btctweets_sentiment_prediction-cnn-lstm-05312020

June 14, 2020

1 Bitcoin Tweets Sentiment Analysis Prediction – CNN-LSTM

[32]:
```
# Data analysis and wrangling
import pandas as pd
import numpy as np
import os
import string
import csv

# Visualization
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn
from wordcloud import WordCloud

# Sentiment prediction
import re
import nltk
from sklearn.model_selection import train_test_split
from keras.preprocessing.text import Tokenizer
from keras.preprocessing import sequence

from tensorflow.keras import backend, models, layers, regularizers
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Dense, Embedding, Conv1D, MaxPooling1D, LSTM
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

[2]:
```
# Upload the processed clean tweets

df = pd.read_csv('cleaned_tweet_data_05312020.csv',
                    header = 0,
                    error_bad_lines=False,
                    engine='python',
                    usecols=[2,5])

df.head()
```

[2]:

```
                                         clean_text sentiment
0  goldman sachs hosting client call bitcoin gold…   neutral
1                ok president trump endorsing tommy  positive
2  bitcoin day nwhen btc first used commercial tr…  positive
3  great icle nif way could avoided know like min…  positive
4                     bitcoin btc current price gbp   neutral
```

[3]:

```
df.tail()
```

[3]:

```
                                            clean_text sentiment
2015  xrp xrpcommunity btc bitcoin eth ltc vet oh ye…  positive
2016  yes world global pandemic world going recessio…   neutral
2017  last days grayscale bitcoin trust bought bitco…   neutral
2018  pretty incredible imho nwe need creativity lik…  positive
2019                             drop sachs nget sats   neutral
```

1.1 Exploratory Data Analysis

[4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2020 entries, 0 to 2019
Data columns (total 2 columns):
clean_text    2020 non-null object
sentiment     2020 non-null object
dtypes: object(2)
memory usage: 31.7+ KB
```

[5]:

```
df.describe()
```

[5]:

```
                   clean_text sentiment
count                    2020      2020
unique                   1391         3
top      tick tock less blocks   neutral
freq                       36       964
```

[6]:

```
tweets_count = df['sentiment'].value_counts()
tweets_count
```

[6]:

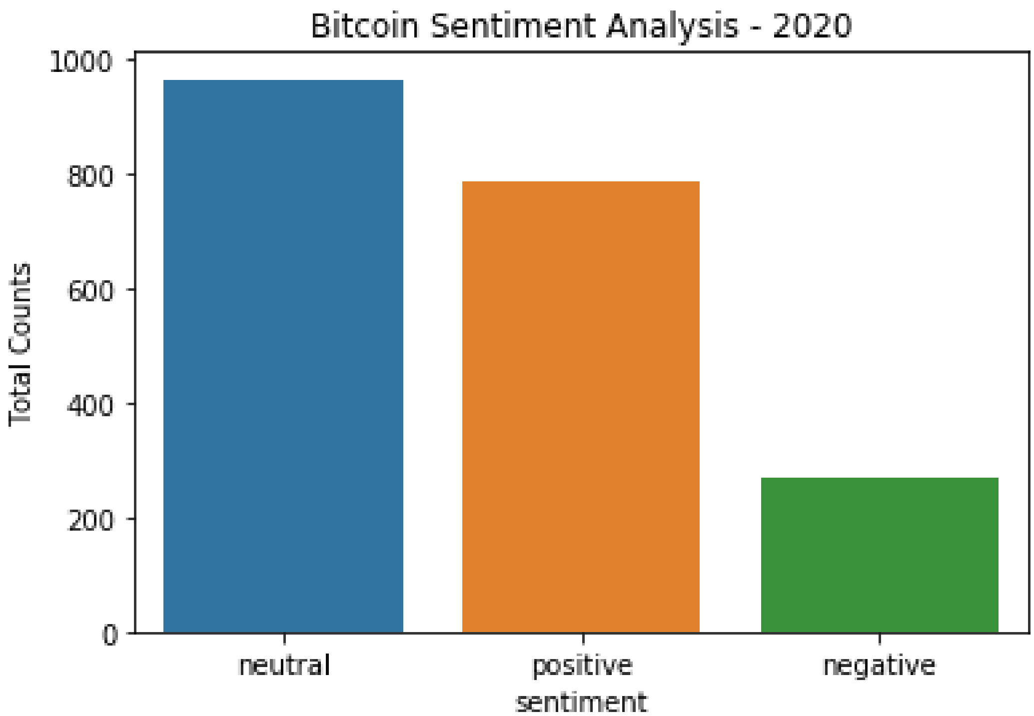
```
neutral     964
positive    788
negative    268
Name: sentiment, dtype: int64
```

