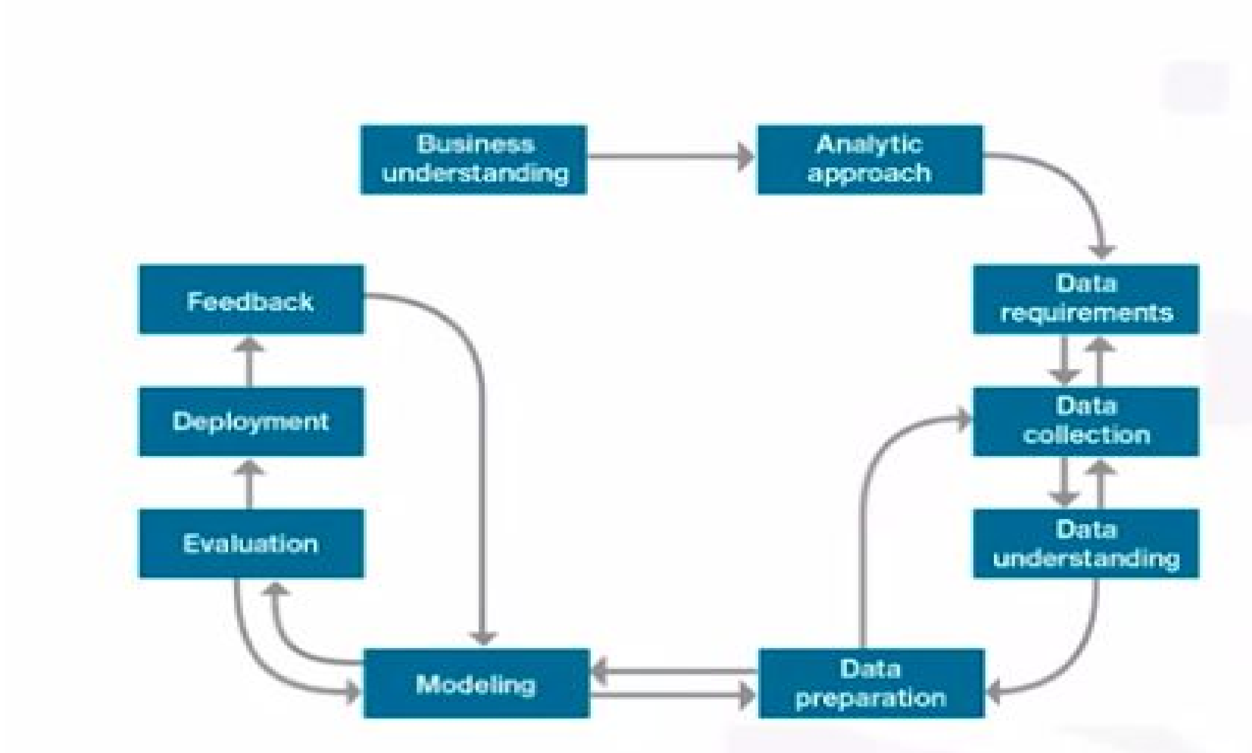
Sentiment Analysis on tweets

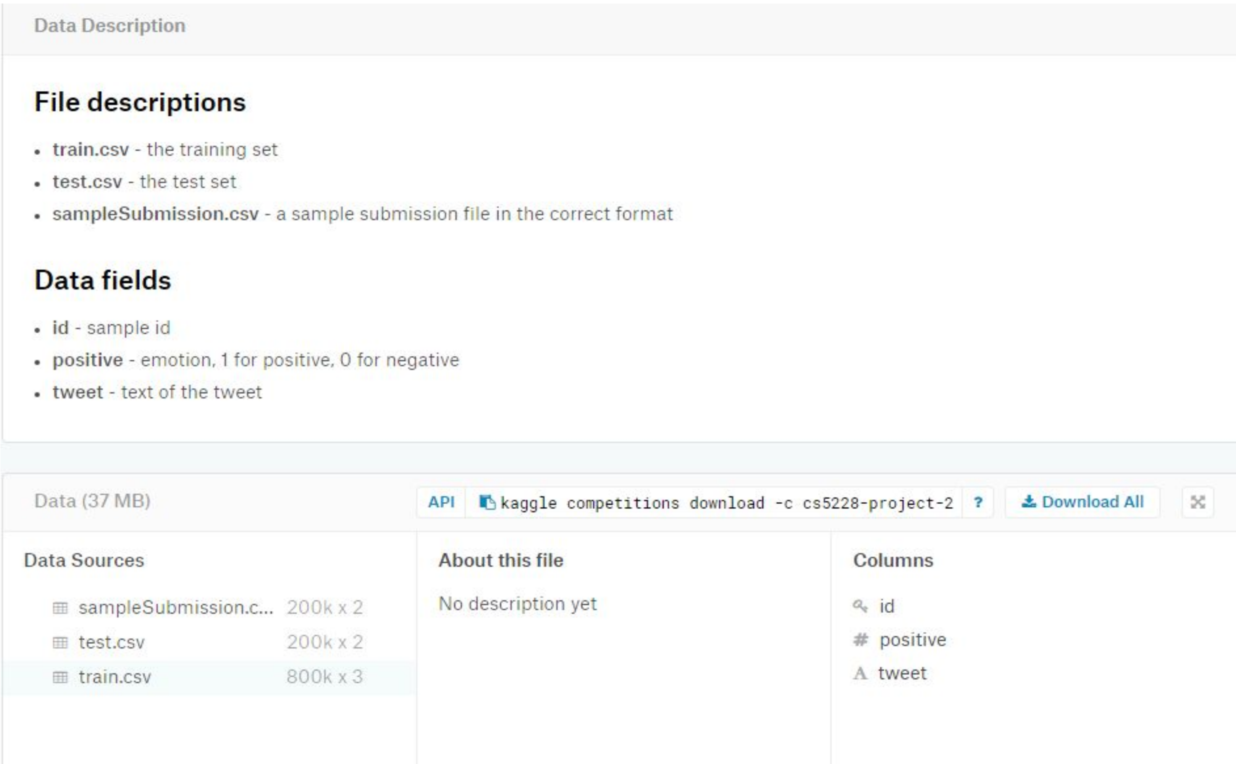
Bhuvana Chandra Atche, Yeshwanth Varada & Varun Mokarala.

