CISC 451 Project: Fake News Classification

November 27th, 2020

Group Members:
Kim, Harry 2004 7381
D'Gama, Ryan 2005 2250
Luo, Zili 2000 1744

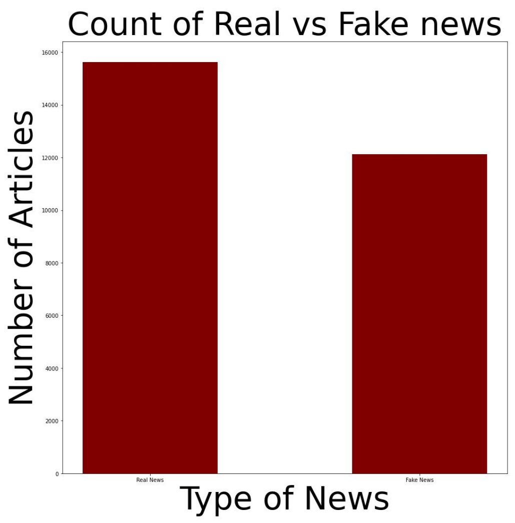
Problem

The media dictates how we understand and interpret events happening around the world. Unfortunately in today's world of ever-growing information, we have seen the introduction of the term "fake news". Fake news is the spread of misinformation in the form of "true news" in hopes to sway a person's understanding of current events. This is particularly dangerous as fake news has the ability to change people's political views, where they go to shop, who they perceive to be good/bad in the world, and much more. Our project aims to look at a data set we obtained through Kaggle and through applying various Natural Language Processing techniques, correctly classify fake news at a high rate of accuracy.

Dataset

Our data set was obtained from Kaggle[1]. The data set from Kaggle was obtained by scraping websites titles and contents along with an extension called "BS Detector", which got the label of fake news and real news for each article.

The data set originally had 33,788 instances. However, after removing those with missing values, since imputation was not possible on corpuses, we were left with a data set of 27,878 values. This data set had four columns, with one being the target label. The predictors consisted of Title, Content and Publications. Out of the total 27,878 instances, 15,712 belonged to the target class "real news" and the rest, 12,166 belonged to the target class "fake news". This can be seen on the visualization provided on the next page.



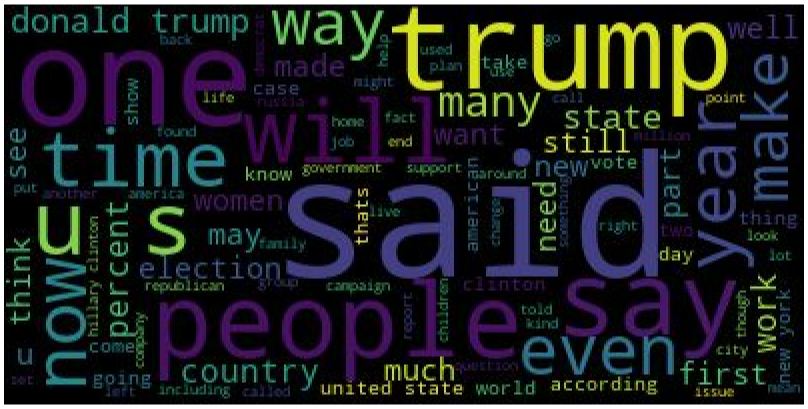
Title consisted of the article's title and they tended to be short. Fake news and real news both having around 11 words on average. However, content, which consists of the article's body, had differing lengths between real and fake news. With real news having an average of 948 words per article while fake news had an average of 660 words.

The biggest difference between the two datasets came from publications. Real news had just ten publishers, all of whom were very reputable. This consisted of high-quality. non-partisan sources such as New York Times, Washington Post, and NPR, as well as some more partisan sources such as Fox News and CNN. Each source had at least 900 articles, with some, such as NPR having more than 2000.

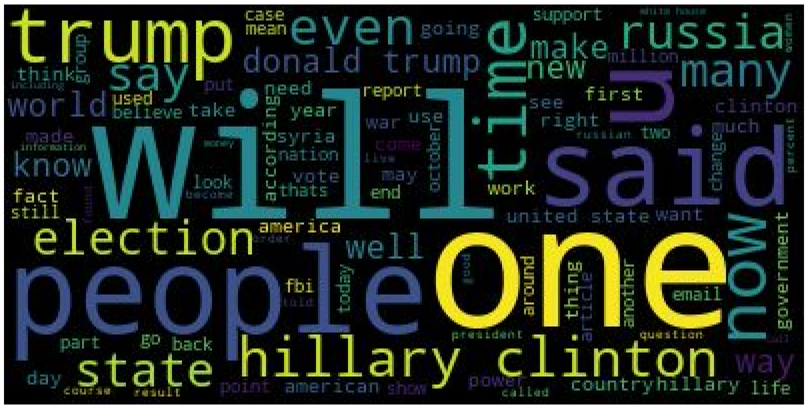
In contrast, the fake data set consisted of 240 sources, with each having very few articles among them. Some sources such as "world new politics" had just a single article. These sources ranged from high-quality sources such as ABC news to some completely unreliable. One example was one source called "magafeed", who's website has been taken down and has just 5 followers on their Instagram page.

