DEEP LEARNING ASSIGNMENT

CLASSIFICATION OF TWEETS TO AIRLINES INTO WHETHER THEY BEAR POSITIVE, NEGATIVE OR NEUTRAL SENTIMENTS USING CONVOLUTIONAL NEURAL NETWORK

DEBADRITA ROY
BCSE-IV (A1)
001910501025
JADAVPUR UNIVERSITY

PROBLEM STATEMENT

Implement a convolutional neural network model for text classification where the input is tokenized as a sequence of characters and a character is represented using one-hot encoding. Use Keras platform for implementation.
Follow the following guidelines for implementing the model.
(1) Split the data into 80% training and 20% testing sets.
(2) Experimentally decide

- a) number of CNN layers required
- b) Which optimizer is the best for this dataset? Adam or RMSprop
- c) Number of filters (kernels) at each layer and filter size
- d) pooling size of the pooling layer added after each CNN layer
- e) Number of epochs
- f) dropout
- g) batch size

Signed

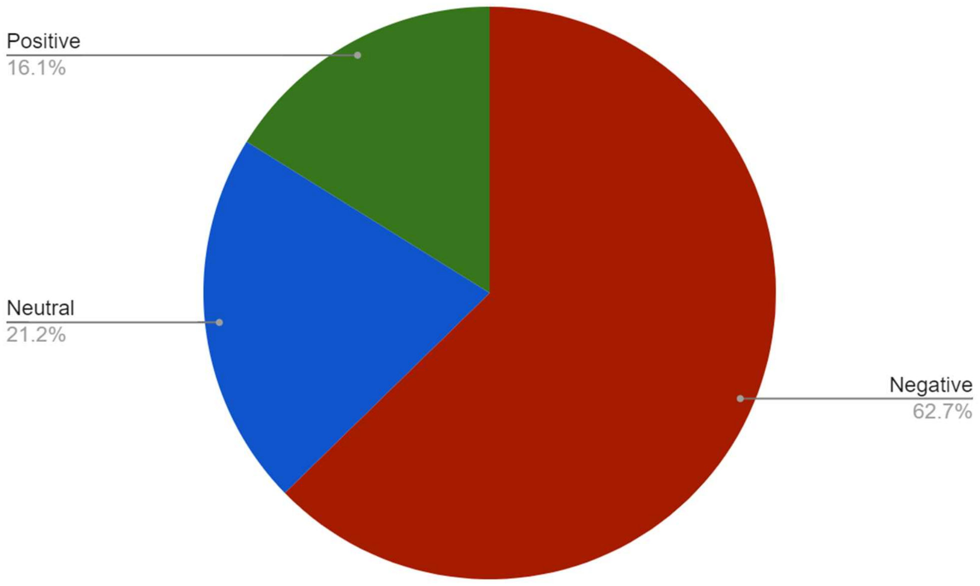
Debadrita Roy

DATASET DESCRIPTION

The given dataset contains tweets posted by customers to several major US airlines in February 2015. Each tweet is classified as having positive, negative or neutral sentiment.

CLASS WISE DISTRIBUTION

The dataset contains 14640 tweets, out of which 9178 are negative, 3099 are neutral and 2363 are positive. So, the dataset is highly skewed towards negative tweets (which makes sense, after all, one usually tends to tweet to express one's displeasure, not to commend the airline for a good flight).



IMBALANCE RATIO

The imbalance ratio (IR) is the most commonly used measure to describe the imbalance extent of a dataset. It is defined as

$$IR = \frac{N_{maj}}{N_{min}}$$

where, N_{maj} is the sample size of the majority class and N_{min} is the sample size of the minority class.

In the given dataset, imbalance ratio = 3.8840 : 1.3114 : 1
(negative: neutral: positive)

DATA PREPROCESSING

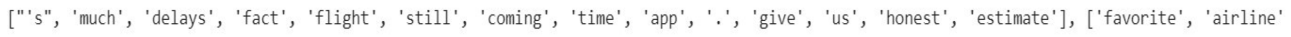
Only the text column and the airline_sentiment column were used to train the model. This was done so that the model can classify future tweets as positive, neutral or negative only on the basis of the text in the tweet.