1 Problem Statement

Twitter is a popular social networking website where members create and interact with messages known as “tweets”. This serves as a means for individuals to express their thoughts or feelings about different subjects. Various different parties such as consumers and marketers have done sentiment analysis on tweets to gather insights into products or to conduct market analysis. Furthermore, with the recent advancements in machine learning algorithms, we are able to improve the accuracy of our sentiment analysis predictions. In this report, we will attempt to conduct sentiment analysis on “tweets” using various different machine learning algorithms. We attempt to classify the polarity of the tweet where it is either positive or negative. If the tweet has both positive and negative elements, the more dominant sentiment should be picked as the final label.



We use the dataset from Kaggle which was crawled and labeled positive/negative. The data provided comes with emoticons, usernames and hashtags which are required to be process



ed and converted into a standard form. We also need to extract useful features from the text such unigrams and bigrams which is a form of representation of the "tweet". We use various machine learning algorithms to conduct sentiment analysis using the extracted features. However, just relying on individual models did not give a high accuracy so we pick the top few models to generate a model ensemble. Ensembling is a form of meta learning algorithm technique where we combine different classifiers in order to improve the prediction accuracy. Finally, we report our experimental results and findings at the end.

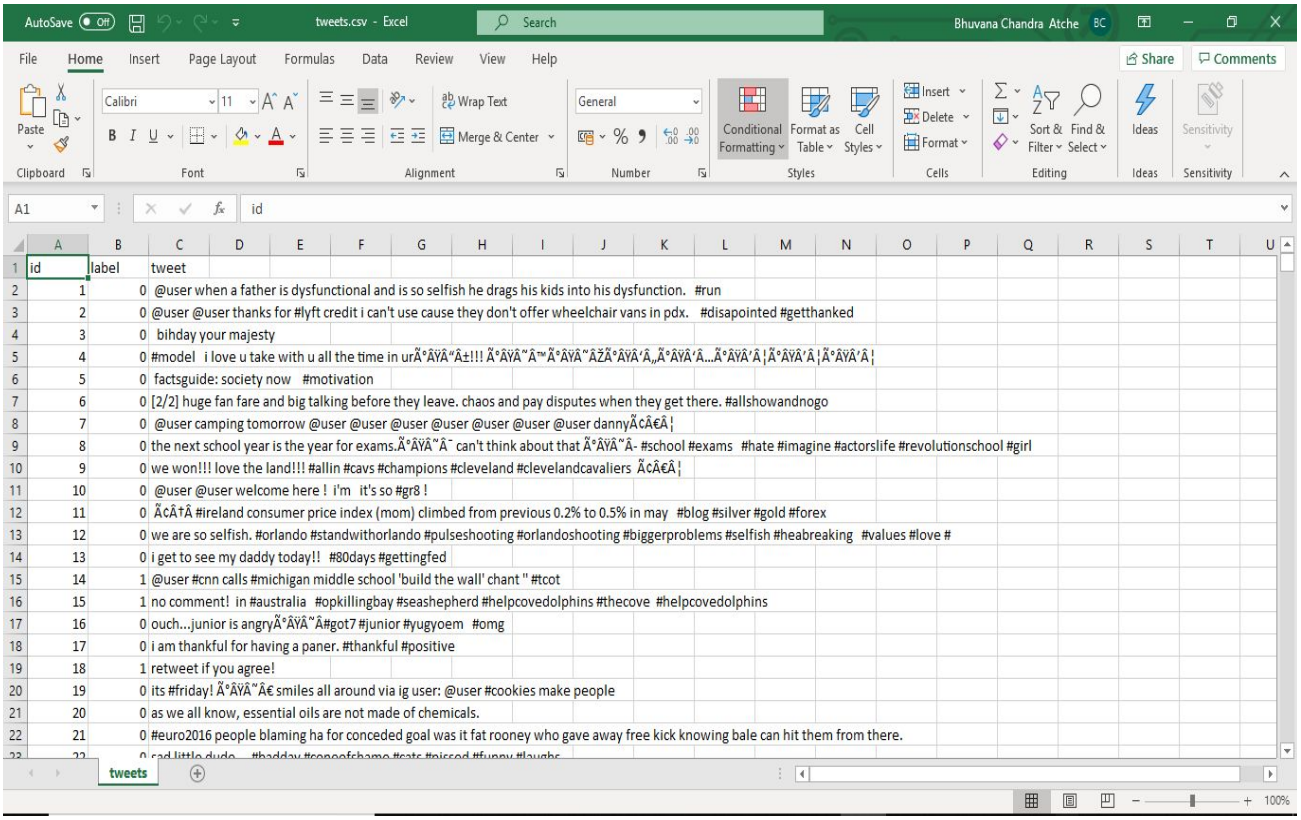
2 Data Description

The data given is in the form of a comma-separated values files with tweets and their corresponding sentiments. The training dataset is a csv file of type tweet_id,sentiment,tweet where thetweet_id is a unique integer identifying the tweet, sentiment is either 1 (positive) or 0 (negative),and tweet is the tweet enclosed in "". Similarly, the test dataset is a csv file of type tweet_id,tweet.

