A Graph Convolution Network-based System for Technical Domain Identification

Alapan Kuila, Ayan Das and Sudeshna Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur,
Kharagpur, WB, India-721302
{alapan.kuila, ayan.das, sudeshna}@cse.iitkgp.ac.in

Abstract

This paper presents the IITKGP contribution at the Technical DOmain Identification (TechDOfication) shared task at ICON 2020. In the preprocessing stage, we applied part-of-speech (PoS) taggers and dependency parsers to tag the data. We trained a graph convolution neural network (GCNN) based system that uses the tokens along with their PoS and dependency relations as features to identify the domain of a given document. We participated in the subtasks for coarse-grained domain classification in the English (Subtask 1a), Bengali (Subtask 1b) and Hindi language (Subtask 1d), and, the subtask for fine-grained domain classification task within Computer Science domain in English language (Subtask 2a).

1 Introduction

Text classification is the task of assigning a category to a given piece of text based on its content from a predefined set of categories (Aggarwal and Zhai, 2012; Kowsari et al., 2019). It is a fundamental natural language processing (NLP) task with several downstream applications such as sentiment analysis, topic labeling and machine translation.

The ICON 2020 shared task on technical domain identification is essentially a text classification problem which involves identification of the technical domain in which a document belongs to a fixed set of domains. In this task, the participants are expected to develop a system that automatically identifies the technical domain of a given text (a small passage) in specified language.

We participated in the following subtasks:

1. **Subtask 1a:** Coarse grained domain classification in English.
2. **Subtask 1b:** Coarse grained domain classification in Bengali.
3. **Subtask 1d:** Coarse grained domain classification in Hindi.
4. **Subtask 2a:** Fine grained domain classification within Computer Science domain in English.

We applied PoS tagging and dependency parsing on the training and test data using PoS taggers and dependency parsers in the corresponding languages. We have used the PoS tags and the dependency relations of the tokens with their parent nodes in the dependency trees as features of the tokens in our system.

In the domain identification stage, the technical domain of a document is predicted from the representation of the document obtained using a GCNN, that takes the contextual representations of the constituent tokens of the document and a graph representation of the document as input. The contextual representations of the tokens are obtained by using a multi-layer bi-directional long-short term memory (LSTM) (Hochreiter and Schmidhuber, 1997; Schuster and Paliwal, 1997) that takes the representations of the tokens as input.

Corresponding to each subtask we submitted single runs of the systems.

2 Our Proposed Model

In this section, we will discus our proposed technical domain identification system in detail. The steps for predicting the domain of a given document are as follows.

S1. We partitioned the document into constituent sentences and tokenized each sentence into its constituent words.

S2. We PoS tagged and dependency parsed each sentence.

S3. We obtained the contextual representation of each token in the document by applying a multi-layer bi-directional LSTM on the representations of the tokens where we considered the entire document as a single sequence.

S4. We used the contextual representations of the tokens obtained in the previous step and the graph representation of the document as input to a GCNN to obtain the final feature representation of each word.

S5. We combined the final feature representations of the tokens to derive the document representation.

S6. We passed the document representation as input to a multi-layer perceptron (MLP) followed by a softmax layer to predict domain of the document.

2.1 Partitioning of Document into Sentences and Tokenization

We derived the sentences from the documents by partitioning the document on the following characters: ".", "?" and "!". The sentences were tokenized based on spaces. The tokens are essentially the space separated word in the sentences.

2.2 PoS Tagging and Dependency Parsing of Sentences

Before training or testing we PoS tagged and dependency parsed the sentences. We PoS tagged and parsed the English sentences in the subtasks *1a* and *2a* using the SpaCy library (Honnibal and Johnson, 2015; Honnibal and Montani, 2017). The Bangla and Hindi sentences for the subtasks *1b* and *1d* respectively were PoS tagged and dependency parsed using a LSTM based sequence tagger (Qi et al., 2018) and a bi-affine graph based dependency parser (Dozat et al., 2017). The Bangla tagger and parser were trained using a Bangla treebank developed in our institute. The Hindi tagger and parser were trained using the Universal Dependencies v2.0 Hindi treebank (Nivre et al., 2016).

2.3 Generation of Contextual Representation of Tokens

We obtained the contextual representation of the tokens in a document by passing the distributed representations of the tokens in the document as input to a 3-layer bi-directional LSTM. Here we treated the entire document as a sequence. Each token is represented as the concatenation of the embedding of the token, the embedding of its PoS tag, the embedding of its dependency relation with its parent in the dependency tree of the sentence, and, the character-level representation of the word obtained by applying a convolutional neural network (Goodfellow et al., 2016) on the embeddings of the constituent characters of the token.

