News Category Classification During Covid-19 Pandemic

Yifan Zhang
Li Du
Tianqi Cao

Abstract

The number of Covid-19 related news booms as the pandemic breakout and spread. When news flood in, text mining technique could be helpful in efficiently classifying articles into different genres, so that news receivers could decide what to read according to the genre labels. In this passage, two main kinds of text analysis method will be applied and several prediction models will be evaluated to determine a relatively accurate text classifier.

1 Introduction to dataset

The dataset *Covid-19 Public Media Dataset by Anacode* from *Kaggle* contains 369,047 records of information of Covid-19 related news articles published during 2020. There are 7 aspects of a news articles in each record.

'author' is the name of a writer or an organization. This column contains missing values. There is some noise text author's name, such as email address; *'date'* is the publish date; *'domain'* is basically the name of the media platform or publishing organization; *'title'* is the headline of a report; *'url'* is the web link to each passage. It may contain information of domain, author and title, but such information is knitted irregularly, which makes it difficult to extract information from it. *'content'* is the news article; and *'topic_area'* is the genre of news, and news were initially classified into 11 types. Some genres have very small amount of records, so close genres could be combined.

2 EDA

Graphs will show distribution patterns of data, and distributions may reveal connection or trend between predictors and response variable. Distributions may also provide clues to better preprocess data.

The original news classification is not quite reasonable. For example, as AI is a branch of technology, these two types should be combined. For types which have fewer samples are combined based on likeness in information contained. *'finance'* and *'business'* also share high similarity, but these two types have big shares in records, so it is better not to bin them into one label to further increase imbalance in the data.

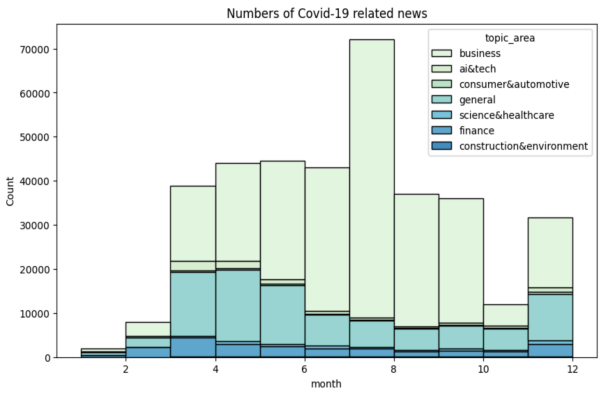

Figure 1

After the data relabeling, the stacked bar chart still demonstrates the severe imbalance in the numbers of Covid-19 related news in each month. Business information's share was higher than expected because economy was a growing concern as countries started lockdowns. *'general'* is the second in size as it includes news from less professional but more interesting aspects, such as entertainment and sport.

There is also fluctuation in news numbers in different months. For example, at the beginning of the pandemic breakout, *'science&healthcare'* takes up about one third of relevant news, however, as the public have obtained general knowledge about the disease, scientific report becomes less popular. The drop in November is interesting, and it would be reasonable to assume that the number of politics news have a sharp rise because of the U.S President Election in November. Therefore, date could be a useful label to include into classification models.

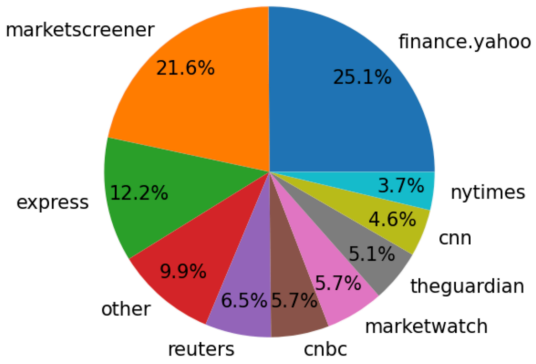

Figure 2

'domain' also seems to be a powerful predictor, as there are highly indicative words in domain names.

3 Data Preprocessing

A sub-dataset which contains about 2000 records of each type is randomly sampled from the original dataset.

