Sentiment Analysis on Airline Tweets Dataset

Steve Song

Aug 12 ,2020

Problem Statement

- Twitter as a part of the marketing strategy for business owners, is a gold mine of customer insights and opportunities to build the brand, drive sales and win fans.
- The dataset we acquired contains tweets on US Airline of February 2015 classified in positive, negative and neutral tweets.
- By applying the sentiment analysis machine learning techniques, we build the predictive models, which enable business owners to predict whether the customers' feedback is positive, negative or neutral.

Data pre-processing

- Tweet preprocessor:

  -- a preprocessing library for tweet data written in Python

  -- supports cleaning, tokenizing and parsing: URLs, Hashtags, Mentions, Reserved words (RT,FAV),Emojis and Smileys.

  -- For example, the original content of the first tweet is **@VirginAmerica What @dhepburn said.**

  -- it was converted into “What said”.

Data pre-processing

- Expanding contractions:

-- Converting each contraction to its expanded, original form often helps with text standardization

-- For example, is "@VirginAmerica I didn't today... Must mean I need to take another trip!. The step converted "didn't " into "did not".

Data pre-processing

- Removing special characters:

  -- Special characters and symbols which are usually non alphanumeric characters often add to the extra noise in unstructured text.

  -- In this project, we use the regular expression '[^a-zA-z0-9\s]'.

Data pre-processing

Stemming and lemmatization

-- Word stems are usually the base form of possible words that can be created by attaching *affixes* like *prefixes* and *suffixes* to the stem to create new words.

-- For example, the words “said” have been converted into “say”.