[7]:

```
# Plot sentiment mood count
seaborn.countplot(x='sentiment', data=df)
plt.title('Bitcoin Sentiment Analysis - 2020')
```

```
plt.ylabel('Total Counts')
```

[7]: Text(0, 0.5, 'Total Counts')



[8]:
```
# add text length column

df['text_length'] = df['clean_text'].apply(len)
```

[9]:
```
df.head()
```

[9]:
```
                                          clean_text sentiment  text_length
0  goldman sachs hosting client call bitcoin gold…   neutral           60
1                     ok president trump endorsing tommy  positive           34
2  bitcoin day nwhen btc first used commercial tr…  positive           87
3  great icle nif way could avoided know like min…  positive           68
4                          bitcoin btc current price gbp   neutral           29
```

[10]:
```
df.tail()
```

[10]:
```
                                             clean_text sentiment  text_length
2015  xrp xrpcommunity btc bitcoin eth ltc vet oh ye…  positive           96
2016  yes world global pandemic world going recessio…   neutral           56
2017  last days grayscale bitcoin trust bought bitco…   neutral           70
```

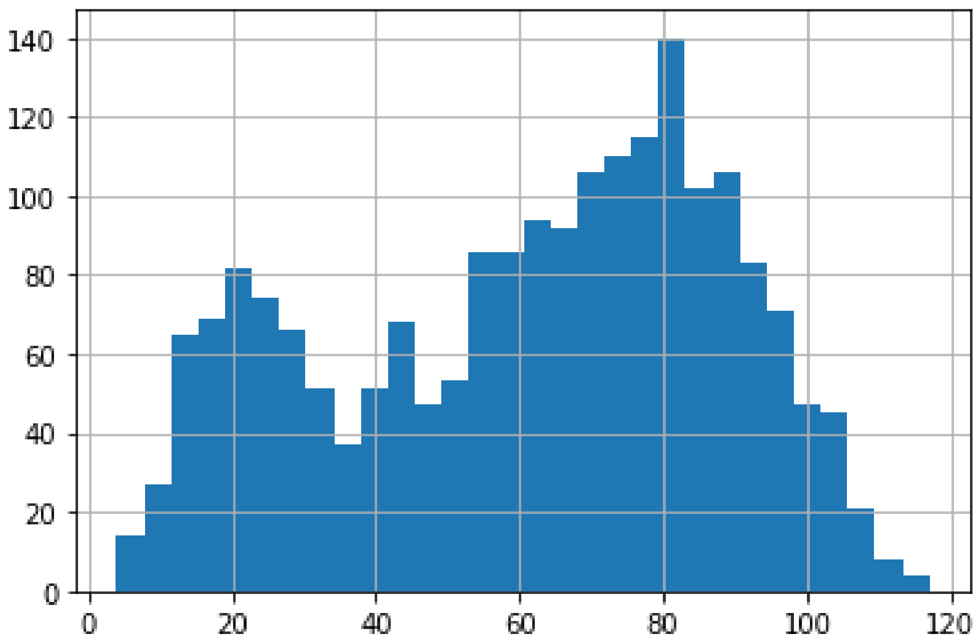
```
2018  pretty incredible imho nwe need creativity lik…  positive          68
2019                                 drop sachs nget sats    neutral           20
```

