AmazonTechnicalProject

April 14, 2020

1 Option 1: Sentiment Identification

Kyle Killion

1.1 BACKGROUND

A large multinational corporation is seeking to automatically identify the sentiment that their customer base talks about on social media. They would like to expand this capability into multiple languages. Many 3rd party tools exist for sentiment analysis, however, they need help with under-resourced languages.

1.2 GOAL

Train a sentiment classifier (Positive, Negative, Neutral) on a corpus of the provided documents. Your goal is to maximize accuracy. There is special interest in being able to accurately detect negative sentiment. The training data includes documents from a wide variety of sources, not merely social media, and some of it may be inconsistently labeled. Please describe the business outcomes in your work sample including how data limitations impact your results and how these limitations could be addressed in a larger project.

1.3 Data Set Information:

Tagged for Sentiment (Positive, Negative, Neutral)

1.4 Attribute Information:

Each record comprises of two string datatype values. One for Comment/Review and the second for sentiment.

1.5 Relevant Papers:

- Sharf, Zareen, and Saif Ur Rahman. â€œPerforming Natural Language Processing On Roman Urdu Datasets.' IJCSNS (January 2018 Volume)
- Sharf, Zareen, and Saif Ur Rahman. 'Lexical normalization of roman Urdu text.' IJCSNS 17.12 (2017): 213.

These were very informative papers over the foundations of the language and what issues you could find yourself running into. Lexical codification was an effort explained, in my mind, as scouring the internet for all combinations of a word and counting the most often deeming it the correct manner since there isn't a standardized format. This was helpful 70% of the time and found some failure cases where the standarization overlapped into other words. It was also mentioned that English words are also found sprinkled within the ecosystem of social media regarding Roman Urdu.

- Muhammad Arslan Manzoor, Saqib Mamoon, Song Kei Tao, Ali Zakir, Muhammad Adil, Jianfeng Lu "Lexical Variation and Sentiment Analysis of Roman Urdu Sentences with Deep Neural Networks" (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 11, No. 2, 2020 https://thesai.org/Downloads/Volume11No2/Paper_90-Lexical_Variation_and_Sentiment_Analysis.pdf

This paper as well goes into the complications of Roman Urdu and the need for standardization/normalization of the language.

"The Roman script does not follow any standard which makes it more complicated than English language dataset. Different spelling refer to same word and identical spelling refer to different contextual words. This phenomenon confuses the embedding of vocabulary and motivated us for normalization of Roman Urdu sentences. Previously, some approaches have been used for normalization purposes to reduce the variation of embedding for the same word: Urdu phone, Similarity function, LexC clustering algorithm, Stemming and Lemmatizing [27], [28]. These approaches depend upon some rules and there is 30% to 40% chance of failure attributed to these rules. Making a set of similar words and clipping suffix or prefix can negatively influence the embedding behavior towards sentence polarity."

1.6 DATA

Link to data: http://archive.ics.uci.edu/ml/datasets/Roman+Urdu+Data+Set

[1]:
```
import re
import pickle
import pandas as pd
from tqdm import tqdm
import numpy as np


from nltk.tokenize import RegexpTokenizer
from gensim import corpora, models
from sklearn.preprocessing import LabelEncoder

from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

plt.rcParams['figure.dpi'] = 150
```

```
unable to import 'smart_open.gcs', disabling that module
```

1.7 Data Exploration -

[2]:
```
df = pd.read_csv(r'http://archive.ics.uci.edu/ml/machine-learning-databases/
↪00458/Roman%20Urdu%20DataSet.csv', header=None)
df.columns = ['text', 'sent','mystery']
df = df.astype(str)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20229 entries, 0 to 20228
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   text     20229 non-null  object
 1   sent     20229 non-null  object
 2   mystery  20229 non-null  object
dtypes: object(3)
memory usage: 474.2+ KB
```

[3]:
```
df.head()
```

[3]:
```
                                                text      sent mystery
0  Sai kha ya her kisi kay bus ki bat nhi hai lak…  Positive     nan
1                                         sahi bt h  Positive     nan
2                                       Kya bt hai,  Positive     nan
3                                        Wah je wah  Positive     nan
4                              Are wha kaya bat hai  Positive     nan
```

1.8 Mystery column will be ommited due to Attribute Information only mentioning Comment/Review and Sentiment

[4]:
```
# Comes in Nan String instead of a real Nan Nan
df.mystery.replace('nan', np.nan, inplace=True)
df.dropna()
```

[4]:
```
                                                    text      sent  \
13637                               movie abi b baki h    Neutral
13653                         Hahahahahaha bilkul sahi    Neutral
14218  tjhe ase mar na chahti hun  tjhe nae tu achi b…  Negative
14810                      Yr tym pta chali kb ata raat m?  Positive
17161  Kya khatab g ledy type ka sahafi la k betha diya   Negative
19499                   kabhi bhai ki bhi aesi pic lele :P   Neutral
19780  Jahil awam ko jahil leader ki hi zroorat hai, …  Negative

                mystery
13637  ----------------
13653        ----------
```

```
14218          -------
14810           ------
17161             9090
19499        till here
19780        till here
```

[5]:
```python
del df['mystery']
```

1.9 Fix the Neative to Negative

[6]:
```python
df['sent'] = df['sent'].str.replace('Neative', 'Negative')
```

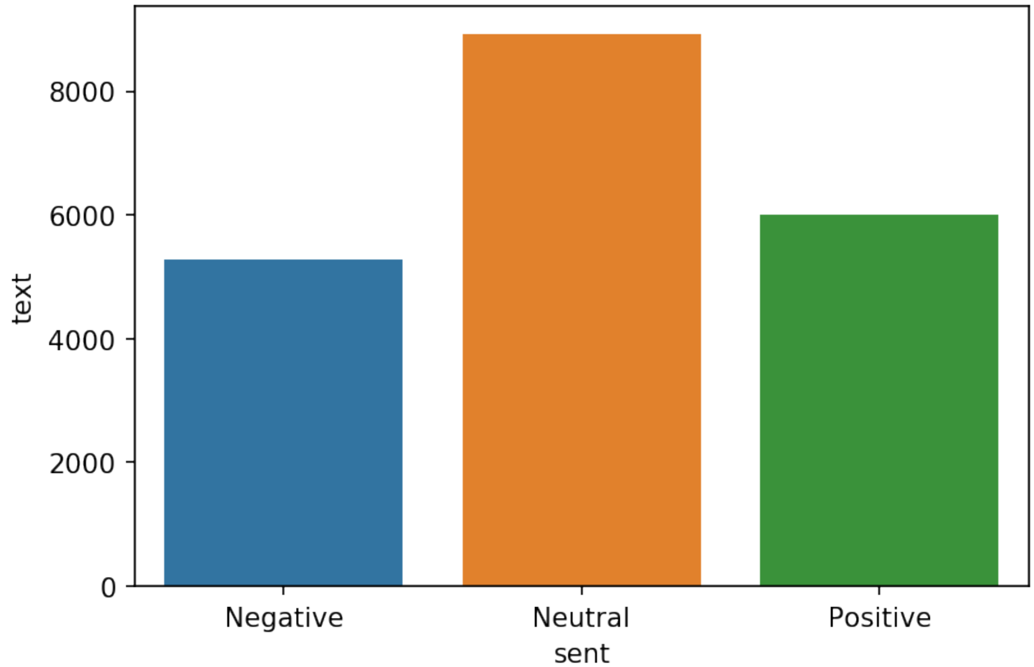
[7]:
```python
data_explore_grp = df.groupby('sent').count()
data_explore_grp
```

[7]:

sent	text
Negative	5287
Neutral	8929
Positive	6013

[8]:
```python
sns.barplot(x='sent', y='text', data=data_explore_grp.reset_index())
```

[8]: <matplotlib.axes._subplots.AxesSubplot at 0x21c6b3b3448>



1.10 Data Preprocessing

1.10.1 Clean Reviews

- Remove stopwords and punctuation
- Set to lowercase
- Tokenize each sentence

1.10.2 Categorize Labels

- **Positive** = 3
- **Negative** = 1
- **Neutral** = 2

1.10.3 Build Emoji Sentiment Class

Scrape the web for emoji sentiment based on the following paper and site:

http://kt.ijs.si/data/Emoji_sentiment_ranking/index.html

- P. Kralj Novak, J. Smailovic, B. Sluban, I. Mozetic, **Sentiment of Emojis**, PLoS ONE 10(12): e0144296, doi:10.1371/journal.pone.0144296, 2015.

Emoji Confidence Feature Feature Creation which attempts to sift through the Pos, Neg, Neut scores and determine which it decidedly favors within the three. Filters prior to

f(x) $= \frac{(x-mu)^2}{sigma}$

- X (here being one of the Pos, Neg, Neut) is filtered if the value isn't greater than .15 and given a zero.
- The deviation has to be greater than .2 from the mean to reject the null that x != x_bar

[9]:
```
try:
  from emoji import UNICODE_EMOJI
except:
  !pip install emoji
  from emoji import UNICODE_EMOJI
from bs4 import BeautifulSoup
import requests as r

class EmojiSent():

    def __init__(self, text):

        # URL param to get emoji sentiment scores
        self.url = r'http://kt.ijs.si/data/Emoji_sentiment_ranking/index.html'
        # pandas series of text
        self.text = text
        # Fetch the Emoji Sentiment Scores
        self._get_emoji_sent()
```

```
    def _get_emoji_sent(self) -> pd.DataFrame:

        print('\nGetting Emoji Sentiment Scores\n')
        html = r.get(self.url).content
        soup = BeautifulSoup(html, 'lxml')

        table = soup.find_all('table')[0]
        self.sentFrame = pd.read_html(str(table))[0]

        # CleanUp the columns
        self.sentFrame.columns =␣
↪['Char','image','Unicodepoint','Occurences','Position','Neg',
                                'Neut','Pos', 'Emoji_SentScore', '_',␣
↪'UnicodeName', 'UnicodeBlock']

        return self.sentFrame


    def text_has_emoji(self, text) -> bool:
        for character in text:
            if character in UNICODE_EMOJI:
                return True
        return False

    def build_sentiment_frame(self) -> pd.DataFrame:

        self._find_emojis()

        print('\nBuilding Emoji Sentiment DataFrame\n')
        self.emoji_sent = self.emoji_match.reset_index()\
        .merge(self.sentFrame, left_on='emoji_match', right_on='Char')\
        .loc[:,['index','Char', 'Position','Neg', 'Neut', 'Pos',␣
↪'Emoji_SentScore']]

        # Multiple Emojis to one index
        # Decided to take the mean of Sentiment within the text
        self.emoji_sent = self.emoji_sent.groupby('index').mean()

        self.text_sentiment = self.emoji_sent.merge(self.text, left_index=True,␣
↪right_index=True)
        self.text_sentiment['emoji_conf'] = self.text_sentiment.loc[:,␣
↪['Neg','Neut','Pos']]\
                                                              .apply(lambda x:␣
↪self.emoji_confidence(x), axis=1)
```

```
        return self.text_sentiment


    def emoji_confidence(self, data) -> int:

        result =  pd.Series([y if y > .2 else np.nan\
                        for y in [np.round((x - np.mean(data))**2 / np.
↪std(data), 3) \
                                  if x > .15 else 0 for x in data]]
                    ).argmax()

        if result < 0:
            return np.nan

        return result + 1


    def _find_emojis(self) -> None:

        print(f'\nFinding Emojis in Text...\n{self.text.head()}')
        text_mask = self.text.apply(self.text_has_emoji)
        self.text = self.text[text_mask == True]

        self.emoji_match = pd.DataFrame()
        for emoji in tqdm(UNICODE_EMOJI, unit=" emojis"):
            try:
                if any(self.text[self.text.str.contains(emoji) == True]):
                    temp = self.text[self.text.str.contains(emoji) == True]
                    self.emoji_match = pd.concat([self.emoji_match, pd.
↪Series(emoji, index=temp.index)])
                else:
                    continue

            # Some Emojis aren't separated in Text and throws an error
            except Exception as e:
                pass

        self.emoji_match.columns = ['emoji_match']
        self.emoji_match.sort_index(inplace=True)
```

