Toxic Comment Classification Challenge

Project Report

Yangmei(Cathy) Deng
ydeng003@odu.edu
UIN:0112416

Table of Contents

Table of Tables

Project Description

Toxic comments make people stop expressing themselves and give up on seeking different opinions online. To effectively facilitate conversations, platforms need tools to improve online conversation. Google and Conversation AI released the competition "Toxic Comment Classification Challenge"[1] on Kaggle last year, because the tool they were using still made errors.

Different from other data science project. In this problem there are six targets. This project build 3 multi-headed models that's capable of detecting different types of toxicity.

Dataset Overview

There are two csv files used in this project. The train.csv has 160k records, each record consists of a comment retrieved from Wikipedia, and 6 labels which were labeled by human raters for toxic behavior. The types of toxicity are: toxic, severe_toxic, obscene, threat, insult, identity_hate. The toxicity is not evenly spread out across classes. Almost 90% of the comments are clean.

```
In [3]: pd.Series(train['toxic']).value_counts()
Out[3]:
0    144277
1     15294
Name: toxic, dtype: int64

In [4]: pd.Series(train['severe_toxic']).value_counts()
Out[4]:
0    157976
1      1595
Name: severe_toxic, dtype: int64

In [5]: pd.Series(train['obscene']).value_counts()
Out[5]:
0    151122
1      8449
Name: obscene, dtype: int64

In [6]: pd.Series(train['threat']).value_counts()
Out[6]:
0    159093
1       478
Name: threat, dtype: int64

In [7]: pd.Series(train['insult']).value_counts()
Out[7]:
0    151694
1      7877
Name: insult, dtype: int64

In [8]: pd.Series(train['identity_hate']).value_counts()
Out[8]:
0    158166
1      1405
Name: identity_hate, dtype: int64
```

Figure 1 Use spark interactive shell to observe data. All the targets are imbalance.

The test.csv has 153k records. After the competition, Kaggle labeled about 6k of the test records for model evaluation. This project will use labels of these 6k records to evaluate the model.

Executive Summary

This project uses pyspark to preprocess the comments and use three main models for tackling this multi-label problem: Binary Relevance, Classifier Chains and Label Powerset.[2][3]

Design and Implementation

Data preprocess

Design

Design a series mapreduce functions to preprocess and calculate tfidf of each comment.
MapReduce 1: prepare comments, the basic algorithm is below (only include the input and output of the whole mapreduce function):

```
Load train.csv
For each comment:
        Split into tokens
        Convert to lowercase
        Filter out tokens that are not alphabetic
        Filter out tokens that are stop words
Return tokens for each comment
```

MapReduce 2: get Inverse Document Frequency(IDF) for each token, the basic algorithm is below (only include the input and output of the whole mapreduce function):

```
Take output of MapReduce 1 as input
Count number of comments containing each term
Calculate IDF for each term
Return (term, term's idf)
```

MapReduce 3: get Term Frequency(TF) for each term in each document, the basic algorithm is below (only include the input and output of the whole mapreduce function):

```
Take output of MapReduce 1 as input
Calculate TF for each term in each comment
Return ((comment, term), term's TF)
```

MapReduce 4: get TFIDF for each term in each document, the basic algorithm is below (only include the input and output of the whole mapreduce function):

```
Take output of MapReduce 3 as input, output of MapReduce2 as cache
file
Calculate TFIDF for each term in each comment
Return ((comment, term), term's TFIDF)
```

Implementation

When I designed the project, I planned to use 4 passes of map and reduce functions to preprocess data. Each stage of map reduce helped me to understand the problem. But as I know, mapreduce framework needs to store output files of each pass to hard disk or file distributed system. During each stage it also needs to read the output file from the last stage. So I need to read at least 8 times of the data to do the data preprocess. This costs a lot of time. So I change to pyspark.