[11]:
```
df['text_length'].describe()
```

[11]:
```
count    2020.000000
mean       61.235149
std        27.054467
min         4.000000
25%        39.000000
50%        66.500000
75%        83.000000
max       117.000000
Name: text_length, dtype: float64
```

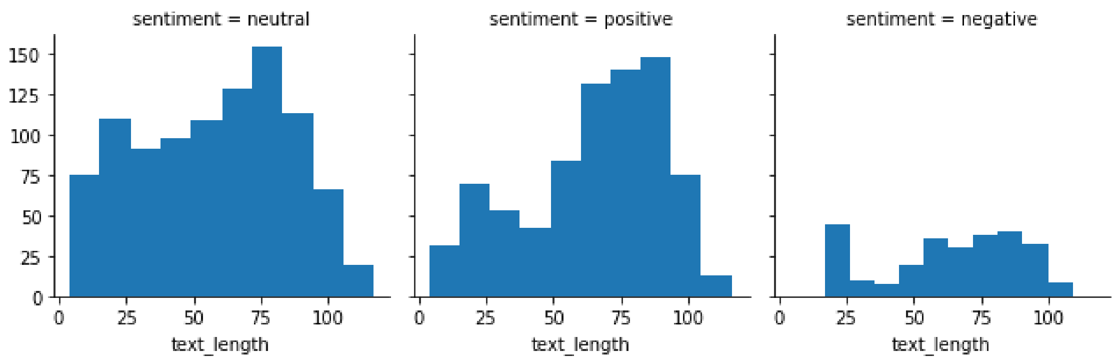
[12]:
```
df['text_length'].hist(bins=30)
```

[12]: <matplotlib.axes._subplots.AxesSubplot at 0x19a0697d148>



[13]:
```
plot = seaborn.FacetGrid(df,col='sentiment')
plot.map(plt.hist,'text_length')
```

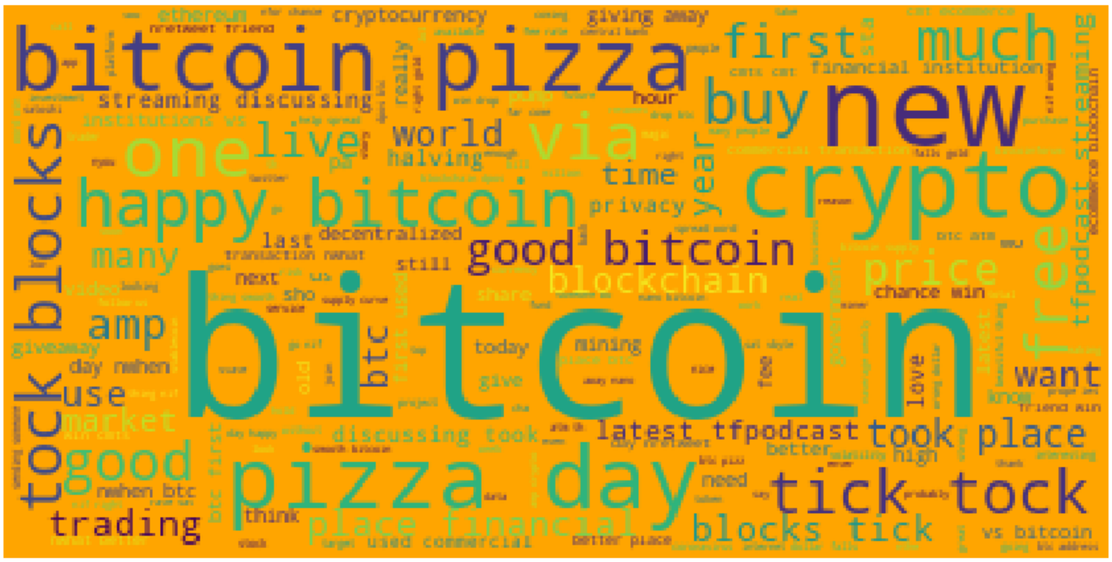
[13]: <seaborn.axisgrid.FacetGrid at 0x19a06a3cd88>



