Feature Engineering on Twitter US Airline Dataset

Methods Used:

1. Bag of Words
2. Bag of N-Grams
3. TF-IDF
4. Cosine Document Similarity
5. word2vec trained on our dataset with tensorflow
6. word2vec trained on our dataset with gensim
7. word2vec trained by Google on News dataset

Import common packages

In [5]:
```
import pandas as pd
import numpy as np
import nltk
```

Load Cleaned Data Set

See previous code submission from project proposal for extensive data cleaning steps.

In [6]:
```
cleanDF = pd.read_csv("C:\git\CSML1010-Group_11-Final-Project\proposal\Tweets_cleaned.csv")
cleanDF.columns
```

Out[6]:
```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline', 'text',
       'text_cleaned', 'emojis_flag', 'emojis', 'emoticons_flag', 'emoticons',
       'text_cleaned_without_emojis_emoticons', 'hashtags',
       'text_cleaned_without_emojis_emoticons_hashtags', 'hashtags_flag',
       'text_cleaned_lower_case', 'text_cleaned_no_abbreviations',
       'text_list_no_stop_words', 'lemmas_list'],
      dtype='object')
```

Bag of Words Model

A bag of words model is a simpleway to represent text data as numeric vectors.

Each column is a word from our data set, each row is a single tweet and the value in each row is the number of occurrences of that word in the tweet.

In [3]:
```python
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))
#.values.astype('U') converts the column of words to a unicode string
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[3]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [4]:
```python
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)

#pd.options.display.max_columns = 100
#pd.set_option('display.max_rows', 100)

#pd.DataFrame(cv_matrix, columns=vocab).to_csv("bagofwords.csv")
```

Out[4]:

	00	000	000114	000lb	00a	00am	00p	00pm	01	01pm	...	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom	z
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14636	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14637	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14638	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14639	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

14640 rows × 8669 columns

Bag of N-Grams

A bag of n-grams model is similar to the bag of words model, but here we extend it to include a "bi-gram" of two words. This allows us to see the number of occurrences of two-word pairs in each of our tweets as a numeric vector.

In [5]:
```
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)

#pd.DataFrame(bv_matrix, columns=vocab).to_csv("bagofngrams.csv")
```

Out[5]:

	00 27	00 bag	00 check	00 don	00 flight	00 goodwill	00 happy	00 phone	00 pm	00 say	...	zone precious	zone space	zone thank	zoom sauce	zoom scroll	zuke non	zuric b
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14636	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14637	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14638	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14639	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

14640 rows × 62070 columns

TF-IDF Model

The TF-IDF model is a slightly more complicated way of vectorizing our text data. It combines two different ways of looking at our text. *term frequency* and *inverse document frequency*.

A more details explanation of how TF-IDF works is available elsewhere. For our purposes, it outputs scaled and normalized values which are more easily comparable and usable than bag of words-based models.

In [7]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)

#pd.DataFrame(np.round(tv_matrix, 2), columns=vocab).to_csv("tfidfmodel.csv")
```

Out[7]:

	00	000	000114	000lb	00a	00am	00p	00pm	01	01pm	...	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14635	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14636	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14637	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14638	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14639	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

14640 rows × 8669 columns

Cosine Document Similarity

This measure of document similarity uses a distance function to compare the vectors of each document (aka tweet) in our TF-IDF model. A value closer to 1 indicates a more similar document and a value closer to 0 indicates a dissimilar document. The values represent the cosine angle between the vectors for each tweet.

In [8]:
```
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(tv_matrix)
similarity_df = pd.DataFrame(similarity_matrix)
similarity_df

#similarity_df.to_csv("documentsimilarity.csv")
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	14630	14631	14632	14633	14634	14635	1
0	1.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.00
1	0.0	1.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.00
2	0.0	0.0	1.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.084757	0.0	0.000000	0.00
3	0.0	0.0	0.000000	1.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.00
4	0.0	0.0	0.000000	0.0	1.0	0.307344	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0.0	0.0	0.000000	0.0	0.0	0.028866	0.0	0.0	0.0	0.0	...	0.30427	0.188921	0.0	0.052285	0.0	1.000000	0.04
14636	0.0	0.0	0.000000	0.0	0.0	0.026716	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.098288	0.0	0.046479	1.00
14637	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.00
14638	0.0	0.0	0.000000	0.0	0.0	0.019337	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.035024	0.0	0.033641	0.03
14639	0.0	0.0	0.114465	0.0	0.0	0.116877	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.130844	0.0	0.070611	0.06

14640 rows × 14640 columns

Word2Vec Word Embedding Model

Training with CBOW (Continuous Bag of Words)

Now, we look at a more advanced model. Here, we implement a word embedding model. Word embedding is a way of representing individual words in such a way that words that are similar are represented similarly.

We do this using a word2vec model which represents words as multi-dimensional vectors (in this case 100 dimensions).

We build a model from scratch here using *keras* and *tensorflow* to build a *Continuous Bag of Words* (CBOW) model. A CBOW takes multiple words that surround a target word as input, and predicts the target word based on that input.

Build Vocabulary

