05_final_report

May 13, 2019

1 MSD 2019 Final Project

1.1 A replication and extension of "Automated Hate Speech Detection and the Problem of Offensive Language" by Thomas Davidson, Dana Warmsley, Michael Macy, Ingmar Weber4 Published in ICWSM 2017.

1.1.1 Ketakee Nimavat (kkn2112), Chandana priya (cg3111), Ankit Peshin (ap3772)

2 Introduction

2.1 Motivation

The main focus of this paper was the differentiation between hate speech and offensive speech. Drawing from the definitions of the paper, hate speech is language that is used to expresses hatred towards a targeted group or is intended to be derogatory, to humiliate, or to insult the members of the group. The targets usually are a minority group or people with disadvantage. In extreme cases, this hate speech can sometimes insinuate violence. On the other hand, offensive speech is defined as the language of sexisim and obscene comments made by one person to another.

Because of its seriousness, many countries like France, UK, Canada pose legal implications/charges to those found guilty of using hate speech. These laws usually extends to TV, media and the internet. Given this scenario, it becomes necessary to distinguish between the two highly similar languages. The paper mainly concentrates on this specific issue. The ground truth for this analysis is collected via crowdsourcing where each tweet/text is classified as hate speech, offensive or neither. The paper also discusses about when this distinction is not feasible.

2.2 Replication & Implementation Overview

To reproduce the results from the paper, we used the same dataset provided by the authors, containing about 25k sample tweets and their corresponding labels assigned via crowdsourcing. Each tweet had mentions, retweets and/or hashtags. To have a count on the number of mentions and urls and to identify if it was a retweet, we replaces any url sequence with 'URLHERE', @_name__ with 'MENTIONHERE' and check if there was a 'rt' (retweet) to indicate if it was a retweet.

We later cleaned the tweets to remove special characters and tokenize the words. We also removed stopwords from the tweets using the NLTK library. Once we have the tweets tokenized to words, we used the stemming technique to identify tokens by their root words. The NLP toolkit provides built in packages for stemming words with many variants. We've used PorterStemmer in the implementation. Furthermore, the Part-of-Speech (POS) tags for unigrams, bigrams, and

trigrams are computed. To capture the quality of each tweet we use modified Flesch-Kincaid scores and also calculated the sentiment score associated with each tweet.

3 Steps for replication

3.1 Data

The data corpus we used for replicating the results from "Automated Hate Speech Detection and the Problem of Offensive Language" is the original dataset, provided by the author. This data has been generated by crowdsourcing various tweets with their corresponding labels. This forms the ground truth for our analysis.

```
In [0]: import pandas as pd
```

```
In [0]: import numpy as np
```

```
In [4]: !pip install vaderSentiment
```

```
Requirement already satisfied: vaderSentiment in /usr/local/lib/python3.6/dist-packages (3.2.1)
```

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.model_selection import train_test_split
        from sklearn.pipeline import make_pipeline
        from sklearn.linear_model import LogisticRegression
        from sklearn.model_selection import cross_val_score
        from sklearn.model_selection import GridSearchCV
        from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer
        import nltk
        import gensim
        from nltk.tokenize import sent_tokenize, word_tokenize
        from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer
        from scipy.sparse import hstack
        from sklearn.metrics import average_precision_score
        from sklearn.metrics import roc_auc_score
        from nltk import word_tokenize,sent_tokenize
        from gensim import corpora
        from sklearn.pipeline import Pipeline
        from sklearn.model_selection import cross_val_predict
        import matplotlib.pyplot as plt
```

3.1.1 Loading the data

```
In [0]: df_train = pd.read_csv("/data/labeled_data.csv")[["tweet","class"]]
```

```
In [7]: df_train.head()
```

```
Out[7]:                                               tweet  class
        0  !!! RT @mayasolovely: As a woman you shouldn't...      2
```

```
1  !!!!! RT @mleew17: boy dats cold...tyga dwn ba...      1
2  !!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...      1
3  !!!!!!!!! RT @C_G_Anderson: @viva_based she lo...      1
4  !!!!!!!!!!!!! RT @ShenikaRoberts: The shit you...      1
```

Here we take equal data from all classes, one class has a little bit less data but that is a minor imbalance that can be tolerated

```
In [8]: X_train = df_train[df_train['class']==0]["tweet"]
        X__train = df_train[df_train['class']==1]["tweet"][:1000]
        X_train = X_train.append(X__train)
        X__train = df_train[df_train['class']==2]["tweet"][:1000]
        X_train = X_train.append(X__train)
        y_train = df_train[df_train['class']==0]["class"]
        y_train = y_train.append(df_train[df_train['class']==1]["class"][:1000])
        y_train = y_train.append(df_train[df_train['class']==2]["class"][:1000])
        len(X_train)