[14]:
```
# visualization using wordcloud for the Neutral tweets

tweets_neutral = df[df['sentiment']=='neutral']
words = ' '.join(tweets_neutral['clean_text'])

wordcloud = WordCloud(background_color='blue').generate(words)

plt.figure(1,figsize=(15, 12))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```

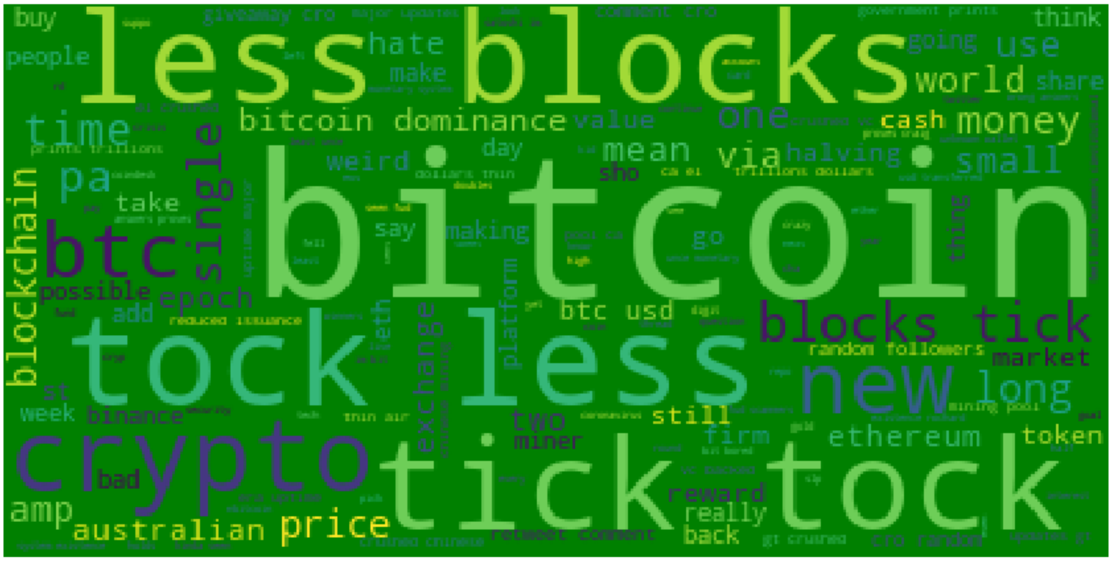
[15]:
```
# visualization using wordcloud for the Positive tweets

tweets_positive = df[df['sentiment']=='positive']
```

```
words = ' '.join(tweets_positive['clean_text'])

wordcloud = WordCloud(background_color='orange').generate(words)

plt.figure(1,figsize=(15, 12))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```



[16]:

```
# visualization using wordcloud for the Negative tweets

tweets_negative = df[df['sentiment']=='negative']
words = ' '.join(tweets_negative['clean_text'])

wordcloud = WordCloud(background_color='green').generate(words)

plt.figure(1,figsize=(15, 12))
plt.imshow(wordcloud)
plt.axis('off')
plt.show()
```



1.2 Training and Testing sets

[33]:
```
# Encode Categorical Variable for Deep Learning
# Apply get_dummies

x = df['clean_text']
y = pd.get_dummies(df['sentiment']).values
num_classes = df['sentiment'].nunique()
```

