French web domain classification

Nabil Madali , Mélanie Karlsen , Virgile Rennard

Abstract

Classification is one of the most common and well defined applications of machine learning. Whether it be for numerical, text, images, or graphical data, many methods have been developed to treat each of this cases. Web page classification is a well treated subject [12], [9], [5],[20]. This kind of classification mainly works with text data [1]. However, the architecture of webpages is easily described with graphs and therefore lends itself especially well to the many node classification methods, [2], especially with the many new Graph Neural Networks methods. In this paper, we will present different ways to translate the methods to French web domain classification, on both text and graph data.

1 Introduction

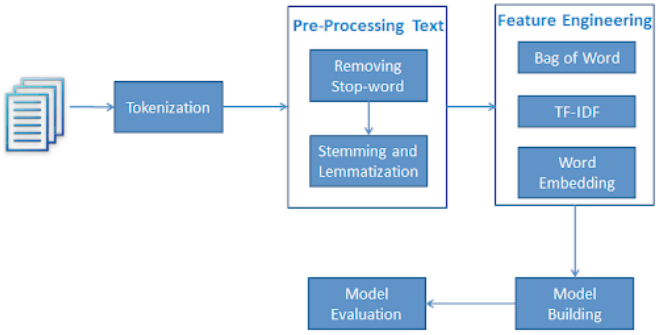
Web domain classification is a common classification task with different uses with a lot of literature and methods developed for it. Because of it, a staggering amount of methods have been developed to work on this problem [12]. This work will try to make sense of some of these methods, and to apply it to french web domains.

As with most classification tasks that rely on text data, the results and literature concern english web pages. The aim of this paper is to try and translate the different usual classification methods developed to french web domain classifications. This is a task that relies on both text and graph data, where the nodes represent each websites, and the edges the hyperlinks. In this case, the text data has been obtained by crawling the html files that were then parsed.

Graphical data has always been a wise choice for web domains applications, as was proven countless times, most notably with PageRank [10]. We will be discussing different state of the arts methods for both text and node classification in graph data, and the results each of these methods had on our problem. As with most data challenges, as well as with most work on text data, an important part will be devoted to both preprocessing steps, as well as parameters tuning. We will therefore start by introducting the multiples text preprocessing methods, such as stemming/lemmatization, the removal of stopwords and text normalization, which is particularly useful as many websites such as blogs and social media benefits hugely from normalization. We will then talk about the different uses of graphical data for this challenge, with an introduction on Node Embedding, obtained with both unsupervised and semi supervised methods, starting with the simpler methods such as DeepWalk [11], to more complex methods, as LINE [13], Node2Vec [7], and SDNE [16]. We will then discuss the application of GNNs to our data, and the results obtained with Convolutional GNN [17], Graph Attention Networks (GAT) [15], and SplineCNNs [6].

2 Text Preprocessing

The first step of text analysis is preprocessing. In this section, we will follow the following pipeline :



2.1 Stemming

Stemming is a process that reduces the distortion of a word, to have its root. Here, ”root” doesn’t represent the true root word, but only the canonical form of the original word.

Stemming uses a heuristic process that cuts off the end of a word to properly convert it to its root form. So the word ”troubled” might actually be translated to troubl instead of trouble because the ends are cut off .

There are different algorithms for stemming, the main one being the Porter stemmer; It is known as one of the easiest and most effective stemming algorithm for english.

Influence of stemming : Stemming is useful for dealing with sparsity issues and standardizing vocabulary. One of the

interest of stemming is that, if you are looking for "English classes", you are also interested in the term "English class". We want all the variations of the words to get the most relevant documents.

However, in our experiments , the classification accuracy was only slightly improved

2.2 Lemmatization

Lemmatization is similar to stemming. Its goal is to remove the inflections and map the word to its root form. The only difference is that inflection reduction attempts to do it the right way. It doesn't just cut it off, it actually turns the words into actual roots. It can use dictionaries such as WordNet for mapping.

WordNet's word reduction effect. Morphological simplification has no clear advantage over search and text classification purposes. In fact, depending on the algorithm you choose, it may be much slower than using a very basic stemmer, and you may have to know the part-of-speech of the related words to get the correct lemma. It is interesting to see how it affects your performance metrics.