Out[8]: 3430

In [9]: len(y_train)

Out[9]: 3430
```

3.2 Feature Enginerring

Since we are dealing with unstructured data, we used preprocessing, stemming, vectorization, POS tag and sentiment scores to consider right words (tokens) to the features set along with some summary statistics like number of number of words, characters, number of times a URL was sited, number of times a tweet was retweeted, number of mentions etc.

1. Lowercase
2. Stem
3. bigram, unigram, trigram features, weighted by its tfidf
4. POS tag
5. FK Grade level
6. FK reading ease score
7. sentiment scores
8. binary indicators for: hashtags, mentions, retweets, urls
9. count indicatiors for :hashtags, mentions, retweets, urls
10. number of characters
11. numbers of words

12. number of syllables

```
In [10]: nltk.download("stopwords")
         from nltk.stem.porter import *

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!


In [0]: stopwords=stopwords = nltk.corpus.stopwords.words("english")

        other_exclusions = ["#ff", "ff", "rt","RT"]
        stopwords.extend(other_exclusions)

        stemmer = PorterStemmer()

        def preprocess(text_string):

            #Lowercase string
            text_string=text_string.lower()
            space_pattern = '\s+'
            giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
                '[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
            mention_regex = '@[\w\-]+'
            hashtag_regex = '#[\w\-]+'
            parsed_text = re.sub(space_pattern, ' ', text_string)
            parsed_text = re.sub(giant_url_regex, 'URLHERE', parsed_text)
            parsed_text = re.sub(mention_regex, 'MENTIONHERE', parsed_text)
            parsed_text = re.sub(hashtag_regex, 'HASHTAGHERE', parsed_text)

            #Stem it
            tweet = " ".join(re.split("[^a-zA-Z]*", parsed_text)).strip()
            tokens = [stemmer.stem(t) for t in tweet.split()]
            return tokens


        def pos_tag_seq(tokens):
            tags = nltk.pos_tag(tokens)
            tag_list = [x[1] for x in tags]
            tag_str = " ".join(tag_list)
            return tag_str



In [0]: def join_sent(l):
          return " ".join(l)

In [0]: df_train=pd.DataFrame(X_train)
```

```
In [0]: df_train.columns=["tweet"]

In [15]: df_train.head()

Out[15]:                                                  tweet
         85   "@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
         89   "@CB_Baby24: @white_thunduh alsarabsss" hes a ...
         110  "@DevilGrimz: @VigxRArts you're fucking gay, b...
         184  "@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
         202  "@NoChillPaz: "At least I'm not a nigger" http...

In [16]: s_train=df_train['tweet'].apply(preprocess)

/usr/lib/python3.6/re.py:212: FutureWarning: split() requires a non-empty pattern match.
  return _compile(pattern, flags).split(string, maxsplit)


In [0]: s_tr=s_train.apply(join_sent)

In [18]: nltk.download('averaged_perceptron_tagger')
         t_tr=s_train.apply(pos_tag_seq)

[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!


In [0]: vectorizer = TfidfVectorizer(
            preprocessor=None,
            lowercase=False,
            ngram_range=(1, 3),
            use_idf=True,
            smooth_idf=False,
            norm=None,
            stop_words=stopwords,
            decode_error='replace',
            max_features=10000,
            min_df=5,
            max_df=0.75
            )

In [0]: pos_vectorizer = TfidfVectorizer(
            tokenizer=None,
            lowercase=False,
            preprocessor=None,
            ngram_range=(1, 3),
            stop_words=None,
            use_idf=False,
```

```
            smooth_idf=False,
            norm=None,
            decode_error='replace',
            max_features=5000,
            min_df=5,
            max_df=0.75,
            )
```

```
In [0]: tfidf_tr = vectorizer.fit_transform(s_tr).toarray()

        vocab = {v:i for i, v in enumerate(vectorizer.get_feature_names())}
        idf_vals = vectorizer.idf_
        idf_dict = {i:idf_vals[i] for i in vocab.values()}
```

```
In [0]: pos_tr = pos_vectorizer.fit_transform(t_tr).toarray()

        pos_vocab = {v:i for i, v in enumerate(pos_vectorizer.get_feature_names())}
```

```
In [0]: from vaderSentiment.vaderSentiment import SentimentIntensityAnalyzer as VS
        sentiment_analyzer = VS()
```

