What's the Tea: Topic Classification of News Articles
Group 3 - CS543

Advith Chegu
Rutgers University
New Brunswick, NJ, USA
ac1771@scarletmail.rutgers.edu

Vipul Gharde
Rutgers University
New Brunswick, NJ, USA
vig4@scarletmail.rutgers.edu

Diksha Wuthoo
Rutgers University
New Brunswick, NJ, USA
dw659@scarletmail.rutgers.edu

***Abstract*—We have implemented Topic Classification for news articles to classify different articles into multiple topics in real-time. We have used a deep learning network model to classify news articles into 42 categories. We trained our classification model to classify different news articles, and then applied this model to real-time Tweets from various authorized Twitter news handles to predict the topics at any given time. We also allow users to view the top 'N' most popular Twitter topics at any given time and see their related Tweets as well.**

***Index Terms*—News, topics, classification, Twitter, Natural Language Processing (NLP), Deep Learning, CNN, LSTM**

I. Introduction

With the rise in the volume of real-time information, consumers are inundated with countless stories and news articles that make it difficult to sift through and search for their topics of interest. We wanted to provide a solution that will help busy consumers stay on top of current events without much effort or many clicks to do so. With this motivation in mind, we wanted to make an application that would allow curious users to know what topics are the most trending in the world and then find articles related to those topics. Our application provides users with real-time trending topics such as Politics, Business, Sports, etc., helping them to view relevant news from the topic of their interest. We fetch real-time Tweets from authorized and reputable news handles on Twitter and use those Tweets to show the user what topics are the most trending.

II. Dataset Description

We are using multiple datasets for this project:

HuffPost Dataset [1]: This dataset is a single JSON file with the fields *link*, *headline*, *category*, *short description*, and *authors*, and *date*, but we do not need to use all those fields for the classification model. It is a labeled dataset (around 100MB in size) and we have used the combination of *short description* and *headline* fields from the dataset for our implementation. The resulting dataset totals around 210k news articles.

RealNews Dataset [2]: This dataset (taken from the paper [3]) is a single JSON file with the fields *article title*, *article text*, *summary*, *authors*, *published date*, *URL*, and *domain*, but we also do not need to use all those fields for the classification model. Since it is a huge dataset (over 100GB in size), we have used the *summary* field from the dataset for our implementation. The resulting dataset totals around 35 million news articles. We found that some articles do not have a summary, so for those records, we have used the *title* field from the dataset instead.

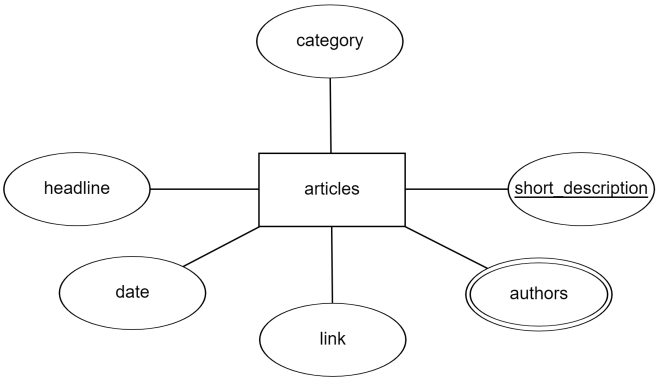

Fig. 1. ER diagram of the HuffPost dataset

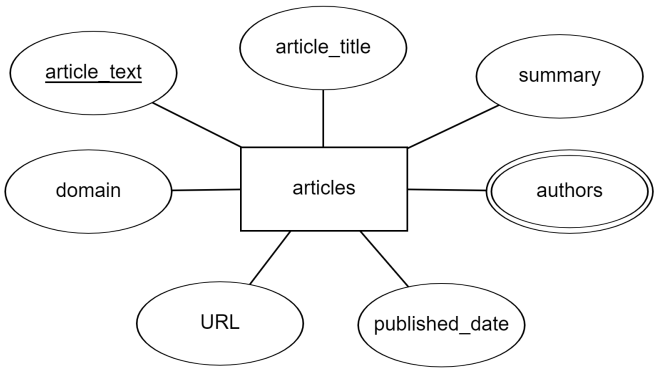

Fig. 2. ER diagram of the RealNews dataset

News Aggregator Dataset [4]: This dataset from Kaggle is a CSV file with the fields *ID*, *TITLE*, *URL*, *PUBLISHER*,

CATEGORY, *STORY*, *HOSTNAME* and *TIMESTAMP*, totaling over 100MB and over 400k news articles. We have used the *TITLE* field.