Before process each comment I need to downsample the train dataset first, because all targets are imbalance. From Fig.1, different labels have different degree of imbalance. For "toxic", about 10% of the comments are labeled as 1. But for "threat", only 3‰ of the comments are labeled as

1. So when I downsample, I kept all comments that have labels of toxicity. And kept half of that number of comments that are clean (labels for 6 targets are all zero). After downsample, the labels for some comments, like “threat”, are still not balance, but it is much better than the original data.

For the test dataset provided by Kaggle, some of the comments are labeled as [-1,-1,-1,-1,-1,-1]. This means they didn’t provide label for these comments. So I need to filter out all these comments from the test dataset first.

When I wrote code I found that after each transformation of RDD, the order of lines might change. So when I process the data, I need to keep target labels and comment of each records together.

I design eight functions to process the data.

Function	Description
getValueTarget(line)	This function will transform each line of RDD to [[list of targets], [whole comment]]
token(comment)	This function tokenize each comment. I filter out all stop words and special characters.
tf(comment)	This function calculate term frequency for each term in the corresponding comment.
tokenList(comment)	This function create a list which include all tokens appears in all comment.
idf(tfList, threshold)	This function create a dictionary which include inverse document frequency(IDF) for each token. Only train dataset is used to create the IDF dictionary. Because in real life we cannot know ahead of time what term will be in the comment that we want to predict.
tfidf(comment, idfDict)	This function use the IDF dictionary to calculate TF-IDF for each term in the corresponding comment.
cmtVector(comment, tokenDict)	These two functions together create a vector for each comment. The vector is a list of frequencies for each unique word in the dataset—the TF-IDF value if the word is in the review, or 0.0 otherwise.
cmtVectorDense(comment, threshold)	

Table 1 Functions for data preprocess

When implemented these eight functions, I mostly use map, filter transformations. Because all transformations in Spark are lazy, they are only computed when an action requires a result. This will make my code run more efficiently.

Model and Evaluation

Design

Use pyspark to do machine learning. Use two main methods for tackling this multi-label problem: problem transformation methods and adaptation methods.

Method 1: transformation methods. This method can be carried out in 3 different ways.

1. Binary Relevance: This method will treat each label as a separate single classification problem. The union of all labels that are predicted will be the final output.
2. Classifier Chains: In this method, each classifier is trained on the input and all the previous classifiers in the chain.
3. Label Powerset: This approach will take all possible combinations of labels into account, which means it will needs worst case $2^6 = 64$ classifiers. This method has a high computational complexity.

Method 2: adapted algorithm

This method will adapt the algorithm to directly perform multi-label prediction, rather than transforming the problem into subproblems.

Implementation

I randomly choose 2000 records from the train dataset, 1000 records from the test dataset as sample data. Run each model on these sample data to see which model is the best.
For each model I use three different thresholds 1000, 3000, and 5000. The threshold means how many tokens are used to create the IDF dictionary.
I use accuracy, precision, recall, time spent to evaluate each model. I also have the confusion matrix available in the output log file for each target of each model.

Method 1: This method transform this problem to six subproblems. Each time we only predict one target.

Method 1- Strategy 1: Binary Relevance

In this case, I build a logistic regression model for each target label. The main assumption here is that the labels are mutually exclusive. It ignores the possible correlations between class labels.

Threshold	Target	Accuracy		Precision		Recall		Time
		Each Target	Average	Each Target	Average	Each Target	Average	
1000	toxic	0.326	0.716	0.101	0.038	0.889	0.515	50.35s
	severe_toxic	0.916		0		0		
	obscene	0.558		0.062		0.632		
	threat	0.991		0		1		
	insult	0.578		0.066		0.571		
	identity_hate	0.925		0		0		
3000	toxic	0.304	0.848	0.101	0.091	0.917	0.428	80.76s
	severe_toxic	0.981		0		0		
	obscene	0.930		0.28		0.368		
	threat	0.998		0		1		
	insult	0.895		0.167		0.286		
	identity_hate	0.979		0		0		
5000	toxic	0.478	0.877	0.13	0.108	0.917	0.495	113.87s
	severe_toxic	0.988		0		0		
	obscene	0.913		0.275		0.579		
	threat	0.998		0		1		
	insult	0.902		0.244		0.477		
	identity_hate	0.983		0		0		

Table 2 Evaluation Statistics for Method 1 - Strategy 1

Table 2 is the result for this strategy. From the result we can see, when the threshold increases, the accuracy , precision and time will increase, too. But the recall doesn't not have a lot of differenc.