[10]:
```
def get_default_learnFrame(df):

    # Convert to a learnFrame from original df, perserve df
    learnFrame = df

    # Clean Reviews
    reviews = cleanReviews(learnFrame['text'], stop=True)
```

```
    learnFrame = pd.merge(learnFrame, pd.Series(reviews,␣
↪name='tokenized_text'), left_index=True, right_index=True)
    revFrame = pd.DataFrame([(' '.join(x)) for x in reviews],␣
↪columns=['cleanReviews'])
    learnFrame = pd.concat([learnFrame, revFrame], axis=1)

    # Encode Labels
    y = df['sent'].values
    targetEncoder = LabelEncoder()
    y = targetEncoder.fit_transform(y)


    learnFrame['sent'] =  y
    # Add 1 to prevent 0 as a label
    learnFrame['sent'] = learnFrame['sent'] + 1


    # Fetch emoji sentiment and confidence
    emo = EmojiSent(learnFrame.text).build_sentiment_frame()

    # Merge and tidy
    learnFrame = pd.merge(learnFrame, emo, left_index=True, right_index=True,␣
↪how='outer').drop(['text_y'], axis=1)
    learnFrame.rename(columns={'text_x' : 'text'}, inplace=True)

    return learnFrame


def cleanReviews(texts, stop=None) -> list:
    """ Takes in a pd.Series of Reviews """
    # Stopwords courtesy and acredited to :
    # Owais Raza
    # https://medium.com/analytics-vidhya/
↪sentiment-analysis-on-roman-urdu-using-python-sklearn-and-nltk-c3a279ef7748

    stopwords=['ai', 'ayi', 'hy', 'hai', 'main', 'ki', 'tha', 'koi', 'ko', 'sy',
            'woh', 'bhi', 'aur', 'wo', 'yeh', 'rha', 'hota', 'ho', 'ga', 'ka',
            'le', 'lye', 'kr', 'kar', 'lye', 'liye', 'hotay', 'waisay', 'gya',
            'gaya', 'kch', 'ab', 'thy', 'thay', 'houn', 'hain', 'han', 'to',
            'is', 'hi', 'jo', 'kya', 'thi', 'se', 'pe', 'phr', 'wala', 'waisay',
            'us', 'na', 'ny', 'hun', 'rha', 'raha', 'ja', 'rahay', 'abi', 'uski',
            'ne', 'haan', 'acha', 'nai', 'sent', 'photo', 'you', 'kafi', 'gai',
            'rhy', 'kuch', 'jata', 'aye', 'ya', 'dono', 'hoa', 'aese', 'de',
            'wohi', 'jati', 'jb', 'krta', 'lg', 'rahi', 'hui', 'karna', 'krna',
            'gi', 'hova', 'yehi', 'jana', 'jye', 'chal', 'mil', 'tu', 'hum',␣
↪'par',
            'hay', 'kis', 'sb', 'gy', 'dain', 'krny', 'tou']
```

```
    tokenizer = RegexpTokenizer(r'(?:^|(?<= ))[a-zA-Z]+(?= |$)')
    reviews = []
    for count, descrip in enumerate(texts.values):
        try:
            # lower
            raw = descrip.lower()

            # token words
            tokens = tokenizer.tokenize(raw)

            # remove stopwords
            if stop:
                tokens = [i for i in tokens if not i in stopwords]

            reviews.append(tokens)

        except AttributeError as a:
            continue

    return reviews


learnFrame = get_default_learnFrame(df)
learnFrame.head()
```

```
Getting Emoji Sentiment Scores


Finding Emojis in Text…
0    Sai kha ya her kisi kay bus ki bat nhi hai lak…
1                                            sahi bt h
2                                          Kya bt hai,
3                                           Wah je wah
4                                  Are wha kaya bat hai
Name: text, dtype: object

100%|     | 2811/2811 [00:05<00:00, 524.14 emojis/s]


Building Emoji Sentiment DataFrame
```

[10]:

```
                                              text  sent  \
0  Sai kha ya her kisi kay bus ki bat nhi hai lak…     3
1                                        sahi bt h     3
2                                      Kya bt hai,     3
```

```
3                                    Wah je wah     3
4                              Are wha kaya bat hai     3

                                    tokenized_text  \
0  [sai, kha, her, kisi, kay, bus, bat, nhi, laki…
1                                  [sahi, bt, h]
2                                           [bt]
3                                   [wah, je, wah]
4                            [are, wha, kaya, bat]

                                    cleanReviews  Position  Neg  Neut  \
0  sai kha her kisi kay bus bat nhi lakin hal kal…       NaN  NaN   NaN
1                                      sahi bt h       NaN  NaN   NaN
2                                             bt       NaN  NaN   NaN
3                                     wah je wah       NaN  NaN   NaN
4                                are wha kaya bat       NaN  NaN   NaN

   Pos  Emoji_SentScore  emoji_conf
0  NaN              NaN         NaN
1  NaN              NaN         NaN
2  NaN              NaN         NaN
3  NaN              NaN         NaN
4  NaN              NaN         NaN
```

1.11 WordClouds

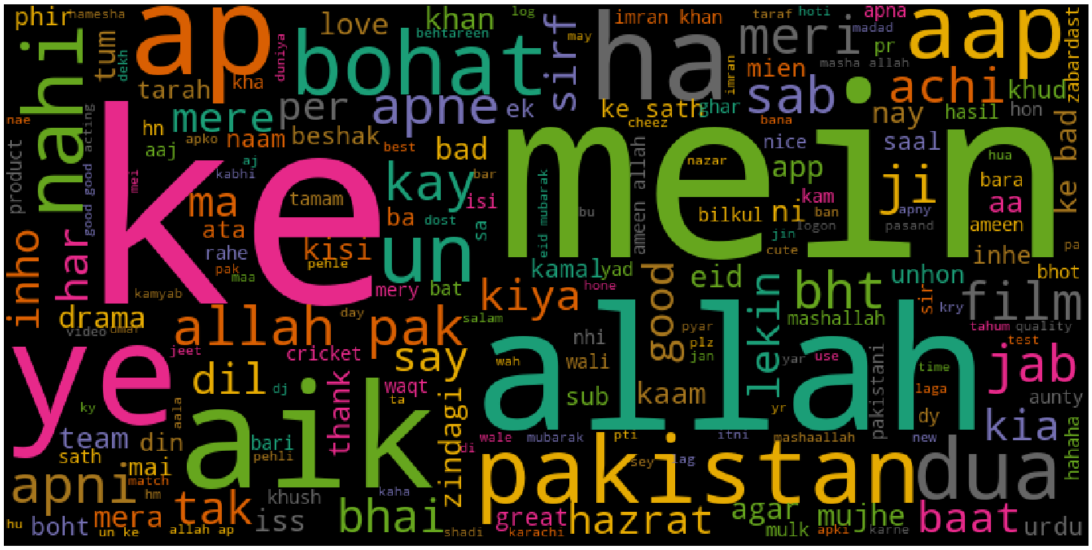
[11]:

```
wc_positive = WordCloud(max_font_size=None, scale = 2, colormap = 'Dark2')\
.generate(learnFrame[learnFrame.sent == 3].cleanReviews.to_string())

wc_negative = WordCloud(max_font_size=None, scale = 2, colormap = 'Dark2')\
.generate(learnFrame[learnFrame.sent == 1].cleanReviews.to_string())

wc_neutral = WordCloud(max_font_size=None, scale = 2, colormap = 'Dark2')\
.generate(learnFrame[learnFrame.sent == 2].cleanReviews.to_string())

fig, ax = plt.subplots(3,1, figsize=(20, 20))

ax[0].imshow(wc_positive)
ax[0].set_title('Positive Reviews Cloud', fontsize = 20)
ax[0].axis("off")

ax[1].imshow(wc_negative)
ax[1].set_title('Negative Reviews Cloud', fontsize = 20)
ax[1].axis("off")

ax[2].imshow(wc_neutral)
ax[2].set_title('Neutral Reviews Cloud', fontsize = 20)
```

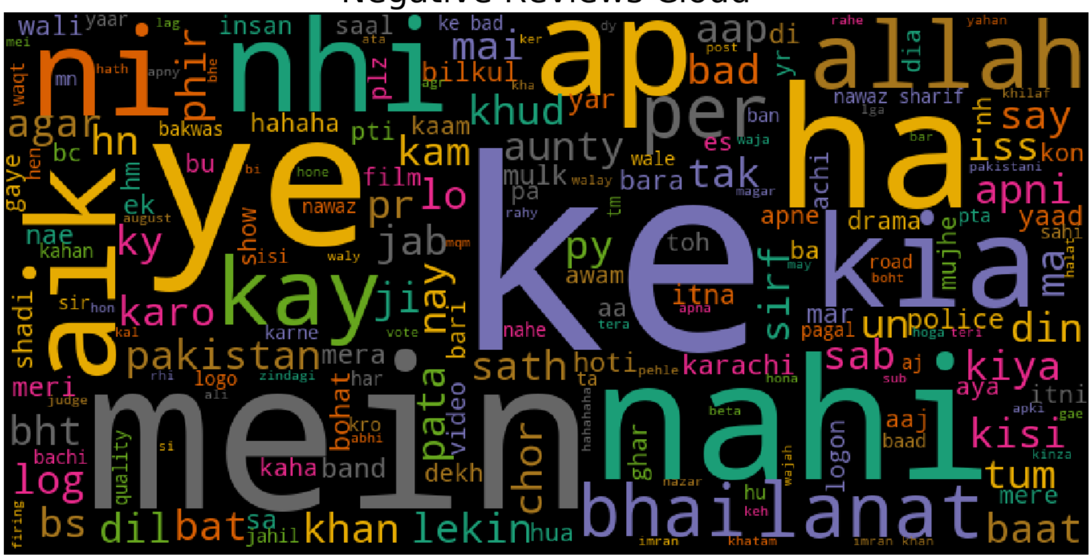
```
ax[2].axis("off")
```

[11]: (-0.5, 799.5, 399.5, -0.5)