2.3 Stop words removal

Stop words are words commonly used in a language. for instance, in English stop words are words such as "a", "the", "is", "are", etc. Removing low-information words from the text gives us better focus on relevant information.

As an exemple, if you search query online is "What is deep learning?", you want the search engine to find surface documents about deep learning. This comes up easier by avoinding the analysis of all the stop words in a given query. Stop words are commonly used in search systems, text classification applications, topic modeling, topic extraction, and more.

In our experiments, deleting stop words, although effective in search and topic extraction systems, appears to be superfluous in classification systems. However, it does help reduce the number of features considered, which helps keep the size of the model in check.

2.4 Text Normalization

A highly overlooked preprocessing step is text normalization. It is the act of normalizing word to their canonical forms. For example, the words "yesss" and "yah" can be normalized to "yes". This is relatively important for us, as blogs, social media and such often have misspelled words (intentionnaly or not). It has been shown to be effective, as an exemple, improving the accuracy of sentiment classification for tweets by 4%

2.5 Term Frequency-Inverse Document Frequency

TF-IDF (Term Frequency-Inverse Document Frequency) is a common weighting technique used for information retrieval and text mining. Assuming a corpus contains multiple files, TF-IDF is used to evaluate the importance of a word to one of the files in a corpus. The importance of a word increases proportionally with the number of times it appears in the file, but also decreases inversely with the frequency of its appearance in the corpus.

Term Frequency (TF) : The term frequency refers to how often a given word appears in the file. For the word t_i in a file, its importance can be expressed as:

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

Among them, $n_{i,j}$ is the number of occurrences of the word in the file d_j, and the denominator is the sum of the occurrences of all words in the file d_j.

Inverse text frequency (IDF) : Inverse document frequency is a measure of the general importance of a word. The idf of a particular word can be obtained by dividing the total number of files by the number of files containing the word, and then taking the log of the quotient to the base 10:

$$idf_i = log \frac{N}{| \{d \in D : t \in d\} |}$$

Where $|D|$ is the total number of files in the corpus and $|j : t_i \in d_j|$ is the number of files containing the term t_i If none of the files contain the word t_i, then the denominator is zero, so $|j : t_i \in d_j| + 1$ is usually used.
Using the TfidfVectorizer class in the sklearn library can help us complete the three steps of vectorization, TF-IDF and standardization.

2.6 Experiments

Hyper-parameter tuning. The experiments compare various convex stochastic and incremental optimization methods. Some algorithms require tuning one or more hyperparameters such as the learning rate. We use grid search to find the best choice of hyperparameters. We choose the value of the hyperparameter that achieves the best objective function value within a given iteration budget.

We use Cross-validation a technique for model validation that assesses how the results of a statistical analysis will generalize to an independent data set. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in an estimate that generally has a lower bias than other such methods.

Model	Text	Translated Text	Normalized Text
Logistic	1.20	**1.16**	1.20
SVM	1.26	1.17	**1.11**
XGBoost	1.19	**1.15**	1.18
Random forest	1.26	**1.20**	1.24
SGD	1.08	1.06	**1.04**

Table 1: Experimental Results

Analysis : In Table 1 are compared various convex stochastic and incremental optimization methods,the SGD methods converges faster in terms of training error while achieving a competitive performance in terms of the performance metric on a held-out set.

3 Node Embedding

3.1 State of the art

Graph-based semi-supervised learning is a typical semi-supervised learning method that maps the entire data set into a graph. Each point in the graph represents a sample. The weight of the edges in the graph is proportional to the similarity between the samples. The difference between graph-based methods and general machine learning methods lies in the use of the relationship between the samples of the graph structure. Different methods use the graph structure, the category labels of the nodes, and the characteristics of the nodes from different angles and in different ways.
There are three major categories:

Label propagation method [8] propagates the label information of labeled samples on the graph, and defines the Laplacian regularity on the labels of neighboring nodes, which is used to restrict. Adjacent nodes with large edge weights have the same category. This type of method uses only the network structure and the labels of the nodes.

Unsupervised graph embedding method uses the network structure and the characteristics of the nodes to learn the feature representation of the nodes, ignoring the labels of the nodes. When embedding is learned, standard supervised learning is applied to these embedding features to train the model to classify nodes.