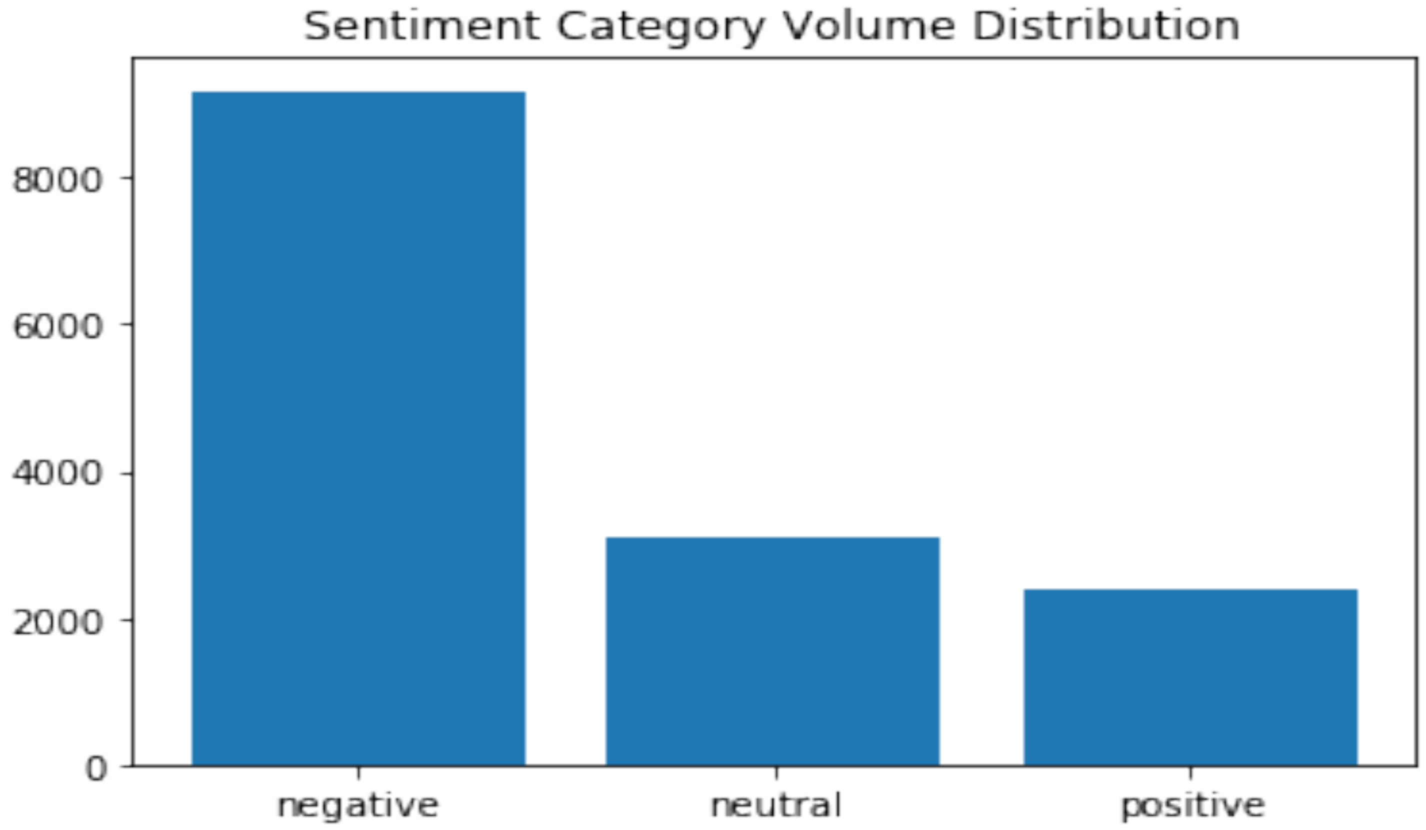
Data pre-processing

Removing stopwords:

-- Words which have little or no significance especially when constructing meaningful features from text are known as stopwords or stop words.

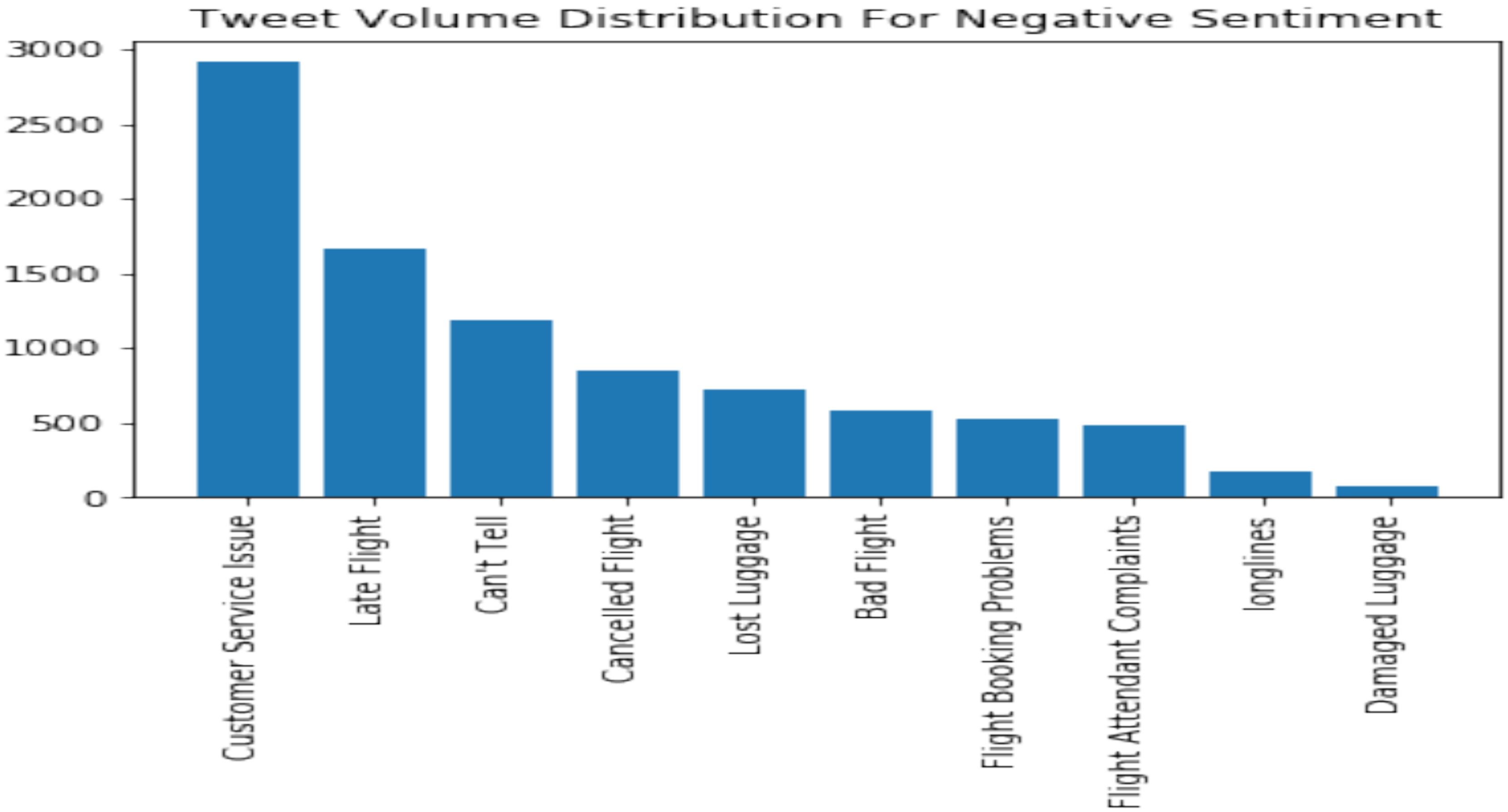
-- We use package NLTK to create a stopwords list, which contains 17 words.