First, as author usually specifies his or her opinion or purposes in the beginning of a passage, only the first five sentences are used to predict news types. This action is mainly meant to reduce computing time and decrease memory utilization. Stop words and punctuations are removed from sentences, and the remaining words are stemmed to root forms. The output from sentences preprocessing is list of tokens. Many forms of data can be generated from the tokens, such as *TF-IDF* scores and word vectors.

Second, use *datetime* library to extract month from *'date'* and form a new column *'month'*

Third, transfer categorical variables *'month'* and *'domain'* into suitable forms. For regression models, *LabelBinarizer* from *Scikit-learn* transforms categorical variables into vectors consists of 0 and 1; for tree models *LabelEncoder* from *Scikit-learn* will suffice. This function simply transforms categorical variables into a column of numbers.

When word preprocessing is done, records are randomly divided into three groups: 80% of records into train dataset which is used to train models, 10% of records into validation dataset which is used to check model's prediction ability and provide clues for tuning parameters, and 10% of records into test dataset which is used to evaluate the overall performance of tuned model.

4 Introduction to different Models

Before starting our prediction, we would like to introduce some models which we will use later.

4.1 Logistic Regression

The first model which is the fundamental one we will use for both bag of words and word embedding is logistic regression. Unlike the linear regression model, the logistic regression is an algorithm to do the classification and the function of it is sigmoid function which map the input into an output between zero and one. The formula and the plot of the function are shown here:

$$f(x) = 1/(1 + e^{-x}).$$

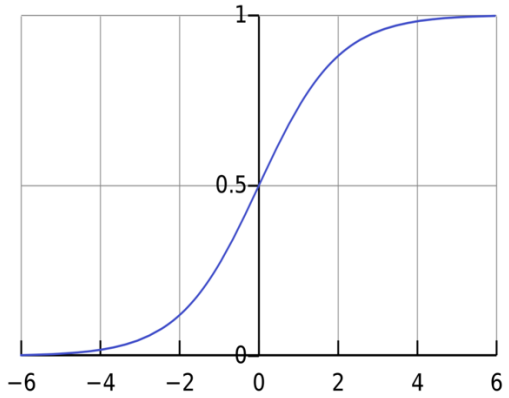

Figure 3

After introducing the basic concept of *logistic regression*, we will also introduce simple application of it. For our project, we will use logistic regression model from *Scikit-learn* and after tuning the hyperparameters, we will fit the model on the train set to build our logistic regression model. Finally, we could use it to predict on test set.

4.2 Extreme Gradient Boosting

Another model we would like to introduce is *Extreme Gradient Boosting*, also known as *XGBoost*. *XGBoost* is a tree ensemble model which is powerful and popular in recent years. Unlike the simple decision tree, *XGBoost* gives the results based on multiple predictions from many trees and we could control all the features of the tree such as depth and leaf nodes.

The application of *XGBoost* is similar to logistic regression in python. We just need to tune different hyper parameters, and then train and fit on our dataset.

4.3 Neural Network

The last model we would like to introduce is neural network model. "Neural networks are so called because they are designed after the way the brain is believed to process and store information. The human brain is composed of hundreds of billions of neurons. Each neuron has a very simple design…Neural networks work in a similar fashion." (*Ainslie, Dreze 7*). Normally, there are three different types of layer in neural network which are input layer, hidden layer and output layer. After setting the appropriate number of node and layer for hidden layer, we could build a strong model.

The application for neural network in python is also similar to first two models. We will use *MLPClassifier in Scikit-learn* to build our neural network model.

5 Bag of words

In natural language processing, we have many models to process the text and we would like to introduce the most basic one, which we will also use for this project, called bag of words.

"The bag-of-words (BOW) model is a representation that turns arbitrary text into fixed-length-vectors. This process is often referred to as vectorization" (*Zhou*). In fact, computer could not accept text directly and process it. Therefore, many people use bag of words model to transform text into vectors which computer is able to recognize and process.

For a deeper understanding of the bag of words model, we would also list the advantages and disadvantages of it.

First of all, bag of words is very easy to learn and use. The concept of it is not complex and there are not many mathematical theories and formulas behind it.

However, the first problem of bag of words is that it may generate a high dimensional vector when vocabulary is large. This will lead to a sparse matrix.

