DETECTING FAKE NEWS THROUGH NATURAL LANGUAGE PROCESSING

ML1010: Group Project

Group: JSMCJ

James Lai

Julia Mitroi

Shahla Molahajloo

Clive Pau

Manvir Sekhon

Getting Real About Fake News: Solving a Text Classification Problem Using Natural Language Processing

1. Problem Statement

Recent major political decisions like Brexit and the last U.S. presidential elections surprised journalists and professional predictors alike with their unexpected outcomes. In the aftermath, one major contributor has been identified as being “fake news”: stories and memes with no basis in truth that may have been created to sway the outcome of these political events. Over the past two years, fake news has become a key societal issue and a technical challenge for media organizations to deal with. This type of content can be difficult to identify because it often hides under the appearance of a legitimate news organization; and because the term "fake news" covers intentionally false, deceptive stories, as well as factual errors and satire, and sometimes stories that a person just does not like (but that may be factually true). Addressing the problem of fake news requires clear definitions and examples, and validated bias-free methods.

Fake news is harmful because it affects the functional logic and integrity of the society. In today’s world, where people are informed mainly through the media and form their political opinions through it, this process is threatened when misinformation spreads through the media. When it is no longer clear, what is false and what is correct, people lose their confidence in official sources. “Is it true, what they tell me on TV? What did I read in the newspaper? What I've heard on the net?” In addition to creating confusion and misunderstanding about important social and political issues, there may be biased and misleading news stories related to medical treatments and life-threatening diseases. Trusting these false stories could lead people to make decisions that may be harmful to their health. Research has shown that news consumers have difficulty distinguishing between real and fake news sources and information.

Fake news is clearly a serious problem, and many wondered what data scientists can do to detect it and stymie its viral spread: How can Artificial Intelligence (AI) and deep learning be leveraged to detect fake news? Efforts in this direction have shown that news content can be analyzed through big data and mathematical models that process human language and classify text. *Natural Language Processing (NLP)* is the field of AI devoted to processing and analyzing any form of natural human language. It aims to bridge the gap between how computers and humans process information, and by using it data scientists and machine learning engineers can analyze large quantities of human communication data. In the context of the fake news problem, NLP enables breaking down news articles into their components and choosing important features, to then construct and train models to identify unreliable information.

The current project addresses the issue of fake news by analyzing news text with NLP. The business objective is to construct a model (or several) that can flag news content as fake, and thus help reduce or stop the destructive consequences of the proliferation of fake news.

2. Project Plan (Approach)

The news text analysis goal noted above is to be achieved in this project by using a) two publicly available news datasets, b) a machine learning pipeline for text classification task

The broad steps of the machine learning project are:

- Identifying the data source(s) (described below, under Data Collection)
- Data preparation: pre-processing/cleaning (Section 4 of this report)
- Selecting the relevant information from the data and convert it into formats recognizable for machine learning algorithms (Section 5: Feature Engineering)
- Identifying the best performing text classification models (Section 6)
- Interpreting and reporting the results (Section 7)

3. Data Understanding

3.1. Data collection

The data used for this project was sourced from two public datasets from the Kaggle Repository, a fake and a real news dataset, which we merged to obtain a balanced mix of real and fake news items. The news articles in both datasets were published from October to December of 2016.

The fake news contains text and metadata from 244 websites and represents 12,999 posts in total posted across 30 consecutive days, tagged by the 'BS Detector' Chrome Extension by Daniel Sieradski (https://github.com/bs-detector/bs-detector). Each website is classified according to the BS Detector. Our group collected (accessed and downloaded) the fake news data from the Kaggle platform, at https://www.kaggle.com/mrisdal/fake-news.

The real news was sourced from a Kaggle dataset named "All the news": https://www.kaggle.com/snapcrack/all-the-news/home. The publications include the New York Times, Breitbart, CNN, Business Insider, the Atlantic, Fox News, Talking Points Memo, Buzzfeed News, National Review, New York Post, the Guardian, NPR, Reuters, Vox, and the Washington Post. The data falls between the years 2015 and 2017.

