Malignant-Comments-Project

Submitted by:

HARPAL SINGH

ACKNOWLEDGMENT

The project is based on Malignant-Comments machine and deep learning model. Various techniques used in the pipeline of Machine Learning and Deep Learning model building is used from various sources like geeksforgeeks, seaborn documentation, pandas library, NumPy library, NLTK Library. Scikit-learn machine learning models have been used for classification Analysis of target feature. Jupyter notebook has been used throughout the project and its various libraries have been called for various operations.

INTRODUCTION

Business Problem Framing

In this Malignant-Comment-problem we have tried to predict whether the comments from various sources are malignant, highly malignant, rude, threat, abuse and loathe. Online hate, described as abusive language, aggression, cyberbullying, hatefulness and many others has been identified as a major threat on online social media platforms. Social media platforms are the most prominent grounds for such toxic behaviour. Our goal is to build a prototype of online hate and abuse comment classifier which can used to classify hate and offensive comments so that it can be controlled and restricted from spreading hatred and cyberbullying.

Conceptual Background of the Domain Problem

There has been a remarkable increase in the cases of cyberbullying and trolls on various social media platforms. Many celebrities and influences are facing backlashes from people and have to come across hateful and offensive comments. This can take a toll on anyone and affect them mentally leading to depression, mental-illness, self-hatred and suicidal thoughts. Internet comments are bastions of hatred and vitriol. While online anonymity has provided a new outlet for aggression and hate-speech, machine learning can be used to fight it. The problem we sought to solve was tagging of internet comments that are aggressive towards other users. This means that insults to third parties such as celebrities will be tagged as unoffensive, but “u are an idiot” is clearly offensive.

Review of Literature

Various techniques for data cleaning, EDA analysis, Data Pre-processing, Feature-Engineering, Machine Learning and Deep Learning Models selection from sklearn library of machine learning have been used. Insights of the data are found by using various data visualization techniques like countplot. geeksforgeeks website, seaborn, pandas and NumPy documentation and sklearn for for the technical reference have been used. NLTK and regex library has been used for text-preprocessing of text. WordCloud has been used.

Motivation for the Problem Undertaken

Objective to build this machine learning and deep learning model is to have a hands on the model building techniques along with new facing new challenges while solving various anomalies in the dataset. And solving issues encountered during the machine learning building model. Since the number of records is quite high so computation takes a lot of time along with careful selection of features during feature engineering also posed some challenge. There were also some challenges encountered during model building and finding correct evaluation metrics for multilabel classification.

ANALYTICAL PROBLEM FRAMING

Mathematical/ Analytical Modeling of the Problem

Linear Algebra and Calculus concepts are used in the machine learning models. Since various Regressor Models have been used in the Machine Learning Algorithms, the mathematics working behind them. Calculation of evaluation metrics also involved mathematical concepts like algebraic summation. Various mathematical concepts have been used in various algorithms like Conditional probability concept in GaussianNB used in 'binaryrelevance' algorithm. RandomForestClassifier algorithm uses a set of decisiontrees randomly selected subset of the training set which consists of entropy and information gain concepts. And in the analytics modelling various techniques have been used like for analysing the data visualization countplot have been used.

Data Sources and their formats

DataSet have been provided and the processing is done on training dataset and the testing dataset have been used for the testing and predicting the result. The dataset was in csv format.

Data Pre-processing Done

In the data pre-processing and cleaning. Data have been checked for datatype matching values. After that the dataset has been checked for the empty values. Data have been checked for null values. In this dataset there were no null values. Text features are present in the dataset which are used for predicting the output. After that for data visualization countplot is used for the multilabel output.

Data Inputs- Logic- Output Relationships

No Inputs-Logic-Output Relationships have been found to exist between the features.

State the set of assumptions (if any) related to the problem under consideration

No assumption taken during model building.

Hardware and Software Requirements and Tools Used

In the hardware a laptop has been used along with an optical mouse.

