

Final project

Toxic Comment Classification Challenge



Members

Mark, Chadorchi

Kevin, Delao

Kevin, Ngo

Tracy, Lan

Table of Contents

[1. Project details 4](#_Toc25934228)

[1.1 Description 4](#_Toc25934229)

[1.2 Requirements 4](#_Toc25934230)

[1.3 Data description 4](#_Toc25934231)

[1.4 Approaches’ Analyzations 5](#_Toc25934232)

[1.4.1 Natural Language Processing (NLP). 5](#_Toc25934233)

[1.4.1.1 One-hot encoding 5](#_Toc25934234)

[1.4.1.2 TF-IDF 9](#_Toc25934238)

[1.4.1.3 Word embedding 12](#_Toc25934241)

[1.4.1.4 Word2Vec 13](#_Toc25934242)

[1.4.2 K-Nearest Neighbors (KNN) Classification. 22](#_Toc25934246)

[1.4.3 Decision Tree Classification. 26](#_Toc25934253)

[1.4.4 Random Forest Classification 29](#_Toc25934259)

[1.4.5 Logistic Regression Classification. 32](#_Toc25934265)

[2. AUC AND ROC RESULTS 37](#_Toc25934270)

[3. Source code 38](#_Toc25934271)

[3.1 Data Files 38](#_Toc25934272)

[3.2 Testing process (includes all files relate to analyzation process) 38](#_Toc25934273)

[3.3 One Hot Encoding 38](#_Toc25934274)

[3.4 TF-IDF 38](#_Toc25934275)

[3.5 Word2Vec 38](#_Toc25934276)

[3.6 Decision Tree Classification 38](#_Toc25934277)

[3.7 Logistic Regression Classification 39](#_Toc25934278)

[3.8 Random Forest Classification 39](#_Toc25934279)

[3.9 MultinomialNB with CountVectorizer 39](#_Toc25934280)

[3.10 AUC and ROC results 39](#_Toc25934281)

[4. Responsibility of each member (Weekly reports) 40](#_Toc25934282)

[4.1 Week 1 – 10/18/2019 40](#_Toc25934283)

[4.2 Week 2 – 10/25/2019 40](#_Toc25934284)

[4.3 Week 3 – 11/01/2019 41](#_Toc25934285)

[4.4 Week 4 – 11/08/2019 41](#_Toc25934286)

[4.5 Week 5 – 11/15/2019 41](#_Toc25934287)

[4.6 Week 6 – 11/22/2019 42](#_Toc25934288)

[4.7 Week 7 – 11/29/2019 42](#_Toc25934289)

# Project details

## Description

During the conversations, it is normal that people have different point of views. Some people are open-minded and comfortable to discuss something they might agree or disagree about. However, the conversation might become tough situation when people are out of control. For instance, people are making threats more and more nowadays, especially during online conversations. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions.

As a result, study of negative online behaviors, like toxic comments (i.e. comments that are rude, disrespectful, or otherwise likely to make someone leave a discussion), are highly demands in the current time.

This project intent focuses on studying how to predict or detect toxic comments to help improve online conversations.

## Requirements

* Final reports: demonstrates the information and process of the project, including project details, requirements, source code, method, algorithms, etc.
* Evaluation: The project should be able to predict a probability for each of the six possible types of comment toxicity (toxic, severe toxic, obscene, threat, insult, identity hate).
* Deadline: December 2019.

## Data description

* train.csv - the training set, contains comments with their binary labels.
* test.csv - the test set, you must predict the toxicity probabilities for these comments. To deter hand labeling, the test set contains some comments which are not included in scoring.
* sample\_submission.csv - a sample submission file in the correct format.
* test\_labels.csv - labels for the test data; value of -1 indicates it was not used for scoring; (Note: file added after competition close!)

## Approaches’ Analyzations

### Natural Language Processing (NLP).

Since machines cannot understand words like human do, we need to “teach” the machine about knowing our natural language (which is English in our project). Before using any algorithm which we have learned from the classes to train our data, we need to do something to help the machine “know English words”. There are several ways to do, in this project, we are going to use two different ways to encode the words into meaningful numeric values.

### One-hot encoding

One hot encoding is an idea that will replace all category features to more numeric features which will only contain 0s and 1s.

In our project which needs to deal with words, we still use the same idea but also add more requirements. Although we still convert words appear in “comment\_text” column to more feature columns, we do not really need every word appears in the dataset. It is possible that certain words only appear in a few data rows.

Hence, after calculating the frequencies of every word, we only take the word that appears over some threshold value. After that, we will convert these certain words to new features which only contain 0s and 1s.

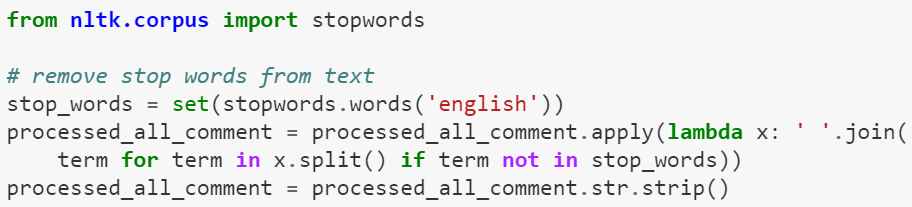
### Why is ONE-HOT ENCODING used in Machine Learning?

In many situations, non-numerical data must be converted into a more quantifiable form for machine learning. One hot encoding, as its name suggests, assigns a ‘one’ to a field for which an identity is true and ‘zeros’ for all the other fields. In this way all possible combinations can be considered. It should be noted that although there may be many valid combinations, not all of them will be logically sensible.

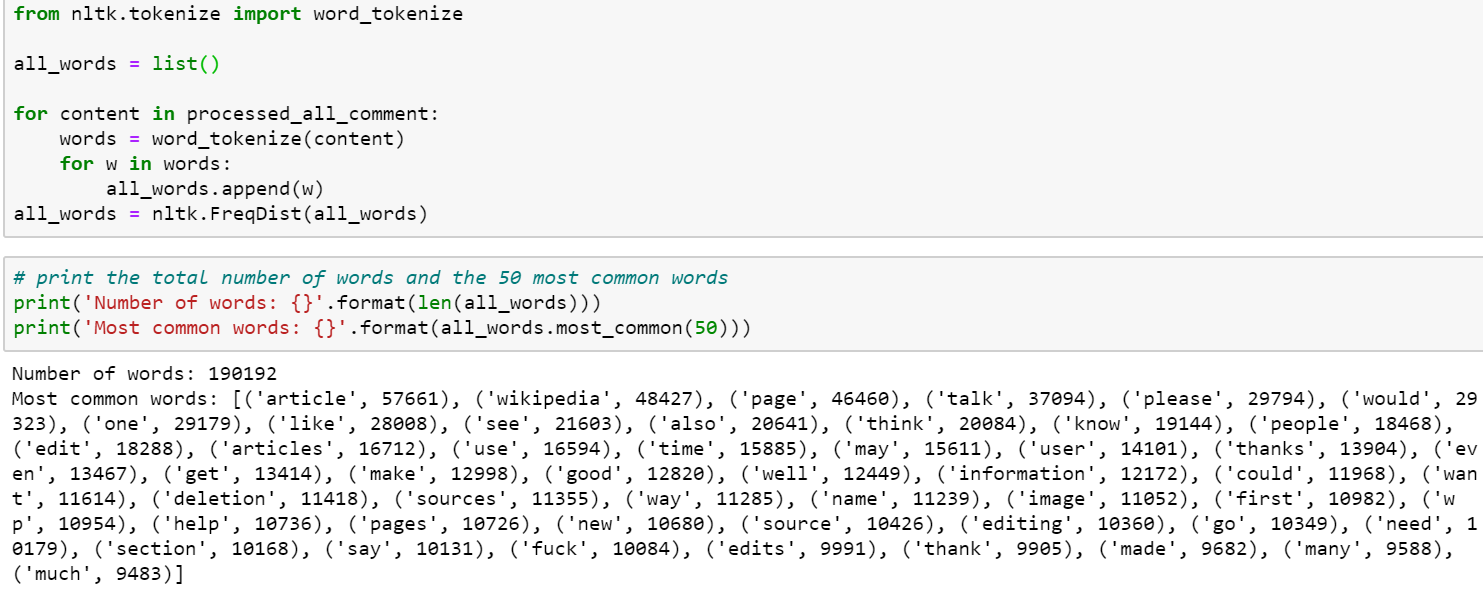
### How to apply one-hot encoding to help predict toxic comment?