3.2. Dataset description

Fake News. This dataset classifies news items described by a set of attributes such as "Bias", "BS", "Conspiracy", "Fake", "Satire". It has 12,999 observations and 20 attributes (columns) – 11 continuous and 8 categorical; and the classification variable, 'Type'. The table below shows the list of variables, and their data types and descriptions.

Attribute	Variable Name	Variable Description
1 (numerical)	uuid	unique identifier

2(numerical)	ord_in_thread	order in thread
3(qualitative)	author	author of story
4(date)	published	date published
5(qualitative)	title	title of the story
6(qualitative)	text	text of story
7(qualitative)	language	data from webhose.io
8 (date)	crawled	date the story was archived
9 (qualitative)	site_url	site URL from BS detector
10(qualitative)	country	data from webhose.io
11(numerical)	domain_rank	data from webhose.io
12(qualitative)	thread_title	thread title
13(numerical)	spam_score	data from webhose.io
14(qualitative)	main_img_url	image from story
15(numerical)	replies_count	number of replies
16(numerical)	participants_count	number of participants
17(numerical)	likes	number of Facebook likes
18(numerical)	comments	number of Facebook comments
19(numerical)	shares	number of Facebook shares
20(qualitative)	type	type of article (label from BS Detector tool)

Real News. This dataset contains news items described by a set of attributes such as publication date, publisher, URL, author. It has 49,920 observations and 10 attributes (columns) – 4 continuous and 6 categorical. The table below shows the list of variables, and their data types and descriptions.

Attribute	**Variable Name**	**Variable Description**
1 (numerical)	id	unique identifier
3(qualitative)	title	article title
4(date)	publication	publication name
5(qualitative)	author	author name
6(qualitative)	date	date of publication
7(qualitative)	year	year of publication
8 (date)	month	month of publication
9 (qualitative)	url	URL for the article
10(qualitative)	content	article content

3.3. Dataset preparation (for combined real and fake news)

The Jupyter Notebook called News_DataPrep_EDA.ipynb on GitHub in the 'notebooks' folder documents the steps for combining real and fake news.

A first step in the data preparation was to merge the two data sources, of real and fake news.

The aim was to obtain a balanced mix of real and fake news items, suitable for modeling purposes. The datasets were first matched by month of publication: In the real news dataset, the data falls between the years 2015 and 2017, and for our project, in order to match the time period in which the fake news articles were published (November 2016), we only used news articles published in October to December 2016.

In the real dataset, we aimed to select several reputable publications, as well as several other 'less-reputable' publications, in order to have a balanced representation of news published by a real news publication company. Those included are listed below. We used a media bias/fact check website (source: https://mediabiasfactcheck.com/) to assess each news publication company. The results are below (each company is rated on their factual reporting).

Factual reporting:

- Very High: Reuters, NPR
- High: new york times, Atlantic, guardian, Washington post, vox
- Mixed: fox news, CNN, national review

From the fake news dataset, we selected English articles only, and also only those articles labeled "BS" by the Chrome BS Detector. We created a Target variable called 'fake' that has a value of 0 for real news articles and 1 for fake news items. The merged real and fake news dataset had 9 columns, 8 of which were common to both data sources, and the Target variable ('fake'), and 26612 rows (news articles).