```
In [0]: def get_sentiment(text):
          sentiment = sentiment_analyzer.polarity_scores(text)
          return sentiment

        #   return sentiment["neg"], sentiment["pos"], sentiment["neu"]
```

```
In [0]: df_train["sent"]=df_train["tweet"].apply(get_sentiment)
```

```
In [26]: df_train.head()
```

```
Out[26]:                                                  tweet  \
         85   "@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
         89   "@CB_Baby24: @white_thunduh alsarabsss" hes a ...
         110  "@DevilGrimz: @VigxRArts you're fucking gay, b...
         184  "@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
         202  "@NoChillPaz: "At least I'm not a nigger" http...

                                                        sent
         85   {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
         89   {'neg': 0.187, 'neu': 0.813, 'pos': 0.0, 'comp...
         110  {'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound...
         184  {'neg': 0.254, 'neu': 0.746, 'pos': 0.0, 'comp...
         202  {'neg': 0.232, 'neu': 0.488, 'pos': 0.28, 'com...
```

```
In [0]: foo_tr = lambda x: pd.Series([x["pos"],x["neg"],x["neu"]])
        rev_tr = df_train['sent'].apply(foo_tr)
```

```
In [0]: rev_tr.columns=["pos","neg","neu"]
```

```
In [29]: rev_tr.head()

Out[29]:       pos    neg    neu
         85   0.00  0.000  1.000
         89   0.00  0.187  0.813
         110  0.00  0.000  1.000
         184  0.00  0.254  0.746
         202  0.28  0.232  0.488
```

3.2.1 Binary count for URL https mentions etc

we replace each URL expression with "URLHERE" and each '@....' with a "MENTIONHERE" so that we can later keep track of the number of occurences of these terms

```
In [0]: def return_cont(parsed_text):
          return(parsed_text.count('urlher'),parsed_text.count('mentionher'),parsed_text.count

In [0]: df_train["counts"]=s_tr.apply(return_cont)

In [32]: df_train["counts"].head()

Out[32]: 85     (0, 3, 0)
         89     (0, 2, 0)
         110    (1, 2, 1)
         184    (1, 1, 0)
         202    (1, 1, 0)
         Name: counts, dtype: object

In [0]: foo = lambda x: pd.Series([x[0],x[1],x[2]])
        mention_counts_tr = df_train['counts'].apply(foo)

In [34]: mention_counts_tr.head()

Out[34]:      0  1  2
         85   0  3  0
         89   0  2  0
         110  1  2  1
         184  1  1  0
         202  1  1  0
```

3.2.2 FKRA and Flesch and number of syllables etc

```
In [35]: !pip install textstat
         from textstat.textstat import *

Requirement already satisfied: textstat in /usr/local/lib/python3.6/dist-packages (0.5.6)
Requirement already satisfied: pyphen in /usr/local/lib/python3.6/dist-packages (from textstat)
Requirement already satisfied: repoze.lru in /usr/local/lib/python3.6/dist-packages (from texts
```

```
In [0]: def get_other_features(text):
            space_pattern = '\s+'
            giant_url_regex = ('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|'
                '[!*\(\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+')
            mention_regex = '@[\w\-]+'
            parsed_text = re.sub(space_pattern, ' ', text)
            parsed_text = re.sub(giant_url_regex, '', parsed_text)
            words = re.sub(mention_regex, '', parsed_text)

            syllables = textstat.syllable_count(words)
            num_chars = sum(len(w) for w in words)
            num_chars_total = len(text)
            num_terms = len(text.split())
            num_words = len(words.split())
            avg_syl = round(float((syllables+0.001))/float(num_words+0.001),4)
            num_unique_terms = len(set(words.split()))

            ###Modified FK grade, where avg words per sentence is just num words/1
            FKRA = round(float(0.39 * float(num_words)/1.0) + float(11.8 * avg_syl) - 15.59,1)
            ##Modified FRE score, where sentence fixed to 1
            FRE = round(206.835 - 1.015*(float(num_words)/1.0) - (84.6*float(avg_syl)),2)

            features = [FKRA, FRE,syllables, avg_syl, num_chars, num_terms, num_words,
                        num_unique_terms]
            return features



In [0]: other_feats_tr=df_train["tweet"].apply(get_other_features)

In [38]: other_feats_tr.head()