Step 1: First remove every special symbol and convert every word into lower cases using python **str.replace()**

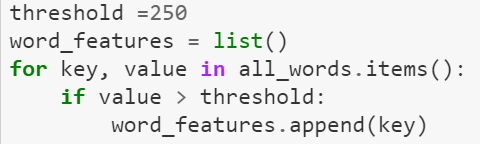
Step 2: Use the **stopword** from **nltk.corpus** to filter out some common stop words which is meaningless for us to recognize whether the word is toxic or not.



Step 3: We use **word\_tokenize** from **nltk.tokenize** to split every remaining word in the dataset and store them into a list. Then we can put the list into **nltk.FreqDist** to let the method to calculate the words and its appearance. Like the result shown below.



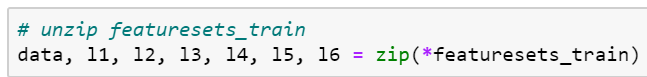
Step 4: We set a threshold and only add the word whose frequency is over that threshold into another list named **word\_features**.



Step 5: Here is the step where we do the **one-hot encoding**. We will combine “comment\_text” and six label columns into a list and send this as a parameter to the method **find\_features\_dict** we created. What this method do is that it will use the **word\_features** list which we got from Step 4 and check if this row contains certain word in the list. If do, we will set the feature (word) column value as 1; if not, value will be 0.



Step 6: After this step, we are done finished setting our new data. We can unzip the **featuresets** we created in step 5 like below.



Step 7: Then assign each label result as a list and store them all in a **y\_train** array. (e.g., **y\_train[**0**]** contains the result of ‘toxic’ label); and convert the result that we got from previous steps to list. Now we can just do the same thing we did on our lab or homework to train and predict the labels.





### Challenges

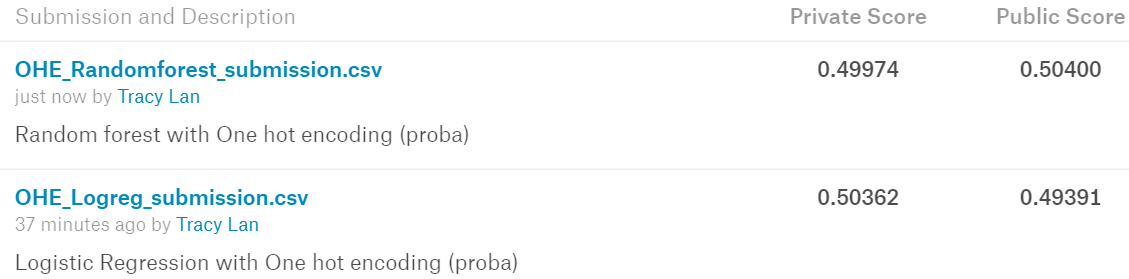
* 1. Time consuming

It is in fact very time consuming when we apply one hot encoding on our dataset to process the words. What makes this even worse is that once the program is dead at some point, we need to restart the whole program and need to wait for the data processing again and again. (Unfortunately, we keep having this situation at first. Because I was keep trying one hot encoding on KNN, and this took forever for me to get the result.) The solution to this problem is that we can use something called pickle which is provided by Python. Pickle help us to store the data once we finish processing the dataset and saved us lots of time. Screenshot below is how we write the data into pickle and how we read the data from pickle.



* 1. Cannot apply on KNN and Decision Tree and the accuracy is bad

When using One Hot Encoding to deal with the data, it will create thousands of feature columns. Hence, it will take us hours if we wish to use either KNN or Decision Tree to help us predict the labels. We can still use Logistic Regression and Random Forest, which will take a bit longer than Logistic Regression, to train the model to predict. The accuracy of these two predictions however are both barely over 50%.



### TF-IDF

TF-IDF, is short for term frequency–inverse document frequency, is a numerical statistic that is intended to reflect how important a word is to a document in a collection or corpus. Like the name it calls, we have 2 variables in this method, term frequency and Inverse document frequency.

* Term frequency is the number of the certain word appears in a document, which will be a row of data in our project.
* Inverse document frequency (IDF) value can tell us how much important this word provide to us depends on the frequency of the word appear in all documents, which will be the whole data set in our case. We can also think of IDF as a weight of the word since at final we will multiply TF with IDF. For example, the IDF value of “is”, “the” are undoubtedly low since they are common to see in every comment.

### Why is TF-IDF used in Machine Learning?[[1]](#footnote-2)

Since TF-IDF can predict the importance of individual words in a data set, it is of paramount importance to the role of machine learning in natural language processing.

### How to apply TF-IDF to help predict toxic comment?[[2]](#footnote-3)

Step 1: build the TF-IDF model by using TfidfVectorizer[[3]](#footnote-4) package to get X\_train and X\_test.

Step 2: Define Naives Bayes Model[[4]](#footnote-5), which is the most straightforward and fast classification algorithm, find out the probability of an event based on the previous events that occurred.

Step 3: Apply the Naives Bayes Model to calculate the probability of each case (toxic comment or not-toxic comment) in each classification (each column of labels)

Step 4: multiply X\_train with the probability we calculate from step 3.

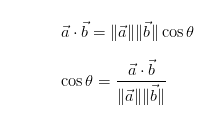
Step 5: using new X\_train (result of step 4) to train the model of any algorithm we could apply to. For instance, Logistic Regression or Decision Tree Classification.

### Word embedding

In word embedding, keywords are embedded in a vector format with the goal of better understanding the similarities in our feature set which in our case would be words.

Step 1: Build vectors of negative sentences.

Step 2: There are many ways of going about finding the similarities between words and a lot of NLP science is built on these nuances. Because of the objective nature of our data we suggest using the cosine distance. We will solve the dot product of the following equation to find the similarities between the vectors.



Step 3: Give a weighted score to each negative sentence to understand their similarities via KNN.

### Word2Vec

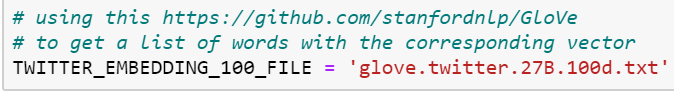
Just like its name, Word2Vec is a technique that convert a text corpus to a set of **vectors**. Word2Vec is one of the most famous and the most efficient methods in Word Embedding. The purpose of Word2Vec is that since it will convert a word to a vector, it makes similar words converting to similar vectors and group similar words in some certain dimensions area.

### Why is Word2Vec used in Machine Learning?

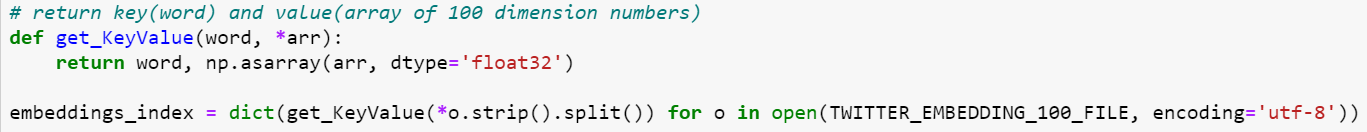
Word2Vec converts the words to vectors, we hence project all words to certain dimensions (based on how the programmer design) and found the similarities. It makes us easier to train numeric data than a non-numerical data.

