Assignment_9

March 4, 2022

1 MSDS 422 Assignment 9: Natural Language Processing with Disaster Tweets

Kay Quiballo | 03/03/2022

1.1 Management/Research Question

In this competition, you're challenged to build a machine learning model that predicts which Tweets are about real disasters and which one's aren't. You'll have access to a dataset of 10,000 tweets that were hand classified.

1.2 Requirements

Conduct your analysis using a cross-validation design. Conduct EDA. Build at least three RNN models based on hyperparameter tuning. Evaluate goodness of fit metrics. Once you have your best-performing models, classify the test data and submit it to Kaggle. Provide your Kaggle.com user name and screen snapshots of your scores. Discuss your model's performance.

Portions of code to for data cleaning and RNN creation from: Arrants, Tucker (2021) Disaster Tweets - EDA, GloVe, RNNs, BERT (Version 51) [Source code]. https://www.kaggle.com/tuckerarrants/disaster-tweets-eda-glove-rnns-bert

[17]:
```
#python basics
from matplotlib import pyplot as plt
import math, os, re, time, random, string
import numpy as np, pandas as pd, seaborn as sns

#this is just cool
from tqdm import tqdm

#visualization
import matplotlib.pyplot as plt
plt.style.use('ggplot')    #for optimum aesthetics
import seaborn as sns

#natural language processing
```

```python
from collections import defaultdict
import wordcloud

#ignore warnings because they are annoying
import warnings
warnings.filterwarnings('ignore')

#for neural nets
import tensorflow as tf
```

[6]:
```python
def seed_everything(seed):
    os.environ['PYTHONHASHSEED']=str(seed)
    tf.random.set_seed(seed)
    np.random.seed(seed)
    random.seed(seed)

seed_everything(34)
```

[7]:
```python
train = pd.read_csv('./train.csv')
test = pd.read_csv('./test.csv')
train.head()
```

[7]:
```
   id keyword location                                               text  \
0   1     NaN      NaN  Our Deeds are the Reason of this #earthquake M…
1   4     NaN      NaN             Forest fire near La Ronge Sask. Canada
2   5     NaN      NaN  All residents asked to 'shelter in place' are …
3   6     NaN      NaN  13,000 people receive #wildfires evacuation or…
4   7     NaN      NaN  Just got sent this photo from Ruby #Alaska as …

   target
0       1
1       1
2       1
3       1
4       1
```

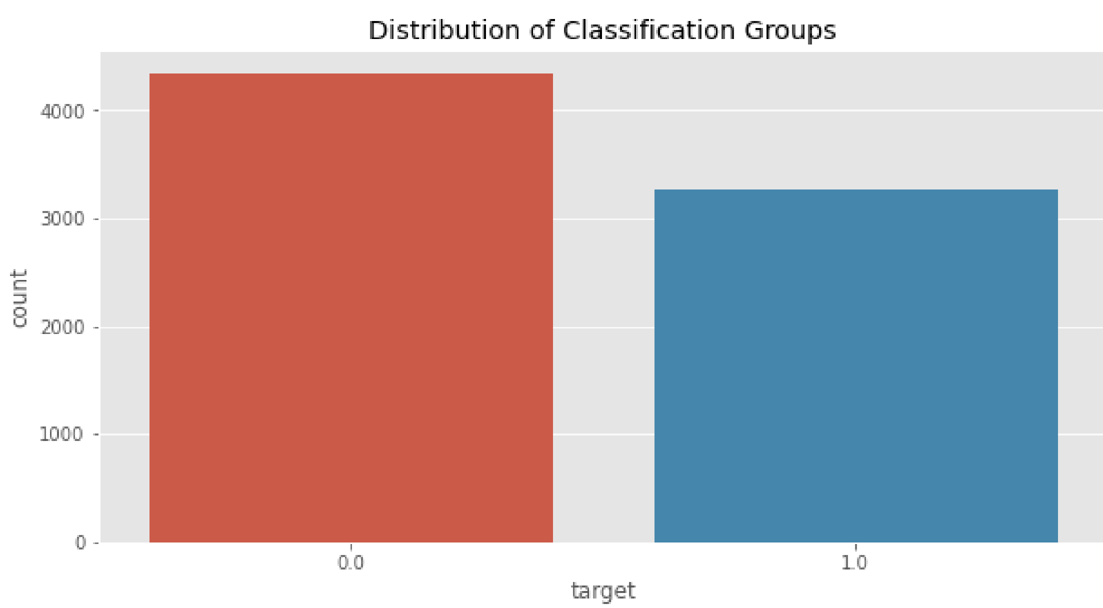
1.3 Data Cleaning

Many of the tweets provided for crisis prediction need to be cleaned before we can feed them into our models.

[8]:
```python
#save ID
test_id = test['id']

#drop from train and test
columns = {'id', 'location'}
```

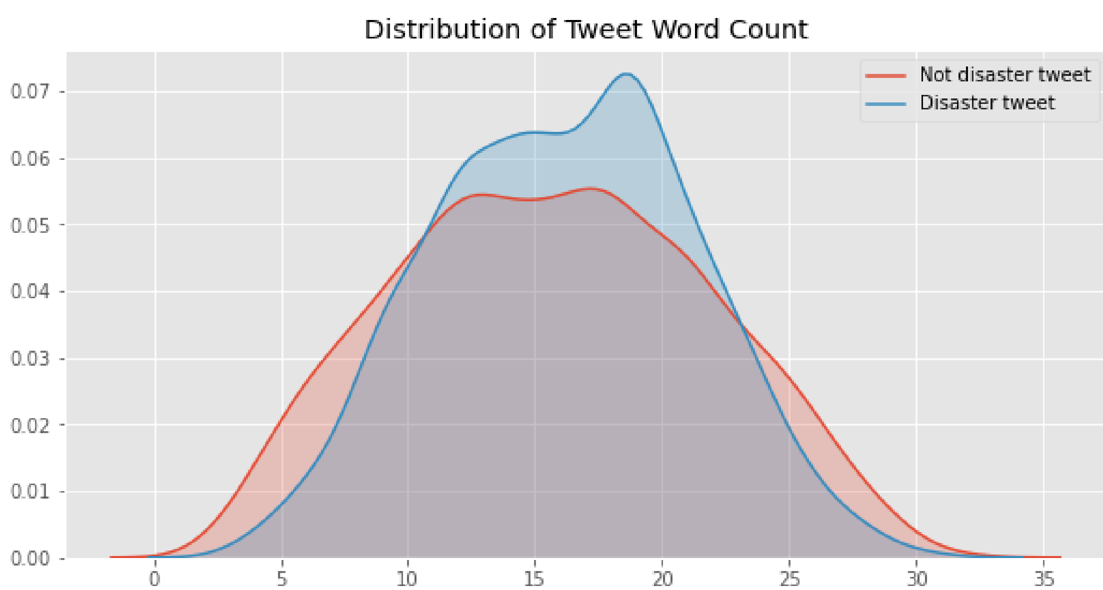
```
train = train.drop(columns = columns)
test = test.drop(columns = columns)

#fill missing with unknown
train['keyword'] = train['keyword'].fillna('unknown')
test['keyword'] = test['keyword'].fillna('unknown')

#add keyword to tweets
train['text'] = train['text'] + ' ' + train['keyword']
test['text'] = test['text'] + ' ' + test['keyword']

#drop fkeyword rom train and test
columns = {'keyword'}
train = train.drop(columns = columns)
test = test.drop(columns = columns)

#combine so we work smarter, not harder
total = train.append(test)
```

[9]:
```
#set figure size
fig, ax = plt.subplots(figsize = (10, 5))

#create graphs
graph1 = sns.countplot(x = 'target', data = total)

#give title and plot
plt.title('Distribution of Classification Groups')
plt.show(graph1)
```



The majority of tweets are not crisis related, but the distribution isn't overly skewed so that we would need to adjust our sample size.