- Factorization based method
- Method based on random walk
- Method based on automatic encoder

Semi-supervised graph embedding methods use neural network models for feature embedding. These methods calculate supervised loss on labeled samples, and use Laplacian Smoothing regularity to calculate unsupervised loss on all samples. Therefore, the network training, node features, and node labels are used in the model training process.

- Planetoid model: single-input dual-input neural network
- SEANO model: two-input two-output neural network
- Graph Convolutional Neural Network GCN

The graph embedding method focuses on using neighbor features to encode nodes into a low-dimensional space, learning the feature encoding of the nodes, and then constructing a classifier based on the learned feature encoding. Context-trained codes can be used to improve the performance of related tasks. Typically, word encoding trained from a language model can be applied to part-of-speech tagging, sentiment classification, and named entity recognition. It is worth noting that not all graph embedding methods are semi-supervised. In fact, most of the early graph embedding methods were unsupervised. These methods put embedding tasks and classification tasks in two different models. First, they used unsupervised embedding to learn node feature representations, and then trained classification based on the learned features Device. This pipeline approach makes graph embedding methods widely used for various tasks of graph analysis, such as node classification [3] , link prediction [8], community detection [4], recommendation system [18] , and visualization [14] .

However, this method lacks the adjustment of supervised information during the training process. The encoded features are not targeted to the category labels, which may make it difficult to separate the nodes of different categories in the feature space. Some recent graph embedding methods combine the two tasks of embedding and classification into a unified end-to-end model, and propose a semi-supervised graph embedding method. **Experimental** results show that on graph node classification tasks, semi-supervised graph embedding requires only a small percentage of labeled samples to achieve good results.

DeepWalk [11] uses the encoding of a node to predict its context in the graph, whose context is generated by random walk. For each training pair (i, c), sample i is the current node, and c is the sequence generated by sample i using random walk sampling among neighbors of a certain window size.

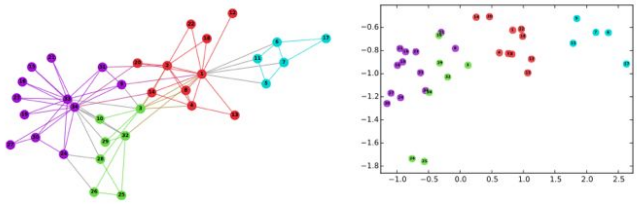


Figure 1: DeepWalk: online learning of social representations

The method is very simple, take a certain point as the starting point, do a random walk to get the sequence of points, and then get the sequence as a sentence, use word2vec to learn, and get the vector of the point.

In essence, random walk is used to capture the local context information of the point in the graph. The learned representation vector reflects the local structure of the point in the graph. The two points share the neighboring points in the graph , the shorter the distance between the corresponding two vectors. As shown above.

LINE [13] algorithm uses the existing edges in the graph to construct an objective function that explicitly depicts the first- and second-order proximity relationships. Then the optimization method is used to learn the expression vectors of the points. This is similar to recommendation. Any recommended algorithm is essentially a smoothing of the user-item relation matrix.

LINE can be used for directed, undirected, weighted and unweighted graphs. It explicitly considers the first-order and second-order proximity relations.

Model of order proximity
u_i is the expression vector of a point c, $p_1(v_i, v_j)$ is the probability corresponding to the first-order proximity relationship between two points

$$p_1(v_i, v_j) = \frac{1}{1 + -\overrightarrow{u}_i^T \overrightarrow{u}_j}$$

And this is the true value of the first-order neighbor relationship, W is the normalization constant

$$\hat{p}_1(v_i, v_j) = \frac{w_{ij}}{W} \quad W = \sum_{(i,j)\in E} w_{ij}$$

The process of learning is the process of reducing the distance between two distributions

$$O_1 = d(\hat{p}_1(.,.), p_1(.,.))$$

The distance d is the KL-divergence and is often used to quantify and describe the difference between two distributions.