A Million News Headlines [5]: This dataset from Kaggle is a CSV file with the fields *publish date* and *headline text*, totaling over 60MB and over 1.2 million news articles. We have used the *headline text* field.

All the News 2.0 [6]: This dataset is a CSV file with the fields *date*, *year*, *month*, *day*, *author*, *title*, *article*, *url*, *section*, and *publication*, totaling over 60MB and over 2.6 million news articles. We have used the *title* field.

India News Headlines Dataset [7]: This dataset from Kaggle is a CSV file with the fields *publish date*, *headline category* and *headline text*, totaling to over 250MB and over 3.6 million news articles. We have used the *headline text* field.

The combined dataset totals over 40 million news records. Since all the datasets except the HuffPost dataset are unlabeled, we had the challenge to convert them to labeled datasets for our supervised learning model. To tackle this issue, we actually used a pre-trained model to generate the output labels. This model converts all the input sentences and labels to vectors first and then finds the cosine similarity between them, and the class with the highest value will be the output label.

III. Questions About the Data

With this dataset, we answer the question of what is being discussed in real-time. For example, if a user wants to see the most trending topics at a given moment, they can use our model to see what's popular, which will show the top 'N' recently tagged topics. Here, 'popular' is defined as the output topic from the model having the most number of counts from recent Tweets. The user can also choose any one of the topics and view Tweets from that particular topic. This will also save users a lot of time to know about the current events of the world.

IV. Implementation

Our implementation of the application consists of multiple steps like pre-processing of the data, generation of word embeddings, training on the implemented deep learning model and then testing and fine-tuning of the model based on the evaluations.

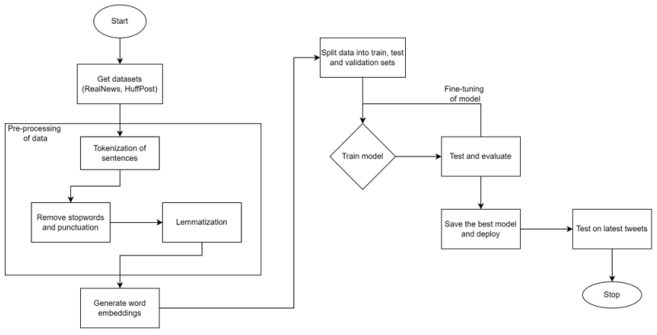

Fig. 3. Conceptual Flowchart of Our Application

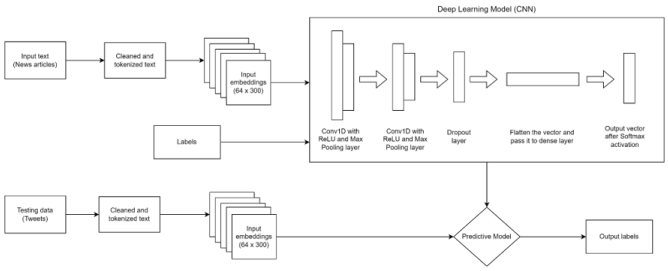

Fig. 4. Architecture Diagram of Our Implemented Deep Learning Model

A. Pre-processing Data and Generating Word Embeddings

We pass these datasets for pre-processing, which includes the removal of stopwords, punctuation, lemmatization, etc. We start with tokenizing the summary for each record using the *NLTK* [8] library, giving us an array of words. After tokenization, we convert all the words to lowercase and remove the stopwords like 'a', 'an', and 'the', which do not contribute to the semantics of the sentence. For the removal of those words, we have used the already provided set of *STOPWORDS* in the *gensim* library. We also remove punctuation marks and any words that are of length 2 or less. After that, we pass the whole dataset for lemmatization, for which we have used the *WordNetLemmatizer* of the *NLTK* library. Lemmatization is a text normalization technique commonly used in Natural Language Processing that helps us to group different forms of the same word and transform them to root forms while preserving the semantics. This is the final step of pre-processing the dataset.

Once we have the cleaned and processed text, we store this data into a PySpark dataframe and apply the spacy embeddings on them. The particular spacy embedding model we use is called *en_core_web_lg* which generates 300 dimension sentence embeddings. This means that every sentence gets converted into a 300-dimension vector of floats no matter the original size of the sentence. We used the spaCy embeddings to avoid having to pad the output embeddings if we had to use a word embedding like *glove*. The *en_core_web_lg* model has been trained on a variety of different text data including wiki articles, news articles, subtitles, blogs, and religious texts. More info can be found here [9].