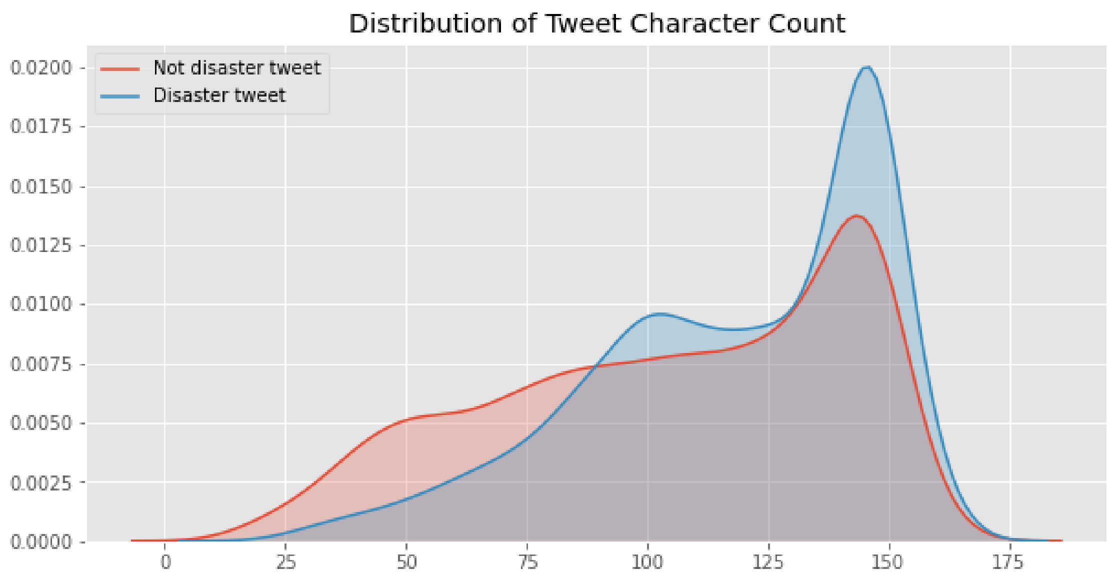
[10]:
```
#create column for the number of words in tweet
total['word count'] = total['text'].apply(lambda x: len(x.split()))

#split so we can use updated train set with new feature
train = total[:len(train)]

#define subplot to see graphs side by side
fig, ax = plt.subplots(figsize = (10, 5))

#create graphs
sns.kdeplot(train['word count'][train['target'] == 0], shade = True, label =␣
↪'Not disaster tweet')
sns.kdeplot(train['word count'][train['target'] == 1], shade = True, label =␣
↪'Disaster tweet')

#set title and plot
plt.title('Distribution of Tweet Word Count')
plt.show()
```



Both the disaster and non-disaster tweets have about 15-20 words on average.

[11]:
```
#create column for the number of characters in a tweet
total['character count'] = total['text'].apply(lambda x: len(x))
```

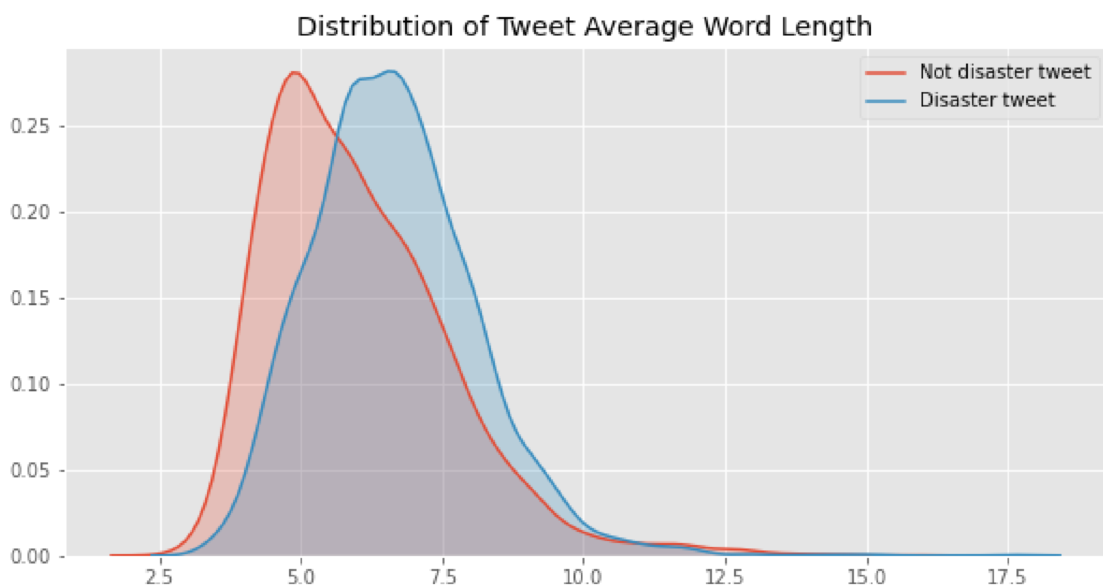
```
#split so we can use updated train set with new feature
train = total[:len(train)]

#define subplot to see graphs side by side
fig, ax = plt.subplots(figsize = (10, 5))

#create graphs
sns.kdeplot(train['character count'][train['target'] == 0], shade = True, label␣
↪= 'Not disaster tweet')
sns.kdeplot(train['character count'][train['target'] == 1], shade = True, label␣
↪= 'Disaster tweet')

#set title and plot
plt.title('Distribution of Tweet Character Count')
plt.show()
```



The distribution of characters is skewed left with the mode being around 125-150 characters per tweet.

[12]:
```
#define function to find average word length
def average_word_length(x):
    x = x.split()
    return np.mean([len(i) for i in x])

#broadcast to text column
total['average word length'] = total['text'].apply(average_word_length)
```

```
#split so we can use updated train set with new feature
train = total[:len(train)]

#define subplot to see graphs side by side
fig, ax = plt.subplots(figsize = (10, 5))

#create graphs
sns.kdeplot(train['average word length'][train['target'] == 0], shade = True,␣
↪label = 'Not disaster tweet')
sns.kdeplot(train['average word length'][train['target'] == 1], shade = True,␣
↪label = 'Disaster tweet')

#set title
plt.title('Distribution of Tweet Average Word Length')

#splot graphs
plt.show()
```



The distributions of average word length is skewed right with a mode of 4-8 characters in the average word.

[62]:
```
#add unique word count
total['unique word count'] = total['text'].apply(lambda x: len(set(x.split())))

#add stopword count
total['stopword count'] = total['text'].apply(lambda x: len([i for i in x.
↪lower().split() if i in wordcloud.STOPWORDS]))
```

```
#add url count
#total['url count'] = total['text'].apply(lambda x: len([i for i in x.lower().
↪split() if 'http' in i or 'https' in i]))

#add mention count
#total['mention count'] = total['text'].apply(lambda x: len([i for i in str(x)␣
↪if i == '@']))

#add hashtag count
#total['hashtag count'] = total['text'].apply(lambda x: len([i for i in str(x)␣
↪if i == '#']))

#add stopword ratio
total['stopword ratio'] = total['stopword count'] / total['word count']

#add punctuation count
total['punctuation count'] = total['text'].apply(lambda x: len([i for i in␣
↪str(x) if i in string.punctuation]))

#split so we can use updated train set
train = total[:len(train)]

disaster = train['target'] == 1
```

[19]:
```
#remove punctuation
def remove_punctuation(x):
    return x.translate(str.maketrans('', '', string.punctuation))

#remove stopwords
def remove_stopwords(x):
    return ' '.join([i for i in x.split() if i not in wordcloud.STOPWORDS])

#remove words less than 4
def remove_less_than(x):
    return ' '.join([i for i in x.split() if len(i) > 3])

#remove words with non-alphabet characters
def remove_non_alphabet(x):
    return ' '.join([i for i in x.split() if i.isalpha()])

def strip_all_entities(x):
    return ' '.join(re.sub("(@[A-Za-z0-9]+)|([^0-9A-Za-z \t])|(\w+:\/\/\S+)","␣
↪",x).split())
```

Punctuation can often ruin the n-grams, and we need them to be as similar as possible when feeding these data into the predictor.

[21]:
```
#install autocorrect
!pip install autocorrect
from autocorrect import Speller

#create function to spell check strings
def spell_check(x):
    spell = Speller(lang='en')
    return " ".join([spell(i) for i in x.split()])

#showcase spellcheck
mispelled = 'Pleaze spelcheck this sentince'
spell_check(mispelled)
```

```
Collecting autocorrect
  Downloading autocorrect-2.6.1.tar.gz (622 kB)
     |                  | 622 kB 2.3 MB/s eta 0:00:01
Building wheels for collected packages: autocorrect
  Building wheel for autocorrect (setup.py) … done
  Created wheel for autocorrect: filename=autocorrect-2.6.1-py3-none-
any.whl size=622380
sha256=54f1fa7416852b4bcc3d2bec1917e4158f0226fb5debcfd376bc3cf6c0e7463e
  Stored in directory: /Users/kagenquiballo/Library/Caches/pip/wheels/72/b8/3b/a
90246d13090e85394a8a44b78c8abf577c0766f29d6543c75
Successfully built autocorrect
Installing collected packages: autocorrect
Successfully installed autocorrect-2.6.1
```

[21]: 'Please spelcheck this sentence'

[22]:
```
PROCESS_TWEETS = False
if PROCESS_TWEETS:
    #apply all of above functions
    total['text'] = total['text'].apply(lambda x: x.lower())
    total['text'] = total['text'].apply(lambda x: re.sub(r'https?://\S+|www\.
↪\S+', '', x, flags = re.MULTILINE))
    total['text'] = total['text'].apply(remove_punctuation)
    total['text'] = total['text'].apply(remove_stopwords)
    total['text'] = total['text'].apply(remove_less_than)
    total['text'] = total['text'].apply(remove_non_alphabet)
    total['text'] = total['text'].apply(spell_check)
```