In software excel, Python Jupyter notebook, have been used. Here is the table with list of libraries, import method and application of that method/function.

Library	Import method/function	Application
NumPy	array	Creating an array
	zeros	For importing a matrix with zeros.
pandas	DataFrame	importing DataFrame and other DataFrame related operations
matplolib, seaborn	countplot	for data visualization
sklearn.preprocessing.sequence	pad_sequences	Used in the tokenizer.
sklearn.preprocessing.text	Tokenizer	Used for tokenizing the preprocessed and cleaned text.
sklearn.metrics	accuracy_score, hamming_loss	For calculating hamming loss and accuracy score of the multilabel models.
	Multilabel_confusion_matrix	For creating the multilabel confusion matrix.
	classification_report	For displaying the classification report of the classification model.
	joblib	for saving the final model
sklearn.ensemble	RandomForestRegressor,	For importing various classifiers for machine learning algorithms
sklearn.neighbors	KNeighborsRegressor	For importing the Regressor.
sklearn.model_selection	train_test_split	for splitting dataset into training and testing data
	cross_val_score	for importing cross_val_score
keras.models	Sequential	For creating a Sequential neural network model.
	Model	For creating neural network model.
keras.layers.core	Activation,Dense	For adding layer in Neural network.
keras.layers	Flatten, LSTM, GlobalMaxPooling1D	For flattening the layer in neural network, adding LSTM layer and maxpooling layer in the model.
	Input	For inputting the layers data.
keras.layers.embeddings	Embedding	For creating embeddings in the model.
regex	re	Used for text cleaning and text preprocessing.
nltk.corpus	stopwords	For removing the stopwords.
wordcloud	WordCloud, STOPWORDS	For calling wordcloud and stopwords.
nltk.stem	WordNetLemmatizer	For importing the lemmatizer
skmultilearn.problem_transform	BinaryRelevance	For importing the binaryrelevanceclassifier.
	ClassifierChain	For importing the classifier algorithm.
	LabelPowerset	For importing the classifier algorithm.

sklearn.naive_bayes	GaussianNB	For importing the classification algorithm
sklearn.neural_network	MLPClassifier	For importing the classification model.
sklearn.ensemble	RandomForestClassifier	For importing the classifier model.

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

In the problem solving approaches text cleaning is the first important task done for any kind of NLP task whether it is multilabel, multiclass classification. In the text cleaning we have to clean the text by removing punctuations, stopwords, html tags, digits etc for a large amount of data it will take a long time for text cleaning. After that tokenizing the lemmatize, text also pose some problem, which can be solved by effective use of tokenizer and converting the tokenized text also pose some kind of problem which can be solved by using embeddings. Creating the model and running it on training data is also a very important step which should be carefully done and then measuring the performance of model using evaluation metrics.

.

Testing of Identified Approaches (Algorithms)

Algorithms used in multilabel text classification includes Binary Relevance, Classifier Chains, Label Powerset, MLPClassifier, RandomForestClassifier and Neural Network. In the Binary Relevance I have used GaussianNB for classification.

Listing down all the algorithms used for the training and testing.

Algorithms used in the training and testing are:

Binary Relevance

Classifier Chains

LabelPowerset

MLPClassifier

RandomForestClassifier

GaussianNB

NeuralNetwork

Run and Evaluate selected models

This is the simplest technique, which basically treats each label as separate single class classification problem. In multi-label classification problem, we can't simply use our normal metrics to calculate the accuracy of our predictions. For that purpose, we

will use accuracy_score metric. This function calculates the subset of accuracy meaning the predicted set of labels should exactly match with the true set of labels.

```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score,hamming_loss
from sklearn import metrics
# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
binarel = BinaryRelevance(GaussianNB())
# train
binarel.fit(X_train, y_train)
predictions = binarel.predict(X_test)
print(accuracy_score(y_test,predictions))
print(hamming_loss(y_test, predictions))
print(multilabel_confusion_matrix(y_test, predictions))
label_names = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
print(classification_report(y_test, predictions,target_names=label_names))
```

