UNIVERSITY OF PADUA

INFORMATION ENGINEERING DEPARTMENT (DEI)

MASTER'S DEGREE IN COMPUTER ENGINEERING

Comparing Graph Neural Network Performances Against Standard Natural Language Techniques

Prof.
Fabio Vandin

Students:
Francesco Caldivezzi : 2037893
Simone D'Antimo : 2052413
Riccardo Rampon : 2052416

Contents

1 Introduction

Text classification is an important and classical problem in **Natural Language Processing**. The goal in Text Classification task is to build systems which are able to automatically classify any kind of documents performing different operations such as **document organization**, **news filtering**, **spam detection**, **opinion mining**, and **computational phenotyping**.
The **Feature Space**, based on the set of unique words in the documents, is typically of very high dimension, and thus document classification is not trivial.
In the past few years, a lot of research, and progress, has been made, thanks to Text Classification Methods such as **Recurrent Neural Network (RNN)**, **Convolution Neural Network (CNN)** and with libraries like **FastTex**, while, only a limited number of studies have explored the more flexible **Graph Convolutional Neural Networks (GCNN)** [6].
In this paper the goal is to compare the effectiveness of classical techniques for **Text Classification** with respect to **Graph Neural Networks (GNN)**. Comparisons, will be done, by using different datasets as well as different tran-validation dataset-split. The aim of this experiment is to explore the potential of the **Graph Neural Networks** compared with classical techniques [5] like : **Rocchio Classifier**, **Support Vector Machine**, **Recurrent Neural Network**, **Recurrent Convolutional Neural Network**, etc., and, find out if using a **Graph Neural Network** solution is better, in terms of performances, than the other methods in this particular scenario .

2 Datasets

To carry out the experiments three different Datasets will be used. Each one, has a different number of classes and entries, in this way, we can test our models in different scenario in order to evaluate in all the possible cases. In particular, the Dataset [2] used, is the combination of three Datasets that are called, respectively : **R8**, **R52**, **Ohsumed** and, have the following characteristics :

- **R8** and **R52** : are subsets of **Reuters** [3] 21578 Dataset which is composed of 12902 documents with 90 classes.
- **Ohsumed** : is a dataset [4] built by excrating medical abstracts from the **MEDLINE** database. In particular, it consists of medical abstracts from the **MeSH** categories of the year 1991 and provides 23 cardiovascular diseases categories as classes.

Dataset	# Train	# Test	# Categories	Avg. Length
R8	5485	2189	8	65.72
R52	6532	2568	52	69.82
Ohsumed	3357	4043	23	135.82

For each Dataset, the table above displays the number of train/test entries, categories, and average length of entries.

3 Methods

The methods used for solving the Text Classification problem in this paper are :

- Rocchio Classifier [5]
- Bagging Classifier [5]
- Boosting [5]
- Support Vector Machine [5]
- Convolution Neural Network [5]
- Recurrent Neural Network [5]
- Recurrent Convolutional Neural Network [5]
- Graph Convolutional Neural Network [6]

As is the case with many NLP models, our models, except the Graph Convolutional Neural Network, use word embeddings as inputs to determine the meaning and semantics of words. Rather than building our own word embedding model, we use a pre-trained one called "**GloVe**" (Global Vectors for Word Representation). In particular we use the "Glove.6B.50d.txt" file which contains the pre-trained word embeddings model that has been trained on a dataset of 6 billion words and has a vocabulary of 400,000 words. Each word in vocabulary is represented by a 50 dimensional vector.

3.1 Rocchio Classifier

Rocchio Classifier [5] is implemented using a Nearest Centroid Model with the euclidean metrics.

3.2 Bagging Classifier

Bagging Classifier [5] model use 10 KNeighborClassifier with uniform weights as estimators. Each estimator use 5 neighbors for the kneighbors query.

3.3 Boosting

Boosting [5] model has 100 estimators.

3.4 Support Vector Machine

Support Vector Machine (SVM) [5] model chosen is the standard linearSVC model.

3.5 Convolutional Neural Network

The **Convolutional Neural Network** [5] model used has a first embedding layer that converts integers representing a text input into dense vectors, as for RNN and RCNN the input length is 500. Then there is two Convolutional layers, each of them has 128 filters and a kernel size equals to 5. Each convolutional layer is followed by a Dropout layer used to reduce overfitting in the model with a rate of 0.5. Both the dropout layers are followed by a max-pooling layer that reduces the dimensionality of the output of the convolutional layer; the first one has a size equal to 5, the second one size is 30. The final Dense layers reduces in three steps the number of units, first dense layer brings the units to 1024, the second one reduce them to 512, and the last produces as output the probabilities for each class using the softmax function as activation function. The model also has an early stopping callback, which stops the training process if the validation loss does not improve for a certain number of epochs. The model is trained using stochastic gradient descent with an Adam optimizer and categorical cross-entropy loss.

