



Syracuse University

Engineering and Computer Science Department

SOCIAL MEDIA DATA MINING PROJECT REPORT

“Malignant Comment Classification”

Nikhil Gohil SUID: 287698643

Tejas Mehta SUID: 526709790

Pratik Bhatia SUID: 862829553

Nikhil Bharadwaj SUID: 469484436

Hemanta Pattnaik SUID: 770755090

Syracuse University, NY, 13244

1. Problem Definition

1. Project Overview

Online platforms when used by normal people can only be comfortably used by them only when they feel that they can express themselves freely and without any reluctance. If they come across any kind of a malignant or toxic type of a reply which can also be a threat or an insult or any kind of harassment which makes them uncomfortable, they might defer to use the social media platform in future. Thus, it becomes extremely essential for any organization or community to have an automated system which can efficiently identify and keep a track of all such comments and thus take any respective action for it, such as reporting or blocking the same to prevent any such kind of issues in the future.

This is a huge concern as in this world, there are 7.7 billion people, and, out of these 7.7 billion, more than 3.5 billion people use some or the other form of online social media. Which means that every one-in-three people uses social media platform. This problem thus can be eliminated as it falls under the category of Natural Language Processing. In this, we try to recognize the intention of the speaker by building a model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate. Moreover, it is crucial to handle any such kind of nuisance, to make a more user-friendly experience, only after which people can actually enjoy in participating in discussions with regard to online conversation.

2. Problem Statement

Given a number of tweets (in Twitter) or any kind of other comments, sentences or paragraphs being used as a comment by a user, our task is to identify the comment as whether it is a malignant comment or no. After that, when we have a collection of all the malignant comments, our main task is to classify the tweets or comments into one or more of the following categories – toxic, severe-toxic, obscene, threat, insult or identity-hate. This problem thus comes under the category of multi-label classification problem.

There is a difference between the traditional and very famous multi-class classification, and the one which we will be using, which is the multi-label classification. In a multi-class classification, each instance is classified into one of three or more classes, whereas, in a multi-label classification, multiple labels (such as – toxic, severe-toxic, obscene, threat, insult or identity-hate) are to be predicted for the same instance.

Multiple ways are there to approach this classification problem. It can be done using –

- Multi-label methods which belong to the problem transformation category: Label Power Set (LP), Binary Relevance (BR), BR+ (BRplus), and classifier chain.

- Base and adapted algorithms like: J48 (Decision Tree), Naïve Bayes, k-Nearest-Neighbor (KNN), SMO (Support Vector Machines), and, BP-MLL neural networks.

Further, out of the total dataset used for experimenting these algorithms, 70% was used for training and 30% was used for testing. Each testing dataset was labelled and thus for each algorithm using the predictions and labels, calculation of metric such as hamming-loss, accuracy and log-loss was done. The final results have been compiled on the basis of values obtained by algorithmic models in hamming-loss and log-loss combined.

3. Evaluation Metrics

- **Label bases metrics** include one-error, average precision, etc. These can be calculated for each labels, and then can be averaged for all without taking into account any relation between the labels if exists.

Average Precision (AP): Average precision is a measure that combines recall and precision for ranked retrieval results. For one information need, the average precision is the mean of the precision scores after each relevant document is retrieved, where, P and R are the precision and recall at the threshold.

- **Example based metrics** include accuracy, hamming loss, etc. These are calculated for each of the examples and then averaged across the test set. Let –

Accuracy is defined as the proportion of correctly predicted labels to the total no. of labels for each instance.

Hamming-loss is defined as the symmetric difference between predicted and true labels, divided by the total no. of labels.

When we have a look at the data, we observe that every 1 in 10 samples is toxic, every 1 in 50 samples is obscene and insulting, but the occurrences of sample being severe_toxic, threat and identity hate is extremely rare. Thus, we have skewed data, and accuracy as metric will not give us the required results. Thus, we will be using hamming-loss as the evaluation metric.

2. Analysis

1. Data Extraction

One of the most time-consuming tasks in Data science is collection and labelling of data. Even when we collect data, a major problem we face is the availability of useful data in the collected dataset. What we noticed was that if we gave the track parameter as (" ") that is space or "a" which is available in all comments, out of 100000 tweets collected around 14000 tweets were toxic which means that many of them were vague or non-toxic. We could not use this collected data for our analysis. So, we collected data using mainly cuss words in the track parameter because of which almost all of the data we collected could be classified as one of the malignant comment classes and we could perform analysis on the predicted results.