To get a better understanding of the words that make up our dataset, we created two word clouds as seen below. The first image represents the class of “True News” while the second image represents “Fake News”. By analyzing these two word clouds it is clear that a political theme is very present within the data. The most frequent word in the true news word cloud is the word “said”. This makes sense as this word typically indicates the presence of a quote. The code for the word clouds below can be found in word_cloud.py.

Real News WordCloud



Fake News WordCloud



Challenges

The two most challenging things are finding a dataset and data prepossessing.
At the beginning stages of our project, we had another data set. However, all the baseline models achieved incredibly high accuracy on both training and testing sets(1.0 for training, 0.98+ for testing) without any fine-tune process. This was due to the fact that all the "real news" and "fake news" articles came from one source each. In order to remove this bias, we found another data set.

The new data set, the one we ended up using for our models provided some inherent bias as well. As previously mentioned, the dataset we obtained from Kaggle used an extension called "BS Detector" by Daniel Sieradski. This extension worked, according to the founder, by "scan[ing] a given web page for the presence of links and then checks the links against a database that has been compiled of fake news sites..."[2].

This obviously brings forth some bias in our data. We do not know exactly what these "fake news" sites were, and how they were picked. One of the problems with this is that a genuine 'real news' could have been labeled as fake news just because it had a link to a website that was in the fake news database. This overall introduces a large human bias into the equation. Therefore, we got rid of the column Publications in preprocessing. However, from looking through the data, most of these "fake news" websites did seem like they were labeled correctly so we kept the dataset.

Lastly, the training time on our models was a challenge. With average length of content being near a thousand, some of the techniques we were trying took a long time to achieve. One example of this was RandomSearch, where even with few parameters set, it took several hours to complete for all three base model algorithms. In order to compensate for this, we used tf-idf and Bag-of-Words models for word embedding. Word-2-vec and Continuous Bag of Words are considered state-of-the-art embedding models yet we were not able to use them. We hope for future usage, we will be able to.

Methodology

Libraries

For this analysis we used Spyder and Jupyter Notebooks as part of the anaconda distribution. We used multiple libraries in our model and pre processing. Most are in the Anaconda Platform. There were some that were not on the platform. For these, you can simply enter on your command line either pip install a_library or conda install a_library where a_library is any of these modules: wordcloud, autocorrect, sklearn, Keras, Tensorflow, Pandas, NumPy, matplotlib, nltk.

Codes

Our code has been distributed into three separate files- preprocess, Final_reportLSTM_CNN, and stacked_model. They were all written and saved as a Jupyter Notebook file, with an ipynb file extension. The file preprocess.ipynb will create a CSV file that will be used in Final_reportLSTM_CNN while stacked_model runs everything internally. Visualization and data exploration are also done in stacked_model.

Preprocessing

Due to the differing nature of our models, we had two separate preprocessing that took place. They both essentially did the same thing.

We have applied following prepossessing method to the news dataset:

1. We removed any rows with missing data as there is no way to impute text articles.

2. We removed any leading spaces and converted all text to be of the form a-z and 0-9.
3. We then utilized nltk and removed all the stopwords from the text corpus.
4. For each tokenized word, we did a spell check to get rid of typos, un-words to cleanse our vocabulary.
5. Lastly we stemmed all words to remove any variations of words, bringing the word down to its root.
6. We removed unnecessary columns (publications)
7. Once the above steps were done, the resulting data frame was saved as ‘clean_data.csv’ for analysis of the stacked model and “news_dataset_preprocessed_all.csv” for the LSTM-CNN model.

For clean_data.csv, the pre-processing happens inside the Jupyter Notebook “stacked_model.ipynb” as one of the cells while “news_dataset_preprocessed_all.csv” is processed in a separate file called “preprocessing.ipynb”

Modelling

Baseline models

To start our modelling process, we began by creating a variety of baseline models using the scikit learn library and TensorFlow. We implemented Logistic Regression, Support Vector Machine(SVM), Random Forest, using the SK-Learn library, We also implemented an ANN using the TensorFlow library. The models had varying degrees of success, with SVM having the highest with 92% and ANN having the lowest with 84%.

