Graph-based Text Classification

WUSTL CSE Independent Study Report

Jerry Xing[1]

Marion Neumann[1]

[1]Department of Computer Science and Engineering,
Washington University in St. Louis,
MO 63130, USA

1. Introduction

Text classification, such as sentiment analysis and spam filtering, is one of the core tasks in natural language processing. Different from facial images, house prices and other objects that can be easily quantified and modeled as vectors, raw text lacks natural numerical representation, and thus cannot be directly used in most machine learning models designed for vector-like input. Therefore, finding a good numerical representation for text is very important. Traditional methods such as term frequency (TF) and term frequency–inverse document frequency (TFIDF) that model text as bag-of-words (BoW) have achieved great success, while ignoring neither the text structure (for example, word order and word dependencies) nor the semantic meaning of words. In recent years, recurrent neural network (RNN) based models overcome the two weakness by regarding text as sequences of words and introducing word vectors to capture the semantic meaning of each word. Moreover, long short term memory (LSTM) network enables RNN to capture long-term dependencies between words and LSTM-based algorithms have achieved most of the cutting-edge performance. LSTM, however, still has difficulty in capturing multiply types of word relationship (for example, word dependency and grammar relations) and very long word dependencies. On the other hand, graph is a more flexible and powerful model than BoW and sequential model, which models words as graph nodes and could capture word dependencies by adding edges among nodes. Text graph can also capture word semantic meaning by setting node vectors and node labels, and capture multiple types dependencies by setting edge weights and labels. But the flexibility of text graph also calls for effective algorithms to exploit the information in the graph and complete the classification task. In this project, we tested different methods to build text graphs and different graph classification algorithms on author identification task.

2. Text Graph

Graph is a powerful and widely used topology structure. The core components of graph are nodes and edges. Graph could also contain additional information such as node label, node attribute, edge weight and edge types. Node labels are usually categorical data, while node attributes are usually numerical vectors. Similarly, edge weights are numerical and edge types are categorical data. In text graph, we often set words as nodes and the relation or dependency between words as edges, and there are many choices for edges and nodes to build a text graph from raw text:

co-occurrence Edge Word co-occurrence is the most widely used method to build text graph edges. This method uses a fixed-length sliding window that scans through whole text. If two words appear in same window, then we can add an edge between nodes of the two words. And the edge weight can be set proportional to the total times when they appear in a same window. And a threshold can be set to filter out the edge between words that co-occurrence very few times, to avoid the graph becoming too dense to be classified.

Grammar and Dependency Edge Grammar edge is based on the grammatical structure of sentences. For example, a verb in a sentence and its object should be closely related, while they may be located far away from each other, making this relation hard be captured by co-occurrence sliding window. On the contrary, sliding window may also link two neighbouring words, which are usually of different grammar blocks. For instance, in the sentence ”The boy who lost his ball knocked my door last night”, ”ball”and ”knocked”are semantically very weakly related, even though they are neighbouring words. And as is shown in figure 1 and 2, if we only link the nodes that share parent node, ”ball”and ”knocked”would not be linked. Word dependency pays more attention on word semantic relations, as is shown in figure 3. Here we should note that there is no formal definition of word dependencies, and actually finding such relations itself is not a easy task. In our experiment, we use Stanford parser to generate the grammar tree of sentence, based on which we get grammatical relations. And we also used Stanford dependency parser to get the word dependencies

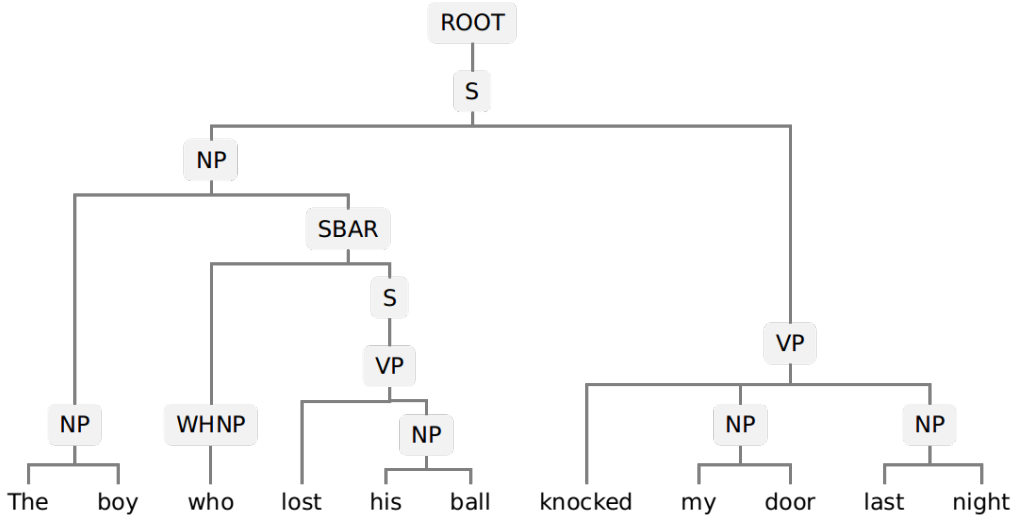

Figura 1. Parse tree of raw text (generated by Stanford parser)

Figura 2. Edges from parse tree by linking leaf nodes with same parent node

Node Labels and Attributes In text graph, nodes can be assigned with labels and attributes to add additional information for classification. In our experiment we used two types of node labels: named entity recognition (NER) label and part-of-speech (PoS) label.

