CS6375.003 - MACHINE LEARNING PROJECT REPORT

PROJECT TITLE:

TOXIC COMMENT CLASSIFICATION

PROJECT MEMBERS:

Asla Aboo (axa174032)

Dafne Navita Maria Joseph (dxm172530)

Nikita Sah (nxs170831)

Ramya Ramineni (rsr170230)

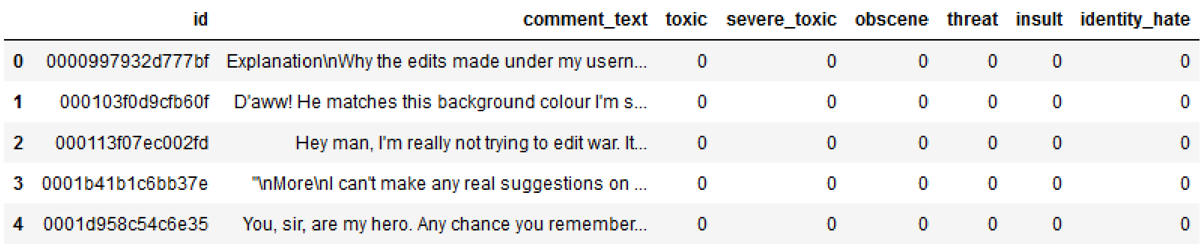
1. Introduction

Internet is designed for people of all race and ages. But with the increase in usage of internet and social networking sites it is the need of the hour to keep a track on the type of content which is being posted on different platforms. It is beyond the reach of any human beings in the organization to keep a check on all the content which is being circulated on web. Machine learning comes to our rescue and makes our life simple. Humans can play the role of training machines so that they can distinguish between toxic and non-toxic content. This concept is called fairness in machine learning and it is one of the most in demand use case in the euphoria of machine learning.

In this project we work on Toxic Comment Classification dataset from Kaggle. We focus on making a probabilistic model where each toxic comment is assigned a probability for each of the six classes mentioned in the dataset. This is how we can filter out toxic comments from the clean comments.

2. Dataset Description

We are using the dataset from kaggle competition. The training dataset is a multi-class target dataset with two training instances and six target classes.



	id	comment_text	toxic	severe_toxic	obscene	threat	insult	identity_hate
0	0000997932d777bf	Explanation\nWhy the edits made under my usern...	0	0	0	0	0	0
1	000103f0d9cfb60f	D'aww! He matches this background colour I'm s...	0	0	0	0	0	0
2	000113f07ec002fd	Hey man, I'm really not trying to edit war. It...	0	0	0	0	0	0
3	0001b41b1c6bb37e	"\nMore\nI can't make any real suggestions on ...	0	0	0	0	0	0
4	0001d958c54c6e35	You, sir, are my hero. Any chance you remember...	0	0	0	0	0	0

Training Instance Descriptions

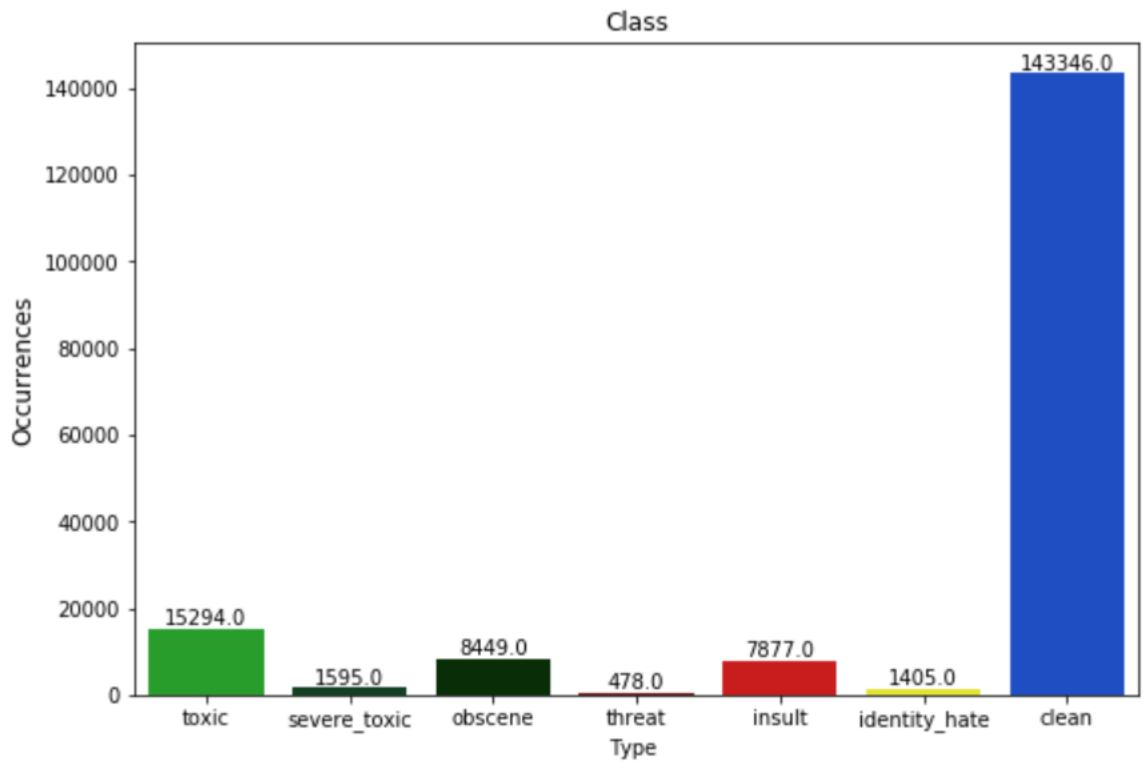
Attribute	**Value**
Comment_text	Textual data
Id	A unique attribute to identify text

Test Instance Descriptions

Attribute	**Value**
Comment_text	Textual data
id	A unique attribute to identify text

The target values for the test dataset is the probability of the six classes predicted by the model for each comment in the test dataset.