Sentiment Analysis on tweets

Bhuvana Chandra Atche, Yeshwanth Varada & Varun Mokarala.

Tweets.csv

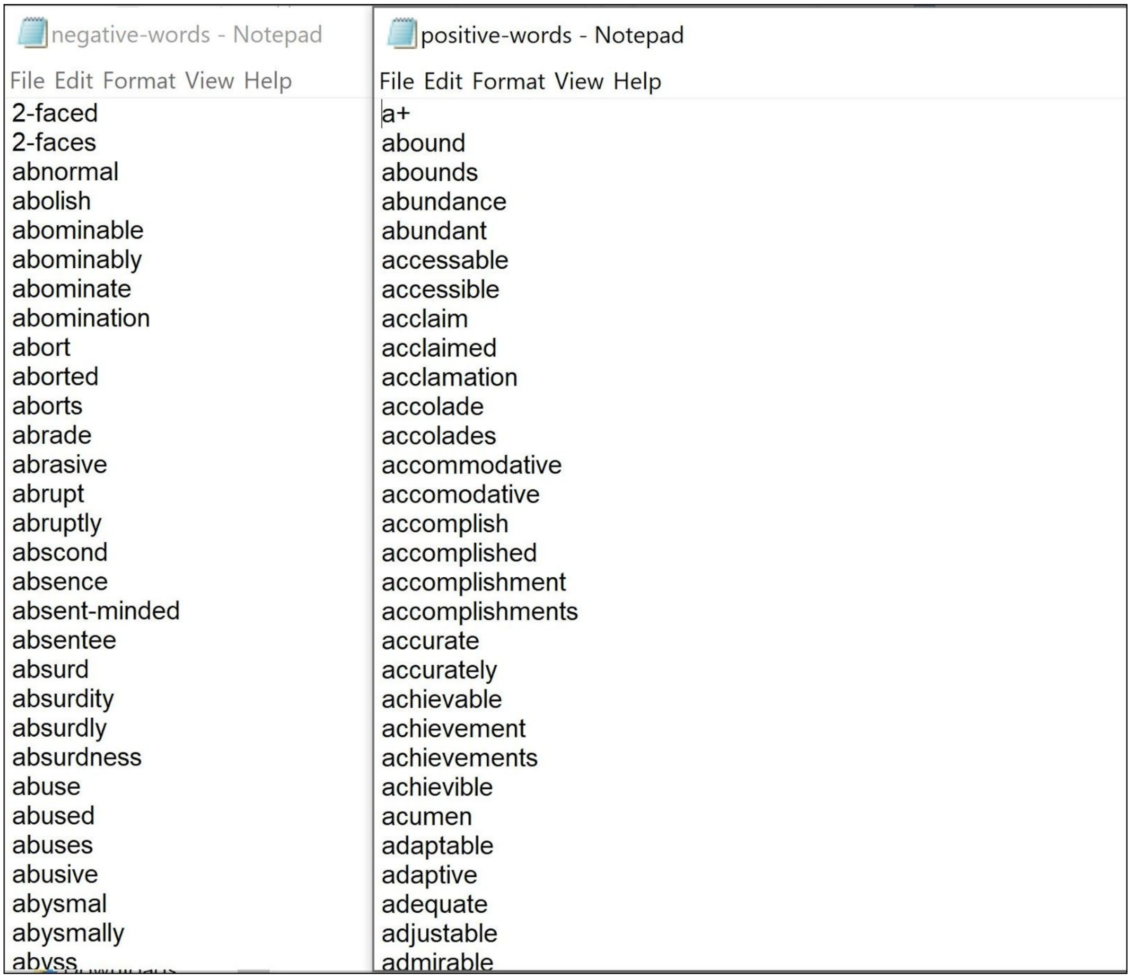


id	label	tweet
1	0	@user when a father is dysfunctional and is so selfish he drags his kids into his dysfunction. #run
2	0	@user @user thanks for #lyft credit i can't use cause they don't offer wheelchair vans in pdx. #disapointed #getthanked
3	0	bihday your majesty
4	0	#model i love u take with u all the time in urÃ°ÂŸÂ“Â±!!! Ã°ÂŸÂ˜Â™Ã°ÂŸÂ˜ÂŽÃ°ÂŸÂ‘Â„Ã°ÂŸÂ‘Â…Ã°ÂŸÂ’Â¦Ã°ÂŸÂ’Â¦Ã°ÂŸÂ’Â¦
5	0	factsguide: society now #motivation
6	0	[2/2] huge fan fare and big talking before they leave. chaos and pay disputes when they get there. #allshowandnogo
7	0	@user camping tomorrow @user @user @user @user @user @user @user dannyÃ¢Â€Â¦
8	0	the next school year is the year for exams.Ã°ÂŸÂ˜Â¯ can't think about that Ã°ÂŸÂ˜Â #school #exams #hate #imagine #actorslife #revolutionschool #girl
9	0	we won!!! love the land!!! #allin #cavs #champions #cleveland #clevelandcavaliers Ã¢Â€Â¦
10	0	@user @user welcome here ! i'm it's so #gr8 !
11	0	Ã¢Â†Â #ireland consumer price index (mom) climbed from previous 0.2% to 0.5% in may #blog #silver #gold #forex
12	0	we are so selfish. #orlando #standwithorlando #pulseshooting #orlandoshooting #biggerproblems #selfish #heabreaking #values #love #
13	0	i get to see my daddy today!! #80days #gettingfed
14	1	@user #cnn calls #michigan middle school 'build the wall' chant " #tcot
15	1	no comment! in #australia #opkillingbay #seashepherd #helpcovedolphins #thecove #helpcovedolphins
16	0	ouch...junior is angryÃ°ÂŸÂ˜Â#got7 #junior #yugyoem #omg
17	0	i am thankful for having a paner. #thankful #positive
18	1	retweet if you agree!
19	0	its #friday! Ã°ÂŸÂ˜Â€ smiles all around via ig user: @user #cookies make people
20	0	as we all know, essential oils are not made of chemicals.
21	0	#euro2016 people blaming ha for conceded goal was it fat rooney who gave away free kick knowing bale can hit them from there.