Model of second-order proximity

For each node, LINE constructs two corresponding vectors, one is the expression vector of the node, and the other is the expression vector when the node is used as the context. The probability that a V_j node appears in the context of a V_i node is

$$p_2(v_j \mid v_i) = \frac{exp(\overrightarrow{u'}_i^T \overrightarrow{u}_j)}{\sum_{k=1}^{|V|}}$$

The true value of the corresponding fit

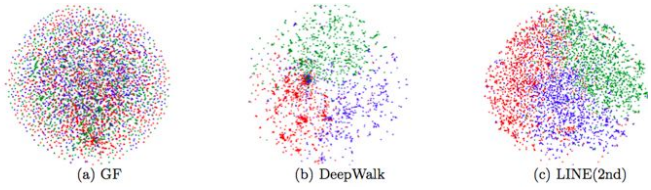
$$\hat{p}_2(v_j \mid v_i) = \frac{w_{ij}}{d_i}$$

Where,d_i is the sum of the weights of all edges, which is the normalization constant. The loss function is:

$$O_2 = \sum_{i\in V} \lambda_i d(\hat{p}_2(.,.), p_2(.,.))$$

In essence, the probability p_2 takes a total of V values, which is the probability that each node is in the context of v_i. The more similar p_2 of the two nodes, the more similar the context of the two nodes, that is, the larger the second-order neighbor relationship.

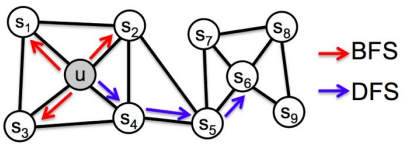
The last thing to do is to concatenate the expression vectors learned from the first-order and second-order relationships. In the original article of LINE, the author compares the different effects of each algorithm .



(a) GF (b) DeepWalk (c) LINE(2nd)

It can be seen that the effect of LINE is the best. DeepWalk also divides the dots of different colors well, but many points in the middle of the figure are squeezed together. The article mentions that for such points, dealing with DeepWalk makes a lot of noise, but this analysis is not detailed. The original text suggests that Deepwalk uses random walks to capture neighboring relationships, which is more like depth-first search; and LINE's method is more like breadth-first search, which is relatively more reasonable.

Node2Vec [7] uses biased random walks to provide a trade-off between width-first (BFS) and depth-first (DFS) searches, and therefore can generate higher quality and more informative codes than DeepWalk. Choosing the right balance allows node2vec to preserve the community structure and the structural equivalence between nodes.



The article points out that the vector learned from graph embedding should be able to represent two kinds of information. One is that if two nodes share many neighboring nodes, they are similar, such as u and s_1; the second is that if two nodes play similar roles in the network, then they should also be similar.

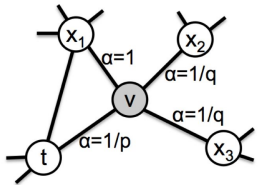
Based on DeepWalk's architecture, this paper optimizes the random walk sequence extraction strategy. This strategy has some hyper-parameters. Different choices of parameters can get different trade-offs in the two similarities mentioned above.

BFS tends to search around the nodes and explore local structural forms, so it tends to learn the second structural equivalence. DFS tends to search higher-order neighbors on a larger scale, so it tends to learn the first homophily. The random walk strategy proposed in this paper is a combination of BFS and DFS.

Suppose the node jumps from t to v, then the probability of jumping from v to x is now proportional to the edge weight w and an adjustment factor:

$$\pi_{v,x}\alpha_{p,q}.w_{v,x}$$

$$\alpha_{pq} = \begin{cases} \frac{1}{p} & \text{if } d_{tx} = 0 \\ 1 & \text{if } d_{tx} = 1 \\ \frac{1}{q} & \text{if } d_{tx} = 2 \end{cases}$$



Among them, d_{tx} is the distance from t to x, there are only three values of 0, 1, and 2, which correspond either to : x is t itself, x and t are directly connected, or x and t are not directly connected.

Obviously, the return parameter p controls the probability of

jumping back to the previous node t. The smaller p is, the more local the walk is. The larger p is, the less the walk will not access the previous node, and it tends to search a larger range; The larger the out parameter q, the more local the search, and the smaller q, the more inclined to DFS.

This random walk strategy is related to the current node v and the previous node t, which is a second-order Markov chain.

Finally, the model assumes that the probability that n_i nodes appear in the context of u nodes is:

$$P_r(n_i \mid f(u)) = \frac{exp(f(n_i)f(u)}{\sum_{v \in V} exp(f(n_i)f(u)}$$

The overall optimization function is:

$$max_f \sum_{u \in V} [-logZ_u + \sum_{n_i \in N_S(u)} f(n_i)f(u)]$$

SDNE [16] uses an automatic encoder to encode nodes in the network.

