DATA CHALLENGE

Kaggle Team:
Barsha barsha

Mohamed Dhouib
Abderrahim Mama

ÉCOLE POLYTECHNIQUE
IP PARIS

FEATURE SELECTION/EXTRACTION

Graph features

The graph represented by the file **coauthorship.edgelist** is a co-authorship network where nodes correspond to authors that have published papers in computer science venues (conference or journal) and two nodes are connected by an edge if they have co-authored at least one paper.

To extract information from the graph we chose to focus on these features :

— **Degree** : Since the edges are unweighted, the degree of a node is simply the number of nodes directly connected to it. The higher is the degree of a node the more likely it has a high h-index.

— **Neighbor's average degree** : The average degree of the neighborhood of a vertex. It captures the average connectivity of the neighborhood.

$$averageneighbor(u) = \frac{\sum_{v \in N(u)} deg(v)}{|N(u)|}$$

where $N(u)$ contains the neighbors of u.

— **Core number** : A subgraph of a graph G is defined to be a k-core of G if it is a maximal subgraph of G in which all vertices have degree at least k. The core number of a vertex u is the highest order core that u belongs to.

— **Onion layer** : The onion decomposition refines the k-core decomposition by providing information on the internal organization of each k-shell where a k-shell is the set of nodes that are in the k-core but not in the $(k+1)$-core.

— **Pagerank** : Pagerank is an algorithm that computes a ranking of the vertices in a graph based on the structure of the incoming egdes. The main idea behind the algorithm is that a vertex spreads its importance to all vertices it links to :

$$PR(u) = \sum_{v \in N(u)} \frac{PR(v)}{deg(u)}$$

— **Eigenvector centrality** : Eigenvector centrality measures a node's importance while giving consideration to the importance of its neighbors. A node which is connected to 2 central nodes (important either because they form a bridge between communities of a graph or because they have a nobel price in our case for example) will have a higher eigenvector centrality than a node connected to two unpopular nodes. The eigenvector centrality is the unique solution of the linear system :

$$Ax = \lambda x$$

where λ is the largest eigenvalue of the adjacency matrix A associated to the graph.

— **Papers number** : the number of papers written by each author. Obviously, the h-index of an author will a have a dependence on the number of research papers he wrote.

— **number of triangles per node** : the number of triangles that include a node as one of the vertices. It captures communities of authors and measures the cohesiveness of those communities.

Then we used χ^2 feature selection on those features to keep only independent features.

Features from abstracts

To be able to use the information in the texts we need to transform them into vectors of numbers. To do that we need to transfer every word to a vector first. One simple method is the use onehotencoding, thus associating an index to each word and transforming it to a vector of zeros except a one in the position of its index. This has two major drawbacks :

— The dimension of the input will be huge : More than 100000 in our case.

— The distance between two vectors is the same. Thus our words lack sense

So to surpass these drawbacks we tried to embed those vectors into a smaller dimension.To do that we used Skip-gram. To train the model we used **negative sampling** : Our model takes as input only one word and as output a word of it's context : And for the outputs we only take NumberOfNegativeSimple neurons each one representing a word (including the chosen word of the context). Thus each have a part of our full network which is much smaller and therefore will train faster.

The goal of the first layer is to encode the information and thus to give each feature more sens.Therefore our embedding matrix will be the matrix associated to the weights of this layers. And the embedding of a word is simply the line corresponding to it's index. So two hyperparameter need to be tuned :

— Embed Size : the size of the first dense layer and thus our embedding size. We tried different values $10, 20, 30, 60, 100, 300$. The one that gave the best result is 100.

— Window size : we tried 5 and 10. The gave similar results.Note that the words are drawn in a way that the closest ones have a more chance to be drawn.

(Cf photo [2] Appendix)

To allow our network train on text data pre-processing was needed :

— Filter stop words : Stop words are really common words in a way that they don't contribute to the sens of the sentence

— Remove extra white spaces from texts

— Remove accented characters : As an example replace café with cafe.

— Expand contractions : Expand shortened words like transforming don't to do not.

— Convert all characters to lowercase

— Remove special characters

— Convert number words to numeric numbers

— lemmatization : it is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma. So we reduce Our vocabulary in a way to simplify the training.

