FLAMING LLAMAS

No drama, just llama!

Citation Prediction Challenge

Data Challenge 2021

Mouselinos Spyridon
Mpaltzi Sofia

Contents

Challenge

The purpose of this document is to report the results for the Team Project of class Data Challenge, May - June 2021. The objective is to identify missing citations between academic papers, formulated as a link prediction - among graph entities - problem. That is achieved by applying natural language preprocessing techniques, taking advantage of graph properties, and combining them into machine learning models. Problem is represented as a binary classification task, where true links are labeled as 1 and false links are labeled as 0. Various classifiers are tested, and the best results are presented in this report.

Evaluation

The performance of the developed classification models is evaluated using the logarithmic loss measure:

$$\mathcal{L} = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i \log(p_i) + (1 - y_i)\log(1 - p_i)\right)$$

Where:

- N is the number of node pairs - edges,
- $\mathrm{y_i}$ is 1 if there is an edge between the two nodes, 0 otherwise, and
- $\mathrm{p_i}$ is the predicted probability that there is an edge between the two nodes.

Given the nature of the metric, logarithmic loss gets higher values the more the predicted probability diverges from the actual value.

Data

The Dataset

The first part of the dataset is the citation network, where the nodes correspond to papers published in the wider field of Artificial Intelligence and the edges to citation relationships among those papers. There are totally 1,091,955 relationships – edges connecting 138,499 papers – vertices, all of them creating an undirected graph. The second and third part of the dataset are the abstracts and the authors of the 138,499 papers, respectively.

Vector y of training set is created by taking $\mathrm{N/2}$ true edges – actual node connections – of the graph. Those connections have a y label equal to 1. In order to balance the classification problem, $\mathrm{N/2}$ false edges – non-existent node connections – are added to vector y, with label equal to 0. Those false connections are not chosen randomly, but, iteratively, a non-existent connection around each existent connection of training set is found. That results to a y vector of length N, with N being less than or equal to double the number of true known edges of the network. Each connection of the y vector is related to a set of k features, calculated for the two vertices of the edge. Those features are calculated based on either graph structure or text information. The combination of the N feature sets results to a matrix of features X, of size $\mathrm{N{\times}k}$. Part (20%) of training set, is used for validation of models and the rest for training and hyperparameter tuning.

The developed models are tested on a predefined part of the network, consisting of 106,692 edges, that are either existent or non-existent in the original graph.

Graph Features Generation

As mentioned in the previous paragraph, a subset of the created features is based on the structure of the network. Before presenting those features, it is important to explain some terms that will be mentioned later on.

Jaccard Index

The Jaccard index, or Jaccard similarity coefficient (originally given the French name coefficient de communauté by Paul Jaccard [1]), is a statistic used for gauging the similarity and diversity of sample sets.

In this context, the Jaccard index of two nodes is calculated as the number of common neighbors of the two nodes divided by the total number of neighbors of the two nodes. Let $\mathrm{N(x)}$ be the neighbors of node x and $\mathrm{N(y)}$ be the neighbors of node y, then Jaccard index is calculated as:

$$J(x,y) = \frac{|N(x) \cap N(y)|}{|N(x) \cup N(y)|}$$

Similarly, second degree Jaccard index is calculated by considering the neighbors of the first neighbors of the two nodes.

Adamic/Adar Index

The Adamic/Adar index is a measure used in link prediction that is based on the number of shared links between two nodes. It was originally introduced in 2003 by Adamic and Adar to predict links in a social network [2].

It is defined as the sum of the inverse logarithmic degree centrality of the neighbors shared by the two nodes. Let $\mathrm{N(x)}$ be the neighbors of node x and $\mathrm{N(y)}$ be the neighbors of node y, then Adamic/Adar index is calculated as:

$$A(x,y) = \sum_{u \in N(x) \cap N(y)} \frac{1}{\log |N(u)|}$$

Similarly, second degree Adamic/Adar index is calculated by considering the neighbors of the first neighbors of the two nodes.