Moreover, bag of words does not consider word order and semantics which may lead to a low accuracy when we want to do the classification.

After introducing the basic concept of the bag of words, we would like to apply it into our project. Actually, we could transform same text into three different vectors by different methods and we will introduce and try all of them and compare with each other to see which one is the best.

5.1 Binary

The first method is just transforming text to vector in binary form.

The mechanism is that with a vocabulary with n terms inside, each document will be transform into n-dimensional vector with either zero or one for each element. If a word is both in document and vocabulary, then computer will assign one to it in the vector, otherwise zero. I will give an example to show it.

Vocabulary = [I, love, watching, NBA, playoff, everyday]

Document: I love watching NBA, NBA love I.

Vector = [1, 1, 1, 1, 0, 0]

After introducing the binary method, we start to build our model for our project by using it. After doing EDA and Preprocessing, the first thing we need to do is to import *CountVectorize* from *sklearn.feature_extraction.text* and set the hyper parameter binary as True to make sure the model will transform our documents into binary vector. Then, we need to set y as *'topic_area'* column, also known as label column, in our dataset and set X by using *fit_transform* method from this model to transform content column into binary vector. The next step is to split train, validation and test sets by using *train_test_split* method from *sklearn.model_selection*. After splitting the X and y, we are able to tune and train the logistic regression model on train and validation sets. We can use predict method on test set when we finish tuning model. The last thing is to see the classification report from test set.

As the report shows, the accuracy is not good which is only about 0.69. Only two predictions

which for construction environment and consumer automotive are above 0.8.

	precision	recall	f1-score	support
ai&tech	0.59	0.69	0.64	194
business	0.57	0.42	0.49	194
construction&environment	0.88	0.64	0.74	131
consumer&automotive	0.83	0.85	0.84	190
finance	0.62	0.7	0.65	187
general	0.69	0.71	0.70	221
science&healthcare	0.73	0.82	0.77	192
accuracy		0.69		1309
macro average	0.7	0.69	0.69	1309
weighted average	0.69	0.69	0.69	1309

Table 1

5.2 Term Frequency

Another method we would like to introduce and use is transforming text into vector as term frequency form.

The mechanism of term frequency is similar to that of binary. The only difference is that if a word is both in document and vocabulary, computer will assign the number of word count in document to it in the vector, otherwise zero. I will give an example to show it.

Vocabulary = [I, love, watching, NBA, playoff, everyday]

Document: I love watching NBA, NBA love I.

Vector = [2, 2, 1, 2, 0, 0]

After introducing term frequency, we will also use it to build a model. Actually, the steps are almost same as binary and we just need to change the binary hyper parameter as False in *CountVectorize*. After training and fitting the new logistic regression model on dataset, we could see the classification report for term frequency method.

As the report shows, the accuracy improves a little bit to 0.7 and labels with good accuracy are still those two which are good in binary model.

	precision	recall	f1-score	support
ai&tech	0.67	0.70	0.68	194
business	0.57	0.47	0.51	194
construction&environment	0.86	0.69	0.76	131
consumer&automotive	0.87	0.85	0.86	190
finance	0.60	0.65	0.63	187
general	0.67	0.69	0.68	221
science&healthcare	0.72	0.85	0.78	192
accuracy		0.70		1309
macro average	0.71	0.70	0.70	1309
weighted average	0.70	0.70	0.70	1309

Table 2

5.3 TF-IDF

TF-IDF (term frequency–inverse document frequency) is a weighed algorithm used in information retrieval and data mining, most commonly used in mining keywords. It is widely applied in text cleansing for its high efficiency.

TF (term frequency) is the frequency of a term in a document. In our model, it is log transformed:

$$\mathrm{TF}(t, d) = log(1 + \mathrm{frequency}(t, d))$$

IDF (inverse document frequency) is represents the weight score of a word in a document. Usually frequent words are considered less important, and as the same time, rare words are assigned with higher weight.

$$IDF(term) = \log\left(\frac{N}{\mathrm{count}(d \in D : t \in d)}\right)$$

When we calculate terms' *IDF* score, for terms who appear less than 55 times in the who corpus, they will be regarded as very rare words and assign 1 to their IDF score.

