Project Report on
Automated Fake News Detection

First Semester 2024-25

CS F425: Deep Learning

Under the Guidance of:

Prof. Aneesh Sreevallabh Chivukula

DATE: 19/12/2024

Group 13 Members	
Name	**Campus ID**
Aryan Gupta	2021A7PS0162H
Rohan Chavan	2021A7PS2739H
Shrey Paunwala	2021A7PS2808H

Contents

ABSTRACT

The proliferating fake news poses a significant challenge to information integrity across all platforms. The need for automated fake news detection arises due to the humongous scale and virality of misinformation. This project aims to compare suitable deep learning models and methods with the existing literature to find the optimal method for fake news detection. We use multiple Natural Language Processing techniques, like part of speech tagging and dependency analysis in conjunction with Deep Neural Networks like LSTM and CNN models. These models are tested and compared with literature using other methods, like SVM and Random Forests. We find the results of deep learning approach to be superior with the dataset's original 6-class labelling. This project adds to the original paper (associated with the dataset) as the paper was not released with corresponding code. The results can be improved upon by further studies into incorporating other architectures and techniques into the model.

INTRODUCTION

Motivation & Background

The widespread reach of social media and biased news means it is much easier to get access to and spread misinformation. This combined with the rise of AI generates a volume of incorrect information, which is impossible to fact check manually before it spreads to a vast audience.

Automated detection is an efficient and scalable solution to these problems.

Problem Statement

This project aims to use multiple Natural Language Processing techniques in conjunction with Deep Neural Networks like LSTM and CNN models and compare their performance with non-deep learning models as described in other papers.

Objectives

1. Preprocess the LIAR dataset using NLP techniques and find their effectiveness. NLP techniques: Part-of-Speech Tagging, Dependency Parsing
2. Train LSTM and CNN models with hyperparameter tuning.
3. Compare the LSTM and CNN model results.
4. Compare the best results of the project with companion projects using non-neural approaches.

Challenges

1. Due to the rapidly evolving AI models, the model will quickly become outdated and will need to be retrained.
2. “Fakeness” of News can be subjective and change with time.
3. Deep Learning models lack explainability. It is hard to interpret how they make decisions.
4. Due to the enormous number of deep learning architectures and NLP techniques present, the project will miss out on many of them.

Approach Overview

1. Pre-Processing: Check the dataset for inconsistencies, remove stop words, count vectorize string content.
2. NLP: Use Part-of-Speech tagging and dependency parsing to add more information from the content.
3. Use GloVe (Global Vectors for Word Representation) 100 dimension pretrained embeddings as our embedding layer.
4. Define LSTM and CNN models using keras, we use dropout for regularization.
5. Train the models using tensorflow.
6. Test the models with various hyperparameters and NLP columns.

RELATED WORK

- Wang (2017), in the original paper introducing the LIAR dataset used in this project, tests out various Models on the dataset. The proposed solutions include baseline majority with 20.4%, SVMs with 25.5%, Logistic Regression with 24.7%, Bidirectional LSTMs with 22.3% and CNNs with 26% accuracy on validation data. The best Accuracy of 27.4% on testing data is achieved with Hybrid CNNs using all the features. Hybrid CNN is the combination of convolution and bidirectional LSTM. The meta data is fed into LSTM layers, while the word embeddings are run through a normal CNN network. We will not be building a hybrid model. This paper doesn't provide us with the implementation of these solutions; thus, our report helps the space in finding one.
- Mansoor9743 (2019) uses the following models with corresponding accuracies for classification: Logistic Regression with 24.7%, multinomial Naïve Bayes with 24.8%, linear kernel SVM with 23.3%, decision tree with 23.2%, Random Forest with 22% and XGBoost with 24.4%. He uses GridSearchCV for hyperparameter tuning, which is useful in non-deep models with lower training time and fewer hyperparameters. It becomes impractical for deep networks.
- Kim (2014) explores the usage of CNNs for sentence classification trained on top of pre-trained word vectors for sentence-level classification. This study ensures the viability of CNNs for our objectives, and gives a starting ground for building the model.
- Zhang et al. (2012) use SVM, KNN and Naïve Bayes architectures on a different data set to achieve ~70% accuracy on each in testing. This result is on a binary classification problem, conducted on a different dataset, thus only used for reference purposes.

