# Hashtags Recommendation On Twitter

#TagFinder

| | | | |
|---|---|---|---|
| Huaiyue Chang | 6365696861 | Manxin Wang | 9537808576 |
| Qianran Ma | 7798937244 | Wanning Li | 5898212888 |
| Yanan Zhou | 4366462176 | Yitong Wang | 1299603893 |

https://github.com/LucyLi2021/Hashtag-recommendation-for-twitter-data

Tweet

# Content

## 1. Executive Summary

Nowadays, Twitter is one of the most widely used social networking sites. Millions of users write Tweets to share life and spread information. Hashtag function is a popular and useful function in Twitter, which helps in categorization and retrieval of the posts based on the content and context. At present, Twitter has the function to recommend trending hashtags to users when they write a tweet, but fails to recommend content-related hashtags to users. Therefore, we are dedicated to realize the function to recommend top possible hashtags according to tweets content to users in this project. One tweet may have more than one hashtag that match the content, so that this project is actually a multilabel classification problem.

This report will cover the dataset preparation, models to solve the multilabel classification problem including TFIDF+LogisticRegression, FastText, LSTM, Text ResNet, and BERT, evaluations of all the models, error analysis based on Text Resnet, a simple demo where we deployed our models on simulated situation, and what we learned during the project and what we can continue doing in the future.

Our best model in this project is TFIDF+LogisticRegression, which achieved an NDCG@3/5/10 of 90% and MAP@3/5/10 of 93%. The errors of deep learning models (LSTM, Text ReNet, BERT) mainly came from lack of training data. They performed well on the first few labels, but poorly on the rest.

## 2. Dataset: Collection and Preprocessing

This section describes the procedures to collect Twitter text data and hashtags, preprocess data, and analyze the statistical features of the final dataset.

## 2.1 Data Collection

Data is collected via tweepy package. It's an easy-to-use Python library for accessing the Twitter API. To help models have a good performance, we focus on a specific "Brand" domain. The query filters make sure the data we get is not a retweet, has hashtags and is English. In the end, more than 700k data is collected with text, created time, hashtags and entity name columns. The data range is 10/25/2022-11/05/2022.

## 2.2 Data preprocessing

Data preprocessing contains four steps of data cleaning, data selection, encoding, and data splitting.

### 2.2.1 Data Cleaning

The dataset is composed of two parts: text data, and hashtags. Different strategies are used to clean these two parts of data. For the text data cleaning, the following steps are performed:

- Remove url, html, emoji;
- Lower letters;
- Remove punctuations except ",.!?:;";
- Word lemmatization;
- Remove hashtags at the end of the text, but retain hashtags in the middle of text.

For the text data cleaning, we didn't conduct a deep data cleaning, because some noise may be helpful to the training of deep learning models. For example, we didn't remove stopwords and some punctuations, including ",.!?:;", as the stopwords and some punctuations may be valuable in some deep learning models' training, like Bert. However, we have further cleaned the data when training models including Logistic Regression and FastText in the third section. The further cleaning strategies include removing all the punctuations, stopwords, and words not composed of all letters.

For the hashtags cleaning, the following steps are performed:

- Remove stopwords;
- Lower letters;
- Remove words not composed of all letters.

### 2.2.2 Data Selection

As there are hundreds of hashtags in the raw dataset, it is challenging to train satisfactory models to solve a hundred-class multilabel classification problem considering the limited data and time, so that we only select the data whose hashtags are in the top 50 frequent hashtag set. Therefore, our project is to solve a 50 class multilabel classification problem. The word cloud picture of the hashtags is shown below:

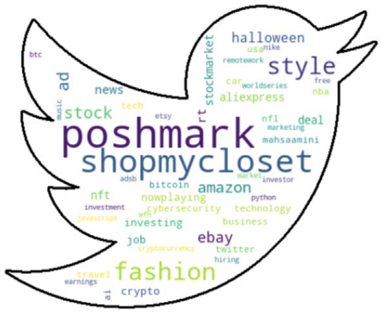


**Fig. 1.** Word cloud of top 50 frequent hashtags

### 2.2.3 Encoding

We used the multi-hot encoding strategy to deal with hashtag encoding. In this part, we did not encode the text data, which is encoded in the model training process.

### 2.2.4 Data Splitting

The dataset was splitted into training, validation, and test set. The training set size is 68,664, the validation set size is 40,960, and the test set size is 27,703. The total number of data is 137,327.

| **Dataset** | **Number** |
|---|---|
| Training dataset (50%) | 68,664 |
| Validation dataset (30%) | 40,960 |
| Test dataset (20%) | 27,703 |
| Total | 137,327 |

**Table 1.** Twitter Hashtag Dataset Statistics

## 2.3 Data Description

This part describes some features of our final dataset, which are useful in the model design.

For the text data X, the sentence length of most data is between 0 to 330, and only few data's sentence length is greater than 330.

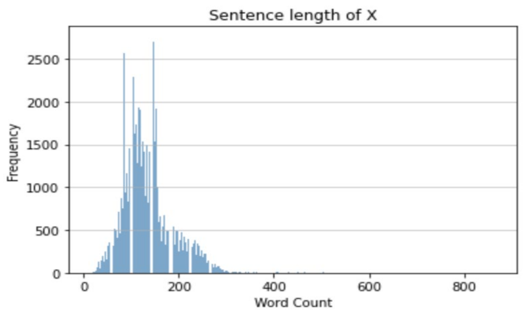


**Fig. 2.** Sentence length of Tweets

For the hashtags data y, most tweets have one or two hashtags (Fig.3). The hashtag with the index of 1 and 2 appears much more frequently than other hashtags (Fig.4). Therefore, the final dataset is actually not balanced. We have tried to use the downsampling and oversampling strategy to deal with the unbalanced data, but the result does not perform well due to the hashtags data's multilabel nature. Therefore, we use other methods to solve this problem during model training, which will be mentioned in section 4.