The text column contains string values (tweet message) while the airline_sentiment column can take on three values: negative, neutral or positive.

Steps involved in cleaning / pre-processing the tweet:

- Removing the usernames (denoted by @<username> in a tweet)
- Changing all letters to lowercase
- Removing stop words (i.e., words which occur frequently in English language, like “a”, “the”, “i”,etc.)

Sample tweet (s) after pre-processing (Each tweet is a list of tokens):



```
["'s", 'much', 'delays', 'fact', 'flight', 'still', 'coming', 'time', 'app', '.', 'give', 'us', 'honest', 'estimate'], ['favorite', 'airline'
```

IMPLEMENTATION

```
# removing usernames
def clean(st):
  ans=''
  ans=re.sub('@\w*','',st)
  return ans

# changing to lowercase
def lower(toks):
  return [tok.lower() for tok in toks]

# removing stop words
def rem_stop_wrds(toks):
  list_stop=stopwords.words('english')
  return [wrd for wrd in toks if wrd not in list_stop]

username_remd_tweets=training.apply(lambda x: clean(x))
tokens_tweets=[word_tokenize(txt) for txt in username_remd_tweets]
tokens_tweets=[lower(tkn) for tkn in tokens_tweets]
tokens_tweets=[rem_stop_wrds(tkn) for tkn in tokens_tweets]
```

REPRESENTATION OF INPUT

Each cleaned tweet is now tokenized into a sequence of characters and each character is represented using one-hot encoding.

The vocabulary (of characters) is made by considering the set of characters in the training examples, plus one OOV character (Out-Of-Vocabulary) to represent those characters in the test examples which were unseen in the training set. Each character is represented using one-hot encoding on this vocabulary, i.e., an array where the value at the position of this character in the vocabulary is 1, and the other values are 0.

Each tweet is padded with arrays of 0’s (for each padded character) so that the length of each tweet is the same (150 characters).

IMPLEMENTATION

```
vocab=set()
for tw in tokens_tweets:
  for w in tw:
    for ch in w:
      # print(ch)
      vocab.add(ch)
#print(len(vocab))

vocab_list=list(vocab)
max_length=150
onehot_encoded_inputs=[]
for tw in tokens_tweets:
  tw_code=[]
  for w in tw:
    for ch in w:
      p=vocab_list.index(ch)
      ll=(len(vocab_list)+1)*[0]
      ll[p]=1
      tw_code.append(ll)
  onehot_encoded_inputs.append(tw_code)
# padding the tweets
for i in range(len(onehot_encoded_inputs)):
  if len(onehot_encoded_inputs[i])<max_length:
    for _ in range(max_length-len(onehot_encoded_inputs[i])):
      onehot_encoded_inputs[i].append((len(vocab_list)+1)*[0])
```