Positive Reviews Cloud



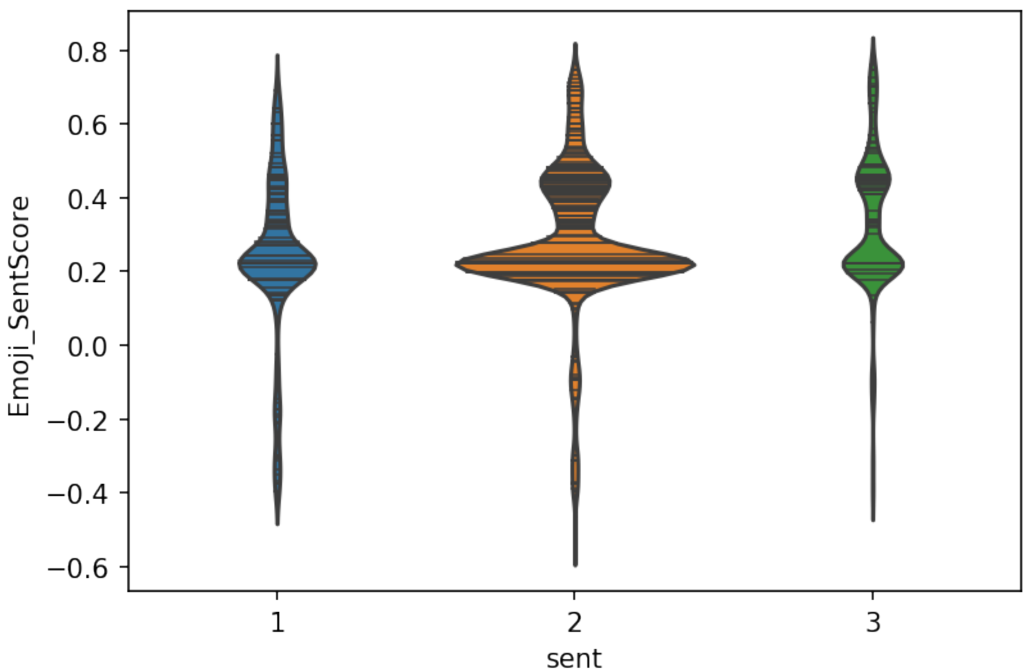
Negative Reviews Cloud



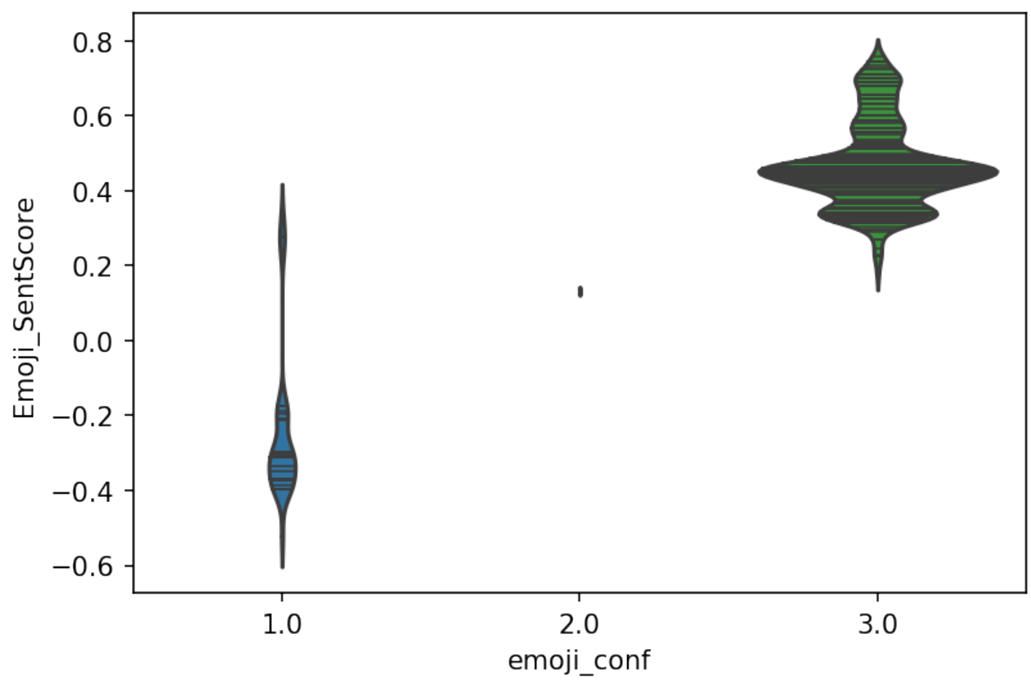
Neutral Reviews Cloud

1.12 Emoji Sentiment Distribution

[12]:
```
sns.violinplot(x='sent', y='Emoji_SentScore', scale='count', inner="stick",
               bw=.2, data=learnFrame.dropna(thresh=1))
plt.show()
sns.violinplot(x='emoji_conf', y='Emoji_SentScore', scale='count',␣
 ↪inner="stick",
               bw=.2, data=learnFrame.dropna())
```



[12]: `<matplotlib.axes._subplots.AxesSubplot at 0x21c730a19c8>`



1.13 Roman Urdu Embeddings - What is an Embedding ?

Here we begin to explore translating the Roman Urdu Language into Vectors of numbers in the space amongst its corpus. For example in English, your knowledge of words constructed like:

`Football = [ 134, 234234, 143124, 14141234, 12341234, 1342424, 97965 ]`

Or we can also explore the sentence as a whole:

`['Football', 'gives', 'you', 'injuries'] = [ 623634, 532, 1346, 96873, 25345, 235, 643 ]`

1.14 Word2Vec

[13]:
```
w2v = models.Word2Vec(learnFrame.tokenized_text, size=50, window=10,␣
 ↪min_count=1, workers=4, seed=43, iter=5)
```

[14]:
```
print(f'Number of reviews : {w2v.corpus_count}\nNumber of Epochs to use : {w2v.
 ↪iter}')
print(f'Total Vectors : {len(w2v.wv.vectors)}')
w2v.wv.vectors[:2]
```

```
Number of reviews : 20229
Number of Epochs to use : 5
Total Vectors : 30178
```

[14]:

```
array([[-0.8238826 , -0.4461032 , -0.4325032 , -3.0964713 , -1.3939183 ,
        -0.5664959 , -0.93210304, -1.0976601 ,  4.1211233 ,  1.122376  ,
         1.0660586 , -1.5519569 ,  0.78365916, -0.5926689 ,  2.685057  ,
        -0.5217332 , -0.13924716,  2.2747908 , -1.2723349 ,  0.33986056,
        -1.8197879 ,  0.9020628 ,  0.6620711 ,  1.4667294 , -1.6158897 ,
         2.5092785 , -2.8915498 ,  1.1281793 ,  1.4682195 ,  2.7365685 ,
         2.1303961 ,  2.4460309 , -1.0366294 , -1.2309722 ,  2.8447988 ,
         2.52849   , -1.4058138 ,  1.638765  , -5.1242757 ,  0.9601724 ,
         2.2923312 ,  0.55055493, -2.6944413 ,  2.2867086 ,  1.4474247 ,
         1.2601706 ,  1.3628854 , -1.2118874 , -3.7526042 , -2.161415  ],
       [-0.8406175 , -0.3523623 , -0.38553706, -3.1970172 , -1.5389323 ,
        -0.5954829 , -0.86650646, -1.0203215 ,  4.102535  ,  1.0687562 ,
         0.8222115 , -1.5025967 ,  0.9511316 , -0.67059094,  2.509272  ,
        -0.47906312, -0.12412572,  2.3723774 , -1.2929142 ,  0.32074204,
        -1.6921949 ,  0.97316056,  0.71381885,  1.24022   , -1.6246711 ,
         2.537393  , -2.8471973 ,  1.1452279 ,  1.3600044 ,  2.8731306 ,
         2.2605355 ,  2.602227  , -0.99599457, -1.1306603 ,  3.0271363 ,
         2.510645  , -1.4327192 ,  1.5918032 , -5.057909  ,  0.9788196 ,
         2.293225  ,  0.4752843 , -2.563385  ,  2.4669929 ,  1.4586138 ,
         1.056494  ,  1.4849403 , -1.1009163 , -3.7507522 , -2.0409162 ]],
      dtype=float32)
```

[15]:

```python
pd.DataFrame(w2v.wv.similar_by_word('aha'), columns=['word','similarity']).
 ↪sort_values(by='similarity')
```

[15]:

	word	similarity
9	larki	0.997825
8	hee	0.997826
7	nuqsan	0.997843
6	pti	0.997857
5	baqi	0.997865
4	daal	0.997876
3	kab	0.997879
2	ham	0.997883
1	tujhe	0.997895
0	kartay	0.997908

1.15 Doc2Vec

My intuition is favoring to go this route over a word2vec/bag of words approach. When considering review-like collections, you really want to preserve the semantics and where words tend to fall within them.

There are two implementations:

- Paragraph Vector - Distributed Memory (PV-DM)
- Paragraph Vector - Distributed Bag of Words (PV-DBOW)

For this we choose the parameter `dm=1` for *PV-DM*

This approach attempts to predict a center word based an average of both context word-vectors and the full document's doc-vector.

[16]:
```
from gensim.models import Doc2Vec
from gensim.models.doc2vec import TaggedDocument

def build_docVecModel(learnFrame, vec_size=10):
    doc_token_text = learnFrame.tokenized_text.values
    document_for_reviews = [TaggedDocument(name, [i]) for i, name in␣
↪enumerate(doc_token_text)]

    # Build Doc2Vec Model
    docVecModel = Doc2Vec(document_for_reviews, dm=1, min_count=1,␣
↪vector_size=vec_size, window=10, workers=3)
    print(f'Total Words in Corpus : {docVecModel.corpus_total_words}')

    return docVecModel, document_for_reviews

docVecModel, document_for_reviews = build_docVecModel(learnFrame)
```

```
Total Words in Corpus : 194627
```

[17]:
```
window = 5
doc_id = 13543

inferred_vector = docVecModel.infer_vector(document_for_reviews[doc_id].words)
sims = docVecModel.docvecs.most_similar([inferred_vector], topn=len(docVecModel.
↪docvecs))

print(f'Model Details Params : {docVecModel}\n')
print('Test Document ({}): «{}»\n'.format(doc_id, ' '.
↪join(document_for_reviews[doc_id].words)))

topThree = [x[0] for x in sims[:window]]
bottomThree = [x[0] for x in sims[-window:]]
midpoint = len(sims)//2
neutralThree = [x[0] for x in sims[midpoint- (window - 2) : midpoint +␣
↪(window-1)]]

print(f"Sentiment Label : {learnFrame.iloc[doc_id]['sent']}")

learnFrame[learnFrame.index.isin(topThree)]
```

```
Model Details Params : Doc2Vec(dm/m,d10,n5,w10,s0.001,t3)

Test Document (13543): «galaa nahi khrab es»
```

```
Sentiment Label : 2
```

[17]:

```
                                                    text  sent  \
1447    Jab ke inho ne 356 one day matches mein kul 5…     3
5283                                       Chalo choro     2
7780                                      Bohat khooob     2
12591   Main nahi janta, Main nahi manta, Aese dastoo…     1
14452                           fatkari hai ya pathkari     2

                                          tokenized_text  \
1447   [jab, ke, inho, one, day, matches, mein, kul, …
5283                                      [chalo, choro]
7780                                     [bohat, khooob]
12591                              [nahi, nahi, dastoor]
14452                                [fatkari, pathkari]

                                        cleanReviews  Position  Neg  Neut  \
1447   jab ke inho one day matches mein kul wickets a…       NaN  NaN   NaN
5283                                       chalo choro       NaN  NaN   NaN
7780                                      bohat khooob       NaN  NaN   NaN
12591                                nahi nahi dastoor       NaN  NaN   NaN
14452                                fatkari pathkari       NaN  NaN   NaN

       Pos  Emoji_SentScore  emoji_conf
1447   NaN              NaN         NaN
5283   NaN              NaN         NaN
7780   NaN              NaN         NaN
12591  NaN              NaN         NaN
14452  NaN              NaN         NaN
```