Exploratory Data Analysis (EDA)



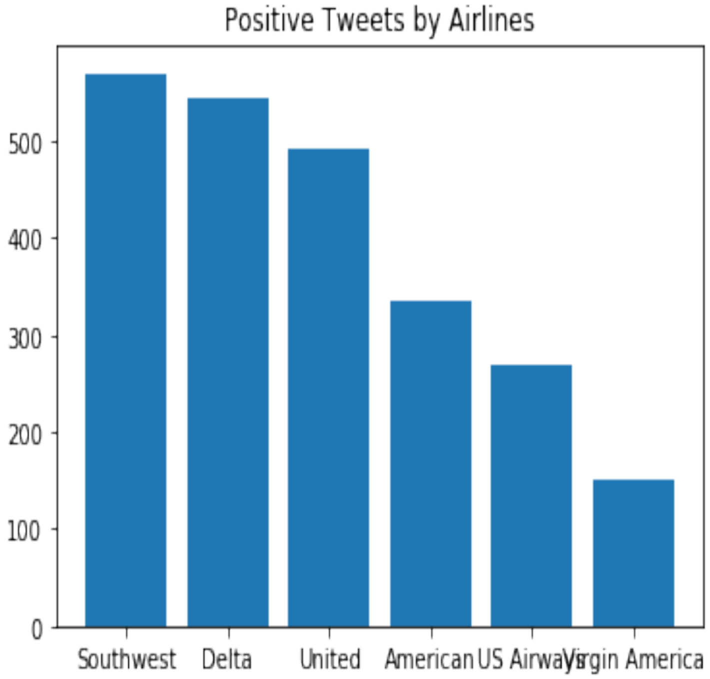
this dataset doesn’t have an issue of ‘imbalance”.

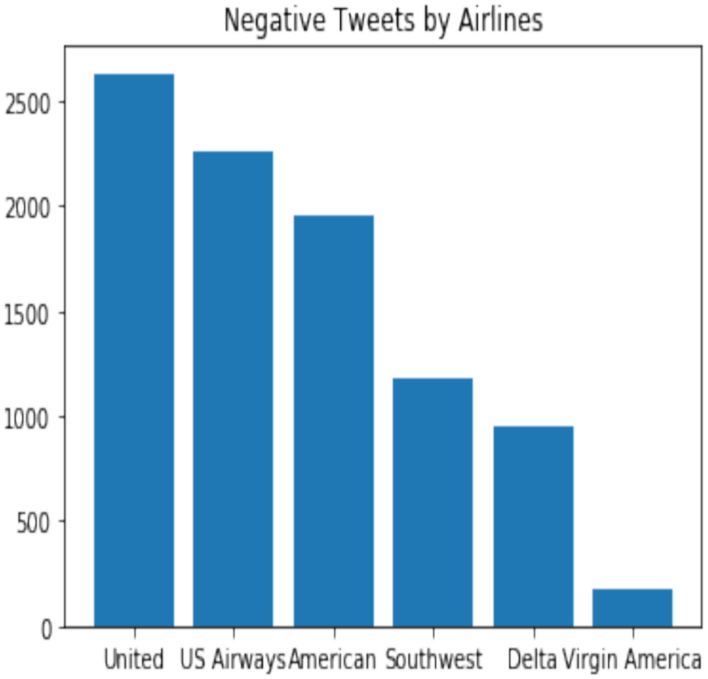
Exploratory Data Analysis (EDA)