REPRESENTATION OF OUTPUT

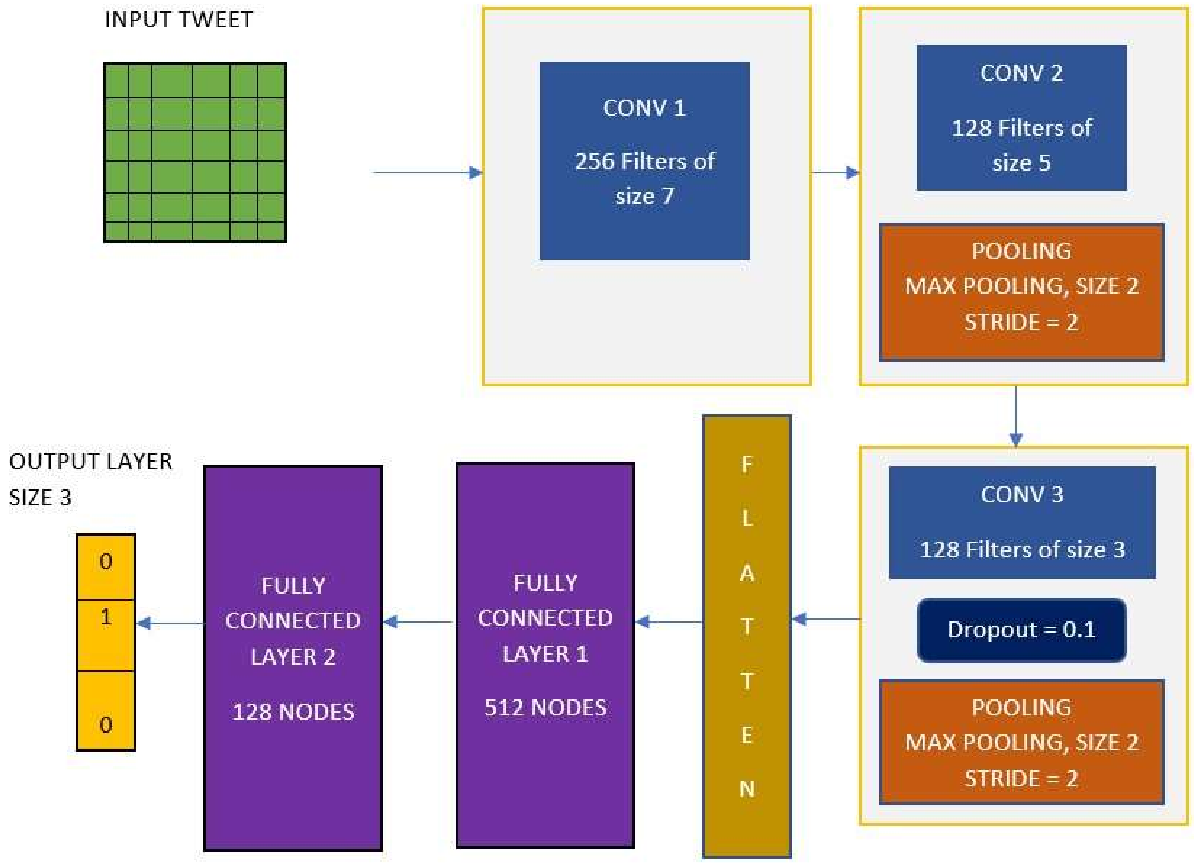
The outputs (airline_sentiment) are given to the model (for training) in the form of one-hot encoded vectors of length 3: [1 0 0] represents negative sentiment, [0 1 0] represents neutral sentiment and [0 0 1] represents positive sentiment.

The model gives as output a vector of length 3, the value of each index gives the likelihood of each sentiment for the tweet. The index with the maximum likelihood is chosen to be the sentiment of the tweet as predicted by the model.

MODEL ARCHITECTURE

A Convolutional Neural Network (CNN) is used to model the problem. A 2-D array consisting of each tweet (150 x (|V|+1)) is given as input, where |V| is the vocabulary size. The output is a vector of size 3, where each value represents the likelihood of a tweet being negative (for index 0), neutral (for index 1) or positive (for index 2).

BLOCK DIAGRAM



** All convolution functions and pooling functions are 1D*

MODEL DESCRIPTION

NUMBER OF CNN LAYERS REQUIRED = 3

CONVOLUTION LAYER 1

Number of filters = 256
Filter size = 7
Activation Function = ReLU
NO Dropout

CONVOLUTION LAYER 2

Number of filters = 128
Filter size = 5
Activation Function = ReLU
Pooling Size = 2
Pooling Stride = 2
Type of Pooling = Max Pooling
NO Dropout

CONVOLUTION LAYER 3

Number of filters = 128
Filter size = 3
Activation Function = ReLU
Dropout = 0.1
Pooling Size = 2
Pooling Stride = 2
Type of Pooling = Max Pooling

FLATTEN LAYER

It flattens the 2D vector into a 1D vector. (here, we get a 1 x 4352 vector)