Evaluation metrics and classification report for the algorithm is shown below.

```
0.1587968040106533
0.38769126325134473
[[[ 6207 22652]
  [  377  2679]]

 [[30319  1275]
  [  302    19]]

 [[ 6747 23453]
  [  217  1498]]

 [[31453   388]
  [   74     0]]

 [[ 5815 24486]
  [  154  1460]]

 [[31047   574]
  [  287     7]]]
```

	precision	recall	f1-score	support
malignant	0.11	0.88	0.19	3056
highly_malignant	0.01	0.06	0.02	321
rude	0.06	0.87	0.11	1715
threat	0.00	0.00	0.00	74
abuse	0.06	0.90	0.11	1614
loathe	0.01	0.02	0.02	294
micro avg	0.07	0.80	0.13	7074
macro avg	0.04	0.46	0.07	7074
weighted avg	0.07	0.80	0.13	7074
samples avg	0.06	0.08	0.07	7074

Second Algorithm is ClassifierChains. They are a way of combining a number of binary classifiers into a single multi-label model that is capable of exploiting correlations among targets. For a multilabel classification problem with N classes, N binary classifiers are assigned an integer between 0 and N-1. These integers define the order of models in the chain. Each classifier is then fit on the available training data plus true labels of the classes whose models were assigned a lower number. When predicting the true labels will not be available. Instead of predictions of each model are passed on to the subsequent models in the chain to be used as features. Clearly the order of the chain is important. The first model in the chain has no information about the other labels while the last model in the chain has features indicating the presence of all of the other labels. In general, one does not know the optimal ordering of the models in the chain so typically many randomly ordered chains are fit and their predictions are averaged together.

```
# using classifier chains
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import GaussianNB
# initialize classifier chains multi-label classifier
# with a gaussian naive bayes base classifier
classchain = ClassifierChain(GaussianNB())
# train
classchain.fit(X_train, y_train)
# predict
predictions = classchain.predict(X_test)
print(accuracy_score(y_test,predictions))
print(hamming_loss(y_test, predictions))
print(multilabel_confusion_matrix(y_test, predictions))
label_names = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
print(classification_report(y_test, predictions,target_names=label_names))
```

Evaluation metrics and classification report for the algorithm is shown below.

```
0.17330408898637004
0.6297195676014413
[[[ 6207 22652]
  [  377  2679]]

 [[30911   683]
  [  310    11]]

 [[ 6083 24117]
  [  195  1520]]

 [[ 9438 22403]
  [    8    66]]

 [[ 6293 24008]
  [  174  1440]]

 [[ 6001 25620]
  [   38   256]]]
```

	precision	recall	f1-score	support
malignant	0.11	0.88	0.19	3056
highly_malignant	0.02	0.03	0.02	321
rude	0.06	0.89	0.11	1715
threat	0.00	0.89	0.01	74
abuse	0.06	0.89	0.11	1614
loathe	0.01	0.87	0.02	294
micro avg	0.05	0.84	0.09	7074
macro avg	0.04	0.74	0.08	7074
weighted avg	0.07	0.84	0.13	7074
samples avg	0.04	0.09	0.05	7074

The third method is LabelPowerset problem which is transformation approach to multilabel classification that transforms a multilabel problem to a multi-class problem with 1 multi-class classifier trained on all unique label combinations found in the training data.
The method maps each combination to a unique combination id number, and performs multi-class classification using classifier as multi-class classifier and combination ids as classes.

```
# using Label Powerset
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
# initialize Label Powerset multi-label classifier
# with a gaussian naive bayes base classifier
labelpst = LabelPowerset(GaussianNB())
# train
labelpst.fit(X_train, y_train)
# predict
predictions = labelpst.predict(X_test)
print(accuracy_score(y_test,predictions))
print(hamming_loss(y_test, predictions))
print(multilabel_confusion_matrix(y_test, predictions))
label_names = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
print(classification_report(y_test, predictions,target_names=label_names))
```

