Delivery #6 - Final Presentation

Social Media Sentiment Analysis

Group #3
AKASH SG (Information science)
SUDHANSHU (Information science)
INDRESH MJ (information science)
PREETHAM KM (information science)
SHASHANKK (information science)

Project Delivery #1

Dataset

Target	Ids	Date	Flag	User	Text
0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOn	@switchfoot http://twitpic.com/2y1zl - Awww, tl
0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by text
0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Mana
0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all. i'm
0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew
0	1467811592	Mon Apr 06 22:20:03 PDT 2009	NO_QUERY	mybirch	Need a hug
0	1467811594	Mon Apr 06 22:20:03 PDT 2009	NO_QUERY	coZZ	@LOLTrish hey long time no see! Yes.. Rains a bit
0	1467811795	Mon Apr 06 22:20:05 PDT 2009	NO_QUERY	2Hood4Hollyw	@Tatiana_K nope they didn't have it
0	1467812025	Mon Apr 06 22:20:09 PDT 2009	NO_QUERY	mimismo	@twittera que me muera ?

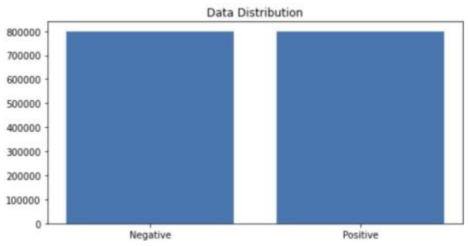
Figure 1. A sample from the dataset.

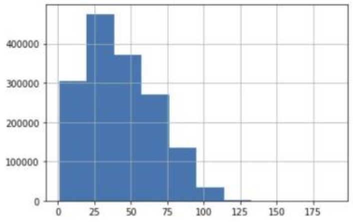
The dataset contains 1,600,000 tweets extracted using the twitter api. The tweets have been classified from 0 (negative) to 4 (positive). The dataset contains 6 fields which are target as integer, ids as integer, date as date, flag as string, user as string and text as string.
These 6 fields are shown below.

- target: The polarity of the tweet (0 - negative, 2 - neutral, 4 - positive)
- ids: The id of the tweet.
- date: The date of the tweet.
- flag: The query. If there is no query, then this value is NO_QUERY.
- user: The user that tweeted.
- text: The text of the tweet

	label	tweet
0	Negative	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	Negative	is upset that he can't update his Facebook by ...
2	Negative	@Kenichan I dived many times for the ball. Man...
3	Negative	my whole body feels itchy and like its on fire
4	Negative	@nationwideclass no, it's not behaving at all....
...	...	...
1599995	Positive	Just woke up. Having no school is the best fee...
1599996	Positive	TheWDB.com - Very cool to hear old Walt interv...
1599997	Positive	Are you ready for your MoJo Makeover? Ask me f...
1599998	Positive	Happy 38th Birthday to my boo of alll time!!! ...
1599999	Positive	happy #charitytuesday @theNSPCC @SparksCharity...

Figure 2. Dataset after reduction.






```
count    1.592328e+06
mean     4.279740e+01
std      2.415896e+01
min      1.000000e+00
25%      2.300000e+01
50%      3.900000e+01
75%      6.000000e+01
max      1.890000e+02
dtype: float64
```

Figure 3. Data distribution.

Project Delivery #2-3
Exploring Your Data

Exploring Our Data in terms of Letter Frequency

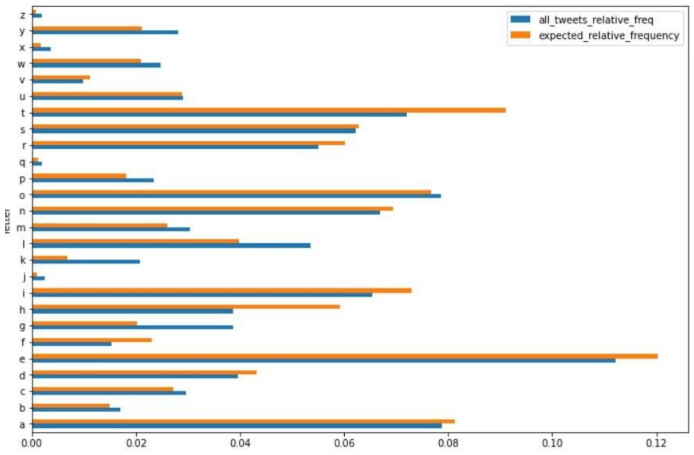


Figure 4. Letter frequencies of each 26 characters in English Alphabet.

	letter	frequency	all_tweets_relative_freq	expected_relative_frequency	expected
0	a	4547601	0.078816	0.081238	4687379.0
1	b	975326	0.016904	0.014893	859300.0
2	c	1705409	0.029557	0.027114	1564464.0
3	d	2289515	0.039680	0.043192	2492128.0
4	e	6471295	0.112156	0.120195	6935169.0
5	f	878849	0.015232	0.023039	1329304.0
6	g	2231747	0.038679	0.020257	1168838.0
7	h	2234047	0.038719	0.059215	3416628.0
8	i	3779579	0.065505	0.073054	4215160.0
9	j	143817	0.002493	0.001031	59502.0
10	k	1197291	0.020751	0.006895	397842.0
11	l	3095498	0.053649	0.039785	2295581.0
12	m	1754377	0.030406	0.026116	1506861.0
13	n	3861185	0.066919	0.069478	4008801.0
14	o	4534414	0.078587	0.076812	4431963.0
15	p	1351301	0.023420	0.018189	1049517.0
16	q	115059	0.001994	0.001125	64883.0
17	r	3179237	0.055100	0.060213	3474231.0
18	s	3595565	0.062316	0.062808	3623936.0
19	t	4153946	0.071993	0.090986	5249801.0
20	u	1676743	0.029060	0.028776	1660364.0
21	v	566733	0.009822	0.011075	639015.0
22	w	1422401	0.024652	0.020949	1208717.0
23	x	203131	0.003521	0.001728	99698.0
24	y	1620980	0.028094	0.021135	1219478.0
25	z	114027	0.001976	0.000702	40512.0

Figure 5. Letter frequency of the dataset, relative frequencies of the dataset, expected relative frequency according to the English language and expected character length according to the English language.

	frequency	expected
frequency	1.000000	0.967421
expected	0.967421	1.000000

Figure 6. Correlation

We got the p-value (p) as 0 which implies that the letter frequency does not follow the same distribution with what we see in English tests, although the Pearson correlation is too high (~96.7%)

.

