

MACHINE LEARNING PROJECT

TITLE
WHY SO HARSH?

TEAM NAME: **MAVERICKS**

TEAM MEMBERS:

HARSH TRIPATHI (MT2022048)

GOPAL GUPTA (MT2022046)



CONTENT

- Introduction to NLP
- Problem Statement
- Dataset Description and Files
- Data Pre-processing
- Vectorization
- Models
- Conclusion

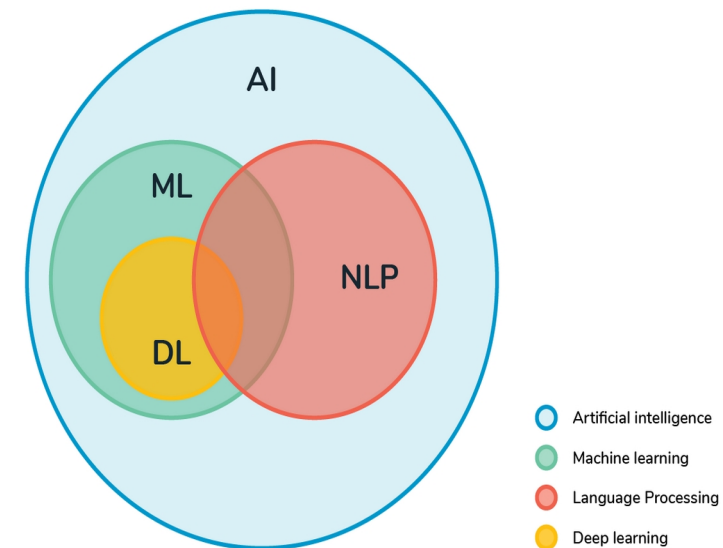
INTRODUCTION TO NLP

Natural language processing (NLP) is the ability of a computer program to understand human language as it is spoken and written -- referred to as natural language. It is a component of artificial intelligence ([AI](#)).

NLP has existed for more than 50 years and has roots in the field of linguistics. It has a variety of real-world applications in a number of fields, including medical research, search engines and business intelligence.

NLP enables computers to understand natural language as humans do. Whether the language is spoken or written, natural language processing uses artificial intelligence to take real-world input, process it, and make sense of it in a way a computer can understand. And just as humans have a brain to process that input, computers have a program to process their respective inputs. At some point in processing, the input is converted to code that the computer can understand.

There are two main phases to natural language processing: [data preprocessing](#) and algorithm development. Data pre-processing involves preparing and "cleaning" text data for machines to be able to analyze it.



PROBLEM STATEMENT

Freedom of speech is one of the fundamental human rights. Keeping this in mind, you have created a blog on UpBlog where you would like to share content about your favorite things, especially movies and sports where the conflicts among the fan groups are enormously high. Since you stand by this “Freedom of Speech” right, you let people comment on your posts to get to know how the other fans feel about your opinion. However, this goes south too soon.

The reason is that there has been a lot of abuse, hate, repugnant, and harsh content in the comments. Some are so hard to even read let alone how the targeted community feels like. Luckily you have started learning about machine learning and you see a clear application of it in this scenario.

TASK

Given a comment, classify it into “harsh”, “extremely harsh”, “vulgar”, “threatening”, “disrespect”, “targeted hate”.

For example, consider a comment, “You useless piece of trash! you are such an asshole that you deserve to rot in the gutter alongside sewage.” That is certainly harsh! So, the label for this would be [1 1 0 0 1 1]. Translating to the comment being harsh, extremely harsh, disrespectful, and targeted hate.

DATASET DESCRIPTION AND FILES

The dataset contains the text associated with a particular comment and the corresponding labels for each of the classes. The classes are:

harsh
extremely_harsh
vulgar
threatening
disrespect
targeted_hate

Files

train.csv - the training set
test.csv - the test set
sample_submission.csv

So, if there is a comment “You useless piece of trash! you are such an asshole that you deserve to rot in the gutter alongside sewage.” with labels [1 1 0 0 1 1] meaning that the comment was

harsh
extremely_harsh
vulgar
threatening
disrespect
targeted_hate

Note: Each class is independent of other classes.

Steps:

1. PRE-PROCESSING

Pre-processing puts data in workable form and highlights features in the text that an algorithm can work with. There are several ways this can be done, including:

[Tokenization](#). This is when text is broken down into smaller units to work with.

[Stop word](#) **removal**. This is when common words are removed from text so unique words that offer the most information about the text remain.

[Lemmatization](#) **and stemming**. This is when words are reduced to their root forms to process.