[23]:
```
contractions = {
"ain't": "am not / are not / is not / has not / have not",
"aren't": "are not / am not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
```

```
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
"haven't": "have not",
"he'd": "he had / he would",
"he'd've": "he would have",
"he'll": "he shall / he will",
"he'll've": "he shall have / he will have",
"he's": "he has / he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how has / how is / how does",
"I'd": "I had / I would",
"I'd've": "I would have",
"I'll": "I shall / I will",
"I'll've": "I shall have / I will have",
"I'm": "I am",
"I've": "I have",
"isn't": "is not",
"it'd": "it had / it would",
"it'd've": "it would have",
"it'll": "it shall / it will",
"it'll've": "it shall have / it will have",
"it's": "it has / it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
```

```
"shan't've": "shall not have",
"she'd": "she had / she would",
"she'd've": "she would have",
"she'll": "she shall / she will",
"she'll've": "she shall have / she will have",
"she's": "she has / she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
"so've": "so have",
"so's": "so as / so is",
"that'd": "that would / that had",
"that'd've": "that would have",
"that's": "that has / that is",
"there'd": "there had / there would",
"there'd've": "there would have",
"there's": "there has / there is",
"they'd": "they had / they would",
"they'd've": "they would have",
"they'll": "they shall / they will",
"they'll've": "they shall have / they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we had / we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what shall / what will",
"what'll've": "what shall have / what will have",
"what're": "what are",
"what's": "what has / what is",
"what've": "what have",
"when's": "when has / when is",
"when've": "when have",
"where'd": "where did",
"where's": "where has / where is",
"where've": "where have",
"who'll": "who shall / who will",
"who'll've": "who shall have / who will have",
"who's": "who has / who is",
"who've": "who have",
"why's": "why has / why is",
```

```
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you had / you would",
"you'd've": "you would have",
"you'll": "you shall / you will",
"you'll've": "you shall have / you will have",
"you're": "you are",
"you've": "you have"
}

contractions_re = re.compile('(%s)' % '|'.join(contractions.keys()))
def expand_contractions(s, contractions = contractions):
    def replace(match):
        return contractions[match.group(0)]
    return contractions_re.sub(replace, s)

expand_contractions("can't stop won't stop")
```

[23]: 'cannot stop will not stop'

[24]:
```
#apply to whole text column
total['text'] = total['text'].apply(expand_contractions)
```

[25]:
```
def clean(tweet):

    #correct some acronyms while we are at it
    tweet = re.sub(r"tnwx", "Tennessee Weather", tweet)
    tweet = re.sub(r"azwx", "Arizona Weather", tweet)
    tweet = re.sub(r"alwx", "Alabama Weather", tweet)
    tweet = re.sub(r"wordpressdotcom", "wordpress", tweet)
    tweet = re.sub(r"gawx", "Georgia Weather", tweet)
    tweet = re.sub(r"scwx", "South Carolina Weather", tweet)
    tweet = re.sub(r"cawx", "California Weather", tweet)
    tweet = re.sub(r"usNWSgov", "United States National Weather Service", tweet)
    tweet = re.sub(r"MH370", "Malaysia Airlines Flight 370", tweet)
    tweet = re.sub(r"okwx", "Oklahoma City Weather", tweet)
    tweet = re.sub(r"arwx", "Arkansas Weather", tweet)
```

```python
    tweet = re.sub(r"lmao", "laughing my ass off", tweet)
    tweet = re.sub(r"amirite", "am I right", tweet)

    #and some typos/abbreviations
    tweet = re.sub(r"w/e", "whatever", tweet)
    tweet = re.sub(r"w/", "with", tweet)
    tweet = re.sub(r"USAgov", "USA government", tweet)
    tweet = re.sub(r"recentlu", "recently", tweet)
    tweet = re.sub(r"Ph0tos", "Photos", tweet)
    tweet = re.sub(r"amirite", "am I right", tweet)
    tweet = re.sub(r"exp0sed", "exposed", tweet)
    tweet = re.sub(r"<3", "love", tweet)
    tweet = re.sub(r"amageddon", "armageddon", tweet)
    tweet = re.sub(r"Trfc", "Traffic", tweet)
    tweet = re.sub(r"WindStorm", "Wind Storm", tweet)
    tweet = re.sub(r"16yr", "16 year", tweet)
    tweet = re.sub(r"TRAUMATISED", "traumatized", tweet)

    #hashtags and usernames
    tweet = re.sub(r"IranDeal", "Iran Deal", tweet)
    tweet = re.sub(r"ArianaGrande", "Ariana Grande", tweet)
    tweet = re.sub(r"camilacabello97", "camila cabello", tweet)
    tweet = re.sub(r"RondaRousey", "Ronda Rousey", tweet)
    tweet = re.sub(r"MTVHottest", "MTV Hottest", tweet)
    tweet = re.sub(r"TrapMusic", "Trap Music", tweet)
    tweet = re.sub(r"ProphetMuhammad", "Prophet Muhammad", tweet)
    tweet = re.sub(r"PantherAttack", "Panther Attack", tweet)
    tweet = re.sub(r"StrategicPatience", "Strategic Patience", tweet)
    tweet = re.sub(r"socialnews", "social news", tweet)
    tweet = re.sub(r"IDPs:", "Internally Displaced People :", tweet)
    tweet = re.sub(r"ArtistsUnited", "Artists United", tweet)
    tweet = re.sub(r"ClaytonBryant", "Clayton Bryant", tweet)
    tweet = re.sub(r"jimmyfallon", "jimmy fallon", tweet)
    tweet = re.sub(r"justinbieber", "justin bieber", tweet)
    tweet = re.sub(r"Time2015", "Time 2015", tweet)
    tweet = re.sub(r"djicemoon", "dj icemoon", tweet)
    tweet = re.sub(r"LivingSafely", "Living Safely", tweet)
    tweet = re.sub(r"FIFA16", "Fifa 2016", tweet)
    tweet = re.sub(r"thisiswhywecanthavenicethings", "this is why we cannot␣
↪have nice things", tweet)
    tweet = re.sub(r"bbcnews", "bbc news", tweet)
    tweet = re.sub(r"UndergroundRailraod", "Underground Railraod", tweet)
    tweet = re.sub(r"c4news", "c4 news", tweet)
    tweet = re.sub(r"MUDSLIDE", "mudslide", tweet)
    tweet = re.sub(r"NoSurrender", "No Surrender", tweet)
    tweet = re.sub(r"NotExplained", "Not Explained", tweet)
    tweet = re.sub(r"greatbritishbakeoff", "great british bake off", tweet)
    tweet = re.sub(r"LondonFire", "London Fire", tweet)
```

```
    tweet = re.sub(r"KOTAWeather", "KOTA Weather", tweet)
    tweet = re.sub(r"LuchaUnderground", "Lucha Underground", tweet)
    tweet = re.sub(r"KOIN6News", "KOIN 6 News", tweet)
    tweet = re.sub(r"LiveOnK2", "Live On K2", tweet)
    tweet = re.sub(r"9NewsGoldCoast", "9 News Gold Coast", tweet)
    tweet = re.sub(r"nikeplus", "nike plus", tweet)
    tweet = re.sub(r"david_cameron", "David Cameron", tweet)
    tweet = re.sub(r"peterjukes", "Peter Jukes", tweet)
    tweet = re.sub(r"MikeParrActor", "Michael Parr", tweet)
    tweet = re.sub(r"4PlayThursdays", "Foreplay Thursdays", tweet)
    tweet = re.sub(r"TGF2015", "Tontitown Grape Festival", tweet)
    tweet = re.sub(r"realmandyrain", "Mandy Rain", tweet)
    tweet = re.sub(r"GraysonDolan", "Grayson Dolan", tweet)
    tweet = re.sub(r"ApolloBrown", "Apollo Brown", tweet)
    tweet = re.sub(r"saddlebrooke", "Saddlebrooke", tweet)
    tweet = re.sub(r"TontitownGrape", "Tontitown Grape", tweet)
    tweet = re.sub(r"AbbsWinston", "Abbs Winston", tweet)
    tweet = re.sub(r"ShaunKing", "Shaun King", tweet)
    tweet = re.sub(r"MeekMill", "Meek Mill", tweet)
    tweet = re.sub(r"TornadoGiveaway", "Tornado Giveaway", tweet)
    tweet = re.sub(r"GRupdates", "GR updates", tweet)
    tweet = re.sub(r"SouthDowns", "South Downs", tweet)
    tweet = re.sub(r"braininjury", "brain injury", tweet)
    tweet = re.sub(r"auspol", "Australian politics", tweet)
    tweet = re.sub(r"PlannedParenthood", "Planned Parenthood", tweet)
    tweet = re.sub(r"calgaryweather", "Calgary Weather", tweet)
    tweet = re.sub(r"weallheartonedirection", "we all heart one direction",␣
↪tweet)
    tweet = re.sub(r"edsheeran", "Ed Sheeran", tweet)
    tweet = re.sub(r"TrueHeroes", "True Heroes", tweet)
    tweet = re.sub(r"ComplexMag", "Complex Magazine", tweet)
    tweet = re.sub(r"TheAdvocateMag", "The Advocate Magazine", tweet)
    tweet = re.sub(r"CityofCalgary", "City of Calgary", tweet)
    tweet = re.sub(r"EbolaOutbreak", "Ebola Outbreak", tweet)
    tweet = re.sub(r"SummerFate", "Summer Fate", tweet)
    tweet = re.sub(r"RAmag", "Royal Academy Magazine", tweet)
    tweet = re.sub(r"offers2go", "offers to go", tweet)
    tweet = re.sub(r"ModiMinistry", "Modi Ministry", tweet)
    tweet = re.sub(r"TAXIWAYS", "taxi ways", tweet)
    tweet = re.sub(r"Calum5SOS", "Calum Hood", tweet)
    tweet = re.sub(r"JamesMelville", "James Melville", tweet)
    tweet = re.sub(r"JamaicaObserver", "Jamaica Observer", tweet)
    tweet = re.sub(r"TweetLikeItsSeptember11th2001", "Tweet like it is␣
↪september 11th 2001", tweet)
    tweet = re.sub(r"cbplawyers", "cbp lawyers", tweet)
    tweet = re.sub(r"fewmoretweets", "few more tweets", tweet)
    tweet = re.sub(r"BlackLivesMatter", "Black Lives Matter", tweet)
```

```python
    tweet = re.sub(r"NASAHurricane", "NASA Hurricane", tweet)
    tweet = re.sub(r"onlinecommunities", "online communities", tweet)
    tweet = re.sub(r"humanconsumption", "human consumption", tweet)
    tweet = re.sub(r"Typhoon-Devastated", "Typhoon Devastated", tweet)
    tweet = re.sub(r"Meat-Loving", "Meat Loving", tweet)
    tweet = re.sub(r"facialabuse", "facial abuse", tweet)
    tweet = re.sub(r"LakeCounty", "Lake County", tweet)
    tweet = re.sub(r"BeingAuthor", "Being Author", tweet)
    tweet = re.sub(r"withheavenly", "with heavenly", tweet)
    tweet = re.sub(r"thankU", "thank you", tweet)
    tweet = re.sub(r"iTunesMusic", "iTunes Music", tweet)
    tweet = re.sub(r"OffensiveContent", "Offensive Content", tweet)
    tweet = re.sub(r"WorstSummerJob", "Worst Summer Job", tweet)
    tweet = re.sub(r"HarryBeCareful", "Harry Be Careful", tweet)
    tweet = re.sub(r"NASASolarSystem", "NASA Solar System", tweet)
    tweet = re.sub(r"animalrescue", "animal rescue", tweet)
    tweet = re.sub(r"KurtSchlichter", "Kurt Schlichter", tweet)
    tweet = re.sub(r"Throwingknifes", "Throwing knives", tweet)
    tweet = re.sub(r"GodsLove", "God's Love", tweet)
    tweet = re.sub(r"bookboost", "book boost", tweet)
    tweet = re.sub(r"ibooklove", "I book love", tweet)
    tweet = re.sub(r"NestleIndia", "Nestle India", tweet)
    tweet = re.sub(r"realDonaldTrump", "Donald Trump", tweet)
    tweet = re.sub(r"DavidVonderhaar", "David Vonderhaar", tweet)
    tweet = re.sub(r"CecilTheLion", "Cecil The Lion", tweet)
    tweet = re.sub(r"weathernetwork", "weather network", tweet)
    tweet = re.sub(r"GOPDebate", "GOP Debate", tweet)
    tweet = re.sub(r"RickPerry", "Rick Perry", tweet)
    tweet = re.sub(r"frontpage", "front page", tweet)
    tweet = re.sub(r"NewsInTweets", "News In Tweets", tweet)
    tweet = re.sub(r"ViralSpell", "Viral Spell", tweet)
    tweet = re.sub(r"til_now", "until now", tweet)
    tweet = re.sub(r"volcanoinRussia", "volcano in Russia", tweet)
    tweet = re.sub(r"ZippedNews", "Zipped News", tweet)
    tweet = re.sub(r"MicheleBachman", "Michele Bachman", tweet)
    tweet = re.sub(r"53inch", "53 inch", tweet)
    tweet = re.sub(r"KerrickTrial", "Kerrick Trial", tweet)
    tweet = re.sub(r"abstorm", "Alberta Storm", tweet)
    tweet = re.sub(r"Beyhive", "Beyonce hive", tweet)
    tweet = re.sub(r"RockyFire", "Rocky Fire", tweet)
    tweet = re.sub(r"Listen/Buy", "Listen / Buy", tweet)
    tweet = re.sub(r"ArtistsUnited", "Artists United", tweet)
    tweet = re.sub(r"ENGvAUS", "England vs Australia", tweet)
    tweet = re.sub(r"ScottWalker", "Scott Walker", tweet)

    return tweet
```

```
total['text'] = total['text'].apply(clean)
```