In order to achieve this result we performed the following steps:-

1. First, we set the authorization using the following commands:

```
auth = OAuthHandler(ckey, csecret)
auth.set_access_token(accessToken, asecret)
```

2. We are then creating a twitterStream as follows

```
twitterStream = Stream(auth, twitter_streaming())
```

3. From the twitterStream we will be filtering the data with the required keywords as follows

```
#twitterStream.filter()
twitterStream.filter(track=["hate", "fu*k", "a**hole",
                           "b**ch", "bomb", "n*gga" ],| languages=["en"])
```

4. I am extracting the following useful information from the twitterStream.

- a. tweetID
- b. dateTime
- c. tweet
- d. screen_name
- e. followers_count
- f. source
- g. Country
- h. country_code
- i. full_name
- j. name
- k. place_type
- l. quote_count
- m. reply_count
- n. retweet_count
- o. favorite_count

5. We have written and extracted the data into a csv file called hate_final.csv using the following command

```
#Use csv writer
csvWriter = csv.writer(csvFile)
```

For the current scope of the project we are using mainly the tweet to predict what kind of toxicity is present in the file. We have extracted additional information like location, followers count etc.

2. Data Exploration

The dataset consists of the following fields-

- id: An 8-digit integer value, to get the identity of the person who had written this comment
- comment_text: A multi-line text field which contains the unfiltered comment
- toxic: binary label which contains 0/1 (0 for no and 1 for yes)
- severe_toxic: binary label which contains 0/1
- obscene: binary label which contains 0/1
- threat: binary label which contains 0/1
- insult: binary label which contains 0/1
- identity_hate: binary label which contains 0/1

Out of these fields, the comment_text field will be preprocessed and fitted into different classifiers to predict whether it belongs to one or more of the labels/outcome variables (i.e. toxic, severe_toxic, obscene, threat, insult and identity_hate).

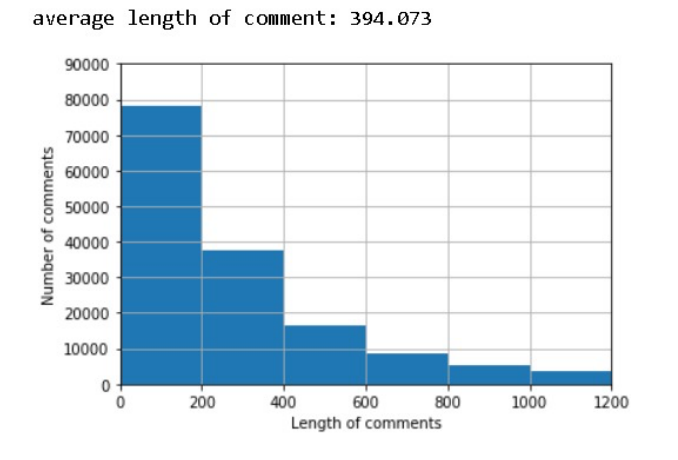
We have a total of 151203 samples of comments and labelled data, which can be loaded from train.csv file. The first 5 samples are as follows:

```
0    Nonsense?  kiss off, geek. what I said is true...
1    "\n\n Please do not vandalize pages, as you di...
2    "\n\n ""Points of interest"" \n\nI removed the...
3    Asking some his nationality is a Racial offenc...
4    The reader here is not going by my say so for ...
Name: comment_text, dtype: object
```

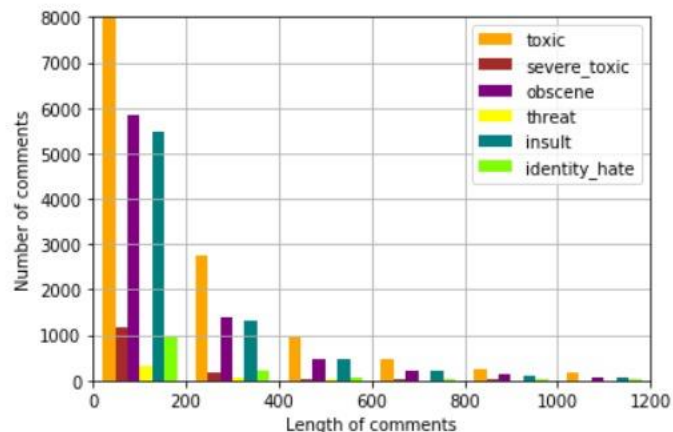
	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	1	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

One more very crucial aspect of the dataset is noticing the frequency occurrence of multi-labelled data. We can easily notice that approximately every 1 data out of 10 is toxic (9000 samples), every 1 in 20 samples is obscene and insulting (5000 samples), but the occurrences of sample being severe_toxic, threat and identity hate is extremely rare (800-900 out of 90000 samples). Overall 9790 samples are those which have at least one label and 5957 samples have 2 or more labels.