[18]:

```python
def build_train_test_frame(docVecModel, learnFrame):

    docFrame = pd.DataFrame(docVecModel.docvecs.vectors_docs,
                 columns = [f'docVec_{count}' for count in range(docVecModel.
 ↪vector_size)])
    learnDocFrame = pd.merge(learnFrame, docFrame, right_index=True,␣
 ↪left_index=True)
    learnDocFrame = learnDocFrame.loc[:, lambda df: (df.dtypes == np.float32) |␣
 ↪\
                                  (df.dtypes == np.float64) | (df.dtypes ==␣
 ↪np.int32)]

    # One Hot Emoji Confidence
    oneHotFrame = pd.get_dummies(learnDocFrame.emoji_conf.dropna().astype(int))
    oneHotFrame.columns = ['emoji_conf_neg','emoji_conf_neut','emoji_conf_pos']
```

```
    # Merge up with Doc2Vec Embeddings
    learnDocFrame = pd.merge(learnDocFrame, oneHotFrame, left_index=True,␣
↪right_index=True, how='outer')\
    .drop(['emoji_conf'], axis=1)

    # Divide up the data 10% Final out-of-sample Testing
    testFrame = learnDocFrame.sample(frac=.1, random_state=43)
    trainFrame = learnDocFrame.loc[~learnDocFrame.index.isin(testFrame.index)]

    return trainFrame, testFrame

trainFrame, testFrame = build_train_test_frame(docVecModel, learnFrame)
trainFrame.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 18206 entries, 0 to 20228
Data columns (total 19 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   sent            18206 non-null  int32
 1   Position        1001 non-null   float64
 2   Neg             1001 non-null   float64
 3   Neut            1001 non-null   float64
 4   Pos             1001 non-null   float64
 5   Emoji_SentScore 1001 non-null   float64
 6   docVec_0        18206 non-null  float32
 7   docVec_1        18206 non-null  float32
 8   docVec_2        18206 non-null  float32
 9   docVec_3        18206 non-null  float32
 10  docVec_4        18206 non-null  float32
 11  docVec_5        18206 non-null  float32
 12  docVec_6        18206 non-null  float32
 13  docVec_7        18206 non-null  float32
 14  docVec_8        18206 non-null  float32
 15  docVec_9        18206 non-null  float32
 16  emoji_conf_neg  414 non-null    float64
 17  emoji_conf_neut 414 non-null    float64
 18  emoji_conf_pos  414 non-null    float64
dtypes: float32(10), float64(8), int32(1)
memory usage: 2.0 MB
```

1.16 Construct Classifiers

[19]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC, LinearSVC
```

```
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
from sklearn.exceptions import ConvergenceWarning
from sklearn.utils._testing import ignore_warnings
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import multilabel_confusion_matrix, accuracy_score,␣
↪precision_score

# list of (estimator, param_grid), where param_grid is used in GridSearchCV
classifiers = [
    (GaussianNB(), {
    }),
    (LogisticRegression(random_state=43), {
        'C': [int(x) for x in np.round(np.logspace(1, 2, 5), 0).tolist()]
    }),
    (LinearSVC(random_state=43), {
        'C': [int(x) for x in np.round(np.logspace(1, 2, 5), 0).tolist()]
    }),
    (GradientBoostingClassifier(n_estimators=50, random_state=43), {
        'learning_rate': np.round(np.logspace(-4, 1, 5), 4)
    }),
    (SVC(random_state=43), {
        'C': [int(x) for x in np.round(np.logspace(1, 2, 5), 0).tolist()]
    }),
    (RandomForestClassifier(random_state=43), {
        'max_depth' : [2, 6, 10],
        'n_estimators' : [int(x) for x in np.round(np.logspace(1, 2, 5), 0).
↪tolist()]
    })
]
```

1.17 Training Classifiers and Predict

[20]:
```
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedShuffleSplit


def train_classifiers(trainFrame, strat=False) -> dict:

    X = trainFrame.loc[:, 'Position':].dropna().values
    y = trainFrame.dropna()['sent'].values

    # Standardize - mean = 0, std = 1
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                         test_size=.2,
                                                         random_state=43)
```

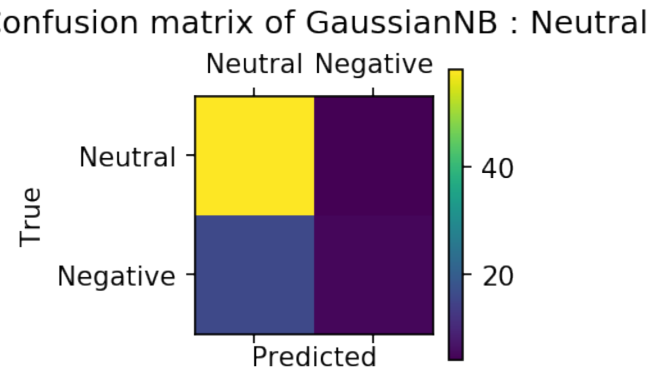
```
    # 80/20 training/testing
    if not strat:
        clfs = {}
        names = [ e.__class__.__name__ for e, g in classifiers]

        # Run thru the classifiers
        for name, (estimator, param_grid) in tqdm(zip(names, classifiers),␣
↪unit='clf'):

            print(name)
            clf = GridSearchCV(estimator=estimator, param_grid=param_grid)
            print(f"HyperParams: {param_grid}")
            with ignore_warnings(category=ConvergenceWarning):
                clf.fit(X_train, y_train)

            y_pred = clf.predict(X_test)

            precision = precision_score(y_test, y_pred, average=None)
            score = accuracy_score(y_test, y_pred)
            conf_mat = multilabel_confusion_matrix(y_test, y_pred)

            clfs[f'{name}'] = {'Classifier' : clf,
                               'Accuracy' : score,
                               'Precision' : precision,
                               'ConfMatrix' : conf_mat}
        return clfs

    # Stratified training/testing
    else:
        splitter = StratifiedShuffleSplit(n_splits=5, test_size=.2,␣
↪random_state=43)
        print(f'Number of Splits : {splitter.n_splits}')

        clfs = {}

        names = [ e.__class__.__name__ for e, g in classifiers]

        # Run thru the classifiers
        for est_idx, (name, (estimator, param_grid)) in \
                tqdm(enumerate(zip(names, classifiers))):

            clf = GridSearchCV(estimator=estimator, param_grid=param_grid)

            for count, (train_index, test_index) in enumerate(splitter.split(X,␣
↪ y), 1):
                print("TRAIN:", train_index, "TEST:", test_index)
```

```
                X_train, X_test = StandardScaler().
↪fit_transform(X[train_index]),
                StandardScaler().fit_transform(X[test_index])

                y_train, y_test = y[train_index], y[test_index]

                with ignore_warnings(category=ConvergenceWarning):
                    clf.fit(X_train, y_train)

                y_pred = clf.predict(X_test)
                precision = precision_score(y_test, y_pred, average=None)
                score = accuracy_score(y_test, y_pred)
                conf_mat = multilabel_confusion_matrix(y_test, y_pred)
                clfs[f'{name}_{count}_{est_idx}'] = {'Classifier' : clf,
                                                     'Accuracy' : score,
                                                     'Precision' : precision,
                                                     'ConfMatrix' : conf_mat}
        return clfs
```

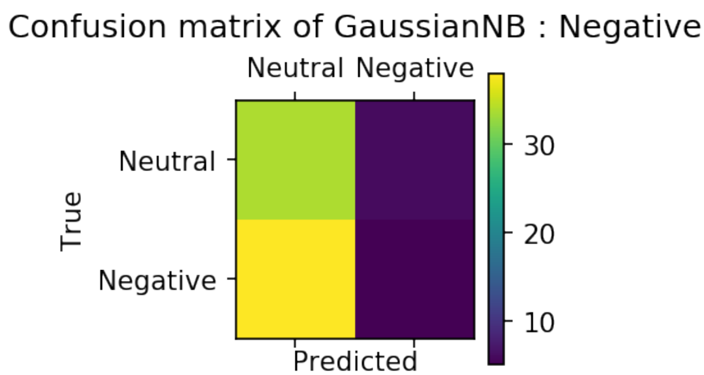
[21]:
```
def model_metrics(clfs):

    labels=['Neutral','Negative','Positive']

    for k in clfs.keys():
        print(f"\n{k} Accuracy : {clfs[k]['Accuracy']}")
        for count, sent in enumerate(labels):
            classifier = k

            print(f"\nConfMatrix {sent} :\n{classifier}")

            try:

                fig = plt.figure(figsize= (2,2))
                ax = fig.add_subplot(111)
                cax = ax.matshow(clfs[k]['ConfMatrix'][count])
                plt.title(f'Confusion matrix of {classifier} :␣
↪{labels[count]}\n')
                fig.colorbar(cax)
                ax.set_xticklabels([''] + labels)
                ax.set_yticklabels([''] + labels)
                plt.xlabel('Predicted')
                plt.ylabel('True')
                plt.show()

            except Exception as e:
                print(f"Didn't get a plot : {str(e)}")
```

```
[22]: clfs = train_classifiers(trainFrame)
model_metrics(clfs)
```

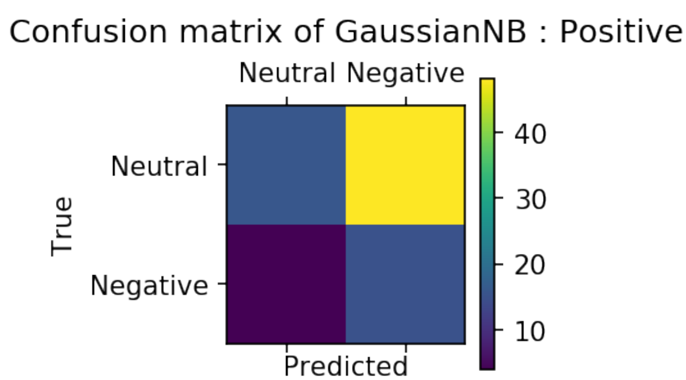
```
0clf [00:00, ?clf/s]

GaussianNB
HyperParams: {}
LogisticRegression
HyperParams: {'C': [10, 18, 32, 56, 100]}

2clf [00:00,  3.34clf/s]

LinearSVC
HyperParams: {'C': [10, 18, 32, 56, 100]}

3clf [00:02,  1.37clf/s]

GradientBoostingClassifier
HyperParams: {'learning_rate': array([1.000e-04, 1.800e-03, 3.160e-02,
5.623e-01, 1.000e+01])}

4clf [00:07,  2.17s/clf]

SVC
HyperParams: {'C': [10, 18, 32, 56, 100]}

5clf [00:08,  1.60s/clf]