[26]:
```
tweets = [tweet for tweet in total['text']]

#split data to update changes
train = total[:len(train)]
test = total[len(train):]
```

The several cells above provide breaking down contractions, adding missing spaces, removing extra punctuation, and correcting abbreviations and typos. All of this data prep helps provide accurate prediction in the models.

[27]:
```
def generate_ngrams(text, n_gram=1):
    token = [token for token in text.lower().split(' ') if token != '' if token␣
 ↪not in wordcloud.STOPWORDS]
    ngrams = zip(*[token[i:] for i in range(n_gram)])
    return [' '.join(ngram) for ngram in ngrams]

###################################################
#### Unigrams
###################################################

disaster_unigrams = defaultdict(int)
for word in total[train['target'] == 1]['text']:
    for word in generate_ngrams(word, n_gram = 1):
        disaster_unigrams[word] += 1
disaster_unigrams = pd.DataFrame(sorted(disaster_unigrams.items(), key=lambda x:
 ↪ x[1])[::-1])

nondisaster_unigrams = defaultdict(int)
for word in total[train['target'] == 0]['text']:
    for word in generate_ngrams(word, n_gram = 1):
        nondisaster_unigrams[word] += 1
nondisaster_unigrams = pd.DataFrame(sorted(nondisaster_unigrams.items(),␣
 ↪key=lambda x: x[1])[::-1])

###################################################
#### Bigrams
###################################################

disaster_bigrams = defaultdict(int)
for word in total[train['target'] == 1]['text']:
    for word in generate_ngrams(word, n_gram = 2):
        disaster_bigrams[word] += 1
disaster_bigrams = pd.DataFrame(sorted(disaster_bigrams.items(), key=lambda x:␣
 ↪x[1])[::-1])
```

```
nondisaster_bigrams = defaultdict(int)
for word in total[train['target'] == 0]['text']:
    for word in generate_ngrams(word, n_gram = 2):
        nondisaster_bigrams[word] += 1
nondisaster_bigrams = pd.DataFrame(sorted(nondisaster_bigrams.items(),␣
↪key=lambda x: x[1])[::-1])

###################################################
#### Trigrams
###################################################

disaster_trigrams = defaultdict(int)
for word in total[train['target'] == 1]['text']:
    for word in generate_ngrams(word, n_gram = 3):
        disaster_trigrams[word] += 1
disaster_trigrams = pd.DataFrame(sorted(disaster_trigrams.items(), key=lambda x:
↪ x[1])[::-1])

nondisaster_trigrams = defaultdict(int)
for word in total[train['target'] == 0]['text']:
    for word in generate_ngrams(word, n_gram = 3):
        nondisaster_trigrams[word] += 1
nondisaster_trigrams = pd.DataFrame(sorted(nondisaster_trigrams.items(),␣
↪key=lambda x: x[1])[::-1])

###################################################
#### 4-grams
###################################################

disaster_4grams = defaultdict(int)
for word in total[train['target'] == 1]['text']:
    for word in generate_ngrams(word, n_gram = 4):
        disaster_4grams[word] += 1
disaster_4grams = pd.DataFrame(sorted(disaster_4grams.items(), key=lambda x:␣
↪x[1])[::-1])

nondisaster_4grams = defaultdict(int)
for word in total[train['target'] == 0]['text']:
    for word in generate_ngrams(word, n_gram = 4):
        nondisaster_4grams[word] += 1
nondisaster_4grams = pd.DataFrame(sorted(nondisaster_4grams.items(), key=lambda␣
↪x: x[1])[::-1])
```

Creating tokens and grams sometimes provides more insight in NLP. A combination of 3 or 4 words may be more meaningful than 1 word on its own.