Method 1- Strategy 2: Classifier Chains

In this strategy, I construct a chain of binary classifiers which means each classifier is depend on all the previous classifiers. For example, for target "toxic", I use all the token's TF-IDFs as the trainX. For target "severe_toxic", I combine column "toxic" together with token's TF-IDFs together to build the second binary classifier. For target "obscene", I use "toxic", "severe_toxic" and token's TF-IDFs to build the third binary classifier, etc.

When I predict, I also combine the prediction of all previous targets together with token's TF-IDFs.

Threshold	Target	Accuracy		Precision		Recall		Time
		Each Target	Average	Each Target	Average	Each Target	Average	
1000	toxic	0.326	0.716	0.101	0.038	0.889	0.515	42.24s
	severe_toxic	0.918		0		0		
	obscene	0.550		0.061		0.631		
	threat	0.988		0		1		
	insult	0.573		0.065		0.571		
	identity_hate	0.925		0		0		
3000	toxic	0.304	0.85	0.101	0.427	0.917	0.428	74s
	severe_toxic	0.981		0		0		
	obscene	0.932		0.292		0.368		
	threat	1		1		1		
	insult	0.897		0.171		0.286		
	identity_hate	0.986		1		0		
5000	toxic	0.477	0.870	0.13	0.269	0.917	0.545	112.87s
	severe_toxic	0.988		0		0		
	obscene	0.881		0.224		0.684		
	threat	0.998		0		1		
	insult	0.890		0.259		0.667		
	identity_hate	0.986		1		0		

Table 3 Evaluation Statistics for Method 1 – Strategy 2

Table 3 is the result for this strategy. Similar to strategy 1, when the threshold increases, the accuracy and time will increase, too. Recall doesn't have much a lot difference. But one thing very interesting is the precision for threshold 3000 is better than it is for threshold 5000. So a proper threshold is very important to this model. It is not the case that bigger threshold will end up with a better result.

Method 1- Strategy 3: Label Powerset

This approach considers each member of the power set of labels in the training set as a single label, which means each combination of the labels will be treated as a single label. So it will create $2^6 = 64$ classifiers. This method has a high complexity because when the number of classes increases the number of distinct label combinations can grow exponentially.

Threshold	Target	Accuracy		Precision		Recall		Time
		Each Target	Average	Each Target	Average	Each Target	Average	
1000	toxic	0.349	0.726	0.107	0.041	0.917	0.495	417.2s
	severe_toxic	0.934		0		0		
	obscene	0.571		0.068		0.579		
	threat	0.995		0		1		
	insult	0.567		0.068		0.477		
	identity_hate	0.094		0		0		
3000	toxic	0.368	0.858	0.102	0.414	0.833	0.434	722s
	severe_toxic	0.981		0		0		
	obscene	0.913		0.219		0.368		
	threat	1		1		1		
	insult	0.902		0.161		0.238		
	identity_hate	0.988		1		0.167		
5000	toxic	0.74	0.907	0.179	0.164	0.583	0.545	1025.71s
	severe_toxic	0.974		0.11		0.25		
	obscene	0.881		0.204		0.589		
	threat	0.993		0.208		0.524		
	insult	0.878		0.259		0.667		
	identity_hate	0.989		0.286		0.33		

Table 4 Evaluation Statistics for Method 1 – Strategy 3

Table 4 is the result for this approach. Similar to strategy 2, when the threshold increases, the accuracy and time will increase, too. But the precision for threshold 3000 is better than threshold 5000. Again, we need to find a proper threshold when we build models.

Method 2: This method directly perform multi-label prediction, rather than transforming the problem into subproblems. Since we cannot use any package in this project. I don't quite understand the math behind this method. So I couldn't implement it.