Class – wise graphical representation of the frequency of comments:



3. Preprocessing

StopWords removal

We used the stop words dictionary file imported from NLTK corpus library which had the list of all stop words which don't really contribute much towards prediction of comments. We then defined a function to remove all stop words from the toxic comments and return other words. Stopwords like [‘a’, ‘an’, ‘the’] were eventually removed from the comments.

Tokenizer

We also used the inbuilt tokenizer to set the maximum features, maximum number of words in a sentence and text to sequence conversion.
We also removed the NaN values like special characters by using ‘re’ package .

Vectorizer

We also used the TF-IDF vectorizer to translate words into a matrix of vectors for comparisons to be easier.

4. Models and Techniques

NB-SVM

This as an ensemble technique for model building. In this technique we use Logistic Regression on top of native Naïve Bayes so improve the overall accuracy. This is a widely used method for text classification and sentimental analysis. We used this classifier from package sklearn.linear_model.

The hyper parameters tuned are:

C = 5   #Inverse of regularization strength

Gradient Boosting

This is another ensemble technique which we tried for model building. We used this classifier from sklearn.ensemble.GradientBoostingClassifier.

It allows for optimization of differential loss functions by adding weak learners in a forward stage wise fashion using gradient descent approach. There are other variants of gradient boosting algorithms like stochastic and penalized gradient boosting.

Following are the hyperparameters chosen for this classifier on the toxic comments data set:

n_estimators =200 #number of tress in forest

max_depth=20 #maximum depth of the tree

min_samples_leaf=5 #minimum number of samples required to be at a leaf node

min_impurity_decrease=3 #node will be split if this split induces a decrease of the impurity greater than or equal to this value

Random Forest

Random forest classifier is an ensemble classifier which fits data to many sets of decision tree models and gives a highly efficient classifier. This classifier avoids overfitting which is a usual scenario in decision tree classifier. This is commonly used for feature engineering. This efficiently works on classification and regression data sets. We used this classifier from sklearn.ensemble.RandomForestClassifier

Following are the hyperparameters chosen for this classifier on the toxic comments data set:

criterion='entropy' #the quality of split is measured using information gain

min_samples_leaf=8 #minimum number of samples required to be in a leaf node is 8

min_samples_split=2 #minimum number of samples required to split a node is 2

max_depth=15 #maximum depth of decision tree is 15

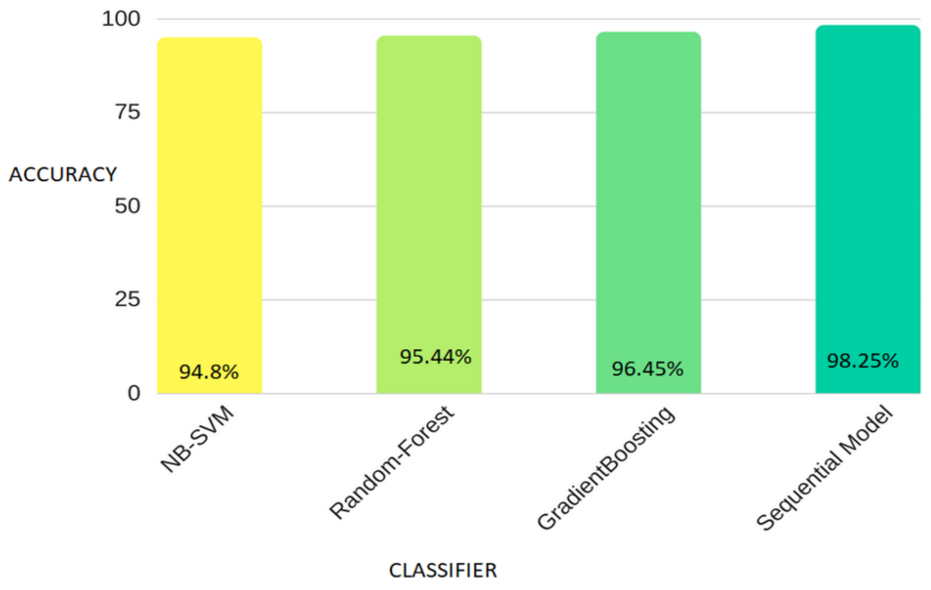
Keras Sequential Model

This is an inbuilt model which comes together with Keras (with TensorFlow as backend), one of the most well-known packages for Machine learning. The sequential model works on deep learning by constructing a deep neural network which consists of a stack of layers. We thought of exploring this model by adding multiple layers with each layer tuned with a specific set of parameters.
We noticed that having a lot of layers actually improved accuracy.
Dense layer takes the input dimensions to specify the shape. Layers like Conv1D is more suitable for text processing since it takes the kernel size and also, we were able to set the activation function.
Dropout layer was important since it keeps a check on overfitting data by dropping some unwanted features.
This model was relatively simple to build since we just had to add the layers that we wanted and finally compile it (i.e run it). While compiling, we noticed that optimizer parameter played a huge role with determining the accuracy of the model that we had built. ‘Adam’ and ‘rmsprop’ optimizers performed much better than ‘sgd’ optimizer. ‘rmsprop’ in particular performed really well on the multi-class datasets. After compiling the build, we just had to fit it into the dataset.
The number of epochs refer to how many times the model runs through the data set for one full cycle and it played a significant role to prevent overfitting and also improve the accuracy on trained model.
with determining the accuracy of the model that we had built. ‘Adam’ and ‘rmsprop’ optimizers performed much better than ‘sgd’ optimizer. ‘rmsprop’ in particular performed really well on the multi-class datasets. After compiling the build, we just had to fit it into the dataset.
The number of epochs refer to how many times the model runs through the data set for one full cycle and it played a significant role to prevent overfitting and also improve the accuracy on trained model.

5. Results

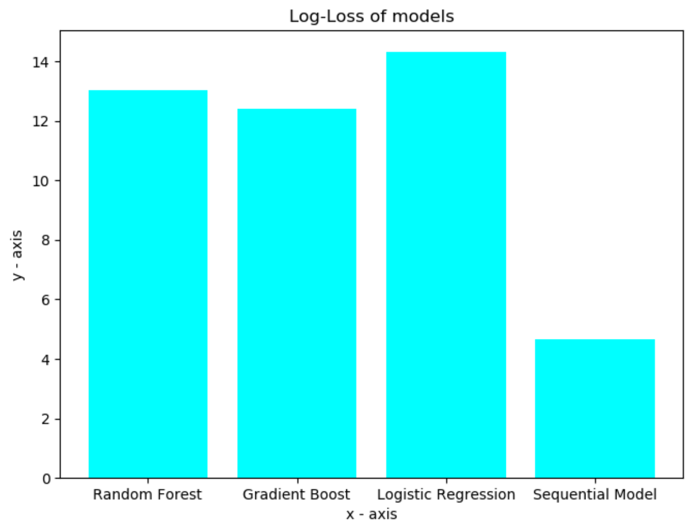
We used ensemble methods and deep learning technique for our model building. After various rounds of trail and errors our results are as follows.

Accuracy of the models



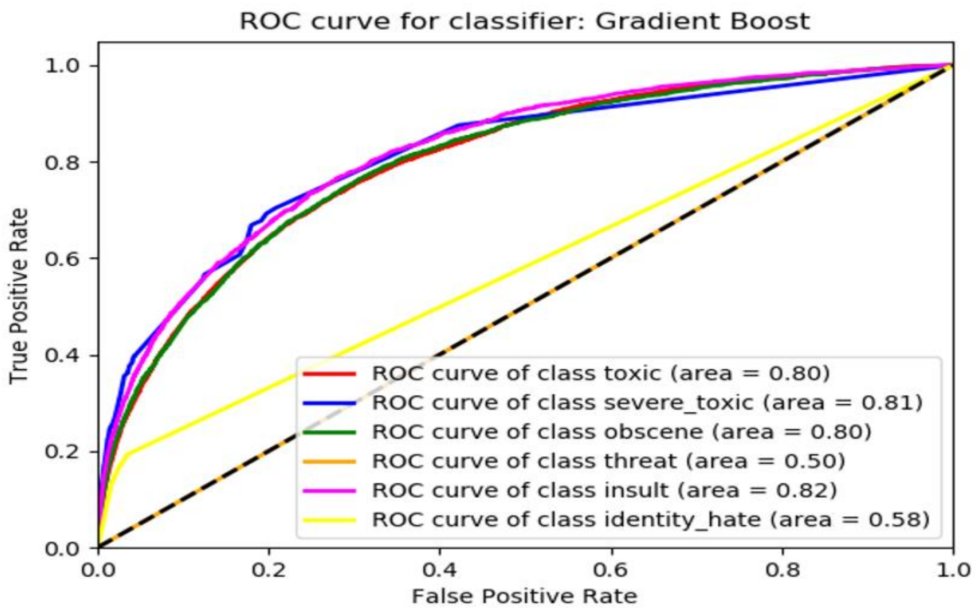
After trying all the four models the highest accuracy was obtained with Sequential model. Hence, we used it as our final model.