For a word in a corpus, its *TF-IDF* score is :

$$TF\text{-}IDF = TF * IDF$$

This score is positively connected with its frequency in a doc, but negatively connected with its frequency in a corpus.

This method is easy to understand, however, its grading standard on a term's importance solely depends on its number of appearances, which is somewhat doubtful. Nevertheless, *TF-IDF* is still widely adopted.

We first train a logistic regression model with *TF-IDF* as the sole variable, and it turns out that the overall accuracy on test dataset is approximately 0.64, which is the lowest among all models using content as the only predictor.

An *XGBoost* model trained from *TF-IDF* scores has an accuracy score of 0.72. This model performs well on subgroups of *'science&healthcare'*, *'construction&environment'*, and *'consumer&automotive'*. Its low scores in *'ai&tech'* subgroup is a surprise, as this is a quite isolate topic. The overall performance is listed below:

	precision	recall	f1-score	support
ai&tech	0.60	0.68	0.64	194
business	0.65	0.65	0.57	194
construction&environment	0.86	0.86	0.79	131
consumer&automotive	0.87	0.87	0.87	190
finance	0.68	0.68	0.69	187
general	0.68	0.68	0.70	221
science&healthcare	0.78	0.78	0.82	192
accuracy		0.72		1309
macro average	0.73	0.72	0.73	1309
weighted average	0.73	0.72	0.72	1309

Table 3

6 Word Embedding

Several problems with the bag-of-words model.

- Dimensional disaster: If there are many words in the vocabulary, then each word is a vector with high dimension.
- No word order detected: The text is only seen as a bag of words.
- Semantic gap: The word vector cannot reflect whether two words are synonymous or not.

In order to solve the problem of bag-of-words model, we can build word vectors through language models.

6.1 Neural Network Language Model

In natural language, a sentence is composed of multiple words in sequential combinations. Simply put, a statistical language model is to calculate the probability of this combination.

Suppose there is a sentence W containing T words, then the probability that the sentence (words are combined in this order) is

$$p(W) = p(w_1^T) = p(w_1, w_2, \cdots, w_T)$$

Using the Bayesian formula, p(W) can be further decomposed as

$$p(w_1^T) = p(w_1) \cdot p(w_2|w_1) \cdot p(w_3|w_1^2) \cdots p(w_T|w_1^{T-1})$$

Then the problem will change from computing the probability p(W) to computing the probabilities $p(w_1)$ and $p(w_k|w_1^{k-1})$. In neural network model, the input is a sequence of words and the output is the probability $p(w_i|w_1^{k-1})$.

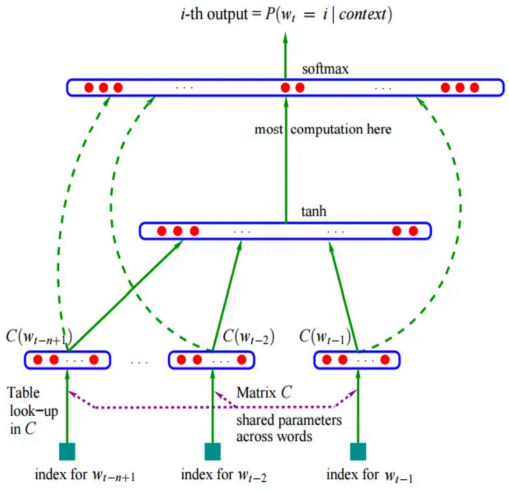

Figure 4

Figure 4. An overview of the network architecture of neural probabilistic language model (*Bengio*).

The neural network model in Figure 4 contains three layers. The first layer is the input layer x, then the second layer is reached by the nonlinear transformation "tanh" and finally the result is output by the nonlinear transformation *'softmax'*. The formula is as follows.

$$x = (C(w_{t-n+1}), \cdots, C(w_{t-2}), C(w_{t-1}))$$

$$y = softmax(b + Wx + Utanh(d + Hx))$$

When solving the neural network language model, the word vector stored in table c is what we want. The final output of the model, y, is instead not what we want. So the word vector is a by-product of the neural network language model.