We can see that there are many user mentions, symbols, emoticons, Hashtags and urls.Which need to be processed before hand so that we have a raw set of words which is done in the following.

```
Calculating frequency distribution
Saved uni-frequency distribution to tweets-sample-freqdist.pkl
Saved bi-frequency distribution to tweets-sample-freqdist-bi.pkl

[Analysis Statistics]
Tweets => Total: 16056, Positive: 1120, Negative: 14936
User Mentions => Total: 8380, Avg: 0.5219, Max: 32
URLs => Total: 4, Avg: 0.0002, Max: 4
Emojis => Total: 340, Positive: 240, Negative: 100, Avg: 0.0212, Max: 2
Words => Total: 188864, Unique: 39672, Avg: 11.7628, Max: 116, Min: 0
Bigrams => Total: 172832, Unique: 121328, Avg: 10.7643
>>> |
Ln: 37 Col: 4
```

Positive and negative words collection in a text file.Around 10000 words in which 5000 are positive and 5000 are negative are collected and compared when labeling tweets as positive or negative.



3 Methodology and Implementation

3.1 Pre-processing

Raw tweets scraped from twitter generally result in a noisy dataset. This is due to the casual nature of people's usage of social media. Tweets have certain special characteristics such as retweets,emotions, user mentions, etc. which have to be suitably extracted. Therefore, raw twitter data has to be normalized to create a dataset which can be easily learned by various classifiers. We have applied an extensive number of pre-processing steps to standardize the dataset and reduce its size. We first do some general pre-processing on tweets which is as follows.
Convert the tweet to lower case.

Replace 2 or more dots (.) with space.
Strip spaces and quotes (" and ’) from the ends of tweet.
Replace 2 or more spaces with a single space.
We handle special twitter features as follows.

3.1.1 URL

Users often share hyperlinks to other webpages in their tweets. Any particular URL is not important for text classification as it would lead to very sparse features. Therefore, we replace all the URLs in tweets with the word URL. The regular expression used to match URLs is ((www\.[\S]+)|(https?://[\S]+)).

3.1.2 User Mention

Every twitter user has a handle associated with them. Users often mention other users in their tweets by @handle. We replace all user mentions with the word USER_MENTION. The regular expression used to match user mention is @[\S]+.

Emoticon(s) Type Regex Replacement
:), :), :-), (:, (:, (-:, :’) Smile (:\s?\)|:-\)|\(\s?:|\(-:|:\’\)) EMO_POS
:D, : D, :-D, xD, x-D, XD, X-D Laugh (:\s?D|:-D|x-?D|X-?D) EMO_POS
;-), ;), ;-D, ;D, (;, (-; Wink (:\s?\(|:-\(|\)\s?:|\)-:) EMO_POS
<3, :* Love (<3|:*) EMO_POS
:-(, : (, :(,):,)-: Sad (:\s?\(|:-\(|\)\s?:|\)-:) EMO_NEG
:,(, :’(, :"(Cry (:,\(|:\’\(|:"\() EMO_NEG
Table 3: List of emoticons matched by our method

3.1.3 Emoticon

Users often use a number of different emoticons in their tweet to convey different emotions. It is impossible to exhaustively match all the different emoticons used on social media as the number is ever increasing. However, we match some common emoticons which are used very frequently. We replace the matched emoticons with either EMO_POS or EMO_NEG depending on whether it is conveying a positive or a negative emotion. A list of all emoticons matched by our method is given in table 3.

3.1.4 Hashtag

Hashtags are unspaced phrases prefixed by the hash symbol (#) which is frequently used by users to mention a trending topic on twitter. We replace all the hashtags with the words with the hash symbol. For example, #hello is replaced by hello. The regular expression used to match hashtags is #(\S+).