FULLY CONNECTED LAYER 1

Number of Nodes = 512
NO Dropout
Activation Function = ReLU

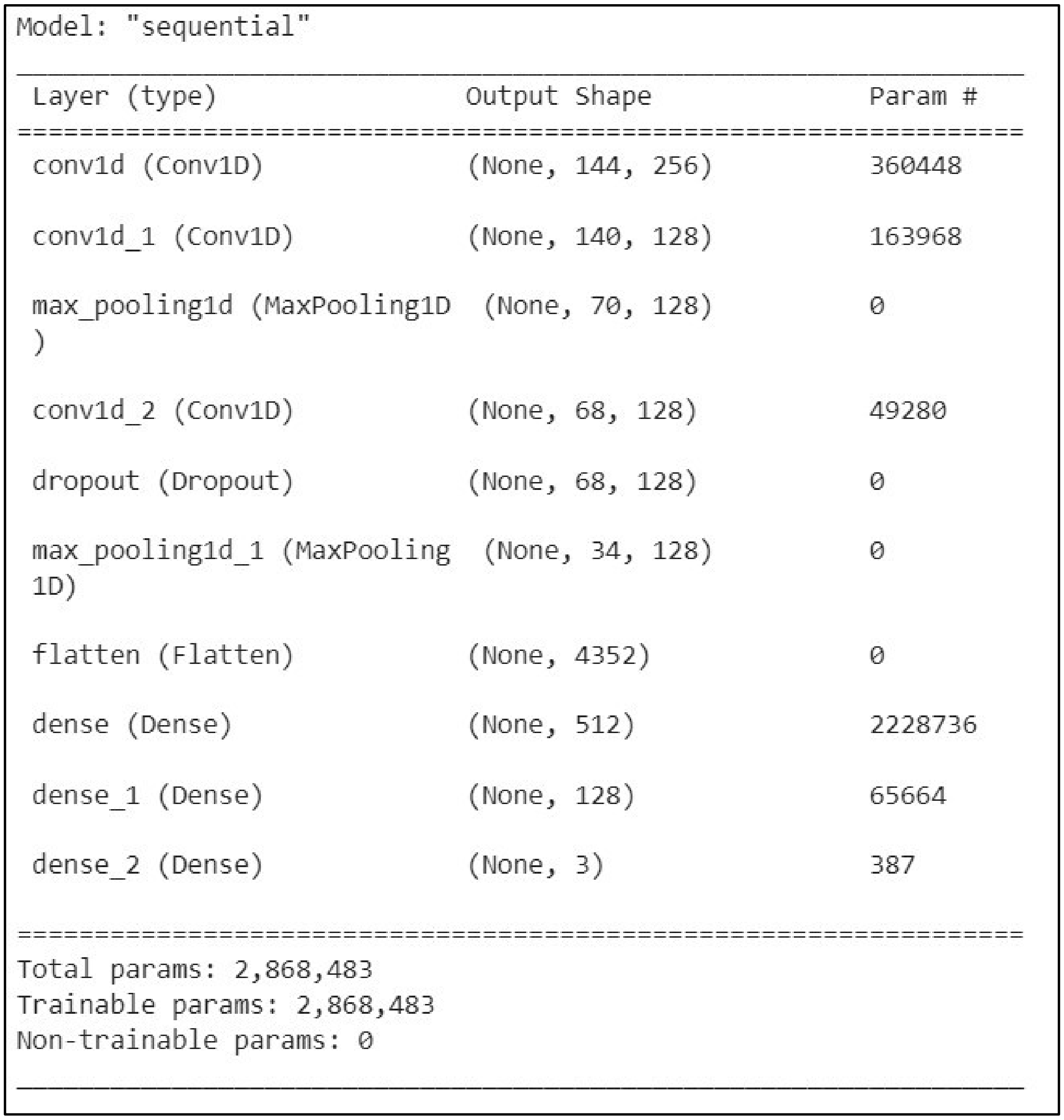
FULLY CONNECTED LAYER 2

Number of Nodes = 128
NO Dropout
Activation Function = ReLU

OUTPUT LAYER

Number of Nodes = 3
Activation Function = SoftMax

MODEL SUMMARY



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d (Conv1D)             (None, 144, 256)          360448

 conv1d_1 (Conv1D)           (None, 140, 128)          163968

 max_pooling1d (MaxPooling1D  (None, 70, 128)          0
 )

 conv1d_2 (Conv1D)           (None, 68, 128)           49280

 dropout (Dropout)           (None, 68, 128)           0

 max_pooling1d_1 (MaxPooling  (None, 34, 128)          0
 1D)

 flatten (Flatten)           (None, 4352)              0

 dense (Dense)               (None, 512)               2228736

 dense_1 (Dense)             (None, 128)               65664

 dense_2 (Dense)             (None, 3)                 387

=================================================================
Total params: 2,868,483
Trainable params: 2,868,483
Non-trainable params: 0
_________________________________________________________________
```

CODE FOR BUILDING AND TRAINING MODEL