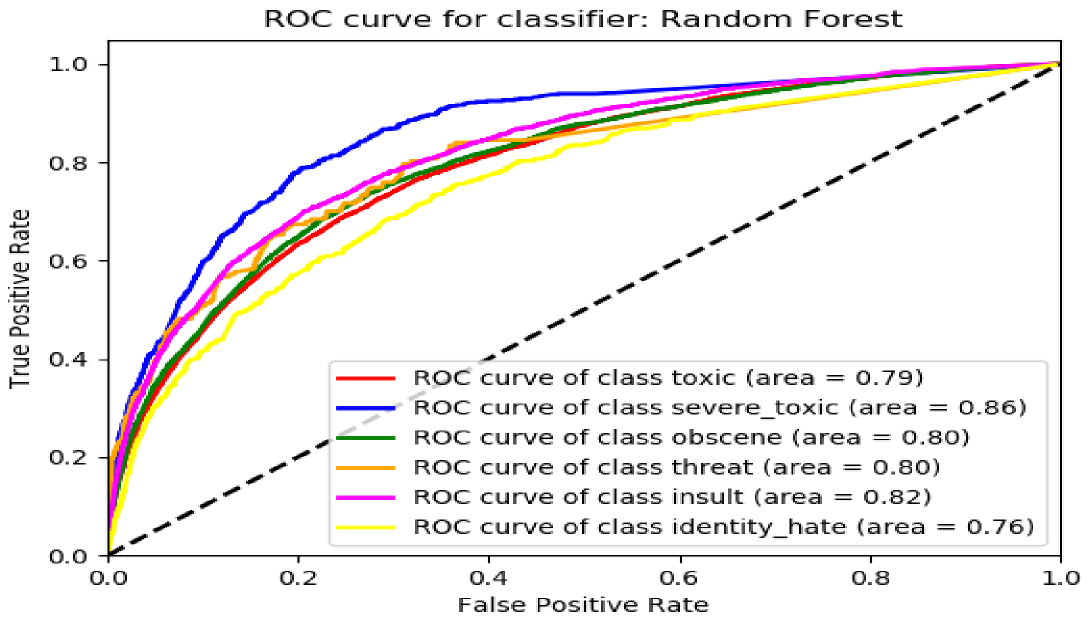
Log-Loss of the Models

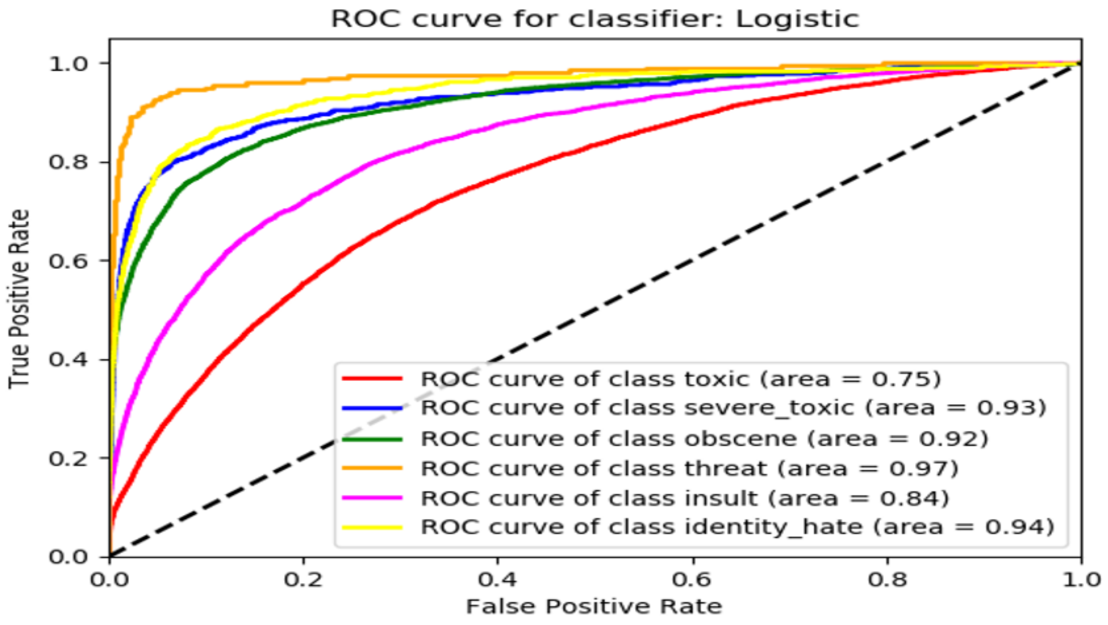


After trying all the four models the least Log-Loss value, 4.67% was obtained with Sequential model. Hence, we used it as our final model.