All the embeddings were randomly initialized and updated during training phase.

2.4 Graph Convolution Neural Network to Generate the Document Representation

In this section, we discuss the implementation of GCNN (Kipf and Welling, 2016) used in our system.

2.4.1 Input Layer

The input to the the GCNN comprises of the contextual representations of the tokens in the document and the directed graph representation of the document in the form of an adjacency matrix.

2.4.2 Graph Construction

The number of nodes $|V|$ in the document graph is the total number of tokens contained in the document and the graph edges are represented by various intra- and inter-sentence dependency edges which are discussed below.

- **Syntactic dependency edges:** The edge set comprises of the edges between all token pairs that are linked by head-dependent relations in the dependency tree of the corresponding sentence. To consider that the information flows in both forward and backward directions of syntactic dependency arcs, we included both forward and backward edges in the graph edge set. More precisely, if there is an arc from a token t_i to a token t_j in the dependency tree, then $A(i, j) = 1$ and $A(j, i) = 1$.

- **Adjacent sentence edges:** In order to keep sequential information flows through the consecutive sentences we also connected the root nodes of the neighbouring sentences. Therefore, $A_{adj}(i, j) = 1$ and $A_{adj}(j, i) = 1$, if the tokens t_i and t_j are the root nodes of the consecutive sentences.

- **Self-node edges:** We also include self node edges in the graph. Therefore, for all the tokens t_i in the document, $A_(i, i) = 1$.

2.4.3 GCNN Layer

GCNN is an advanced version of CNN operating on graphs that induce the node features based on the properties of their neighbouring nodes. GCNN with one layer of convolution can capture information of only immediate neighbours. When multiple GCNN layers are stacked, information from larger neighbourhoods are accumulated. Let A be the adjacency matrix of the text-graph. A contains the edges as stated in Section 2.4.2 and the adjacency matrix is of the dimension $|V| \times |V|$, where $|V|$ is the number of tokens in the document. After stacking k GCNN layers we get the adjacency matrix for the k-th-order text graph A^k, where, $A^k = (A)^k$. A^k holds all the k-hop paths in the text-graph. The ith word representation of the $(k+1)$-layer text-graph (A^k) is calculated as:

$$h_i^{k+1} = g(h_i^k, A^k)$$

where, $g(.)$ refers to the graph convolution function and $+$ indicates element wise summation operation. The function $g(.)$ is defined as:

$$g\left(h_i^k, A^k\right) = \sigma(\sum_{j=1}^{|V|}(A^k(i,j)(W_A^k * h^k + b_A^k)))$$

Here, σ indicates the ReLU activation function. W_A^k and b_A^k are the weight matrix and bias item for A^k.

The initial value of the representation of the i^{th} token h_i^0 is the contextual representation of the token t_i from the output of the LSTM (Section 2.3).

In our system we have trained a stacked GCNN of 3 blocks. The model architecture is shown in Figure 1.

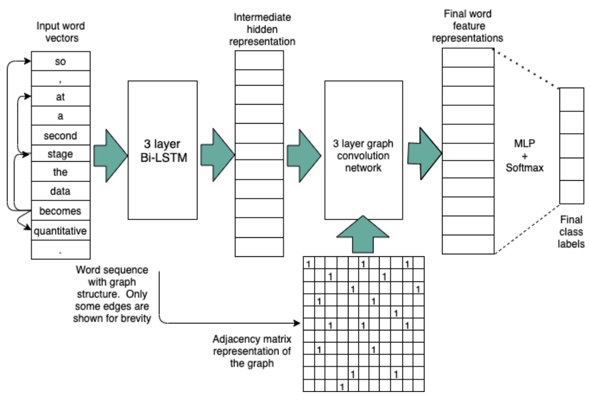
Input word vectors
so
,
at
a
second
stage
the
data
becomes
quantitative
.
3 layer Bi-LSTM
Intermediate hidden representation
3 layer graph convolution network
Final word feature representations
MLP + Softmax
Final class labels
Word sequence with graph structure. Only some edges are shown for brevity
Adjacency matrix representation of the graph

Figure 1: Block diagram of our proposed model

Sub-task	Classes	# of docs.		
		Train	Dev	Test
1a	cse, che, physics, law, math	23962	4850	2500
1b	bioche, mgmt, com_tech, phy, cse	58500	5842	1921
1d	other, mgmt, phy, com_tech, cse, math, bioche	148445	14338	4211
2a	ca, se, algo, cn, pro, ai, dbms	13580	1360	1929

