Kaggle - Report

Team: OTCNet

1. DESCRIPTION

The goal of this competition is to predict missing links in a citation network of research articles. A citation network is represented as a graph G = (V, E), where the nodes correspond to scientific articles and the existence of a directed edge between nodes u and v, indicates that paper u cites paper v. Each node (i.e., article) is also associated with information such as the title of the paper, publication year, author names and a short abstract.

A number of edges have been randomly removed from the original citation network. Therefore, the goal is to reconstruct the initial directed graph by predicting the missing edges.

2. DATA

To help us in this classification task, we have at our disposal the file 'node-information.csv', which contains for each of the 27,770 papers in the citation network 6 different type of information:

1. The paper's unique ID (integer)
2. The publication year of the paper (integer, between 1993 and 2003)
3. The title of the paper (string)
4. The authors of the paper (strings separated by a comma)
5. The name of the journal where the paper was published (string, however this information is not always available)
6. The abstract that sums up the content of the paper (string)

The training set consists in 615,512 labeled node pairs. One pair and label per row, as:

1. Source node ID
2. Target node ID
3. The label, 1 or 0 (1 is there is an edge between the source and the target nodes, 0 otherwise).

The IDs match the papers'IDs contained in the 'node-information.csv' file.

The testing set consists in 32,648 node pairs. One pair per row, as:
1. Source node ID
2. Target node ID

3. METHODOLOGY

To build our models, we followed the following pipeline:

1. From the training set, we saved in 2 distinct lists nodes = [papers'IDs] and edges = [the labels, 1 or 0, associated to each row of the training set, that is the pairs (source node, target node)].
2. Using iGraph (or networkX) libraries, we reconstructed a directed graph.
3. From there, we randomly selected 10% of the training set to design features (see section 5 - 'Features' for more details) given the huge size of the training set - +600,000 rows. We convert the above-mentioned training features into a numpy array and scale them.
4. We split the training set (with +600,000 rows) into 2 parts to have a training and a test sets. The test set will help us assess the score that we may obtain on the platform after submitting our predictions.
5. We fit different classifiers and tune their hyperparameters on the new training set (around 300,000 rows) with the newly created features and the corresponding labels (made of 1s and 0s).
6. We then compute the score on the testing set we created.
7. We compute the same features for the 'real' testing set.
8. We pick the model that obtains the best F1 score on the testing set and use it to make predictions on the 32,648 node pairs contained in the 'real' testing set.

4. FEATURES

The first and most important part of the job was to create new features that would enable us to improve our prediction accuracy.

We created additional features to the 3 ones provided by the Python script ((1) number of overlapping

words in paper titles, (2) difference in publication years, and (3) number of common authors). We mostly used neighborhood-based methods:

1. **Number of common keywords in between source and target abstracts** (of the source and target nodes): this feature required some preprocessing steps. After cleaning the abstracts (we remove the punctuation, special characters, figures, stopwords), we computed a tf-idf index for each remaining word of the vocabulary of the abstracts. From these scores, we removed the 100 most frequent words (we assumed that they we not the most important ones) and kept the 1,000 rarest words, assuming that they would bear the key themes of the abstracts. For each row of the reduced training set (= 10% of the training set), we kept track in a list of the number of these 1,000 rare words that would appear in both abstracts. The underlying idea behind this feature is that 2 papers that share common rare words are likely to be related to the same field and therefore are likely to cite one another.

2. **The number of common words between the name of the journal** where both papers (source and target nodes) were published. If 2 papers appear in the same or in similar journals then there are likely to be related.

3. **Degree relation**: A list that contains the sum, for each pair of papers (source, target), between the out-degree of the source node and the in-degree of the target. A paper that is often cited is very much likely to be a reference and therefore is likely to be cited again by other papers.

4. **Preferential attachment**: The intuition behind this score is that nodes prefer to create ties with 'popular' nodes. This score is equal to the product of the number of neighbors of both the source and the target nodes.

5. **Number of common neighbors**: For each pair (source, target), count the number of common neighbors shared by both nodes.

6. **Jaccard similarity score**: This score is the probability that the source and the target nodes share common neighbors.

7. **Betweenness centrality**: A node is important if it lies in many shortest paths. It shows that certain nodes are essential to pass the information in the network.

8. **Closeness centrality**: It measures the mean distance between a node to other nodes. If a node is close to everybody else, then it has a high influence on the other nodes in the network.

9. **Adamic-Adar index**: Assigns large weights to common neighbors of the source and the target nodes which themselves have few neighbors (weight rare features more heavily). The Adamic/Adar predictor formalizes the intuitive notion that rare features are more telling.

5. MODEL

Note that the Kaggle platform provided us with a Python script containing 2 baselines:

1. A random guessing: F1 score of approximately **0.5**
2. A linear SVM with three features (1) number of overlapping words in paper titles, (2) difference in publication years, and (3) number of common authors: F1 score of approximately **0.68**

As a consequence, our model has to score higher than 0.68 on the platform.

To improve on this baseline model, we optimized the hyperparameters, thanks to a grid search, on the 3 following models on a division of the training set between training and testing sets (the final results presented here are on a training set of size 275,000 and testing set of 35,000):

1. **Linear SVM** with hyperparameters set at C= 100, we obtained a accuracy score of **93.2%**

2. **Logistic regression** with hyperparameters set at C (inverse of normalization coefficient) = 1000 , panalty = 'l2',
we obtained a accuracy score of **91.9%**

3. **Random forest** with hyperparameters set at:
max depth (The maximum depth of the tree): 3, max features (The number of features to consider when looking for the best split): 3
min samples split (The minimum number of samples required to split an internal node): 13
min samples leaf (The minimum number of samples required to be at a leaf node): 11
bootstrap: False
"criterion": "entropy"
we obtained an accuracy score of **94,6%**

Finally, we chose to work with the best performing model, i.e. the Random forest classifier.

6. EVALUATION

The evaluation metric for this competition is **Mean F1-Score**. The F1 score measures accuracy using precision and recall.
Precision is the ratio of true positives (tp) to all predicted positives (tp + fp).

Recall is the ratio of true positives to all actual positives (tp + fn).
Accuracy is the ratio of true positives and true negative (tp + tn) to total population.