### How to apply Word2Vec to help predict toxic comment?

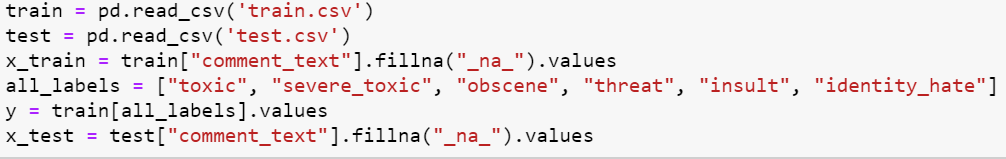
Step 1: We will first download the pre-trained word vectors file which is provided by <https://github.com/stanfordnlp/GloVe>. The file contains a big amount of words and their corresponding vector, which are all 100 dimensions.



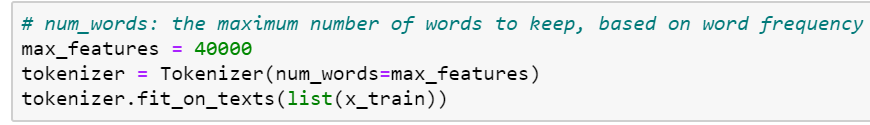
Step 2: Convert the result we got from step1 to a dictionary and store it in **embeddings\_index**, so we can use it easily later in the program.



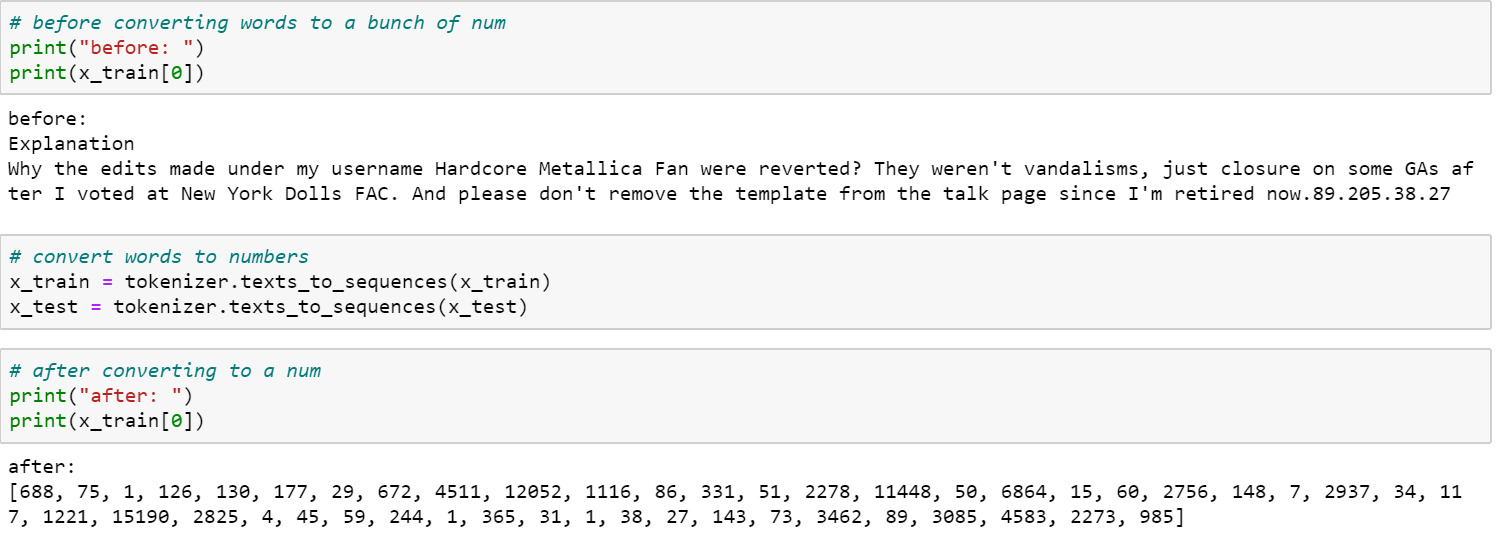
Step 3: Load **train.csv** and **test.csv**. Assign values of “**comment\_text**” in **train.csv** to **x\_train** and values in **test.csv** in **x\_test**; **y** contains the label values in **train.csv**.



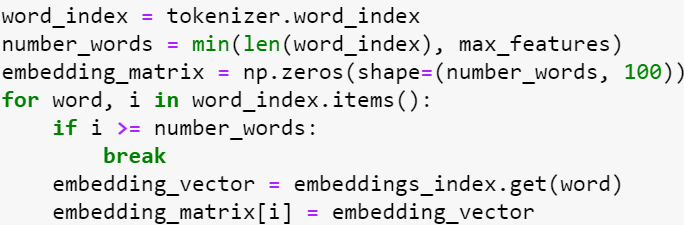
Step 4: Now we are going to use **Tokenizer** from **keras.preprocessing.text** to vectorize a text corpus. Now **tokenizer** contains a dictionary **word\_index** based on the frequency rank of the word among the dataset, such as if you write like **tokenizer.word\_index[**'why'**]**, it will return the frequency rank of “why”.



Step 5: We are now converting all the words in **x\_train** and **x\_test** to numbers which is defined by **tokenizer** from step 4. The result will like the screenshot shown below. If the word is not in **tokenizer.word\_index**, we will just ignore it.



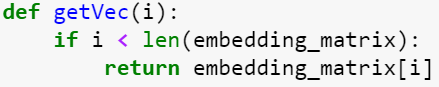
Step 6: In step 2, we got **embeddings\_index** to store every word with their corresponding 100-dimension vector in the pre-trained word vectors file. Now, we are going to create a new matrix named **embedding\_matrix** to store every word in **tokenizer.word\_index** whose frequency rank is less than 40000 and its corresponding 100-dimension vector.

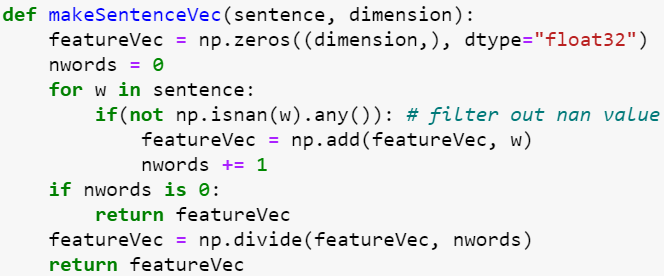


Step 7: Create two methods which we are going to use in step 8:

**getVec()** will just return the vector of the certain word which already is converted to a number i.

**makeSentenceVec()** will calculate the average of all not null words 100-dimension vector in this sentence, and return this average vector to be the vector of this sentence.





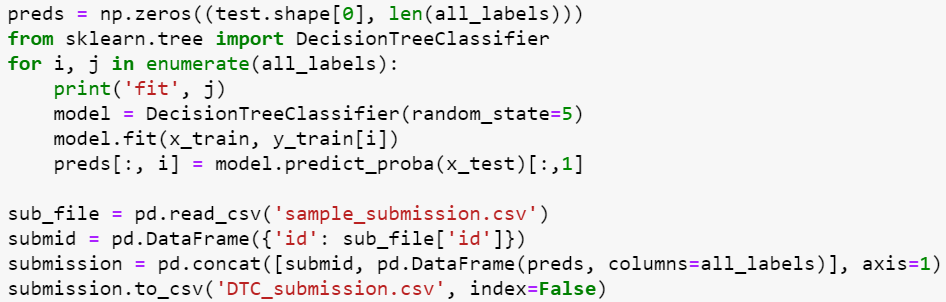
Step 8: Convert every sentence in **x\_train** and **x\_test** to a 100-dimension vector.