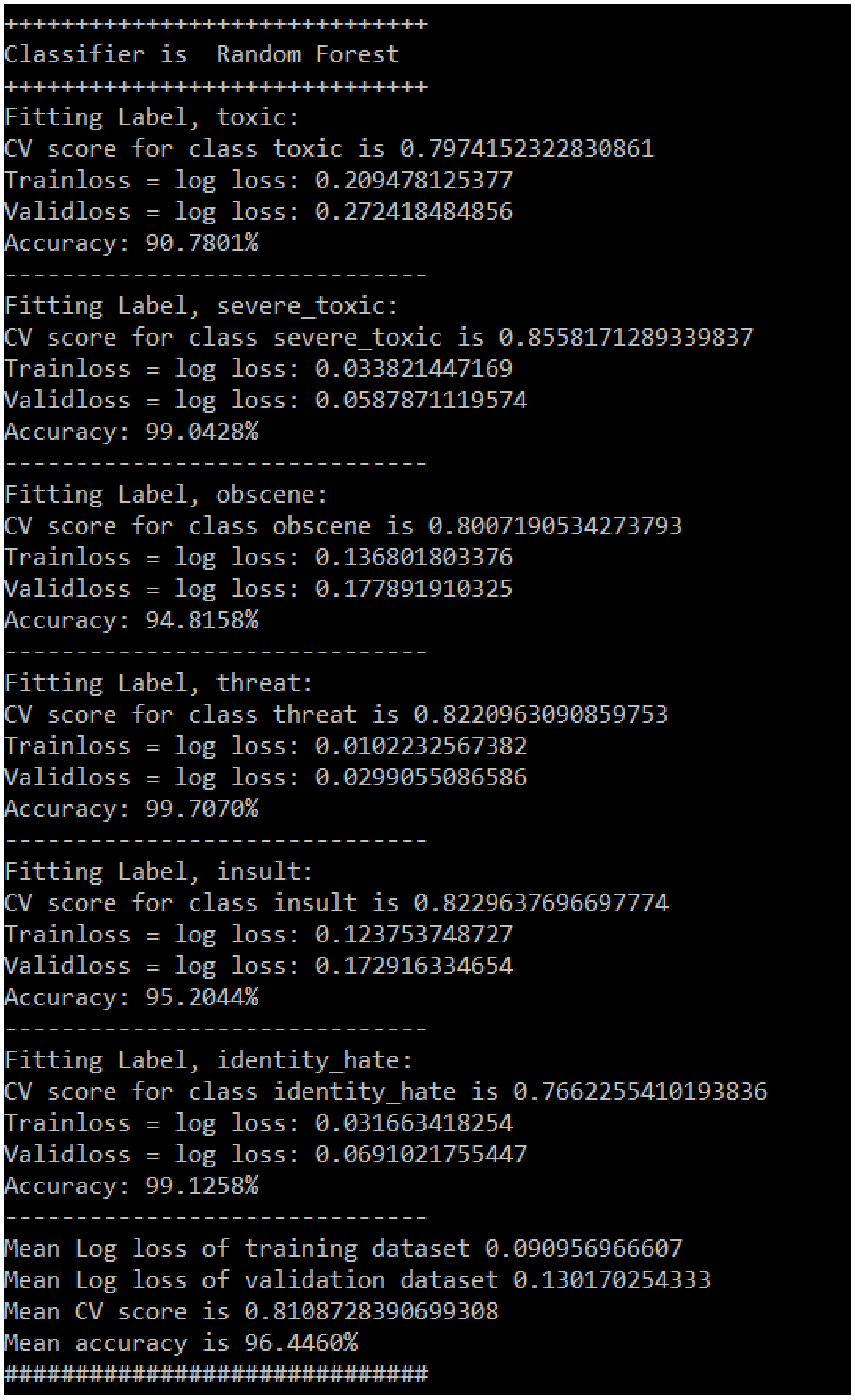
ROC-Curves for the models



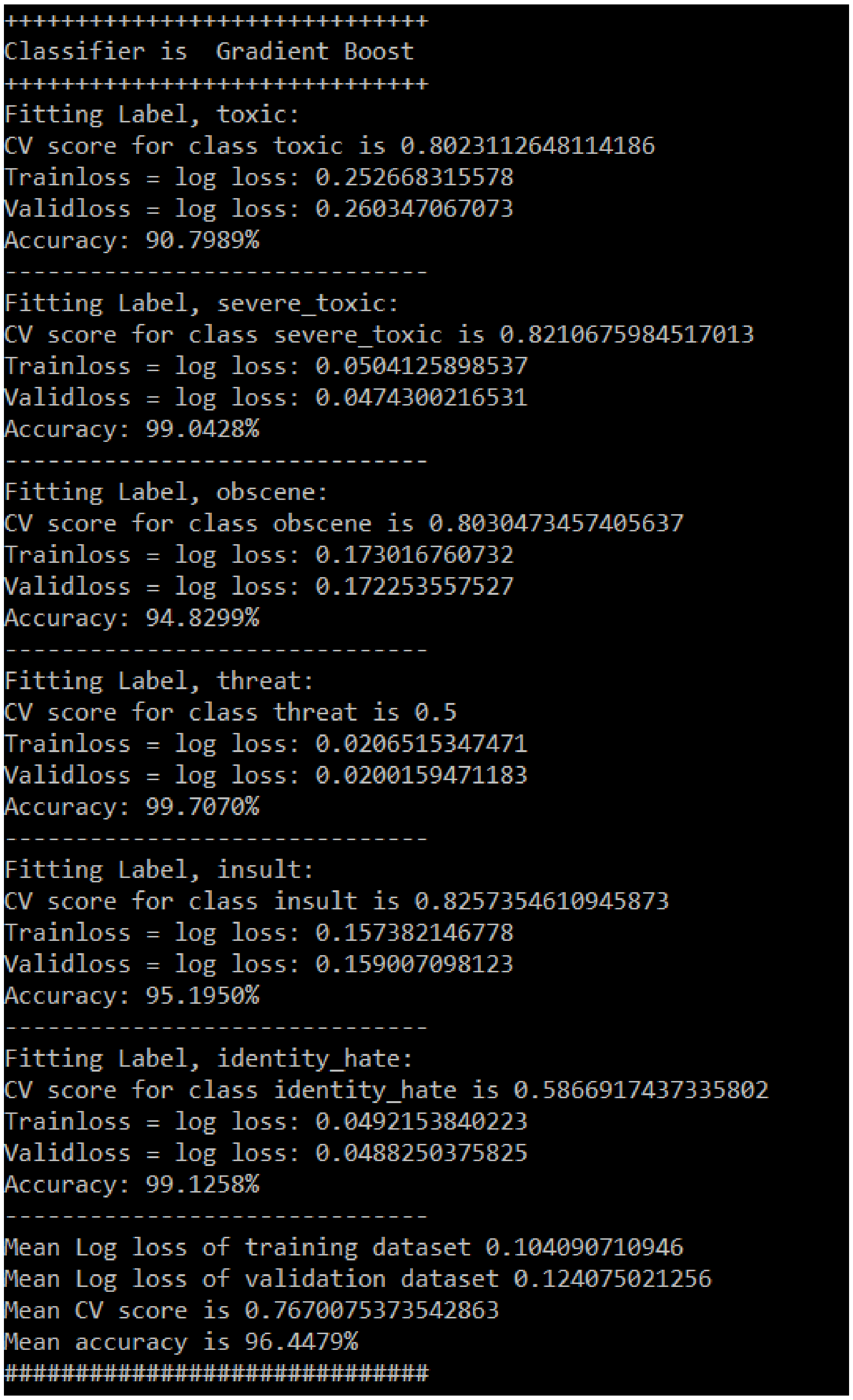

ROC curve for classifier: Random Forest
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class toxic (area = 0.79)
ROC curve of class severe_toxic (area = 0.86)
ROC curve of class obscene (area = 0.80)
ROC curve of class threat (area = 0.80)
ROC curve of class insult (area = 0.82)
ROC curve of class identity_hate (area = 0.76)