```
def CNN(train_x,train_y,len_out):
  model=Sequential()
  model.add(Conv1D(filters=256,kernel_size=7,activation='relu'))
  model.add(Conv1D(filters=128,kernel_size=5,activation='relu'))
  model.add(MaxPooling1D(pool_size=2,strides=2,padding='valid'))
  model.add(Conv1D(filters=128,kernel_size=3,activation='relu'))
  model.add(Dropout(0.1))
  model.add(MaxPooling1D(pool_size=2,strides=2,padding='valid'))
  model.add(Flatten())
  model.add(Dense(512,activation='relu'))
  model.add(Dense(128,activation='relu'))
  model.add(Dense(len_out,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=[metrics.Categoric
alAccuracy()])
  model.fit(train_x,train_y,epochs=10,batch_size=128,verbose=2)
  model.save('/content/drive/MyDrive/modelseq58.h5')
  model.summary()
  return model
```

MODEL TRAINING

TRAINING DATA

80% of the examples in the dataset were randomly selected to be the training examples.
Number of training examples = 11712

Following training parameters were experimentally found to be best for the problem.

NUMBER OF EPOCHS

The model is trained over 10 epochs, i.e., all the training examples are presented 10 times to the model to learn from.

BATCH SIZE

Batch size of 128 is used, i.e., weights are updated after 128 training examples have been presented to the model.

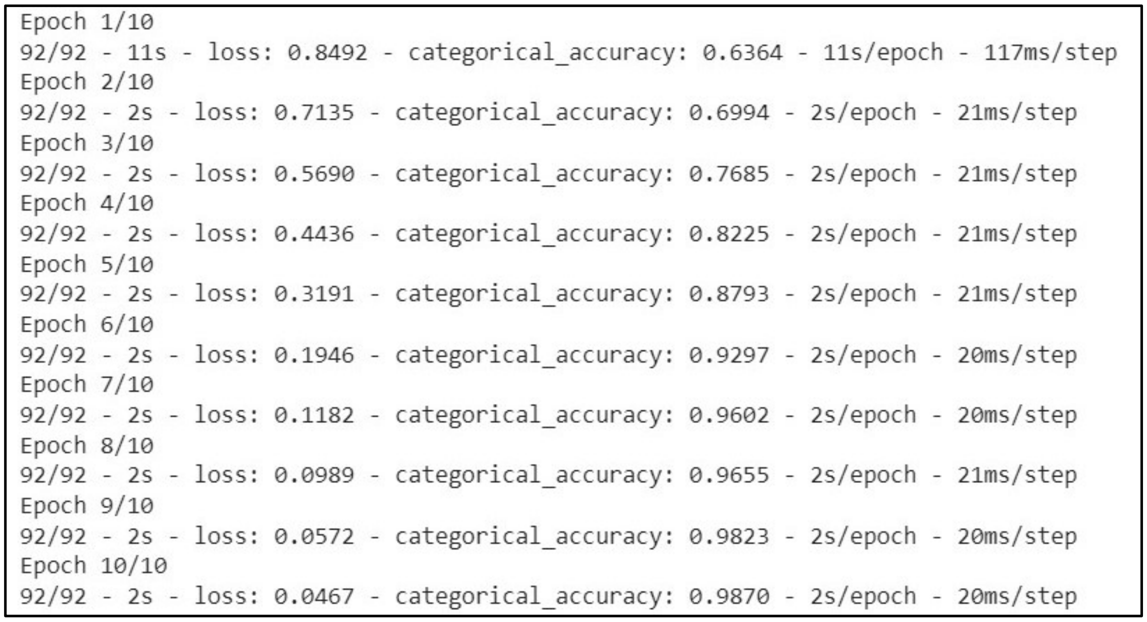
OPTIMIZER

The Adam optimizer function was used. It is an optimization technique for gradient descent which is really efficient when working with a large problem involving a lot of data or parameters. It requires less memory. As the given dataset has a lot of data, Adam optimizer gives better results than RMSProp.

LOSS FUNCTION

Categorical Cross Entropy is used as the loss function. (as it is a multi-class classification problem)

TRAINING THE MODEL