DATASET & FEATURES

Dataset Description

The original file (source) is in TSV format. The description of the columns are as follows (as given in the source).

1. The ID of the statement ([ID].json).
2. The label.
3. The statement. The content to judge for falseness.
4. The subject(s).
5. The speaker.
6. The speaker's job title.
7. The state info.
8. The party affiliation.

9-13. The total credit history count, including the current statement.

9. Barely true counts.
10. False counts.
11. Half true counts.
12. Mostly true counts.
13. Pants on fire counts.
14. The context (venue / location of the speech or statement).

The method to obtain full verdict report for any statement is outlined in Code/Data/README.txt.

Feature Engineering

We convert all categorical data to numerical using frequency tokenization. We do not use one hot or dummy encoding, since these categories are usually in the hundreds. We also assign the same number; i.e. the last value in the frequencies to values which occur sparsely (< 1%) in the data. We chose against TF-IDF, since the sentences are short. We did not use a transformer, since we will be employing an embedding layer. All this data is referred to as the meta data in the following report.

For the statements, we start by removing stop words using Python's nltk library. In addition to count tokenization, we perform Part of Speech Tagging and Dependency parsing using Python's spacy module.

Data Splitting

The data is divided into 80-10-10 split where 80% is used for training, and 10% each for training and testing. This is done to reduce some noise in the comparison between the results of other papers, as this is the split recommended in the original paper.

METHODOLOGY

Task Definition

Identify the best model and feature combination to predict the falseness of statements.

Model Selection

We choose Bi-directional LSTM and text-CNN models to use in our project. These were chosen since our task involves working with long text sequences, these deep models are also parallelizable, making their training and testing faster. This allows us better control over hyper-parameters, easy to alter models, and to perform multiple experiments in these architectures.

Text-CNN is an application of CNN architecture using a 1D convolution layer. This allows us to apply the strengths of CNN in capturing spatial patterns to capture sequential patterns in text.

Model Description

Both models employ ReLU activation in the hidden layers and softmax in the output layer. We choose dropout as our regularization method. Sentence Input layer is passed through a pretrained Embedding Layer, before being passed to rest of the layers.

Hyperparameter Tuning

We use the validation dataset with cross validation to tune the hyperparameters. The hyperparameters are: initial learning rate, clipping value.

Other Parameters

Parameters like the number and size of layers, the number of input features to use (out of part of speech tags [pos], dependency parse [dep], and rest of the categorical features [meta]) are not trained using cross validation, and are instead tested manually.

Training Strategy

We use a pre-trained embedding layer from GloVe (source). For training the models, we use Momentum based Stochastic Gradient Descent algorithm, called the Nesterov SGD Algorithm. This is used in conjugation with a clipping value to prevent exploding and vanishing gradients. We use categorical cross entropy as our loss function, since we are dealing with multi class output.

For even faster compute, we employ parallelized training and predictions using local GPU hardware.

Implementation Details

The spacy library is used in conjugation with NLTK package for all NLP tasks. We use Pandas for data manipulation and Pickle to store and load trained models. Tensorflow package and Keras library are used for creation, training and testing of all models.

The following is the workflow of the project:

1. Pre-Processing:
    - Check the dataset for inconsistencies; i.e missing, corrupted or non-ascii values.
    - Remove stop words
    - Count vectorize string content.
2. NLP
    - Part-of-Speech tagging
    - Dependency parsing

3. Use GloVe (Global Vectors for Word Representation) 100 dimension pretrained embeddings as our embedding layer.
    - Download the embeddings if not available.
4. Define CNN model
    - Input, Embedding, 1D Convolution, Global 1D Max Pooling, Dense, Output Layer in Order
5. Define LSTM model
    - Input, Embedding, Bi-directional LSTM, Dense, Output Layer in Order
6. Train the models using TensorFlow, using GPU hardware.
7. Test the models with various hyperparameters and NLP features.