Model	Accuracy
Logistic Regression	91%
SVM	92%
Random Forest	85%
ANN	84%

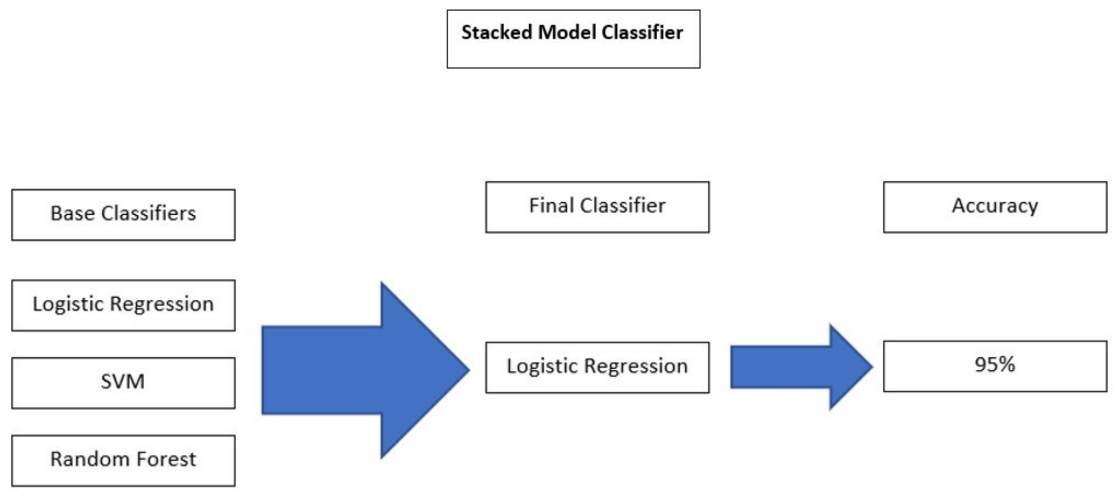
Model Improvements

After we run all the baseline models on the dataset. We applied some fine-tune methods on our baselines as well as trying to build more robust models based on our baselines:

Model Improvement 1: Stacked model classifier

Random Search was applied for the base models in order to find the best hyperparameters. In order to do this, we used SK-Learn's RandomSearchCV with CV (cross validation) set to 10. This was very expensive to run but after it was run once, we did not have to run it again.

Once we had our optimal parameters provided to us by Random Search, we then applied ensemble learning to our baseline models. The initial classifiers consisted of logistic regression, random forest, and svm. The final classifier used was logistic regression. By applying this technique as well as the optimal parameters we were able to achieve an accuracy of 95%. The way this model works is by training the base classifiers on the original dataset and then the final classifier is trained on the output of the base classifiers. To implement this ensemble method, we utilized sklearn's StackingClassifier method. The diagram below illustrates the model.