4. Exploratory Data Analysis

4.1. Data exploration

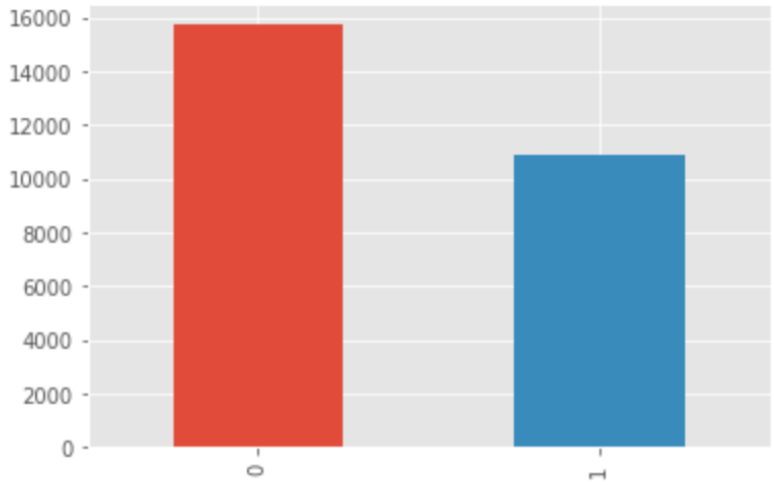
The Jupyter Notebook called News_DataPrep_EDA.ipynb on GitHub in the 'notebooks' folder includes the details of our Exploratory Data Analysis (EDA).

In any NLP project, a first activity is to prepare the data for model building, which involves loading the dataset into Python, exploring the data, and performing basic pre-processing and cleaning.

In the Jupyter notebook News_DataPrep_EDA.ipynb, Summary statistics were computed, and patterns and distributions were charted. The 'Title' column in the fake news dataset had missing values; however, we noticed that the 'Title' and 'Thread_title' variables were the same. We, therefore, replaced the missing values in 'Title' with the values from 'Thread_title'.

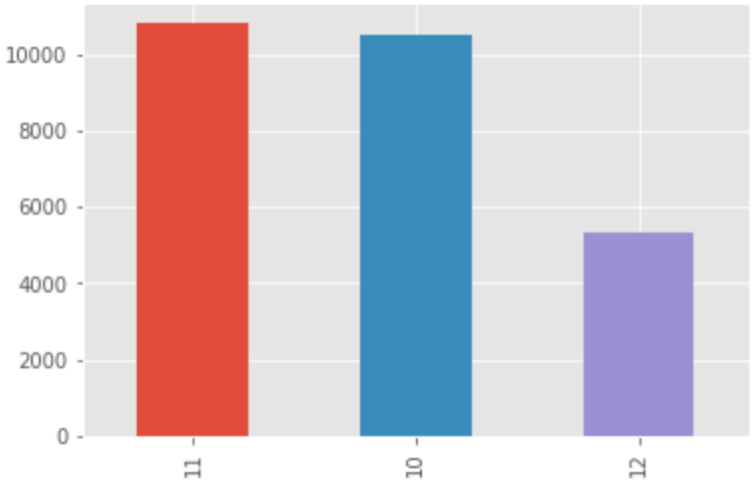
As noted above, only English-language news records were kept – filtered by 'Language' in the fake news dataset. We also created a 'Month' variable extracted from the publication date from the real and fake news datasets.

The following bar graph shows the distribution of our Target variable, 'fake'.