[29]:
```
from keras.preprocessing.text import Tokenizer

#find way to tokenize punctuation
to_exclude = '*+-/()%\n[\\]{|}^_`~\t'
to_tokenize = '!"#$&?:;<=>@'
tokenizer = Tokenizer(filters = to_exclude)
text = 'Why are you so f%#@ing angry all the time?!'
text = re.sub(r'(['+to_tokenize+'])', r' \1 ', text)
tokenizer.fit_on_texts([text])

#view new text
print(tokenizer.word_index)
```

```
{'why': 1, 'are': 2, 'you': 3, 'so': 4, 'f': 5, '#': 6, '@': 7, 'ing': 8,
'angry': 9, 'all': 10, 'the': 11, 'time': 12, '?': 13, '!': 14}
```

[30]:
```
from keras.preprocessing.sequence import pad_sequences
from keras import Input

#define tokenizer options
tokenizer = Tokenizer()
#tokenizer = Tokenizer(oov_token = '<OOV>')            #if you wanted to␣
 ↪tokenized OOV words
#tokenizer = Tokenizer(filters = to_exclude)          #if you wanted to include␣
 ↪punctuation
tokenizer.fit_on_texts(tweets)
sequences = tokenizer.texts_to_sequences(tweets)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))

data = pad_sequences(sequences)
labels = train['target']
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', labels.shape)

nlp_train = data[:len(train)]
labels = labels
nlp_test = data[len(train):]

MAX_SEQUENCE_LENGTH = data.shape[1]
```

```
Found 29279 unique tokens.
Shape of data tensor: (10876, 40)
Shape of label tensor: (7613,)
```

[31]:
```
#get GloVe vector embeddings
embeddings_index = {}
with open('../input/glove-global-vectors-for-word-representation/glove.6B.200d.
↪txt','r') as f:
    for line in tqdm(f):
        values = line.split()
        word = values[0]
        coefs = np.asarray(values[1:], dtype='float32')
        embeddings_index[word] = coefs
f.close()

print('Found %s word vectors in the GloVe library' % len(embeddings_index))
```

```
      ␣
↪---------------------------------------------------------------------------

       FileNotFoundError                         Traceback (most recent call␣
↪last)

       <ipython-input-31-e1fbb6bc2a3c> in <module>
         1 #get GloVe vector embeddings
         2 embeddings_index = {}
   ----> 3 with open('../input/glove-global-vectors-for-word-representation/
↪glove.6B.200d.txt','r') as f:
         4     for line in tqdm(f):
         5         values = line.split()


       FileNotFoundError: [Errno 2] No such file or directory: '../input/
↪glove-global-vectors-for-word-representation/glove.6B.200d.txt'
```

[39]:
```
EMBEDDING_DIM = 200   #defined by size of GloVe word vector dimensions
```

[40]:
```
#initialize embedding matrix with zeros
embedding_matrix = np.zeros((len(word_index) + 1, EMBEDDING_DIM))

#add glove word encodings to our library
for word, i in tqdm(word_index.items()):
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:

        #words not found in embedding index will be all-zeros.
        embedding_matrix[i] = embedding_vector

print("Our embedded matrix is of dimension", embedding_matrix.shape)
```

```
100%|          | 29279/29279 [00:00<00:00, 780566.88it/s]
```

```
Our embedded matrix is of dimension (29280, 200)
```

1.4 Modeling

1.4.1 Model 1

[41]:
```
#import neural network basic
from keras.layers import Embedding, LSTM, Dense, SpatialDropout1D,␣
↪Bidirectional, Dropout, Concatenate, LeakyReLU, GRU
from keras import Input, Model, regularizers
from tensorflow.keras.optimizers import Adam
from keras.models import Sequential
from keras.callbacks import EarlyStopping

embedding = Embedding(len(word_index) + 1, EMBEDDING_DIM, weights =␣
↪[embedding_matrix],
                      input_length = MAX_SEQUENCE_LENGTH, trainable = False)

#we do not want embedding layer to train since it has been pretrained
```

[42]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler

def scale(df, scaler):
    return scaler.fit_transform(df.iloc[:, 2:])

#and scal
meta_train = scale(train, StandardScaler())
meta_test = scale(test, StandardScaler())
```

[43]:
```
#function to create lstm model
def create_lstm(spatial_dropout, dropout, recurrent_dropout, learning_rate,␣
↪bidirectional = False):
    #define activation
    activation = LeakyReLU(alpha = 0.01)

    #define inputs
    nlp_input = Input(shape = (MAX_SEQUENCE_LENGTH,), name = 'nlp_input')
    meta_input_train = Input(shape = (7, ), name = 'meta_train')
    emb = embedding(nlp_input)
    emb = SpatialDropout1D(dropout)(emb)

    #add LSTM layer
    if bidirectional:
```

```
        nlp_out = (Bidirectional(LSTM(100, dropout = dropout, recurrent_dropout␣
↪= recurrent_dropout,
                                 kernel_initializer = 'orthogonal')))(emb)
    else:
        nlp_out = (LSTM(100, dropout = dropout, recurrent_dropout =␣
↪recurrent_dropout,
                                 kernel_initializer = 'orthogonal'))(emb)

    #add meta data
    x = Concatenate()([nlp_out, meta_input_train])

    #add output layer
    x = Dropout(dropout)(x)
    preds = Dense(1, activation='sigmoid', kernel_regularizer = regularizers.
↪l2(1e-4))(x)

    #compile model
    model = Model(inputs=[nlp_input , meta_input_train], outputs = preds)
    optimizer = Adam(learning_rate = learning_rate)
    model.compile(loss = 'binary_crossentropy', optimizer = optimizer, metrics␣
↪= ['accuracy'])

    return model
```

[44]:
```
#define conveient training function to visualize learning curves
def plot_learning_curves(history):
    fig, ax = plt.subplots(1, 2, figsize = (20, 10))

    ax[0].plot(history.history['accuracy'])
    ax[0].plot(history.history['val_accuracy'])

    ax[1].plot(history.history['loss'])
    ax[1].plot(history.history['val_loss'])

    ax[0].legend(['train', 'validation'], loc = 'upper left')
    ax[1].legend(['train', 'validation'], loc = 'upper left')

    fig.suptitle("Model Accuracy", fontsize=14)

    ax[0].set_ylabel('Accuracy')
    ax[0].set_xlabel('Epoch')
    ax[1].set_ylabel('Loss')
    ax[1].set_xlabel('Epoch')

    return plt.show()
```

[45]:
```
#create our first model
lstm = create_lstm(spatial_dropout = .2, dropout = .2, recurrent_dropout = .2,
                   learning_rate = 3e-4, bidirectional = True)
lstm.summary()
```

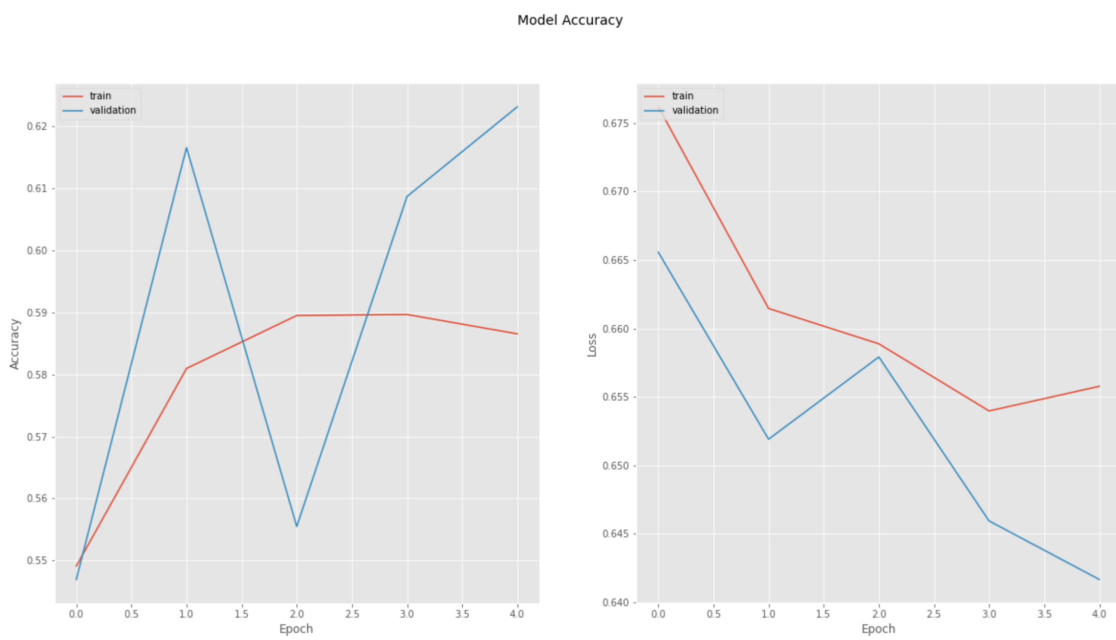
```
Model: "model"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape          Param #     Connected to
================================================================================
==================
 nlp_input (InputLayer)         [(None, 40)]          0           []

 embedding (Embedding)          (None, 40, 200)       5856000
['nlp_input[0][0]']

 spatial_dropout1d (SpatialDrop  (None, 40, 200)      0
['embedding[0][0]']
 out1D)

 bidirectional (Bidirectional)  (None, 200)           240800
['spatial_dropout1d[0][0]']

 meta_train (InputLayer)        [(None, 7)]           0           []

 concatenate (Concatenate)      (None, 207)           0
['bidirectional[0][0]',
'meta_train[0][0]']

 dropout (Dropout)              (None, 207)           0
['concatenate[0][0]']

 dense (Dense)                  (None, 1)             208
['dropout[0][0]']