Evaluation metrics and classification report for the algorithm is shown below.

```
0.0007833307222309259
0.4676745521959371
[[[11603 17256]
  [  794  2262]]

 [[30133  1461]
  [  300    21]]

 [[20095 10105]
  [ 1334   381]]

 [[13889 17952]
  [   18    56]]

 [[10961 19340]
  [  719   895]]

 [[11472 20149]
  [  127   167]]]
                precision    recall  f1-score   support

       malignant       0.12      0.74      0.20      3056
highly_malignant       0.01      0.07      0.02       321
            rude       0.04      0.22      0.06      1715
          threat       0.00      0.76      0.01        74
           abuse       0.04      0.55      0.08      1614
          loathe       0.01      0.57      0.02       294

       micro avg       0.04      0.53      0.08      7074
       macro avg       0.04      0.48      0.07      7074
    weighted avg       0.07      0.53      0.12      7074
     samples avg       0.04      0.06      0.04      7074
```

The Fourth Algorithm is MLPClassifier which trains iteratively since at each timestep the partial derivatives of the loss function with respect to the model parameters are computed to update the parameters. It can also have a regularization term added to the loss function that shrinks model parameters to prevent overfitting. This implementation works with data represented as dense NumPy arrays or sparse SciPy arrays of floating point:

```
from sklearn.neural_network import MLPClassifier
mlpcf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
mlpcf.predict_proba(X_test)
predictions=mlpcf.predict(X_test)
mlpcf.score(X_test, y_test)
print(accuracy_score(y_test,predictions))
print(hamming_loss(y_test, predictions))
print(multilabel_confusion_matrix(y_test, predictions))
label_names = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
print(classification_report(y_test, predictions,target_names=label_names))
```

The algorithm and evaluation metric used are:

```
0.8981983393388688
0.03699932111337407
[[[28854     5]
  [ 3056     0]]

 [[31594     0]
  [  321     0]]

 [[30196     4]
  [ 1715     0]]

 [[31841     0]
  [   74     0]]

 [[30299     2]
  [ 1614     0]]

 [[31621     0]
  [  294     0]]]
```

	precision	recall	f1-score	support
malignant	0.00	0.00	0.00	3056
highly_malignant	0.00	0.00	0.00	321
rude	0.00	0.00	0.00	1715
threat	0.00	0.00	0.00	74
abuse	0.00	0.00	0.00	1614
loathe	0.00	0.00	0.00	294
micro avg	0.00	0.00	0.00	7074
macro avg	0.00	0.00	0.00	7074
weighted avg	0.00	0.00	0.00	7074
samples avg	0.00	0.00	0.00	7074

The fifth algorithm is RandomForestClassifier which is used for the classification is a type of ensemble technique which is generated using a random selection of attributes at each node to determine the split. During classification each tree votes and the most popular class is returned. It is comparable to adaboost in accuracy but more robust to errors and outliers. This technique is insensitive to the number of attributes selected for consideration at each split, and faster than bagging and boosting.

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(max_depth=2, random_state=0)
rf.fit(X_train, y_train)
predictions=rf.predict(X_test)
print(accuracy_score(y_test,predictions))
print(hamming_loss(y_test, predictions))
print(multilabel_confusion_matrix(y_test, predictions))
label_names = ["malignant", "highly_malignant", "rude", "threat", "abuse", "loathe"]
print(classification_report(y_test, predictions,target_names=label_names))
```

The Evaluation Metric used in the algorithm is:

```
0.898355005483315
0.036941876860410464
[[[28859     0]
  [ 3056     0]]

 [[31594     0]
  [  321     0]]

 [[30200     0]
  [ 1715     0]]

 [[31841     0]
  [   74     0]]

 [[30301     0]
  [ 1614     0]]

