. *Nucleic Acids Res*, 1;34, 2006.

[2] StellarGraph. https://stellargraph.readthedocs.io/en/stable/index.html.

[3] Databricks. Spark-xml. https://github.com/databricks/spark-xml.

[4] Dirk Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.