B. Model Building and Training

For creating the deep learning model, we have used a Convolutional Neural Network (CNN) [10] [11] as convolution layers are better at understanding the features from the feature embeddings that we already generated in the last step. We first split our input embeddings into train, test, and validation sets in the ratio of 8:1:1. The training set is passed to the network in a batch size of 64. We are using 2 convolutional layers where each Conv1D layer is followed by a ReLU activation layer and a Max Pooling Layer to reduce the dimensionality of the embeddings and also to get meaningful features. As our input embedding is a vector of size 300 and we have a batch size

```
[
    {
        "handle": "cnnbrk",
        "id": 428333
    },
    {
        "handle": "CNN",
        "id": 759251
    },
    {
        "handle": "nytimes",
        "id": 807095
    },
    {
        "handle": "BBCBreaking",
        "id": 5402612
    },
    {
        "handle": "BBCWorld",
        "id": 742143
    },
```


Fig. 5. JSON of Twitter news accounts and their Twitter IDs

of 64, we pass this as input to the first Conv1D layer which has 64 neurons, kernel size 3, and stride 1. The output of this layer after Max pooling is passed to the next convolution layer which has X neurons and is again followed by a Max Pooling Layer. After this, we add a dropout layer of 0.25 (optimal rate after experimenting) which is then flattened to a 1-D vector and passed to a dense layer, followed by a Softmax layer to get the probabilities for each of the 42 output classes.

C. Model Testing and Fine Tuning

As our CNN model is now trained on news articles, we have to test this model to predict the topics and evaluate the performance of our implemented model. We run the model on 10% of the input data for around 20 epochs. Figure 6 shows the different hyperparameters that we tried on the test set and got the best accuracy on Model 2. We saved this model to get future predictions on it.

	Learning Rate	Number of Layers	Epochs	Accuracy
Model 1	0.1	2	20	55.5
Model 2	0.05	2	20	57.9
Model 3	0.01	3	50	54.7

Fig. 6. Accuracy of the Model with Different Hyperparameteres

Fine-Tuning: We are using CategoricalCrossEntropy as the loss function which is most commonly used in the case of multi-class classification. We are also using Stochastic Gradient Descent(SGD) optimizer which gave us better results than Adam and RMSProp optimizers. We tried training models with different learning rates like 0.1, 0.05, 0.01, and 0.005 but we found that the model gave the best accuracy when the learning rate was set to 0.05. We trained and tested the model on 50 epochs but the accuracy plateaued after 20 so we reduced the epoch count to 20.

We also ran the prediction on real-time latest Tweets to classify them into specific topics. We have selected 34 authorized and reliable Twitter news accounts (shown in 5) to fetch the latest 10 Tweets from each of those accounts. For retrieving the Tweets, we have used the Tweepy package which provides APIs to fetch the data. After retrieving the Tweets, we then do the pre-processing steps as we did before for the entire dataset which includes tokenization, removal of stopwords, punctuation, and lemmatization. The raw data from the Tweets also include some links at the end of each Tweet which we remove using the regex filter. Now that we have the cleaned words, we then use the same *spaCy* model (created in the model building step) to get the word embeddings and pass it to the saved classification model for predictions. The results of the predictions are shown in the next section.

V. Experimental Results and Analysis

We plotted the test accuracy of the model with respect to the number of epochs it took to train the model. We could see that the accuracy had greatly improved initially but then plateaued after around 20 epochs in figure 7. Due to this reason, we reduced the number of epochs for training our model.

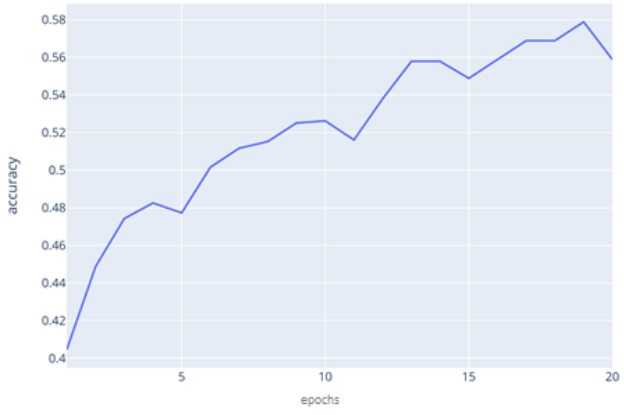

Fig. 7. Plot of accuracy vs epochs For Training the Model

For Twitter data evaluation, we took the latest 340 Tweets and passed them to the classification model which predicted these categories to be the most trending topics on Twitter. We can see that Politics, World News, and Business are the most commonly discussed topics on Twitter at the time of prediction in figure 8.