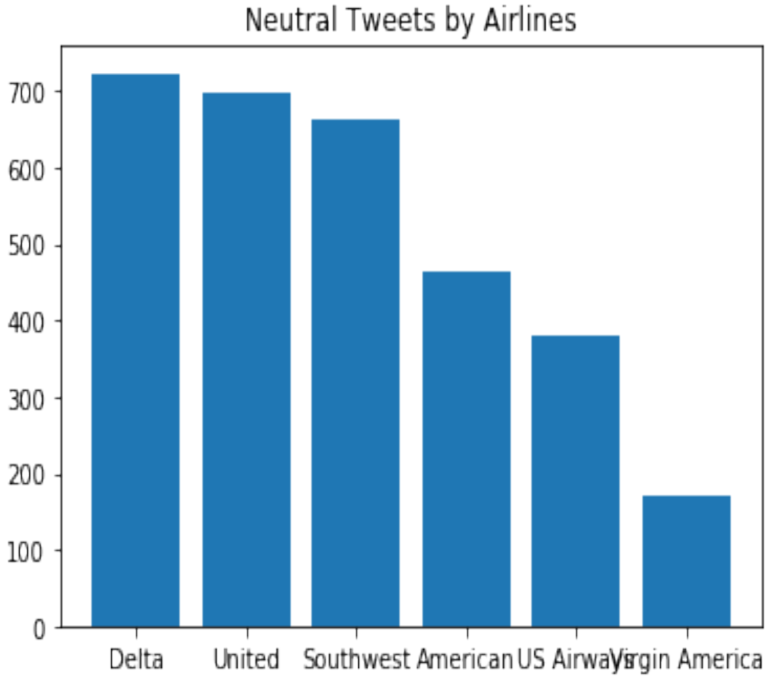


top 3 reasons are Customer Service, Late Flight and Can’t tell.

Exploratory Data Analysis (EDA)







the Southwest had the best performance in terms of positive tweets; United had the worst performance measured by negative tweets.

Exploratory Data Analysis (EDA)

Themost frequently occurred words are '@', '.', 'to', 'I', 'the', '!', '?', 'a', ',', 'for'.

Text data encoding

Word Counts with CountVectorizer

	00	000	000419	00a	00am	00p	00pm	0200	03	04	...	zagg	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom	zurich
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
2	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
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	0	0

5 rows × 6875 columns

we created a data frame with 6875 columns.

Text data encoding

Word Frequencies with TfidfVectorizer

	00	000	000419	00a	00am	00p	00pm	0200	03	04	...	zagg	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom	zurich
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

5 rows × 6875 columns

Traditional Machine Learning Model Building

-- CountVectorization

Naïve Bayes multinomial classification

Accuracy: 0.7491

Support Vector classification

Accuracy: 0.7692

-- TF-IDF

Naïve Bayes multinomial classification

Accuracy: 0.6841

Support Vector classification

Accuracy: 0.7651

Using word embeddings

Word2vec is one of the most popular technique to learn word embeddings using a two-layer neural network.

Gensim library will enable us to develop word embeddings by training our own word2vec models on a custom corpus either with CBOW of skip-grams algorithms.

```
num_features = 100
min_word_count = 5
num_workers = multiprocessing.cpu_count()
context_size = 5
seed = 1
```

Using word embeddings

Logistic Regression

Accuracy: 0.7485

Support Vector classification

Accuracy: 0.7593

-- Gensim also enables us to use pre-trained word2vec model

-- we will import pre-trained Google News model with 300 dimensions for word vector.

SVC	decision tree	logistic regression
0.7823	0.6226	0.7659

Deep learning on Keras

- Learning word embeddings with the embedding layer

-- for example , the sequences of word index for the first five documents in the corpus.

```
[[35],
 [389, 186, 969, 99, 4428],
 [2, 45, 561, 229, 23, 34, 84, 102],
 [73, 2703, 2307, 3267, 753, 1569, 860, 24, 349, 2038],
 [73, 296, 39, 177]]
```

Deep learning on Keras

- the total number of unique tokens is 8,535

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 22, 64)            546304
_________________________________________________________________
flatten_1 (Flatten)          (None, 1408)              0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 4227
=================================================================
Total params: 550,531
Trainable params: 550,531
Non-trainable params: 0
_________________________________________________________________
None
```

Deep learning on Keras