— Tokenization : splitting each abstract into a list of words. It's done to simplify the model training as words will need to be extracted.

Also the abstracts contained many aspects that make learning more difficult : Words in a language different than english, concatenated words without spaces between them, equations ... To solve that we decided to delete each word that is not repeated more than 5 in all the documents (we tried also 10 times and we had roughly the same results). This will have another positive effect : Even for "normal" words that are repeated less than 5 times, the model won't have enough information to give a sens to these words. So they will be just noise for our Skip-Gram model. Thus deleting them will make learning more efficient and calculated a text embedding more representative. Finally to calculate an embedding for each author we calculated the mean of the embedding of all the words in his abstracts.

This gave us a little problem because some abstracts are missing, so we had two options : give those authors an embedding of zeros or simply not train on the data related for these authors.We tried both and it gave similar results

MODEL TUNING AND COMPARISON

We divided our training set into a training and a validation set. We did the split in a way that we respected the distribution of h-index. The distribution of hindex in the validation and the training set. And to deal with the imbalacment of data between the high and low h-indexes, we tried over-sampling with replacement to balance the data. With this new data, our different models overfit the minority classes (high hindexes) as we repeat the same sample several times.

Thus, we tried another method which is the penalization of low h-indexes which has slightly improved the results : in the loss function to optimize we add weights that take into consideration the proportion of correspondent samples :

$$w(h) = \frac{n_{samples}}{n_{sampleshavingthesameh-index}}$$

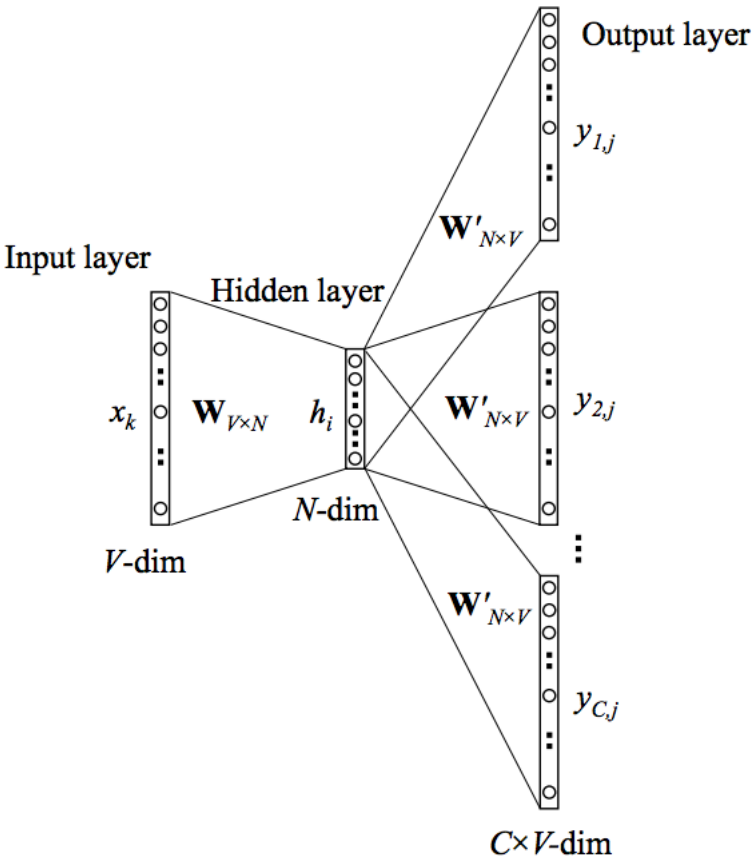
where h is a certain hindex.

The results below are those of the validation set after standardization.

HOME-MADE EMBEDDING

Regressor	Graph	Embeddings	Graph + embeddings
XGboost	98.7	95.1	64.2
LightGbm	**93.5**	**88.3**	**59.8**
Lasso	125.1	103.3	82.8
MLP	97.1	97.8	65.1
Knn regressor	130.2	120.5	88.3

After this, we tried to compare with pretrained embeddings of ConceptNet Numberbatch and it gave very close results.



[2] Skip-Gram model