EXPERIMENTS, RESULTS & DISCUSSION

Experimental Setup

The Project ran in an environment with the following configuration:

- Software:
    1. Windows 11, WSL (Ubuntu 22.04)
    2. Python 3.10.12 (Virtual Environment)
    3. requirements.txt installed (automated install dependencies)
- Hardware
    1. Intel i9 14650HX
    2. GeForce RTX 4060 (Laptop)

Use of GPU for parallelization by TensorFlow package is possible only in a Linux or WSL environment. Thus, WSL was used for efficiency. TensorFlow was chosen due the experience the members of our team had with it.

Evaluation Metrics

The following metrics were used for evaluation amongst models:

1. Accuracy (% of classes correctly predicted)
2. Binary confusion matrix: We reduce the 6 classes into 2 (barely-true to true as positive and rest as negative)

These metrics are chosen due to the categorical nature of the output labels. This also lets us easily compare our results with other studies reporting their results in the same metrics.

Results

Hyper-parameter values providing the best results across most models: initial learning rate = 0.025 (range: 0.001 to 0.1, logarithmic increments), clip value = 0.3 (range:0.1 to 0.9, linear increments).

Following are the observed accuracies of CNN and LSTM models for different combinations of features along with the main content – the count vectorized statements:

1. Raw (only vectorized statements): 21.15, 20.36
2. Using POS: 24.39, 21.62
3. Using Dep: 23.75, 20.36
4. Using Meta: 23.36, 21.86
5. Using POS and Meta: 26.28, 24.23
6. Using Dep and Meta: 25.33, 24.23
7. Using POS and Dep: 23.20, 23.04
8. Using all three: 26.50, 24.53

The best results come from using CNN model with all three features, we were able to achieve an accuracy of 26.5% (exact comparison with 6 classes).

We were able to consistently achieve better results with deep CNN model compared to other models as described in related work section.

Discussion

Analysis of Results

Larger and lower values of the parameters leading to lower accuracies, is believed to be due to the model overshooting and undershooting the optima.

Using more data leads to better results in deep models, thus using all three features in CNN model gives us the best results. This also correlates with other studies.

Error Analysis

No errors outside of expectation occurred. There might be dependency version issues while running the project. Thus, it is recommended to install the versions of dependencies listed in “requirements.txt” in a virtual environment. This has been automated in the python notebook and it is only required to run the notebook using a virtual environment python kernel. To employ GPU resources for training, the project must be run in a Linux environment. WSL can be used if Windows is present.

CONCLUSION

Summary of Findings

The best performing model is found to be CNN, trained on all available features (added features from NLP methods like POS and dep). We were able to achieve an accuracy of 26.5%. This is close to the best accuracy achieved on this dataset by using a Hybrid CNN (combination of CNN and Bi-LSTM), trained on only the text and meta data, as reported in the original paper (Wang 2017). The deep learning model was found to be better than corresponding non-deep models described in the related work section. The project was able to achieve its intended objectives.

Contributions

This project gives a direct comparison between non-deep models with LSTM and CNN architectures. This project also solves the problem of missing implementation of the original paper (Wang 2017). The results can improve model selection for various tasks in future work.

Implications & Applications

The results of this project imply that deeper models trained on more data are better for complex classification tasks. This project also shows the viability of CNN architecture in sentence classification. The results of this project can be used in future projects and also to build apps to combat misinformation on various social and news platforms. This is especially valuable with the rise of AI.

Limitations

The project was built with the following limitations:

1. Limited Compute capacity and time: We could not try out complex and deep models because of this (e.g. Hybrid CNNs). Also, we cannot employ large datasets because of this limitation.
2. Lack of recent data: The data used in the project is a little outdated, this also carries on to the models. Using latest data with higher will make the model relevant and more powerful

Future Work

The model can be improved in the following ways:

1. Combining multiple models (e.g. Hybrid CNNs)
2. Training on more modern data
3. Employing various NLP techniques not documented in this project (e.g. using transformers).