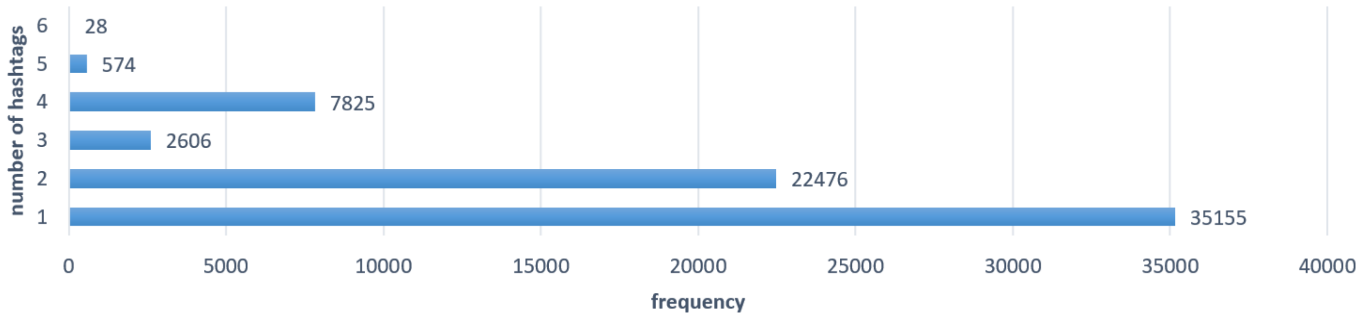


**Fig. 3.** Count of hashtag numbers

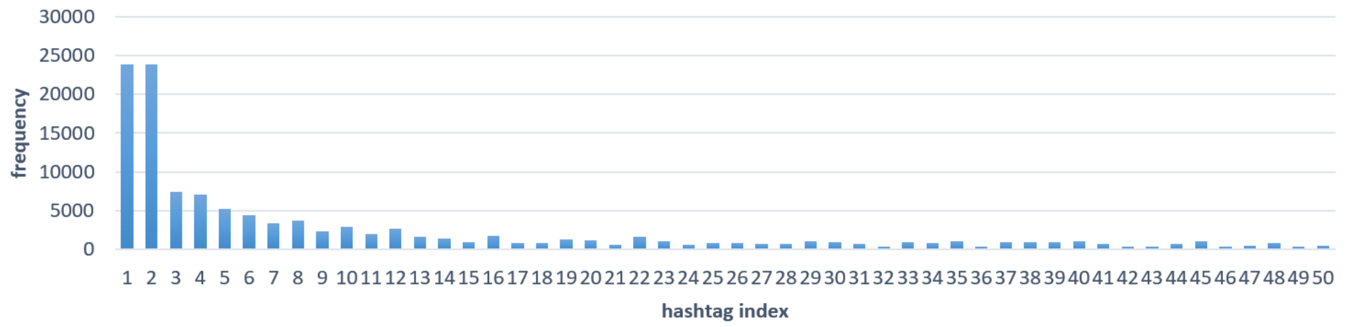


**Fig. 4.** Hashtag counts

# 3. Proposal Approach

This section is going to show all the models we used in the project. The technical pipeline is shown in Fig.5.

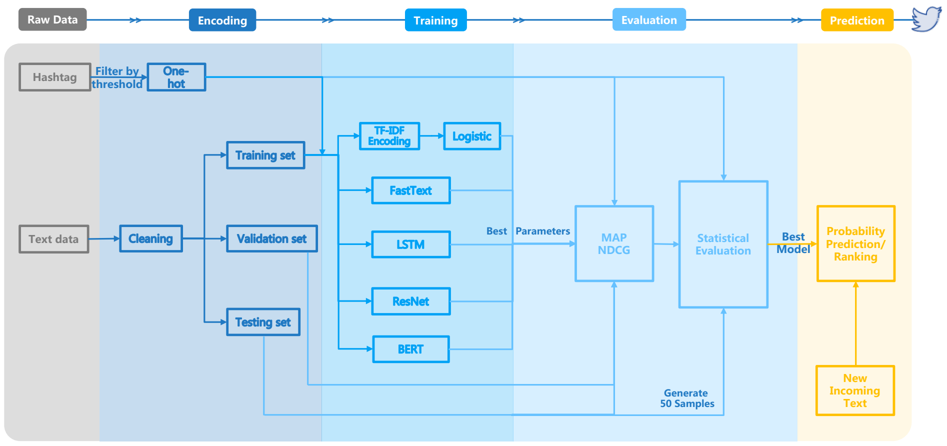


**Fig. 5.** Technical pipeline

## 3.1 Non-Deep Learning Models

In this section, we applied two popular non-deep learning language models on the task. The first one is TFIDF+LogisticRegression and the second one is FastText.

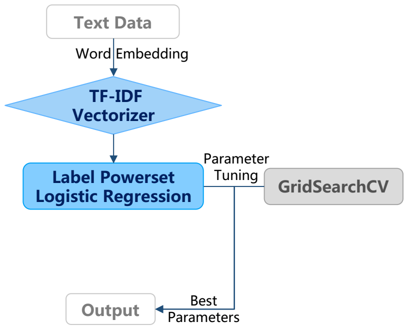


**Fig. 6.** Structure of TFIDF + Logistic Regression

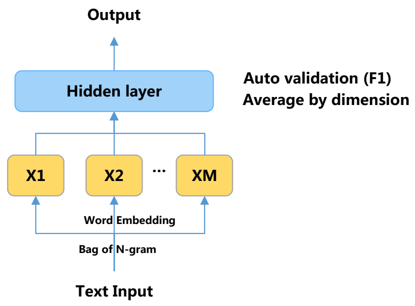


**Fig. 7.** Structure of FastText

### 3.1.1 TFIDF + Logistic Regression

Firstly, we tried TF-IDF vectorizer to extract features and chose logistic regression as the model. Based on linear regression, logistic regression introduces nonlinear factors through the Sigmoid function. After data preprocessing, our problem becomes a multilabel classification problem. Since logistic regression cannot solve multilabel classification directly, we use Label Powerset to transfer a multilabel classification problem to a multiclass classification problem. We use GirdSearchCV to tune the hyperparameters. The result shows that the best parameters are 10 for C, 100 for max iteration and l2 for penalty.