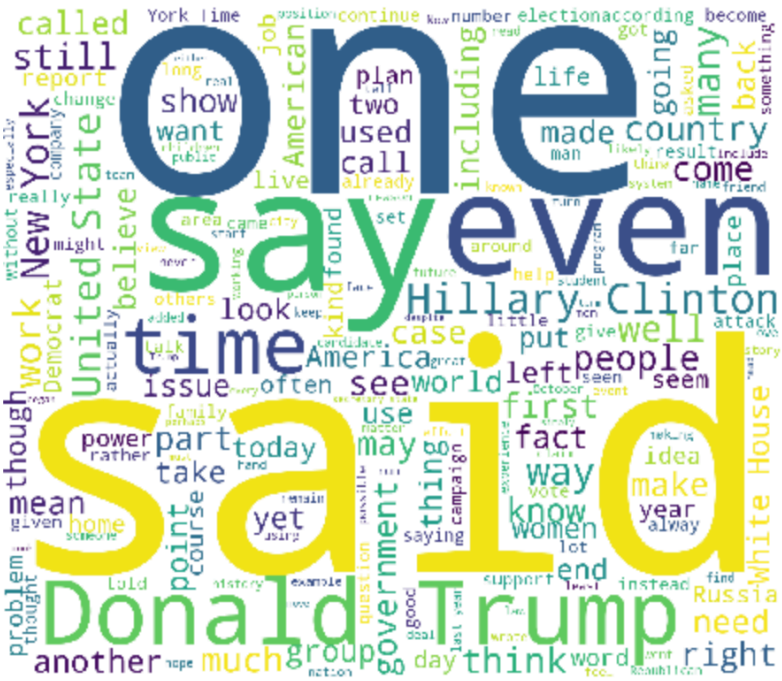


There are 15725 real news articles and 10900 fake news articles in the data. The Target variable was used for building supervised machine learning classification models.

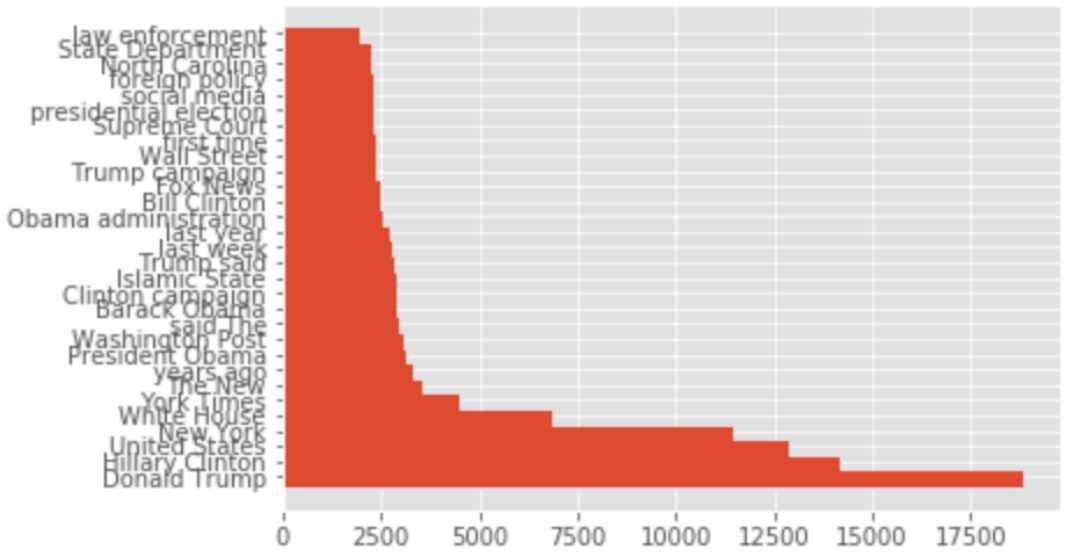
The following bar graph shows the distribution of the 'month' variable. The counts for articles published in the months of October, November, and December 2016 were 10483, 10816, and 5313 respectively.



We also performed exploration of the concatenated news articles' title and text, using Python's NLTK library. After tokenization and stopword removal, we created a WordCloud (or Tag cloud), a data visualization technique used for representing text data in which the size of each word indicates its frequency or importance. The WordCloud on our news data reveals that words or pairs of words such as "one", "said", "Donald Trump", "time", "Hillary Clinton", and "New York" have the highest frequency in the text data.



Further, we did a bigram (word pairs) analysis using NLTK to generate all possible bigrams and select the ones with the highest frequency. The following visualization shows the most frequent bigrams in the news text data; word pairs with the highest frequency include "Donald Trump", "Hillary Clinton", "New York", "White House", "United States", "President Obama", "Washington Post", "Trump said", "Wall Street", "Islamic State", and "last week".