Exploring Our Data in terms of Letter Frequency

We counted the number of characters for each tweet and analyzed the data frame according to maximum number of characters, minimum number of characters, mean of the number of characters column and its standard deviation. Our longest tweet is 189 characters long, the shortest tweet is 1 character long and mean of all tweets' character length 42.78. The standard deviation of all tweet character length is 24.16 as shown in Figure 8.

	label	tweet	number_of_characters
0	Negative	awww bummer shoulda got david carr third day	44
1	Negative	upset update facebook texting might cry result...	69
2	Negative	dived many times ball managed save 50 rest go ...	52
3	Negative	whole body feels itchy like fire	32
4	Negative	behaving mad see	16
...	...	...	...
1599995	Positive	woke school best feeling ever	29
1599996	Positive	thewdb com cool hear old walt interviews	40
1599997	Positive	ready mojo makeover ask details	31
1599998	Positive	happy 38th birthday boo alll time tupac amaru ...	52
1599999	Positive	happy charitytuesday thenspcc sparkscharity sp...	57

Figure 7. Number of characters.

```
df1.number_of_characters.max()
```

```
189
```

```
df1.number_of_characters.min()
```

```
1
```

```
df1.number_of_characters.mean()
```

```
42.7974010379771
```

```
df1.number_of_characters.std()
```

```
24.158961650697616
```

Figure 8. Letter frequency of the dataset, relative frequencies of the dataset, expected relative frequency according to the English language and expected character length according to the English language.

Exploring Our Data in terms of Word Frequency

We counted the number of words for each tweet and analyzed the data frame according to maximum number of words, minimum number of words, mean of the number of words column and its standard deviation. Our longest tweet is 50 words long, the shortest tweet is 1 word long and the mean of all tweets' word length is 7.24. The standard deviation of all tweet character length is 4.03 as shown in Figure 10.

	label	tweet	number_of_characters	number_of_words
0	Negative	awww bummer shoulda got david carr third day	44	8
1	Negative	upset update facebook texting might cry result...	69	11
2	Negative	dived many times ball managed save 50 rest go ...	52	10
3	Negative	whole body feels itchy like fire	32	6
4	Negative	behaving mad see	16	3
...	...	...	...	...
1599995	Positive	woke school best feeling ever	29	5
1599996	Positive	thewdb com cool hear old walt interviews	40	7
1599997	Positive	ready mojo makeover ask details	31	5
1599998	Positive	happy 38th birthday boo alll time tupac amaru ...	52	9
1599999	Positive	happy charitytuesday thenspcc sparkscharity sp...	57	5

Figure 9. Number of words of each tweet.

```
df1.number_of_words.max()
```

```
50
```

```
df1.number_of_words.min()
```

```
1
```

```
df1.number_of_words.mean()
```

```
7.244474128445898
```

```
df1.number_of_words.std()
```

```
4.030421805719796
```

Figure 10. Max, min, mean and standard deviation of each tweet in terms of number of words.

Exploring Our Data in terms of Word Frequency

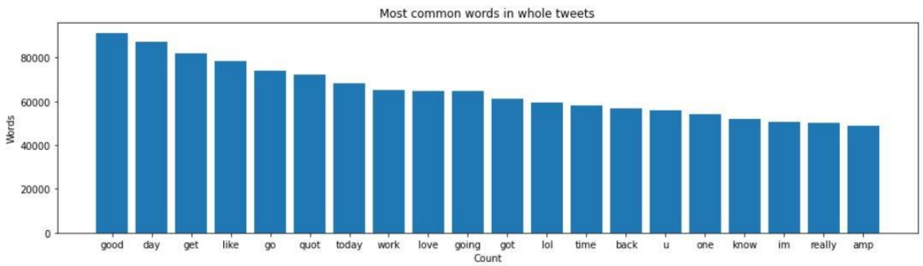


Figure 11. Most common words in whole data.

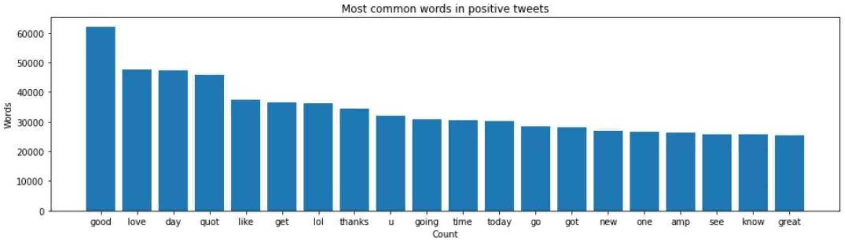


Figure 12. Most common words in positive tweets.

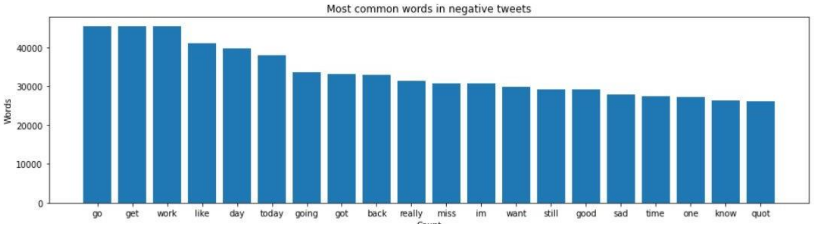


Figure 13. Most common words in positive tweets.

Embedding - Glove

We can train the embedding ourselves. However, that approach can take a long time to train. So, we use transfer learning technique, and we use GloVe: Global Vectors for Word Representation.

