DATA CHALLENGE

Kaggle Team:
BARSHA BARSHA

Mohamed DHOUIB
Abderrahim MAMA

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 use the texts information, we need to transform words into vectors of numbers. Thus, we need to transfer each word to a vector first. One simple method is the use one hot encoding, 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. Two distance between vectors does not properly express the real distance between words.

As a remedy to this, we tried to embed those vectors into a smaller dimension. To do that, we used Skip-gram. The core feature of the skip-gram model is the use of softmax operations to compute the conditional probability of generating context word w_o based on the given central target word w_c.

$$\mathbb{P}(w_o \mid w_c) = \frac{exp(u_o^\top v_c)}{\sum_{i\in\mathcal{V}} exp(u_i^\top v_c)}.$$

The logarithmic loss corresponding to the conditional probability is given as

$$-\log \mathbb{P}(w_o \mid w_c) = -u_o^\top v_c + \log\left(\sum_{i\in\mathcal{V}} exp(u_i^\top v_c)\right).$$

Because the softmax operation has considered that the context word could be any word in the dictionary $\mathcal{V}$, the loss mentioned above actually includes the sum of the number of items in the dictionary size. Hence, the gradient computation for each step contains the sum of the number of items in the dictionary size. For larger dictionaries with hundreds of thousands or even millions of words, the overhead for computing each gradient may be too high. In order to reduce such computational complexity, we will introduce an approximate training method in this section : **negative sampling**

Negative sampling modifies the original objective function. Given a context window for the central target word w_c, we will treat it as an event for context word w_o to appear in the context window and compute the probability of this event from

$$\mathbb{P}(D = 1 \mid w_c, w_o) = \sigma(u_o^\top v_c),$$

Here, the σ function has the same definition as the sigmoid activation function :

$$\sigma(x) = \frac{1}{1 + \exp(-x)}.$$

We will first consider training the word vector by maximizing the joint probability of all events in the text sequence. Given a text sequence of length T, we assume that the word at time step t is $w^{(t)}$ and the context window size is m. Now we consider maximizing the joint probability

$$\prod_{t=1}^{T} \prod_{-m\leq j\leq m,\ j\neq 0} \mathbb{P}(D=1 \mid w^{(t)}, w^{(t+j)}).$$

However, the events included in the model only consider positive examples. In this case, only when all the word vectors are equal and their values approach infinity can the joint probability above be maximized to 1. Obviously, such word vectors are meaningless. Negative sampling makes the objective function more meaningful by sampling with an addition of negative examples. Assume that event P occurs when context word w_o to appear in the context window of central target word w_c, and we sample K words that do not appear in the context window according to the distribution $\mathbb{P}(w)$ to act as noise words. We assume the event for noise word $w_k (k = 1, \dots, K)$ to not appear in the context window of central target word w_c is N_k. Suppose that events $P and N_1, \dots, N_K$ for both positive and negative examples are independent of each other. By considering negative sampling, we can rewrite the joint probability above, which only considers the positive examples, as

$$\prod_{t=1}^{T} \prod_{-m\leq j\leq m,\ j\neq 0} \mathbb{P}(w^{(t+j)} \mid w^{(t)}),$$

Here, the conditional probability is approximated to be

$$\mathbb{P}(w^{(t+j)} \mid w^{(t)}) = \mathbb{P}(D=1 \mid w^{(t)}, w^{(t+j)}) \prod_{k=1,\ w_k\sim\mathbb{P}(w)}^{K} \mathbb{P}(D=0 \mid w^{(t)}, w_k).$$

Let the text sequence index of word $w^{(t)}$ at time step t be i_t and h_k for noise word w_k in the dictionary. The logarithmic loss for the conditional probability above is

$$\text{-}\log \mathbb{P}(w^{(t+j)} \mid w^{(t)}) = -\log \mathbb{P}(D=1 \mid w^{(t)}, w^{(t+j)}) - \textstyle\sum_{k=1,\ w_k\sim\mathbb{P}(w)}^{K} \log \mathbb{P}(D=0 \mid w^{(t)}, w_k)$$

$$= \text{-} \textstyle\sum_{k=1,\ w_k\sim\mathbb{P}(w)}^{K} \log\left(1-\sigma\left(u_{h_k}^{\top} v_{i_t}\right)\right) - \log\, \sigma\left(u_{i_{t+j}}^{\top} v_{i_t}\right)$$

$$= \text{-} \textstyle\sum_{k=1,\ w_k\sim\mathbb{P}(w)}^{K} \log \sigma\left(-u_{h_k}^{\top} v_{i_t}\right) - \log\, \sigma\left(u_{i_{t+j}}^{\top} v_{i_t}\right).$$

Here, the gradient computation in each step of the training is no longer related to the dictionary size, but linearly related to K. When K takes a smaller constant, the negative sampling has a lower computational overhead for each step.

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.

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 (example : transform 'don't' to 'do not').

— Lowercase all characters.

— Remove special characters.

— Convert number words to numeric numbers

— lemmatization : it is the process of grouping 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 to simplify the training. (example : transform 'giving' to 'give')

— Tokenization : splitting each abstract into a list of words.

Besides, the abstracts contained many aspects that make the learning more difficult : Non English words, concatenated words without spaces , equations ... To solve that we decided to delete each word that is not repeated more than 5 times 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 the learning more efficient and the embedding more representative. Finally to calculate an embedding for each author we calculated the mean of the embeddings of all the words in his abstracts.

This gave us a little problem because some abstracts are missing, so we had two options : give the correspondent authors zero-embeddings or simply not take them into account in the training phase .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 :

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

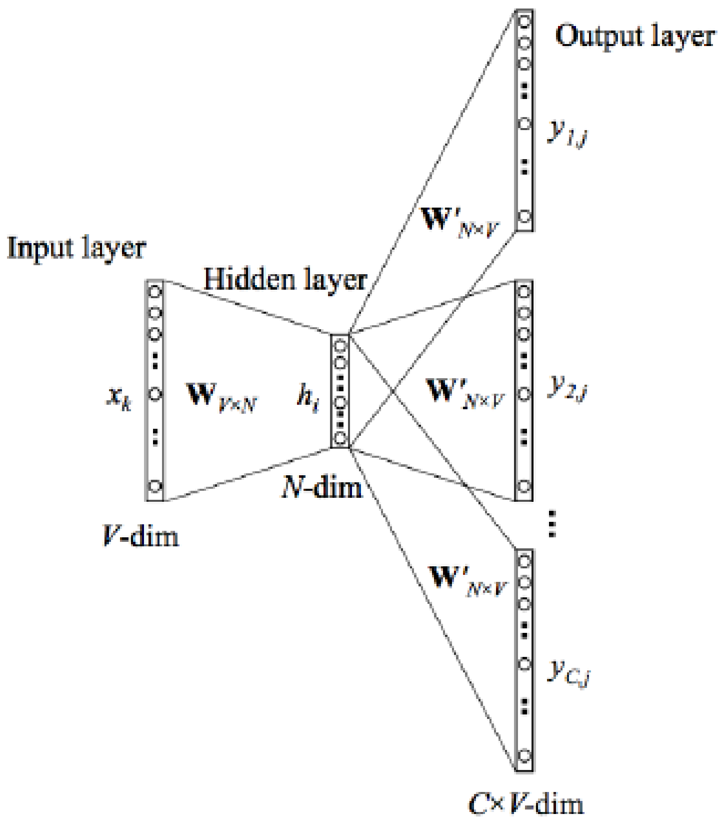
where h is a certain hindex.

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