4.2. Text preprocessing and normalization

One way to process the text from our news dataset to build models to flag fake news would be to analyze the text from the headlines (article title) only. However, based on previous work done in the area of fake news detection (e.g., in this project: https://medium.com/@Genyunus/detecting-fake-news-with-nlp-c893ec31dee8), the text from the headline only is not sufficient for accurate classifications and predictions. That leads to the following 3 alternatives:

1. Build classification models on the news body.
2. Build classification models on the concatenated texts of news body and news title. However, we lose information regarding what constitutes a title and what not.
3. Build ensemble models based on classifiers trained on news body and news title, and possibly other features such as authors, dates, etc.

In the experiment done for this project proposal, only the 1st option is implemented and evaluated with machine learning algorithms. We plan to experiment with options 2 and 3 in the next phase of the project.

Normalization is done following these steps:

1. Strip HTML tags.
2. Remove accented character
3. Contraction expansion
4. Convert all letter to lowercase
5. Remove redundant newlines
6. Insert spaces between special characters to isolate them

7. Lemmatize text
8. Remove special characters
9. Remove redundant white spaces
10. Remove stop words

The normalization steps, and code sections, were taken from the in-class materials provided at the ML1010 GitHub page
/ML1010_InClass/Day1/3_sentiment/Text_Normalization_Demo.ipynb

To generate normalized news body and news titles, run the following command:

python3 codes/normalization.py

5. Feature Engineering

Our experiments on feature extraction and modeling are documented on GitHub in the notebook found at "notebooks/model_evaluations.ipynb".

Three categories of features are used in the experiments for the proposal submission: count-vector, term frequency-inverse document frequency (tf-idf), and word embedding. These three methods from Python's scikit-learn library.

5.1. Count-vector

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary. Implementation steps are as follows.

a. Create an instance of the *CountVectorizer* class

b. Learn a vocabulary from one or more documents to determine what features CountVectorizer will base its transformations on

c. Use a function that transforms words into vectors from the train and test data, to encode each as a vector.

5.2. Word frequencies (TF-IDF)

One of the popular methods for counting the frequencies of words is called TF-IDF. This is an acronym that stands for "*Term Frequency-Inverse Document Frequency*", which are the components of the resulting scores assigned to each word.

Term Frequency: This summarizes how often a given word appears within a document.

Inverse Document Frequency: This downscales words that appear a lot across documents.

TF-IDF is word frequency scores that attempt to highlight words that are more interesting, e.g., frequent in a document but not across documents.

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allows the encoding of new documents. It is implemented as follows.

a. Create an instance of the TfidfVectorizer class

b. Learn a vocabulary from one or more documents to determine what features CountVectorizer will base its transformations on

c. Use a function that transforms words into vectors from the train and test data, to encode each as a vector.

5.3. Word embedding

A pre-trained word vector file (wiki-news-300d-1M.vec), trained on 1 million word vectors on Wikipedia 2017, UMBC web base corpus and statmt.org news dataset (16B tokens), are downloaded from this source [https://fasttext.cc/docs/en/english-vectors.html].

The word vectors are only used in CNN in the experiments. All the unique words from the news data are first extracted. Then an embedding matrix is built, which is essentially a dictionary whose keys are words and whose values are the word vectors taken from the downloaded file. As seen from the network structure, the embedding matrix is then set as layers immediately following the input layers.

In the next phase of the project, we plan to experiment with additional word embedding such as the GloVe model. We can also experiment with word embedding with other classifiers such as logistic regression or random forest.

6. Feature Selection

Codes for feature selections are documented in "notebooks/ Feature_Selection.ipynb".

During the experiments for this project proposal submission, we have tried feature selection methods. However, we have not included them into the machine pipeline. We plan to experiment more with feature selection in the next phase of the project.