The Global Vectors for Word Representation, or GloVe, algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford. It 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.

We download the GloVe. Then, we initialize an embedding index that has 400000 word vectors, and embedding matrix.

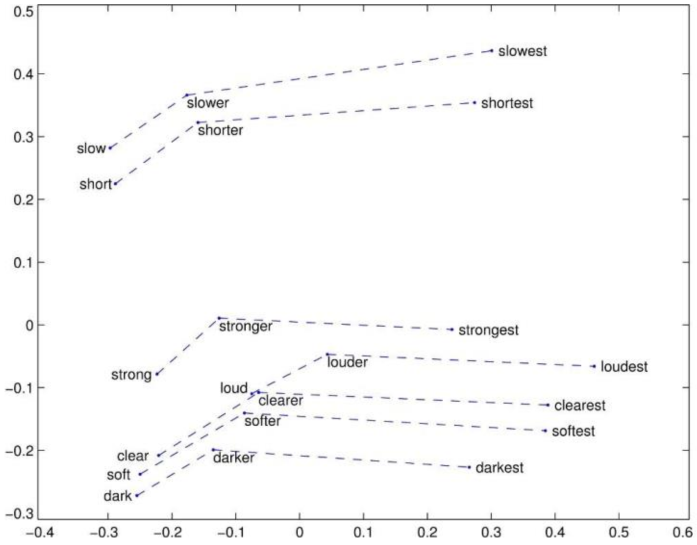


Figure 14. GloVe embedding example

Feature Extraction - Scatter Plot

We used feature extraction methods, bag-of-words, and word embedding. Bag of words with TF-IDF is a common and simple way of feature extraction. Bag-of-Words is a representation model of text data and TF-IDF is a calculation method to score the importance of words in a document.

After applying bag-of-words with TF-IDF, we create the scatter plot according to these results.

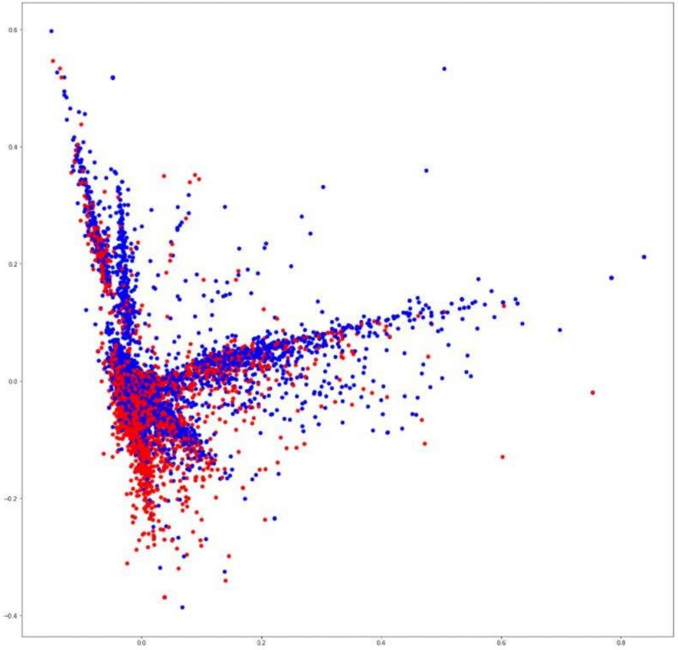


Figure 15. Scatter plot that shows correlation of words in the corpus: red indicates negatives, blue indicates positives.

Project Delivery #4
Predictive Analysis

Entropy

At the beginning, our dataset had 6 features which were target, id, date, query, user and text. We chose two of them for our purpose which are target and text. We can see that the entropy decreases significantly after this transformation.

- First entropy of dataset = 41.08269441306875
- Entropy after preprocess = 14.73368002815221

Classification/Regression

For classification/regression experiments, the test set percentage is set to be 20%.

- Total Data = Train Data (80%) + Test Data (20%)

Used 3 different algorithm and 6 different model for classification. These are :

- LSTM with 1024 Batch Size
- LSTM with 512 Batch Size
- CNN with 1024 Batch Size
- CNN with 512 Batch Size
- Multinomial Naive Bayes with Count Vectorizer
- Multinomial Naive Bayes with TF-IDF Vectorizer

LSTM Model - 1

- Batch Size = 1024

```
sequence_input = Input(shape=(30,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2))(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(sequence_input, outputs)


model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])
ReduceLROnPlateau = ReduceLROnPlateau(factor=0.1, min_lr = 0.01, monitor = 'val_loss', verbose = 1)
```

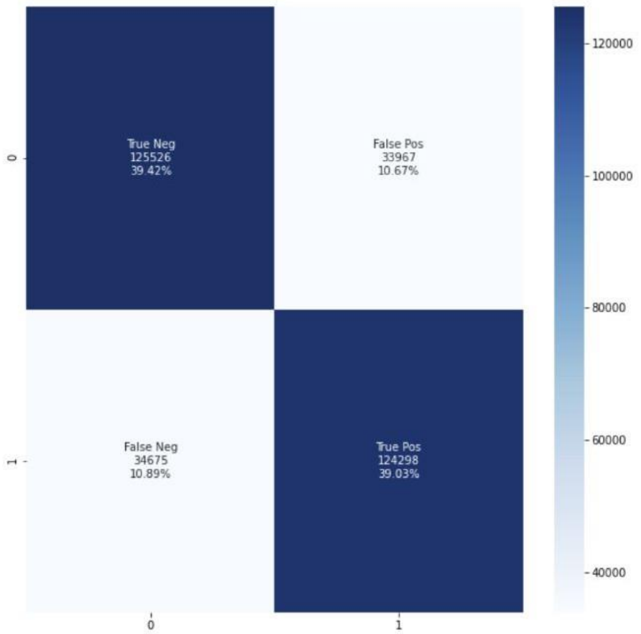
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
embedding (Embedding)        (None, 30, 300)           87137400
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 30, 300)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 26, 64)            96064
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               66048
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 87,628,729
Trainable params: 491,329
Non-trainable params: 87,137,400
_________________________________________________________________
```

Evaluation Metrics of LSTM Model - 1

LSTM Model - 1 :

	precision	recall	f1-score	support
Negative	0.78	0.79	0.79	159493
Positive	0.79	0.78	0.78	158973
accuracy			0.78	318466
macro avg	0.78	0.78	0.78	318466
weighted avg	0.78	0.78	0.78	318466

Confusion Matrix of LSTM Model - 1 :



LSTM Model - 2

- Batch Size = 512