We also have a plot of a heatmap that shows what topics are accurately classified by our model. It shows the ground

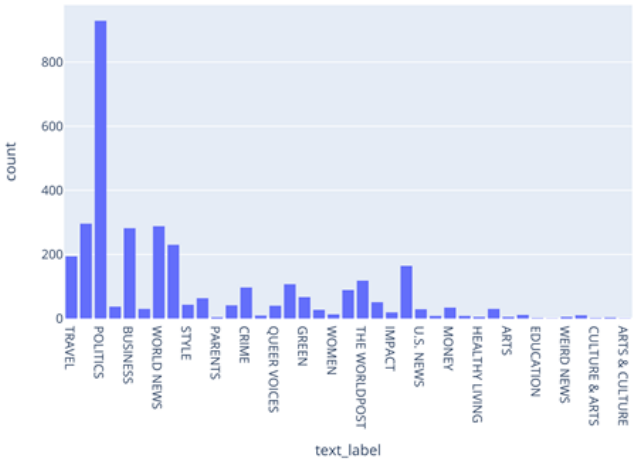

Fig. 8. Bar Graph of the Most Trending Topics Based on Tweets

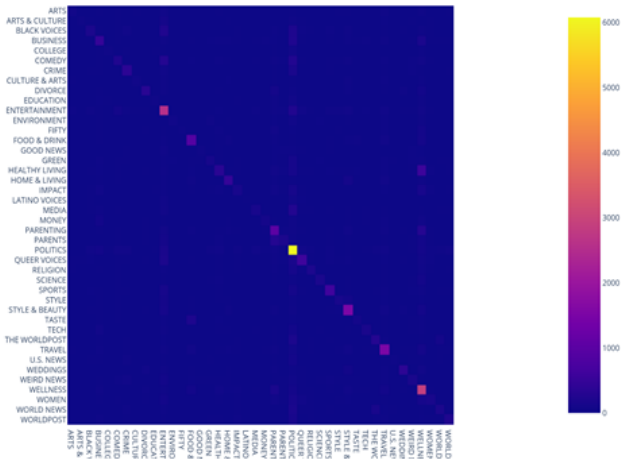

Fig. 9. Heat map of Actual vs Predicted Topics

truth classes at on the y-axis and the predicted classes on the x-axis. As shown in figure 9, many of the classifications fall in a diagonal line in the image meaning that the predicted class and the ground truth label are the same. However, there are some faint vertical lines around the Entertainment and Politics categories which means that our model is misclassifying other category data into those popular categories. This is likely due to the fact that we had an unbalanced source data set that emphasized news relating to Politics and Entertainment among other categories to the detriment of categories like Weird News and Parenting. Another reason why this may be happening is that the categorization of news is highly subjective and certain news articles may fit into multiple categories. For example, the news about the Peruvian president who tried to overthrow his congress could technically be classified under both the Politics and World News labels. One solution to this would be to use a simpler labeling scheme and cut down on the number of labels from 42 to something more manageable like 10.

We also plotted a word cloud for a few topics (shown in figures 10 and 11 to see what words occur the most in these categories. The words that appear bigger in the word cloud are the words that more frequently occur in their corresponding topic.

Fig. 10. Word Cloud of Politics topic

Fig. 11. Word Cloud of World News topic

VI. Tools and Technologies

We are using *PySpark* which supports numerous libraries and packages related to Natural Language Processing (NLP) for tokenization, lemmatization, and stop words removal to format our training and testing dataset. We have used *NLTK* for all the NLP-related techniques. We also used *spaCy* which is a doc-to-vector converting library that helps us to generate the word embeddings for all the input sentences. We are implementing our CNN model in *PyTorch* [12] which provides a lot of in-built libraries to train and test the model. We are using *Plotly* and *MatplotLib* to generate the graphs like test accuracy, heatmap, and word cloud. For the User Interface, we used the Flask API to deploy our model and used Tweepy to fetch the latest Tweets.