ROC curve for classifier: Logistic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class toxic (area = 0.75)
ROC curve of class severe_toxic (area = 0.93)
ROC curve of class obscene (area = 0.92)
ROC curve of class threat (area = 0.97)
ROC curve of class insult (area = 0.84)
ROC curve of class identity_hate (area = 0.94)

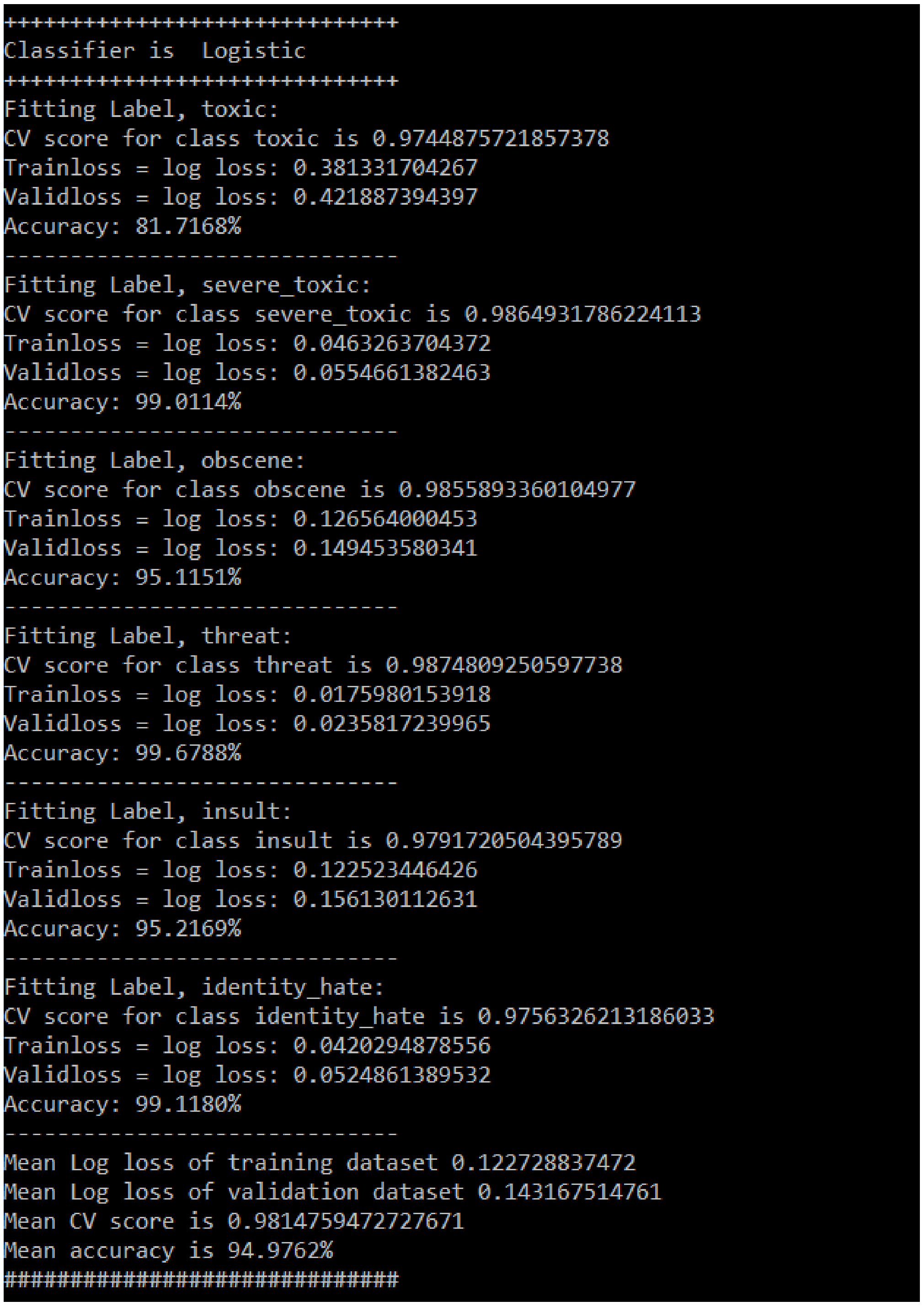
Results of the models



```
++++++++++++++++++++++++++++++
Classifier is  Random Forest
++++++++++++++++++++++++++++++
Fitting Label, toxic:
CV score for class toxic is 0.7974152322830861
Trainloss = log loss: 0.209478125377
Validloss = log loss: 0.272418484856
Accuracy: 90.7801%
------------------------------
Fitting Label, severe_toxic:
CV score for class severe_toxic is 0.8558171289339837
Trainloss = log loss: 0.033821447169
Validloss = log loss: 0.0587871119574
Accuracy: 99.0428%
------------------------------
Fitting Label, obscene:
CV score for class obscene is 0.8007190534273793
Trainloss = log loss: 0.136801803376
Validloss = log loss: 0.177891910325
Accuracy: 94.8158%
------------------------------
Fitting Label, threat:
CV score for class threat is 0.8220963090859753
Trainloss = log loss: 0.0102232567382
Validloss = log loss: 0.0299055086586
Accuracy: 99.7070%
------------------------------
Fitting Label, insult:
CV score for class insult is 0.8229637696697774
Trainloss = log loss: 0.123753748727
Validloss = log loss: 0.172916334654
Accuracy: 95.2044%
------------------------------
Fitting Label, identity_hate:
CV score for class identity_hate is 0.7662255410193836
Trainloss = log loss: 0.031663418254
Validloss = log loss: 0.0691021755447
Accuracy: 99.1258%
------------------------------
Mean Log loss of training dataset 0.090956966607
Mean Log loss of validation dataset 0.130170254333
Mean CV score is 0.8108728390699308
Mean accuracy is 96.4460%
##############################
```