Discussion of Results

Strategy	Target	Accuracy		Precision		Recall		Time
		Each Target	Average	Each Target	Average	Each Target	Average	
Strategy1	toxic	0.478	0.877	0.13	0.108	0.917	0.495	113.87s
	severe_toxic	0.988		0		0		
	obscene	0.913		0.275		0.684		
	threat	0.998		0		1		
	insult	0.902		0.244		0.619		
	identity_hate	0.983		0		0		
Strategy2	toxic	0.477	0.870	0.13	0.269	0.917	0.545	112.87s
	severe_toxic	0.988		0		0		
	obscene	0.881		0.224		0.684		
	threat	0.998		0		1		
	insult	0.890		0.259		0.667		
	identity_hate	0.986		1		0		
Strategy3	toxic	0.74	0.907	0.179	0.164	0.583	0.545	1025.71s
	severe_toxic	0.974		0.11		0.25		
	obscene	0.881		0.204		0.589		
	threat	0.993		0.208		0.524		
	insult	0.878		0.259		0.667		
	identity_hate	0.989		0.286		0.33		

Table 5 Compare 3 strategies with threshold of 5000

Since when the threshold is 5000, we have the best accuracy and recall for all 3 strategies. So I compare these three models with the statistics of threshold of 5000. From table 5 we can see, we

have the best precision, recall and time efficiency for strategy2. It means the six toxicity labels are related. Even though the accuracy of strategy 2 is a little lower than the other two strategies. But the goal of this project is to build a model that's capable of detecting different types of toxicity. Since for each toxicity label our data is very imbalance, high accuracy doesn't mean a good model. So Classifier Chains would be a good model for this project.

Scalability Study

To test the scalability of each model, I test it on the whole dataset, half of the dataset, one fourth of the dataset when threshold is 3000. Following is the table shows how many time spent for each model on different size of dataset. The whole dataset has 160K records in train, 153K records in test.

Dataset Size	Strategy	Time
Whole Dataset	Strategy 1	2hr 40mins
	Strategy 2	3hr 22mins
	Strategy 3	>20hrs
Half Dataset	Strategy 1	1hr 54mins
	Strategy 2	1hr 20mins
	Strategy 3	15hr 18mins
¼ Dataset	Strategy 1	50 mins
	Strategy 2	41 mins
	Strategy 3	6hrs 22mins

Table 6 Model scalability

Table of Source Code Files

File Name	Purpose
tfidf.py	Preprocess data, get TF-IDF for tokens of each comment
confusionMatrix.py	Evaluate model, get confusion matrix, accuracy, precision, recall of each model.
LogisticRegression.py	Logistic Regression. It includes function for single prediction and multi-prediction.
Strategy1.py	Implementation for the first strategy: Binary Revelance
Strategy2.py	Implementation for the second strategy: Classifier Chains
Strategy3.py	Implementation for the third strategy: Label Powerset
tfidf.sh	Start the application, run tfidf.py.
s1.sh	Run Strategy1.py to get result of first model
s2.sh	Run Strategy2.py to get result of second model
s3.sh	Run Strategy3.py to get result of third model

Table 7 Purpose of source code files

Future Enhancements

This project focuses on comparing results for different threshold and different strategies. Due to time constraints, there are some potential techniques can be used to process large dataset. Some of these enhancements are listed below.

- **Use PCA to do feature selection**: For a high threshold, each comment will have a large number of features. This will affect the efficiency of our algorithm. Unfortunately I didn't have time to implement PCA or other feature selection method after data preprocess.

- **Predict different targets at the same time**: For strategy 1, different targets are considered as mutually exclusive. So if the model can predict the targets simultaneously instead of sequentially. The model will be much faster. It is the same for strategy 3. After create $2^6 = 64$ labels, we can consider all labels are mutually exclusive.

Reference

[1] Toxic Comment Classification Challenge
https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data

[2] Deep dive into multi-label classification..! (With detailed Case Study)
https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

[3] Solving Multi-Label Classification problems (Case studies included)
https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/