1. PRE-PROCESSING

IMPORTING THE LIBRARIES

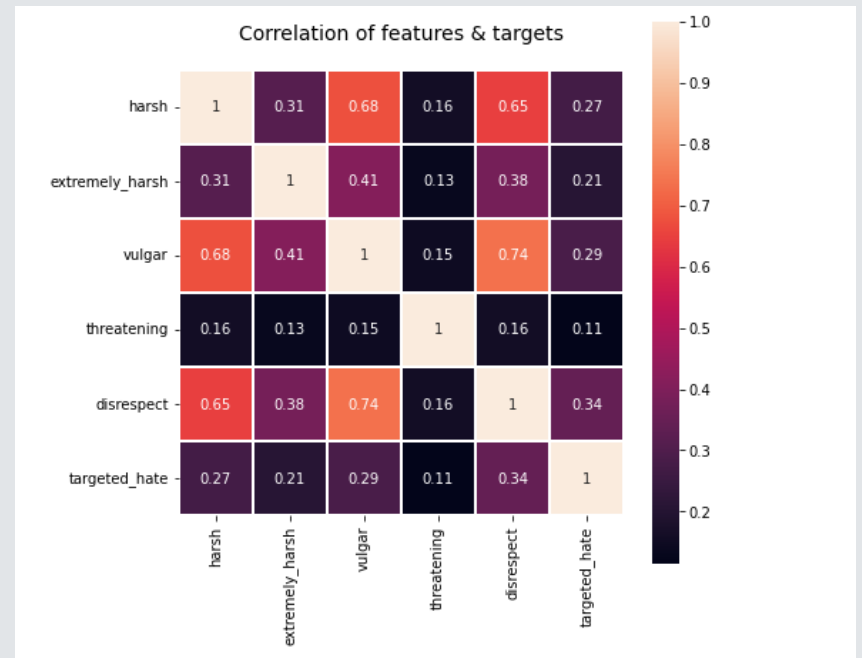
Importing the relevant libraries used in our project. Some of the important libraries are:

- Matplotlib for plotting the graph
- Nltk – natural language toolkit
- numpy- for array manipulations
- Pandas- for manipulating and importing our dataset
- Re – for regular expressions , etc

IMPORTING THE DATASET

ANALYSIS OF DATASET

Following are the Co-relation matrix



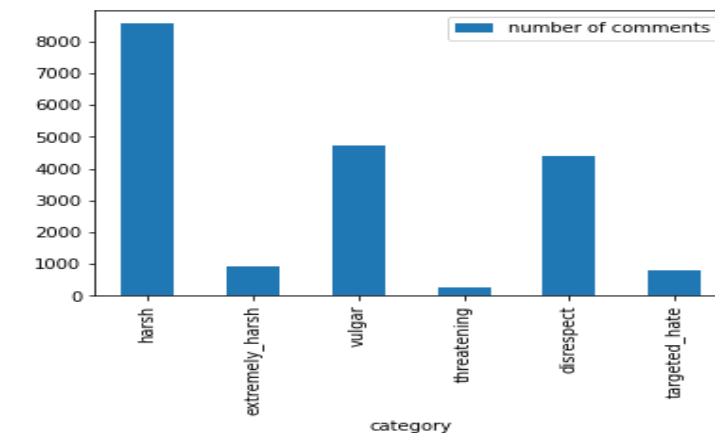
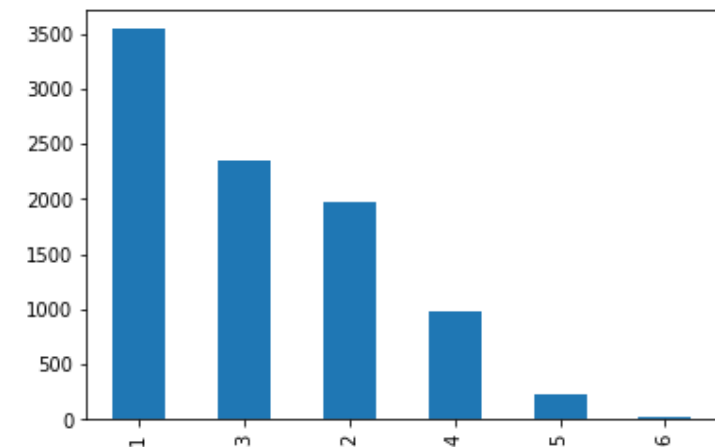
1. PRE-PROCESSING

ANALYSIS OF DATASET

- Checking percentage of comments that are harsh (and other categories)
- Distribution of the count
- Checking for the missing values
- Correlation of features and targets
- Checking the number of comments in particular category

PREPROCESS THE TEXT

- Converting to lowercase
- Removing everything except alphabetical characters
- Tokenize (split into list and remove whitespace)