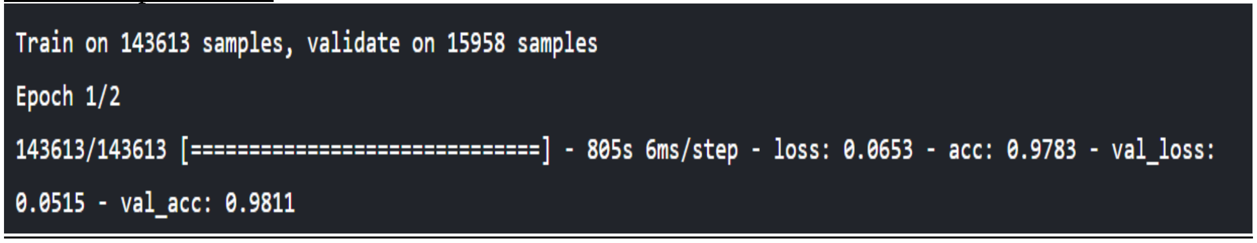


```
++++++++++++++++++++++++++++++
Classifier is  Gradient Boost
++++++++++++++++++++++++++++++
Fitting Label, toxic:
CV score for class toxic is 0.8023112648114186
Trainloss = log loss: 0.252668315578
Validloss = log loss: 0.260347067073
Accuracy: 90.7989%
------------------------------
Fitting Label, severe_toxic:
CV score for class severe_toxic is 0.8210675984517013
Trainloss = log loss: 0.0504125898537
Validloss = log loss: 0.0474300216531
Accuracy: 99.0428%
------------------------------
Fitting Label, obscene:
CV score for class obscene is 0.8030473457405637
Trainloss = log loss: 0.173016760732
Validloss = log loss: 0.172253557527
Accuracy: 94.8299%
------------------------------
Fitting Label, threat:
CV score for class threat is 0.5
Trainloss = log loss: 0.0206515347471
Validloss = log loss: 0.0200159471183
Accuracy: 99.7070%
------------------------------
Fitting Label, insult:
CV score for class insult is 0.8257354610945873
Trainloss = log loss: 0.157382146778
Validloss = log loss: 0.159007098123
Accuracy: 95.1950%
------------------------------
Fitting Label, identity_hate:
CV score for class identity_hate is 0.5866917437335802
Trainloss = log loss: 0.0492153840223
Validloss = log loss: 0.0488250375825
Accuracy: 99.1258%
------------------------------
Mean Log loss of training dataset 0.104090710946
Mean Log loss of validation dataset 0.124075021256
Mean CV score is 0.7670075373542863
Mean accuracy is 96.4479%
##############################
```



```
++++++++++++++++++++++++++++++
Classifier is  Logistic
++++++++++++++++++++++++++++++
Fitting Label, toxic:
CV score for class toxic is 0.9744875721857378
Trainloss = log loss: 0.381331704267
Validloss = log loss: 0.421887394397
Accuracy: 81.7168%
------------------------------
Fitting Label, severe_toxic:
CV score for class severe_toxic is 0.9864931786224113
Trainloss = log loss: 0.0463263704372
Validloss = log loss: 0.0554661382463
Accuracy: 99.0114%
------------------------------
Fitting Label, obscene:
CV score for class obscene is 0.9855893360104977
Trainloss = log loss: 0.126564000453
Validloss = log loss: 0.149453580341
Accuracy: 95.1151%
------------------------------
Fitting Label, threat:
CV score for class threat is 0.9874809250597738
Trainloss = log loss: 0.0175980153918
Validloss = log loss: 0.0235817239965
Accuracy: 99.6788%
------------------------------
Fitting Label, insult:
CV score for class insult is 0.9791720504395789
Trainloss = log loss: 0.122523446426
Validloss = log loss: 0.156130112631
Accuracy: 95.2169%
------------------------------
Fitting Label, identity_hate:
CV score for class identity_hate is 0.9756326213186033
Trainloss = log loss: 0.0420294878556
Validloss = log loss: 0.0524861389532
Accuracy: 99.1180%
------------------------------
Mean Log loss of training dataset 0.122728837472
Mean Log loss of validation dataset 0.143167514761
Mean CV score is 0.9814759472727671
Mean accuracy is 94.9762%
##############################
```