6.2 Word2Vec

The neural network language model contains two nonlinear layers, which are complex to solve. Fortunately, more efficient neural network models have emerged.

In 2013, researchers of Google *Tomas Mikolov* published a paper *Efficient Estimation of Word Representations in Vector Space* and proposed an approach named *Word2Vec*. It uses small neural networks to calculate word embeddings based on words' context (*Mikolov*). There are two approaches to implement this approach.

6.2.2 CBOW

CBOW stands for *Continuous Bag-of-Words*. Unlike neural network language models, *CBOW* removed the nonlinear hidden layer which is the most time-consuming part. In this approach, the network tries to predict which word is most likely given its context. Words that are equally likely to appear can be interpreted as having a shared dimension.

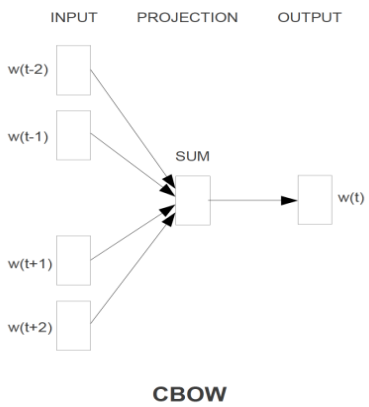

Figure 5

Figure 5: The CBOW architecture predicts the current word based on the context

From the figure, we can see that the CBOW model will predict

$$p(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$$

Since only two words are taken before and after the target word w(t) in the figure, the total size of the window is 2. Assuming that k words are taken before and after the target word w(t) , i.e., the size of the window is k, then the *CBOW* model will predict

$$p(w_t|w_{t-k}, w_{t-(k-1)}, \cdots, w_{t-1}, w_{t+1}, \cdots, w_{t+(k-1)}, w_{t+k})$$

6.2.3 Skip-Gram

The second approach is *skip-gram*. The idea is very similar, but the network works the other way around. That is, it uses the target word to predict its context.

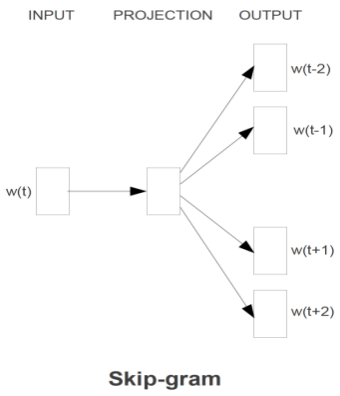

Figure 6

Figure 6: The Skip-gram predicts surrounding words given the current word.

From the figure, we can see that the Skip-Gram model will predict

$$p(w_{t-2}|w_t), p(w_{t-1}|w_t), p(w_{t+1}|w_t), p(w_{t+2}|w_t)$$

Since only two words are taken before and after the word w(t) in the figure, the total size of the window is 2. Assuming that the word w(t) is taken before and after each of the k words, i.e., the size of the window is k, then the Skip-Gram model will predict

$$p(w_{t+p}|w_t)(-k \leq p \leq k, k \neq 0)$$

6.2.4 Model Performance

In the COVID-19 news analysis, we tried to classify the news category by using Word2Vec model.

First, after preprocessing the news content, we built a vocabulary based on descending word frequencies of news contents. Then, we trained our *skipgram Word2vec* model with the corpus generated by ourselves. After that, we fed the logistic regression model with the word feature matrix in order to classify the topic area of these news. After further tuning our model with grid search, we got the model performance like this:

	precision	recall	f1-score	support
ai&tech	0.63	0.71	0.66	194
business	0.66	0.49	0.56	194
construction&environment	0.83	0.79	0.81	131
consumer&automotive	0.77	0.76	0.76	190
finance	0.64	0.76	0.69	187
general	0.79	0.70	0.74	221
science&healthcare	0.75	0.82	0.79	192
accuracy		0.72		1309
macro average	0.72	0.72	0.72	1309
weighted average	0.72	0.71	0.71	1309

Table 4

The accuracy and the f1-score of our model are both around 0.72, which is slightly better than the bag-of-words model. From the report, we can tell that our model performed great in predicting *'science&healthcare'* news (we reached 0.79 f1-score in this category). But we only got 0.65 f1-score with *'ai&tech'* news classification.