RandomForestClassifier
HyperParams: {'max_depth': [2, 6, 10], 'n_estimators': [10, 18, 32, 56, 100]}

6clf [00:13,  2.19s/clf]


GaussianNB Accuracy : 0.30120481927710846

ConfMatrix Neutral :
GaussianNB
```



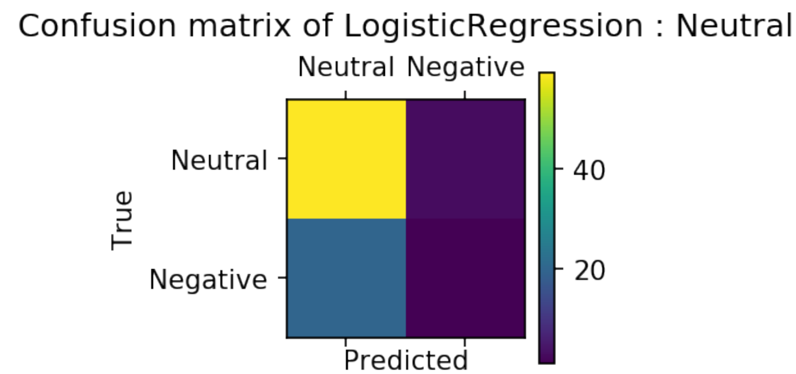
ConfMatrix Negative :
GaussianNB



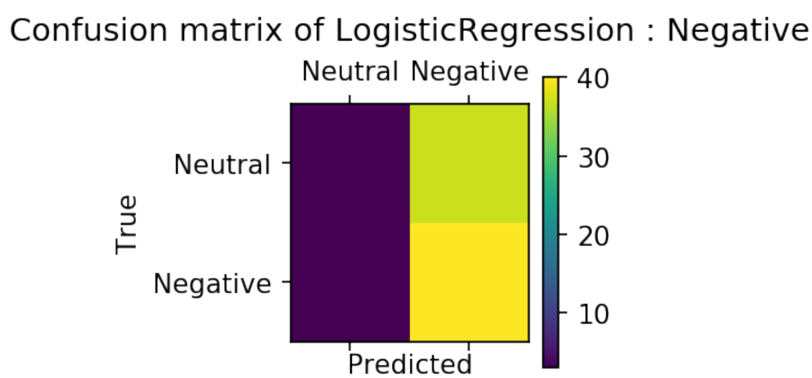
ConfMatrix Positive :
GaussianNB



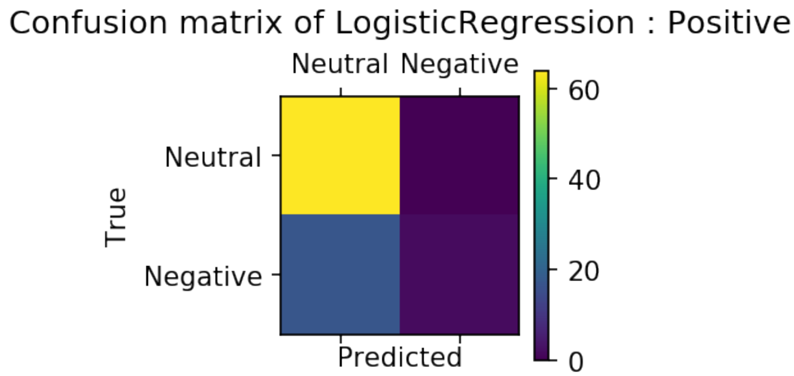
```
LogisticRegression Accuracy : 0.5180722891566265

ConfMatrix Neutral :
LogisticRegression
```



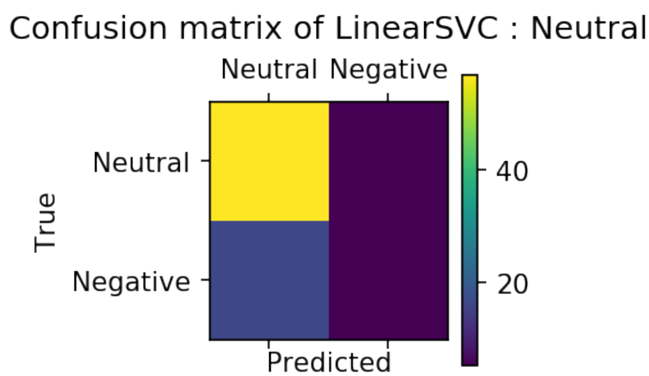

```
ConfMatrix Negative :
LogisticRegression
```



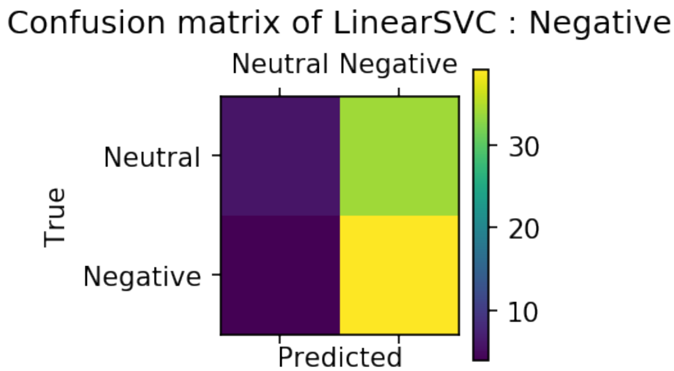

```
ConfMatrix Positive :
LogisticRegression
```



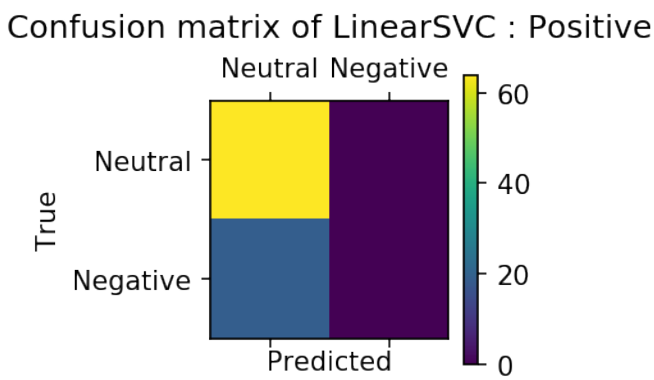

```
LinearSVC Accuracy : 0.5301204819277109

ConfMatrix Neutral :
LinearSVC
```



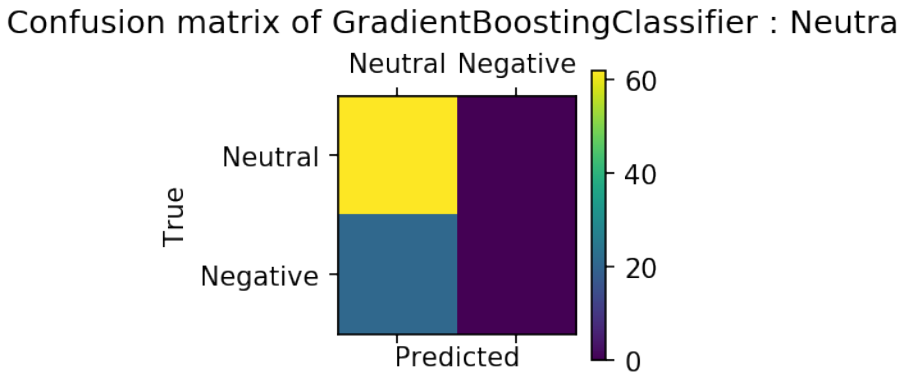

```
ConfMatrix Negative :
LinearSVC
```



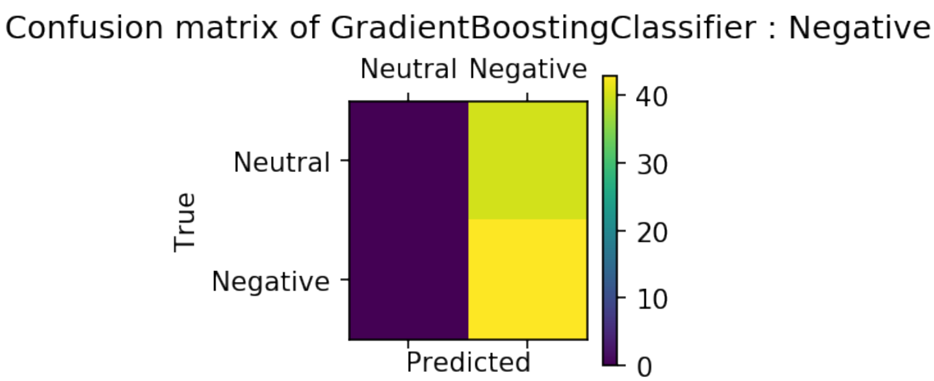

```
ConfMatrix Positive :
LinearSVC
```



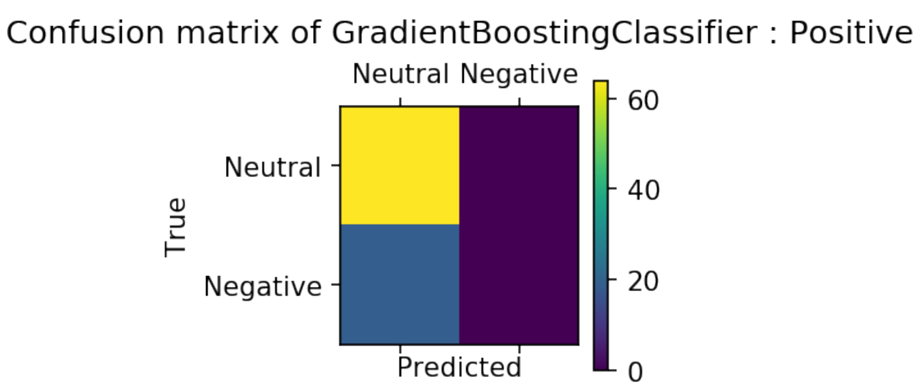

```
GradientBoostingClassifier Accuracy : 0.5180722891566265

ConfMatrix Neutral :
GradientBoostingClassifier
```



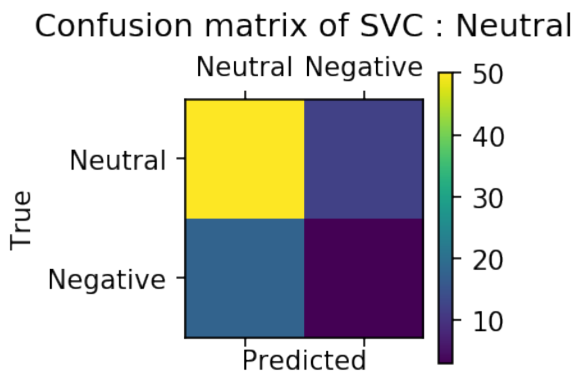
ConfMatrix Negative :
GradientBoostingClassifier



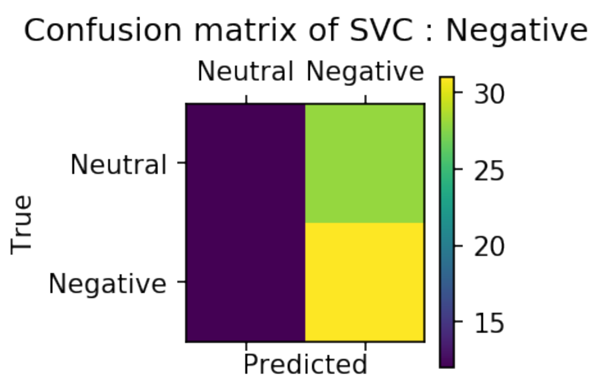
ConfMatrix Positive :
GradientBoostingClassifier



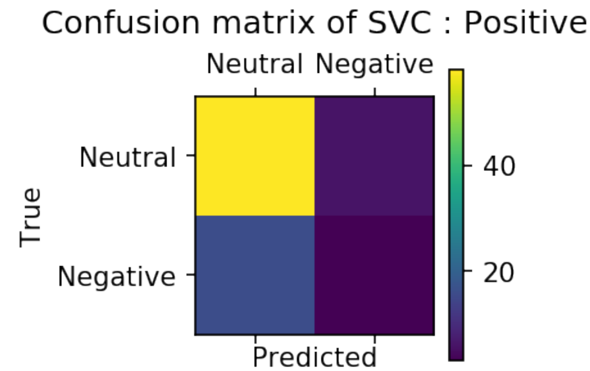
```
SVC Accuracy : 0.4457831325301205