Out[38]: 85         [9.2, 34.62, 6, 1.9997, 18, 5, 3, 3]
         89        [6.8, 67.76, 18, 1.5, 59, 13, 12, 10]
         110    [9.1, 49.55, 19, 1.7272, 76, 13, 11, 11]
         184    [5.2, 84.46, 19, 1.2666, 78, 15, 15, 15]
         202        [2.3, 94.3, 11, 1.2222, 38, 9, 9, 9]
         Name: tweet, dtype: object

In [0]: other_features_names = ["FKRA", "FRE","num_syllables", "avg_syl_per_word", "num_chars"

In [0]: foo = lambda x: pd.Series(elem for elem in x)
        of_counts_tr = other_feats_tr.apply(foo)

In [41]: of_counts_tr.head()

Out[41]:        0      1     2       3     4     5     6     7
         85   9.2  34.62   6.0  1.9997  18.0   5.0   3.0   3.0
         89   6.8  67.76  18.0  1.5000  59.0  13.0  12.0  10.0
```

```
110  9.1  49.55  19.0  1.7272  76.0  13.0  11.0  11.0
184  5.2  84.46  19.0  1.2666  78.0  15.0  15.0  15.0
202  2.3  94.30  11.0  1.2222  38.0   9.0   9.0   9.0
```

```
In [0]: of_counts_tr.columns=other_features_names
```

```
In [43]: of_counts_tr.head()
```

```
Out[43]:      FKRA    FRE  num_syllables  avg_syl_per_word  num_chars  num_terms  \
85     9.2  34.62            6.0            1.9997       18.0        5.0
89     6.8  67.76           18.0            1.5000       59.0       13.0
110    9.1  49.55           19.0            1.7272       76.0       13.0
184    5.2  84.46           19.0            1.2666       78.0       15.0
202    2.3  94.30           11.0            1.2222       38.0        9.0

     num_words  num_unique_words
85         3.0               3.0
89        12.0              10.0
110       11.0              11.0
184       15.0              15.0
202        9.0               9.0
```

```
In [44]: df_train.drop([ "sent","counts"], axis=1)
```

```
Out[44]:                                                 tweet
85    "@Blackman38Tide: @WhaleLookyHere @HowdyDowdy1...
89    "@CB_Baby24: @white_thunduh alsarabsss" hes a ...
110   "@DevilGrimz: @VigxRArts you're fucking gay, b...
184   "@MarkRoundtreeJr: LMFAOOOO I HATE BLACK PEOPL...
202   "@NoChillPaz: "At least I'm not a nigger" http...
204   "@NotoriousBM95: @_WhitePonyJr_ Ariza is a sna...
219   "@RTNBA: Drakes new shoes that will be release...
260   "@TheoMaxximus: #GerrysHalloweenParty http://t...
312   "@ashlingwilde: @ItsNotAdam is bored supposed ...
315   "@bigbootybishopp: @white_thunduh lassen cc , ...
349   "@jayswaggkillah: Jackies a retard #blondeprob...
352   "@jgabsss: Stacey Dash won &#128166; http://t...
437   "Don't worry about the nigga you see, worry ab...
459   "Hey go look at that video of the man that fou...
519   "Let's kill cracker babies!". WTF did I just h...
526   "My grandma used to call me a porch monkey all...
531   "Nah its You @NoMeek_JustMilz: &#128514;&#1285...
540   "Our people". Now is the time for the Aryan ra...
565   "These sour apple bitter bitches, I'm not fuck...
582   "We hate niggers, we hate faggots and we hate ...
583   "We're out here, and we're queer!"\n" 2, 4, 6,...
587         "Who the fuck you callin jiggaboo, nigga?!"
588   "Why people think gay marriage is okay is beyo...
603             "You ain't gunna do shit spear chucker"
```

