Introduction to Data Mining Project Report

Cpt\_S 315

Washington State University

Toxic Comment Classification

Created by

Ben Kaufmann

Daniel Semenko

April 2022

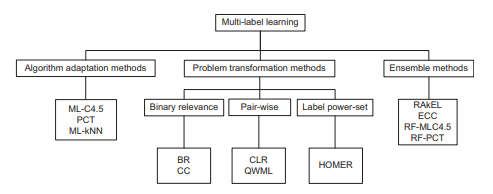
**1.0 Introduction**

DISCLAIMER FROM THE DATA SOURCE: The dataset contains text that may be considered profane, vulgar, or offensive.

**1.1 Motivations and Questions**

Discussing topics in any sort of online format can be challenging, for many different reasons. The threat of conversational toxicity is an issue that can lead people to no longer feel comfortable, out of the fear of aggravation and abuse online. This is a common problem in online video games and forums, where the ability to “hide behind your screen” allows malicious users to leave these comments, without risk of repercussion. The goal of this project is to use Machine Learning algorithms to train a model to identify toxic behavior in comments. This could be used to discourage those hostile users from posting potentially toxic or hurtful messages, allowing positive, civil discourse to occur.

The goal of this project is to implement a few different algorithms – specifically – to find the best approach to classifying toxic comments. Through the project, the authors will evaluate and analyze the effectiveness of two different algorithms. These different algorithms stemmed from the type of problem presented in this project – the comments having multiple “labels”. Also known as Multi-Label Classification, it allows the authors to attempt many different methods (see Fig. 1), as Multi-Label Classification has many different approaches to solve it. These can include Binary Classification, Multi-Class Classification, Ensemble methods, in addition to classic algorithms being adjusted to the multi-label task – including kNN, Decision Trees, Neural Networks, and more (Madjarov, Kocev, Gjorgjevikj, & Džeroski, 2012). These were the starting point of many of our questions for this project. What is the best approach to classifying toxic behavior? Are there correlations between labels, and why do those correlations exist? How would the authors’ approaches change if just classifying as toxic vs non-toxic – single-label classification?



**Fig. 1.** Multi-label learning methods

The decision to choose this topic, and therefore this project, stems from both authors’ familiarity with toxic comments and hostile online environments. Both play video games and engage in online forums, where these types of hostile interactions are all too common. In some situations, these online platforms struggle to facilitate online conversations, leading to the removal of user comments and text chats. The authors hope that with the implementation of algorithms such as the one the authors explore, their usage can lead to improvements in online conversation with more productive and respectful discussion.

**1.2 The Task**

The authors ran into many challenges throughout the duration of the project, the first coming on just the first day! This project takes data from the *Jigsaw Toxic Comment Classification Challenge*, hosted by Kaggle, in which the data provided is a dataset of comments from Wikipedia’s talk page edits. The very first problem was the formatting of the provided csv files, as during the download, the delimiting (occurring on commas) didn’t work as intended. Therefore, the authors had to work with the format more to figure out how best to fix it. Another problem was the inclusion of apostrophes in cleaned comments, as it would conflict with stop words.

The main aspect of the project revolved around testing different multi-label classification methods against each other. To get there, there were a few steps before. First, the authors determined the definition of the problem, and how they would go about solving and evaluating them. The problem being the challenge found [here](https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge) on Kaggle – build a model capable of detecting different types of toxicity, which predicts the probability of each type of toxicity for each comment, trained using a highly-imbalanced training data set. The authors could then evaluate the performances of the different methods by comparing the accuracies of each, and the CPU computation time. The next step was to import and clean up the data – the preprocessing step. Here, the authors make the data as nice and clean as possible to make the next steps easier. At this point, the data is prepared for the different multi-label classification methods. Once those methods have been successfully implemented and run, the authors can evaluate the effectiveness of the algorithms, by comparing their accuracies and compute times.

When it comes to multi-label classification, there are many possible approaches to solving them. These can include Problem Transformation Methods, Adapted Algorithms, Learning Paradigms, Multi-Label Stream classification, and Statistics and evaluation methods, among others (Wikipedia, 2022). Out of those, the author’s decided to use two problem transformation methods and adapted algorithms. The authors will use the Binary Relevance method, and the label powerset transformation.