6.3 GloVe

GloVe and *Word2Vec*, both models can encode words into a vector based on the 'co-occurrence' (i.e. the frequency of occurrence of a word in the corpus) information of the words. The most intuitive difference is that word2vec is a "predictive" model, while GloVe is a "count-based" model (*Pennington*).

Count-based model is to reduce the dimension of the co-occurrence matrix. First, a co-occurrence matrix of words is constructed, where each row is a word and each column is a context. The co-occurrence matrix is to calculate the frequency of each word in each context (*Marco Baroni*). Since the context is a combination of multiple words, its dimensionality is very large, and we want to learn the low-dimensional representation of the word by dimensionality reduction on the context like network embedding. This process can be regarded

as the reconstruction problem of co-occurrence matrix, i.e., reconstruction loss.

6.4 Comparison of *Word2Vec* and *GloVe*

- Since *GloVe* can construct matrix on one time, so it is efficient for large corpora. But since this one-time cost could be expensive, it could be relatively slow for small or medium corpora.
- *GloVe* combined *Word2Vec* and *CBOW* to some extent. The difference is that, *Word2Vec* mainly focuses on local sliding windows, but *GloVe* is able to combine global and local features. So *GloVe* will be More flexible with the values in matrix.

In fact, the performance of *GloVe* model in our case is unsatisfactory. We only got 0.63 f1-score in the test dataset. It might because the corpus (Trained on Wikipedia) is not super related to the news contents.

7 Further Improvement

As content may contain noise tokens that smooth news feature, which make the it difficult to identify, therefore some more indicative features are added into variables to find out the best predictive model.

After transforming *'date'* and *'domain'* into forms that libraries can read and suitable for different types of models, *'date'* was first added to predict with content. The overall accuracy was only improved from 0.62 to 0.64. According to former stacked bar chart of monthly news, except in Jan, Feb and Mar, *'date'* seems not to be very indicative. If *'domain'* and *'content'* are used together to predict topic area, the overall performance will have a huge improvement. In logistic regression models, when *'domain'* and TF-IDF scores are used as predictors, the accuracy score would be 0.92; when *'domain'* and term frequencies are used as predictors, the accuracy score would be 0.98; when *'domain'* and word embedding vectors are used as predictors, accuracy score rockets to 1. In tree models and neural network models, when domain is added, models' accuracy scores are almost 1.

'domain' is a super powerful indicator in this case. When we further check *'domain'*, we find that, all media platform listed in this dataset publish only one type of news, which means, if *'domain'* is used as a sole predictor, model's accuracy would also be extremely high.

However, if *'domain'* is the only predictor, once a new domain name appears, the model's performance will endure a downgrade. The secure option is to use both *'domain'* and content as indicators. When a new domain name is inputted in, the model can still partially depend on content to make a prediction.

References

Victor Zhou. 2019. *A Simple Explanation of the Bag-of-Words Model.* https://towardsdatascience.com/a-simple-explanation-of-the-bag-of-words-model-b88fc4f4971

Ainslie and X. Dreze. Data Mining: *Using Neural Networks as a Benchmark for Model Building.* Page:1-18.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. *Efficient estimation of word representations in vector space. ICLR Workshop*

Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin. 2003. *A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155, 2003.*

Bruno Stecanella, *What is TF-IDF?* https://monkeylearn.com/blog/what-is-tf-idf/

Anonymous, *TF-IDF weighting*. https://nlp.stanford.edu/IR-book/html/htmledition/tf-idf-weighting-1.html

Omid Shahmirzadi, Adam Lugowski, and Kenneth Younge, *Text Similarity in Vector Space Models:A Comparative Study*

Jeffrey Pennington, Richard Socher, and Christopher D. Manning.2014. GloVe: Global Vectors for Word Representation.

Marco Baroni, Georgiana Dinu, and German Kruszewski,2014. *Don't count, predict! A systematic comparison of context-counting vs. context-predicting semantic vectors. Association for Computational Linguistics,*

Vishal Morde, Venkat Anurag Setty, *XGBoost Algorithm: Long May She Reign!* https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d