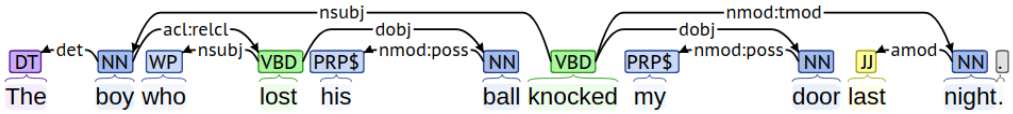

Figura 3. Word dependencies (generated by Stanford dependency parser)

NER label indicates the name of category of the words. For example, word ”apple”can have NER label as ”fruit”. PoS label use the part of speech of words, such as noun, verb, adjective, etc., as categorical node labels. In our experiment, we used Stanford 7-class NER tagger that can recognize location, person, organization, money, percent, date and time words. For the words that are not of the 7 classes, we labeled them as ”others”. Node attributes are often numerical vectors, and we use word vector which is generated by neural networks [Mikolov et al. 2013] as node attribute.

3. Graph Classification

Introduction There are four main graph classification approaches: graph mining, graph embedding, graph kernel and graph convolutional network. Graph mining based methods take the statistical characteristics of graphs (for example, average node degree and average node neighbor numbers) and create vectors based on these characteristics [Jiang et al. 2010]. This approach pays more attention on global structural information of graphs, and thus it is hard to extract local features and make use of additional information, such as node vectors and edge weights. Graph embedding take one more step in this direction. Graph embedding methods try to find a equivalent vector presentation of given graph, which keeps as much information as possible. After getting the vector representation, we can solve the classification task by traditional machine learning algorithms. Another method is graph kernel, which using graph kernel function to compute the similarity of graphs and use kernel machine, such as SVM, to implement the classification task. The fourth approach is GCN, which borrows idea from CNN and gets more and more attention in recent years. In the remaining part this section, we will introduce more about the last 3 methods and analyse their strengths and weakness.

Graph embedding Graph embedding, which is also often called network embedding, aims to encode the structure and node information into vectors. There are two types of graph embedding: node embedding (sub)graph embedding [Hamilton et al. 2017]. Node embedding aims to update a node’s information by its neighbors, and thus implicitly encode the local structure of the node’s neighbour into its vector. Note that this mechanism is very similar to that of RNN-based methods for NLP, where a word’s vector is updated by its former neighbor words. But after node embedding we still have graphs instead of vectors that can be used in traditional machine learning algorithms.

On the other hand, (sub)graph embedding aims to embed the whole graph as a vector. One way of this is based on node embedding. Since node embedding has encoded the graph’s structure information in the node attributes, we can ignore the graph representation and take the text as ”bag-of-nodes”, and generate a vector representation of the graph based on the updated nodes. One simple way to get the vector could be taking the average of the node attribute vectors.

We used this node embedding based methods. However, many popular node embedding algorithms such as LINE [Tang et al. 2015] doesn't take node vectors into consideration and thus doesn't apply yo our task. In our experiment, we used accelerated attributed network embedding (AANE) introduced by [Huang et al. 2017], which uses both network structure and node attributes.

Graph Kernel Kernel-based methods use graph kernel functions to compute the similarity of graphs, and use kernel machines, such as SVM, to classification. Based on how the graph structure is captured, we can distinguish graph kernels into 4 classes: kernels based on walks [Kashima et al. 2003] and paths [Feragen et al. 2013], kernels based on limited-size subgraphs [Kriege and Mutzel 2012], and kernels based on structure propagation [Markov et al. 2006]. In our experiment, we used propagation kernel, which is introduced by [Neumann et al. 2016] and can handle graph with node labels, node attributes and egde weights.

Graph Convolutional Network With the great success of neural network, there is a growing interest in designing neural networks on graphs. Based on the idea of CNN, many graph convolutional networks (GCN) are introduced to solve graph classification problem. Same as CNN, GCN can automatically extract features from graphs that often beat manually designed features used in graph embedding and graph kernels. The key for designing GCN in the convolutional layer and pooling layer, and different GCNs can have very different implementations. In our experiments, we used DGCNN introduced by [Zhang et al. 2018] which could make use of both network structure and node attributes.

4. Experiments

Corpus and Preprocessing In our experiment, we used the State of the Union Corpus dataset from [Tatman 2017], which includes full text of the state of the union address between 1989 and 2017. The texts are splitted into paragraphs and our task is to predict the speaker (i.e. which president) when given a paragraph. And paragraphs with fewer than 15 words were filtered out. In total we have 2207 pieces of text of 5 classes. 80% of them were used as training set and 20% as test set.