This paper believes that the disadvantage of DeepWalk is the lack of a clear optimization goal, and that LINE learns the local and global information of the network separately. Finally, simply connecting the two expression vectors is obviously not the best practice.

The advantage of deep learning to do graph embedding is naturally that the ability of non-linear expression is stronger, and then an objective function that describes both local and global network information is designed, and the semi-supervised method is used to fit and optimize.

The network structure information of the graph is divided into local and global, which is the relationship between first-order neighbors and second-order neighbors. The second-order neighbors are obtained using an unsupervised autoencoder, and the context information of the nodes is restored. The first-order neighbors are obtained using a supervised method. The corresponding supervised samples are the directly connected nodes.

The network structure of the model is as follows

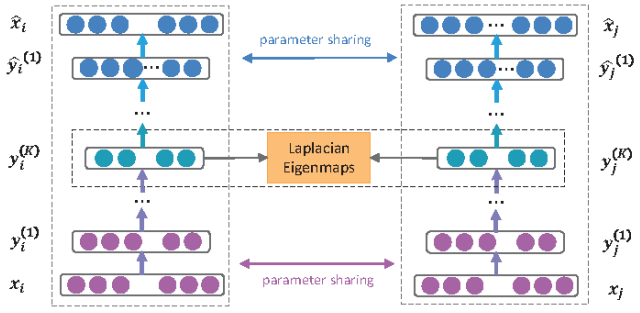


Figure 2: SDNE Model Framework

In SDNE, $x_i = A_i$ is the i-th row of the adjacency matrix A, which reflects the network structure vector of node v_i.

By implementing an automatic encoder on x_i, the learned code $h_i = y_i^{(K)}$ can maintain high-order dependencies. Because the autoencoder constrains the output $\hat{x}_i$ to be consistent with the input x_i, this enables the encoding h_i to reason about the node's context (network connection relationship), thereby maintaining high-order dependencies. It is not difficult to see that SDNE adds the constraints of the network structure on the basis of general automatic encoders. The constraint used here is the Laplacian Eigenmaps introduced earlier. Constraint nodes on the network have similar embedding results. Therefore, SDNE can maintain both first-order and higher-order approximations, and its loss function is:

$$\begin{aligned}\mathcal{L}_{mix} &= \mathcal{L}_{2nd} + \alpha\mathcal{L}_{1st} + \nu\mathcal{L}_{reg} \\ &= \left\|(\hat{X} - X) \odot B\right\|_F^2 + \alpha \sum_{ij}^{n} A_{ij} \left\|h_i - h_j\right\|_2^2 + \nu\mathcal{L}_{reg}\end{aligned}$$

Among them, $\mathcal{L}_{2nd}$ made some modifications to the original autoencoder loss, introduced a penalty term B, and imposed more penalties on the reconstruction error of non-zero elements than zero elements, making it easier for the model to reconstruct 1 in X instead of 0. For B, SDNE is defined as: if $A_{ij} = 0$, $b_{ij} = 1$, otherwise $b_{ij} = \beta > 1$. The starting point of Design B is that edges are connected to indicate that two points are similar, but no edges are connected, which does not necessarily mean that they are not similar. In this way, the structural information of higher-order neighbors can be reconstructed.

3.2 Experiments

Hyper-parameter tuning. The experiments compare various convex stochastic and incremental optimization methods . Some algorithms require tuning one or more hyperparameters such as the learning rate. We use grid search to find the best choice of the hyperparameters, we then choose the value of the hyperparameter that achieves the best objective function value within a given iteration budget.

We use Cross-validation a technique for model validation that assesses how the results of a statistical analysis will generalize to an independent data set. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in an estimate that generally has a lower bias than other such methods.

Analysis. In Table 2 are compare various convex stochastic and incremental optimization methods,the SVM based on Node2Vec methods converge faster in terms of training error while achieving a competitive performance in terms of the performance metric on a held-out set.