3. Exploratory Visualisation



→ From the first visualization we can observe that comments have varying lengths from within 200 up to 1200. The majority of comments have length up to 200, and as we move towards greater lengths, the number of comments keep on falling. Since including very long length comments for training will increase the number of words manifold, it is important to set a threshold value for optimum results.



→ From the second visualization, we can observe the number of words falling under the six different outcome labels toxic, severe_toxic, obscene, etc. along with their lengths. Here also similar to the first plot, we observe that most of the abusive comments have lengths under 200, and this number falls with the length of comments.

- Based on these plots, taking comments having lengths upto 400 for training is a good estimation of our data and can be expected to give acceptable results on testing later. Hence before preprocessing, we will be removing all comments with length more than 400 which will serve as our threshold.

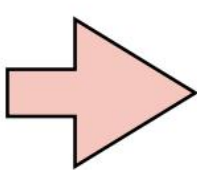
4. Algorithms & Techniques

As discussed in the Problem Statement section, any multi-label classification can be solved using either Problem Statement methods or Adaptation Algorithms.

PROBLEM TRANSFORMATION METHODS:

- **Binary Relevance Method:** This method does not take into account the interdependence of labels. Each label is solved separately like a single label classification problem. This is the simplest approach to be applied.

X	Y ₁	Y ₂	Y ₃	Y ₄
x ⁽¹⁾	0	1	1	0
x ⁽²⁾	1	0	0	0
x ⁽³⁾	0	1	0	0
x ⁽⁴⁾	1	0	0	1
x ⁽⁵⁾	0	0	0	1



X	Y ₁
x ⁽¹⁾	0
x ⁽²⁾	1
x ⁽³⁾	0
x ⁽⁴⁾	1
x ⁽⁵⁾	0

X	Y ₂
x ⁽¹⁾	1
x ⁽²⁾	0
x ⁽³⁾	1
x ⁽⁴⁾	0
x ⁽⁵⁾	0

X	Y ₃
x ⁽¹⁾	1
x ⁽²⁾	0
x ⁽³⁾	0
x ⁽⁴⁾	0
x ⁽⁵⁾	0

X	Y ₄
x ⁽¹⁾	0
x ⁽²⁾	0
x ⁽³⁾	0
x ⁽⁴⁾	1
x ⁽⁵⁾	1

- **Classifier Chain Method:** In this method, the first classifier is trained on input data and then each of the next classifier is trained on the input space and previous classifier, and so on. Hence this method takes into account some interdependence between labels and input data. Some classifiers may show dependence such as toxic and severe_toxic. Hence it is a fair deal to use this method.

X	y1
x1	0
x2	1
x3	0

Classifier 1

X	y1	y2
x1	0	1
x2	1	0
x3	0	1

Classifier 2

X	y1	y2	y3
x1	0	1	1
x2	1	0	0
x3	0	1	0

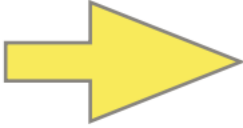
Classifier 3

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0

Classifier 4

- **Label Powerset Method:** In this method, we consider all unique combinations of labels possible. Any one particular combination hence serves as a label, converting our multi-label problem to a multi class classification problem. Considering our dataset, many comments are such that they have 0 for false labels all together and many are such that obscene and insult are true together. Hence, this algorithm seems to be a good method to be applied.

X	y1	y2	y3	y4
x1	0	1	1	0
x2	1	0	0	0
x3	0	1	0	0
x4	0	1	1	0
x5	1	1	1	1
x6	0	1	0	0



X	y1
x1	1
x2	2
x3	3
x4	1
x5	4
x6	3

In this, we find that x1 and x4 have the same labels, similarly, x3 and x6 have the same set of labels. So, label power set transforms this problem into a single multi-class problem.