[34]:
```
# fix random seed for reproducibility
np.random.seed(42)

# Split into 80% train and 20% test sets
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size=0.20,
                                                    stratify=y,
                                                    random_state=42)

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

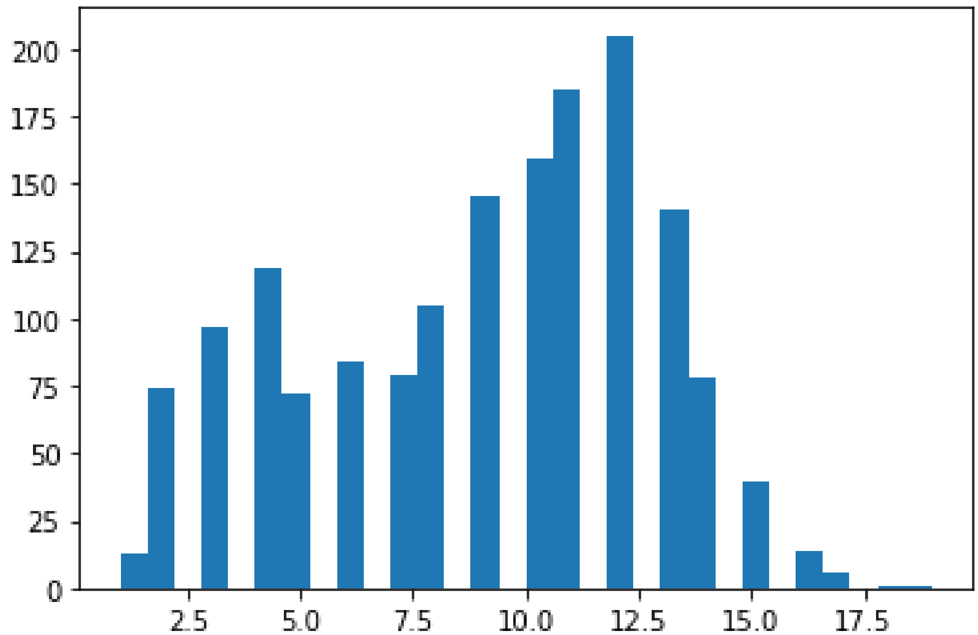
```
(1616,) (404,) (1616, 3) (404, 3)
```

1.2.1 Tokenize Text

[35]:
```
# Tokenize Text

# Define max number of words in dictionary
max_features = 20000
```

```
tokenizer = Tokenizer(num_words=max_features, lower=True)
tokenizer.fit_on_texts(list(x_train))

x_train = tokenizer.texts_to_sequences(x_train)
x_test = tokenizer.texts_to_sequences(x_test)
```

[36]:
```
totalNumWords = [len(feature) for feature in x_train]
plt.hist(totalNumWords,bins = 30)
plt.show()
```



[37]:
```
# Based on the plot histogram, setting feature maxWords
maxWords = 20

x_train = sequence.pad_sequences(x_train, maxlen=maxWords)
x_test = sequence.pad_sequences(x_test, maxlen=maxWords)

print(x_train.shape,x_test.shape)
```

```
(1616, 20) (404, 20)
```

1.3 Model – CNN Long Short-Term Memory Network (CNN LSTM)

[38]:
```
# Create a CNN-LSTM Model