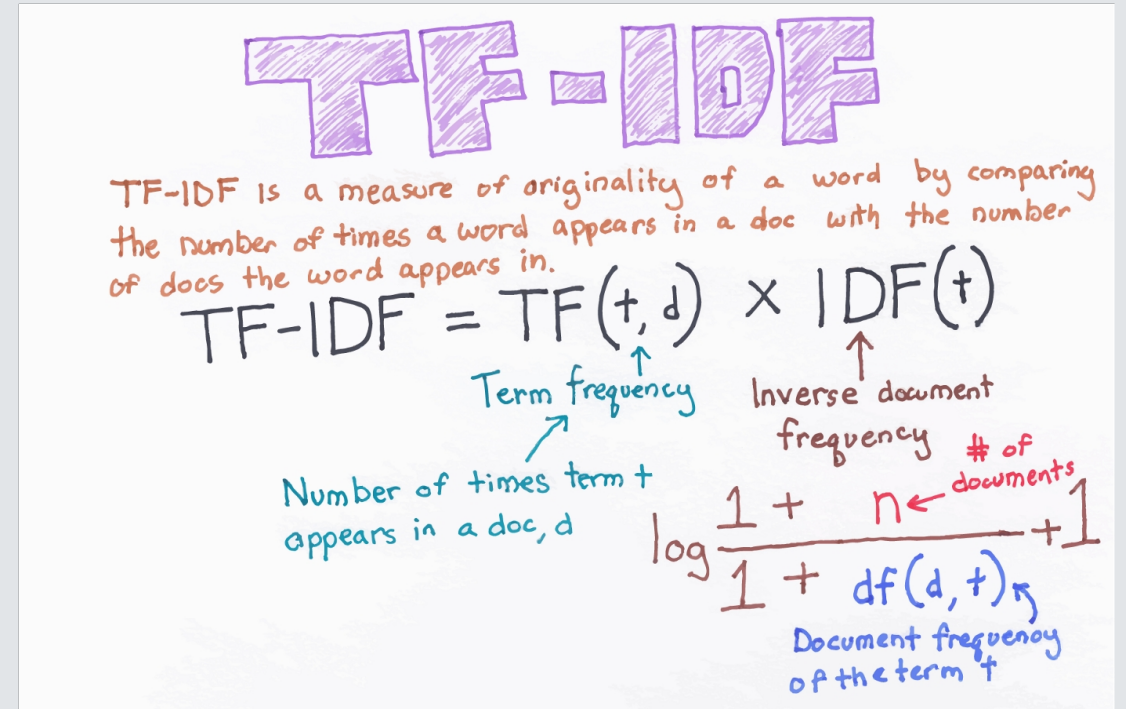


2. DATA VECTORIZATION

Creating the Vectorize

Vectorization is jargon for a classic approach of converting input data from its raw format (i.e. text) into vectors of real numbers which is the format that ML models support. This approach has been there ever since computers were first built, it has worked wonderfully across various domains, and it's now used in NLP.

In Machine Learning, vectorization is a step in feature extraction. The idea is to get some distinct features out of the text for the model to train on, by converting text to numerical vectors.



TF-IDF

TF-IDF is a measure of originality of a word by comparing the number of times a word appears in a doc with the number of docs the word appears in.

$$\text{TF-IDF} = \text{TF}(t, d) \times \text{IDF}(t)$$

Term frequency

Number of times term t appears in a doc, d

Inverse document frequency

$\log \frac{1 + n}{1 + df(d, t)}$

n ← # of documents

$df(d, t)$ ← Document frequency of the term t

2. DATA VECTORIZATION

Vectorization using TF-IDF

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm for prediction.

For better results we vectorize both words as well char.

Following are the parameters used in the vectorization.

max_features	If not None, build a vocabulary that only consider the top max_features ordered by term frequency across the corpus.
strip_accents	Remove accents and perform other character normalization during the preprocessing step. Unicode works on all.
analyzer	Whether the feature should be made of word or character n-grams.
ngram_range	he lower and upper boundary of the range of n-values for different n-grams to be extracted.eg (1,1) is unigram while (1,2) is bigram.
min_df	When building the vocabulary ignore terms that have a document frequency strictly lower than the given threshold.
max_df	When building the vocabulary ignore terms that have a document frequency strictly higher than the given threshold (corpus-specific stop words)

2. DATA VECTORIZATION

Vectorization using Word2Vec

Word2Vec is a method to construct such an embedding. It can be obtained using two methods (both involving Neural Networks): Skip Gram and Common Bag Of Words (CBOW)

It also helps us to find the relation between the words.

```
num_features = 300 # Word vector dimensionality
min_word_count = 40 # Minimum word count
num_workers = 4 # Number of threads to run in parallel
context = 10 # Context window size
downsampling = 1e-3 # Downsample setting for frequent words
# Initialize and train the model (this will take some time)
w2v_model = Word2Vec(pd.concat([X_train_w, X_test_w]), workers=num_workers,
                      vector_size=num_features, min_count = min_word_count, window = context, sample = downsampling)
```

