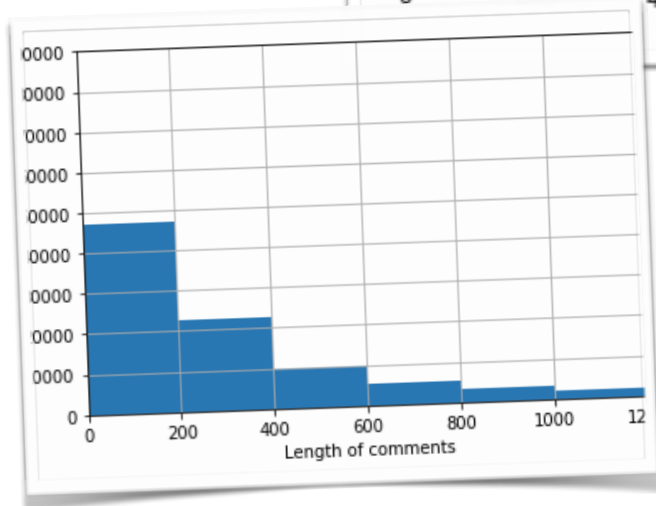
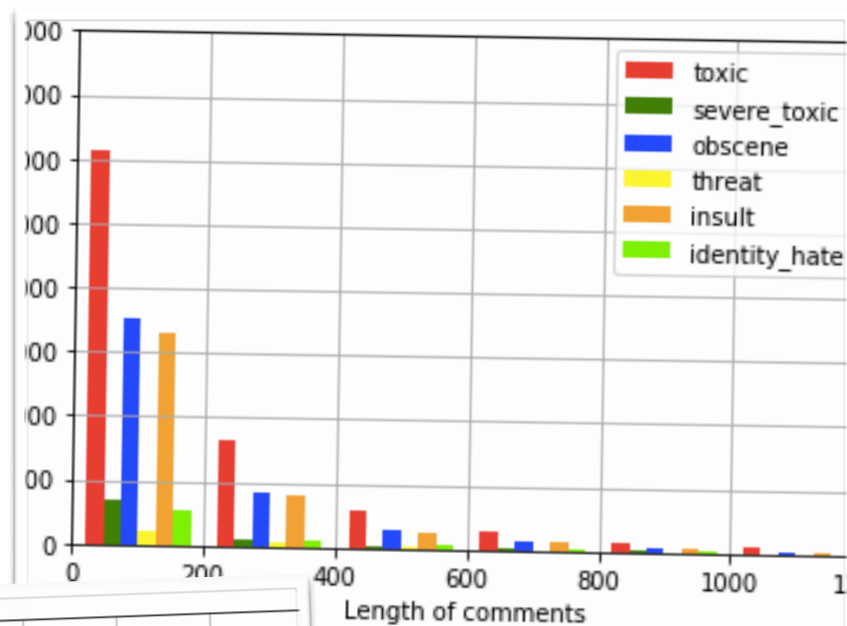


Toxic Comment Classification

Machine Learning Engineer Nanodegree

Capstone Project Report



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

1. Problem Definition

1. Project Overview

People express themselves freely and without reluctance at online platforms only when they feel comfortable. Any threat of abuse or harassment will make them leave the conversation and prohibit them from participating in any good conversation in future. It is hence, a vital requirement for any organisation or community to have an automated system which can identify such toxic comments and report/block the same immediately.

This problem falls under the category of Natural Language Processing where we try to identify the intention of the speaker, and act accordingly. Research on modelling a solution to this problem has already begun. I personally feel, it is important to handle any such nuisance and create a more user friendly experience with regard to online conversation for each one of us. Only then people will enjoy participating in discussions. The idea of my project and the dataset has been taken from [kaggle](#). Under the name of toxic comment classification challenge, it aims to identify and classify online toxic comments. It is being conducted as research by the [Conversation AI](#) team, which is an initiative by Jigsaw and Google.

2. Problem Statement

Given a group of sentences or paragraphs, which was used as a comment by a user in an online platform, our task is to classify it to belong to one or more of the following categories - toxic, severe-toxic, obscene, threat, insult or identity-hate with approximate probabilities or discrete values (0/1). This is clearly a [Multi-label classification](#) problem.