3.1.5 Retweet

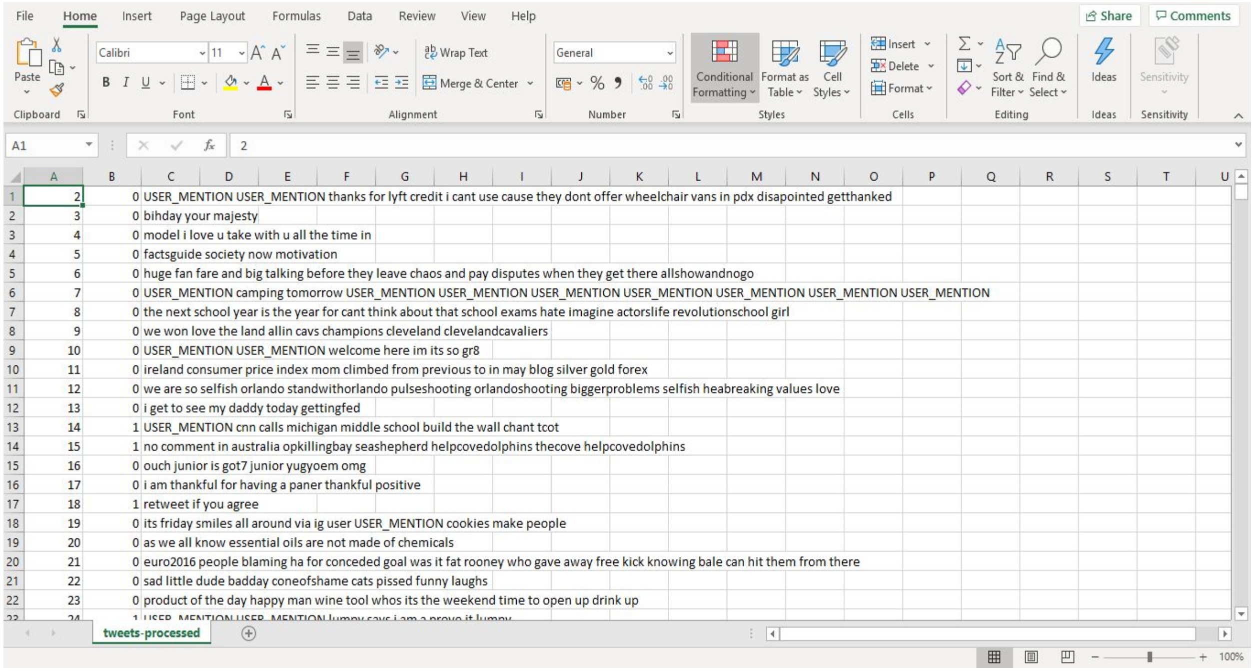
Retweets are tweets which have already been sent by someone else and are shared by other users. Retweets begin with the letters RT. We remove RT from the tweets as it is not an important feature for text classification. The regular expression used to match retweets is \brt\b.
After applying tweet level pre-processing, we processed individual words of tweets as follows.
Strip any punctuation [’"?!,.():;] from the word.
Convert 2 or more letter repetitions to 2 letters. Some people send tweets like I am sooooo happppppy adding multiple characters to emphasize on certain words. This is done to handle such tweets by converting them to I am soo happy.
Remove - and ’. This is done to handle words like t-shirt and their’s by converting them to the more general form tshirt and theirs.
Check if the word is valid and accept it only if it is. We define a valid word as a word which begins with an alphabet with successive characters being alphabets, numbers or one of dot (.) and underscore(_).

```
def preprocess_word(word):
    # Remove punctuation
    word = word.strip('\'"?!,.():;')
    # Convert more than 2 letter repetitions to 2 letter
    # funnnnny --> funny
    word = re.sub(r'(.)\1+', r'\1\1', word)
    # Remove - & '
    word = re.sub(r'(-|\')', '', word)
    return word

def is_valid_word(word):
    # Check if word begins with an alphabet
    return (re.search(r'^[a-zA-Z][a-z0-9A-Z\._]*$', word) is not None)

def handle_emojis(tweet):
    # Smile -- :), : ), :-), (:, ( :, (-:, :')
    tweet = re.sub(r'(:\s?\)|:-\)|\(\s?:|\(-:|:\'\))', ' EMO_POS ', tweet)
    # Laugh -- :D, : D, :-D, xD, x-D, XD, X-D
    tweet = re.sub(r'(:\s?D|:-D|x-?D|X-?D)', ' EMO_POS ', tweet)
    # Love -- <3, :*
    tweet = re.sub(r'(<3|:\*)', ' EMO_POS ', tweet)
```

```
    # Wink -- ;-), ;), ;-D, ;D, (;,  (-;
    tweet = re.sub(r'(;-?\)|;-?D|\(-?;)', ' EMO_POS ', tweet)
    # Sad -- :-(, : (, :(, ):, )-:
    tweet = re.sub(r'(:\s?\(|:-\(|\)\s?:|\)-:)', ' EMO_NEG ', tweet)
    # Cry -- :,(, :'(, :"(
    tweet = re.sub(r'(:,\(|:\'\(|:"\()', ' EMO_NEG ', tweet)
    return tweet

def preprocess_tweet(tweet):
    processed_tweet = []
    # Convert to lower case
    tweet = tweet.lower()
    # Replaces URLs with the word URL
    tweet = re.sub(r'((www\.[\S]+)|(https?://[\S]+))', ' URL ', tweet)
    # Replace @handle with the word USER_MENTION
    tweet = re.sub(r'@[\S]+', 'USER_MENTION', tweet)
    # Replaces #hashtag with hashtag
    tweet = re.sub(r'#(\S+)', r' \1 ', tweet)
    # Remove RT (retweet)
    tweet = re.sub(r'\brt\b', '', tweet)
    # Replace 2+ dots with space
    tweet = re.sub(r'\.{2,}', ' ', tweet)
    # Strip space, " and ' from tweet
    tweet = tweet.strip(' "\'')
    # Replace emojis with either EMO_POS or EMO_NEG
    tweet = handle_emojis(tweet)
    # Replace multiple spaces with a single space
    tweet = re.sub(r'\s+', ' ', tweet)
    words = tweet.split()
    for word in words:
    word = preprocess_word(word)
    if is_valid_word(word):
    if use_stemmer:
        word = str(porter_stemmer.stem(word))
    processed_tweet.append(word)
    return ' '.join(processed_tweet)
```

Sentiment Analysis on tweets

Bhuvana Chandra Atche, Yeshwanth Varada & Varun Mokarala.