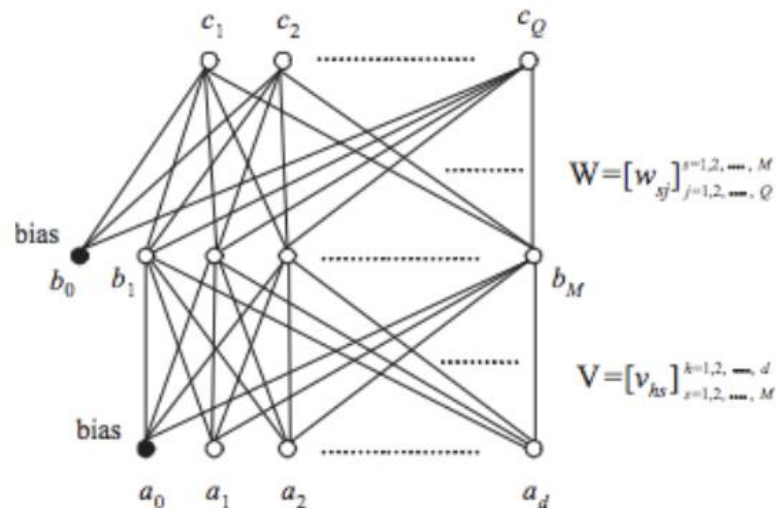
ADAPTION ALGORITHMS:

→ **MLKNN**: This is the adapted multi label version of K-nearest neighbors. Similar to this classification algorithm is the BRkNNaClassifier and BRkNNvClassifier which are based on K-Nearest Neighbors Method. This algorithm proves to give superior performance in some datasets such as yeast gene functional analysis, natural scene classification and automatic web page categorization.

Since this is somewhat similar to the page categorization problem, it is expected to give acceptable results. However, the time complexity involved is large and therefore it will be preferable to train it on smaller dataset.

→ **BP-MLL Neural Networks**: Back propogation Multi-label Neural Networks is an architecture that aims at minimizing pair-wise ranking error.

An architecture of one hidden layer feed forward neural network is as follows:



The input layer is fully connected with the hidden layer and the hidden layer is fully connected with the output layer. Since we have 6 output labels, the output layer will have 6 nodes. This algorithm can be trained in a reasonable amount of time with appropriate number of nodes in the hidden layer.

4. Benchmark Model

As we proposed in our proposal, we will be using Support Vector Machine with radial basis kernel (rbf) as the benchmark model (using Binary Relevance Method) Implementation of this model has been done along with other models using the Binary Relevance Method in the implementation section.

Since we have a large dataset, other classifiers using the bag of words model such as the MultinomialNB and GausseanNB are expected to work than this model.

But, in practice, it performs quite well. A detailed comparison and analysis between all these classifiers will hence be provided.

3. Methodology

1. Data Preprocessing

The following steps were taken to process the data:

→ **A string without all punctuations to be prepared:**

1. The string library contains punctuation characters. This is imported and all numbers are appended to this string. Our comment_text field contains strings such as won't, didn't, etc. which contain apostrophe character('). To prevent these words from being converted to wont or didn't, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.
2. make_trans(intab, outtab) function is used. It returns a translation table that maps each character in the intab into the character at the same position in the outtab string.

→ **Updating the list of stop words:**

```
print (stop_words)
```

```
['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any', 'are', 'aren't', 'as', 'at', 'be',  
'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by', 'can't', 'cannot', 'could', 'couldn't', 'di  
d', 'didn't', 'do', 'does', 'doesn't', 'doing', 'don't', 'down', 'during', 'each', 'few', 'for', 'from', 'further', 'had',  
'hadn't', 'has', 'hasn't', 'have', 'haven't', 'having', 'he', 'he'd', 'he'll', 'he's', 'her', 'here', 'here's', 'hers', 'h  
erself', 'him', 'himself', 'his', 'how', 'how's', 'i', 'i'd', 'i'll', 'i'm', 'i've', 'if', 'in', 'into', 'is', 'isn't', 'i  
t', 'it's', 'its', 'itself', 'let's', 'me', 'more', 'most', 'mustn't', 'my', 'myself', 'no', 'nor', 'not', 'of', 'off', 'o  
n', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours', 'ourselves', 'out', 'over', 'own', 'same', 'shan't', 'she', 'sh  
e'd', 'she'll', 'she's', 'should', 'shouldn't', 'so', 'some', 'such', 'than', 'that', 'that's', 'the', 'their', 'theirs',  
'them', 'themselves', 'then', 'there', 'there's', 'these', 'they', 'they'd', 'they'll', 'they're', 'they've', 'this', 'tho  
se', 'through', 'to', 'too', 'under', 'until', 'up', 'very', 'was', 'wasn't', 'we', 'we'd', 'we'll', 'we're', 'we've', 'we  
re', 'weren't', 'what', 'what's', 'when', 'when's', 'where', 'where's', 'which', 'while', 'who', 'who's', 'whom', 'why',  
'why's', 'with', 'won't', 'would', 'wouldn't', 'you', 'you'd', 'you'll', 'you're', 'you've', 'your', 'yours', 'yourself',  
'yourselves', ',', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',  
'w', 'x', 'y', 'z']
```

1. Stop words are those words that are frequently used in both written and verbal communication and thereby do not have either a positive or negative impact on our statement like “is, this, us, etc.”.
2. Single letter words if existing or created due to any preprocessing step do not convey any useful meaning and so they can be directly removed. Hence letters from b to z, will be added to the list of stop words imported directly.