### 3.1.2 FastText

Developed by Facebook, FastText is an efficient yet strong solution for text classification and learning word embedding. It uses bag of N-gram to represent text and N = 1 in our case. The embedding of an instance is the average of individual tokens we got. And we adapt matrix vectors with 55 dimensions after tuning parameters on the validations set. For classifier implementation, FastText uses multinomial logistic regression with hierarchical Softmax, and organizes categories on Huffman tree. Thus, it takes less training time and memory than deep learning models in our classification task.

## 3.2 Deep Learning Models

A common difficulty we encountered during the model training step is we do not have enough resources and time to tune parameters and add more layers. After different attempts, here are our common settings.

| | |
|---|---|
| **Sequence Length = 200** | Truncate the embedding sequence if longer, pad with 0 if shorter |
| **Batch Size = 1024** | Drop last batch if less than 1024 for training and keep for validation and test |
| **Epoch = 30** | Finish training and store model checkpoint for each 10 epochs |
| **IDF Weight** | For binary cross entropy loss function to deal with imbalanced data |
| **Threshold = 0.3** | 1 if the predicted probability >= 0.3, else 0 |

**Table 2.** Common settings for deep learning models

### 3.2.1 LSTM

LSTM is short for long short time memory. For each unit, we have three inputs in one LSTM layer: long term memory, short term memory, and new input. In each unit, we use short term memory and new input to update long term memory and in return use long term memory to update short term memory.

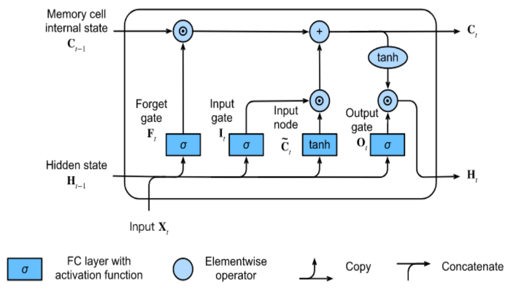


**Fig. 8.** The structure of a LSTM unit

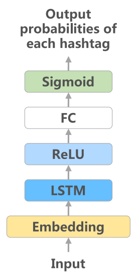


**Fig. 9.** The structure of LSTM

The whole LSTM model's structure is shown in Fig.9. The input is processed by embedding layer, LSTM layer, ReLU function, FC layer and Sigmoid function. The LSTM layer used is not bidirectional and not stacked.

### 3.2.2 Text ResNet

Resnet (Residual Network) applies a technique called skip connections, which connects activations of a layer to further layers by skipping some layers in between, to solve the problem of the vanishing or exploding gradient problem.

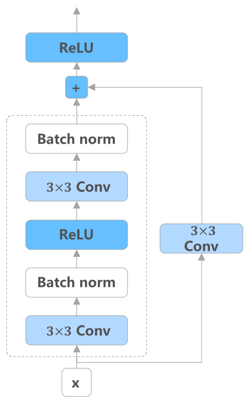


**Fig. 10.** The structure of a residual block

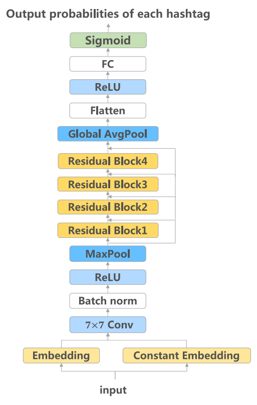


**Fig. 11.** The structure of Text Resnet

We first encapsulate the residual strategy into a residual block (Fig.10.). The residual block has two convolutional layers with the same number of output channels. Each convolutional layer is followed by a batch normalization layer and a ReLU activation function. Then, we skip these two convolution operations and add the input directly before the final ReLU activation function.

The whole Text Resnet model's structure is shown in Fig.11. We used two strategies of embedding: one is a Word2Vec embedding based on the dataset, which will be trained, the other strategy is a constant embedding based on the global vector, which remains the same during training. The first two layers of Text ResNet are a 7*7 convolutional layer with 64 output channels and a stride of 2, followed by the max-pooling layer with a stride of 2. The following layers are four residual blocks, and an Adaptive Average Pool layer. The final layer is a linear layer as a fully connected layer, and the sigmoid function is used as the activation function. The output is the probabilities of each hashtag.

Some optimization methods are also applied to improve the performance of Text Resnet model, including dropout with 0.5, Xavier weight initialization, batch normalization, and Adam.

### 3.2.3 BERT

BERT is a popular NLP model for language features extraction. There are 2 main ideas in BERT Designing:

- **EMLo structure:** it is originally built by forward and backward LSTM.
- **Self-attention layer:** in the Transformer Encoder layer, this layer is trying to capture the relevant information of a word with all the words in the sentence, which solves the problem that LSTM only goes in a single direction and can make use of the full context of a word.

Thus, replacing the LSTM with Transformer Encoder layer in EMLo, we get the basic structure of BERT.

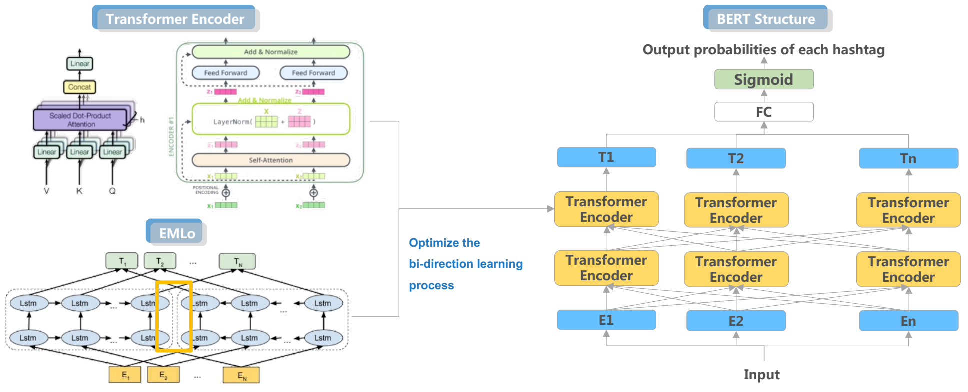


**Fig. 12.** The structure of BERT

Beside from the designing structure, the second reason that BERT is useful is related to the pre-training process, a pre-trained BERT will be trained in two ways to help understand sentence semantic features better:

- **Masked-Language Model:** making a mask on some words in the sentence and predict it by context.
- **Next Sentence Prediction:** identifying the relationship between two separate sentences.