 [[31621     0]
  [  294     0]]]
```

	precision	recall	f1-score	support
malignant	0.00	0.00	0.00	3056
highly_malignant	0.00	0.00	0.00	321
rude	0.00	0.00	0.00	1715
threat	0.00	0.00	0.00	74
abuse	0.00	0.00	0.00	1614
loathe	0.00	0.00	0.00	294
micro avg	0.00	0.00	0.00	7074
macro avg	0.00	0.00	0.00	7074
weighted avg	0.00	0.00	0.00	7074
samples avg	0.00	0.00	0.00	7074

The sixth algorithm is GaussianNB which implements the Naïve Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian.

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

This algorithm is used in each and every classifier.

The seventh Algorithm is Neural Network which is created and using embedding layer, LSTM layer and dense_layer and finally the accuracy_score is predicted based on the model.

```
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix], trainable=False)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
dense_layer_1 = Dense(6, activation='sigmoid')(LSTM_Layer_1)
model = Model(inputs=deep_inputs, outputs=dense_layer_1)
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['acc'])
```

```
score = model.evaluate(X_test, y_test, verbose=1)
print("Test Score:", score[0])
print("Test Accuracy:", score[1])
```

```
998/998 [==============================] - 108s 108ms/step - loss: 0.0563 - acc: 0.9940
Test Score: 0.05628538876771927
Test Accuracy: 0.993984043598175
```