DeepWalk Algorithm

DeepWalk [3] learns embeddings of a graph's vertices, by modeling a stream of short random walks. Paper-Linkage representations are latent features of the vertices that capture neighborhood similarity and community membership. These latent representations encode relations in a continuous vector space with a relatively small number of dimensions. It generalizes neural language models to process a special language composed of a set of randomly-generated walks. For the creation of walk embeddings, the Word2Vec [4] model is utilized, operating on this special language, modeling node similarity in a few-shot fashion through negative mining.

Graph Features

Based on the nodes' connections the following features are created:

1. Sum of degrees of edge vertices,
2. Difference of degrees of edge vertices,
3. Sum of second degree of edge vertices,
4. Difference of second degree of edge vertices,

5. Jaccard Index of edge vertices,
6. Jaccard Index of second degree of edge vertices,
7. Adamic Adar Index of edge vertices,
8. Adamic Adar Index of second degree of edge vertices,
9. Shortest path distance, after removing the direct connections, and
10. Cosine similarity between DeepWalk node embeddings.

Text Features Generation

The second category of features are those that are created using the abstracts and the authors of the papers. Before generating those features, some text preprocessing is mandatory.

Abstracts' Preprocessing

Regarding the abstracts, the following transformations are applied:

- All characters are converted to lowercase,
- Non-word characters, symbols and single characters are removed,
- Numbers are removed,
- Multiple spaces are substituted by single spaces,
- Words are lemmatized, and
- Stop words are removed.

Also, as a final step the TF-IDF vector of abstracts is created and then decomposed by applying SVD, in order to capture the concepts of the abstracts.

Abstracts' Features

Based on the preprocessed corpus of the abstracts the following features are created:

11. Number of common terms between the abstracts of two vertices,
12. Number of common terms between the abstracts of two vertices normalized to the total number of words of the abstracts,
13. Number of non-common terms between the abstracts of two vertices,
14. Number of non-common terms between the abstracts of two vertices normalized to the total number of words of the abstracts,
15. Cosine similarity between TF-IDF vectors (unigram and bigram level) of abstracts of two vertices, and
16. Cosine similarity between SVD (applied on TF-IDF) vectors of abstracts of two vertices.

Authors' Preprocessing

Authors are processed in a similar manner:

- All characters are converted to lowercase,
- Text is split to comma to give a list of all authors,
- Non-word characters are removed,
- Multiple spaces are substituted by single spaces, and
- Only first letter of first name is kept and combined with the whole last name, it gives the name of each author.

Authors' Features

Based on the preprocessed matrix of the authors the following features are created:

17. Number of common authors between two vertices,
18. Number of common authors' last names between two vertices,
19. Number of non-common authors between two vertices, and
20. Number of non-common authors' last names between two vertices.

Hack Feature Generation

The last feature is one that simulates node clustering – but is computationally feasible. It is observed that nodes with labels that are close, are also close in the network. That feature only works in the current context and it would not work if the labels were to be mixed. Therefore, the following feature is created:

21. Log of absolute value of difference of node labels.

Modeling

In this paragraph, the classification methods that are used to solve the problem are presented. The hyperparameters of the models are tuned to giving the best possible results.

Logistic Regression

Logistic regression is a linear model used for binary classification. The nature of the classifier does not make it suitable for the complexity of the problem, but it is used as a sanity check in order to evaluate the quality of the features. Training time when the whole training dataset is used is 3.5 minutes on average.

XGBoost

XGBoost is an optimized distributed gradient boosting library. It implements machine learning algorithms under the Gradient Boosting framework [5]. It is selected over Scikit-Learn's Gradient Boosting Trees algorithm implementation, since it provides GPU support that makes training more time efficient.

The model's hyperparameters are tuned via grid-search, resulting to the following choices:

```
max_depth=5, n_estimators=250, learning_rate=0.1, min_child_weight=5,
          gamma=0.1, subsample=0.6, colsample_bytree=0.6,
   objective='binary:logistic', scale_pos_rate=1, reg_lambda=0.01
```

Standalone, it is the best performing algorithm among the three that are tested. Training time when the whole training dataset is used is 10 seconds on average.