================================================================================
==================
Total params: 6,097,008
Trainable params: 241,008
Non-trainable params: 5,856,000
________________________________________________________________________________
__________________
```

[46]:
```
#fit model
history1 = lstm.fit([nlp_train, meta_train], labels, validation_split = .2,
                     epochs = 5, batch_size = 21, verbose = 1)
```

```
Epoch 1/5
```

```
290/290 [==============================] - 40s 127ms/step - loss: 0.6762 -
accuracy: 0.5491 - val_loss: 0.6656 - val_accuracy: 0.5469
Epoch 2/5
290/290 [==============================] - 43s 148ms/step - loss: 0.6615 -
accuracy: 0.5810 - val_loss: 0.6519 - val_accuracy: 0.6165
Epoch 3/5
290/290 [==============================] - 38s 132ms/step - loss: 0.6589 -
accuracy: 0.5895 - val_loss: 0.6579 - val_accuracy: 0.5555
Epoch 4/5
290/290 [==============================] - 50s 172ms/step - loss: 0.6540 -
accuracy: 0.5897 - val_loss: 0.6459 - val_accuracy: 0.6087
Epoch 5/5
290/290 [==============================] - 57s 196ms/step - loss: 0.6558 -
accuracy: 0.5865 - val_loss: 0.6417 - val_accuracy: 0.6231
```

[47]:
```python
#view model 1 learning curves
plot_learning_curves(history1)
```



[48]:
```python
#define early stopping callback
callback = EarlyStopping(monitor = 'val_loss', patience = 4)

#include it in your models training
#history = model.fit(train, labels, validation_split = .2, epochs = 100,␣
↪callbacks = [callback])
```

Model 1 has accuracy: 0.6231. As seen from the graphs above, the more epochs used, the higher accuracy and less loss.

1.5 Model 2

[49]:
```
#function to create lstm model
def create_lstm_2(spatial_dropout, dropout, recurrent_dropout, learning_rate,␣
↪bidirectional = False):
    #define activation
    activation = LeakyReLU(alpha = 0.01)

    #define inputs
    nlp_input = Input(shape = (MAX_SEQUENCE_LENGTH,), name = 'nlp_input')
    meta_input_train = Input(shape = (7, ), name = 'meta_train')
    emb = embedding(nlp_input)
    emb = SpatialDropout1D(dropout)(emb)

    #add LSTM layer
    if bidirectional:
        nlp_out = (Bidirectional(LSTM(100, dropout = dropout, recurrent_dropout␣
↪= recurrent_dropout,
                                kernel_initializer = 'orthogonal')))(emb)
    else:
        nlp_out = (LSTM(100, dropout = dropout, recurrent_dropout =␣
↪recurrent_dropout,
                        kernel_initializer = 'orthogonal'))(emb)

    #add meta data
    x = Concatenate()([nlp_out, meta_input_train])

    #add second hidden layer
    x = Dropout(dropout)(x)
    x = (Dense(100, activation = activation, kernel_regularizer = regularizers.
↪l2(1e-4),
                kernel_initializer = 'he_normal'))(x)

    #add output layer
    x = Dropout(dropout)(x)
    preds = Dense(1, activation='sigmoid', kernel_regularizer = regularizers.
↪l2(1e-4))(x)

    #compile model
    model = Model(inputs=[nlp_input , meta_input_train], outputs = preds)
    optimizer = Adam(learning_rate = learning_rate)
    model.compile(loss = 'binary_crossentropy', optimizer = optimizer, metrics␣
↪= ['accuracy'])

    return model
```

[50]:
```
#define new model
lstm_2 = create_lstm_2(spatial_dropout = .4, dropout = .4, recurrent_dropout = .
↪4,
                        learning_rate = 3e-4, bidirectional = True)

lstm_2.summary()
```

```
Model: "model_1"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 nlp_input (InputLayer)         [(None, 40)]         0           []

 embedding (Embedding)          (None, 40, 200)      5856000
['nlp_input[0][0]']

 spatial_dropout1d_1 (SpatialDr  (None, 40, 200)     0
['embedding[1][0]']
 opout1D)

 bidirectional_1 (Bidirectional  (None, 200)         240800
['spatial_dropout1d_1[0][0]']
 )

 meta_train (InputLayer)        [(None, 7)]          0           []

 concatenate_1 (Concatenate)    (None, 207)          0
['bidirectional_1[0][0]',
'meta_train[0][0]']

 dropout_1 (Dropout)            (None, 207)          0
['concatenate_1[0][0]']

 dense_1 (Dense)                (None, 100)          20800
['dropout_1[0][0]']

 dropout_2 (Dropout)            (None, 100)          0
['dense_1[0][0]']

 dense_2 (Dense)                (None, 1)            101
['dropout_2[0][0]']

================================================================================
==================
Total params: 6,117,701
```

```
Trainable params: 261,701
Non-trainable params: 5,856,000
--------------------------------------------------------------------------------
------------------
```

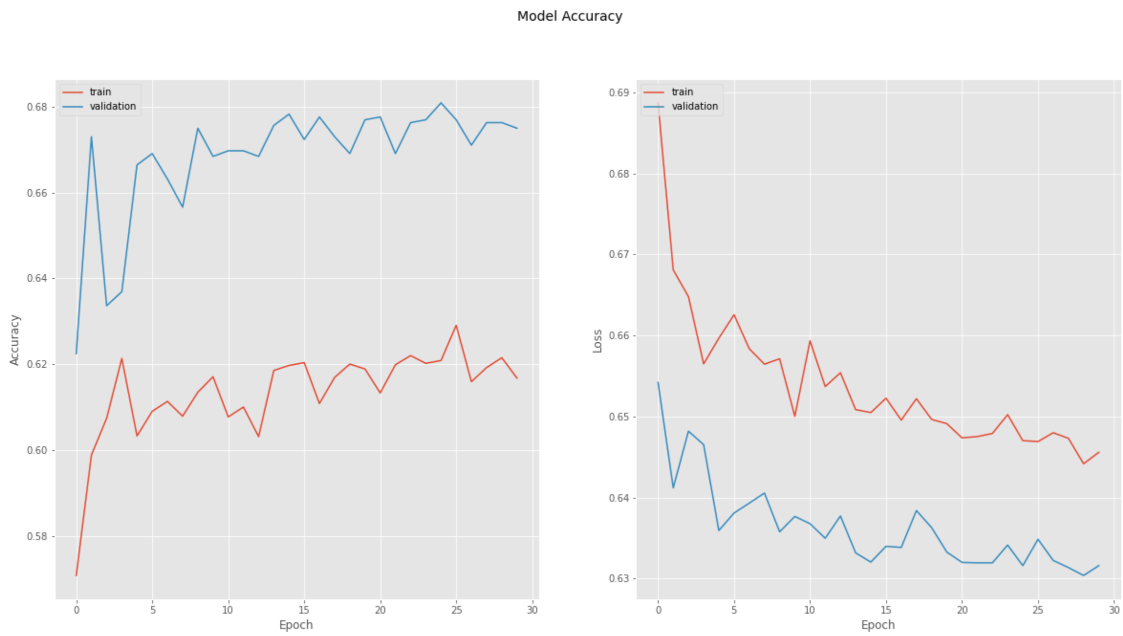
[51]:
```
#fit model
history2 = lstm_2.fit([nlp_train, meta_train], labels, validation_split = .2,
                       epochs = 30, batch_size = 21, verbose = 1)
```

```
Epoch 1/30
290/290 [==============================] - 58s 183ms/step - loss: 0.6887 -
accuracy: 0.5708 - val_loss: 0.6542 - val_accuracy: 0.6225
Epoch 2/30
290/290 [==============================] - 47s 163ms/step - loss: 0.6681 -
accuracy: 0.5989 - val_loss: 0.6412 - val_accuracy: 0.6730
Epoch 3/30
290/290 [==============================] - 49s 169ms/step - loss: 0.6648 -
accuracy: 0.6074 - val_loss: 0.6482 - val_accuracy: 0.6336
Epoch 4/30
290/290 [==============================] - 38s 130ms/step - loss: 0.6565 -
accuracy: 0.6213 - val_loss: 0.6465 - val_accuracy: 0.6369
Epoch 5/30
290/290 [==============================] - 34s 119ms/step - loss: 0.6597 -
accuracy: 0.6033 - val_loss: 0.6359 - val_accuracy: 0.6664
Epoch 6/30
290/290 [==============================] - 44s 150ms/step - loss: 0.6626 -
accuracy: 0.6090 - val_loss: 0.6381 - val_accuracy: 0.6691
Epoch 7/30
290/290 [==============================] - 61s 209ms/step - loss: 0.6584 -
accuracy: 0.6113 - val_loss: 0.6393 - val_accuracy: 0.6632
Epoch 8/30
290/290 [==============================] - 42s 144ms/step - loss: 0.6565 -
accuracy: 0.6079 - val_loss: 0.6406 - val_accuracy: 0.6566
Epoch 9/30
290/290 [==============================] - 38s 132ms/step - loss: 0.6571 -
accuracy: 0.6135 - val_loss: 0.6358 - val_accuracy: 0.6750
Epoch 10/30
290/290 [==============================] - 39s 134ms/step - loss: 0.6500 -
accuracy: 0.6171 - val_loss: 0.6377 - val_accuracy: 0.6684
Epoch 11/30
290/290 [==============================] - 37s 127ms/step - loss: 0.6593 -
accuracy: 0.6077 - val_loss: 0.6368 - val_accuracy: 0.6697
Epoch 12/30
290/290 [==============================] - 35s 119ms/step - loss: 0.6537 -
accuracy: 0.6100 - val_loss: 0.6350 - val_accuracy: 0.6697
Epoch 13/30
290/290 [==============================] - 35s 120ms/step - loss: 0.6554 -
accuracy: 0.6031 - val_loss: 0.6377 - val_accuracy: 0.6684
```