Feature selection can be used to identify the most important features to include in a model. It can be used to reduce the training time of the models and to reduce the risk of overfitting the training dataset. We are planning to implement and compare the results of the following

methods: f-test, mutual information, variance threshold, recursive feature elimination, and model-based feature selection.

F-test and mutual information are both filter methods, that determine the importance of features based on the relationships between the features and the target variable. F-test is a statistical test that can be used to determine whether the difference between models is significant. The test is done to determine the best model that fits a dataset. The f-test allows us to determine if the inclusion of a feature in a model results in a significant improvement. Thus, it allows us to rank the importance of the features. Mutual information is a measure of the dependence of one variable on another. We can rank features based on their mutual information with the target variable. Unlike the f-test, mutual information can do well with the non-linear relationship between a feature and the target variable.

Variance threshold is used to eliminate features that have variances that are below a pre-specified threshold.

Recursive feature elimination is an iterative feature selection method which starts by building a model with all the features and discards the least important features based on the model. A new model is then built with the remaining features and this step is repeated until only a pre-specified number of features remain. We will use recursive feature elimination to build a random forest classifier for each step.

Model-based feature selection takes advantage of the fact that certain classifiers, such as random forests, allow us to rank the importance of the features and this allows us to keep only the most important ones. The models that are used for feature selection don't need to be the same models that are used for the final supervised modeling. We will specifically use random forest and extra tree classifiers for model-based feature selection.

7. Modeling

The experiments on feature extractions and modeling are documented in the GitHub notebook at "notebook/model_evaluations.ipynb".

Modeling and evaluation procedures are documented as follows.

a) Cross-validation

Following data preparation, the news dataset was split into train and validation sets. In the first phase of experiments, we divided the dataset in the following way:

- Training set: 80%
- Validation set: 10%
- Test set: 10%

Due to the concern of long training time on the neural networks, we have not used K-fold or

repeated K-fold as our cross-validation methods. In the next phase of the project, we will incorporate K-fold for classifiers other than CNN. If given access to GPU, we will also perform K-fold for CNN.

With the cross-validation mechanism in place, we trained our models on the training set. Then, we evaluated the models on the validation set by calculating the AUC and accuracy metrics. Based on the AUC scores on the validation set, we selected the best model and features. Then a model was built by training the selected model on the training and validation set. Finally, the accuracy and AUC score is reported for the selected model on the test set.

b) Logistic Linear Regression

We started with a simple Logistic Linear Regression, to get our hands on the project. We used Logistion Regression method on both '**train_text_ctv**' and '**train_text_tfv**'.

c) Naive Bayes

Naive Bayes is a family of algorithms based on applying Bayes theorem with a strong (naive) assumption, that every feature is independent of the others, in order to predict the category of a given sample. They are probabilistic classifiers, therefore will calculate the probability of each category using Bayes theorem, and the category with the highest probability will be output. Naive Bayes classifiers have been successfully applied to many domains, particularly Natural Language Processing