Model	Logistic	SVM	SGD	XGBoost
Deep Walk	1.14	1.13	**1.11**	1.17
Node2Vec	1.19	**1.04**	1.06	1.13
LINE	1.21	**1.15**	1.17	1.16
SDNE	1.26	1.24	**1.19**	1.22
SVD	1.25	**1.20**	1.22	1.26

Table 2: Experiment Results

4 Graph Neural Network

4.1 ConvGNN

ConvGNN [19] are divided into two major methods, the frequency-based methods and the spatial-based methods.

The frequency domain method is similar to the spectral graph theory. Assuming that A is an adjacency matrix, you can define a graph Laplacian matrix and perform eigenvalue decomposition

$$L = I_n - D^{-1/2}AD^{-1/2} = U\Lambda U^T$$

Where D is a diagonal matrix in which each element represents the degree of the corresponding node. Consider that each node has a value that constitutes an n-dimensional vector x. If we convolve it with another n-dimensional vector g, we have

$$X *_G g_\theta = U g_\theta U^T x \tag{1}$$

Where g is a filter. Different filters correspond to different operations in the frequency domain of the graph. It can be understood that the matrix U is equivalent to an orthogonal basis, and each vector of the matrix U specifies a value for each node on the graph, which is equivalent to a mode. The vector corresponding to the larger eigenvalue corresponds to a lower frequency mode. The above formula can be seen as first transforming x to the frequency domain, and after performing an operation on a certain frequency domain, then transform it back to the spatial domain. g can be controlled by parameters, and different parameter control methods can lead to different frequency domain ConvGNN. For example, g is directly represented by this parameter .

$$H_{:,k}^{(k)} = \sigma(\sum_{i=1}^{f_{k-1}} U\Theta_{i,j}^{(k)}U^T H_{:,k}^{(k-1)}) \quad j = (1, 2, ..., f_k) \tag{2}$$

Using Chebyshev polynomial or Cayley polynomial to approximate g we get

$$X *_G g_\theta = \sum_{i=0}^{K} \theta_i T_i(\tilde{L})x \tag{3}$$

$$X *_G g_\theta = c_0 x + 2Re\{\sum_{j=1}^{r} c_j(hL - iI)^j(hL + iI)^{-j}x\} \tag{4}$$

Spectral-based method has major limitations: the eigenvalue decomposition of the Laplacian matrix of the entire graph is needed. When the graph is very large, the time and space complexity of the method will be particularly high (although ChebNet And CayleyNet reduce the complexity by approximation, but they still need to get all the graphs for one-time processing); when there is a small change in any position in the graph, the entire eigenvalue system will change; the learnt filter only applies to this. A single graph cannot be generalized to other graphs; spectral-based is only applicable to undirected graphs, and cannot be processed for directed graphs or some more complex graphs (such as heterogeneous graphs with connected edges).The spatial-based method can overcome the above limitations. It does not need to perform eigenvalue decomposition. It only needs to know a node and its surrounding nodes to perform calculations, so it has higher computing efficiency, generalization performance, and scalability.

A first-order approximation (K = 1) based on ChebNet can get the well-known graph convolutional network (GCN)

$$X *_G g_\theta = \theta(I_n + D^{-1/2}AD^{-1/2})x \tag{5}$$

At the same time, GCN can also be expressed as a spatial-based form.

$$h_v = f(\Theta^T(\sum_{u \in \{N(v) \cup v\}} \bar{A}_{u,v}x_u)) \quad \forall v \in V \tag{6}$$

That is, in the calculation, only the nodes around each node need to be known.

4.2 Graph Attention Networks

Graph convolutional network require the use of pre-built graphs, which is problematic. In the Graph Attention Networks, each node in the graph can be assigned different weights according to the characteristics of its neighboring nodes. Reasonably, the structure is simple, moreover, it shows how powerful the attention mechanism really is. The GAT model can be applied to graph-based inductive and transductive learning problems.