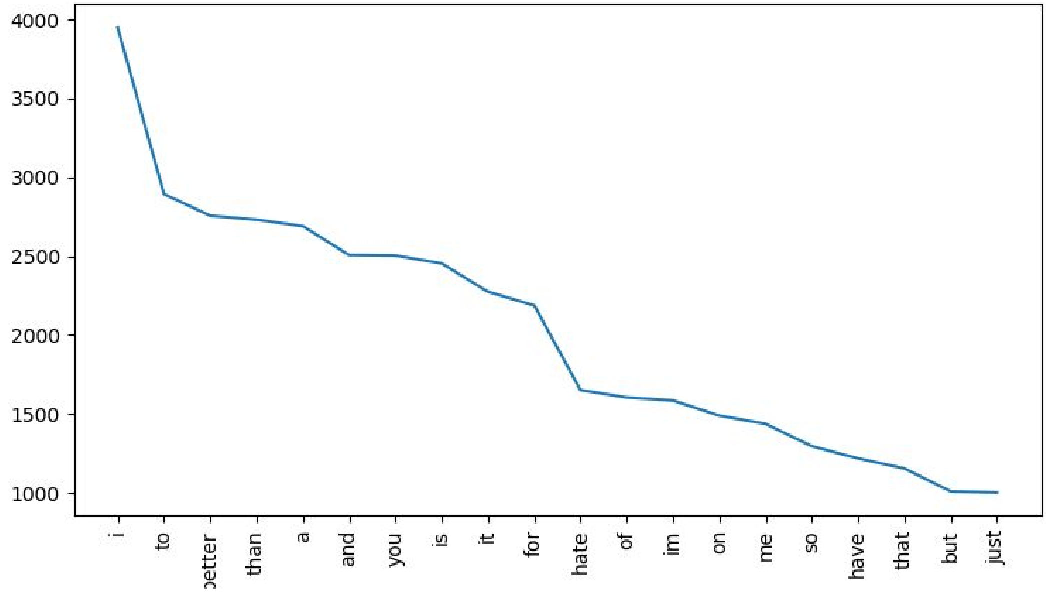
	A	B	C
1	2	0	USER_MENTION USER_MENTION thanks for lyft credit i cant use cause they dont offer wheelchair vans in pdx disapointed getthanked
2	3	0	bihday your majesty
3	4	0	model i love u take with u all the time in
4	5	0	factsguide society now motivation
5	6	0	huge fan fare and big talking before they leave chaos and pay disputes when they get there allshowandnogo
6	7	0	USER_MENTION camping tomorrow USER_MENTION USER_MENTION USER_MENTION USER_MENTION USER_MENTION USER_MENTION USER_MENTION
7	8	0	the next school year is the year for cant think about that school exams hate imagine actorslife revolutionschool girl
8	9	0	we won love the land allin cavs champions cleveland clevelandcavaliers
9	10	0	USER_MENTION USER_MENTION welcome here im its so gr8
10	11	0	ireland consumer price index mom climbed from previous to in may blog silver gold forex
11	12	0	we are so selfish orlando standwithorlando pulseshooting orlandoshooting biggerproblems selfish heabreaking values love
12	13	0	i get to see my daddy today gettingfed
13	14	1	USER_MENTION cnn calls michigan middle school build the wall chant tcot
14	15	1	no comment in australia opkillingbay seashepherd helpcovedolphins thecove helpcovedolphins
15	16	0	ouch junior is got7 junior yugyoem omg
16	17	0	i am thankful for having a paner thankful positive
17	18	1	retweet if you agree
18	19	0	its friday smiles all around via ig user USER_MENTION cookies make people
19	20	0	as we all know essential oils are not made of chemicals
20	21	0	euro2016 people blaming ha for conceded goal was it fat rooney who gave away free kick knowing bale can hit them from there
21	22	0	sad little dude badday coneofshame cats pissed funny laughs
22	23	0	product of the day happy man wine tool whos its the weekend time to open up drink up

3.2 Feature Extraction

We extract two types of features from our dataset, namely unigrams and bigrams. We create a frequency distribution of the unigrams and bigrams present in the dataset and choose top N unigrams and bigrams for our analysis.

3.2.1 Unigrams

Probably the simplest and the most commonly used features for text classification is the presence of single words or tokens in the text. We extract single words from the training dataset and create a frequency distribution of these words. A total of 39672 words are extracted from the dataset. Out of these words, most of the words at the end of the frequency spectrum are noise and occur very few times to influence classification. We, therefore, only use top N words from these to create our vocabulary where N is 9672 for sparse vector classification and 30000 for dense vector classification. The frequency distribution of top 20 words in our vocabulary is shown in figure .