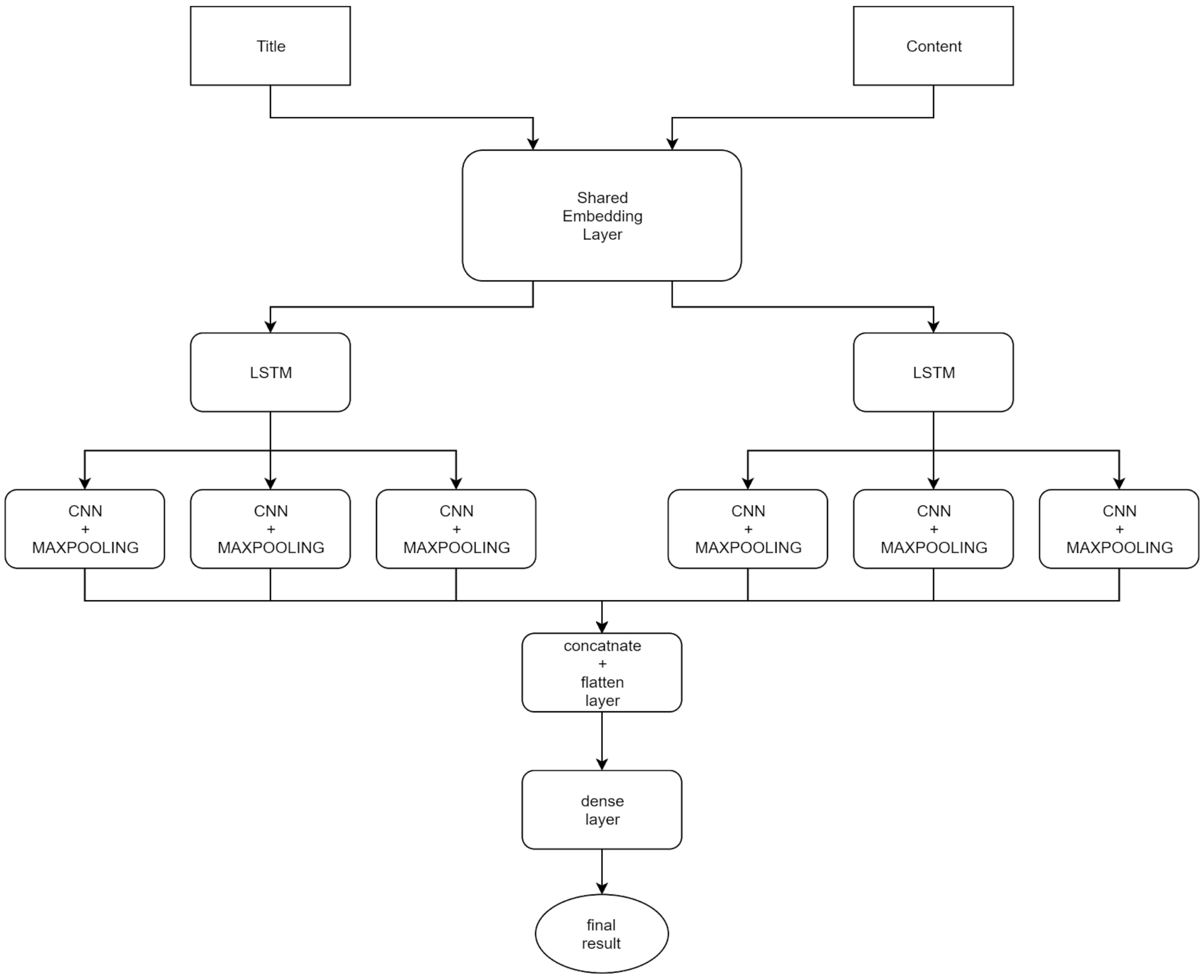


Model Improvement 2: LSTM-CNN Neural Network

After we finished our baseline ANN model, we tried to push the ANN model to achieve higher performance. Starting from the Neuron Network structure, we decided to use more functional layers to replace the dense layer. This resulted in a LSTM-CNN-Dense Structure, where we used the bag of words (BOW) embedding method instead of the tf-idf.

The model processed the title and content separately but with a shared embedding layer to reduce the bias of the same words in different places(title and content). It then feeds the embedding vector into a LSTM layer. Instead of using the final sequence of LSTM output, we made the LSTM return the whole sequence which makes the output a 2d vector instead of 1d vector. With these 2d vectors, we applied 2D convolution to them. The way we applied convolution was slightly different, we applied 3 convolutional layers with different kernel size simultaneously to the LSTM output and stacked the result from the title part and content part together, then flatten them into a dense layer to produce the final result.

The general model structure is described by the flowchart below:



Looking into the detail of the model structure, We think the title and content have different importance on determining whether a news is fake or real. Also, they are normally different in length. In our preprocessing, all the titles are padded to a length of 64 tokens while all the contents are padded to a length of 512 tokens.So we decide to process those two parts separately in our LSTM-CNN model.

Despite the important difference, as we are using a BOW approach, a separated embedding layer would lead to highly biased vocabulary(mainly for title). Our solution to this was using a shared embedding layer, which makes a bag containing all the existing words.

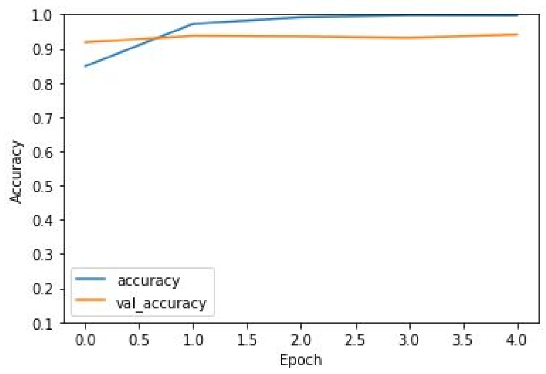
As we mentioned before, instead of using the last sequence of the LSTM layer, we took advantage of the LSTM's timedistributed feature, making LSTM return the whole sequence. Which would result in a 2D vector output instead of 1D, containing more information.