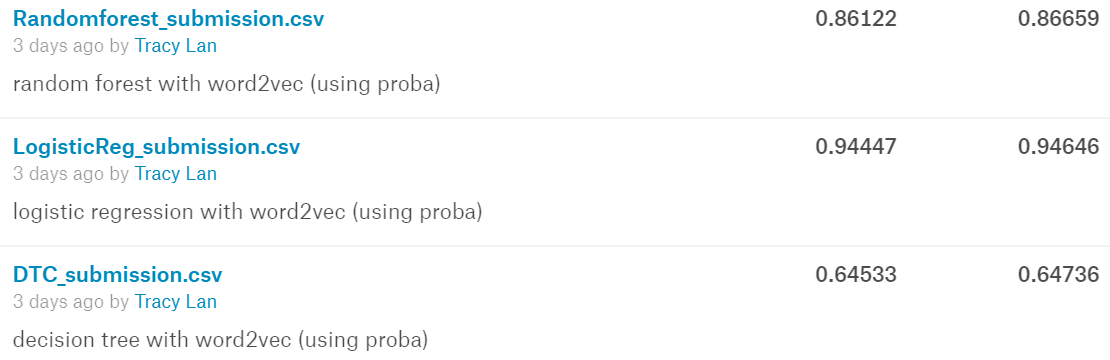
Step 9: Get **y\_train** from y, which contains 6 list of values for each different label.



Step 10: Now we can use the data now we have to predict the models. This time, only KNN will take too long time, but Decision Tree Classifier, Logistic Regression, and Random Forest works well.



### Result: We got a pretty nice accuracy when using Logistic Regression and Random Forest. Although the accuracy of Decision Tree Classifier is not that high, but the result and the processing time is still much better than One-Hot Encoding.



CountVectorizer

CountVectorizer is a tokenizing method used in word processing and allows the conversion of words into tokens to be used for classification. Where tokenizing converts strings and gives an integer id for each possible token. CountVectorizer returns a vector with a length of the entire vocabulary and an integer count for the number of times each word appeared in the document. One of the main benefits of using CountVectorizer with our data that consists purely on non-numerical strings is to able to use our standard ML with the tokens CounterVectorizer generates.

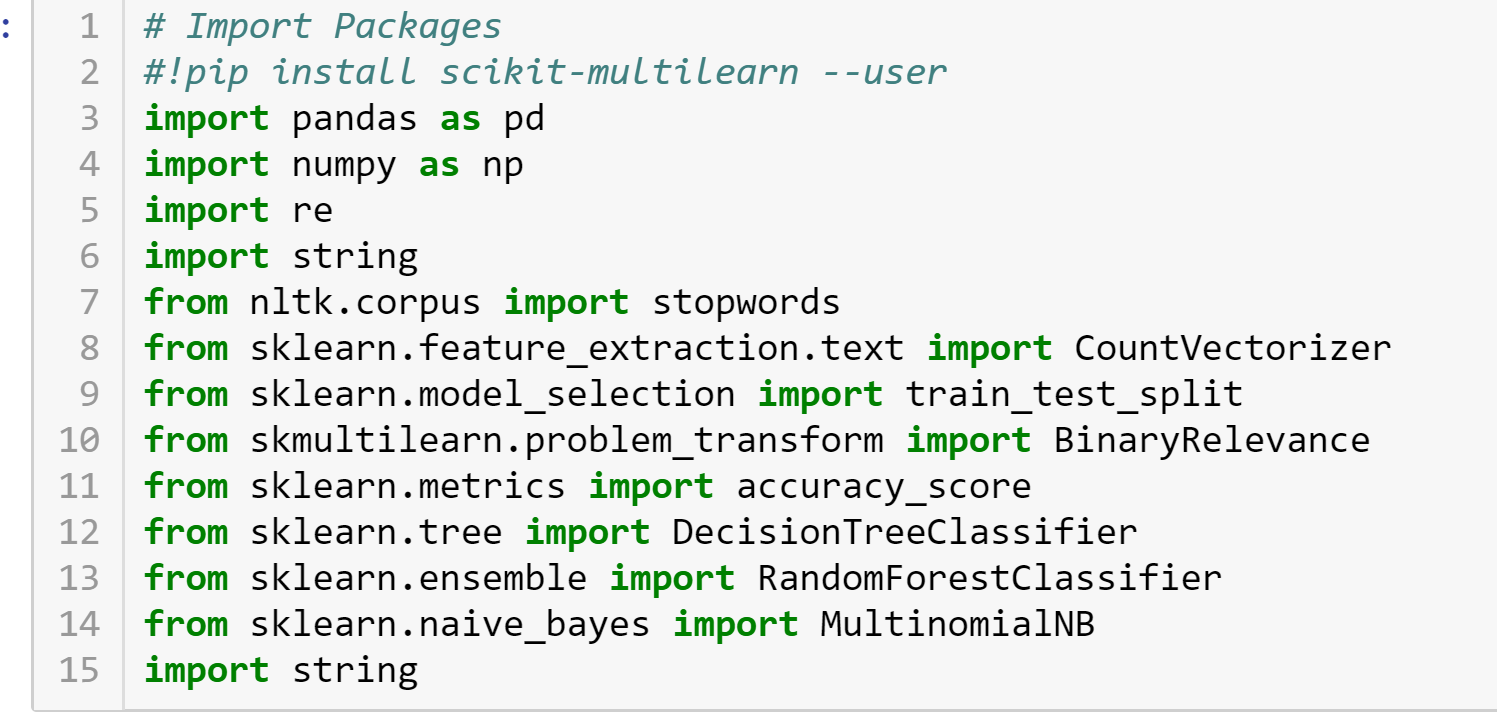
Why is CounterVectorizer used for Machine Learning?

Often time datasets contain features that are non-numerical and cannot be easily converted to become numerical. As in the case with the toxic comment data that has only one feature that consists of sentences. Normally techniques such as OneHotEncoding are often used to make dummy columns of the non-numerical features, we cannot do so with sentences because there is potentially an infinite of sentence combinations that could be possible. CounterVectorizer provides a method of converting word text into tokens, this method is often referred to as Bag-of-Words Model. CounterVectorizer is like TF-IDF that focuses on the occurrence of a word in a document, CountVectorizer will return an integer count for the number of times a given word appears in a document as well as encode the words. Now that the words have tokenized, we can use common ML techniques such as decision tree or random forest to train and test our data.

How to apply to CountVectorizer to toxic comments?

Step 1:

First import packages that will be used



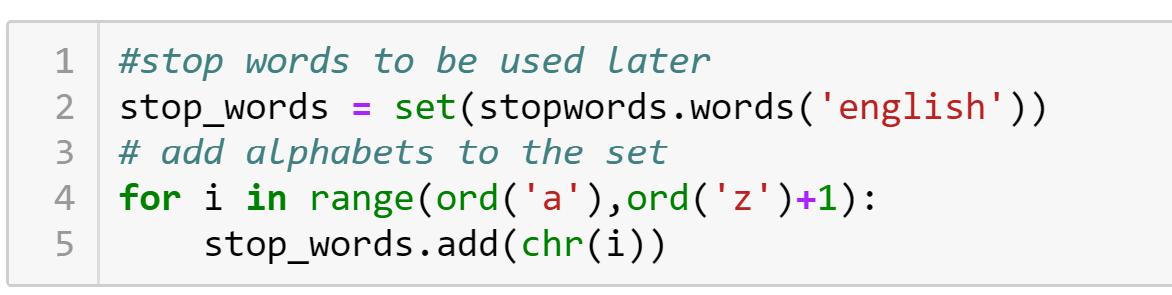
Step 2:

Next, we import the document files and remove unknown words in the document.