```
sequence_input = Input(shape=(30,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = Bidirectional(LSTM(64, dropout=0.2, recurrent_dropout=0.2))(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
outputs = Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(sequence_input, outputs)


model.compile(optimizer=Adam(learning_rate=1e-3), loss='binary_crossentropy', metrics=['accuracy'])
ReduceLROnPlateau = ReduceLROnPlateau(factor=0.1, min_lr = 0.01, monitor = 'val_loss', verbose = 1)
```

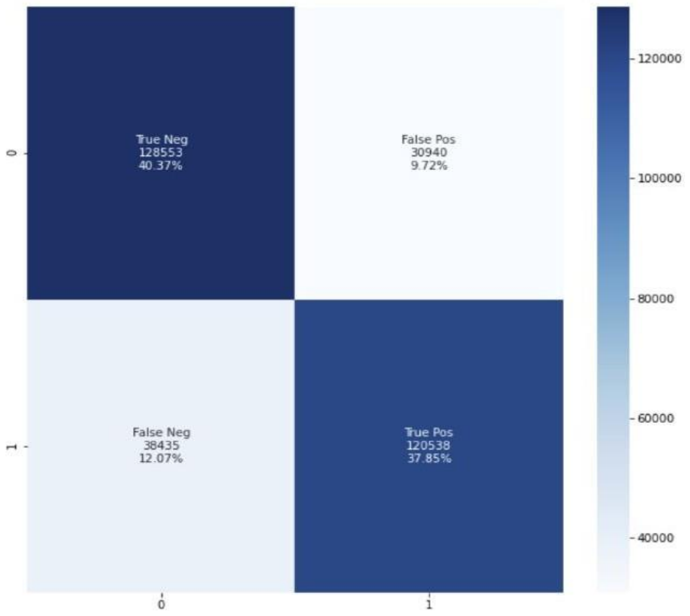
```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
embedding (Embedding)        (None, 30, 300)           87137400
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 30, 300)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 26, 64)            96064
_________________________________________________________________
bidirectional (Bidirectional (None, 128)               66048
_________________________________________________________________
dense (Dense)                (None, 512)               66048
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               262656
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 513
=================================================================
Total params: 87,628,729
Trainable params: 491,329
Non-trainable params: 87,137,400
_________________________________________________________________
```

Evaluation Metrics of LSTM Model - 2

LSTM Model - 2 :

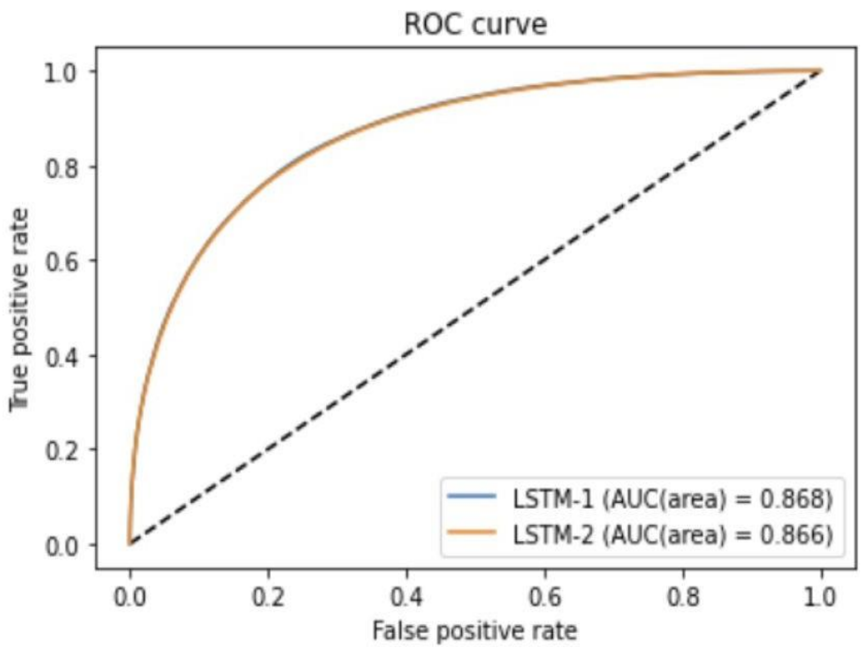
	precision	recall	f1-score	support
Negative	0.77	0.81	0.79	159493
Positive	0.80	0.76	0.78	158973
accuracy			0.78	318466
macro avg	0.78	0.78	0.78	318466
weighted avg	0.78	0.78	0.78	318466

Confusion Matrix of LSTM Model - 2 :



LSTM Model - 1 and LSTM Model - 2

- Decreasing the batch size from 1024 to 512 did not make a significant change in accuracy.



CNN Model - 1

- Batch Size = 1024

```
sequence_input = Input(shape=(30,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = MaxPooling1D(pool_size=2)(x)
x = Flatten()(x)
outputs = Dense(1, activation='sigmoid')(x)
model = tf.keras.Model(sequence_input, outputs)
history = model.fit(X_train, y_train, batch_size=1024,
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 30)]	0
embedding (Embedding)	(None, 30, 300)	87137400
spatial_dropout1d (SpatialDr	(None, 30, 300)	0
conv1d (Conv1D)	(None, 26, 64)	96064
dense (Dense)	(None, 26, 512)	33280
dropout (Dropout)	(None, 26, 512)	0
dense_1 (Dense)	(None, 26, 512)	262656
max_pooling1d (MaxPooling1D)	(None, 13, 512)	0
flatten (Flatten)	(None, 6656)	0
dense_2 (Dense)	(None, 1)	6657