backend.clear_session()
```

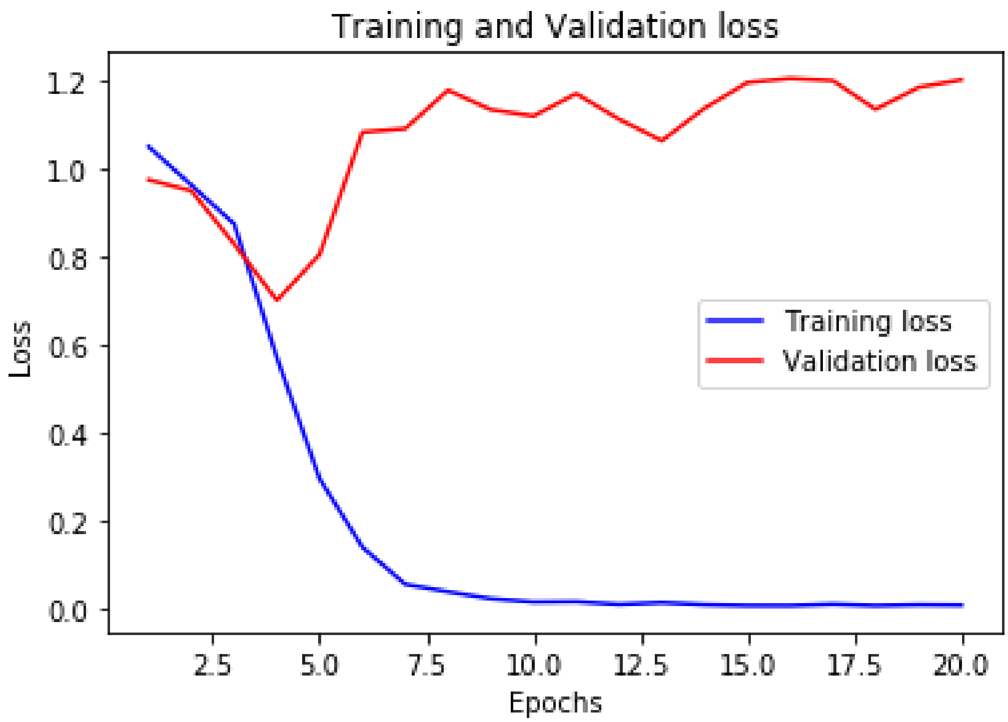
```
embedding_size = 300
epoch = 20
batch_size = 128

model = models.Sequential()
model.add(Embedding(max_features, embedding_size, input_length=x_train.
↪shape[1]))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(Conv1D(filters=32, kernel_size=3, padding='same', activation='relu'))
model.add(MaxPooling1D(pool_size=2))
model.add(LSTM(96, dropout=0.15, recurrent_dropout=0.15))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer = 'adam',
               loss = 'categorical_crossentropy',
               metrics = ['accuracy'])
print(model.summary())


# Train the model
history = model.fit(x_train,
                    y_train,
                    epochs = epoch,
                    batch_size = batch_size,
                    validation_data=(x_test, y_test),
                    verbose = 1)


history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']
epochs = range(1, len(history_dict['acc']) + 1)

hist = pd.DataFrame(history.history)
print(hist.head())

plt.plot(epochs, loss_values, 'b', label = 'Training loss')
plt.plot(epochs, val_loss_values, 'r', label = 'Validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