```
Epoch 1/10
92/92 - 11s - loss: 0.8492 - categorical_accuracy: 0.6364 - 11s/epoch - 117ms/step
Epoch 2/10
92/92 - 2s - loss: 0.7135 - categorical_accuracy: 0.6994 - 2s/epoch - 21ms/step
Epoch 3/10
92/92 - 2s - loss: 0.5690 - categorical_accuracy: 0.7685 - 2s/epoch - 21ms/step
Epoch 4/10
92/92 - 2s - loss: 0.4436 - categorical_accuracy: 0.8225 - 2s/epoch - 21ms/step
Epoch 5/10
92/92 - 2s - loss: 0.3191 - categorical_accuracy: 0.8793 - 2s/epoch - 21ms/step
Epoch 6/10
92/92 - 2s - loss: 0.1946 - categorical_accuracy: 0.9297 - 2s/epoch - 20ms/step
Epoch 7/10
92/92 - 2s - loss: 0.1182 - categorical_accuracy: 0.9602 - 2s/epoch - 20ms/step
Epoch 8/10
92/92 - 2s - loss: 0.0989 - categorical_accuracy: 0.9655 - 2s/epoch - 21ms/step
Epoch 9/10
92/92 - 2s - loss: 0.0572 - categorical_accuracy: 0.9823 - 2s/epoch - 20ms/step
Epoch 10/10
92/92 - 2s - loss: 0.0467 - categorical_accuracy: 0.9870 - 2s/epoch - 20ms/step
```

MODEL EVALUATION

TEST DATA

20% of the examples in the dataset were randomly selected to be the test examples (remaining examples after training examples are removed).
Number of test examples = 2928

EVALUATION METRICS

The model was evaluated using the following metrics.

Accuracy: It is the ratio of the number of correct predictions to the total number of predictions made for a dataset.

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}$$

Precision: It is the ratio of True Positives to all the positives predicted by the model. It is useful for skewed and unbalanced datasets. The more False positives the model predicts, the lower the precision.

$$Precision = \frac{TP}{TP + FP}$$

Recall: It is the ratio of true positives to all the positives in the dataset. It measures the model's ability to detect positive samples. The more false negatives the model predicts, the lower the recall.

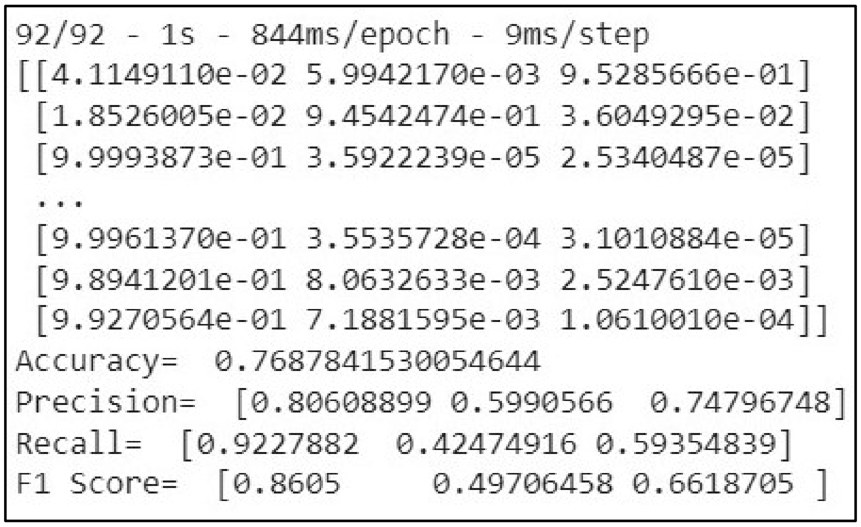
$$Recall = \frac{TP}{TP + FN}$$

F1 Measure: It is a single metric that combines both Precision and Recall. The higher the F1 score, the better is the performance of the model. The range for F1-score is [0,1].
F1 score is the weighted average of precision and recall. The classifier will only get a high F1-score if both precision and recall are high.

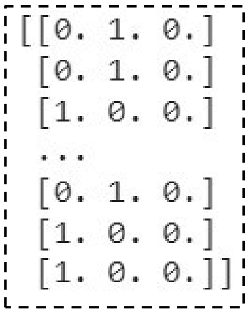
$$F1\ Measure = \frac{2 \times precision \times recall}{precision + recall}$$

RESULTS

The results of the model on the test set are as follows.