→ **Stemming and Lemmatizing:**

1. The process of converting inflected/derived words to their word stem or the root form is called stemming. Many similar origin words are converted to the same word e.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem".
2. Lemmatizing is the process of grouping together the inflected forms of a word so they can be analyzed as a single item. This is quite similar to stemming in its working but differs since it depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document.

3. The wordnet library in nltk will be used for this purpose. Stemmer and Lemmatizer are also imported from nltk.

→ **Applying Count Vectorizer:**

1. To convert a string of words into a matrix of words with column headers represented by words and their values signifying the frequency of occurrence of the word Count Vectorizer is used.
2. Stop words were accepted, convert to lowercase, and regular expression as its parameters. Here, we will be supplying our custom list of stop words created earlier and using lowercase option. Regular expression will have its default value.

→ **Splitting dataset into Training and Testing:**

1. Since the system was going out of memory using train_test_split, I had jumbled all the indexes in the beginning itself.
2. The shuffle function defined here performs the task of assigning first 2/3rd values to train and remaining 1/3rd values to the test set.

2. Implementation

- We will be defining **function evaluate_score** for printing all evaluation metrics: Some implementations return a sparse matrix for the predictions and others return a dense matrix. So, a try except block were used to handle it.

```
from sklearn.metrics import hamming_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import log_loss

def evaluate_score(Y_test, predict):
    loss = hamming_loss(Y_test, predict)
    print("Hamming_loss : {}".format(loss*100))
    accuracy = accuracy_score(Y_test, predict)
    print("Accuracy : {}".format(accuracy*100))
    try :
        loss = log_loss(Y_test, predict)
    except :
        loss = log_loss(Y_test, predict.toarray())
    print("Log_loss : {}".format(loss))
```

- Then, we will start implementing various **problem transformation methods**. Binary Relevance, Label Powerset and Classifier Chain methods were included.

→ **Binary Relevance** method were implemented from scratch. It does not consider the interdependence of labels and basically creates a separate classifier for each of the labels. The code is as follows:

```
from sklearn.naive_bayes import MultinomialNB

# clf will be the list of the classifiers for all the 6 labels
# each classifier is fit with the training data and corresponding classifier
clf = []
for ix in range(6):
    clf.append(MultinomialNB())
    clf[ix].fit(X_train, Y_train[:, ix])

# predict list contains the predictions, it is transposed later to get the proper shape
predict = []
for ix in range(6):
    predict.append(clf[ix].predict(X_test))

predict = np.asarray(np.transpose(predict))
print(predict.shape)

(38636, 6)
```

→ Next, Binary Relevance method for other classifiers (SVC, multinomialNB, gausseanNB) is directly imported from the **Scikit-multilearn** library. Classifiers for **Classifier Chain** and **Label Powerset** are imported and tested.

```
#create and fit classifier
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.svm import SVC
classifier = BinaryRelevance(classifier = SVC(), require_dense = [False, True])
classifier.fit(X_train, Y_train)

BinaryRelevance(classifier=SVC(C=1.0, cache_size=200, class_weight=None,
                                coef0=0.0, decision_function_shape='ovr',
                                degree=3, gamma='auto_deprecated', kernel='rbf',
                                max_iter=-1, probability=False,
                                random_state=None, shrinking=True, tol=0.001,
                                verbose=False),
                require_dense=[False, True])

#predictions
predictions = classifier.predict(X_test)

#calculate scores
evaluate_score(Y_test, predictions)

Hamming_loss : 4.292197259895779
Accuracy : 88.38906719122063
Log_loss : 0.4614351050527784
```