```
614        "You ol trout mouth ass bitch" \nDEEEEAAAADD
625   "ayo i even kill handicapped and crippled bitc...
635   "fuck you you pussy ass hater go suck a dick a...
646   "on my way to fuck your bitch in the name of T...
647                "poor whitey" http://t.co/3UkKeyznz8
663   #AZmonsoon lot of rain, too bad it wasn't enou...
...                                                 ...
5743                            @clangloisss hmu negro
5744  @claraoswined cute, but I love her yellow rain...
5746                               @cleggzta Ho ho ho.
5751  @cnnbrk let me guess they have yellow cake too...
5758  @contrarian11 @YouTube yeah. great song. just ...
5768  @craigcalcaterra Hes behind the plate for the ...
5779  @cryancarfield haven't you ever heard the stor...
5783  @cullenbunn Maybe Baylee Ann, Tater Pud, Nurse...
5784                 @cumkwats thank you based pee slit
5788  @cxslug I never did those games, but I was obs...
5789                   @d6_9b idk... It's trash though!
5790  @dabbba @Pinchehonkey Far as I can tell coon t...
5792  @daggerbyte @MichaelSmartGuy @spacej_me @pizza...
5794  @daishialopez how could you say such a thing w...
5802    @danifreshh some weird local ghetto school haha
5806  @danram70 pussy. It's science. Donaire via Rac...
5812                       @darealwalt_jr he see the jig
5813  @darrenburton_ and take the trash out and sepa...
5815  @datachick Humanity really needs to get past t...
5825  @dchestnut26 don't shake your head at me. I ju...
5845          @derek_gatewood it's not ghetto you fool.
5847  @desamador you gotta be like me n the twins, j...
5850  @dgardner @YoniFreedhoff Sounds like the hyste...
5855  @discordianslip I hope my fuzzy head doesnt ti...
5857  @dish \nWas just wondering if yo espect to add...
5869  @dminion25 it's kinda screwed up that they wer...
5871  @dolphrudager Nope, never taken neighbors actu...
5872  @dolphrudager Only one current Yankee sells! O...
5885  @dreadywhiteboy @JwanUrebay then the kid he so...
5893           @dustincmc @KevinKillsThngs ghetto=black

[3430 rows x 1 columns]
```

```
In [45]: for elem in [pd.DataFrame(tfidf_tr),pd.DataFrame(pos_tr),rev_tr,mention_counts_tr, of
           print(len(elem))
```

```
3430
3430
3430
3430
3430
```

```
In [0]: # x_train=np.column_stack([tfidf,pos,rev,mention_counts, other_feats])
        x_train=np.concatenate([pd.DataFrame(tfidf_tr),pd.DataFrame(pos_tr),rev_tr,mention_cou
        x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_size=0.2)
```

```
In [47]: print(len(x_train))
```

```
2744
```

```
In [48]: print(len(y_train))
```

```
2744
```

3.3 Model

To replicate the model, we passed the parameters mentioned in the paper for the classifier to see if we can reproduce the results. The parameters are L2 regularization on a LogisticRegression model after doing dimensioin reduction via feature selection.

```
In [80]: from sklearn.model_selection import StratifiedKFold, GridSearchCV
         from sklearn.pipeline import Pipeline
         from sklearn.feature_selection import SelectFromModel
         import numpy as np
         pipe = Pipeline(
                 [('select', SelectFromModel(LogisticRegression(class_weight='balanced',
                                                                penalty="l1", C=0.01))),
                 ('model', LogisticRegression(class_weight='balanced',penalty='l2'))])
         param_grid = [{}]
         grid_search = GridSearchCV(pipe,
                                    param_grid,
                                    cv=StratifiedKFold(n_splits=5,
                                                       random_state=42).split(x_train, y_train
                                    verbose=2)
         model = grid_search.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
[CV]  ...


[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)
```

```
[CV] ... , total=   0.2s
[CV]  ...


[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.2s remaining:    0.0s
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)


[CV] ... , total=   0.2s
[CV]  ...


/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)


[CV] ... , total=   0.3s
[CV]  ...


/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: Def
  "this warning.", FutureWarning)


[CV] ... , total=   0.2s
[CV]  ...


/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: Def
  FutureWarning)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)


[CV] ... , total=   0.2s


[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    1.3s finished
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:433: FutureWarning: De
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:460: FutureWarning: De
  "this warning.", FutureWarning)
```

```
In [0]: y_preds = model.predict(x_test)
```

```
In [82]: model.best_score_
```

```
Out[82]: 0.7274052478134111
```

```
In [83]: model.cv_results_
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You ar
  warnings.warn(*warn_args, **warn_kwargs)
```

```
Out[83]: {'mean_fit_time': array([0.23730369]),
          'mean_score_time': array([0.00467587]),
          'mean_test_score': array([0.72740525]),
```

```
 'mean_train_score': array([0.73651688]),
 'params': [{}],
 'rank_test_score': array([1], dtype=int32),
 'split0_test_score': array([0.72727273]),
 'split0_train_score': array([0.7461258]),
 'split1_test_score': array([0.72727273]),
 'split1_train_score': array([0.73154057]),
 'split2_test_score': array([0.70437956]),
 'split2_train_score': array([0.73907104]),
 'split3_test_score': array([0.75547445]),
 'split3_train_score': array([0.73315118]),
 'split4_test_score': array([0.72262774]),
 'split4_train_score': array([0.73269581]),
 'std_fit_time': array([0.02594034]),
 'std_score_time': array([0.00180653]),
 'std_test_score': array([0.0163644]),
 'std_train_score': array([0.00546895])}
```