3.6 Recurrent Neural Network

Recurrent Neural Network [5] model used has a first embedding layer that converts integers representing a text input into dense vectors. Then we have 4 GRU (Gated Recurrent Units) layers, each of them maintains an internal state that encodes information about the input seen so far. Each GRU layer has 256 units and is followed by a Dropout layer with a 0.2 rate, used to reduce overfitting in the model. The final layer produces a 2D Tensor with X outputs units where X is the number of classes. The model also has an early stopping callback, which stops the training process if the validation loss does not improve for a certain number of epochs.

3.7 Recurrent Convolutional Neural Network

The **Recurrent Convolutional Neural Network** [5] model combine the RNN and CNN techniques in order to take advantage of both of them. Our model has a first embedding layer that converts integers representing a text input into dense vectors. After the embedding layer there is a dropout layer with a 0.25 dropout rate. Then, after this first embedding parts there are the "Convolutional layers" that consits of 4 layers of 1D Convolution with 256 filters and kernel size equals to 2, alternated with 1D MaxPooling layers with pool size equals to 2. Next, there are the "Recurrent layers", in particular, we used LSTM layers with 256 units, with a own Dropout rate of

0,25, in order to be able to capture long-term dependencies. In the end, two Dense layers produce as output the class probabilities for the input, using the softmax function as activation function. The first Dense layer reduces the output to 1024 units, while, the second one to the number of classes of the dataset. The model also has an early stopping callback, which stops the training process if the validation loss does not improve for a certain number of epochs.

3.8 Graph Convolutional Neural Network

The **Graph Convolutional Neural Network** [6], is constructed using the Deep Graph Library, in particular from a corpus of text documents, with nodes representing either individual words or documents. We create edges between nodes based on word co-occurrences in the documents and across the entire corpus. The weight of an edge between a document node and a word node is calculated using the term frequency-inverse document frequency (TF-IDF) of the word in the document instead of the GloVe approach used for other methods. To incorporate global word co-occurrence information, a sliding window of size equal to 20 is used to gather co-occurrence statistics, and the pointwise mutual information (PMI) measure is used to calculate the weights between word nodes. The embedding size of the first convolutional layer is 200. The train process is made using backpropagation to compute the gradient of loss function, then we use the Adam optimizer with a learning rate equal to 0.02, we repeat this process for 100 epoch or until the early stopping callback is called.

4 Results

In this section we report the **results** [1] that we have obtained. In particular, in the following we will list the values of **Precision**, **Recall**, **F1-Score**, **Accuracy** obtained across the different models as well as across the different datasets. In addition to that, we are also going to list two main sets of results due to the choice of splitting the dataset as follow :

- 70% used for Training and Validation, which was splitted as well as, into two main ways :
  - Train 80% and Validation 20%.
  - Train 70% and Validation 30%.
- 30% used for Testing.

In the following, we will list the results obtained :

- Train 80% and Validation 20%:
  - **Precision** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.46	0.87
Boosting	0.95	0.64	0.89
Convolutional Neural Network (CNN)	**0.97**	0.75	0.96
Recurrent Convolutional Neural Network (RCNN)	0.95	0.03	0.91
Recurrent Neural Network (RNN)	**0.97**	0.74	**0.98**
Rocchio Classifier	0.85	0.38	0.79
Support Vector Machine (SVM)	0.92	0.46	0.85
Graph Convolutional Neural Network (GCNN)	0.95	**0.85**	0.90

  - **Recall** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.47	0.87
Boosting	0.95	0.62	0.89
Convolutional Neural Network (CNN)	**0.97**	0.70	0.95
Recurrent Convolutional Neural Network (RCNN)	0.95	0.17	0.91
Recurrent Neural Network (RNN)	**0.97**	0.74	**0.98**
Rocchio Classifier	0.85	0.27	0.64
Support Vector Machine (SVM)	0.92	0.48	0.86
Graph Convolutional Neural Network (GCNN)	0.96	**0.85**	0.93

- **F1-Score** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.46	0.86
Boosting	0.95	0.62	0.89
Convolutional Neural Network (CNN)	**0.97**	0.72	0.95
Recurrent Convolutional Neural Network (RCNN)	0.95	0.05	0.91
Recurrent Neural Network (RNN)	**0.97**	0.74	**0.98**
Rocchio Classifier	0.81	0.28	0.70
Support Vector Machine (SVM)	0.92	0.44	0.85
Graph Convolutional Neural Network (GCNN)	0.96	**0.85**	0.91