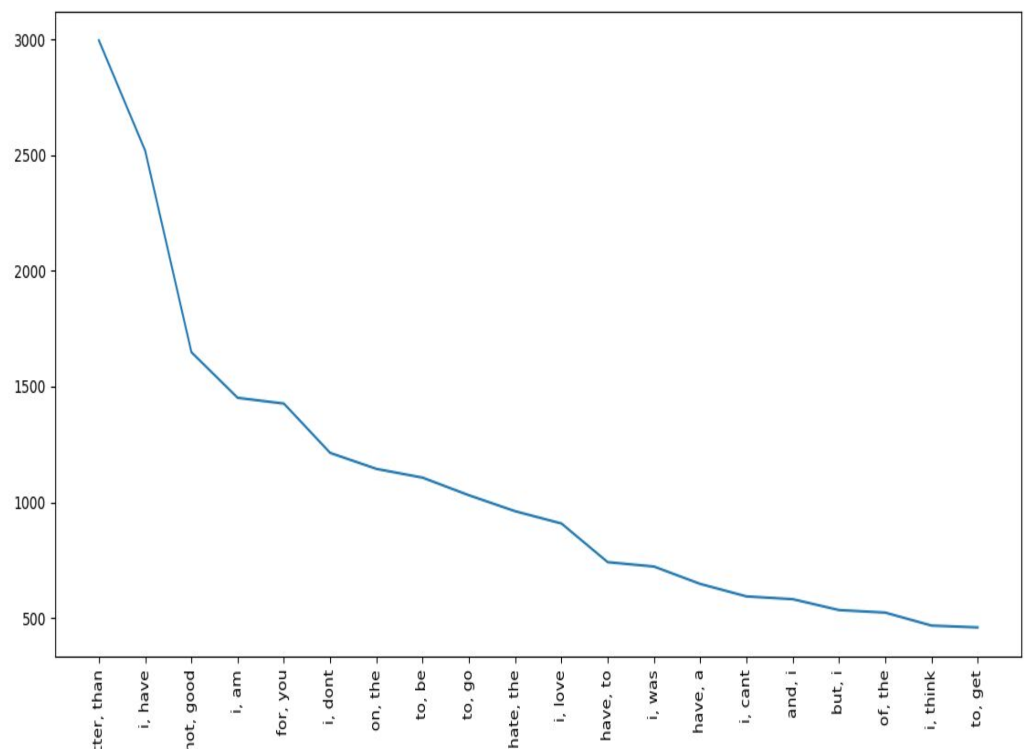
3.2.2 Bigrams

Bigrams are word pairs in the dataset which occur in succession in the corpus. These features are a good way to model negation in natural language like in the phrase – This is not good. A total of 1372832 unique bigrams were extracted from the dataset. Out of these, most of the bigrams at end of frequency spectrum are noise and occur very few times to influence classification. We therefore use only top 10000 bigrams from these to create our vocabulary.

```
def extract_features(tweets, batch_size=500, test_file=True, feat_type='presence'):
    num_batches = int(np.ceil(len(tweets) / float(batch_size)))
    for i in xrange(num_batches):
    batch = tweets[i * batch_size: (i + 1) * batch_size]
    features = lil_matrix((batch_size, VOCAB_SIZE))
    labels = np.zeros(batch_size)
    for j, tweet in enumerate(batch):
    if test_file:
        tweet_words = tweet[1][0]
        tweet_bigrams = tweet[1][1]
    else:
        tweet_words = tweet[2][0]
        tweet_bigrams = tweet[2][1]
        labels[j] = tweet[1]
```

```
if feat_type == 'presence':
    tweet_words = set(tweet_words)
    tweet_bigrams = set(tweet_bigrams)
for word in tweet_words:
    idx = unigrams.get(word)
    if idx:
    features[j, idx] += 1
if USE_BIGRAMS:
    for bigram in tweet_bigrams:
    idx = bigrams.get(bigram)
    if idx:
    features[j, UNIGRAM_SIZE + idx] += 1
yield features, labels
```

Distribution of top 20 bigrams is as follows:



3.3 Vector Representation

3.3.1 Sparse Vector Representation

Depending on whether or not we are using bigram features, the sparse vector representation of each tweet is either of length 5000 (when considering only unigrams) or 15000 (when considering unigrams and bigrams). Each unigram (and bigram) is given a unique index depending on its rank. The feature vector for a tweet has a positive value at the indices of unigrams (and bigrams) which are present in that tweet and zero elsewhere which is why the vector is sparse. The positive value at the indices of unigrams (and bigrams) depends on the feature type we specify which is one of presence and frequency. • presence In the case of presence feature type, the feature vector has a 1 at indices of unigrams (and bigrams) present in a tweet and 0 elsewhere. • frequency In the case of frequency feature type, the feature vector has a positive integer at indices of unigrams (and bigrams) which is the frequency of that unigram (or bigram) in the tweet and 0 elsewhere. A matrix of such term-frequency vectors is constructed for the entire training dataset and then each term frequency is scaled by the inverse-document-frequency of the term (idf) to assign higher values to important terms. The inverse-document-frequency of a term t is defined as. idf(t) = log 1 + nd 1 + df(d,t)+ 1 where nd is the total number of documents and df(d,t) is the number of documents in which the term t occurs. Handling Memory Issues Which dealing with sparse vector representations, the feature vector for each tweet is of length 2000 and the total number of tweets in the training set is 16000 which means allocation of memory for a matrix of size 16000×2000. Assuming 4 bytes are required to represent each float value in the matrix, this matrix needs a memory of 8×1024 bytes (≈ 2 GB) which is far greater than the memory available in common notebooks. To tackle this issue, we used scipy.sparse.lil_matrix data structure provided by Scipy which is a memory efficient linked list based implementation of sparse matrices. In addition to that, we used Python generators wherever possible instead of keeping the entire dataset in memory.

3.3.2 Dense Vector Representation

For dense vector representation we use a vocabulary of unigrams of size 10000 i.e. the top 10000 words in the dataset. We assign an integer index to each word depending on its rank (starting from 1) which means that the most common word is assigned the number 1, the second most common word is assigned the number 2 and so on. Each tweet is then represented by a vector of these indices which is a dense vector.

```
def get_feature_vector(tweet):
        uni_feature_vector = []
        bi_feature_vector = []
```

```
words = tweet.split()
for i in xrange(len(words) - 1):
word = words[i]
next_word = words[i + 1]
if unigrams.get(word):
uni_feature_vector.append(word)
if USE_BIGRAMS:
if bigrams.get((word, next_word)):
        bi_feature_vector.append((word, next_word))
if len(words) >= 1:
if unigrams.get(words[-1]):
uni_feature_vector.append(words[-1])
return uni_feature_vector, bi_feature_vector
```

3.4 Classifiers

3.4.1 Naive Bayes

Naive Bayes is a simple model which can be used for text classification. In this model, the class ˆ c is assigned to a tweet t, where In the formula above, fi represents the i-th feature of total n features. P(c) and P(fi|c) can be obtained through maximum likelihood estimates.

$$\hat{c} = \underset{c}{argmax}\ \mathrm{P}(\mathrm{c}|\mathrm{t})$$

$$\mathrm{P}(\mathrm{c}|\mathrm{t}) \propto \mathrm{P}(\mathrm{c}) \prod_{i=1}^{n} \mathrm{P}(f_i|\mathrm{c})$$

So what do we do? Simple! We use **word frequencies**. That is, we ignore the word order and sentence construction, treating every document as a set of the words it contains. Our features will be the counts of each of these words. Even though it may seem too simplistic an approach, it works surprisingly well.