A multi-label classification problem differs from a multi-class classification problem (in which each sample can only be assigned to one of the many-labels). Hence in our problem each comment can belong to more than one label for eg, a comment can be obscene, toxic and insult, all at the same time.

There can be multiple ways to approach this classification problem such as using-

- ❖ Problem transformation methods like binary relevance method, label power set, classifier chain and random k-label sets (RAKEL) algorithm
- ❖ Adapted algorithms like the AdaBoost MH, AdaBoost MR, k-nearest neighbours, decision trees and BP-MLL neural networks.

I tried using majority of the algorithms listed above. But, some algorithms were found to be unsuitable, because of the exceedingly high amount of training time involved on a relatively big dataset. The details of this analysis will be provided later.

Out of the total dataset used, 70% was used for training and 30% for testing. Each testing dataset was labelled and hence for each algorithm using the predictions and labels, calculation of metrics 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 based metrics** include one-error, average precision, etc. These are calculated separately for each of the labels, and then averaged for all without taking into account any relation between the labels.

$AP = \sum_n (R_n - R_{n-1})P_n$ Average Precision (AP) : AP summarises a precision-recall curve as the weighted mean of precisions achieved at each threshold, with the increase in recall from the previous threshold used as the weight: where P_n and R_n are the precision and recall at the n th threshold. (reference).

$one-error(f) = \frac{1}{p} \sum_{i=1}^p \mathbb{I}[\arg \max_{y \in \mathcal{Y}} f(\mathbf{x}_i, y)] \notin Y_i$ The one-error evaluates the fraction of examples whose top-ranked label is not in the relevant label set.

- **Example based metrics** include accuracy, hamming loss, etc. These are calculated for each example and then averaged across the test set. Information about metrics has been obtained from this [research paper](#). Let-

Y_i - predicted subset of labels for the i^{th} instance

D - multi label dataset

Z_i - true subset of labels for the i^{th} instance

L - full set of labels used in dataset

$$Accuracy = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|}$$

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

$$hamming-loss = \frac{1}{|D|} \sum_{i=1}^{|D|} \frac{|Y_i \Delta Z_i|}{|L|}$$

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

On having a glance 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. Hence we have skewed data, and accuracy as metric will not give suitable results. Therefore, I will be using *hamming-loss* as the evaluation metric.

Also, as suggested through reviews on this project's proposal, *Log-loss* will be another evaluation metric used. Log loss measures the performance of a classification model where the prediction input is a probability value between 0 and 1. This will specifically suit for evaluation of our neural network models, where the output will be obtained in the range from 0 to 1.

Label based metrics were explained along with example based metrics as both of these play an important role in multi-label classification. My analysis will involve only example based metrics for comparison between models.

2. Analysis

1. Data Exploration

The dataset consists of the following fields-

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

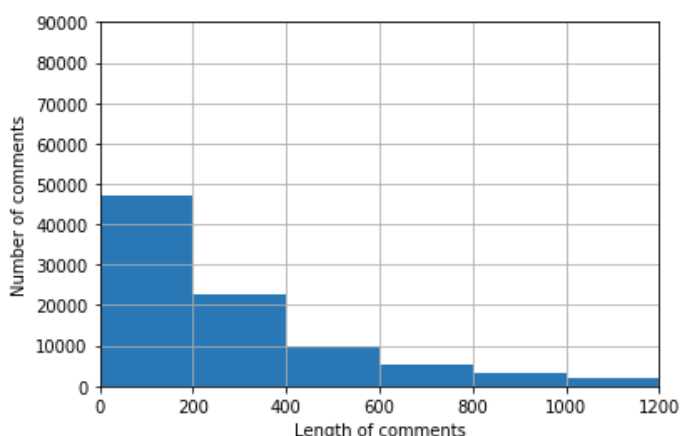
We have a total of 95981 samples of comments and labelled data, which can be loaded from train.csv file. The first 5 samples are as follows:

```
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
```