ConfMatrix Neutral :
SVC
```



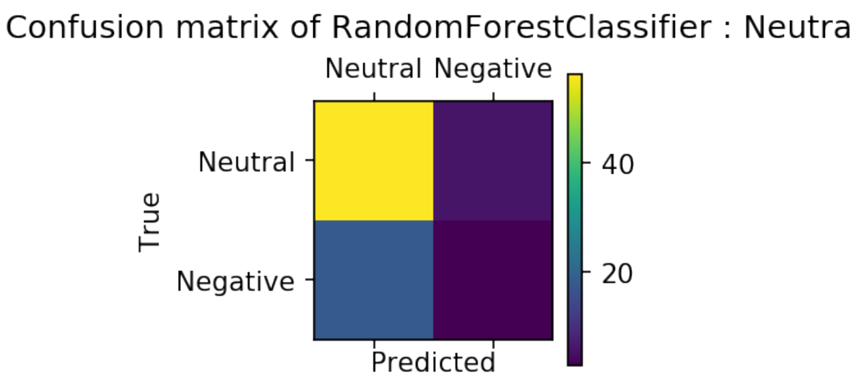

```
ConfMatrix Negative :
SVC
```



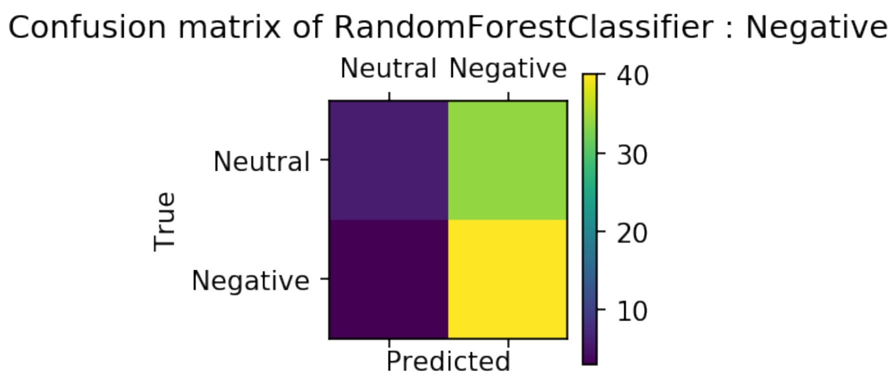

```
ConfMatrix Positive :
SVC
```



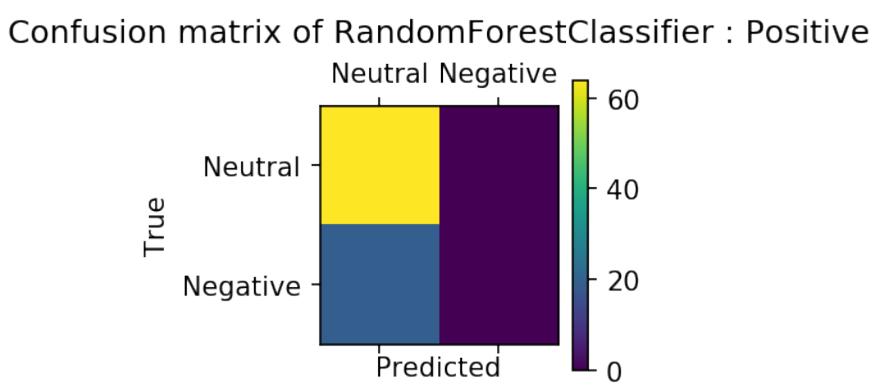

```
RandomForestClassifier Accuracy : 0.5180722891566265

ConfMatrix Neutral :
RandomForestClassifier
```




```
ConfMatrix Negative :
RandomForestClassifier
```




```
ConfMatrix Positive :
RandomForestClassifier
```



1.18 Out-of-Sample Testing

[25]:
```
for k, v in clfs.items():
    print(f"{k} : {np.round(clfs[k]['Accuracy'], 2)}\nPrecision :␣
 ↪{clfs[k]['Precision']}")
```

```
GaussianNB : 0.3
Precision : [0.55555556 0.45454545 0.23809524]
LogisticRegression : 0.52
Precision : [0.25       0.51948052 1.        ]
LinearSVC : 0.53
Precision : [0.5        0.53424658 0.        ]
GradientBoostingClassifier : 0.52
Precision : [0.         0.51807229 0.        ]
SVC : 0.45
```

```
Precision : [0.2        0.52542373 0.33333333]
RandomForestClassifier : 0.52
Precision : [0.33333333 0.54054054 0.        ]
```

[26]:
```
# Test on out of sample data
X_Final = X = StandardScaler().fit_transform(testFrame.loc[:, 'Position':].
 ↪dropna().values)
y_Final = testFrame.dropna().loc[:,['sent']].values


clf = clfs['LogisticRegression']['Classifier']
pred = clf.predict(X_Final)
actual = y_Final

conf_mat_Final = multilabel_confusion_matrix(actual, pred)
precision = precision_score(actual, pred, average=None)
acc = accuracy_score(actual, pred)
print(f'Final Test :\n {conf_mat_Final}\n')
print(f'Accuracy : {acc}')
print(f'Precision score : Neutral : {precision[0]}, Negative : {precision[1]},␣
 ↪Positive : {precision[2]}')

print(f'Predictions: {pred}')
```

```
Final Test :
 [[[29  2]
  [ 8  2]]

 [[ 2 14]
  [ 2 23]]

 [[35  0]
  [ 6  0]]]

Accuracy : 0.6097560975609756
Precision score : Neutral : 0.5, Negative : 0.6216216216216216, Positive : 0.0
Predictions: [2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 1 2 2 2 2 2 2 2 2 2 2 2 2 2 1 2
2 2 2 2
 2 2 2 1]
```

1.19 Neural Network - Embeddings, CNN, Pooling, and LSTM

[31]:
```
from keras.utils import to_categorical
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers import Dense,  Dropout, Activation
from keras.layers import Embedding
```

```python
from keras.layers import Conv1D, LSTM, MaxPooling1D
from sklearn.model_selection import train_test_split


# Retrain Doc2Vec model with a wider vector space
# docVecModel_new = build_docVecModel(learnFrame, vec_size=80)
# trainFrame, testFrame = build_train_test_frame(docVecModel_new, learnFrame)

dictionary = corpora.Dictionary(learnFrame.tokenized_text)
idx_seq = [dictionary.doc2idx(word) for word in learnFrame.tokenized_text]

# set parameters:
max_features = len(dictionary.cfs.keys())+1
maxlen = max(len(x) for x in idx_seq )

batch_size = 300
hidden_dims = 250
epochs = 7

# Embedding
embedding_dims = 500

# Convolution
filters = 250
kernel_size = 3

# Pooling Features
pool_size = 4

# LSTM
lstm_output_size = 240

# Recap Layer
hidden_dims = 240

# Training
batch_size = 32
epochs = 7

# Embedding Index Sequences
y = learnFrame.loc[:,['sent']].values
X = idx_seq
print(f'Targets: {np.unique(y)}')

x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=.2,
                                                    random_state=43)
```

```
# Arrange for categorical crossentropy
# probabilities for multiple classes
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')


print('Pad sequences (samples x times)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print('Build model...')
model = Sequential()

# we start off with an efficient embedding layer which maps
# our vocab indices into embedding_dims dimensions
model.add(Embedding(max_features,
                    embedding_dims,
                    input_length=maxlen))

model.add(Dropout(0.3))
# Start to Feed the Network with our data
# Trying to filter contexts
model.add(Conv1D(
                 filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1)
         )

model.add(MaxPooling1D(pool_size=pool_size))

model.add(LSTM(lstm_output_size))
model.add(Dropout(0.3))

model.add(Dense(hidden_dims))
model.add(Activation('relu'))


model.add(Dense(4))
model.add(Activation('softmax'))
```

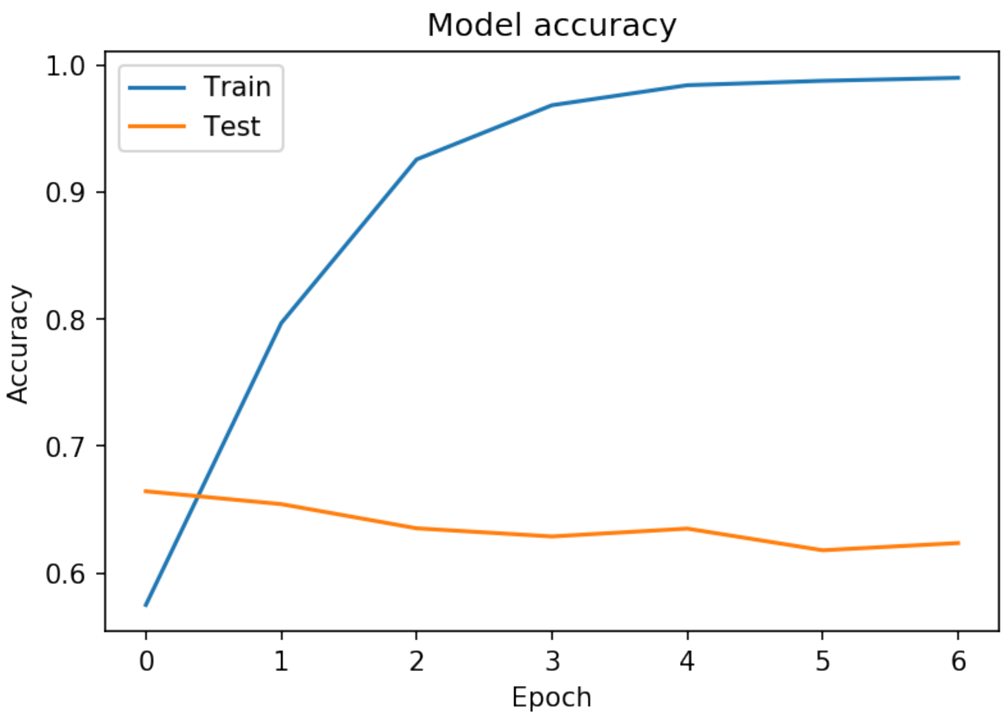
```python
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])