Neural Network

The third classifier, is an MLP model, consisting of 4 Dense layers of 200 units. Each layer is followed by a Batch Normalization layer and 50% Dropout rate. Each of these layers uses ReLU as activation function. The output layer is a Dense layer of 2 units that uses Softmax as activation function.

The model is trained in batches of 256 for maximum of 300 epochs, monitoring validation accuracy in a split equal to 20% of the training set. Adam optimizer is used to minimize the loss function that is Categorical Crossentropy. Also, label smoothing is used, in order to keep predictions from getting extreme values (either too close to 0 or too close to 1), since log-loss metric heavily penalizes wrong predictions with large (or small) probabilities. That is the reason why Sigmoid / Binary Crossentropy combination is not used.

Early stopping is used as a callback, terminating the training process if a better validation accuracy is not achieved for 10 consecutive epochs. This solution is better than the Logistic Regression one but worse than the XGBoost one.

Training time when the whole training dataset is used is 3 minutes on average.

Discussion

What worked

Ensemble solutions

As mentioned before, log-loss penalizes overconfident estimations. In order to avoid such results, the two different estimation methods that give similar performance, XGBoost and Neural Network, are pared. The output probabilities of the XGBoost model are summed with the probabilities of the Neural Network and then divided by 2. That ensemble method gives the second-best score of 0.12.

In the same fashion, the best-scoring solution utilizes the Bagging Method, combining 5 classifiers trained on non-overlapping splits of the same training set. This reduces model variance while maintaining a single hyperparameter configuration among all classifiers.

Using fewer training examples

Surprisingly enough, using increasingly more training examples does not seem to boost the test performance of the models. To this end, an ablation study is conducted using the initial 12,000 example dataset and then adding 10,000 new examples at each iteration up until 500,000 examples. This is done in hope to reduce modeling variance and explore unvisited combinations that may arise in the test set. Apparently, this is not the case, since the models seem to overfit proportionally to the dataset increasing in size. The best results are achieved at 12,000 examples overall.

What did not work

DeepWalk

DeepWalk performs decently in absence of other graph features, with the initial settings of 5 walks per node, walk length of 8, 5 epochs of Word2Vec training and every other setting left to default, giving 0.17 test score. However, slightly changing the number of walks and the number of epochs violently alters the result. To make matters worse, the DeepWalk does not seem to cooperate well with the 2^{nd} Jaccard and 2^{nd} Adar features, leading to poor combined performance of around 0.20. After examination of the feature importance analysis done by the XGBoost classifier, DeepWalk seems to absorb importance from the former features.

Sequentially sampled dataset

During the first attempts of creating the dataset, features are created by using each node pair in order, as they are listed on the edge list dataset. That leads to poor performance that is steadily improved by adding more examples into the training step. The same effect of performance saturation is again noticed, where after a number of examples the test performance stops improving. This is the main factor behind switching to a randomly sampled dataset, as described in previous paragraph.

Conclusion

Summing up, the best solution utilizes all but the DeepWalk feature, a small randomly sampled dataset, and ensemble bagging of XGBoost classifiers. The best result achieved gives 0.117 log-loss score on 50% of the test dataset.

References

[1] Jaccard, Paul (1912). "THE DISTRIBUTION OF THE FLORA IN THE ALPINE ZONE.1", New Phytologist. 10.1111/j.1469-8137.1912.tb05611.x

[2] Adamic, Lada A; Adar, Eytan (2003). "Friends and neighbors on the web", Social Networks. 10.1016/S0378-8733(03)00009-1

[3] Bryan Perozzi, Rami Al-Rfou, Steven Skiena (2014). "DeepWalk: Online Learning of Social Representations". https://arxiv.org/abs/1403.6652

[4] Mikolov, Chen, Corrado, Dean (2013). "Efficient Estimation of Word Representations in Vector Space". https://arxiv.org/abs/1301.3781

[5] eXtreme Gradient Boosting, https://github.com/dmlc/xgboost