Total params: 87,536,057
Trainable params: 398,657
Non-trainable params: 87,137,400

Evaluation Metrics of CNN Model - 1

CNN Model - 1 :

	precision	recall	f1-score	support
Negative	0.78	0.78	0.78	159493
Positive	0.78	0.78	0.78	158973
accuracy			0.78	318466
macro avg	0.78	0.78	0.78	318466
weighted avg	0.78	0.78	0.78	318466

Confusion Matrix of CNN Model - 1 :

CNN Model - 2

- Batch Size = 512

```
sequence_input = Input(shape=(30,), dtype='int32')
embedding_sequences = embedding_layer(sequence_input)
x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(32, 5, activation='relu')(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.5)(x)
x = Dense(256, activation='relu')(x)
x = MaxPooling1D(pool_size=2)(x)
x = Flatten()(x)
outputs = Dense(1, activation='sigmoid')(x)
model2 = tf.keras.Model(sequence_input, outputs)


model2.compile(optimizer=Adam(learning_rate=1e-3), loss=
ReduceLROnPlateau = ReduceLROnPlateau(factor=0.1, min_lr
history2 = model2.fit(X_train, y_train, batch_size=512,
```

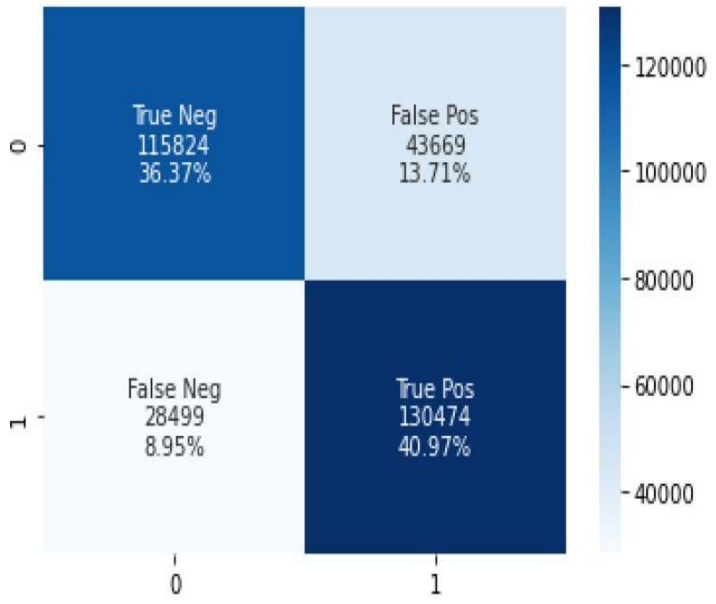
```
Model: "functional_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 30)]              0
_________________________________________________________________
embedding (Embedding)        (None, 30, 300)           87137400
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, 30, 300)           0
_________________________________________________________________
conv1d (Conv1D)              (None, 26, 32)            48032
_________________________________________________________________
dense (Dense)                (None, 26, 256)           8448
_________________________________________________________________
dropout (Dropout)            (None, 26, 256)           0
_________________________________________________________________
dense_1 (Dense)              (None, 26, 256)           65792
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 13, 256)           0
_________________________________________________________________
flatten (Flatten)            (None, 3328)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 3329
=================================================================
Total params: 87,263,001
Trainable params: 125,601
Non-trainable params: 87,137,400
```

Evaluation Metrics of CNN Model - 2

CNN Model - 2 :

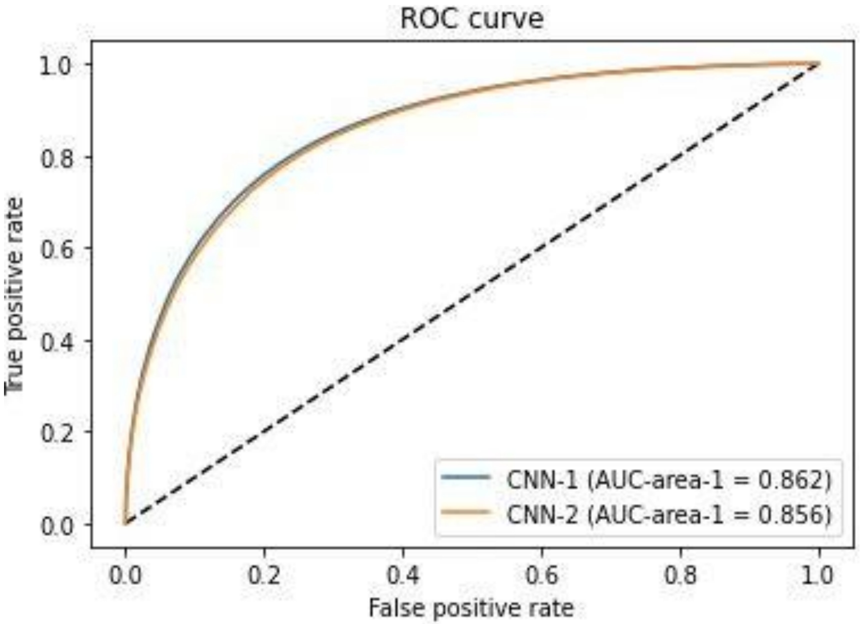
	precision	recall	f1-score	support
Negative	0.80	0.73	0.76	159493
Positive	0.75	0.82	0.78	158973
accuracy			0.77	318466
macro avg	0.78	0.77	0.77	318466
weighted avg	0.78	0.77	0.77	318466

Confusion Matrix of CNN Model - 2 :