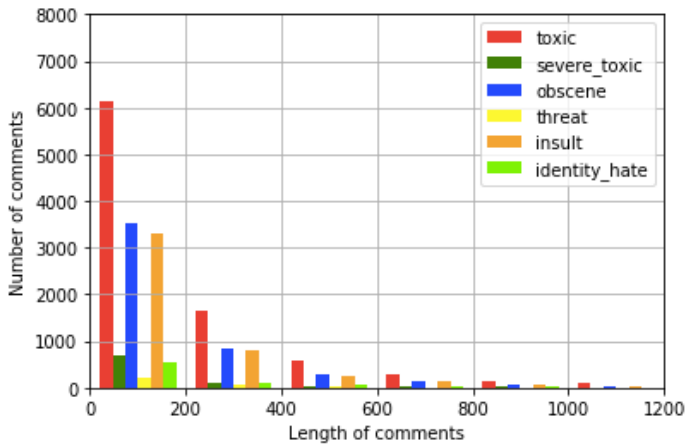
	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	1	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

Another important aspect of the dataset it noticing the frequency of 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 90,000 samples). Overall 9790 samples are those which have at least one label and 5957 samples have 2 or more labels.

2. Exploratory Visualisation



From the first visualisation we can observe that comments have varying lengths from within 200 upto 1200. The majority of comments have length upto 200, and as we move towards greater lengths, the number of comments keeps 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 visualisation, 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.

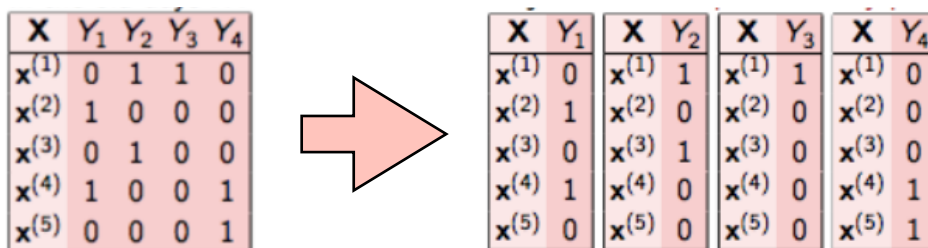
Based on these plots, taking comments having lengths upto 400 for training is a good estimation of our data and can be expected to give acceptable results on testing later. Hence before preprocessing, we will be removing all comments with length more than 400 which will serve as our threshold.

3. Algorithms & Techniques

As discussed in the Problem Statement section, any multi-label classification problem can be solved using either Problem Transformation 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.



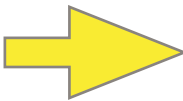
❖ **Classifier Chain Method** : In this method, the first classifier is trained on input data and then each next classifier is trained on the input space and previous classifier, and so on. Hence this method takes into account some interdependence between labels and input data. Some classifiers may show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



❖ **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 or 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.

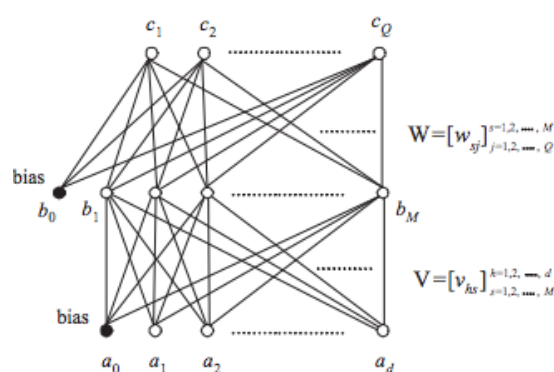
ADAPTATION ALGORITHMS :

❖ **MLKNN** : This is the adapted multi label version of K-nearest neighbours. Similar to this classification algorithm is the BRkNNaClassifier and BRkNNbClassifier which are based on K-Nearest Neighbours 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 ([reference](#)).

Since this is somewhat similar to the page categorisation 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. I will therefore try training it on small valued neighbours (k=2, etc)

❖ **BP-MLL Neural Networks** : Back propagation Multi-label Neural Networks is an architecture that aims at minimising 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 I said in my project proposal, I will be using a 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 better than this model.