```
Epoch 14/30
290/290 [==============================] - 44s 153ms/step - loss: 0.6509 -
accuracy: 0.6186 - val_loss: 0.6332 - val_accuracy: 0.6756
Epoch 15/30
290/290 [==============================] - 39s 136ms/step - loss: 0.6505 -
accuracy: 0.6197 - val_loss: 0.6320 - val_accuracy: 0.6783
Epoch 16/30
290/290 [==============================] - 40s 139ms/step - loss: 0.6523 -
accuracy: 0.6204 - val_loss: 0.6340 - val_accuracy: 0.6724
Epoch 17/30
290/290 [==============================] - 40s 138ms/step - loss: 0.6496 -
accuracy: 0.6108 - val_loss: 0.6339 - val_accuracy: 0.6776
Epoch 18/30
290/290 [==============================] - 39s 135ms/step - loss: 0.6522 -
accuracy: 0.6169 - val_loss: 0.6384 - val_accuracy: 0.6730
Epoch 19/30
290/290 [==============================] - 39s 135ms/step - loss: 0.6496 -
accuracy: 0.6200 - val_loss: 0.6363 - val_accuracy: 0.6691
Epoch 20/30
290/290 [==============================] - 41s 142ms/step - loss: 0.6491 -
accuracy: 0.6189 - val_loss: 0.6333 - val_accuracy: 0.6770
Epoch 21/30
290/290 [==============================] - 39s 135ms/step - loss: 0.6474 -
accuracy: 0.6133 - val_loss: 0.6320 - val_accuracy: 0.6776
Epoch 22/30
290/290 [==============================] - 41s 140ms/step - loss: 0.6475 -
accuracy: 0.6199 - val_loss: 0.6319 - val_accuracy: 0.6691
Epoch 23/30
290/290 [==============================] - 38s 132ms/step - loss: 0.6479 -
accuracy: 0.6220 - val_loss: 0.6320 - val_accuracy: 0.6763
Epoch 24/30
290/290 [==============================] - 39s 133ms/step - loss: 0.6502 -
accuracy: 0.6202 - val_loss: 0.6341 - val_accuracy: 0.6770
Epoch 25/30
290/290 [==============================] - 40s 138ms/step - loss: 0.6470 -
accuracy: 0.6209 - val_loss: 0.6316 - val_accuracy: 0.6809
Epoch 26/30
290/290 [==============================] - 40s 138ms/step - loss: 0.6469 -
accuracy: 0.6291 - val_loss: 0.6349 - val_accuracy: 0.6770
Epoch 27/30
290/290 [==============================] - 43s 147ms/step - loss: 0.6480 -
accuracy: 0.6159 - val_loss: 0.6322 - val_accuracy: 0.6710
Epoch 28/30
290/290 [==============================] - 43s 150ms/step - loss: 0.6473 -
accuracy: 0.6192 - val_loss: 0.6314 - val_accuracy: 0.6763
Epoch 29/30
290/290 [==============================] - 44s 151ms/step - loss: 0.6442 -
accuracy: 0.6215 - val_loss: 0.6304 - val_accuracy: 0.6763
```

```
Epoch 30/30
290/290 [==============================] - 35s 121ms/step - loss: 0.6456 -
accuracy: 0.6167 - val_loss: 0.6316 - val_accuracy: 0.6750
```

[52]:
```python
plot_learning_curves(history2)
```



[53]:
```python
#create submission for complex lstm model
submission_lstm = pd.DataFrame()
submission_lstm['id'] = test_id
submission_lstm['prob'] = lstm_2.predict([nlp_test, meta_test])
submission_lstm['target'] = submission_lstm['prob'].apply(lambda x: 0 if x < .5␣
↪else 1)
submission_lstm.head(10)
```

[53]:

	id	prob	target
0	0	0.228178	0
1	2	0.428097	0
2	3	0.275927	0
3	9	0.482103	0
4	11	0.259960	0
5	12	0.327913	0
6	21	0.350315	0
7	22	0.153014	0
8	27	0.140478	0
9	29	0.226307	0

From this model 2, we see it has accuracy: 0.6750. The testing dataset has less accuracy and higher loss than the training set as suspected. Like model 1, it isn't great accuracy, but it is still slightly better than guessing.

1.6 Model 3

[54]:
```
#function to create dual lstm model
def create_dual_lstm(spatial_dropout, dropout, recurrent_dropout,␣
↪learning_rate, bidirectional = False):
    #define activation
    activation = LeakyReLU(alpha = 0.01)

    #define inputs
    nlp_input = Input(shape = (MAX_SEQUENCE_LENGTH,), name = 'nlp_input')
    meta_input_train = Input(shape = (7, ), name = 'meta_train')
    emb = embedding(nlp_input)
    emb = SpatialDropout1D(dropout)(emb)

    #add dual LSTM layers
    if bidirectional:
        nlp_out = (Bidirectional(LSTM(100, dropout = dropout, recurrent_dropout␣
↪= recurrent_dropout,
                                kernel_initializer = 'orthogonal',␣
↪return_sequences = True)))(emb)
        nlp_out = SpatialDropout1D(dropout)(nlp_out)
        nlp_out = (Bidirectional(LSTM(100, dropout = dropout, recurrent_dropout␣
↪= recurrent_dropout,
                                kernel_initializer = 'orthogonal')))(emb)
    else:
        nlp_out = (LSTM(100, dropout = dropout, recurrent_dropout =␣
↪recurrent_dropout,
                                kernel_initializer = 'orthogonal',␣
↪return_sequences = True))(emb)
        nlp_out = SpatialDropout1D(dropout)(nlp_out)
        nlp_out = (LSTM(100, dropout = dropout, recurrent_dropout =␣
↪recurrent_dropout,
                                kernel_initializer = 'orthogonal'))(emb)


    #add meta data
    x = Concatenate()([nlp_out, meta_input_train])

    #add second hidden layer
    #x = Dropout(dropout)(x)
    #x = (Dense(100, activation = activation, kernel_regularizer = regularizers.
↪l2(1e-4),
            #kernel_initializer = 'he_normal'))(x)
```

```
    #add output layer
    x = Dropout(dropout)(x)
    preds = Dense(1, activation='sigmoid', kernel_regularizer = regularizers.
↪l2(1e-4))(x)

    #compile model
    model = Model(inputs=[nlp_input , meta_input_train], outputs = preds)
    optimizer = Adam(learning_rate = learning_rate)
    model.compile(loss = 'binary_crossentropy', optimizer = optimizer, metrics␣
↪= ['accuracy'])

    return model
```

[55]:
```
#define new model
dual_lstm = create_dual_lstm(spatial_dropout = .4, dropout = .4,␣
↪recurrent_dropout = .4,
                             learning_rate = 3e-4, bidirectional = True)

dual_lstm.summary()
```

```
Model: "model_2"
________________________________________________________________________________
__________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================
==================
 nlp_input (InputLayer)         [(None, 40)]         0           []

 embedding (Embedding)          (None, 40, 200)      5856000
['nlp_input[0][0]']

 spatial_dropout1d_2 (SpatialDr  (None, 40, 200)     0
['embedding[2][0]']
 opout1D)

 bidirectional_3 (Bidirectional  (None, 200)         240800
['spatial_dropout1d_2[0][0]']
 )

 meta_train (InputLayer)        [(None, 7)]          0           []

 concatenate_2 (Concatenate)    (None, 207)          0
['bidirectional_3[0][0]',
'meta_train[0][0]']

 dropout_3 (Dropout)            (None, 207)          0
```

```
['concatenate_2[0][0]']

 dense_3 (Dense)                (None, 1)            208
['dropout_3[0][0]']

================================================================================
==================
Total params: 6,097,008
Trainable params: 241,008
Non-trainable params: 5,856,000
--------------------------------------------------------------------------------
------------------
```

[56]:
```
history3 = dual_lstm.fit([nlp_train, meta_train], labels, validation_split = .2,
         epochs = 25, batch_size = 21, verbose = 1) #callbacks = [callback]
```