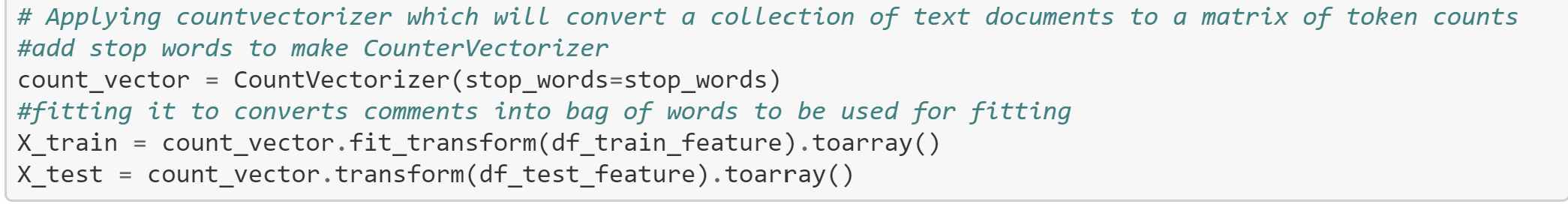
Step 3:

Next, we add the alphabet to our stop words, the reason we do this is to initialize our CountVectorizer which uses stop words for tokenization of text



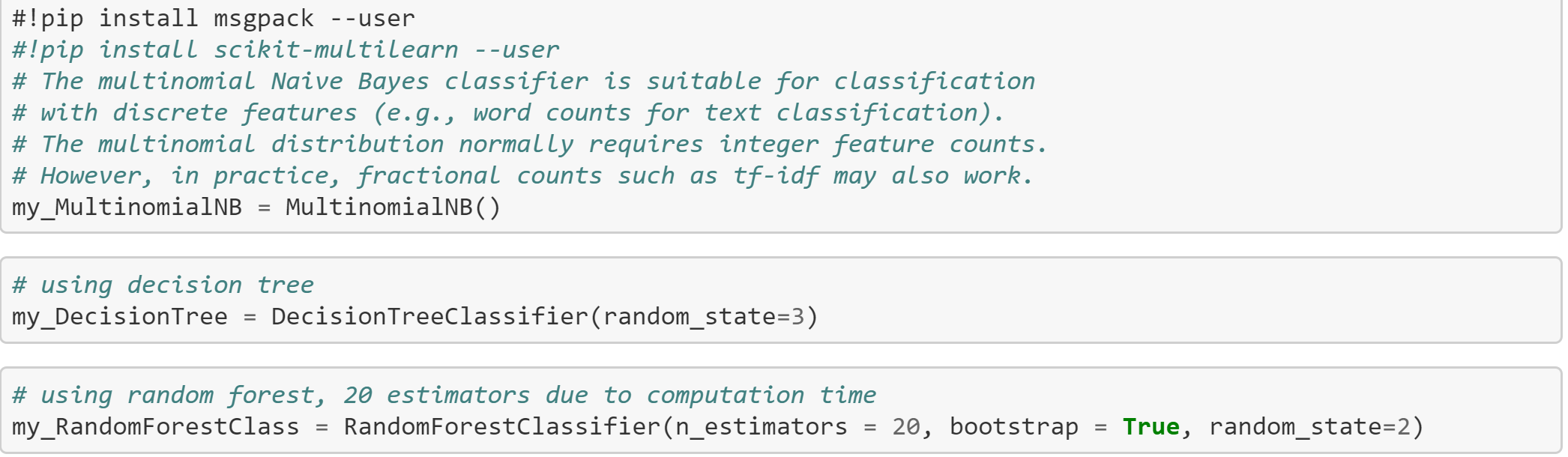
Step 4:

Now that we have our stop words, we perform the actual vectorization of our document. We first fit our CountVectorizer to our training data then we use that same count vectorizer to transform our test data



Step 5:

Once we have tokenized test and train data, we can perform classification, need to instantiate the classifier that will be used namely MultinomialNB, Decision Tree, and Random Forest



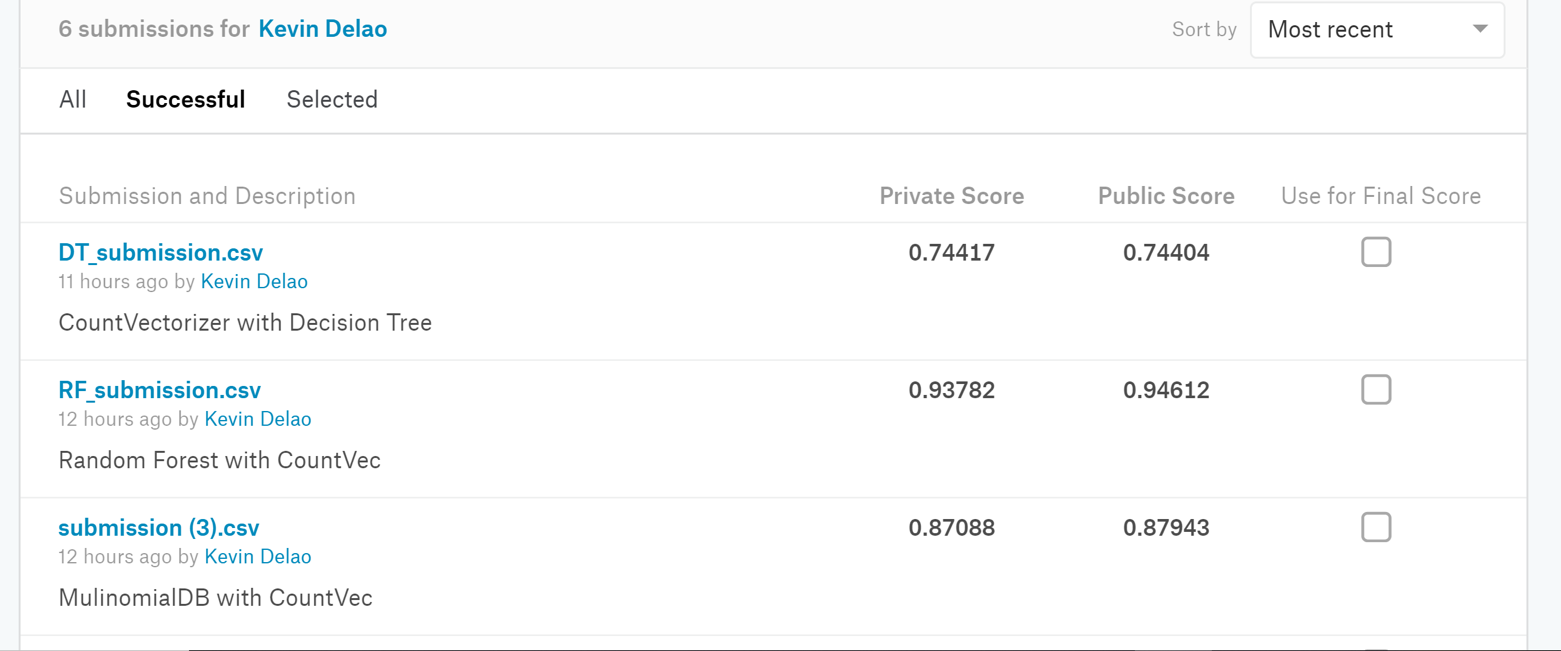
Step 6:

Finally, we do classification for all three ML methods and save to submission files. The way de classification is to train the model for each individual label and predict the probability of X\_test using our trained model. Overall MultinomialNB and Random Forest performs the best, Decision Tree takes much longer than the other two to finish.



Result:

Overall Random Forest seems to give the best accuracy.