```
In [9]: from keras.preprocessing import text
        from keras.utils import np_utils
        from keras.preprocessing import sequence

        tokenizer = text.Tokenizer()
        tokenizer.fit_on_texts(cleanDF['lemmas_list'].values.astype('U'))
        word2id = tokenizer.word_index

        word2id['PAD'] = 0
        id2word = {v:k for k, v in word2id.items()}
        wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in cleanDF['lemmas_list'].values
        .astype('U')]

        vocab_size = len(word2id)
        embed_size = 100
        window_size = 2

        print('Vocabulary Size:', vocab_size)
        print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Using TensorFlow backend.

Vocabulary Size: 8872
Vocabulary Sample: [('flight', 1), ('thank', 2), ('hour', 3), ('cancel', 4), ('service', 5), ('time',
6), ('delay', 7), ('customer', 8), ('help', 9), ('get', 10)]
```

Build (context_words, target_word) pair generator

```
In [10]: def generate_context_word_pairs(corpus, window_size, vocab_size):
             context_length = window_size*2
             for words in corpus:
                 sentence_length = len(words)
                 for index, word in enumerate(words):
                     context_words = []
                     label_word    = []
                     start = index - window_size
                     end = index + window_size + 1

                     context_words.append([words[i]
                                          for i in range(start, end)
                                          if 0 <= i < sentence_length
                                          and i != index])
                     label_word.append(word)

                     x = sequence.pad_sequences(context_words, maxlen=context_length)
                     y = np_utils.to_categorical(label_word, vocab_size)
                     yield (x, y)
```

In [11]:
```
i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argwhere(y[0])[
0][0]])

        if i == 10:
            break
        i += 1
```

```
Context (X): ['plus', 'add', 'experience', 'tacky'] -> Target (Y): commercial
Context (X): ['aggressive', 'blast', 'entertainment', 'guest'] -> Target (Y): obnoxious
Context (X): ['blast', 'obnoxious', 'guest', 'face'] -> Target (Y): entertainment
Context (X): ['obnoxious', 'entertainment', 'face', 'little'] -> Target (Y): guest
Context (X): ['entertainment', 'guest', 'little', 'recourse'] -> Target (Y): face
Context (X): ['seriously', 'pay', 'seat', 'playing'] -> Target (Y): flight
Context (X): ['pay', 'flight', 'playing', 'bad'] -> Target (Y): seat
Context (X): ['flight', 'seat', 'bad', 'thing'] -> Target (Y): playing
Context (X): ['seat', 'playing', 'thing', 'fly'] -> Target (Y): bad
Context (X): ['playing', 'bad', 'fly', 'va'] -> Target (Y): thing
Context (X): ['yes', 'nearly', 'fly', 'vx'] -> Target (Y): time
```

Build CBOW Deep Network Model

In [12]:
```
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

import tensorflow as tf
with tf.device('/gpu:0'):

    cbow = Sequential()
    cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
    cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
    cbow.add(Dense(vocab_size, activation='softmax'))

    cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')
    print(cbow.summary())
```

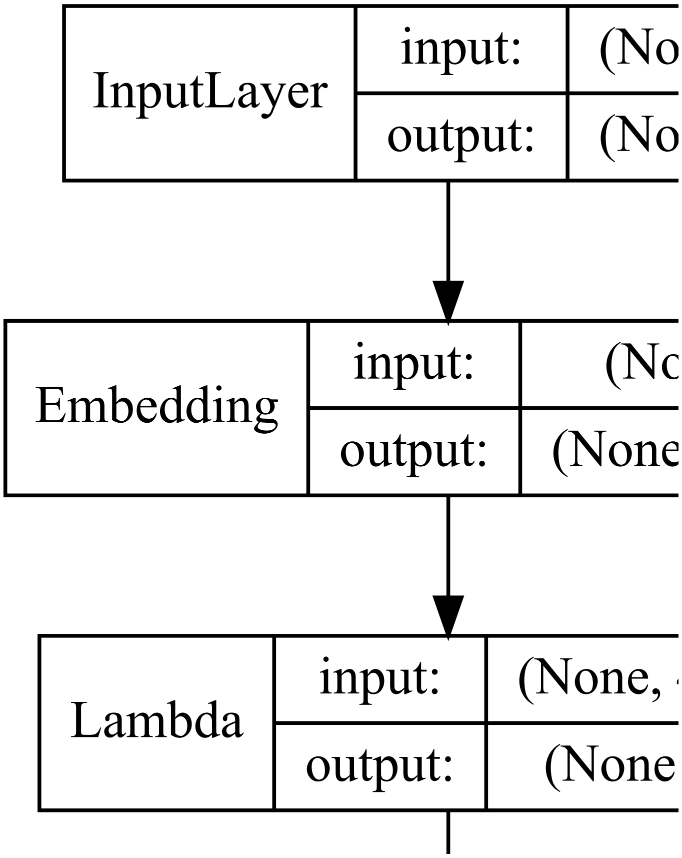
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            887200    
_________________________________________________________________
lambda_1 (Lambda)            (None, 100)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 8872)              896072    
=================================================================
Total params: 1,783,272
Trainable params: 1,783,272
Non-trainable params: 0
_________________________________________________________________
None
```