REFERENCES

1. William Yang Wang. 2017. " liar, liar pants on fire": A new benchmark dataset for fake news detection. arXiv:1705.00648.
2. Mansoor9743. 2019. Natural Language Processing NLP - Fake news Detection. github.com/mansoor9743/Fake-News-Detection
3. Zhang, H., Fan, Z., Zeng, J. & Liu, Q. (2012). An Improving Deception Detection Method in Computer-Mediated Communication. Journal of Networks, 7 (11).
4. Yoon Kim. 2014. Convolutional neural networks for sentence classification. arXiv:1408.5882.
5. Jeffrey Pennington, Richard Socher, Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation. https://nlp.stanford.edu/projects/glove/

APPENDICES

Code

The code can be found in the following GitHub repository: https://github.com/Neo-Panther/Fake-News-Detection/tree/submit

The main branch is being used to develop a frontend feature based on the code, our team is submitting the “submit” branch.

Repository Structure

- Data
  - test.tsv
  - train.tsv
  - valid.tsv
  - README.txt
- cnn_weights.keras
- lstm_weights.keras
- Main.ipynb
- README.md
- requirements.txt

Repository Description

Data folder contains the raw data used in the project. test, train, valid, each containing training, testing, validation data respectively. The included README.txt contains the original dataset description included with the data.

.keras files contain the weights of the best model trained by the project. Main.ipynb contains the main code for the project, separated using python notebook cells and markdown. README.md contains further documentation for the project. requirements.txt contains the project dependencies with their required versions, to be installed automatically while running the code in a virtual environment.

Additional Figures/Tables

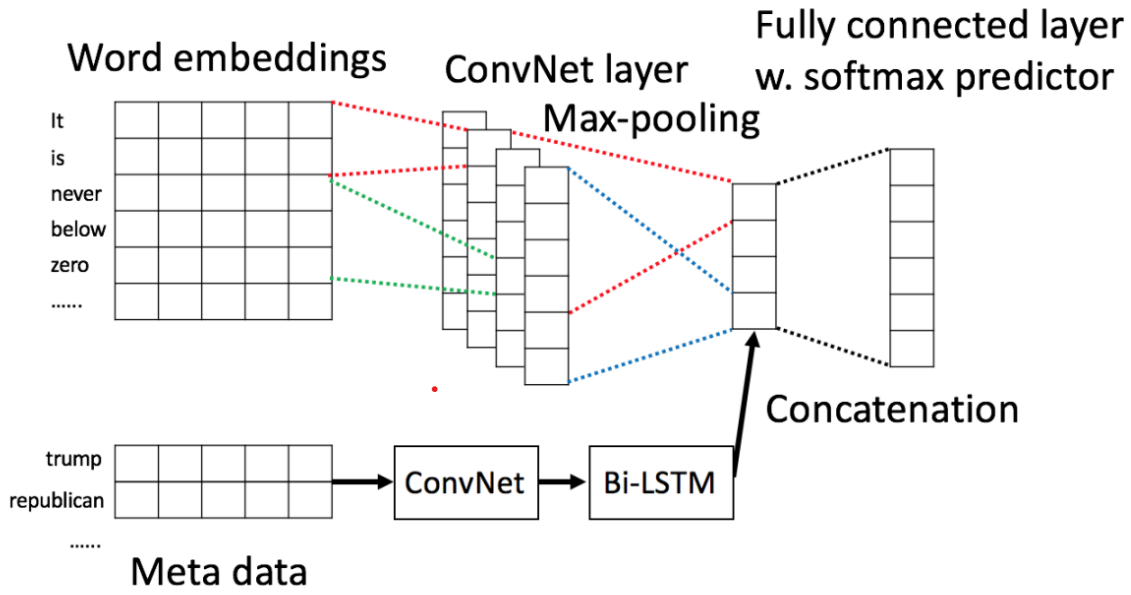

Fig. 1 Hybrid CNN

Ref: Wang 2017

Models	Valid.	Test
Majority	0.204	0.208
SVMs	0.258	0.255
Logistic Regress0ion	0.257	0.247
Bi-LSTMs	0.223	0.233
CNNs	0.260	0.270
Hybrid CNNs		
Text + Subject	0.263	0.235
Text + Speaker	**0.277**	0.248
Text + Job	0.270	0.258
Text + State	0.246	0.256
Text + Party	0.259	0.248
Text + Context	0.251	0.243
Text + History	0.246	0.241
Text + All	0.247	**0.274**

Table. 1: Results from Original Paper

Ref: Wang 2017