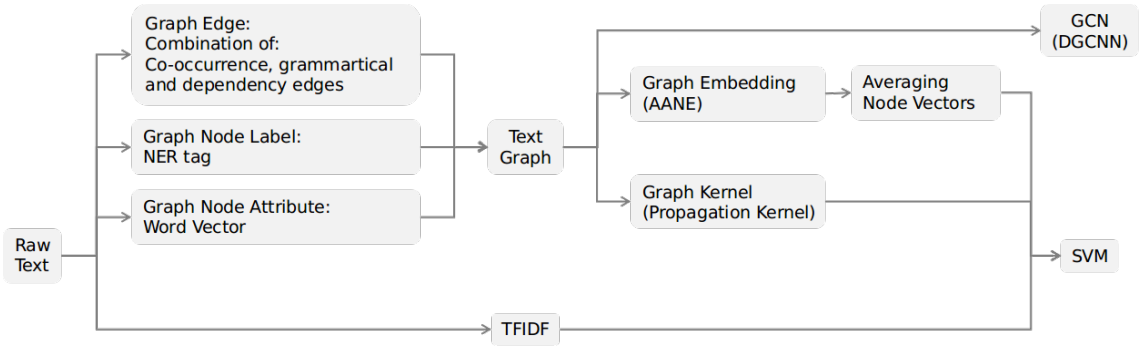

Figura 4. Experiment Design

Algorithms We test AANE, PK and DGCNN on the State of the Union corpus, as is shown in figure 4. Because kernel methods uses great amount of memory that our computer is hard to support, we only computed the kernel with 1000 paragraphs as training set. And we took the average of node vectors computed by AANE as graph vectors. We also did TFIDF for comparison. A SVM classifier is fitted with averaged AANE vector, TFIDF vector or propagation kernel, and the parameters of SVM were tuned by grid search and cross validation. The results are shown in table 2:

Tabela 1. Experiemnt Results with Graph Kernel

Edge Type	Model	Test Accuracy(%)
co-occurrence	PK	35.4
dependency	PK	33.8
grammartical	PK	34.2
co-occurrence+dep	PK	36.2
co-occurrence+grammartical	PK	34.6
co-occurrence+grammartical+dependency	PK	36.5

Tabela 2. Experiemnt Results of Other Methods

Edge Type	Model	Test Accuracy(%)
TFIDF	–	52.3
co-occurrence	AANE	39.0
dependency	AANE	38.8
grammartical	AANE	37.8
co-occurrence+dependency	AANE	38.6
co-occurrence+grammartical	AANE	36.9
co-occurrence+grammartical+dependency	AANE	38.8
co-occurrence	DGCNN	37.1
dependency	DGCNN	42.1
grammartical	DGCNN	39.0
co-occurrence+dependency	DGCNN	45.5
co-occurrence+grammartical	DGCNN	40.3
co-occurrence+grammartical+dependency	DGCNN	46.1

Conclusion From the experiment results, we can draw conclusion that:

1. TFIDF works better than all graph-based methods. Among the graph-based method, DGCNN has the highest test accuracy.
2. There's no best way to build text graphs for classification, as AANE works best with only co-occurrence edges, but DGCNN works best with all three kinds of edges.

Referências

Feragen, A., Kasenburg, N., Petersen, J., de Bruijne, M., and Borgwardt, K. (2013). Scalable kernels for graphs with continuous attributes. In *Advances in Neural Information Processing Systems*, pages 216–224.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*.

Huang, X., Li, J., and Hu, X. (2017). Accelerated attributed network embedding. In *SIAM International Conference on Data Mining*, pages 633–641.

Jiang, C., Coenen, F., Sanderson, R., and Zito, M. (2010). Text classification using graph mining-based feature extraction. In *Research and Development in Intelligent Systems XXVI*, pages 21–34. Springer.

Kashima, H., Tsuda, K., and Inokuchi, A. (2003). Marginalized kernels between labeled graphs. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 321–328.

Kriege, N. and Mutzel, P. (2012). Subgraph matching kernels for attributed graphs. *arXiv preprint arXiv:1206.6483*.

Markov, A., Last, M., and Kandel, A. (2006). Fast categorization of web documents represented by graphs. In *International Workshop on Knowledge Discovery on the Web*, pages 56–71. Springer.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Neumann, M., Garnett, R., Bauckhage, C., and Kersting, K. (2016). Propagation kernels: efficient graph kernels from propagated information. *Machine Learning*, 102(2):209–245.

Tang, J., Qu, M., Wang, M., Zhang, M., Yan, J., and Mei, Q. (2015). Line: Large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077. International World Wide Web Conferences Steering Committee.

Tatman, R. (2017). State of the union corpus (1989 - 2017). `https://www.kaggle.com/rtatman/state-of-the-union-corpus-1989-2017/`. Accessed April 4, 2018.

Zhang, M., Cui, Z., Neumann, M., and Chen, Y. (2018). An end-to-end deep learning architecture for graph classification. In *Proceedings of AAAI Conference on Artificial Inteligence*.