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

3. Methodology

1. Data Preprocessing

Preprocessing involved the following steps:

❖ **Preparing a string containing all punctuations to be removed :**

1. The string library contains punctuation characters. This is imported and all numbers are appended to this string.
2. Also, we can notice that 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/didnt, the character ' represented as \' in escape sequence notation is replaced by empty character in the punctuation string.
3. make_trans(intab, outtab) function is also applied. It returns a translation table that maps each character in the intab into the character at the same position in the outtab string i.e. it replaces every character in the removal list with a space.

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

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

❖ **Stemming and Lemmatising :**

1. Stemming is the process of converting inflected/derived words to their word stem or the root form. Basically, a large number of similar origin words are converted to the same word.E.g. words like "stems", "stemmer", "stemming", "stemmed" as based on "stem". This helps in achieving the training process with a better accuracy.
2. Lemmatizing is the process of grouping together the inflected forms of a word so they can be analysed 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 neighbouring 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 Vectoriser :**

1. Count Vectoriser is used for converting 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.

2. It accepts stop words, convert to lowercase(set as true), 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

- ❖ First, we will be defining **function evaluate_score** for printing all evaluation metrics : Some implementations return a sparse matrix for the predictions, while others return a dense matrix. Hence, a try except block will be 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))
```

- ❖ Next, we will start implementing various **problem transformation methods**. These include the Binary Relevance, Label Powerset and Classifier Chain methods. Implementations of these methods is available in the scikit-multilearn library.
- ❖ I will be implementing the **Binary Relevance** method from scratch. It does not take into account 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)
```

```
(23209, 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 also imported and tested. The code for each of these models is similar to each other and appears as the following code :

```
#create and fit classifier
classifier = BinaryRelevance(classifier = MultinomialNB(), require_dense = [False, True])
classifier.fit(X_train, Y_train)
```

```
BinaryRelevance(classifier=MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True),
                 require_dense=[False, True])
```

```
# predictions
predictions = classifier.predict(X_test)
```

```
#calculate scores
evaluate_score(Y_test,predictions)
```

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

- ❖ Finally I will be working with **Adaptation Algorithms**. 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 I will be using 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 because 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, we will round the predicted results. However, log loss can compute loss directly without modifying the values.

3. Refinement

PROBLEM TRANSFORMATION METHODS :

◆ My initial model was the Binary Relevance method using **MultinomialNB** classifier. It lead to the following results:

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

◆ I then tried using 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 my **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

Thus our final model obtained from each of the 2 methods above is given as follows:

- ✦ **For the problem transformation method** : The label powerset method with MultinomialNB classifier (It had a hamming loss of 3.17 and log loss of 1.47)
- ✦ **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 the model has been selected after heavy process of testing and training and can be expected to give acceptable results to any given data.

The choice of selection of 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 results from GridsearchCV can be shown as under :

```
'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

I had discussed that since this dataset has not been worked on before, and also with this large size a dataset SVMs are not expected to give the best results. Thus, I do not have any benchmark model available. Nevertheless, we can still 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 -