```
Train on 11712 samples, validate on 2928 samples
Epoch 1/10
11712/11712 [==============================] - 7s 632us/step - loss: 0.8143 - acc: 0.6535 - val_loss: 0.5426 - val_ac
c: 0.7971
Epoch 2/10
11712/11712 [==============================] - 4s 378us/step - loss: 0.5394 - acc: 0.7903 - val_loss: 0.4743 - val_ac
c: 0.8224
Epoch 3/10
11712/11712 [==============================] - 5s 427us/step - loss: 0.3999 - acc: 0.8558 - val_loss: 0.4590 - val_ac
c: 0.8221
Epoch 4/10
11712/11712 [==============================] - 5s 417us/step - loss: 0.3059 - acc: 0.8968 - val_loss: 0.4729 - val_ac
c: 0.8159
Epoch 5/10
11712/11712 [==============================] - 5s 407us/step - loss: 0.2355 - acc: 0.9254 - val_loss: 0.4894 - val_ac
c: 0.8101
Epoch 6/10
11712/11712 [==============================] - 5s 407us/step - loss: 0.1817 - acc: 0.9476 - val_loss: 0.5189 - val_ac
c: 0.8105
Epoch 7/10
11712/11712 [==============================] - 5s 465us/step - loss: 0.1441 - acc: 0.9605 - val_loss: 0.5489 - val_ac
c: 0.8070
Epoch 8/10
11712/11712 [==============================] - 7s 568us/step - loss: 0.1174 - acc: 0.9684 - val_loss: 0.5743 - val_ac
c: 0.8060
Epoch 9/10
11712/11712 [==============================] - 6s 540us/step - loss: 0.0973 - acc: 0.9743 - val_loss: 0.6027 - val_ac
c: 0.8012
Epoch 10/10
11712/11712 [==============================] - 6s 506us/step - loss: 0.0828 - acc: 0.9805 - val_loss: 0.6303 - val_ac
c: 0.8005
```

Deep learning on Keras

- Using pre-trained word embedding, importing 100-dimensional "Glove" database, we created the dictionary of embedding index.
-

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 22, 100)           865200
_________________________________________________________________
flatten_3 (Flatten)          (None, 2200)              0
_________________________________________________________________
dense_4 (Dense)              (None, 32)                70432
_________________________________________________________________
dense_5 (Dense)              (None, 3)                 99
=================================================================
Total params: 935,731
Trainable params: 935,731
Non-trainable params: 0
_________________________________________________________________
```

Deep learning on Keras

```
Train on 11712 samples, validate on 2928 samples
Epoch 1/10
11712/11712 [==============================] - 6s 495us/step - loss: 0.7063 - acc: 0.7130 - val_loss: 0.5474 - val_ac
c: 0.7889
Epoch 2/10
11712/11712 [==============================] - 5s 423us/step - loss: 0.5741 - acc: 0.7693 - val_loss: 0.5228 - val_ac
c: 0.8057
Epoch 3/10
11712/11712 [==============================] - 5s 426us/step - loss: 0.5090 - acc: 0.7989 - val_loss: 0.5450 - val_ac
c: 0.7951
Epoch 4/10
11712/11712 [==============================] - 5s 424us/step - loss: 0.4494 - acc: 0.8250 - val_loss: 0.5787 - val_ac
c: 0.7862
Epoch 5/10
11712/11712 [==============================] - 5s 433us/step - loss: 0.3941 - acc: 0.8502 - val_loss: 0.5863 - val_ac
c: 0.7913
Epoch 6/10
11712/11712 [==============================] - 5s 423us/step - loss: 0.3422 - acc: 0.8727 - val_loss: 0.6409 - val_ac
c: 0.7746
Epoch 7/10
11712/11712 [==============================] - 5s 431us/step - loss: 0.2927 - acc: 0.8965 - val_loss: 0.6780 - val_ac
c: 0.7671
Epoch 8/10
11712/11712 [==============================] - 5s 429us/step - loss: 0.2515 - acc: 0.9107 - val_loss: 0.7410 - val_ac
c: 0.7408
Epoch 9/10
11712/11712 [==============================] - 5s 424us/step - loss: 0.2127 - acc: 0.9296 - val_loss: 0.7528 - val_ac
c: 0.7835
Epoch 10/10
11712/11712 [==============================] - 5s 422us/step - loss: 0.1820 - acc: 0.9409 - val_loss: 0.8168 - val_ac
c: 0.7698
```