- Then **Adaptation Algorithms** were used. To be precise, MLkNN will be imported from the scikit-multilearn library and BP-MLL will be implemented from scratch.
- Back Propagation MultiLabel Neural Networks (BP-MLL) can be implemented using the Sequential Model from Keras. Checkpointer is used to show the intermediate results from different epochs.
- The model architecture is as follows:

```
#define model architecture
model = Sequential()
model.add(Dense(4, activation='relu', input_dim = X_train.shape[1]))
model.add(Dropout(0.3))
model.add(Dense(6, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 4)	211956
dropout_1 (Dropout)	(None, 4)	0
dense_2 (Dense)	(None, 6)	30
Total params: 211,986		
Trainable params: 211,986		
Non-trainable params: 0		

- The **evaluate_score function** cannot be used to operate directly on the predictions of this model as it returns **probabilities in a range of values from 0 to 1**. The hamming_loss and accuracy_score cannot work on these values directly. Hence, predicted results were rounded.

3. Refinement

PROBLEM TRANSFORMATION METHODS:

- Our initial model was the Binary Relevance method using **MultinomialNB** classifier. It led to the following results:

```
Hamming_loss : 3.2796185387852415
Accuracy : 88.29764315567236
Log_loss : 1.9296193554647956
```

- We then tried other classifiers like **SVC** and **GaussianNB**:

Hamming_loss : 4.26702285033105
Accuracy : 88.276099788875
Log_loss : 0.4616701016298293

(SVC)

Hamming_loss : 20.746262225860658
Accuracy : 52.199577750010775
Log_loss : 1.4227365989183884

(GaussianNB)

→ Since MultinomialNB was giving the best results until now, **Classifier Chain** and **Label Powerset** was implemented using this classifier only.

Hamming_loss : 3.5647090927370133
Accuracy : 88.25886509543712
Log_loss : 1.506849253150214

(Classifier Chain with MultinomialNB)

Hamming_loss : 3.1726198170250046
Accuracy : 88.80606661209013
Log_loss : 1.4765486777963348

(Label Powerset with MultinomialNB)

ADAPTATION ALGORITHMS:

→ MLkNN was causing the kernel to go out of memory and hence I tried improving our **BP-MLL model**. The initial model gave the following results:

Log_loss : 0.36017768848519677
Hamming_loss : 13.960101684691285
Accuracy : 29.52302985910638

→ GridSearchCV was then applied for param grid having –
[nodes in the hidden layer = 16,32,64, learning rate = 0.001, 0.002, 0.003, epochs = 10,20,30]

```
print(grid_result)
```

```
GridSearchCV(cv=None, error_score='raise',  
             estimator=<keras.wrappers.scikit_learn.KerasClassifier object at 0x7ffc79278>,  
             fit_params=None, iid=True, n_jobs=1,  
             param_grid={'epochs': [10, 20, 30], 'nodes': [16, 32, 64], 'lr': [0.001, 0.002, 0.003]},  
             pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',  
             scoring=None, verbose=2)
```

```
print('Best estimator : {}'.format(grid.best_estimator_))  
print('Best score : {}'.format(grid.best_score_))  
print('Best params : {}'.format(grid.best_params_))
```

```
Best estimator : <keras.wrappers.scikit_learn.KerasClassifier object at 0x887d8e8d0>  
Best score : 0.9802662760148995  
Best params : {'epochs': 10, 'lr': 0.001, 'nodes': 16}
```

The results on predictions were:

Log_loss : 0.3504030505499869
Hamming_loss : 15.158630990851249
Accuracy : 21.612305571114653

4. Results

1. Model Evaluation and Validation

Our final model that we got from each of the 2 methods above is given as follows:

- **For the problem transformation method:** The label powerset method with MultinomialNB classifier.
- **For the adaptation algorithm method:** The BP-MLL model with params = {nodes in hidden layer = 16, learning rate = 0.001, epochs = 10, batch size = 64}

ROBUSTNESS AND OTHER FACTORS:

- Since we had around 46418 samples for training purposes only and another 23209 samples for testing purposes, involving largely varying datasets, hence we can expect the correct results for any data set given to this model.
- The BP-MLL model has also been done with training on large number of parameters varying from numbers of nodes in hidden layer to different learning rates and epoch sizes, along with **3-fold cross validation**. Training with GridsearchCV took approximately 15-20 hours and will definitely result in the best results only. The GridsearchCV results are shown below:

```
'nodes': 64}}, 'split0_test_score': array([0.98351968, 0.97725069, 0.95779745, 0.98009436, 0.96833193,
0.95088218, 0.97544109, 0.95139921, 0.97214503, 0.97382537,
0.95404899, 0.93013637, 0.96426032, 0.94584114, 0.90609449,
0.95165773, 0.91869709, 0.90305694, 0.96005946, 0.92302721,
0.89995476, 0.94920184, 0.91766303, 0.90758095, 0.94842629,
0.93239837, 0.92606476]), 'split1_test_score': array([0.9747948 , 0.97679829, 0.96361404, 0.9777031 , 0.972403
54,
0.96561753, 0.97608738, 0.96955988, 0.96348478, 0.97033542,
0.95740968, 0.94674594, 0.97621664, 0.94357914, 0.90674077,
0.95960706, 0.94286822, 0.92451367, 0.96975376, 0.9380857 ,
0.92179926, 0.94222193, 0.92515996, 0.92257481, 0.94144639,
0.91307439, 0.92386738]), 'split2_test_score': array([0.98248449, 0.97440538, 0.95934592, 0.97330662, 0.969040
85,
0.96464581, 0.9663909 , 0.95314116, 0.95643744, 0.9709152 ,
0.94525595, 0.92638314, 0.9473242 , 0.93181231, 0.92567218,
0.94234747, 0.92418563, 0.87609876, 0.96063857, 0.93265253,
0.90324457, 0.93368666, 0.89438987, 0.90059462, 0.94034385,
0.91171148, 0.88139866]), 'mean_test_score': array([0.98026628, 0.97615149, 0.96025249, 0.97703477, 0.9699254
6,
0.96038175, 0.97263992, 0.95803352, 0.96402258, 0.97169202,
0.95223836, 0.93442199, 0.96260072, 0.94041105, 0.91283554,
0.95120427, 0.92858374, 0.90122366, 0.96348399, 0.93125512,
0.90833297, 0.94170365, 0.91240467, 0.91025033, 0.94340558,
0.91906157, 0.91044422]), 'std_test_score': array([0.00389199, 0.00124839, 0.00245964, 0.00281108, 0.00177604,
0.00672902, 0.00442646, 0.00818146, 0.00642383, 0.00152699,
0.00512424, 0.00884817, 0.01185346, 0.00614976, 0.00908042,
0.00705345, 0.01034636, 0.01980766, 0.00443977, 0.00622657,
0.00961651, 0.0063446 , 0.01310052, 0.0091697 , 0.00357866,
0.00944709, 0.02055724]), 'rank_test_score': array([ 1,  3, 11,  2,  6, 10,  4, 12,  7,  5, 13, 18,  9, 17, 2
2, 14, 20,
27,  8, 19, 26, 16, 23, 25, 15, 21, 24]), dtype=int32), 'split0_train_score': array([0.98545807, 0.97676523, 0.
```

2. Justification

We can compare our results with the SVM model which we generated during the implementation section while we were trying out different classifiers with the Binary Relevance Method.

The SVM model had hamming loss of 4.267% while the log loss was 0.461. Comparing with our best model i.e –

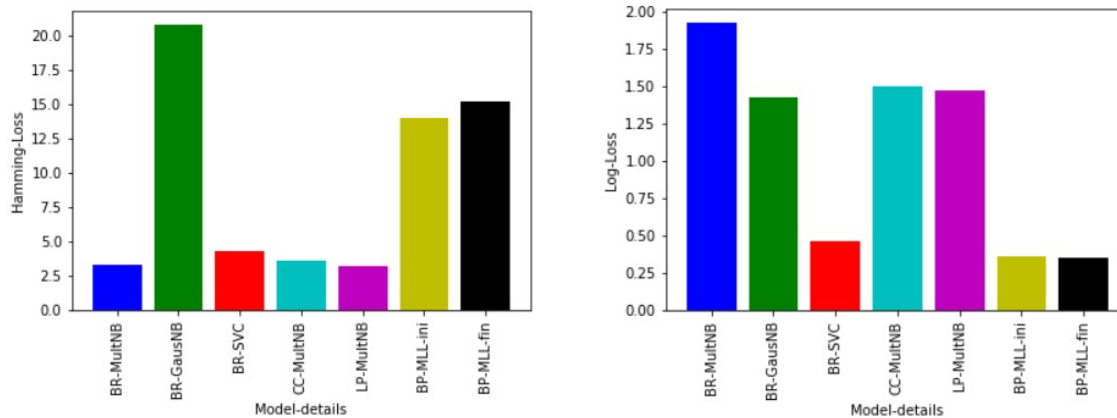
- MultinomialNB model in LP method (hamming loss of 3.17% and log loss of 1.47)
 - Hamming loss improved by 1.09%
 - Log loss increased by 1.009
- BP-MLL model with params as {nodes in hidden layer = 16, learning rate = 0.001, epochs = 10, batch size = 64} and (hamming loss of 15.15 and log loss of 0.35)
 - Hamming loss increased by 10.8%
 - Log loss decreased by 0.11

Thus, our model has some improvement over the SVM model.

5. Conclusion

1. Free Form Visualization

The hamming-loss and log-loss of different models used. We plotted in 2 scatter plots:



- Thus if we compare all the models on hamming-loss: The best model will be LP-MultiNB i.e. Label Powerset Model with MultinomialNB classifier
- If we compare all the models on log-loss: The best model will be BP-MLL model with params {nodes in hidden layer = 16, learning rate = 0.001, epochs = 10, batch size = 64}

2.Reflection

- This is the summary of the things we followed in this project:
- The first step involved collecting data and deciding what part of it is suitable for training : This step was extremely crucial since including only very small length comments would give poor results if the length was increased whereas including very long length comments would increase the number of words drastically, hence increasing the training time exponentially and causing system (jupyter kernel) to go out of memory and die eventually.