VII. Timeline

We have prepared a Gantt Chart for this project shown in figure 12. In the first week, our main focus was to study different models which would have a good performance on our textual dataset. We also spent a major portion of the time in the second and third weeks labeling the RealNews dataset as we are doing supervised learning. We simultaneously started working on implementing the model in the 3rd week. Testing and fine-tuning of the model was done in the 4th week which took a lot of time as we had to try multiple combinations of the hyperparameters. In the final week, we were working on minor tweaks in the model and the documentation of the project.

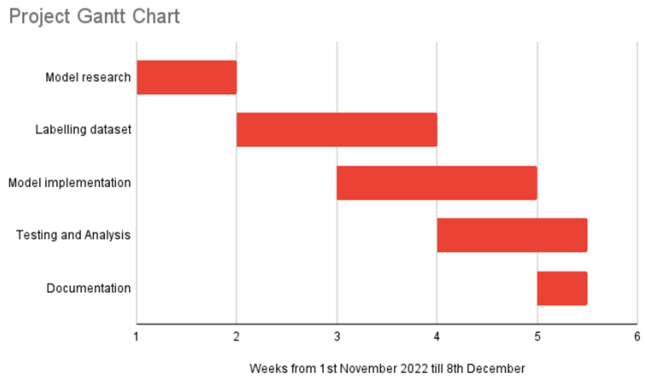

Fig. 12. What's The Tea Gantt Chart

VIII. Conclusion and Future Work

In this project, we implemented an application with a user interface to classify news Tweets to some pre-defined topics (42 in our dataset). We first create the word embeddings using the pre-trained *spaCy* model which are then passed to the deep learning model created using CNN. We built the UI using Flask, used Tweepy to fetch the latest 340 Tweets, predict the topics using the CNN model and show the top 'N' trending topics to the user. The user is also allowed to search for Tweets that belong to a particular topic.

Some setbacks that we faced during project 1 were improved:

1. The reduced dimensionality of our document vectors was changed by using a different model for generating the embeddings. Due to this, certain categories which were overlapping against each other were significantly improved.

2. More flexible results as per the user needs, we provide APIs to fetch the top 'N' trending topics that the user wants instead of just showing the top 5 and also provide the user functionality to fetch Tweets related to any particular topic.

In the future, we can extend the application to classify different text articles apart from news Tweets. We would also try to improve the accuracy by using different word embedding models like GloVe and have more complex models like ResNet and BiDirectionalLSTM. We tried to implement a new form of semi-supervised learning called 'Active Learning' in this project but couldn't complete it given the time and memory constraints, however, we can try to implement it and see the accuracy as it works well with learning from unlabeled data.

References

[1] News Category Dataset — Kaggle. [Online]. Available: https://www.kaggle.com/datasets/rmisra/news-category-dataset

[2] RealNews Dataset — Papers With Code. [Online]. Available: https://paperswithcode.com/dataset/realnews

[3] Defending Against Neural Fake News. [Online]. Available: https://arxiv.org/pdf/1905.12616v3.pdf

[4] News Aggregator Dataset — Kaggle. [Online]. Available: https://www.kaggle.com/datasets/uciml/news-aggregator-dataset

[5] A Million News Headlines — Kaggle. [Online]. Available: https://www.kaggle.com/datasets/therohk/million-headlines

[6] All the News 2.0 — 2.7 million news articles and essays from 27 American publications - Components. [Online]. Available: https://components.one/datasets/all-the-news-2-news-articles-dataset/

[7] India News Headlines Dataset — Kaggle. [Online]. Available: https://www.kaggle.com/datasets/therohk/india-headlines-news-dataset

[8] NLTK :: Natural Language Toolkit. [Online]. Available: https://www.nltk.org/

[9] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[10] Text classification using CNN. In this article, we are going to do... — by Vijay Choubey — Voice Tech Podcast — Medium. [Online]. Available: https://medium.com/voice-tech-podcast/text-classification-using-cnn-9ade8155dfb9

[11] Deep Learning Techniques for Text Classification — by Diardano Raihan — Towards Data Science. [Online]. Available: https://towardsdatascience.com/deep-learning-techniques-for-text-classification-78d9dc40bf7c

[12] Defining a Neural Network in PyTorch - PyTorch Tutorials 1.12.1+cu102 documentation. [Online]. Available: https://pytorch.org/tutorials/recipes/recipes/defining_a_neural_network.html