Deep learning on Keras

- Keras also provides two different layers, SimpleRNN & LSTM. We can use them together with embedding layer to construct neural network model.

```
Train on 11712 samples, validate on 2928 samples
Epoch 1/10
11712/11712 [==============================] - 6s 518us/step - loss: 0.7643 - acc: 0.6867 - val_loss: 0.5623 - val_ac
c: 0.7964
Epoch 2/10
11712/11712 [==============================] - 5s 394us/step - loss: 0.5662 - acc: 0.7836 - val_loss: 0.5132 - val_ac
c: 0.8012
Epoch 3/10
11712/11712 [==============================] - 5s 389us/step - loss: 0.4733 - acc: 0.8227 - val_loss: 0.5505 - val_ac
c: 0.7862
Epoch 4/10
11712/11712 [==============================] - 5s 400us/step - loss: 0.4126 - acc: 0.8487 - val_loss: 0.6630 - val_ac
c: 0.7377
Epoch 5/10
11712/11712 [==============================] - 5s 390us/step - loss: 0.3578 - acc: 0.8725 - val_loss: 0.5906 - val_ac
c: 0.7848
Epoch 6/10
11712/11712 [==============================] - 5s 393us/step - loss: 0.3133 - acc: 0.8881 - val_loss: 0.5705 - val_ac
c: 0.7900
Epoch 7/10
11712/11712 [==============================] - 5s 399us/step - loss: 0.2772 - acc: 0.9016 - val_loss: 0.5892 - val_ac
c: 0.7937
Epoch 8/10
11712/11712 [==============================] - 5s 407us/step - loss: 0.2402 - acc: 0.9152 - val_loss: 0.7106 - val_ac
c: 0.7490
Epoch 9/10
11712/11712 [==============================] - 5s 422us/step - loss: 0.2084 - acc: 0.9263 - val_loss: 0.6624 - val_ac
c: 0.7702
Epoch 10/10
11712/11712 [==============================] - 5s 424us/step - loss: 0.1825 - acc: 0.9365 - val_loss: 0.6946 - val_ac
c: 0.7811
```

Deep learning on Keras

```
Train on 11712 samples, validate on 2928 samples
Epoch 1/10
11712/11712 [==============================] - 15s 1ms/step - loss: 0.8327 - acc: 0.6361 - val_loss: 0.5610 - val_ac
c: 0.7688
Epoch 2/10
11712/11712 [==============================] - 18s 2ms/step - loss: 0.6253 - acc: 0.7229 - val_loss: 0.5160 - val_ac
c: 0.7971
Epoch 3/10
11712/11712 [==============================] - 20s 2ms/step - loss: 0.5508 - acc: 0.7717 - val_loss: 0.5224 - val_ac
c: 0.7900
Epoch 4/10
11712/11712 [==============================] - 31s 3ms/step - loss: 0.5086 - acc: 0.7978 - val_loss: 0.5100 - val_ac
c: 0.7917
Epoch 5/10
11712/11712 [==============================] - 26s 2ms/step - loss: 0.4782 - acc: 0.8147 - val_loss: 0.5026 - val_ac
c: 0.8033
Epoch 6/10
11712/11712 [==============================] - 23s 2ms/step - loss: 0.4554 - acc: 0.8262 - val_loss: 0.5226 - val_ac
c: 0.8053
Epoch 7/10
11712/11712 [==============================] - 24s 2ms/step - loss: 0.4338 - acc: 0.8336 - val_loss: 0.5225 - val_ac
c: 0.7848
Epoch 8/10
11712/11712 [==============================] - 24s 2ms/step - loss: 0.4145 - acc: 0.8459 - val_loss: 0.5235 - val_ac
c: 0.8023
Epoch 9/10
11712/11712 [==============================] - 28s 2ms/step - loss: 0.3908 - acc: 0.8595 - val_loss: 0.5188 - val_ac
c: 0.8016
Epoch 10/10
11712/11712 [==============================] - 27s 2ms/step - loss: 0.3643 - acc: 0.8711 - val_loss: 0.5492 - val_ac
c: 0.7934
```

Business Implication

- Out of all these models, we found the Keras-based model using Embedding layer outperforms other counterparts by the accuracy score of 0.8224.
- If this model is put into practice, the business owner will have the much high ability of predicting the sentiment for incoming messages. This can definitely help enhance the operational efficiency and improve the customers’ satisfaction.