- The second major step was performing cleaning of data including punctuation removal, stop word removal, stemming and lemmatizing: This step was also crucial since the occurrence of similar origin words but having different spellings will intend to give similar classification, but computer cannot recognize this on its own. Hence, this step helped to a large extent in both removing and modifying existing words.
- The third step was choosing models to train on: Since I had a wide variety of models(3 for problem transformation) and classifiers(not bounded) along with number of adaptation models in BP-MLL, selecting which all models to train and test took lots of efforts.

- Finally comparing on the basis of different evaluation metrics: The two major evaluation metrics I planned to compare on were hamming-loss and log-loss. Hence the final model selection was done on the basis of the combination of both these losses.

3. Improvement

- Although we have tried quite several parameters in refining my model, there can exist a better model which gives greater accuracy.
- Yes. We were unable to find a clear implementation of the Adaboost.MH decision tree model which we had planned to use. The scikit-multilearn library doesn't even mention of such a model. Also, the research papers were a little vague regarding implementation details.

3. Future Scope

The current project predicts the type or toxicity in the comment. We are planning to add the following features in the future:

- Analyse which age group is being toxic towards a particular group or brand.
- Add feature to automatically sensitize words which are classified as toxic.
- Automatically send alerts to the concerned authority if threats are classified as severe.
- Build a feedback loop to further increase the efficiency of the model.
- Handle mistakes and short forms of words to get better accuracy of the result.

References

- Wikipedia : https://en.wikipedia.org/wiki/Multi-label_classification
- Kaggle challenge page for datasets and ideas : [https://www.kaggle.com/c/](https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge)
- jigsaw-toxic-comment-classification-challenge
- Conversation AI git page : <https://conversationalai.github.io/>
- Research Paper titled "Multi-label Classification: Problem Transformation methods in Tamil Phoneme classification" : [https://ac.els-cdn.com/](https://ac.els-cdn.com/S1877050917319440/1-s2.0-S1877050917319440-main.pdf?_tid=eced1a38-f8fa-11e7-b8ef-00000aabb0f27&acdnat=1515914406_0f244d3e6313bb049c435bf43504bd52)
- S1877050917319440/1-s2.0-S1877050917319440-main.pdf?_tid=eced1a38-f8fa-11e7-b8ef-00000aabb0f27&acdnat=1515914406_0f244d3e6313bb049c435bf43504bd52
- Research Paper titled "Benchmarking Multi-label Classification Algorithms" :
- http://ceur-ws.org/Vol-1751/AICS_2016_paper_33.pdf
- Problem Transformation Methods : [https://www.analyticsvidhya.com/blog/](https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/)
- 2017/08/introduction-to-multi-label-classification/
- Research Paper on BP-MLL : [http://citeseerx.ist.psu.edu/viewdoc/](http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.507.910&rep=rep1&type=pdf)
- download?doi=10.1.1.507.910&rep=rep1&type=pdf
- GridsearchCV on Sequential Models : [https://dzubo.github.io/machine-](https://dzubo.github.io/machine-learning/2017/05/25/increasing-model-accuracy-by-tuning-parameters.html)
- learning/2017/05/25/increasing-model-accuracy-by-tuning-parameters.html

- MI-knn - A Lazy Learning Approach to Multi-Label Learning : <https://cs.nju.edu.cn/zhoush/zhoush.files/publication/pr07.pdf>
- Average Precision score : http://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html