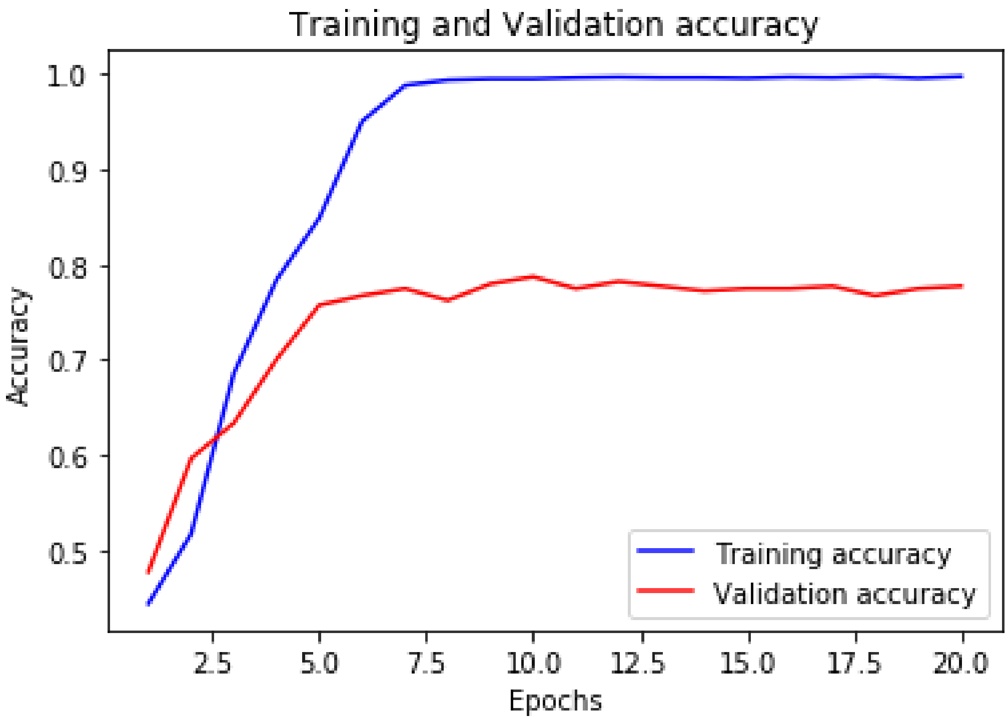
```python
plt.plot(epochs, acc_values, 'b', label = 'Training accuracy')
plt.plot(epochs, val_acc_values, 'r', label = 'Validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

results = model.evaluate(x_test, y_test)
print(model.metrics_names)
print(results)
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 20, 300)           6000000
_________________________________________________________________
conv1d (Conv1D)              (None, 20, 32)            28832
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 10, 32)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 10, 32)            3104
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 5, 32)             0
_________________________________________________________________
lstm (LSTM)                  (None, 96)                49536
_________________________________________________________________
dense (Dense)                (None, 3)                 291
=================================================================
Total params: 6,081,763
Trainable params: 6,081,763
Non-trainable params: 0
_________________________________________________________________
None
Train on 1616 samples, validate on 404 samples
Epoch 1/20
1616/1616 [==============================] - 3s 2ms/sample - loss: 1.0509 - acc:
0.4443 - val_loss: 0.9757 - val_acc: 0.4777
Epoch 2/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.9637 - acc:
0.5173 - val_loss: 0.9507 - val_acc: 0.5965
Epoch 3/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.8756 - acc:
0.6856 - val_loss: 0.8299 - val_acc: 0.6337
Epoch 4/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.5724 - acc:
0.7840 - val_loss: 0.7021 - val_acc: 0.7005
```

```
Epoch 5/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.2973 - acc:
0.8484 - val_loss: 0.8062 - val_acc: 0.7574
Epoch 6/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.1422 - acc:
0.9499 - val_loss: 1.0838 - val_acc: 0.7673
Epoch 7/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0577 - acc:
0.9876 - val_loss: 1.0919 - val_acc: 0.7748
Epoch 8/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0407 - acc:
0.9932 - val_loss: 1.1792 - val_acc: 0.7624
Epoch 9/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0249 - acc:
0.9944 - val_loss: 1.1351 - val_acc: 0.7797
Epoch 10/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0175 - acc:
0.9944 - val_loss: 1.1205 - val_acc: 0.7871
Epoch 11/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0183 - acc:
0.9957 - val_loss: 1.1715 - val_acc: 0.7748
Epoch 12/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0120 - acc:
0.9963 - val_loss: 1.1134 - val_acc: 0.7822
Epoch 13/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0161 - acc:
0.9957 - val_loss: 1.0645 - val_acc: 0.7772
Epoch 14/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0113 - acc:
0.9957 - val_loss: 1.1386 - val_acc: 0.7723
Epoch 15/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0093 - acc:
0.9950 - val_loss: 1.1964 - val_acc: 0.7748
Epoch 16/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0091 - acc:
0.9963 - val_loss: 1.2056 - val_acc: 0.7748
Epoch 17/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0125 - acc:
0.9957 - val_loss: 1.2007 - val_acc: 0.7772
Epoch 18/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0092 - acc:
0.9969 - val_loss: 1.1359 - val_acc: 0.7673
Epoch 19/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0112 - acc:
0.9950 - val_loss: 1.1854 - val_acc: 0.7748
Epoch 20/20
1616/1616 [==============================] - 2s 1ms/sample - loss: 0.0104 - acc:
0.9969 - val_loss: 1.2025 - val_acc: 0.7772
```