The GAT is implemented by stacking graph attention layers. Here we describe the graph attention layer. The input of this layer is the set of node features:

$$\{\vec{h}_1, ..., \vec{h}_N\}, \vec{h}_i \in R^F$$

The output is a new set of node features

$$\{\vec{h}'_1, ..., \vec{h}'_N\}, \vec{h}'_i \in R^{F'}$$

In order to calculate the weight of each neighbor node, an $F \times F'$ shared weight matrix W is applied to each node, and then the attention coefficient can be calculated

$$e_{ij} = a(W\vec{h}_i, W\vec{h}_j)$$

This coefficient can represent the importance of node j relative to node i. The paper only calculates the neighbor nodes of node i, this is called masked attention. Then comes the normalized weight coefficient:

$$\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{exp(e_{ij})}{\sum_j exp(e_{ij})}$$

To improve the formula, a vector is $F' \times F'$, using LeakyReLU (α = 0.2):

$$a_{ij} = \frac{exp(\text{LeakyReLU}(\vec{a}^T))[W\vec{h}_i \parallel W\vec{h}_j]}{\sum_{k\in\mathcal{N}_i} exp(\text{LeakyReLU}(\vec{a}^T))[W\vec{h}_i \parallel W\vec{h}_j]}$$

This gives us the representation of node i:

$$\vec{h}'_i = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ij} W\vec{h}_i)$$

In order for self-attention to succesfully represent node i, a multi-headed attention mechanism is also used here, and the formula is further adjusted as:

$$\|_{k=1}^{K} = \sigma(\sum_{j\in\mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_i))$$

Using the above-mentioned multi-headed attention layer as the output layer directly is not suitable. One solution is to average K heads :

$$\vec{h}'_i = \sigma(\sum_{k=1}^{K} \sum_{j\in\mathcal{N}_i} \alpha_{ij}^k W^k \vec{h}_i)$$

A visual representation of the model is shown below.

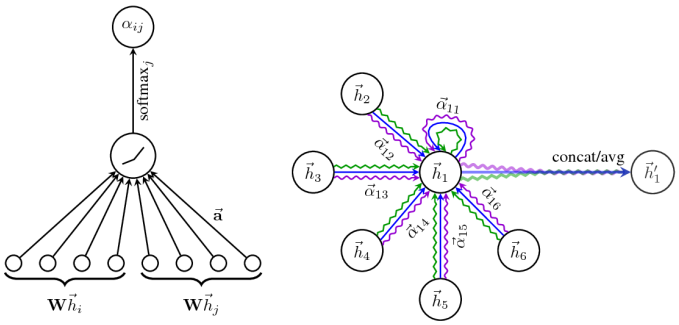


Figure 3: **Left**: The attention mechanism $a(W\vec{h}_i, W\vec{h}_j)$ employed by the model, parametrized by a weight vector $\vec{a} \in R^{2F'}$, applying a LeakyReLU activation.**Right**:An illustration of multi-head attention (with K= 3 heads) by node 1 on its neighborhood. Different arrow styles andcolors denote independent attention computations. The aggregated features from each head areconcatenated or averaged to obtain $\vec{h}'_1$.

4.3 SplineCNNs

[6] proposed spline-based convolutional neural networks (SplineCNNs), a variant of deep neural networks for irregularly structured and geometric inputs such as graphics or grids. The main contribution is a new type of convolution operator based on B-splines. Due to the local support of B-spline basis functions, the calculation time is independent of the kernel size. Therefore, we obtain the generalization of the traditional CNN convolution operator by using a fixed number of trainable weights to parameterize the continuous kernel function. Compared with related methods of filtering in the frequency domain, the proposed method aggregates features purely in the spatial domain. As a major advantage, SplineCNN allows the entire end-to-end training of deep architectures, using only geometry as input, rather than hand-crafted feature descriptors.

4.4 Experiments

Hyper-parameter tuning. The experiments compare various convex stochastic and incremental optimization methods . Some algorithms require tuning one or more hyperparameters such as the learning rate.We use grid search to find the best choice of the hyperparameters,we choose the value of the hyperparameter that achieves the best objective function value within a given iteration budget.

We use Cross-validation a technique for model validation that assesses how the results of a statistical analysis will generalize to an independent data set. It is commonly used in applied machine learning to compare and select a model for a given predictive modeling problem because it is easy to understand, easy to implement, and results in an estimate that generally has a lower bias than other such methods.

Model	TEXT Features	Node Features	Both
GCN	1.21	1.19	**1.16**
GAT	1.11	**1.06**	1.12
SplineCNN	1.24	**1.20**	1.21
ChebNet	**1.15**	1.17	1.20
AttentionWalk	1.25	**1.14**	1.20

Table 3: Experiment Results

Analysis. In Table 3 are compared various convex stochastic and incremental optimization methods,the GAT based on Node2Vec Features methods converge faster in terms of training error while achieving a competitive performance in terms of the performance metric on a held-out set.