```
In [84]: model.best_params_
```

```
Out[84]: {}
```

4 Results and Analaysis

5 Precison, Recall and f1-score

By definition, we know that precison is the number of datapoints that are actually true divided over the total number of datapoints predicted true, while recall is the fraction of predicted true datapoints that are actually true. From the below results we can see that precision for the "Hate" class is quite high. This could be because of the intersection of definitions of "Hate" and "Offensive" data/tweets. Thus we can see that there could be a lot of false positives associated with the "Hate" class because of which the the precision value is high.

We can further notice that the recall value for "Hate" class is the comparitively less. Thus, the fraction of those predicted true over those actually true is low. So in a sense, we have a very picky classifier, which only picks hate tweets when there is a high probability - meaning a lot of hate tweets are also misclassified as for other classes. This is one huge shortcoming of the base model. In essence, our aim was to create a high-recall-low-precision classifier, so that no hate-tweet goes unnoticed. Should we create an automated moderator, any tweets misclassified as hate can definitely be repealed on dispute.

```
In [85]: from sklearn.metrics import classification_report
         report = classification_report( y_test, y_preds )
         print(report)
```

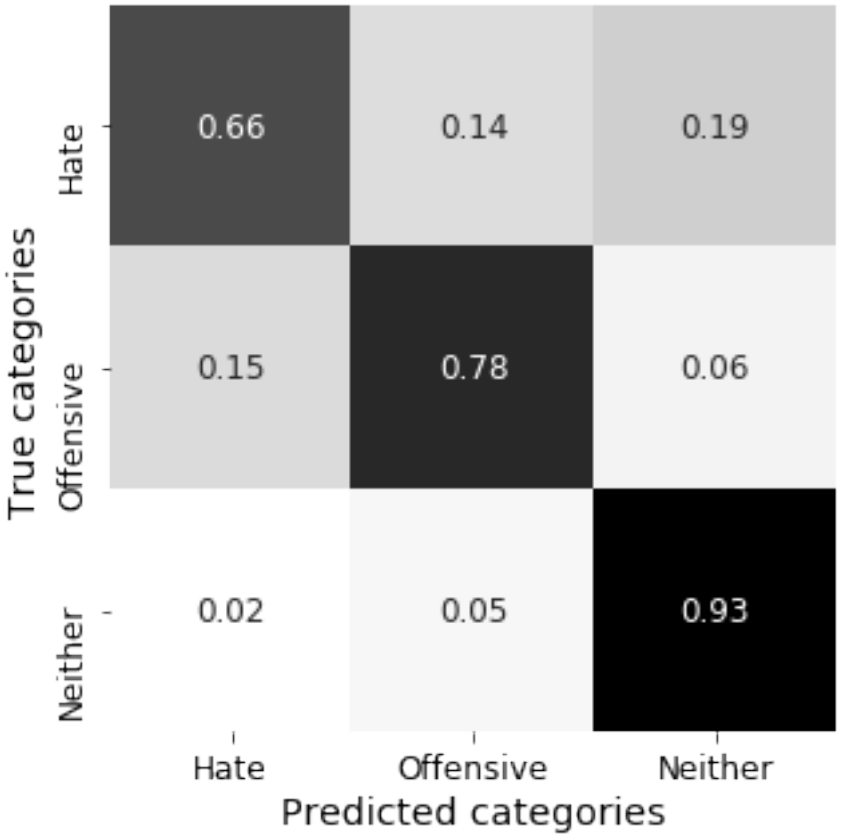
```
              precision    recall  f1-score   support

           0       0.83      0.66      0.74       270
           1       0.78      0.78      0.78       218
```