**1.3 Brief Result Overview**

The authors found that the Binary Relevance algorithm had an accuracy of 0.8586 and a computation time of ~36936ms. For the Label Powerset method, they found that it had an accuracy of 0.8926 and a computation time of ~83466ms.

**2.0 Data Mining Task**

**2.1 Task Details**

Participants of the Toxic Comment Classification challenge, the authors included, are provided with many Wikipedia comments which have been labeled by human raters for toxic behavior. The types of toxicity in which the authors must classify comments are:

* toxic
* severe\_toxic
* obscene
* threat
* insult
* identity\_hate

The data provided for the Kaggle competition are given to participants in a couple of different files.

* train.csv – the training set, containing comments with their binary labels
* test.csv – the test set, of which the authors predict toxicity probabilities of comments

**2.2 Authors’ Questions**

As stated before, the authors set out to answer a multitude of questions, revolving around the effectiveness of different multi-label classification algorithms. Of the two methods, which is the most accurate? That is, which algorithmic approach trains the model the best? Which of the two methods takes the shortest amount of time? Which is the longest? Why does this happen? In what cases would it make sense to use one method over another?

**2.3 Key Challenges**

One of the first key challenges the authors faced with this project was the downloading of data, pulled from the Kaggle challenge. As a .csv file, the provided training and testing data are split by the comma delimiter. Normally, this isn’t a problem with different types of data, but since the data contains comments as one of the columns, the delimiters messed up sometimes. This would include it messing up on the newline characters within comments, or commas within comments also bringing up issues. While a rather large problem up front, it ended up being fixed rather easily. The authors just saved the .csv files into a different .csv format, fixing the delimiting, and allowing for the code to correctly input the data.

Another challenge was during the pre-processing step of our algorithm, specifically removal of stop words within the comments. Since the algorithm used the NLTK stop words dataset, the authors also initially opted to use the *word\_tokenize()* function, also part of the NLTK library framework. But this ran into issues where it would tokenize not only on spaces (as it should), but also on apostrophes. This resulted in contraction words being split and stored as different words. For example, the word *I’m* would then be stored as *I’* and *m*, for some strange reason. The authors decided instead to create their own *tokenize()* function, to remedy the problem. By doing so, the code correctly removed stop words without splitting incorrectly.

The largest problem the authors came across was when they changed from hard coding the algorithms, to implementing existing functions found in libraries such as sklearn. While writing the code was easier, the correct implementation was difficult, as the authors weren’t accustomed to matching the multiple different types of data frames, classification metrics and targets, and many other variables.

Finally, the last problem the authors ran into was running the code. The algorithms ran correctly, but the huge amount of data could not be run by either of the authors’ computers (see Fig. 2). The amount of memory required for compiling the project far exceeded the memory on either device. Therefore, they had to size down the data, from 1276568 comments to 5000 comments.

**Text

Description automatically generated**

**Fig. 2.** Large Data Error

**3.0 Technical Approach**

**3.1 Algorithm**

3.1.1 Exploratory Data Analysis

The focus of this step is to learn about our data, figuring out what questions to ask, how the data might have to be manipulated, what the answers tell the authors, among others.

To get there, the authors first had to load the data, using a pandas data frame, to look at all the aspects of information about the data. To get actual useful information, the authors first had to collect basic information on the input data, including information such as the number of rows and columns in the data, and the labels of each column. From here, the authors could look at more important information, such as the number of comments in each toxicity category, the ratio of clean comments to comments with labels, and comments with multiple labels (see Fig. 3).

A screenshot of a computer

Description automatically generated with medium confidence

Chart, bar chart

Description automatically generated Chart

Description automatically generated

**Fig. 3.** Exploratory Data Analysis

3.1.2 Data Pre-Processing

While this step isn’t a difficult or interesting step, it is very important. For the models to be trained well, the data must be taken from the raw data (in which comments may include special characters, HTML tags, punctuation, etc.) into TF\_IDF vectors with no stop words and stemmed phrases.

First, the data must be cleaned. In this step, the authors clean comments one-by-one, by setting each comment to lower case and removing non-alphabetic/non-numeric characters. They do so by creating a cleanComment() function, which can be applied to the pandas data frame.