print('Train...')
print(model.summary())
hist1 = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test), verbose=1)
```

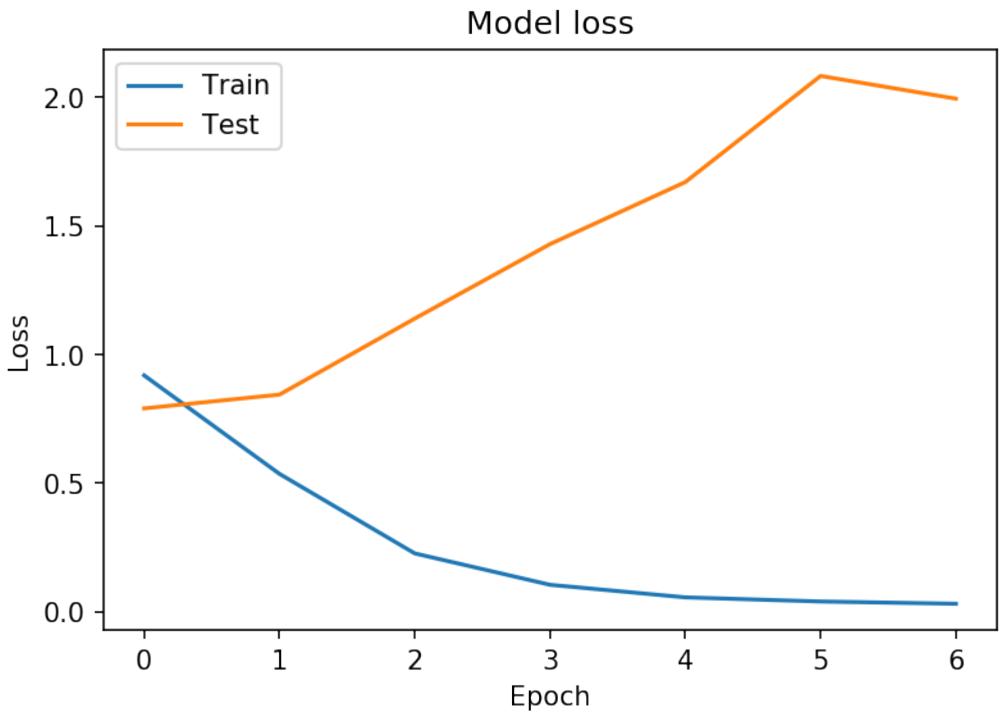
```
Targets: [1 2 3]
16183 train sequences
4046 test sequences
Pad sequences (samples x times)
x_train shape: (16183, 190)
x_test shape: (4046, 190)
Build model…
Train…
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, 190, 500)          15089500
_________________________________________________________________
dropout_1 (Dropout)          (None, 190, 500)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 188, 250)          375250
_________________________________________________________________
max_pooling1d_1 (MaxPooling1 (None, 47, 250)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 240)               471360
_________________________________________________________________
dropout_2 (Dropout)          (None, 240)               0
_________________________________________________________________
dense_1 (Dense)              (None, 240)               57840
_________________________________________________________________
activation_1 (Activation)    (None, 240)               0
_________________________________________________________________
dense_2 (Dense)              (None, 4)                 964
_________________________________________________________________
activation_2 (Activation)    (None, 4)                 0
=================================================================
Total params: 15,994,914
Trainable params: 15,994,914
Non-trainable params: 0
```

```
-----------------------------------------------------------------
None
Train on 16183 samples, validate on 4046 samples
Epoch 1/7
16183/16183 [==============================] - 346s 21ms/step - loss: 0.9193 -
categorical_accuracy: 0.5749 - val_loss: 0.7911 - val_categorical_accuracy:
0.6644
Epoch 2/7
16183/16183 [==============================] - 348s 22ms/step - loss: 0.5366 -
categorical_accuracy: 0.7968 - val_loss: 0.8445 - val_categorical_accuracy:
0.6542
Epoch 3/7
16183/16183 [==============================] - 374s 23ms/step - loss: 0.2274 -
categorical_accuracy: 0.9257 - val_loss: 1.1401 - val_categorical_accuracy:
0.6352
Epoch 4/7
16183/16183 [==============================] - 376s 23ms/step - loss: 0.1053 -
categorical_accuracy: 0.9684 - val_loss: 1.4295 - val_categorical_accuracy:
0.6288
Epoch 5/7
16183/16183 [==============================] - 472s 29ms/step - loss: 0.0564 -
categorical_accuracy: 0.9841 - val_loss: 1.6707 - val_categorical_accuracy:
0.6349
Epoch 6/7
16183/16183 [==============================] - 469s 29ms/step - loss: 0.0404 -
categorical_accuracy: 0.9876 - val_loss: 2.0833 - val_categorical_accuracy:
0.6179
Epoch 7/7
16183/16183 [==============================] - 460s 28ms/step - loss: 0.0317 -
categorical_accuracy: 0.9899 - val_loss: 1.9950 - val_categorical_accuracy:
0.6236
```

[32]:
```
def plot_model(hist):

    # Plot training & validation accuracy values
    plt.plot(hist.history['categorical_accuracy'])
    plt.plot(hist.history['val_categorical_accuracy'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

    # Plot training & validation loss values
    plt.plot(hist.history['loss'])
    plt.plot(hist.history['val_loss'])
    plt.title('Model loss')
```

```
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc='upper left')
    plt.show()

plot_model(hist1)
```





[33]:
```
from keras.preprocessing import sequence
from sklearn.model_selection import train_test_split
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.layers import Embedding
from keras.layers import Conv1D, GlobalMaxPooling1D
from keras.utils import to_categorical

testFrame = learnFrame.sample(frac=.1, random_state=43)
trainFrame = learnFrame.loc[~learnFrame.index.isin(testFrame.index)]

dictionary = corpora.Dictionary(learnFrame.tokenized_text)
idx_seq = [dictionary.doc2idx(word) for word in learnFrame.tokenized_text]


# set parameters:
max_features = len(dictionary.cfs.keys())+1
maxlen = max(len(x) for x in idx_seq )
batch_size = 300
embedding_dims = 500
filters = 250
kernel_size = 3
hidden_dims = 250
```

```
epochs = 7

y = learnFrame.loc[:,['sent']].values
X = idx_seq
print(f'Targets: {np.unique(y)}')

x_train, x_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=.2,
                                                    random_state=43)

# Arrange for categorical crossentropy
# probabilities for multiple classes
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

print(len(x_train), 'train sequences')
print(len(x_test), 'test sequences')


print('Pad sequences (samples x times)')
x_train = sequence.pad_sequences(x_train, maxlen=maxlen)
x_test = sequence.pad_sequences(x_test, maxlen=maxlen)
print('x_train shape:', x_train.shape)
print('x_test shape:', x_test.shape)

print('Build model...')
model = Sequential()

# we start off with an efficient embedding layer which maps
# our vocab indices into embedding_dims dimensions
model.add(Embedding(max_features,
                    embedding_dims,
                    input_length=maxlen))

model.add(Dropout(0.3))

# we add a Convolution1D, which will learn filters
# word group filters of size kernel_size
# scan 1 letter at a time
model.add(Conv1D(filters,
                 kernel_size,
                 padding='valid',
                 activation='relu',
                 strides=1))

# we use max pooling, want to look at the Convolution features
# and then summarize them up and take the Max Feature that pops
```

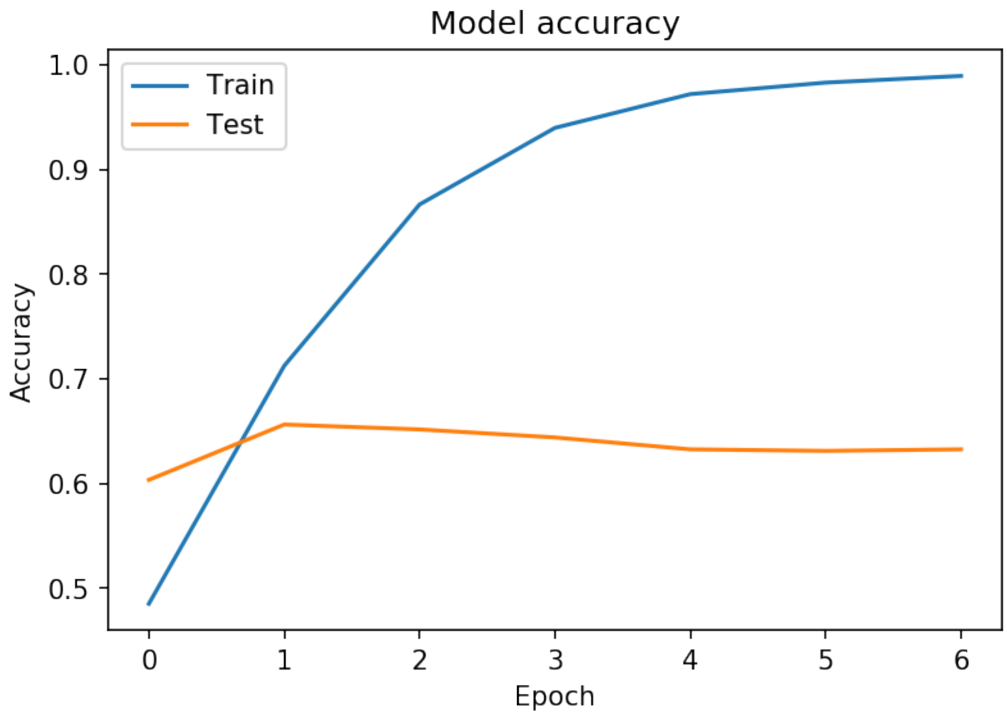
```python
model.add(GlobalMaxPooling1D())

# Adding another hidden layer to learn weighting of the Max Features:
model.add(Dense(hidden_dims))
model.add(Dropout(0.3))
model.add(Activation('relu'))