Conclusion:

Overall CounterVectorizer does provide a method to perform on classification on features that are not numerical such as text, but CountVectorizer is inefficient for accuracy and performance due it being similar to OneHotEncoding. CountVectorizer will count the appearance of a word in the document and transform the word into a numerical column which in principle is useful, but with many words our features rapidly grows in dimension. Dimensionality reduction could potentially be used to reduce the dimensions, but I am not sure if there will information lost if reduction is performed.

### K-Nearest Neighbors (KNN) Classification.

### Purpose: Using KNN Classification to build a model predicting toxic comments.

### Approach:

Step 1: import “train.csv” to train data frame by Panda framework

Step 2: Create feature matrix **X\_train** based on train data frame with column “comment\_text”

Step 3: Create a series of labels **y\_train** (binary labels) of train data frame.

Step 4: import “test.csv” to test data frame.

Step 5: Create feature matrix **X\_test** based on test data frame with column “comment\_text”.

Step 6: import “test\_labels.csv” to test\_labels data frame.

Step 7: Create a series of labels **y\_test** (binary labels) of test\_labels data frame.

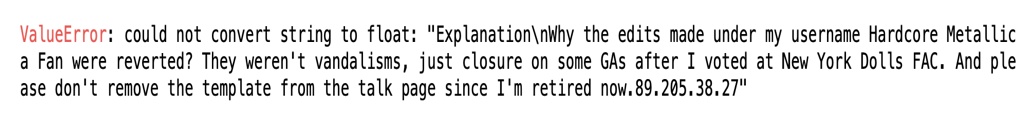
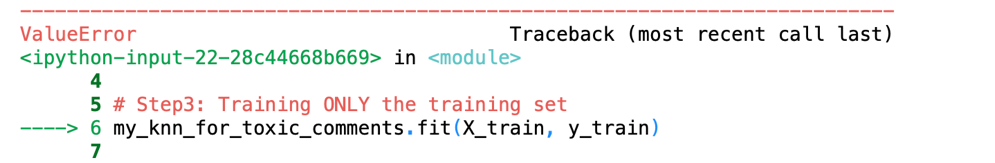
Step 8: Instantiate a KNN object and build KNN model with **X\_train** and **y\_train**

Step 9: Use KNN model above to predict **X\_test.** This step returns **y\_predict.**

Step 10: Check the accuracy between the prediction above (**y\_predict**) with **y\_test**. This step returns the **accuracy.**

### Challenges:

### Unable to build KNN model (Step 8). The errors are showed as below



The reason is because column “comment\_text” in X\_train contains String values instead of numerical values.

Solutions:

Encode “comment\_text”. We have two possible ways to encode value: OneHotCoding, TF-IDF, and LabelEncoder

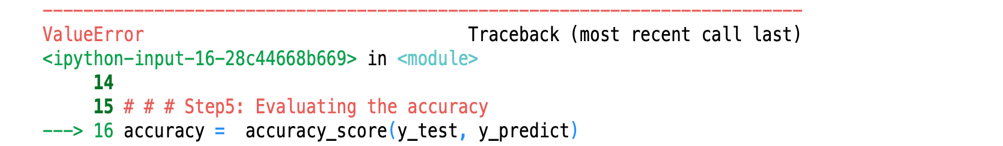
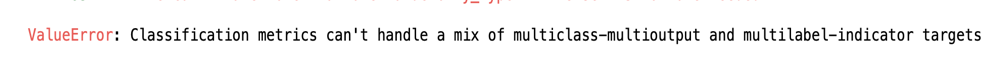
* OneHotCoding approach: inefficient in this case because each comment is mostly different (unless 2 comments are copied from each other). As a result, the approach might create huge number of columns. We did try to encode by OneHotCoding, but the process took more than 1 hour and still couldn’t finish encoding process.
* TF-IDF approach[[5]](#footnote-6): we have successfully [5 steps of applying TF-IDF.](#_How_to_apply) However, after getting the train model, it took very long time to make a predict model.
* LabelEncoder approach: because of OneHotCoding’s difficulties, it left us to encode by LabelEncoder, which encode labels with a value between 0 and n\_classes-1 where n is the number of distinct labels. As a result, we don’t create any columns or modify our data frame. We simply replace “texts” by numerical.

Therefore, encoding by LabelEncoder seems to be the better choice for this situation. It is worth to notice that, based on researches from a variety source in internet, encoding by OneHotCoding or TF-IDF is recommended for Machine Learning as well as Data Science.

A screenshot of a social media post

Description automatically generatedAfter encoding by LabelEncoder, we get the result below. The warning is because we didn’t give detail information which is text values of “comment\_text” column. This is impossible in this case because we don’t know exactly how many types of comments were given.

### Unable to evaluate the accuracy (Step 10). The errors are showed as below.



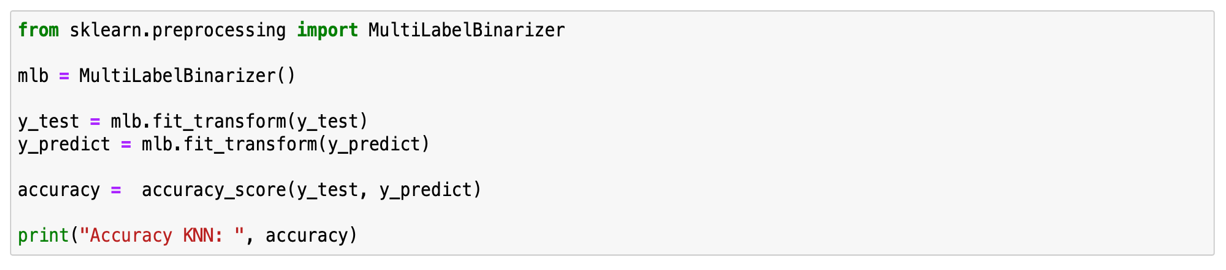
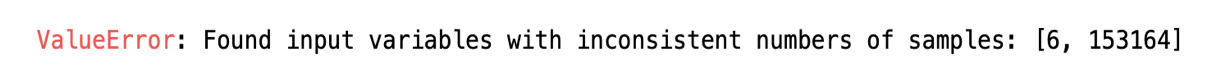
The reason is because the **y\_predict** is multilabel type (we encoded **X\_train** and use it to build the KNN model), while **y\_test** is multiclass-multioutput type.

The errors demonstrate that sklearn.metrics.accuracy\_score doesn’t support to evaluate the accuracy between the mix of two different types of parameters.

Solutions:

There is one possible way to solve the situation. That is instead of using KNN, using Multilabel KNN (MLkNN) which is also supported from sklearn. However, after trying MLkNN, the errors still exist.

Because MLkNN couldn’t solve the problems, we tried MultiLabelBinarizer technique from sklearn.preprocessing. However, we got another type of error.



We figured out that because after we transformed y\_test and y\_predict by MultiLabelBinarizer the shape of both y\_test and y\_predict are changed.

Before MultiLabelBinarizer:

A close up of a logo

Description automatically generated

A close up of a logo

Description automatically generatedAfter MultiLabelBinarizer:

### Conclusion

Due to 2 main factors from Challenges section, we all agree that using KNN for this project is not applicable.

* The first main reason is we have more than 2 labels to classify (6 labels in this case), while KNN supports, mostly, for classification of only 1 label.
* The second reason is the “comment\_text” contains all randomly string causing difficulty of encoding process.

Suggestion:

To solve the problems that we analyzed of KNN approach, we must come up with a way to encode “comment\_text” column more efficiency and dealing with classification of multiple labels. With multiple labels problem, we realized that applying Decision Tree or Logistic Regression Classification is a better approach than KNN Classification approach.