Next, the data gets stop words removed. Stop words are any words in our stop list which are filtered out before processing the data, to save space and time in processing this large data set. The authors had done a similar step in a previous homework assignment by hand but decided to use an existing library containing common English stop words. This library is the Natural Language Toolkit for python, which the authors imported for this step. The authors referenced helpful steps to do this [here](https://www.geeksforgeeks.org/removing-stop-words-nltk-python/) (GeeksforGeeks, 2021). Then, the authors created a removeStopWords() function which, using NLTK functions, was then applied to the pandas data frame.

The next step is stemming. “Stemming is the process of producing morphological variants of a root/base word. Stemming programs are commonly referred to as stemming algorithms or stemmers. A stemming algorithm reduces the words ‘chocolates’, ‘chocolatey’, ‘choco’ to the root word, ‘chocolate’ and ‘retrieval’, ‘retrieved’, ‘retrieves’ reduce to the stem ‘retrieve’” (GeeksforGeeks, 2021). Again, the authors referenced helpful steps found [here](https://www.geeksforgeeks.org/python-stemming-words-with-nltk/) for using NLTK again, to stem. Like before, the authors created a stemComment() function which, using NLTK functions, could then be applied to the data.

Finally, the last step of pre-processing is creating TF-IDF vectors. This is done using the sklearn python library, and the TfidfVectorizer function found within the feature\_extraction.text sub library of sklearn. The authors want to calculate TF-IDF scores for every word in every comment, to show which words are significant to each other. Doing so allows the creation of TF-IDF vectors, which will be used in our Multi-Label Classification methods.

3.1.3 Multi-Label Classification

At this point, the data has been pre-processed, and is ready to be evaluated by the two methods the authors chose to analyze.

The first method is Binary Relevance. This method ended up being the simplest, by treating each label as a separate single classification problem. The key assumption being that there is no correlation among different labels. Clearly, this is an oversight in the algorithm (that the authors are aware of), as some comment labels often correlated with others (See Fig. 3). When researching for this project, the authors found sklearn libraries that suited their needs perfectly – in particular the sklearn libraries. For Binary Relevance, the authors employed the use of the *BinaryRelevance* functions and *GaussianNB* (Gaussian Naïve Bayes) functions.

Next was Label Powerset Transformation. This transformation method would change a multi-label problem into a multi-class problem with 1 multi-class classifier trained on the label combinations found in the training data (Szymański & Kajdanowicz, 2017). Unlike with Binary Relevance, this method does consider possible correlations between different labels. The problem with this method is the computational complexity, as when the number of classes increases, the number of distinct label combinations grows exponentially. The authors again employed existing functions from the sklearn libraries. They used the *LogisticRegression* functions, and the *LabelPowerset* functions.

3.1.4 Evaluation

Finally, the last step of the code is to evaluate the different methods. This is simply done by comparing the accuracies and the computational times of the methods.

**4.0 Evaluation Methodology**

**4.1 Input Data**

As the input data comes from a challenge provided by Jigsaw, and organized on Kaggle, the input data is clean and easy to get started with. Participants of the challenge are provided with many Wikipedia comments which have been labeled by human raters for toxic behavior. The training data has eight columns, including the comment ID, comment text, and the six toxicity labels (see Fig. 4).

Graphical user interface, text, application

Description automatically generated

**Fig. 4.** Five random rows of the training data

The testing data only has two columns, the comment ID and the comment text (see Fig. 5).

Graphical user interface, text, application, email

Description automatically generated

**Fig. 5.** Five random rows of the testing data

**4.2 Evaluation Metrics**

There are two metrics that we are evaluating, when comparing the effectiveness of the two algorithms we employ. The first metric is accuracy, in which the actual effectiveness of the model is evaluated. How effective is the model at making correct predictions on the classification of toxic comments? The other metric we evaluate is computational time, in which we evaluate how long the model takes to classify the testing data. Since we use the same data for each method, we can evaluate the time to evaluate for each. Is the implementation of a specific model even feasible, based on the computational time or processing power required to evaluate it? This may not be as big a deal on (relatively) smaller sets of data, like in this project, but as data sizes increase from hundreds of thousands to millions and hundreds of millions, the downfalls of an algorithm with a poor computational complexity will show. It is important to choose a method that suits an implementation’s needs. If a method has a smaller set of data, then it might be smarter to choose the most accurate method, regardless of the computational complexity. On the other hand, if a program has a huge set of data, it may be worth the trade off to have a lower accuracy to compute across the huge amount of data.