5 Weighted arithmetic mean

With ensemble learning, we sometimes wish for one of the predictors to contribute more to a prediction, and perhaps other less skillful predictors that may be useful should contribute less to an ensemble prediction. A weighted average ensemble is an approach that allows multiple models to contribute to a prediction in proportion to their trust or estimated performance.

Weighted average ensembles allow the contribution of each ensemble member to a prediction to be weighted proportionally to the trust or performance of the member on a holdout dataset.

The contribution of each ensemble member is weighted by a coefficient α that indicates the trust or expected performance of the model. Weight values are small values between 0 and 1 and are treated like a percentage, in such a way that the weights across all ensemble members sum to one:

$$y_{pred} = (1-\alpha)\hat{y}_1 + \alpha\hat{y}_2$$

Where ,$\hat{y}_1$ is the output of the first model based on node embedding features and $\hat{y}_2$ the output of the second model based on TF-IDF features .

We use grid search to find the best choice of the hyperparameter,we choose the value of the hyperparameter that achieves the best objective function value within a given iteration budget.

α	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Score	1.09	1.06	1.03	1.00	0.99	**0.98**	1.07	1.09

Table 4: Experiment Results

References

[1] Charu C. Aggarwal and ChengXiang Zhai. A survey of text classification algorithms. In Charu C. Aggarwal and ChengXiang Zhai, editors, *Mining Text Data*, pages 163–222. Springer, 2012.

[2] Ralitsa Angelova and Gerhard Weikum. Graph-based text classification: learn from your neighbors. In Efthimis N. Efthimiadis, Susan T. Dumais, David Hawking, and Kalervo Järvelin, editors, *SIGIR*, pages 485–492. ACM, 2006.

[3] Smriti Bhagat, Graham Cormode, and S. Muthukrishnan. Node classification in social networks. *CoRR*, abs/1101.3291, 2011.

[4] Sandro Cavallari, Vincent W. Zheng, Hongyun Cai, Kevin Chen Chuan Chang, and Erik Cambria. Learning community embedding with community detection and node embedding on graphs. In *CIKM 2017 - Proceedings of the 2017 ACM Conference on Information and Knowledge Management*, International Conference on Information and Knowledge Management, Proceedings, pages 377–386. Association for Computing Machinery, 11 2017.

[5] Susan Dumais. Hierarchical classification of web content. pages 256–263. ACM Press, 2000.

[6] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. *CoRR*, abs/1711.08920, 2017.

[7] Aditya Grover and Jure Leskovec. Node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[8] David Liben-Nowell and Jon Kleinberg. The link-prediction problem for social networks. *J. Am. Soc. Inf. Sci. Technol.*, 58(7):1019–1031, May 2007.

[9] Aytug Onan. Classifier and feature set ensembles for web page classification. *Journal of Information Science*, 42, 06 2015.

[10] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. In *Proceedings of the 7th International World Wide Web Conference*, pages 161–172, Brisbane, Australia, 1998.

[11] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[12] Xiaoguang Qi and Brian D. Davison. Web page classification: Features and algorithms. *ACM Comput. Surv.*, 41(2):1–31, 2009.

[13] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, page 1067–1077, Republic and Canton of Geneva, CHE, 2015. International World Wide Web Conferences Steering Committee.

[14] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. 2008.

[15] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2017.

[16] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 1225–1234, New York, NY, USA, 2016. Association for Computing Machinery.

[17] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition. *CoRR*, abs/1801.07455, 2018.

[18] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. Personalized entity recommendation: A heterogeneous information network approach. In *WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, WSDM 2014 - Proceedings of the 7th ACM International Conference on Web Search and Data Mining, pages 283–292. Association for Computing Machinery, 1 2014.

[19] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *CoRR*, abs/1812.08434, 2018.

[20] Sven Meyer zu Eissen and Benno Stein. Genre classification of web pages.. In Susanne Biundo, Thom W. Frühwirth, and Günther Palm, editors, *KI*, volume 3238 of *Lecture Notes in Computer Science*, pages 256–269. Springer, 2004.