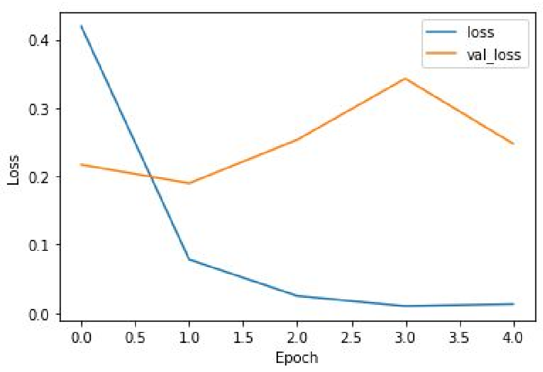
Convolutional layer is good at extracting features from the input, in order to squeeze more features from the LSTM, we feed the output sequence into three convolutional layers with kernel size of (3,3),(4,4),(5,5) simultaneously. Each of those convolutional layers is connected to a max-pooling layer paired to its kernel size.

After pooling layers, we combined the title vectors and the content vectors into a single news vector by stacking all the outputs from Convolutional layers together. Before feeding the news vector into the dense layer, we applied a flatten operation to change it into a 1D vector.

Our dense layers are relatively simple, it contains two fully connected layers with 128 nodes and 1 node each. The first layer uses ReLu activation function while the second layer uses sigmoid as activation function to predict the final result.

The LSTM-CNN model achieved 99.64% training accuracy and 94.07% testing accuracy after 5 epochs of training processes.





From the training and testing performance plot seen above, we believe the LSTM-CNN model could achieve higher testing accuracy after a few more epochs. But the training time was immense and we were satisfied with the result we achieved.

Evaluation

Model	Training Accuracy	Testing Accuracy
Logistic Regression (Baseline)	94%	91%
SVM (Baseline)	99%	92%
Random Forest (Baseline)	99%	85%
ANN (Baseline)	88.08%	84%
Stacked Model Classifier (with random search)	98%	95%
LSTM - CNN	99.64%	94.07%

As seen in the table above, Our LSTM-CNN and Stacked Model Classifier performed the best, exceeding our best baseline model by 2-3%. The LSTM-CNN model which was much more complex was as accurate as the stacked model when executed on our dataset. This allows us to come to a conclusion that the stacked model may be a better fit if the model were to be implemented into production. The reasoning behind this is because the LSTM-CNN model although performs well, requires much more computation power and overall storage to run the model compared to the stacked model. Overall, we were able to achieve a relatively high accuracy beating our baseline models.

Conclusions

The problem of correctly classifying articles continues to be more and more relevant recently. We tackle this problem using a Kaggle dataset containing over 20 thousand article's body and title. We first attempt models like Logistic Regression and ANN in order to get a baseline. From here, we implement two different techniques- Stacked Model Classifier and LSTM- CNN. We find a big jump in accuracy from both the stacked model and LSTM-CNN. Thus we show that given proper NLP processing and competent models, we are able to classify fake news and real news articles to a high degree of success.

Future Work

There are few things that can be improved from this. First, despite the fact that both of our own models performed better than the baseline ones, the training time was significant. For classification models to be applied in real-world usage, it will likely have to be able to classify in real time. That means that future models should be more efficient in classifying.

The CNN-LSTM model still has a lot of space to be improved. We used BOW as our embedding method and achieved a good result. But normally tf-idf performs better than BOW. Also we could modify the model structure to make it possible to use Word2Vec embedding or BERT embedding without parameter explosion as our model currently has 542m parameters which is incredibly large.

Another thing we can improve on is accuracy. Once the accuracy is at a higher rate (around 98% or greater) the model can be deployed in a real time classification system. To add more detail to this, we can examine a use case of this project. When a user scrolls through their facebook feed, they usually come across different news articles embedded in their feed as links to websites. If facebook were to deploy a model such as this, they would be able to in real time classify the articles as either being either "True News" or "Fake News" and letting the user know before clicking on the link that the article may not be verified in turn reducing the spread of fake news. A model such as this could be applied to various mediums such as Twitter, Reddit, and other social media or news sharing programs.

Sources

[1]
https://www.kaggle.com/anthonyc1/fake-news-classifier-final-project/output?fbclid=IwAR3ro9zLIXff93-nP6PQUkKCsrQgJHOwWLGORSYmRljkAgcpigwDND2DQTY

[2]
https://www.cbc.ca/radio/asithappens/as-it-happens-friday-edition-1.3878672/tech-designer-schools-facebook-creates-fake-news-detector-in-an-hour-1.3878682