Thus, BERT is a huge model with a huge size of parameters. In this project, we chose the pre-trained BERT model, which includes 110 million parameters. We cannot fine-tune and add more layers to the model due to the device limitation, we only added one more linear layer and a sigmoid layer after the model for our project.

# 4. Experiment Results

In this section, we are going to discuss and compare the models' performance based on NDCG@k, MAP@k, F1 Score and confusion matrix. In addition, to make the results more convincing, we also did statistical evaluation on 10 samples of data.

## 4.1 Benchmark – Random Model

A baseline model is necessary to act as a benchmark for more complex models trained. Therefore, we generated a random model for the multilabel classification problem as the baseline. We can regard each label as an independent random Bernoulli experiment. For each label, the estimated probability of Bernoulli should be the frequency of it in the training set.

Since when calculating NDCG@k and MAP@k, we are expecting to use the predicted probabilities to rank all the labels and get top k hashtags, and the predicted probabilities estimated from the training set are fixed all the time. Hence, for each new tweet, we would give it the same predicted probabilities.

| Label | Predicted probability | Label | Predicted probability | Label | Predicted probability | Label | Predicted probability | Label | Predicted probability |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.3481009 | 10 | 0.02908365 | 20 | 0.00895666 | 30 | 0.01128685 | 40 | 0.00997612 |
| 1 | 0.34795526 | 11 | 0.03927822 | 21 | 0.02413201 | 31 | 0.00630607 | 41 | 0.00643714 |
| 2 | 0.10814983 | 12 | 0.0241757 | 22 | 0.01504427 | 32 | 0.01334032 | 42 | 0.00557789 |
| 3 | 0.1026302 | 13 | 0.02022894 | 23 | 0.00902948 | 33 | 0.01160725 | 43 | 0.01112665 |
| 4 | 0.07554177 | 14 | 0.01457824 | 24 | 0.01175288 | 34 | 0.01615111 | 44 | 0.01604917 |
| 5 | 0.06444425 | 15 | 0.02590877 | 25 | 0.01165094 | 35 | 0.00632063 | 45 | 0.00502447 |
| 6 | 0.04983689 | 16 | 0.0124665 | 26 | 0.01099557 | 36 | 0.01438891 | 46 | 0.00723815 |
| 7 | 0.05451183 | 17 | 0.01175288 | 27 | 0.01019457 | 37 | 0.01338401 | 47 | 0.01168007 |
| 8 | 0.03496738 | 18 | 0.01856868 | 28 | 0.01526273 | 38 | 0.01341314 | 48 | 0.00535943 |
| 9 | 0.04348713 | 19 | 0.01766573 | 29 | 0.01328207 | 39 | 0.01533555 | 49 | 0.00696144 |

**Table 3.** Probabilities of hashtags

## 4.2 Models Evaluation

In this section, we collected the results of deep learning models in different epochs and compared all the models we applied in the projects with NDCG@k, MAP@k, F1 Score and confusion matrix to get the best model.

### 4.2.1 Deep Learning Models Performance on Different Epochs

To make the models trainable with limited resources, we trained models with training data and checked the models' performance every 10 epochs to make sure the model is not overfitted. The results of three models are shown in Table 4.

| | | NDCG@3 | NDCG@5 | NDCG@10 | MAP@3 | MAP@5 | MAP@10 | Time(s) | Model Size |
|---|---|---|---|---|---|---|---|---|---|
| Epoch=10 | LSTM | 0.7081 | 0.7291 | 0.7467 | 0.9489 | 0.9001 | 0.8482 | 39 | |
| | Resnet | 0.8287 | 0.8399 | 0.8399 | **0.9596** | **0.9352** | **0.9004** | 122 | |
| | BERT | **0.8595** | **0.8716** | **0.8823** | 0.9514 | 0.9252 | 0.8943 | **1574** | |
| Epoch=20 | LSTM | 0.7117 | 0.7322 | 0.75 | 0.9496 | 0.902 | 0.8501 | 79 | |
| | Resnet | 0.8319 | 0.8427 | 0.8427 | **0.9627** | **0.9396** | **0.9067** | 251 | |
| | BERT | **0.8745** | **0.885** | **0.8946** | 0.9535 | 0.9311 | 0.9034 | **3164** | |
| Epoch=30 | LSTM | 0.7126 | 0.7333 | 0.7508 | 0.9487 | 0.9006 | 0.8495 | 119 | 27.5M |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Resnet | 0.8338 | 0.8432 | 0.8432 | 0.9628 | 0.9422 | 0.9096 | 380 | 42.5M |
| | BERT | 0.8824 | 0.892 | 0.9009 | 0.9565 | 0.9361 | 0.9098 | 4753 | 436M |

**Table 4.** Deep learning models performance

From the metric values in Table 5, we can conclude that:

- BERT is the largest model and takes the longest time to train
- BERT performs the best on NDCG scores, higher than LSTM and ResNet about 5% on average. The results of LSTM are the lowest. ResNet performs better on MAP scores.
- The scores of BERT increases more rapidly than others with epoch number increases. The total NDCG increments of LSTM and Resnet are lower than 1%, but BERT is around 2% or more.
- As for the MAP score, the gap between ResNet and BERT is decreasing as epoch number increases, even at epoch 30, BERT performs better on MAP@10.