```
92/92 - 1s - 844ms/epoch - 9ms/step
[[4.1149110e-02 5.9942170e-03 9.5285666e-01]
 [1.8526005e-02 9.4542474e-01 3.6049295e-02]
 [9.9993873e-01 3.5922239e-05 2.5340487e-05]
 ...
 [9.9961370e-01 3.5535728e-04 3.1010884e-05]
 [9.8941201e-01 8.0632633e-03 2.5247610e-03]
 [9.9270564e-01 7.1881595e-03 1.0610010e-04]]
Accuracy=  0.7687841530054644
Precision=  [0.80608899 0.5990566  0.74796748]
Recall=  [0.9227882  0.42474916 0.59354839]
F1 Score=  [0.8605     0.49706458 0.6618705 ]
```



```
[[0. 1. 0.]
 [0. 1. 0.]
 [1. 0. 0.]
 ...
 [0. 1. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

The first few lines show the output vectors for the test set as given by the model. It can be contrasted with the actual sentiments for the test set. As we can see, the model correctly predicts the second, third, second-last and last tweets out of the six tweets that can be seen.

The accuracy is 76.88 % and precision values for the negative, neutral and positive classes are 80.61%, 59.91% and 74.80% respectively. Recall values for the negative, neutral and positive classes are 92.28%, 42.47% and 59.35% respectively while F1 Scores for the negative, neutral and positive classes are 0.8605, 0.4971 and 0.6619 respectively.

ANALYSIS

The model has a high precision and very high recall for the negative class (as the dataset is highly imbalanced with almost two-thirds of all the tweets being negative, the model was able to learn the features for a negative tweet very well). Although accuracy is not a good indicator for the goodness of the model in case of an imbalanced dataset, the model has a good accuracy at around 77%. Precision for the positive class and the neutral class is high as well. F1 scores for the negative and the positive classes are good as well.

PROS OF THE MODEL

- Airlines are usually concerned with the negative tweets, either for solving the issue faced by the passenger or for doing damage control. As our model gives a very high recall for the negative class, it can be used to effectively filter out the negative tweets out of a large number of tweets. Over 92% of the total negative tweets will be correctly identified as negative by our model. (less “false negatives” for the negative class, i.e. frequency with which negative tweets are classified as positive or neutral)
- Precision scores are high. So, if the model classifies a tweet as negative or neutral or positive, there is a high likelihood that the tweet is actually negative or neutral or positive as the case may be. (less “false positives”)
- The model incorporates emojis, which are very frequently used in tweets. So, a tweet having only emojis or having a major part as emojis can also be classified by the model (as opposed to models which only consider the text in the tweet for classification). Each emoji is treated as an unique character and while training, the model learns which emojis are more likely to be used in negative or positive or neutral tweets.
- The model has less classification time, 2928 test examples were classified in 1 second using GPU.
- The model is relatively less complex (having only three convolution layers).

CONS OF THE MODEL

- Recall for neutral tweets is low. So, the model cannot be used to effectively filter out neutral tweets (unlike negative ones).
- The model is not very effective at learning features for neutral tweets, mostly due to less number of neutral tweets in the dataset and also because of the inherent ambiguity in determining whether a tweet is neutral as opposed to

positive / negative. (has low recall and f1 score, but a reasonably good — around 60% — precision).

CONCLUSION

We designed a convolutional neural network to learn features from tweets to several US airlines and classify a given tweet as negative, neutral or positive. The model is very good at identifying negative tweets, and good at identifying others. It can be used to effectively filter out negative tweets from a large number of tweets, which is of interest to the airlines (High recall for negative tweets is the most important criterion for the airline to choose which model to use).

REFERENCES

- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.
- https://medium.com/analytics-vidhya/evaluation-metrics-for-classification-models-e2f0d8009d69
- https://keras.io