CNN Model - 1 and CNN Model - 2

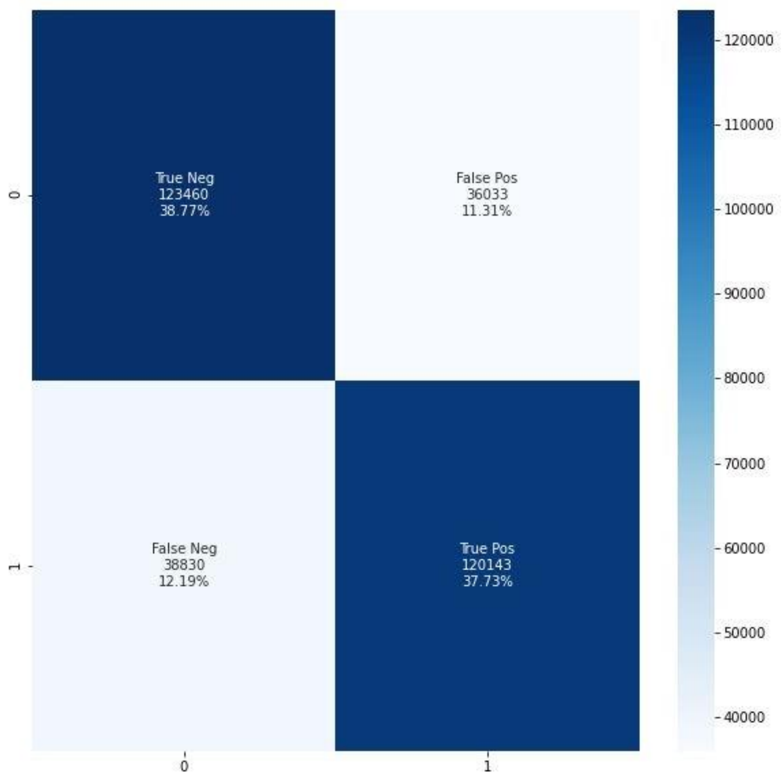
- Decreasing the batch size from 1024 to 512 did not make a significant change in accuracy.



Multinomial Naive Bayes with Count Vectorizer

```
classifier = MultinomialNB()
classifier.fit(vec, train_data['label'])
```

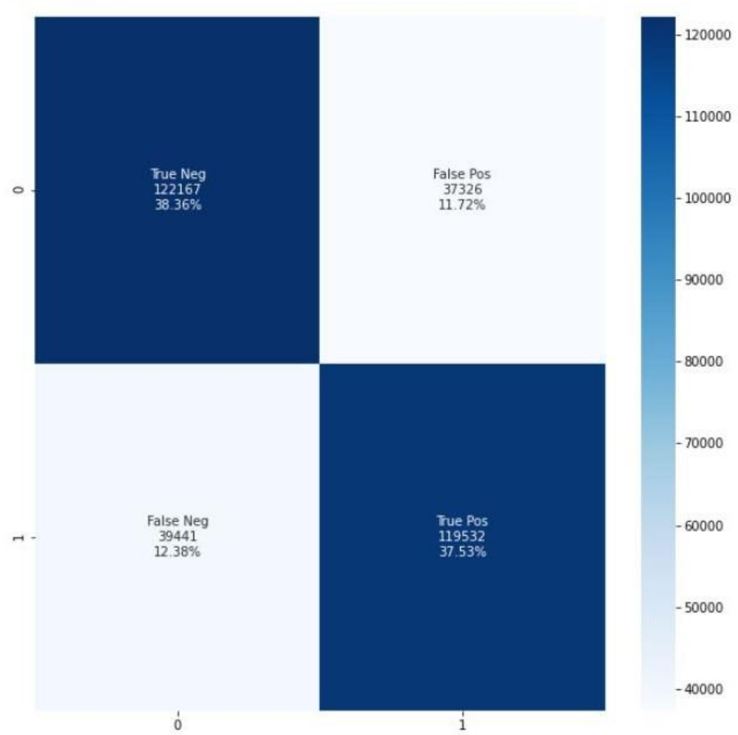
	precision	recall	f1-score	support
Negative	0.76	0.77	0.77	159493
Positive	0.77	0.76	0.76	158973
accuracy			0.76	318466
macro avg	0.77	0.76	0.76	318466
weighted avg	0.77	0.76	0.76	318466



Multinomial Naive Bayes with TF-IDF

```
tfidf_classifier = MultinomialNB()
tfidf_classifier.fit(tfidf_vec, train_data['label'])
```

	precision	recall	f1-score	support
Negative	0.76	0.77	0.76	159493
Positive	0.76	0.75	0.76	158973
accuracy			0.76	318466
macro avg	0.76	0.76	0.76	318466
weighted avg	0.76	0.76	0.76	318466



Statistical Significance Analysis

- Best performing model is LSTM Model - 1 with accuracy 0.789
- Second best performing model is CNN Model - 1 with accuracy 0.781
- Multinomial Naive Bayes with TF-IDF is the worst performing algorithm among them with accuracy 0.758.

LSTM Model 1

	precision	recall	f1-score	support
Negative	0.78	0.79	0.79	159493
Positive	0.79	0.78	0.78	158973
accuracy			0.78	318466
macro avg	0.78	0.78	0.78	318466
weighted avg	0.78	0.78	0.78	318466