- **Accuracy** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.47	0.87
Boosting	0.95	0.62	0.89
Convolutional Neural Network (CNN)	**0.97**	0.70	0.95
Recurrent Convolutional Neural Network (RCNN)	0.95	0.17	0.91
Recurrent Neural Network (RNN)	**0.97**	0.74	**0.98**
Rocchio Classifier	0.80	0.27	0.64
Support Vector Machine (SVM)	0.92	0.48	0.86
Graph Convolutional Neural Network (GCNN)	0.96	**0.85**	0.93

• Train 70% and Validation 30%:

- **Precision** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.60	0.87
Boosting	0.94	0.61	0.89
Convolutional Neural Network (CNN)	0.97	0.66	**0.95**
Recurrent Convolutional Neural Network (RCNN)	0.95	0.65	0.90
Recurrent Neural Network (RNN)	**0.98**	0.72	**0.95**
Rocchio Classifier	0.85	0.38	0.81
Support Vector Machine (SVM)	0.92	0.47	0.85
Graph Convolutional Neural Network (GCNN)	0.96	**0.82**	0.85

- **Recall** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.60	0.87
Boosting	0.94	0.60	0.89
Convolutional Neural Network (CNN)	0.97	0.51	**0.95**
Recurrent Convolutional Neural Network (RCNN)	0.95	0.64	0.90
Recurrent Neural Network (RNN)	**0.98**	0.71	**0.95**
Rocchio Classifier	0.79	0.27	0.63
Support Vector Machine (SVM)	0.92	0.47	0.86
Graph Convolutional Neural Network (GCNN)	0.96	**0.82**	0.89

- **F1-Score** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.60	0.86
Boosting	0.94	0.61	0.89
Convolutional Neural Network (CNN)	0.97	0.53	**0.95**
Recurrent Convolutional Neural Network (RCNN)	0.95	0.63	0.89
Recurrent Neural Network (RNN)	**0.98**	0.71	**0.95**
Rocchio Classifier	0.81	0.28	0.69
Support Vector Machine (SVM)	0.92	0.43	0.85
Graph Convolutional Neural Network (GCNN)	0.96	**0.82**	0.87

- **Accuracy** :

Model	**R8**	**OH**	**R52**
Bagging	0.93	0.60	0.87
Boosting	0.94	0.62	0.89
Convolutional Neural Network (CNN)	0.97	0.51	**0.95**
Recurrent Convolutional Neural Network (RCNN)	0.95	0.64	0.90
Recurrent Neural Network (RNN)	**0.98**	0.71	**0.95**
Rocchio Classifier	0.79	0.27	0.63
Support Vector Machine (SVM)	0.92	0.47	0.86
Graph Convolutional Neural Network (GCNN)	0.96	**0.82**	0.89

5 Conclusions

The results obtained are the following :

- As expected the Graph Convolutional Neural Network (GCNN), which is the state of the art in text classification problem, obtained on average the best performances compared to the other methods.
- If we consider the results on the single dataset, however, Convulutional Neural Network (CNN) and Recurrent Neural Network (RNN) are the ones that performed better, even if slightly, on the datasets R8 and R52, while on the OH dataset the GCNN performed significantly better over the other models of roughly 15%.
- An other result as expected is the one that consist of analyzing the performances by changing the size of the training-validation percentages. In particular, what has been obtained is that reducing the size of the training set, results has decreased of roughly 5%. In particular, in OH dataset, in which the average length of the phrases are the longest one, we notice a decrease of performances in this change of size between training ad validation set.

Finally, the GCNN is the method that adapts better not only because of its performances but also because of its training time. In fact, for training the GCNN in the worst case scenario only 20 minutes where required, while, for others top performance method (RNN, RCNN) in the worst case it took between 3 and 3 and an half hours for training. Notice that those results where computed with same CPU and GPU.

6 Contributions

This work of this project was divided as follow :

- Simone has developed : Rocchio, Bagging, Boosting and Svm methods, and has written : Introduction, Methods, Results and Conclusion of the report.
- Francesco has developed : RNN and GCNN methods, and has written : Dataset, Results and Conclusion of the report.
- Riccardo has developed : CNN and RCNN methods, and has written : Dataset, Results and Conclusion of the report.

Notice that, to have a better overview on the contributions and who has done what please see the repository [1].

7 References

[1] Github website where there is the code of this project. `https://github.com/caldi99/Learning-From-Network-Project`.

[2] R8, r52 and ohsumed datasets. `https://www.kaggle.com/datasets/weipengfei/ohr8r52`.

[3] Reuters-21578 text categorization collection data set. `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`.

[4] Text categorization corpora. `http://disi.unitn.it/moschitti/corpora.htm`.

[5] Kamran Kowsari, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. Text classification algorithms: A survey. *Information*, 10(4):150, 2019.

[6] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.