```
              2       0.74      0.93      0.82       198

    micro avg       0.78      0.78      0.78       686
    macro avg       0.78      0.79      0.78       686
weighted avg       0.79      0.78      0.78       686
```

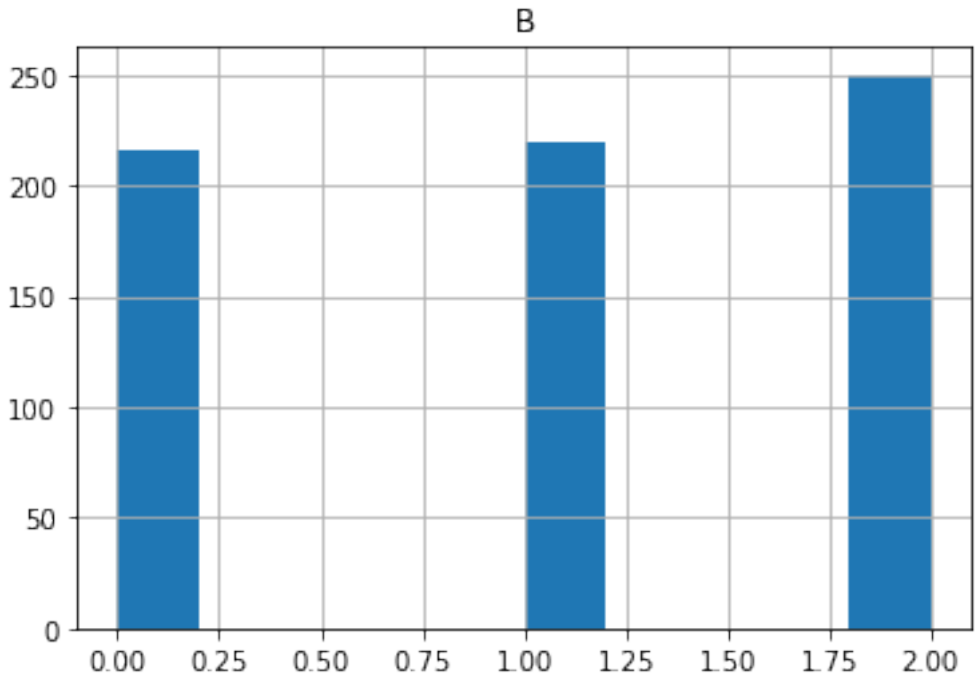
```
In [86]: from sklearn.metrics import confusion_matrix
         import seaborn
         confusion_matrix = confusion_matrix(y_test,y_preds)
         matrix_proportions = np.zeros((3,3))
         for i in range(0,3):
             matrix_proportions[i,:] = confusion_matrix[i,:]/float(confusion_matrix[i,:].sum())
         names=['Hate','Offensive','Neither']
         confusion_df = pd.DataFrame(matrix_proportions, index=names,columns=names)
         plt.figure(figsize=(5,5))
         seaborn.heatmap(confusion_df,annot=True,annot_kws={"size": 12},cmap='gist_gray_r',cba
         plt.ylabel(r'True categories',fontsize=14)
         plt.xlabel(r'Predicted categories',fontsize=14)
         plt.tick_params(labelsize=12)
```



Analyzing the confusion matrix for some interesting trends, we notice that : - 'Neither' has the highest accuracy and is rarely misclassified. - Quite understandably, most 'offensive' tweets that are misclassified are assigned to category 'hate'. This is argued in the base paper as well as one of the issues plaguing such classifier that categorize offensive tweets as hate speech. - Interestingly enough, misclassified hate tweets are more likely to be categorized as 'neither' than as 'offensive'. This is another problem that plagues classifiers that simply identify hate vs non-hate by a few key words.

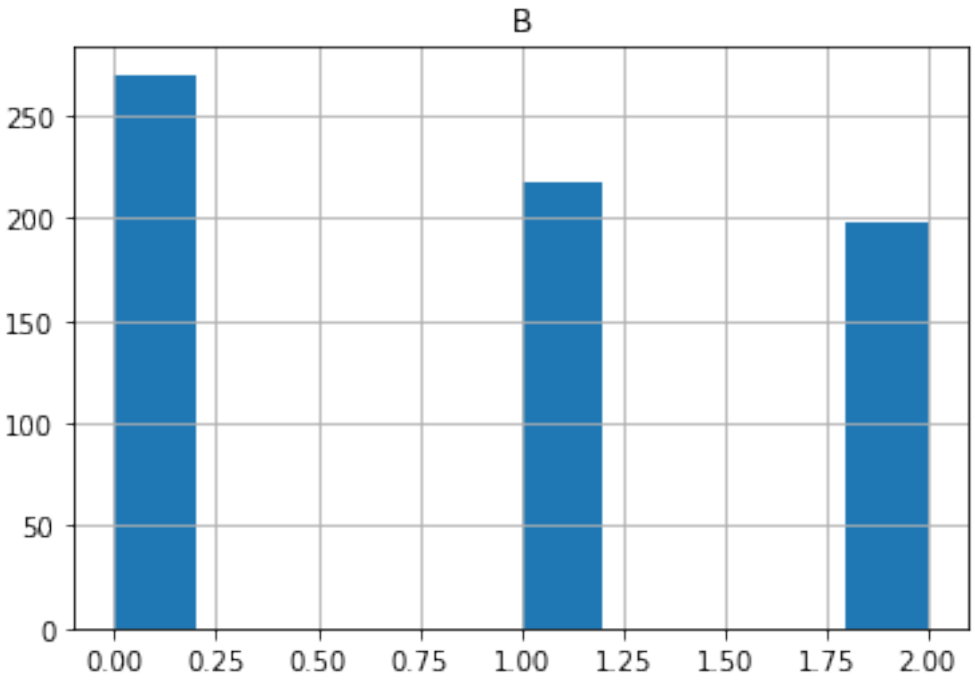
```
In [88]: from pandas import read_csv
         from matplotlib import pyplot
         import pandas as pd
         # load results file
         results = pd.DataFrame()
         results['B'] = y_preds
         # descriptive stats
         print(results.describe())
         # box and whisker plot
         # histogram
         results.hist()
         pyplot.show()
```