# Connect to 4 targets output layer
# activate with softmax:
model.add(Dense(4))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['categorical_accuracy'])

print(model.summary())
hist2 = model.fit(x_train, y_train,
          batch_size=batch_size,
          epochs=epochs,
          validation_data=(x_test, y_test), verbose=1)
```

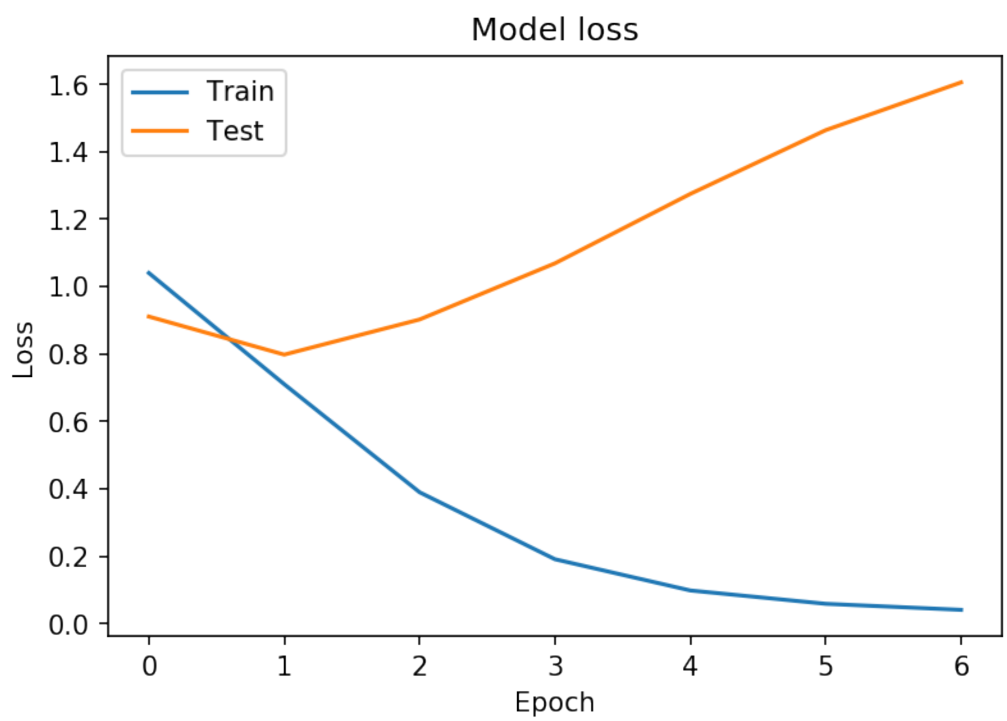
```
Targets: [1 2 3]
16183 train sequences
4046 test sequences
Pad sequences (samples x times)
x_train shape: (16183, 190)
x_test shape: (4046, 190)
Build model…
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, 190, 500)          15089500
_________________________________________________________________
dropout_3 (Dropout)          (None, 190, 500)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 188, 250)          375250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_3 (Dense)              (None, 250)               62750
_________________________________________________________________
dropout_4 (Dropout)          (None, 250)               0
_________________________________________________________________
activation_3 (Activation)    (None, 250)               0
_________________________________________________________________
```

```
dense_4 (Dense)              (None, 4)                 1004
_________________________________________________________________
activation_4 (Activation)    (None, 4)                 0
=================================================================
Total params: 15,528,504
Trainable params: 15,528,504
Non-trainable params: 0
_________________________________________________________________
None
Train on 16183 samples, validate on 4046 samples
Epoch 1/7
16183/16183 [==============================] - 153s 9ms/step - loss: 1.0400 -
categorical_accuracy: 0.4850 - val_loss: 0.9107 - val_categorical_accuracy:
0.6033
Epoch 2/7
16183/16183 [==============================] - 156s 10ms/step - loss: 0.7100 -
categorical_accuracy: 0.7123 - val_loss: 0.7977 - val_categorical_accuracy:
0.6562
Epoch 3/7
16183/16183 [==============================] - 151s 9ms/step - loss: 0.3893 -
categorical_accuracy: 0.8665 - val_loss: 0.9020 - val_categorical_accuracy:
0.6515
Epoch 4/7
16183/16183 [==============================] - 149s 9ms/step - loss: 0.1906 -
categorical_accuracy: 0.9396 - val_loss: 1.0686 - val_categorical_accuracy:
0.6438
Epoch 5/7
16183/16183 [==============================] - 150s 9ms/step - loss: 0.0979 -
categorical_accuracy: 0.9719 - val_loss: 1.2746 - val_categorical_accuracy:
0.6325
Epoch 6/7
16183/16183 [==============================] - 149s 9ms/step - loss: 0.0583 -
categorical_accuracy: 0.9829 - val_loss: 1.4638 - val_categorical_accuracy:
0.6310
Epoch 7/7
16183/16183 [==============================] - 150s 9ms/step - loss: 0.0404 -
categorical_accuracy: 0.9892 - val_loss: 1.6056 - val_categorical_accuracy:
0.6325
```

[34]:

```
plot_model(hist2)
```


Model accuracy
Train
Test
Accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0
1
2
3
4
5
6
Epoch


Model loss
Train
Test
Loss
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
1
2
3
4
5
6
Epoch

1.20 Custom Neural Lemmatization Build

Erick Fonseca, State-of-the-art Multilingual Lemmatization, Mar 11, 2019, https://towardsdatascience.com/state-of-the-art-multilingual-lemmatization-f303e8ff1a8

Business case to undergo a build that aims to aid in the standardization of Roman Urdu with machine learning resulting in lemmetizing the language.

[35]:
```
try:
    import spacy
    import xx_ent_wiki_sm
    nlp = xx_ent_wiki_sm.load()

except:
    !pip install spacy
    # To DL multilanguage set:
    !python -m spacy download xx_ent_wiki_sm --user
    import xx_ent_wiki_sm
    nlp = xx_ent_wiki_sm.load()

from collections import Counter
import string
from pprint import pprint

# Remove pesky punctuation and smart quotes
doc = learnFrame.text.iloc[50:60].str.replace('[^\w\s]', '').apply(nlp)

pprint(doc.apply(lambda x: ([(X.text, X.label_) for X in x.ents])))

# for sent in doc:
#     print(spacy.displacy.render(nlp(str(sent)), jupyter=True, style='ent'))
```

```
50    [(Rehmat Gramophone House, MISC), (Udhar Zinda…
51                                     [(Niazi ki, MISC)]
52                                      [(Lekin us, ORG)]
53                                  [(Radio aur TV, ORG)]
54           [(Lekin, PER), (Attaullah Esakhelvi, PER)]
55    [(Attaullah Esakhelvi, PER), (Mianwali, LOC), …
56    [(Mianwali, LOC), (Attaullah Esakhelvi, PER), …
57    [(Phir, PER), (Pakistan, LOC), (Attaullah Esak…
58    [(Attaullah Esakhelvi, PER), (Attaullah Esakhe…
59                           [(Attaullah Esakhelvi, PER)]
Name: text, dtype: object
```

[38]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img=mpimg.imread("./AmazonTechnicalTest1/spacyPic.png")
imgplot = plt.imshow(img)
plt.axis('off')
```

```
plt.show()
```

Lekin us ORG daur ke mustand singers ne unhe be surr Iqraar de kar mustarld kar dlya

None

Radio aur TV ORG jo maand parhte sitaron ki kafalat he nahi bal ke ubharte hue artists ke hausla afzaal ki riwayaat bhi rakhte hain

None

Lekin PER dosri taraf village dehaat qasbon buses wagons aur chae khano mein Attaullah Esakhelvi PER ki awaaz ka tota bol raha tha

None

Taweel musafat ke liye buses ka ticket lene wale pehle yeh istafsaar bhi karte ke Attaullah Esakhelvi PER ki cassette lagaoge Mianwali LOC ke logon ne inhe bataya ke tumhare naam se Esakhel PER mein hotel khul gae hain

None

Jagah jagah tumhari taswirein lagi hain aur Mianwali LOC mein sirf tumhari cassette he farokht hoti hai to Attaullah Esakhelvi PER ne phir Esakhel PER jane ka qasd kiya

1.21 Conclusion

Please describe the business outcomes in your work sample including how data limitations impact your results and how these limitations could be addressed in a larger project.

1.21.1 Situation

The Roman Urdu data set is a challenging one which exposes a lot of data scientist on some luxuries we have from prior research. Reading through all the papers, you can see the frustrations and yearn for standardization which is pretty messy and unorganized.

But not all was lost being that I was able to locate Roman Urdu stopwords and clean the text for words only. This allowed me to explore and cut a path that could lead to some results for classifying sentiment correctly.

1.21.2 Tasks

Emoji Sentiment *A picture is worth a thousand words* so I wanted to explore emojis and see if there was anything to derive from them. I needed a way to score and also attribute how much of the emoji swayed the classification.

NLP Techniques I needed to build a usable corpus that could produce usable embeddings. Gensim and Keras were used to explore embeddings and ensemble combinations to feed classifiers.

- Gensim - Word2Vec - Word Embeddings that consisted of the Reviews
- Gensim - Doc2Vec - Reviews as a whole Document Embeddings
- Keras API - Deep Learning - Text Sentiment
- SpaCy - POS - Point of Speech

Classifiers

```
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassifier
```

Conduct a GridSearch through them all and explore a range of parameters of Cost and Learning Rate depending on the model.

1.21.3 Actions

I discovered an emoji data set that I was able to leverage and build sentiment with. These were transformed to explore how much confidence did the emojis used contribute into determination of positive, negative, or neutral. The emoji combination would attribute to a average sentiment spread across these three classes. A basic deviation from the mean between them would determine exactly which it was confident in.

Next within the analysis, I decided to pursue Doc2Vec Embeddings accompanied with Emoji Sentiment Confidence. My thought process was to capture the more "birds eye view" of the review which would preserve context and semantics, not just words for more Genearlization. I could preserve semantics and then use these embeddings with all the classifiers, in addition to feeding Neural Nets as first layers of input.

Deep learning with Roman Urdu was the next step. Keras allowed me to build the whole model top to bottom. The model architecture for the best network was as follows: * Embedding layer that injested a sequence of indexes within the corpus. I observed an decrease in val_loss expanding the deminsionality. * Drop about 20% of the neurons here for balance, preventing overfitting, and dependency. * Convolutional which allows me to scan over sentences a letter at a time extracting patterned features * MaxPooling historically is layed next summarizing these features coming from the Convolution Layer. These pooled summaries get a max feature and gives you what "pops" from all the features. * Dense Layer added after pooling to learn all these MaxPooling summary combinations and how they interact. If we went directly to an output layer here we would miss weighting attributing among these before classifying. * Finally our Output layer to settle up and make a classification utilizing a softmax activation. This activation is important here since we want our probabilities to account for the other classes. Sigmoid would range between 0 and 1 but these classes are not independent from another and have a relationship.

1.21.4 Results

Doc2Vec Emoji Classification The Doc2Vec coupled with Emoji Sentiment was terrible and performed horribly. There just wasn't anything the classifiers were able to leverage their weight onto to make some definitive classifications. However, it was great and I felt like I learned quite a bit more about embeddings when trying to classify them. The vector representation of that review broken down into individual features did not offer anything no matter the hyperparamter of cost or learning-rate. This also is indicative of when attempting to feed these same embeddings into a Convolutional Layer. The granularity of word versus the review as a whole made a decidingly big difference.

I abandoned the Doc2Vec embeddings and wanted to see if the emoji sentiment data would benefit in classifying negative sentiment. This also proved somewhat trivial and would like to see more data on it.

Deep Learning Best Resulting Epoch:

```
Epoch 2/7
16183/16183 [==============================] - 156s 10ms/step -
loss: 0.7100
- categorical_accuracy: 0.7123
- val_loss: 0.7977
- val_categorical_accuracy: 0.6562
```

This network is easily and quick to overfit on the text which isn't a very good indication a lot of learning and knowledge is being discovered. The Lexicon codification and standardization issue with Roman Urdu I believe are being observed and show the need. I did not attempt to lemmatize the text which, if done effectively, could prove to be highly advantageous. I start to show previously with SpaCy and how you could utilize POS (Part of Speech) within comments or reviews to start to tackle Lexical Variations and ambiguity. From what I have observed on Twitter, the Roman Urdu language appears to naturally grow with irregular and morph with words as it finds its way much like American, British, and Australian language comes to being and including these irregular combinations within training will be imperative. These would be the data limitations that I would want to be addressed when considering Roman Urdu sentiment analysis within a larger project.