This means that BERT has the potential to be better with more training.

### 4.2.2 Models Comparison

As for the comparison between all the models, we calculated and evaluated the performance on test sets including precision, recall, f1 score, NDCG as well as MAP as shown in Table 5. We find that FastText has the highest micro scores, 0.66, and macro f1 scores, 0.55, and TF-IDF and logistic model has the highest NDCG@k scores at around 0.9, MAP@k scores at around 0.94 except for MAP@3. Thus, we can conclude that the TF-IDF and logistic model's performance is outstanding compared with all other models.

| | F1 Score | | NDCG | | | MAP | | |
|---|---|---|---|---|---|---|---|---|
| | Micro | Macro | @3 | @5 | @10 | @3 | @5 | @10 |
| Random | | | 0.3516 | 0.3799 | 0.4403 | 0.9850 | 0.8518 | 0.6556 |
| TF-IDF + Logistic | 0.65 | 0.51 | 0.9038 | 0.9109 | 0.9173 | 0.9681 | 0.9529 | 0.9338 |
| FastText | 0.66 | 0.55 | 0.8337 | 0.7379 | 0.6591 | 0.8587 | 0.6814 | 0.5659 |
| LSTM | 0.55 | 0.37 | 0.7126 | 0.7333 | 0.7508 | 0.9487 | 0.9006 | 0.8495 |
| ResNet | 0.62 | 0.48 | 0.8338 | 0.8432 | 0.8432 | 0.9628 | 0.9422 | 0.9096 |
| Bert | 0.64 | 0.52 | 0.8824 | 0.8920 | 0.9009 | 0.9565 | 0.9361 | 0.9098 |

**Table 5.** All models' evaluation performance on test set

Further, we conduct a detailed evaluation on TF-IDF and logistic model including the confusion matrix as shown in Fig.13, ROC curve shown in Fig.14, and precision vs. recall curve shown in Fig.15.

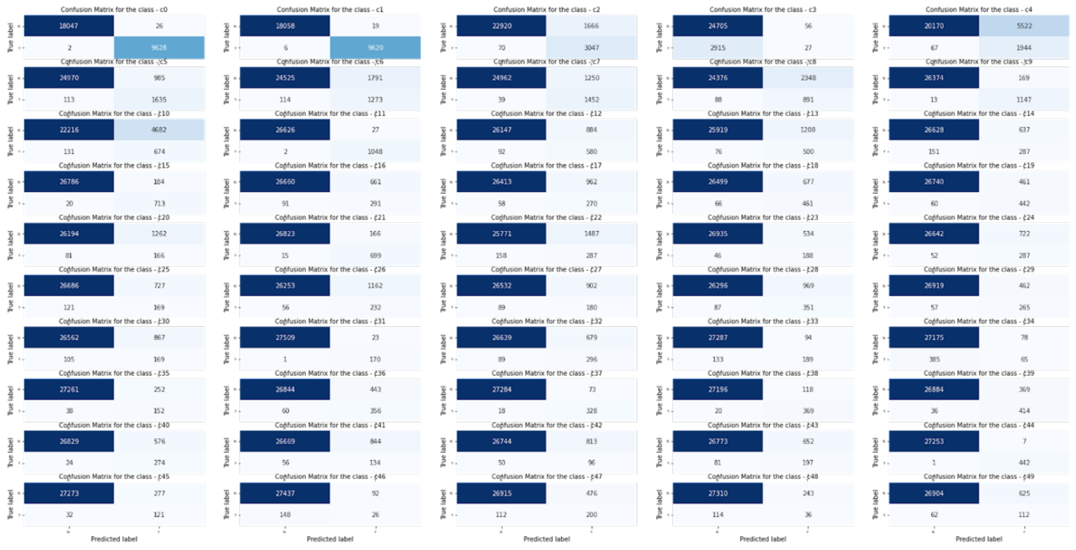


**Fig. 13.** The confusion matrix of TFIDF + Logistic Regression

For the ROC curve, we find that the model's all classes' overall ROC performance is excellent, and the AUC scores of commonly used hashtag classes are close or equal to one. Nevertheless, multi-classes imbalance shows in our precision and recall curve. The line tendencies' difference between the commonly used hashtag classes and the rarely used hashtag is large which means that our model does well in predicting high frequency hashtag but not the less frequency.