```
                B
count  686.000000
mean     1.049563
std      0.823305
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```




```
In [89]: from pandas import read_csv
         from matplotlib import pyplot
         import pandas as pd
         # load results file
         results = pd.DataFrame()
         results['B'] = y_test
         # descriptive stats
         print(results.describe())
         # box and whisker plot
         # histogram
         results.hist()
         pyplot.show()
```

```
                B
count  686.000000
mean     0.895044
std      0.819866
min      0.000000
25%      0.000000
50%      1.000000
75%      2.000000
max      2.000000
```



As we can observe, while the vast majority of our ground truth labels in test set are 'hate' and the other two have similar counts. This base classifier however tends to predict most observations as 'neutral'.

6 Challenges

Given the task, most of the paper was simple to replicate and we were able to do it easily. The major questions that we had to ask at each step were fundamental and weren't always answered in the paper.

Two of the issues are as follows: The authors hadn't handled data imbalance Testing was done only in sample Elaborating upon that: 1. The original data has a major class imbalance. Even the paper states that only 1.5% of the tweets were labelled as "hate speech". In all our replications, we've handled the imbalance by undersampling the data. 2. The authors did cross-validation but while generating predictions, the data passed is only the training data. There is no explicit testing data. While not challenges, they were just things that we thought could be done differently. In our replication and extension, we have changed it.

6.0.1 Was everything clearly spelled out in the paper?

The authors did clearly spell out the procedures and the paper was clear and lucid enough.However, we also had to make guesses at multiple places regarding what the authors were doing, especially regarding the data imbalance part, since they hadn't mentioned dealing with it in the paper, we assumed the results given are on imbalanced data.

6.0.2 If your results differ from those in the paper, explain how and speculate as to why.

Our results differ from the authors on multi-class classification, this is primarily because we handled data imbalance.
We obtained higher precision, accuracy than the paper. We suspect this is because the data was imbalanced in the original code, then on balancing it, since the dataset becomes much much smaller as compared to the original code, the accuracy increased.

7 Extensions

7.1 Idea

Offense is a personal thing, just because someone is offended by what is being said doesn't necessarily mean that the thing is intended to harm or hurt the person. Hate speech on the other hand is derogatory or is intended to hurt the person. Given the accuracy of the classifier, we thought it would be interesting to ultimately see how applicable the given algorithm is to a real world situation. We decided to see this in three steps:

```
1.Extend the code to another dataset
2.Use a deep learning method based on word embeddings to reduce keyword based reliance.
```

Given the small amount of data, high result metrics was very exciting. However, it also meant we should be cautious about it. One of the things was that we did to see how well the algorithm generalized was to test it on another dataset. Hence, we chose the below dataset: https://github.com/aitor-garcia-p/hate-speech-dataset and applied the author's idea. Distribution of words in the two classes

If you decide to collect new data, you may either use a pre-curated data set or build your own data set——e.g. via APIs or public web sites. We hence used the above mentioned dataset, the dataset consists of tweetids. We scrape tweets using tweepy API and find that out of the 16k given ids, only 9k are valid. Out of that, class 1 has n tweets, class 2 has m tweets. For the third extension, we also build our own dataset using youtube comments If you scrape data from a public website, make sure this is permitted and doesn't violate the site's terms of service. In either case, make sure explicitly to note details of how the data were collected and to provide code for obtaining the data. The code is available at [TwitterScraper.py]