**5.0 Results and Discussion**

**5.1 Results**

After finishing the code and running our program (on 5000 comments), we found that the Binary Relevance method had a compilation time of 36936ms and an accuracy of 0.8586. And the Label Powerset method had a compilation time of 83466ms and an accuracy of 0.0.8926 (See Fig. 6).

Text

Description automatically generated

**Fig. 6.** Binary Relevance and Label Powerset results

Therefore, the authors concluded that the Label Powerset has a greater accuracy, but a longer computation time, while the Binary Relevance has a (slightly) lower accuracy, but a shorter computation time. When it comes to deciding which algorithm is best, the authors decided that there is no clear answer. Sure, the Label Powerset method clearly showed a greater testing accuracy, yet the time it took to compile was more than two times longer. So, the overall “better” method of the two depends on the size of the data.

One part of this project that the authors found most difficult was during the implementation of the two algorithms. Switching from hard coding to the usage of functions from existing libraries had positives and negatives for the authors. It was good for them because it made writing the code easier, but not implementing the code. This is because matching the multiple different types of data frames, classification metrics and targets, and many other variables that the authors aren’t fully accustomed to, was difficult.

**6.0 Lessons Learned**

One important lesson the authors learned is the classic line: “If it ain’t broke, don’t fix it.” This is especially the case for the implementation of the two algorithms of this project. The authors initially attempted to hard code the algorithms, based on the ideas of the algorithms, but had a hard time implementing it with the pandas data frames. But after doing more research and finding out about the existing libraries, they opted to instead use those. It made coding the algorithms easier, execution faster, and overall increased the strength of our project.

Another important lesson the authors learned, was the importance of understanding why certain algorithms may be used over others. For example, a model with a lower accuracy but faster computation time may be used over a more accurate method, due to the size of the data being used with the model. Therefore, the best approach of a data problem will always be different, based on what you are looking for.

**7.0 Acknowledgements**

**7.1 Bibliography**

GeeksforGeeks. (2021, May 31). *Removing stop words with NLTK in Python*. Retrieved from GeeksForGeeks: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

GeeksforGeeks. (2021, July 5). *Stemming words with NLTK in Python*. Retrieved from GeeksforGeeks: https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

Madjarov, G., Kocev, D., Gjorgjevikj, D., & Džeroski, S. (2012). An extensive experimental comparison of methods for multi-label learning. *Pattern Recognition*.

Nooney, K. (2018, June 7). *Deep dive into multi-label classification..!* Retrieved from Towards Data Science: https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

Raj, R. (2020, January 1). Machine Learning | Multi Label Classification. YouTube. Retrieved from https://www.youtube.com/watch?v=265-t5HxOR4

Szymański, P., & Kajdanowicz, T. (2017, February 5). *Label Powerset*. Retrieved from Scikit-multilearn: http://scikit.ml/api/skmultilearn.problem\_transform.lp.html

Wikipedia. (2022, February 5). *Multi-label classification*. Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Multi-label\_classification

Zhang, M.-L., & Zhou, Z.-H. (2007). ML-KNN: A lazy learning approach to multi-label learning. *Pattern Recognition*.

Zhang, M.-L., Li, Y.-K., Liu, X.-Y., & Geng, X. (2018). Binary relevance for multi-label learning: An Overview. *Frontiers of Computer Science, 12*(2).

**7.2 Appendix 1 Python Code Libraries/Frameworks**

pandas – Python Data Analysis Library. <https://pandas.pydata.org/>

numpy – Python Scientific Computing Library. <https://numpy.org/>

re – Python Regular Expression Operations. <https://docs.python.org/3/library/re.html>

NLTK – Python Natural Language Toolkit. <https://www.nltk.org/>

Sklearn – Software Machine Learning Library: <https://scikit-learn.org/>

Sk-multilearn – Multi-Label Classification Library: <http://scikit.ml/>