Being Naive

So here comes the *Naive* part: we assume that every word in a sentence is **independent** of the other ones. This means that we're no longer looking at entire sentences, but rather at individual words. So for our purposes, "this was a fun party" is the same as "this party was fun" and "party fun was this".

We write this as:

$$P(a\,very\,close\,game) = P(a) \times P(very) \times P(close) \times P(game)$$

This assumption is very strong but super useful. It's what makes this model work well with little data or data that may be mislabeled. The next step is just applying this to what we had before:

$$P(a\,very\,close\,game|Sports) = P(a|Sports) \times P(very|Sports) \times P(close|Sports) \times P(game|Sports)$$

And now, all of these individual words actually show up several times in our training data, and we can calculate them!

```
np.random.seed(1337)
    unigrams = utils.top_n_words(FREQ_DIST_FILE, UNIGRAM_SIZE)
    if USE_BIGRAMS:
    bigrams = utils.top_n_bigrams(BI_FREQ_DIST_FILE, BIGRAM_SIZE)
    tweets = process_tweets(TRAIN_PROCESSED_FILE, test_file=False)
    if TRAIN:
    train_tweets, val_tweets = utils.split_data(tweets)
    else:
    random.shuffle(tweets)
    train_tweets = tweets
    del tweets
    print 'Extracting features & training batches'
    clf = MultinomialNB()
    batch_size = len(train_tweets)
    i = 1
    n_train_batches = int(np.ceil(len(train_tweets) / float(batch_size)))
    for training_set_X, training_set_y in extract_features(train_tweets, test_file=False,
feat_type=FEAT_TYPE, batch_size=batch_size):
    utils.write_status(i, n_train_batches)
    i += 1
    if FEAT_TYPE == 'frequency':
    tfidf = apply_tf_idf(training_set_X)
    training_set_X = tfidf.transform(training_set_X)
```

```
clf.partial_fit(training_set_X, training_set_y, classes=[0, 1])
print '\n'
print 'Testing'
if TRAIN:
correct, total = 0, len(val_tweets)
i = 1
batch_size = len(val_tweets)
n_val_batches = int(np.ceil(len(val_tweets) / float(batch_size)))
for val_set_X, val_set_y in extract_features(val_tweets, test_file=False,
feat_type=FEAT_TYPE, batch_size=batch_size):
if FEAT_TYPE == 'frequency':
    val_set_X = tfidf.transform(val_set_X)
prediction = clf.predict(val_set_X)
correct += np.sum(prediction == val_set_y)
utils.write_status(i, n_val_batches)
i += 1
print '\nCorrect: %d/%d = %.4f %%' % (correct, total, correct * 100. / total)
else:
del train_tweets
test_tweets = process_tweets(TEST_PROCESSED_FILE, test_file=True)
n_test_batches = int(np.ceil(len(test_tweets) / float(batch_size)))
predictions = np.array([])
print 'Predicting batches'
i = 1
for test_set_X, _ in extract_features(test_tweets, test_file=True,
feat_type=FEAT_TYPE):
if FEAT_TYPE == 'frequency':
    test_set_X = tfidf.transform(test_set_X)
prediction = clf.predict(test_set_X)
predictions = np.concatenate((predictions, prediction))
utils.write_status(i, n_test_batches)
i += 1
predictions = [(str(j), int(predictions[j]))
    for j in range(len(test_tweets))]
utils.save_results_to_csv(predictions, 'naivebayes.csv')
print '\nSaved to naivebayes.csv'
```

```
Extracting features & training batches
Processing 1/1

Testing
Processing 1/1
 Accuracy: 2919/581 = 83.4%
```

4 Conclusion

4.1 Summary of achievements

The provided tweets were a mixture of words, emoticons, URLs, hastags, user mentions, and symbols. Before training the we pre-process the tweets to make it suitable for feeding into models. We implemented several machine learning algorithms like Naive Bayes and Decision Tree to classify the polarity of the tweet. We used two types of features namely unigrams and bigrams for classification and observes that augmenting the feature vector with bigrams improved the accuracy. Once the feature has been extracted it was represented as either a sparse vector or a dense vector. It has been observed that presence in the sparse vector representation recorded a better performance than frequency.
Our model achieved an accuracy of 83.4% on Kaggle dataset.

4.2 Future directions

Handling emotion ranges: We can improve and train our models to handle a range of sentiments. Tweets don’t always have positive or negative sentiment. At times they may have no sentiment i.e. neutral. Sentiment can also have gradations like the sentence, This is good, is positive but the sentence, This is extraordinary. is somewhat more positive than the first. We can therefore classify the sentiment in ranges, say from -2 to +2.
Using symbols: During our pre-processing, we discard most of the symbols like commas, full-stops, and exclamation mark. These symbols may be helpful in assigning sentiment to a sentence.