d) Convolutional Neural Network

For this project, we also experimented with convolutional neural networks (CNN). For the proposal submission, we tried CNN with word vectors from "wiki-news-300d-1M.vec" (downloaded from this source [https://fasttext.cc/docs/en/english-vectors.html]). Also, since the news articles have varying length, we have padded the vectored input with 0s so that all input vectors have the same length – 14736 in this case. The number "14736" is the maximum number of words in one news item among all the news.
The network structure in our experiment is as in the table below. As seen, this is a simple network with only a few layers, with max pooling and drop-out layers as regularization methods. Yet, as will be seen in the section "summary of findings/recommendations", this model still exhibits an over-fitting phenomenon even with the presence of such regularization layers.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 14736)             0
_________________________________________________________________
embedding_1 (Embedding)      (None, 14736, 300)        40498500
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 14736, 300)        0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 14734, 100)        90100
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 50)                5050
_________________________________________________________________
dropout_1 (Dropout)          (None, 50)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 51
=================================================================
Total params: 40,593,701
Trainable params: 95,201
Non-trainable params: 40,498,500
_________________________________________________________________
```

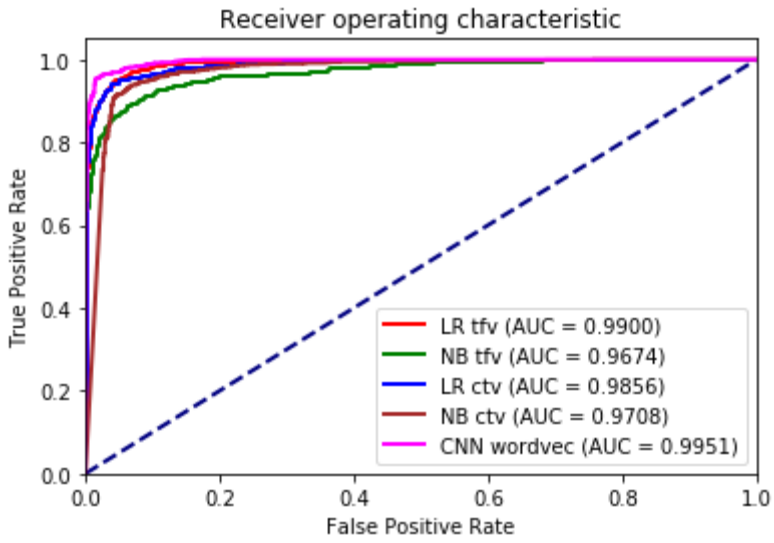
8. Models Evaluation

The experiments on feature extraction and modeling are documented on GitHub in the notebook at "notebook/model_evaluations.ipynb".

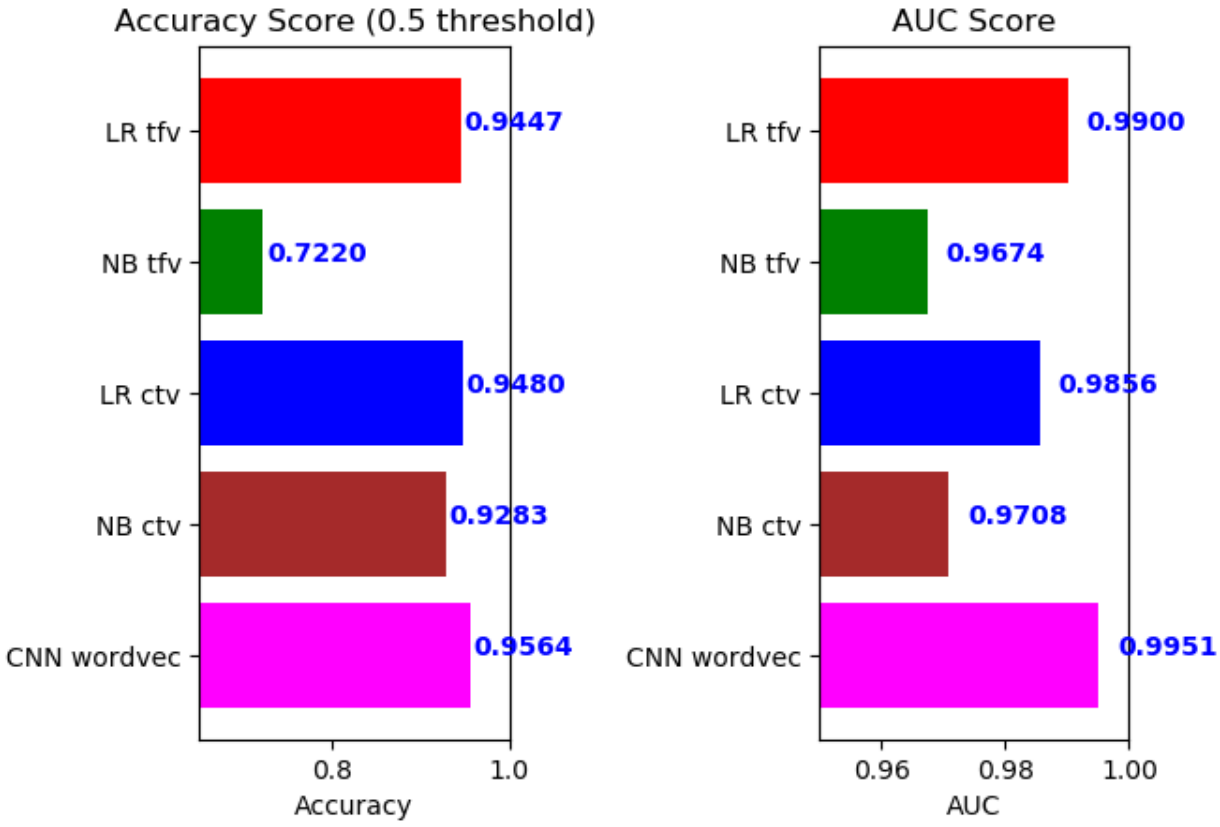
The following table summarizes the models' accuracy on the training set as well as the validation set.

Feature	***Count-vector***	***TF-IDF***	***Count-vector***	***TF-IDF***	***Word Embedding*** (wiki-news-300d-1M)
Algorithm	***Logistic Regression***	***Logistic Regression***	***Naive Bayes***	***Naive Bayes***	***CNN***
Accuracy on the training set	***0.9999***	***0.9778***	***0.9970***	***0.7553***	***0.9975***
Accuracy on the validation set	***0.9480***	***0.9447***	***0.9283***	***0.7220***	***0.9564***

The following plot depicts the ROC/AUC evaluation results performed on the validation set.