```
score=model.evaluate(X_train,y_train, verbose=1)
print('Training Score:',score[0])
print("Training Accuracy:", score[1])
```

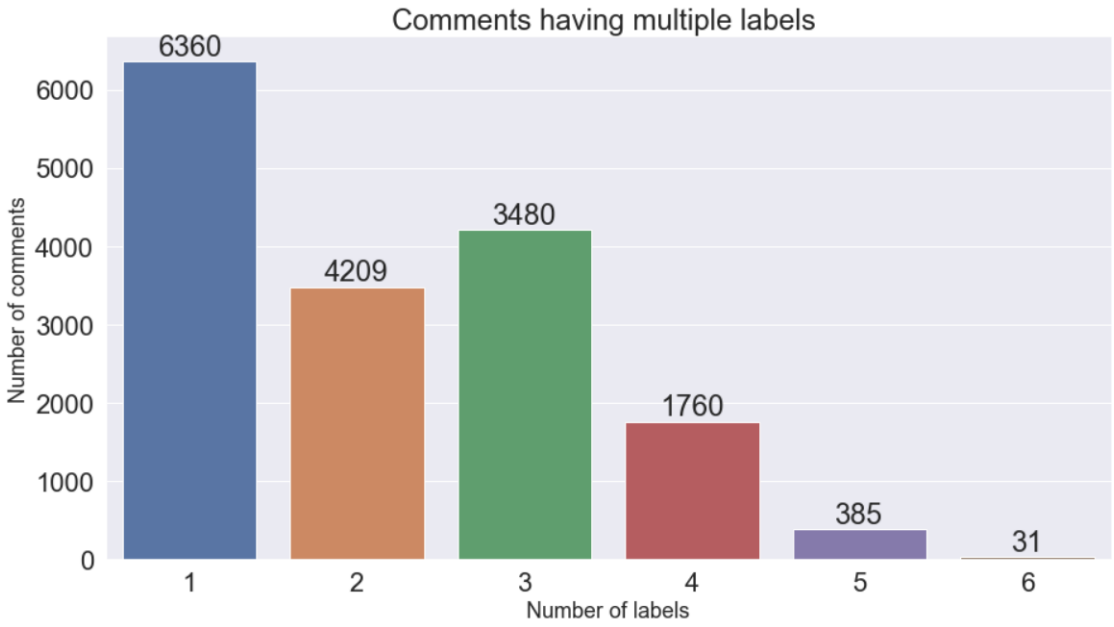
```
3990/3990 [==============================] - 381s 96ms/step - loss: 0.0542 - acc: 0.9941
Training Score: 0.05417363718152046
Training Accuracy: 0.9941248297691345
```

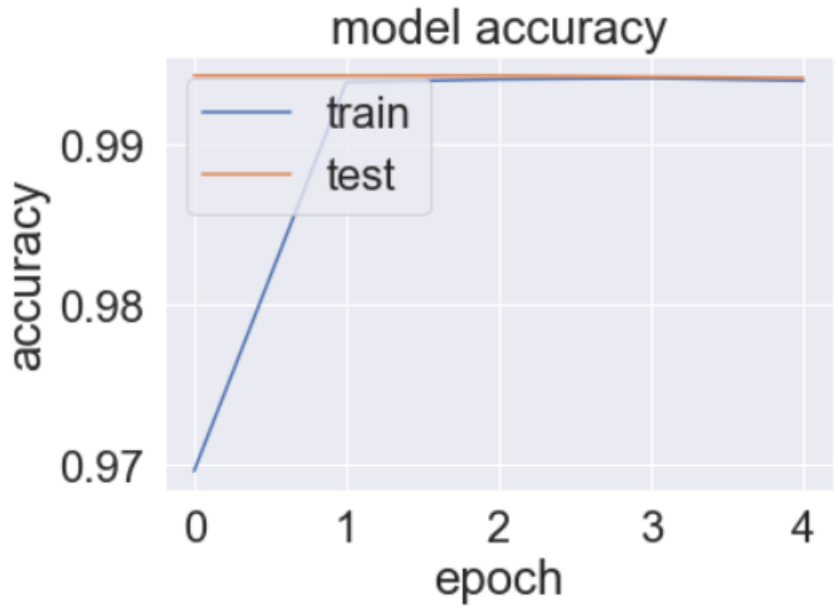
Key Metrics for success in solving problem under consideration

Key Metrics for success in solving problem under consideration is accuracy_score, recall and precision which are used for all classification algorithms and are having different values.

Visualizations

Visualization plots that have been used in the project includes countplots:





Interpretation of the Results

From the Data Visualization we can see those comments having number of labels 5 are most in number followed by 3 followed by 2 and so on. Since the dataset we have used is having a large number of data pre-processing should be done carefully so as to remove the various unwanted html tags, punctuation marks etc and finally tokenization along with word embeddings. Accuracies along with precision and recall is found for various algorithms and for neural network it is found maximum. Hamming loss is also found for various algorithms. The accuracies for various algorithms are found to be very less like for BinaryRelevance it is found to be 0.158 with hamming loss of 0.3876 precision and recall for some labels are also found to be 0. Precision,Recall and f1-score for MLPClassifier and RandomForestClassifier are found to be 0 for all the labels. FalsePositive for some algorithms are found to be more than TruePositive that means model is predicting some datapoints as true even if they are false.

.

CONCLUSION

Key Findings and Conclusions of the Study

From the complete project we came to know that starting from importing the dataset and to the end of the model building and checking its f1_score there come a lot of challenges. Challenges include starting from text-cleaning and text-preprocessing which includes checking for null and empty values. Then other pre-processing techniques like Data Visualization are also used. The results obtained from the Neural Network are best accuracy wise since models with other accuracies are far below that of the Neural Network. The training accuracy increases with increase in the number of epoch whereas the testing accuracy is found to be constant with increase in number of epochs. The training and testing loss is decreasing as the epochs increases from 0 to 4. The testing epochs decreases with steep decrease whereas the training epochs decreases with gradual decrease.

Learning Outcomes of the Study in respect of Data Science

Learnings includes deep analysis of various data visualization techniques to get insights of the data and the variation of data in features with respect to label. Challenges in this project include feature selection, Exhaustive computation for text-preprocessing since number of rows are very large. Selecting the best algorithm for the Final Model. Creating the Neural Network and training datapoints on it.

Limitations of this work and Scope for Future Work

Limitations of this work is that more robust model can be build using some more advanced techniques in machine learning and deep learning model building and Exhaustive EDA can also improve the accuracy of the model using some more techniques for the EDA analysis can solve the problem of less accuracy. Future work can be done in the pre-processing and data visualization part for analysing data more effectively.