	loss	acc	val_loss	val_acc
0	1.050863	0.444307	0.975717	0.477723
1	0.963667	0.517327	0.950701	0.596535
2	0.875615	0.685644	0.829926	0.633663
3	0.572408	0.784035	0.702105	0.700495
4	0.297252	0.848391	0.806181	0.757426






```
404/404 [==============================] - 0s 135us/sample - loss: 1.2025 - acc:
0.7772
['loss', 'acc']
[1.2024905970781157, 0.7772277]
```

1.4 Model Evaluation

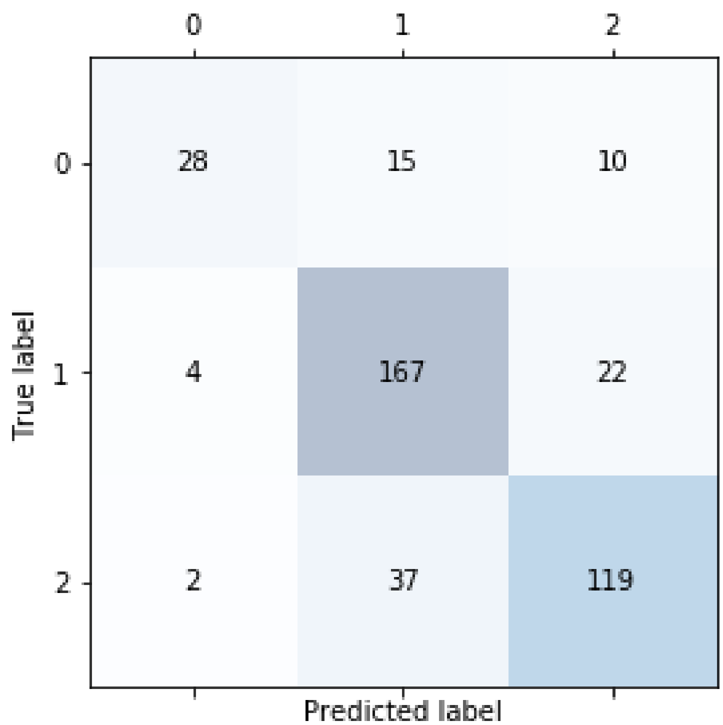
[39]:

```
# Reference link: https://stackoverflow.com/questions/44189119/
↪how-to-plot-confusion-matrix-correctly

# Evaluate model with Test set

def model_evaluation():
    # predict class with test set
    y_pred_test =  model.predict_classes(x_test, batch_size=batch_size,␣
↪verbose=0)
    print('Accuracy:\t{:0.1f}%'.format(accuracy_score(np.
↪argmax(y_test,axis=1),y_pred_test)*100))

    #classification report
    print('\n')
    print(classification_report(np.argmax(y_test,axis=1), y_pred_test))

    #confusion matrix
```

```python
    confmat = confusion_matrix(np.argmax(y_test,axis=1), y_pred_test)

    fig, ax = plt.subplots(figsize=(4, 4))
    ax.matshow(confmat, cmap=plt.cm.Blues, alpha=0.3)
    for i in range(confmat.shape[0]):
        for j in range(confmat.shape[1]):
            ax.text(x=j, y=i, s=confmat[i, j], va='center', ha='center')
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.tight_layout()
```

[40]:
```python
print(model_evaluation())
```

```
Accuracy:       77.7%


              precision    recall  f1-score   support

           0       0.82      0.53      0.64        53
           1       0.76      0.87      0.81       193
           2       0.79      0.75      0.77       158

    accuracy                           0.78       404
   macro avg       0.79      0.72      0.74       404
weighted avg       0.78      0.78      0.77       404

None
```


0
1
2
0
1
2
28
15
10
4
167
22
2
37
119
True label
Predicted label