And the following plots depict the evaluation results in a comprehensive way, with both the accuracy and AUC score results in one figure, to help us visually select the best models.



9. Summary and Next Steps

We summarize our findings and thinking from the initial experiments as follows.

1. Overfitting. From the comparison of accuracy on both the training set and validation set, we clearly see that top performers such as CNN and count-vectors with logistic regression have an overfitting problem. This should be further verified with K-fold cross-validation. But we also think about adding heavier regularization to these classifiers.
2. Efficiency and accuracy. Logistic regression with count vectors is fast and achieved a 99.99% accuracy on the training set. Logistic regression is fast! No need for GPU. As compared with CNN + word embedding, CNN offers better generalization capabilities (less gap between training accuracy and validation accuracy), but at the expense of much higher demand for computing resources and time.
3. When we convert our tokens to count-vector features, we might accidentally introduce overfit. We have fit the count-vectors to the whole dataset we have. Yet, we probably should have fit it only to the training set. We should experiment on this in the next phase of the project.
4. From the perspective of AUC and accuracy on the validation set, we selected the winner: CNN + word embedding. We then trained this model on the training set and validation set. With the updated model, the accuracy on the test set is 97.47%, and the AUC score is 0.9967.

In the next phase of the project, items that the team considers pursuing are as follows.

1. Perform K-fold cross-validations on the classifiers other than CNN. Perform K-fold on CNN if we have access to GPU.
2. We have three categories of features and three types of classifiers. There is a total of 9 combinations but we have only experimented with 5 of them. In the next phase, we will exhaust all 9 combinations.
3. Optionally experiment with 3 more algorithms/classifiers: SVM, random forest, xgboost.
4. Optionally experiment with one more word embedding such as GloVe.
5. Experiment with heavier regularizations on logistic regression and Naive Bayes to see if they can reduce the overfit.
6. Optionally experiment with deeper CNNs and heavier regularizations.
7. Build ensemble models. Think about how to use the rest of the features in the dataset, such as the news title, authors, date, etc.
8. Experiment with feature selection methods and evaluate the impact on accuracy and AUC.