### Decision Tree Classification.

### Purpose: Use Decisions Tree as the learning algorithm to predict toxic comments.

### Decision Tree Background:

Decision tree is a learning method used for supervised learning. A decision tree is essentially a binary tree where the nodes represent choices. Each node at each level of our tree will represent a feature that provides the most information about the labels of the data. At each node we will also split the data based on our node to finally arrive at the prediction for a row of data.

Setup:

We need to import our libraries such as NumPy, Panda, and import DecisionTreeClassifier from sklearn.tree so we can use pythons provided decision tree classifier.

Training Stage:

1. We will first import the ‘train.csv’ file which contains the features and labels for the data.
2. We will make a variable called X and assign it the feature ‘comment\_text’ from train.csv file
3. We will import the ‘test\_Labels.csv’ file and use it assign a variable y the labels of the train.csv file.
4. We will then use our DecisionTreeClassifier method to fit our data using our X and y variables

Testing Stage:

1. Once we have our model, we trained from our testing stage we will now use our model to predict the labels for testing data.
2. We will import our ‘testing.csv’ file and use our model from our testing stage to predict the labels of our testing data that only contains features.
3. Once we have our prediction, we can then compare our accuracy by using the ‘test\_labels’ file which contain the actual labels of the testing data, we will then have a measure of how well our learning model performed.

### Challenges and Problems

As it stands if we were to try to run the decision tree classifier on either the testing or training data, we will run into an error telling us we cannot train the model with the provided features. The reason being is that the only feature of the data being ‘comment\_text’ are String values instead of numerical values that are meant to be used for decision tree classifiers.

Often when non numerical data exist, the non-numerical can be encoded to numerical values to allow the classifier to work. Unfortunately, there no easy way to do so with the ‘comment\_text’ values because each of the values of the features are sentences that vary greatly from each other and that show consistent pattern that we use to encode them to numerical values.

### Possible Solutions:

One possible solution to encode our feature data is to research a way to find a pattern where we can encode the features. This can be done by looking into word processing where we would be able to figure out how words can be classified and then create corresponding numerical values for our features.

Another possible solution is to use feature extraction algorithms that would allow us to use new and different features that can represent our data and still allow us to predict the labels of our data.

### Conclusion:

Decision Tree Classifier provide a very useful model for supervised learning, but currently as it stands, we are unable to use a decision tree classifier for our data due to the features of our data being non numerical. If we are able to figure out a method to encode our feature values to numerical values or come up with new features by using word processing and sort our words into groups such as with K Means algorithm, then we can use Decision Tree Classifiers for the data.

### Random Forest Classification

### Purpose: Use Random Forest as the learning algorithm to predict which comments are toxic.

### Random Forest Background:

Random Forest is a classification algorithm very similar to Decision Tree with a few key differences. A random forest consists of an ensemble of random decision trees. The benefit of using an ensemble rather than say just averaging a group of decision tree together is due to random forests avoiding overfitting and cancels out any biases. The other reason random forests are very useful is that it will generate the decision tree that is best overall for the data we provide. There are some draw backs when it comes to computation time.

Setup: We need to import our libraries such as NumPy, Panda, and import **RandomForestClassifier** from **sklearn.ensemble** so we can use pythons provided random forest classifier.

Training Stage

1. We will first import the ‘train.csv’ file which contains the features and labels for the data.
2. We will make a variable called X and assign it the feature ‘comment\_text’ from train.csv file
3. We will import the ‘test\_Labels.csv’ file and use it assign a variable y the labels of the train.csv file.
4. We will then use our Random Forest method to fit our data using our X and y variables

Testing Stage

1. Once we have our model, we trained from our testing stage we will now use our model to predict the labels for testing data.
2. We will import our ‘testing.csv’ file and use our model from our testing stage to predict the labels of our testing data that only contains features.
3. Once we have our prediction, we can then compare our accuracy by using the ‘test\_labels’ file which contain the actual labels of the testing data, we will then have a measure of how well our learning model performed.

### Challenges and Problems:

Like the decision tree section, we still don’t have a solution to convert our toxic comment section into features we can use to predict our labels. If we try to run the data with the random forest classifier as is we will get errors. I initially tried to use **OneHotEncoding** to create dummy columns for out one feature column, but the problem with using **OneHotEncoding** is that it will create a dummy column for each variation of a given column, meaning if we have an infinite amount of possible sentences then we will have an infinite amount of dummy columns.

### Possible Solutions:

One avenue I have begun to research is Sentiment Analysis which tries as the name implies classify the sentiment of a given word or sentence. Sentiment Analysis is currently being used in a variety of applications from Facebook and YouTube comments to Movie Reviews. I plan to use the methods applied in Movie Reviews to classify review sentiment and apply it to classifying a sentence as toxic or not.

### Conclusion:

Random Forests provide a powerful method to develop a large assortment of decision trees to best represent our data. Random forests also avoid the problem of overfitting that is present with decision trees. Random Forests aren't perfect however due to the high computation time associated with them and the complex results that it produces.

### Logistic Regression Classification.

Due to fact that the project requires prediction of probability of multi-labels which are different type of toxic comments, applying Logistic Regression might be a good approach in this case.

Why does Logistic Regression classification help to predict the probability of multi-labels problems?[[6]](#footnote-7)

Logistic Regression helps close the gap between regression and classification and provides a formulaic approach to quantifying certain data with a view to classify entries into a discrete number of groups.

Is Decision Tree better than Logistic Regression approach?[[7]](#footnote-8)

Both Decision Trees and Logistic Regression have their pros and cons. Like many choices in computer science a clear winner is not so easy to pick. Logistic regression provides us a more quantitative way of approaching classification that may help while decision trees, especially when taken as a forest are a powerful analytical tool that can consider many different factors in data in a way other structure cannot.

### 

### Challenges

We can’t just apply Logistic Regression to the data without encoding it because the data contains a bunch of comments which are texts. Therefore, we must process the data before using Logistic Regression to predict the toxic comments.

### Applying Logistic Regression

After getting the TF-IDF model, to build the Logistic model for training and testing purpose, we should apply the Naives Bayes Model to calculate the average of TF-IDF (probability). Next, we build a Logistic Regression model based on the data after performing TF-IDF model and Naives Bayes Model. We shall build the Logistic Regression model with regularization constant (C = 4) because the lower the constant, the higher the regularization. After we get the train model, now we shall use the train model make the predictions of the test data. In Logistic Regression, to predict the probability we use method “predict\_proba” instead of “predict” normally.