Table 1: Statistics of datasets for the different subtasks

Training settings	
Loss function	Cross-entopy over the domain classes
Optimizer	Adam (lr=1e-4, eps=0.1)
System settings	
Token embeddings	140
PoS embeddings	20
Relation embeddings	20
Character embeddings	20
# of char CNN kernel	3
# of char CNN filter	30
LSTM layers	3
LSTM hidden size	100
GCNN stack	3
GCNN size	500, 200, 200
Dropout rate	0.25

Table 2: System settings and hyper-parameters

2.5 Document Representation and Domain Prediction

A representation of the document is obtained as the point-wise mean of the token representations obtained as the output of the GCNN (Section 2.4). Finally, the document representation is passed as input to a MLP layer followed by a softmax layer. The output of the softmax layer is a probability distribution over the possible domain classes. The class label with the highest probability value is predicted as the class of the document.

3 Data Description

The statistics of the data corresponding to the different subtasks in which we have participated are summarized in Table 1.

Subtask	Accuracy		Precision		Recall		F1-score	
	GCN	Best	GCN	Best	GCN	Best	GCN	Best
Subtask-1a	0.6564	**0.8156**	0.6850	**0.8155**	0.6564	**0.8156**	0.6560	**0.8144**
Subtask-1b	0.6878	**0.8335**	0.7432	**0.8420**	0.7023	**0.8515**	0.6897	**0.8353**
Subtask-1d	0.4656	**0.6117**	0.4706	**0.6478**	0.4523	**0.5989**	0.4523	**0.6044**
Subtask-2a	0.7029	**0.8252**	0.7021	**0.8265**	0.7034	**0.8252**	0.7008	**0.8244**

Table 3: Performance of our system for the different subtasks and comparison with overall best performing systems. The column heading *GCN* indicates *our system* and *Best* indicates overall best performing system.

4 Training Setup

In this section we present the settings and hyper-parameters used in our domain prediction system. In Table 2 we summarize the system settings.

5 Results

In this section, we present the performance of our systems corresponding to the different subtasks. In Table 3 we present the performance of our systems in terms of their accuracy, precision, recall and F1-score on the test data and compare them with the best results achieved for that subtask.

6 Error Analysis

We have inspected the outputs of our system on the development set. Thorough error analysis has revealed several type of errors. Aside from the errors propagated from the PoS tagger and dependency parser, some of the errors generated in the output are discussed bellow:

1. As some of the classes are very related and lots of common terms exist in some of these classes, the system has confused to detect the correct class.

    - **S1:** and so we have shown e i square of x is e i of x for all x .
    - **S2:** so , grad f at r t naught is a vector , whose dot product with any tangent vector is 0.

    These instances are from Subtask 1a. S1 is detected as *physics* (actual class: *math*) where as S2 is detected as *math* (actual class: *physics*). Same errors happen in case of Subtask 1b for classes: *cse* (Computer Science) and *com_tech*(Communication Technology)

2. Most of the candidate texts are single sentences. Sometimes system has failed to identify the correct class as it could not understand the contextual information from those single sentences. For example:

    - **S3:** Why that is what is the reason why?

System has failed to classify the domain of these sentences properly. From these errors, it is evident that the problem of domain selection (among highly related classes) is really a challenging task. The accuracy and F-1 score of our system are also lower than the best system submitted. In future we will apply some other data processing techniques and smarter machine learning models to improve our performance.

References

Charu C. Aggarwal and ChengXiang Zhai. 2012. A survey of text classification algorithms. pages 163–222.

Timothy Dozat, Peng Qi, and Christopher D. Manning. 2017. Stanford's graph-based neural dependency parser at the CoNLL 2017 shared task. In *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 20–30, Vancouver, Canada. Association for Computational Linguistics.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Matthew Honnibal and Mark Johnson. 2015. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1373–1378, Lisbon, Portugal. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura E. Barnes, and Donald E. Brown. 2019. Text classification algorithms: A survey.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the 10th International Conference on Language Resources and Evaluation (LREC 2016)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association.

Peng Qi, Timothy Dozat, Yuhao Zhang, and Christopher D. Manning. 2018. Universal dependency parsing from scratch. In *Proceedings of the CoNLL 2018 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*, pages 160–170, Brussels, Belgium. Association for Computational Linguistics.

M. Schuster and K. K. Paliwal. 1997. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.