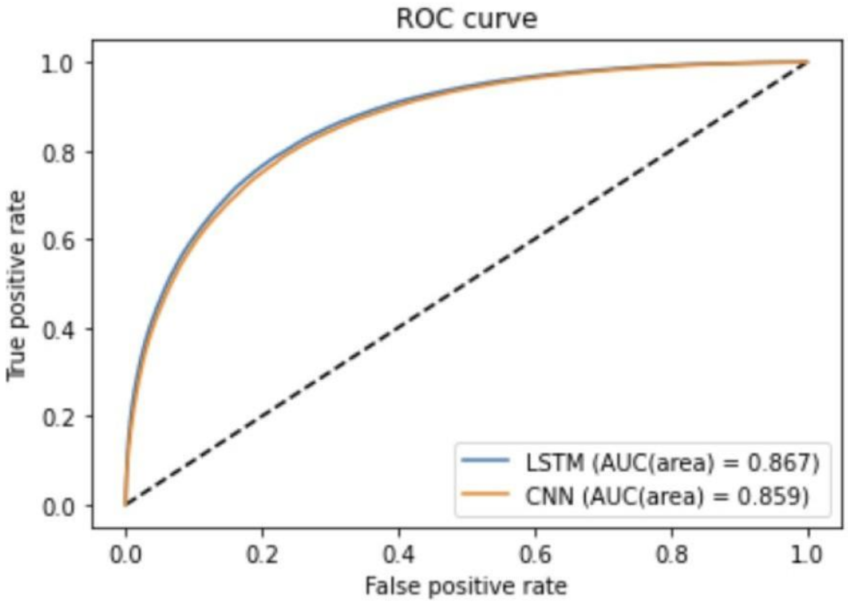
CNN Model 1

	precision	recall	f1-score	support
Negative	0.78	0.78	0.78	159493
Positive	0.78	0.78	0.78	158973
accuracy			0.78	318466
macro avg	0.78	0.78	0.78	318466
weighted avg	0.78	0.78	0.78	318466

Statistical Significance Analysis

- LSTM and CNN results are veryclose to each other.
- Naive Bayes models performed slightly worse.
- Naive Bayes models have thebest training time durations.

ROC Curve of best LSTM model and best CNN model :



Result

- LSTM Model-1 has 78.9% accuracy rate and LSTM model-2 has 78.6% accuracy rate. CNN model-1 has 78.2%accuracy rate and CNN model-2 has 77.2% accuracy rate.

- Both algorithms have better training times with 512 batch size, are better than their 1024 batch sized models and their accuracy rates are really close.

- As a result of these, we can say that LSTM and CNN models with 1024 batch size are better for accuracy rate. But, models with 512 batch size have close accuracy rates within better training times.

- For accuracy rates of Naive Bayes models there is a small difference like 1.5%. As a result of that, we can say that Naive Bayes with the CountVectorizer method gives better results than Naive Bayes with the TF-IDF method.

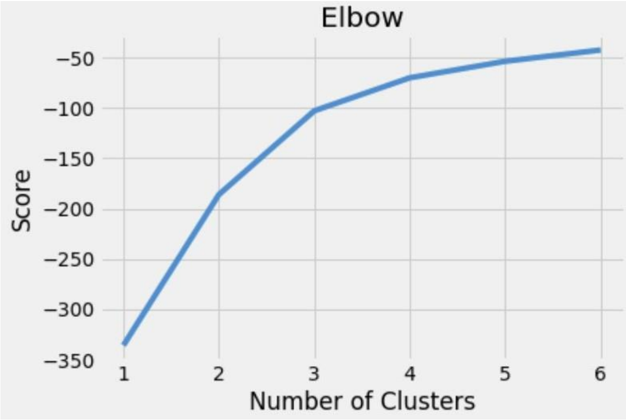
Project Delivery #5
Descriptive Analytics

Project Delivery #5 - Descriptive Analytics

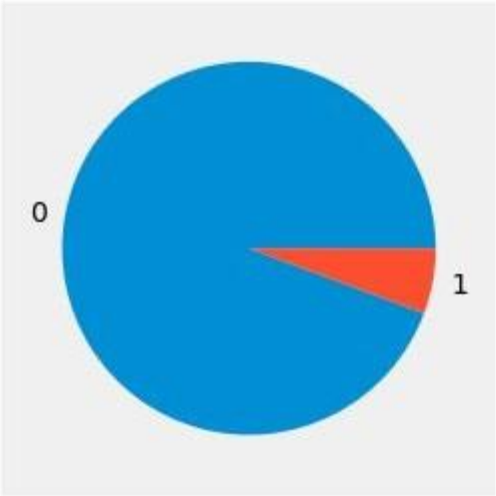
We use Term Frequency-Inverse Document Frequency (TF-IDF) to transform the text data. You can obtain the tf-idf array.

	00	000	0000	002	00am	00pm	01	02	026	02am	...	½sklov	½ssen	½sunday	½t	½tieï	½tobe	½u	½ve	½y	½ï
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Then, we use the Elbow method to make sure we choose the optimal number of clusters. We decided to make experiments 2 and 3 number of clusters.

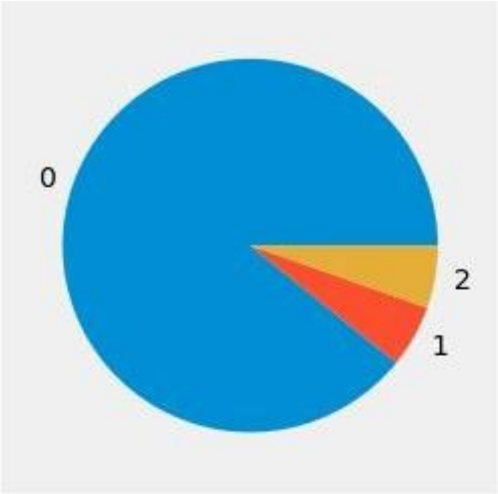


Instance Distribution - Pie Charts



28300 1700

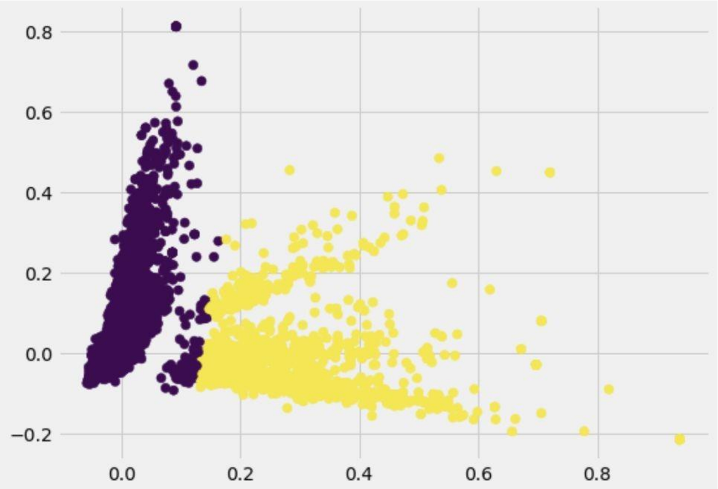
Cluster 0 Percentage = 94.3%
Cluster 1 Percentage = 5.7%
Figure 3. A pie chart showing the instance distributions for 2 clusters.