### TF-IDF

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

A screenshot of a social media post

Description automatically generated

### Logistic Regression



A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Result from Kaggle

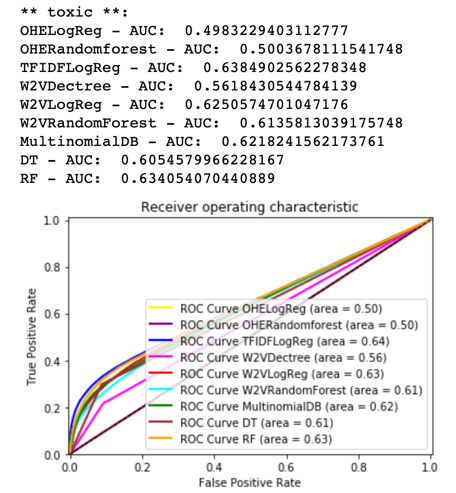
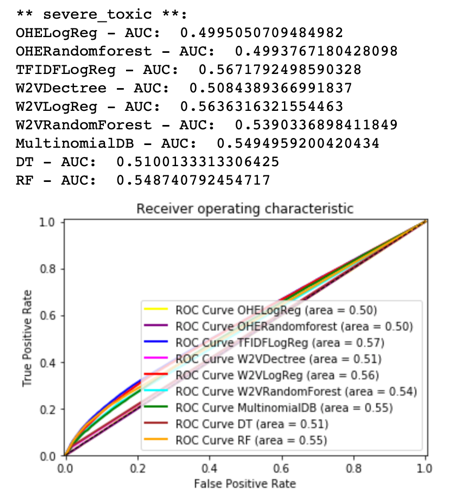
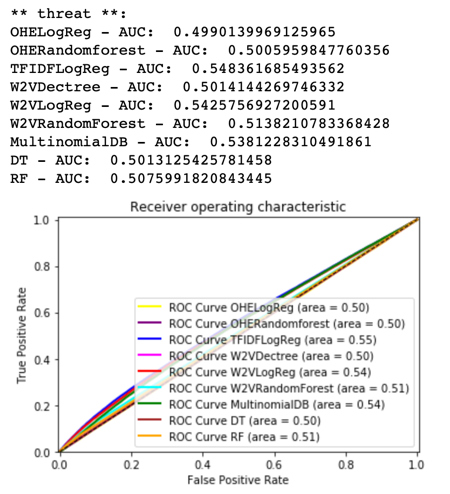
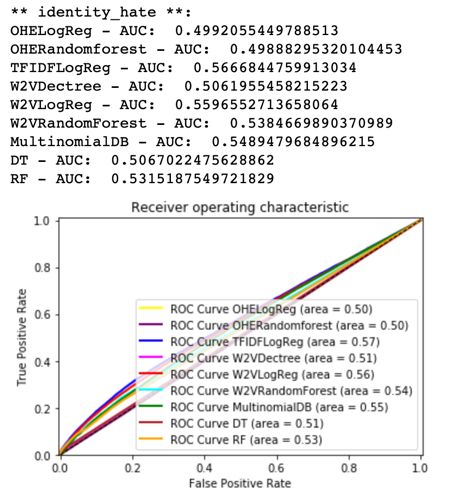
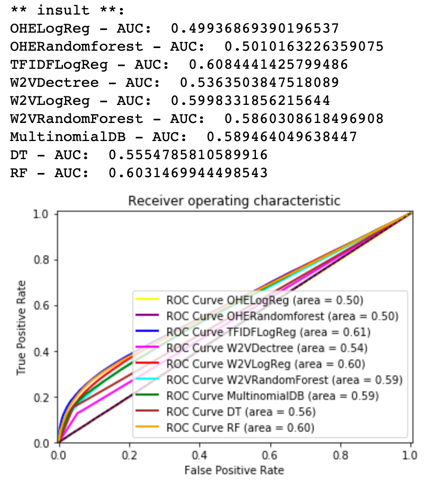
A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated

# AUC AND ROC RESULTS

Overall, the results are not very good due to the complexity of analyzing features from ‘comment\_text’ input.

# source code

## Data Files

<https://github.com/fkvn/Toxic-Comments/tree/master/Files>

## Testing process (includes all files relate to analyzation process)

<https://github.com/fkvn/Toxic-Comments/tree/master/Testing%20processes>

## One Hot Encoding

<https://github.com/TracyCYLan/ToxicComment/blob/master/One%20Hot%20Encoding.ipynb>

## TF-IDF

<https://github.com/fkvn/Toxic-Comments/blob/master/TFIDF-only.ipynb>

## Word2Vec

<https://github.com/TracyCYLan/ToxicComment/blob/master/Word2vec.ipynb>

## Decision Tree Classification

Test Code:

<https://github.com/mars404/DataScience-Group-Code/blob/master/Test%20Code%20Decision%20Tree.ipynb>

DT with CountVectorizer:

<https://github.com/mars404/DataScience-Group-Code/blob/master/Project%20Code.ipynb>

## Logistic Regression Classification

<https://github.com/fkvn/Toxic-Comments/blob/master/Predic%20Toxic%20Comments%20with%20TF-IDF%20and%20Logistic%20Regression.ipynb>

## Random Forest Classification

Test Code:

<https://github.com/mars404/DataScience-Group-Code/blob/master/Test%20Code%20Random%20Forest.ipynb>

RF with CountVectorizer:

<https://github.com/mars404/DataScience-Group-Code/blob/master/Project%20Code.ipynb>

## MultinomialNB with CountVectorizer

<https://github.com/mars404/DataScience-Group-Code/blob/master/Project%20Code.ipynb>

## AUC and ROC results

<https://github.com/fkvn/Toxic-Comments/blob/master/Results/AUC%20and%20ROC%20results.ipynb>

# Responsibility of each member (Weekly reports)

## Week 1 – 10/18/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Making template for project final report. Analyzing project’s details including description, requirements, and data description. |
| Kevin D. | Analyzing Decision Tree Classification to evaluate if the approach is suitable for the project’s requirements. Identify advantages and disadvantages (challenges) when applying Decision Tree to the project |
| Kevin N. | Analyzing KNN Classification to identify steps, challenges, and make suggestions to solve if there are problems by applying KNN for the final project. Providing related information which supports the purpose of the final project |
| Tracy L. | Researching and analyzing NLP (Natural Language Processing) to apply for the final project |

## Week 2 – 10/25/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Assisted with one Hot Encoding for our project. |
| Kevin D. | Looked into Random Forest and how we can use it to do our classification for toxic or non-toxic |
| Kevin N. | Re-modeling TF-IDF and implement Logistic Regression algorithm to the TF-IDF model |
| Tracy L. | Do certain research on one hot encoding |

## Week 3 – 11/01/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Word Embedding and different ways this information can be used for classification. |
| Kevin D. | Looked into Sentiment Analysis and how we can use methods used to classify words in our project |
| Kevin N. | Finishing TF-IDF and Logistic Regression. Organizing and proofing final project report. |
| Tracy L. | Work on one hot encoding codes and test its accuracy using KNN |

## Week 4 – 11/08/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Researched word embedding techniques |
| Kevin D. | Looked into the Natural Language Toolkit related to sentiment analysis |
| Kevin N. | Researching to improve TFIDF accuracy. Figure it out if there are any way to boost words analyzed process for the features. |
| Tracy L. | Do research on word2Vec |

## Week 5 – 11/15/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Researched techniques of classification. |
| Kevin D. | Applying the Natural Language Toolkit for Decision Tree and Random Forest |
| Kevin N. | Testing Decision Tree and Logistic Regression with TFIDF |
| Tracy L. | Doing code using word2Vec, trying with different models like Decision Tree, Logistic Regression, and Random Forest |

## Week 6 – 11/22/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Researched Glove for word embedding |
| Kevin D. | Switched from the natural processing toolkit to CountVectorizer to use alongside, DT, Random Forest and MultinomialNB |
| Kevin N. | Researching to improve TFIDF Logistic regression result |
| Tracy L. | Redo one hot encoding, trying to solve the time-consuming problem which we got in previous weeks. |

## Week 7 – 11/29/2019

|  |  |
| --- | --- |
|  | Responsibility |
| Mark C. | Finishing slides on word embedding |
| Kevin D. | Finishing documentation for CountVectorizer |
| Kevin N. | Finishing reports and exports AUC + ROC results for all methods and making slides for final project. |
| Tracy L. | Doing the documentation part in word2vec and one-hot encoding. |

1. <https://monkeylearn.com/blog/what-is-tf-idf/> [↑](#footnote-ref-2)
2. <https://medium.com/@raghav96_77243/what-the-3d5a12b06f50> [↑](#footnote-ref-3)
3. <https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html> [↑](#footnote-ref-4)
4. <https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn> [↑](#footnote-ref-5)
5. <https://github.com/fkvn/Toxic-Comments/blob/master/KNN_Testing_with%20TFIDF.ipynb> [↑](#footnote-ref-6)
6. <https://www.statisticssolutions.com/what-is-logistic-regression/> [↑](#footnote-ref-7)
7. <https://www.displayr.com/decision-trees-are-usually-better-than-logistic-regression/> [↑](#footnote-ref-8)