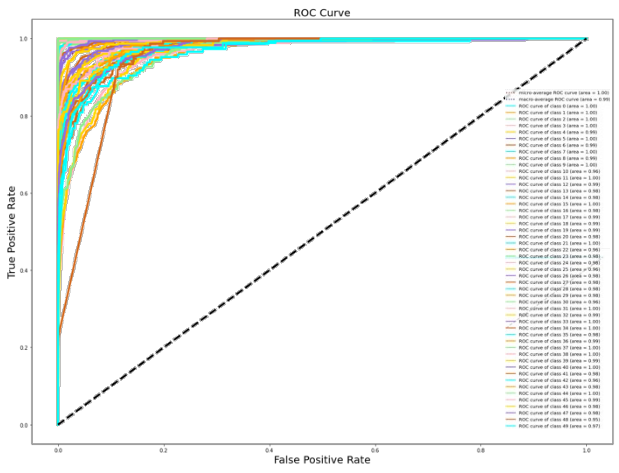


**Fig. 14.** The ROC curve of TFIDF + Logistic Regression

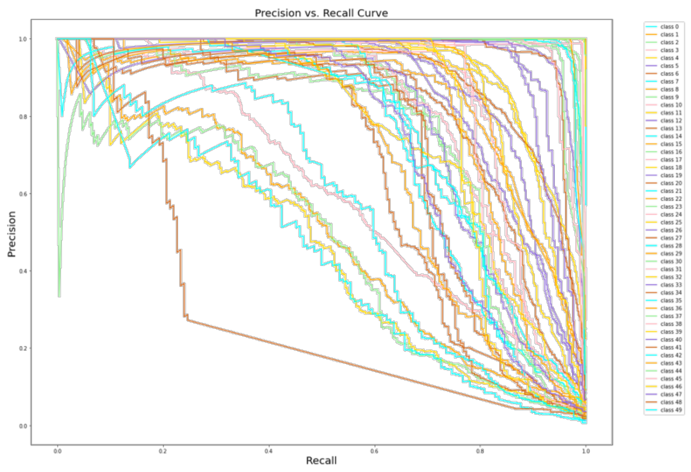


**Fig. 15.** The precision vs. recall curve of TFIDF + Logistic Regression

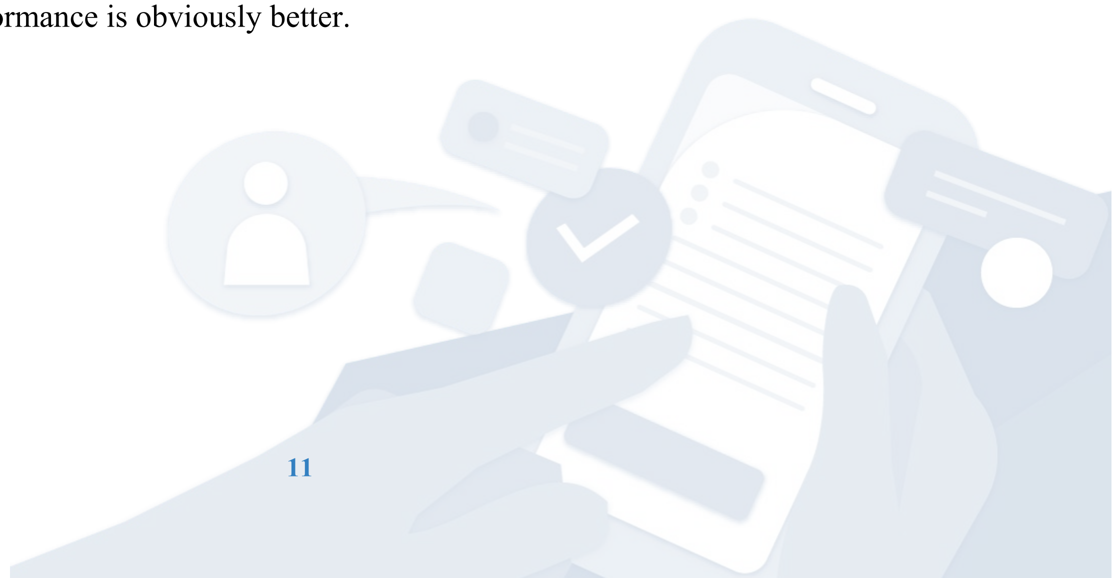
## 4.3 Statistical Evaluation

In this section, we did a statistical analysis for each model. We generate 50 samples from testset, predict each sample set and calculate NDCG and MAP scores, calculate 95% confidence interval to estimate population scores. In this way, we could have a better understanding of model performance.

| | NDCG@3 | | NDCG@5 | | NDCG@10 | | MAP@3 | | MAP@5 | | MAP@10 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Mean | Confidence Interval | Mean | Confidence Interval | Mean | Confidence Interval | Mean | Confidence Interval | Mean | Confidence Interval | Mean | Confidence Interval |
| Tfidf+Logistic | 0.9038 | (0.9004, 0.9073) | 0.9109 | (0.9077, 0.9141) | 0.9173 | (0.9144, 0.9202) | 0.9681 | (0.9662, 0.9699) | 0.9529 | (0.9510, 0.9549) | 0.9338 | (0.9312, 0.9364) |
| FastText | 0.4801 | (0.4745, 0.4857) | 0.8247 | (0.8201, 0.8292) | 0.5605 | (0.5557, 0.5654) | 0.6665 | (0.6615, 0.6716) | 0.6373 | (0.6336, 0.6411) | 0.5426 | (0.5380, 0.5472) |
| LSTM | 0.7093 | (0.7045, 0.7141) | 0.7297 | (0.7251, 0.7342) | 0.7480 | (0.7439, 0.7521) | 0.9478 | (0.9456, 0.9498) | 0.9001 | (0.8972, 0.9030) | 0.8471 | (0.8431, 0.8510) |
| ResNet | 0.8356 | (0.8321, 0.8391) | 0.8450 | (0.8416, 0.8483) | 0.8561 | (0.8529, 0.8594) | 0.9653 | (0.9635, 0.9671) | 0.9447 | (0.9424, 0.9469) | 0.9120 | (0.9091, 0.9149) |
| BERT | 0.8841 | (0.8806, 0.8877) | 0.8939 | (0.8907, 0.8971) | 0.9027 | (0.8997, 0.9056) | 0.9591 | (0.9571, 0.9612) | 0.9383 | (0.9358, 0.9408) | 0.9128 | (0.9101, 0.9155) |

**Table 6.** Statistical results

It shows that the lower range of TFIDF with logistic regression is better than the higher range of other four models, which means the performance is obviously better.



## 5. Discussion

In this section, we did further error analysis on deep learning models to figure out the reason why they did not perform so well. In addition, we summarized the conclusion we made based on all the work above.

### 5.1 Error Analysis

Based on the evaluation of all the models, we found that the simple Logistic Regression outperformed the deep learning models. Therefore, we performed an error analysis on the Resnet model.

| NDCG | Frequency |
|---|---|
| 0.0 – 0.2 | 2254 |
| 0.2 – 0.4 | 1212 |
| 0.4 – 0.6 | 1023 |
| 0.6 – 0.8 | 888 |
| 0.8 – 1.0 | 22326 |

**Table 7.** NDCG distribution on Text ResNet

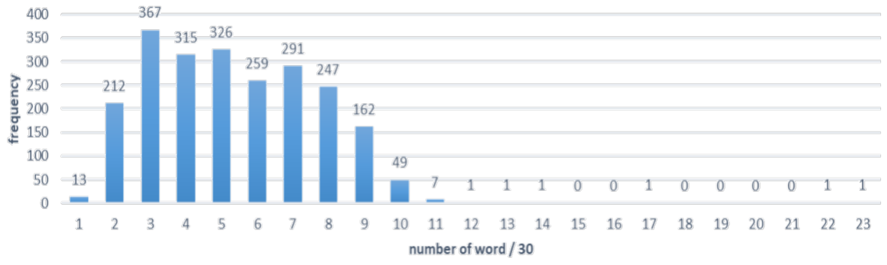


**Fig. 16.** Sentence lengths of data with NDCG between 0 and 0.2

We chose data with the NDCG score between 0 and 0.2. First, we count the sentence lengths. The data with sentence length less than 200 is very large, so the error is not from sentence truncation.