26584 1609 1609

Cluster 0 Percentage = 88.6%
Cluster 1 Percentage = 5.7%
Cluster 2 Percentage = 5.7%
Figure 4. A pie chart showing the instance distributions for 3 clusters.

Evaluation of Experiments - Experiment 1 Number of Clusters = 2



init	time	inertia	homo	compl	v-meas	ARI	AMI	NMI	silhouette
k-means++	0.093s	38122	0.973	0.970	0.971	0.991	0.971	0.971	0.814
random	0.108s	38122	0.975	0.970	0.972	0.991	0.972	0.972	0.749
PCA-based	0.050s	38985	0.011	0.010	0.010	0.068	0.010	0.010	0.723

Evaluation metrics for 2 clusters.

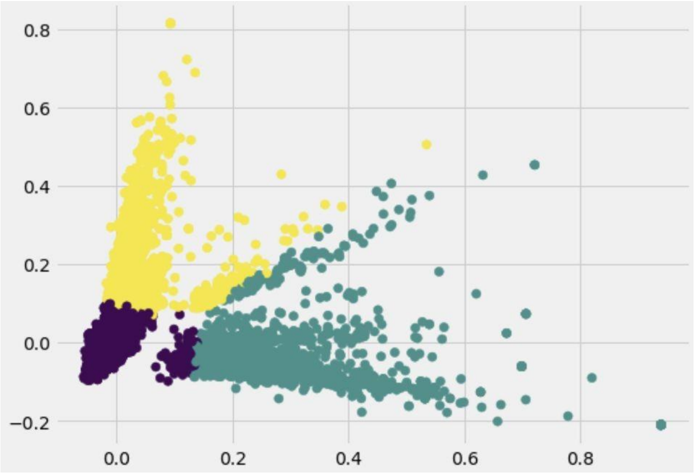
	word	score
0	just	0.015132
1	day	0.012346
2	today	0.011476
3	like	0.010374
4	want	0.010060
5	going	0.010016
6	don	0.009887
7	really	0.009350
8	got	0.009332
9	sad	0.008994
10	good	0.008851
11	miss	0.008415
12	time	0.008402
13	know	0.008327
14	im	0.008257
15	wish	0.008104
16	home	0.008088
17	sorry	0.007745
18	sleep	0.007660
19	night	0.007330

Most important words in Cluster 0

	word	score
0	work	0.302107
1	tomorrow	0.028365
2	day	0.027841
3	today	0.027249
4	going	0.023979
5	ready	0.017308
6	time	0.015752
7	home	0.015085
8	got	0.014318
9	want	0.014249
10	morning	0.014045
11	bed	0.013746
12	getting	0.013460
13	don	0.012802
14	just	0.012368
15	tired	0.011848
16	night	0.011170
17	sleep	0.010848
18	gotta	0.010505
19	hours	0.010329

Most important words in Cluster 1

Evaluation of Experiments - Experiment 2 Number of Clusters = 3



init	time	inertia	homo	compl	v-meas	ARI	AMI	NMI	silhouette
k-means++	0.172s	49102	0.455	0.426	0.440	0.471	0.440	0.440	0.681
random	0.210s	49102	0.454	0.424	0.438	0.470	0.438	0.438	0.717
PCA-based	0.062s	49102	0.455	0.425	0.439	0.471	0.439	0.439	0.662

Evaluation metrics for 2 clusters.

	word	score
0	just	0.015528
1	like	0.010524
2	want	0.009992
3	don	0.009947
4	got	0.009460
5	really	0.009307
6	sad	0.008864
7	going	0.008797
8	miss	0.008738
9	know	0.008549
10	im	0.008349
11	good	0.008327
12	time	0.008284
13	wish	0.008086
14	sorry	0.008076
15	today	0.007920
16	home	0.007797
17	sleep	0.007639
18	need	0.007373
19	night	0.007292

Most important words in Cluster 0

	word	score
0	work	0.308336
1	tomorrow	0.027660
2	today	0.025722
3	going	0.024375
4	day	0.018545
5	ready	0.017846
6	time	0.016309
7	home	0.015314
8	got	0.014329
9	want	0.014295
10	morning	0.014260
11	getting	0.014222
12	bed	0.014176
13	don	0.013260
14	just	0.012587
15	tired	0.012140
16	sleep	0.011213
17	night	0.010812
18	hours	0.010564
19	need	0.010554

Most important words in Cluster 1

	word	score
0	day	0.199634
1	today	0.065946
2	school	0.059420
3	tomorrow	0.057504
4	going	0.028303
5	good	0.017077
6	long	0.015431
7	beautiful	0.013727
8	break	0.013477
9	bad	0.012683
10	home	0.012522
11	bed	0.012221
12	want	0.011225
13	morning	0.010797
14	sad	0.010686
15	feeling	0.010684
16	work	0.010277
17	spring	0.010034
18	time	0.010008
19	really	0.009986

Most important words in Cluster 2

Result

K-means is a very simple and powerful algorithm to cluster a dataset. However, one of the problems is that clusters are spherical. Therefore, it can not be reliable for all situations.

We are using text data for our project. So, we need to represent the data as the model understands. For this reason, firstly, we vectorize our data with tf-idf vectorizer. Then, we use the elbow method to make sure we choose the optimal number of clusters. We decided to make experiments with 2 and 3 numbers of clusters.

The K-means is clustering words according to some semblance of meaning in our experiments, but experiments can be developed with even more accurate parameters.

Thank you for listening