Keras sample result :



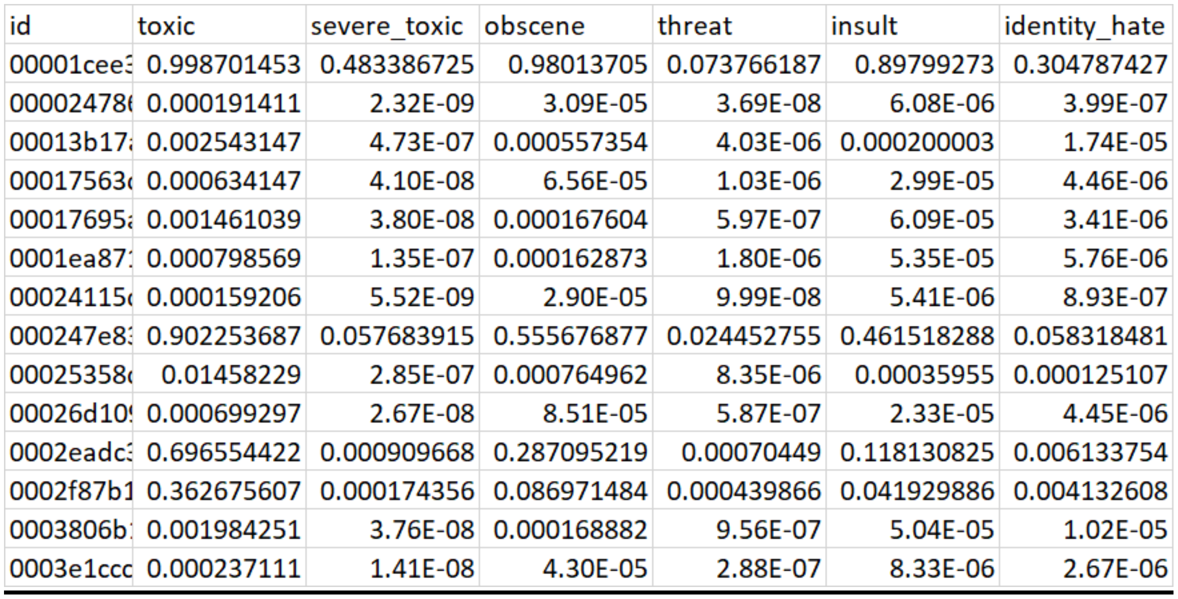
```
Train on 143613 samples, validate on 15958 samples

Epoch 1/2

143613/143613 [==============================] - 805s 6ms/step - loss: 0.0653 - acc: 0.9783 - val_loss:

0.0515 - val_acc: 0.9811
```

<u>Sample Results :</u> The below is a sample of the result file generated by the final model :



id	toxic	severe_toxic	obscene	threat	insult	identity_hate
00001cee	0.998701453	0.483386725	0.98013705	0.073766187	0.89799273	0.304787427
00002478	0.000191411	2.32E-09	3.09E-05	3.69E-08	6.08E-06	3.99E-07
00013b17	0.002543147	4.73E-07	0.000557354	4.03E-06	0.000200003	1.74E-05
00017563	0.000634147	4.10E-08	6.56E-05	1.03E-06	2.99E-05	4.46E-06
00017695	0.001461039	3.80E-08	0.000167604	5.97E-07	6.09E-05	3.41E-06
0001ea87	0.000798569	1.35E-07	0.000162873	1.80E-06	5.35E-05	5.76E-06
00024115	0.000159206	5.52E-09	2.90E-05	9.99E-08	5.41E-06	8.93E-07
000247e8	0.902253687	0.057683915	0.555676877	0.024452755	0.461518288	0.058318481
00025358	0.01458229	2.85E-07	0.000764962	8.35E-06	0.00035955	0.000125107
00026d10	0.000699297	2.67E-08	8.51E-05	5.87E-07	2.33E-05	4.45E-06
0002eadc	0.696554422	0.000909668	0.287095219	0.00070449	0.118130825	0.006133754
0002f87b1	0.362675607	0.000174356	0.086971484	0.000439866	0.041929886	0.004132608
0003806b	0.001984251	3.76E-08	0.000168882	9.56E-07	5.04E-05	1.02E-05
0003e1ccc	0.000237111	1.41E-08	4.30E-05	2.88E-07	8.33E-06	2.67E-06

6.Conclusion

We initially started out with normal techniques like decision tree, neural networks and simple classifiers which were not really effective. We then moved on to try more complex models like random forest (since a combination of weak classifiers is proved to perform better than one strong classifier). All of us individually started working on each model. That's when we realized we could also explore more on deep learning techniques since we always considered it something fancy. We did have some trials and were finally able to achieve a good accuracy score which was relatively higher than individual classifiers and finally we were a little optimistic about our results. Therefore, we chose keras sequential model as our final model. If we had some more time, we would have tried to explore more about deep learning methodologies and have a further clear understanding of the underlying assumptions involved in it.

7. References

[1] http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[2] J. Huang, S. Ma, and C.-H. Zhang, *Adaptive Lasso for sparse high-dimensional regression models*, Statist. Sinica. 18 (2008), pp. 1603–1618.
[3] https://www.nltk.org/book/ch02.html
[4] https://keras.io/getting-started/sequential-model-guide/
[5] https://keras.io/layers/pooling/
[6] http://scikit-learn.org/stable/modules/ensemble.html