We then analyzed which real hashtags are mispredicted. We plotted the proportion of false negatives and false positives to the true label numbers, shown as Fig.17. We also plotted the true number of hashtags present across all training datasets as the gray line. The horizontal axis is in descending order of the number of hashtags.

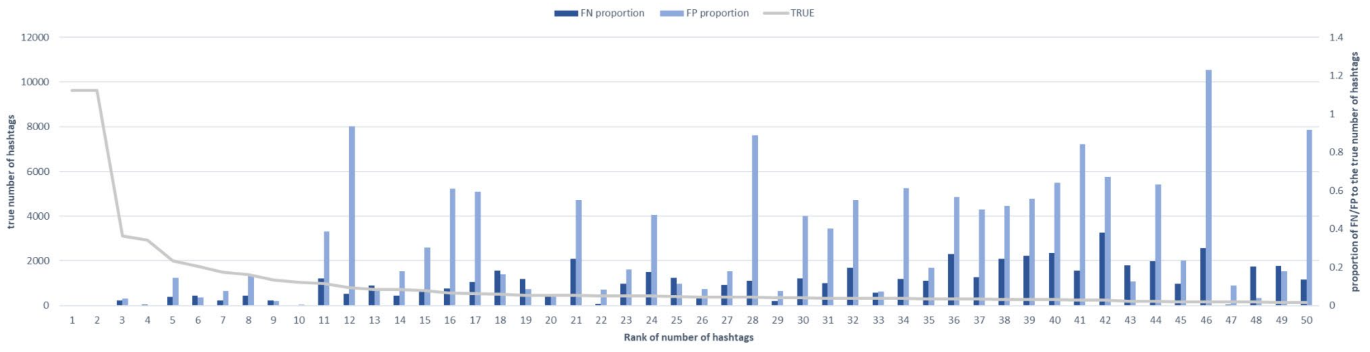


**Fig. 17.** FN/FP of data with NDCG between 0 and 0.2

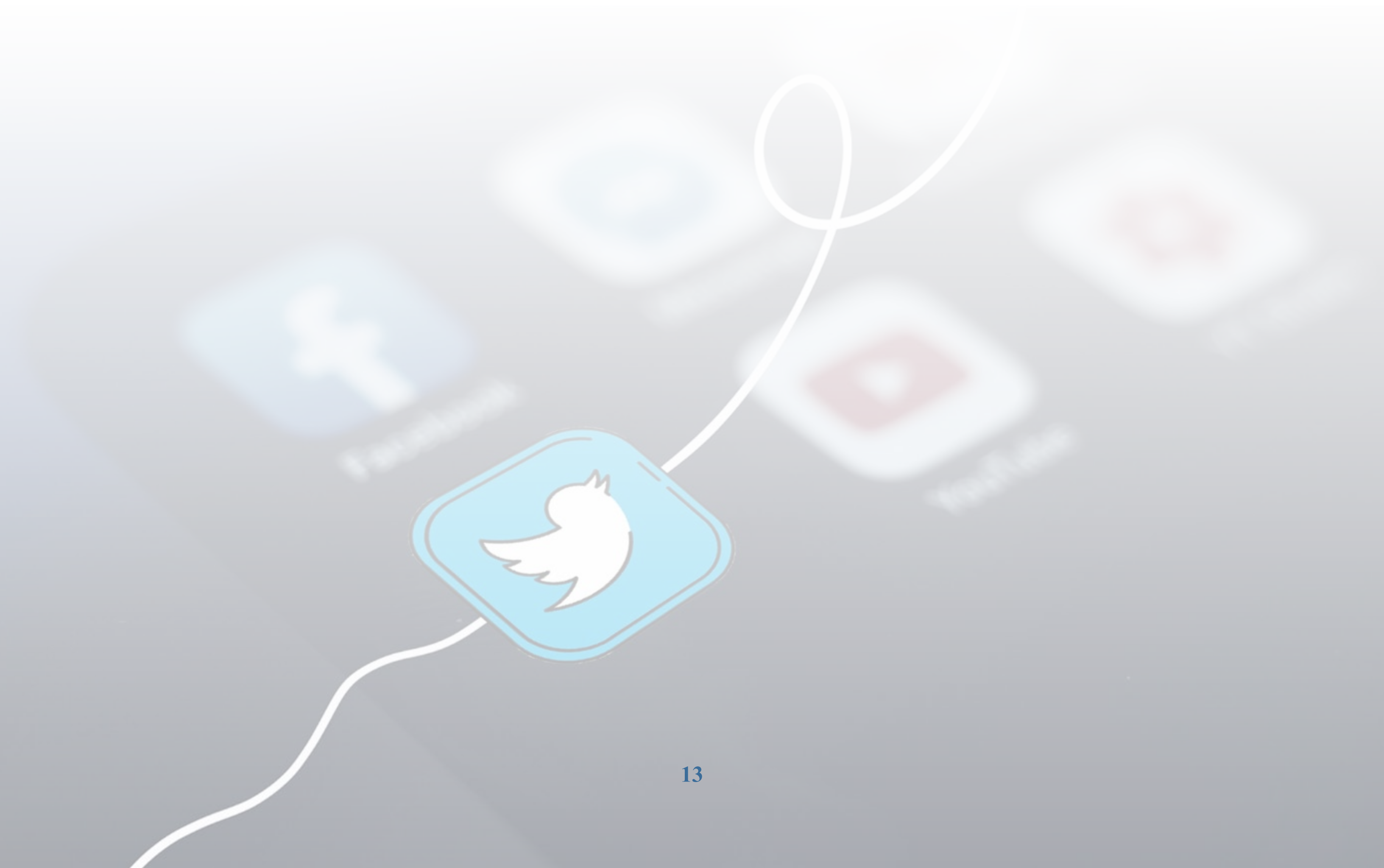
The ratio of FN and FP is low for the first few labels, but high for the rest labels. As the first few hashtags have the largest dataset, while the remaining hashtags have very small dataset. Therefore, we concluded that the error may come from lack of training data, so that the deep learning model performs well on the first few labels, but poorly on the rest.