```
Epoch 1/25
290/290 [==============================] - 44s 137ms/step - loss: 0.7013 -
accuracy: 0.5253 - val_loss: 0.6956 - val_accuracy: 0.5338
Epoch 2/25
290/290 [==============================] - 40s 137ms/step - loss: 0.6825 -
accuracy: 0.5617 - val_loss: 0.6743 - val_accuracy: 0.5463
Epoch 3/25
290/290 [==============================] - 42s 146ms/step - loss: 0.6706 -
accuracy: 0.5816 - val_loss: 0.6737 - val_accuracy: 0.5384
Epoch 4/25
290/290 [==============================] - 39s 134ms/step - loss: 0.6631 -
accuracy: 0.5872 - val_loss: 0.6584 - val_accuracy: 0.5561
Epoch 5/25
290/290 [==============================] - 42s 145ms/step - loss: 0.6641 -
accuracy: 0.5755 - val_loss: 0.6525 - val_accuracy: 0.5850
Epoch 6/25
290/290 [==============================] - 44s 151ms/step - loss: 0.6627 -
accuracy: 0.5788 - val_loss: 0.6492 - val_accuracy: 0.6041
Epoch 7/25
290/290 [==============================] - 43s 148ms/step - loss: 0.6584 -
accuracy: 0.5924 - val_loss: 0.6516 - val_accuracy: 0.5824
Epoch 8/25
290/290 [==============================] - 41s 141ms/step - loss: 0.6605 -
accuracy: 0.5856 - val_loss: 0.6475 - val_accuracy: 0.5995
Epoch 9/25
290/290 [==============================] - 44s 150ms/step - loss: 0.6603 -
accuracy: 0.5852 - val_loss: 0.6425 - val_accuracy: 0.6198
Epoch 10/25
290/290 [==============================] - 40s 139ms/step - loss: 0.6582 -
accuracy: 0.5885 - val_loss: 0.6453 - val_accuracy: 0.5988
Epoch 11/25
290/290 [==============================] - 38s 130ms/step - loss: 0.6596 -
```

```
accuracy: 0.5847 - val_loss: 0.6459 - val_accuracy: 0.5968
Epoch 12/25
290/290 [==============================] - 40s 139ms/step - loss: 0.6605 -
accuracy: 0.5798 - val_loss: 0.6420 - val_accuracy: 0.6179
Epoch 13/25
290/290 [==============================] - 48s 165ms/step - loss: 0.6599 -
accuracy: 0.5928 - val_loss: 0.6427 - val_accuracy: 0.6146
Epoch 14/25
290/290 [==============================] - 39s 135ms/step - loss: 0.6566 -
accuracy: 0.5895 - val_loss: 0.6409 - val_accuracy: 0.6218
Epoch 15/25
290/290 [==============================] - 39s 135ms/step - loss: 0.6587 -
accuracy: 0.5962 - val_loss: 0.6399 - val_accuracy: 0.6284
Epoch 16/25
290/290 [==============================] - 40s 136ms/step - loss: 0.6567 -
accuracy: 0.5938 - val_loss: 0.6427 - val_accuracy: 0.6139
Epoch 17/25
290/290 [==============================] - 37s 129ms/step - loss: 0.6567 -
accuracy: 0.5913 - val_loss: 0.6398 - val_accuracy: 0.6264
Epoch 18/25
290/290 [==============================] - 38s 132ms/step - loss: 0.6603 -
accuracy: 0.5862 - val_loss: 0.6417 - val_accuracy: 0.6179
Epoch 19/25
290/290 [==============================] - 40s 139ms/step - loss: 0.6599 -
accuracy: 0.5888 - val_loss: 0.6428 - val_accuracy: 0.6106
Epoch 20/25
290/290 [==============================] - 43s 149ms/step - loss: 0.6582 -
accuracy: 0.5862 - val_loss: 0.6420 - val_accuracy: 0.6192
Epoch 21/25
290/290 [==============================] - 43s 149ms/step - loss: 0.6571 -
accuracy: 0.5844 - val_loss: 0.6398 - val_accuracy: 0.6271
Epoch 22/25
290/290 [==============================] - 45s 155ms/step - loss: 0.6579 -
accuracy: 0.5878 - val_loss: 0.6417 - val_accuracy: 0.6159
Epoch 23/25
290/290 [==============================] - 38s 132ms/step - loss: 0.6571 -
accuracy: 0.5905 - val_loss: 0.6429 - val_accuracy: 0.6087
Epoch 24/25
290/290 [==============================] - 48s 164ms/step - loss: 0.6598 -
accuracy: 0.5852 - val_loss: 0.6427 - val_accuracy: 0.6080
Epoch 25/25
290/290 [==============================] - 48s 167ms/step - loss: 0.6565 -
accuracy: 0.5952 - val_loss: 0.6422 - val_accuracy: 0.6152
```

[57]:
```
#create submission for complex lstm model
submission_lstm2 = pd.DataFrame()
submission_lstm2['id'] = test_id
```

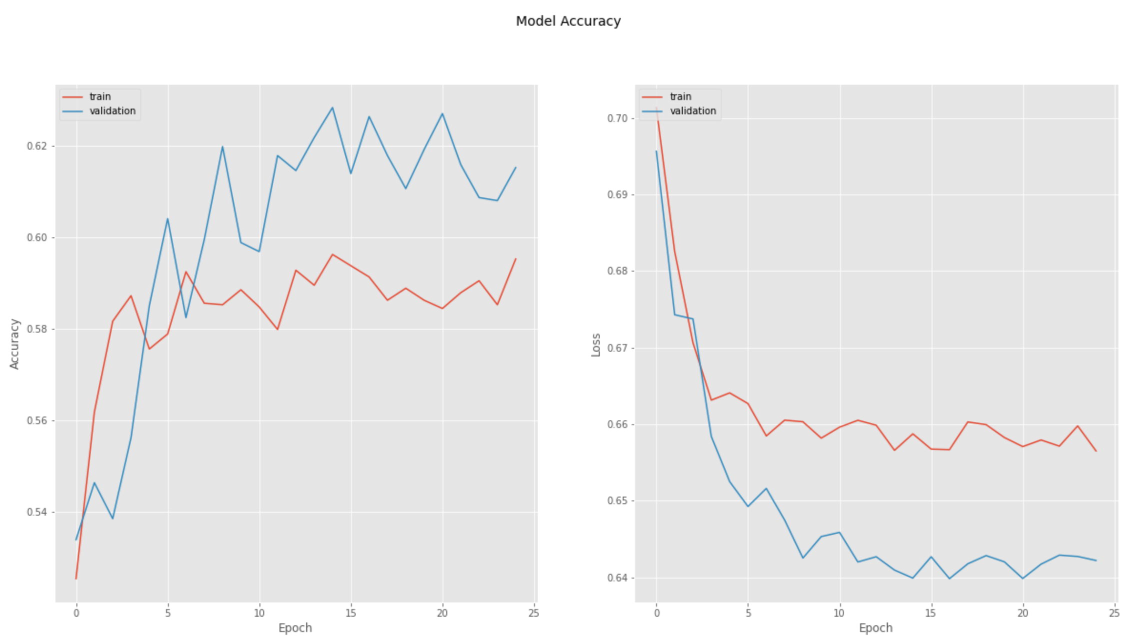
```python
submission_lstm2['prob'] = dual_lstm.predict([nlp_test, meta_test])
submission_lstm2['target'] = submission_lstm2['prob'].apply(lambda x: 0 if x < .
↪5 else 1)
submission_lstm2.head(10)
```

[57]:

	id	prob	target
0	0	0.279007	0
1	2	0.367270	0
2	3	0.300857	0
3	9	0.423140	0
4	11	0.304403	0
5	12	0.311978	0
6	21	0.334066	0
7	22	0.222073	0
8	27	0.217252	0
9	29	0.283380	0

[58]:

```python
plot_learning_curves(history3)
```



The last model 3 has accuracy: 0.6152. This model again has better accuracy and less loss with more epochs provided. Similar to the first 2 models, this one is slightly better than guessing in terms of accuracy, and there is definitely room for improvement.

1.7 Submissions

```
[61]: submission_lstm = submission_lstm[["id", "target"]]
submission_lstm2 = submission_lstm2[["id", "target"]]

submission_lstm.to_csv("submission_lstm.csv", index=False)
submission_lstm2.to_csv("submission_complex.csv", index=False)
```

1.8 Conclusion

In this assignment, Natural Language Processing with tokenized tweets were processed in Recurrent Neural Networks to predict if a tweet was actually about a crisis or if it was not crisis-related. After cleaning and tokenizing the tweets, 3 different RNNs were used: 2 lstm models and 1 dual lstm model. The reason RNN was effectively used was that context for these tweets was needed and by looking at 'previous' text data and evaluating a many-to one relationship as in many tokens in a tweet to 1 prediction. All 3 of the models produced performed slightly above average (where average would be the rate of guessing a binary outcome correctly: 50%). Surprisingly, when submitting the predictions to Kaggle, the first model performed better with a score of 0.65 whereas the more complex model performed more poorly with a score of 0.58. All in all, there is room for improvement in these models.