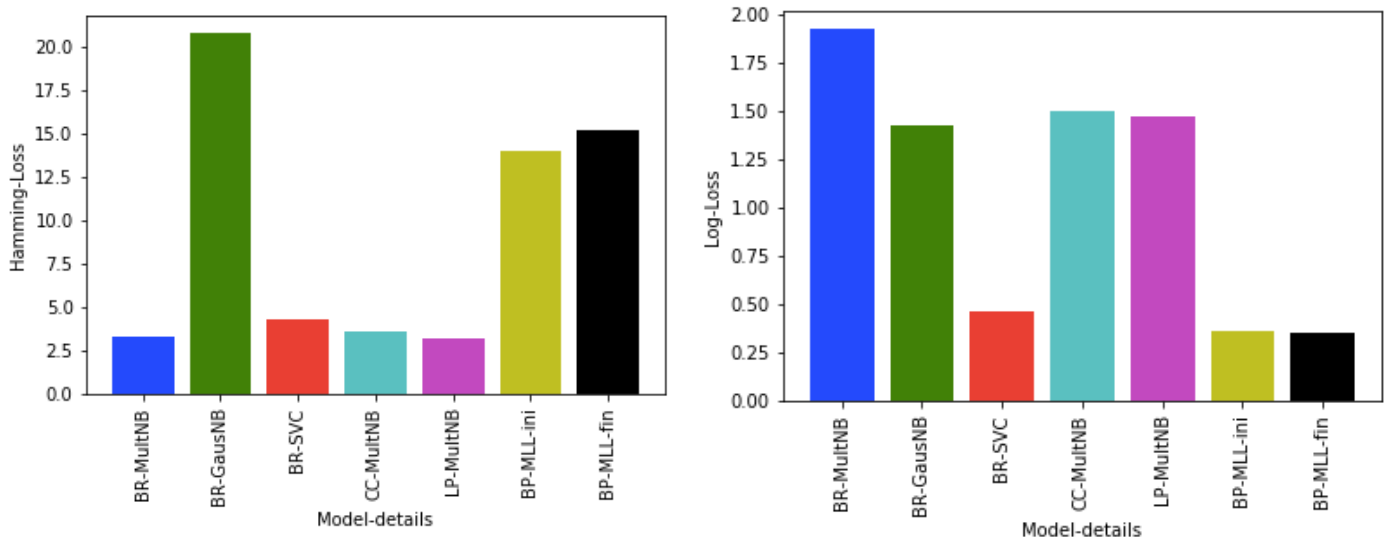
- ◆ MultinomialNB model in LP method (hamming loss of 3.17% and log loss of 1.47)
 - Hamming loss improved by 1.09%
 - Log loss increased by 1.009
- ◆ BP-MLL model with params as {nodes in hidden layer = 16, learning rate = 0.001, epochs = 10, batch size = 64} and (hamming loss of 15.15 and log loss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

Thus we can say our model has some improvement over the SVM model.

5. Conclusion

1. Free Form Visualisation

The hamming-loss and log-loss of different models used, can be plotted in 2 scatter plots as below :



◆ Thus if 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

As a summary to the whole process I 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 recognise 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.

2 Particular Aspects I found interesting/difficult were as follows :

1. **Kernel died because of memory out error** : At every stage of the process, this error appeared again and again. This was primarily because I was dealing with a pretty big dataset and any process which wanted more training or space caused an error. It forced me to reduce and alter my sample words as effectively as I can and chose only the most time efficient algorithms.
2. **Research and explore more** : Since my topic of project involving multi-label classification was not already taught in the course, I had to go through a number of different research papers and resources to gather necessary information. It was time consuming, but finally it increased my knowledge of the subject and made me better.

3. Improvement

- ✦ Although I have tried quite a number of parameters in refining my model, there can definitely exist a better model which gives greater accuracy.
- ✦ Yes. I was unable to find a clear implementation of the Adaboost.MH decision tree model which I 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.
- ✦ I believe my model gives satisfiable results on any given data. But, further research may yield an even better model than this new benchmark model.

References

3. Wikipedia : https://en.wikipedia.org/wiki/Multi-label_classification
4. Kaggle challenge page for datasets and ideas : <https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge>
5. Conversation AI git page : <https://conversationalai.github.io/>
6. Research Paper titled "Multi-label Classification: Problem Transformation methods in Tamil Phoneme classification" : https://ac.els-cdn.com/S1877050917319440/1-s2.0-S1877050917319440-main.pdf?_tid=eced1a38-f8fa-11e7-b8ef-00000aabb0f27&acdnat=1515914406_0f244d3e6313bb049c435bf43504bd52
7. Research Paper titled "Benchmarking Multi-label Classification Algorithms" : http://ceur-ws.org/Vol-1751/AICS_2016_paper_33.pdf
8. Problem Transformation Methods : <https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/>
9. Research Paper on BP-MLL : <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.507.910&rep=rep1&type=pdf>
10. GridsearchCV on Sequential Models : <https://dzubo.github.io/machine-learning/2017/05/25/increasing-model-accuracy-by-tuning-parameters.html>
11. ML-knn - A Lazy Learning Approach to Multi-Label Learning : <https://cs.nju.edu.cn/zhoush/zhoush.files/publication/pr07.pdf>
12. Average Precision score : http://scikit-learn.org/stable/modules/generated/sklearn.metrics.average_precision_score.html