## 5.2 Conclusion

According to all the work we have done, we can come to the conclusion that **TFIDF + Logistic Regression performed best among all the models.** The reasons can be concluded as below:

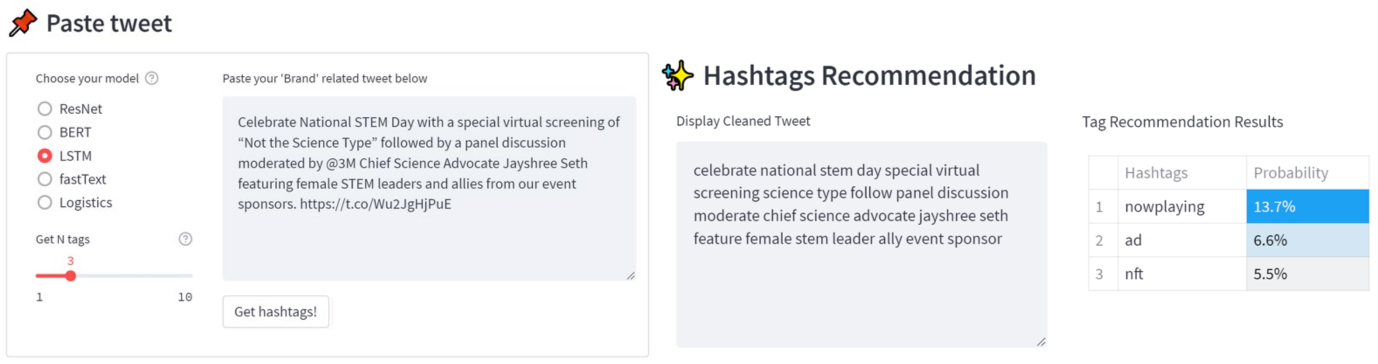
- **The best model often comes from the model and dataset scale which matches best with each other instead of just the largest dataset or the most advanced algorithm.** The size of the dataset we used is still too small to learn for a complex deep learning model. We have only 50 hashtags in our hashtag repository and the training set only includes around 60,000 tweets. But a deep learning model has too many parameters to be tuned. With such a small dataset, models are not able to learn very well.

- **We have limited resources on fine-tuning or training a complex model.** The deep learning models kept making the computer crash during the training. Thus, we had to make the training process simpler and this might lead to insufficient le arning for the models.



## 6. Demo

To better visualize the models and share our ideas, we build a simple demo via Streamlit, which is an open source app framework. As shown in Fig.18., our demo allows interaction and has functions of displaying cleaned tweets and prediction results.


Paste tweet

Choose your model
ResNet
BERT
[✓ LSTM]
fastText
Logistics

Get N tags: 3 (1–10)

Paste your 'Brand' related tweet below

Celebrate National STEM Day with a special virtual screening of "Not the Science Type" followed by a panel discussion moderated by @3M Chief Science Advocate Jayshree Seth featuring female STEM leaders and allies from our event sponsors. https://t.co/Wu2JgHjPuE

Get hashtags!

Hashtags Recommendation

Display Cleaned Tweet

celebrate national stem day special virtual screening science type follow panel discussion moderate chief science advocate jayshree seth feature female stem leader ally event sponsor

Tag Recommendation Results

| | Hashtags | Probability |
|---|---|---|
| 1 | nowplaying | 13.7% |
| 2 | ad | 6.6% |
| 3 | nft | 5.5% |

**Fig. 18.** An Example of Demo Work

# 7. Learnings and Future Work

In this section, we are going to discuss the learnings we got when doing the project, and the possible future work we can do to improve our work.

## 7.1 Learnings

The first learning we have is that multilabel is not the same as multiclass. With multilabel, we need to use binary cross entropy loss function, and use Sigmoid instead of Softmax. Before debugging this problem, our model's performance is very poor.

The second learning is about LSTM. Usually, we need to set a max length for sentence embedding, and if the length is less than the threshold, we need to pad it. LSTM is a sequence model, the padding should be at the beginning of the sentence, otherwise the last output is always close to 0.

The third learning is about dataset size. Some of the hashtags in the repository only have hundreds of data, The small data is not suitable for a large deep learning model with large parameters. Similarly, a big challenge during the model training step is GPU limit. We have to estimate the spaces we need when training and sacrifice some parameter tuning process.

## 7.2 Future Work

Based on the learnings, in the future, we want to collect more data to provide better dataset, upgrade GPU limits to better tune parameters. An exciting thing is that we are building a Streamlit cloud to deploy models and we will add new functions.

# Reference

Joulin, Armand & Grave, Edouard & Bojanowski, Piotr & Mikolov, Tomas. (2016). Bag of Tricks for Efficient Text Classification.

Winda Kurnia Sari, Rini DP, Reza Firsandaya Malik, Iman Saladin B. Azhar. Multilabel Text Classification in News Articles Using Long-Term Memory with Word2Vec. Jurnal RESTI (Rekayasa Sistem dan Teknologi Informasi) (Online). 2020;4(2):276-285.doi:10.29207/resti.v4i2.1655

Zhang, Aston & Lipton, Zachary & Li, Mu & Smola, Alexander. (2021). Dive into Deep Learning.