+ Code

+ Markdown

```
w2v_model.wv.most_similar('nigga')
```

```
[('twat', 0.814550518989563),
 ('cum', 0.8137764930725098),
 ('kick', 0.7954991459846497),
 ('pussy', 0.79521244764328),
 ('motherfucking', 0.7934619784355164),
 ('fuckin', 0.7922528982162476),
 ('delanoy', 0.7886987924575806),
 ('whore', 0.7860848903656006),
 ('wat', 0.7838656306266785),
 ('wanker', 0.781448245048523)]
```

2. DATA VECTORIZATION

Vectorization using gloVe

GloVe is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

It is Pre-trained model.

```
# create embedding index
embedding_index = {}
with open('../input/glove6b100dtxt/glove.6B.100d.txt', encoding='utf-8') as f:
    for line in f:
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embedding_index[word] = coefs
```

+ Code

+ Markdown

```
embedding_index['good']
```

```
array([-0.030769 ,  0.11993 ,  0.53909 , -0.43696 , -0.73937 ,
        -0.15345 ,  0.081126 , -0.38559 , -0.68797 , -0.41632 ,
        -0.13183 , -0.24922 ,  0.441   ,  0.085919 ,  0.20871 ,
        -0.063582 ,  0.062228 , -0.051234 , -0.13398 ,  1.1418 ,
         0.036526 ,  0.49029 , -0.24567 , -0.412   ,  0.12349 ,
         0.41336 , -0.48397 , -0.54243 , -0.27787 , -0.26015 ,
        -0.38485 ,  0.78656 ,  0.1023  , -0.20712 ,  0.40751 ,
         0.32026 , -0.51052 ,  0.48362 , -0.0099498, -0.38685 ,
```

3. DATA MODELLING

Linear SVC

Similar to SVC with parameter kernel='linear', but implemented in terms of liblinear rather than libsvm, so it has more flexibility in the choice of penalties and loss functions and should scale better to large numbers of samples.

Following are the screenshot of classification report and confusion matrix of few labels.

Only Parameter used is class_weight.

```
... Processing threatening
      precision    recall  f1-score   support

         0       1.00      1.00      1.00     29395
         1       0.52      0.28      0.36         94

   accuracy          1.00     29489
  macro avg       0.76      0.64      0.68     29489
 weighted avg       1.00      1.00      1.00     29489

[[29371   24]
 [   68   26]]
... Processing disrespect
      precision    recall  f1-score   support

         0       0.98      0.99      0.99     28031
         1       0.79      0.59      0.67     1458

   accuracy          0.97     29489
  macro avg       0.88      0.79      0.83     29489
 weighted avg       0.97      0.97      0.97     29489

[[27798   233]
 [   605   853]]
```

3. DATA MODELLING

Logistic and Ridge Classification

Ridge classification works by adding a penalty term to the cost function that discourages complexity. The penalty term is typically the sum of the squared coefficients of the features in the model. This forces the coefficients to remain small, which prevents overfitting.

Since our data is quite small so overfitting is common. To avoid this issue we have used ridge classification

For Parameter we have used $\alpha=17$, which is regularization.

```
... Processing harsh
harsh 0.977939723095994
... Processing extremely_harsh
extremely_harsh 0.9845253352792922
... Processing vulgar
vulgar 0.9907925284798883
... Processing threatening
threatening 0.9842081262915607
... Processing disrespect
disrespect 0.9802833909292764
... Processing targeted_hate
targeted_hate 0.9795055400683048
average score 0.9828757740240528
```

3. DATA MODELLING

Long Short Term Memory(LSTM)

LSTM stands for Long-Short Term Memory. LSTM is a type of recurrent neural network but is better than traditional recurrent neural networks in terms of memory. Having a good hold over memorizing certain patterns LSTMs perform fairly better. As with every other NN, LSTM can have multiple hidden layers and as it passes through every layer, the relevant information is kept and all the irrelevant information gets discarded in every single cell.

Following are the models with 3 hidden layers, with activations 'relu' and 'sigmoid' and optimizer 'Adam'.

```
model = models.Sequential()
model.add(layers.Embedding(vocab_size, 128, input_length=50))
model.add(layers.LSTM(512, dropout=0.2, recurrent_dropout=0.2, return_sequences=True))
model.add(layers.LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(6, activation='sigmoid'))

model.compile(
    loss='binary_crossentropy',
    optimizer='Adamax',
    metrics=['accuracy'])


model.summary()
```


Conclusion

We have got our best accuracy by using the combination of Logistic and Ridge Classification, since both models tries to avoid overfitting by applying regularization terms. Since we are using binary classification so we are considering each label is independent of other so we pick different model for different lables.

Highest Accuracy(roc-auc) : **0.98551**

Kind Regards,
Mavericks



THANK YOU