Visualize Model

In [13]:
```python
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Out[13]:



Train model for 5 epochs

In [ ]:
```
with tf.device('/gpu:0'):
    for epoch in range(1, 6):
        loss = 0.
        i = 0
        for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab
_size):
            i += 1
            loss += cbow.train_on_batch(x, y)
            if i % 100000 == 0:
                print('Processed {} (context, word) pairs'.format(i))

        print('Epoch:', epoch, '\tLoss:', loss)
        print()
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\tensorflow_core\python\framework\indexed_s
lices.py:433: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This m
ay consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Processed 100000 (context, word) pairs
Epoch: 1        Loss: 980699.2105463557

Processed 100000 (context, word) pairs
Epoch: 2        Loss: 1144282.866285345

Processed 100000 (context, word) pairs
Epoch: 3        Loss: 1337395.7755461363

Processed 100000 (context, word) pairs
Epoch: 4        Loss: 1453879.772978104
```

Get word embeddings

In [30]:
```
#load weights (COMMENT OUT BELOW if re-training)
weights = pd.read_csv("word2vecCBOWtrained.csv")
weights = cbow.get_weights()[0]
weights = weights[1:]

print(weights.shape)



#word2vecCBOWtrained = pd.DataFrame(weights, index=list(id2word.values())[1:]).head()

#pd.DataFrame(weights, index=list(id2word.values())[1:]).to_csv("word2vecCBOWtrained.csv")
```

```
(8871, 100)
```

Build a distance matrix to view the most similar words (contextually)¶

In [40]:
```
#load word embeddings (COMMENT OUT BELOW if re-training)
word2vecCBOWtrained = pd.read_csv("word2vecCBOWtrained.csv")

from sklearn.metrics.pairwise import euclidean_distances

# compute pairwise distance matrix
distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

# view contextually similar words
similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort
()[1:6]+1]
                   for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}

similar_words
```

```
(8871, 8871)
```

Out[40]:
```
{'flight': ['7hours', 'functionality', 'tvs', 'us2065', 'typical'],
 'airline': ['rocky', 'reduction', 'squeak', 'dedicated', 'prohibit'],
 'good': ['dnx58v', 'fog', 'fax', 'rr', '💚'],
 'bad': ['ua1266', 'win', 'lekvhg', 'crucial', 'underserved'],
 'time': ['practice', 'm', 'aa3230', 'wifi', 'loading'],
 'seat': ['taiwan', 'bogota', 'brendan', 'fudgin', 'yaayy'],
 'amazing': ['steve', "nat'l", '24hours', 'row7', 'eyewitness'],
 'experience': ['jean', 'any1', 'attitude', 'baby', 'hayes']}
```

Visualize word embeddings

In [44]:
```
word2vecCBOWtrained.iloc[:, 0]
```

Out[44]:
```
0             thank
1              hour
2            cancel
3           service
4              time
           ...     
8823         stuffy
8824    arbitrarily
8825    retribution
8826          aires
8827            PAD
Name: Unnamed: 0, Length: 8828, dtype: object
```

Word2Vec Word Embedding Model

Using gensim to train model

Now, to give us another feature to compare with, we will generate a similar word2vec model, using the Continuous Bag of Words model - this time, using gensim. We are again training this on our own data set, so we expect similar results. It will be a good double-check that both models are performing similarly.

In [46]:
```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in (cleanDF['lemmas_list'].values.astype('U'
))]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                  for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}
similar_words
```

Out[46]:
```
{'flight': ['afternoon', 'flightr', 'proactively', 'fvf9yw', 'characterize'],
 'airline': ['saddening', 'investment', 'business', 'absolute', 'bmi'],
 'good': ['comfortable', 'appleton', 'surprised', 'lmaooo', 'bad'],
 'bad': ['horrible', 'terrible', 'awful', 'difficulty', 'archaic'],
 'time': ['min', 'ua51', 'gla', 'yeah', 'estimate'],
 'seat': ['economy', 'seating', 'leg', 'row', 'voluntarily'],
 'amazing': ['fantastic', '👍', '😊', 'awesome', '⤴⤴'],
 'experience': ['ua5184', 'confrontational', 'wpg', 'respect', 'guru']}
```

In [47]:
```
from gensim.models import KeyedVectors

#export model
#w2v_model.wv.save_word2vec_format(fname='gensim_word2vec_trained.bin', binary=True)
```

Visualize word embeddings

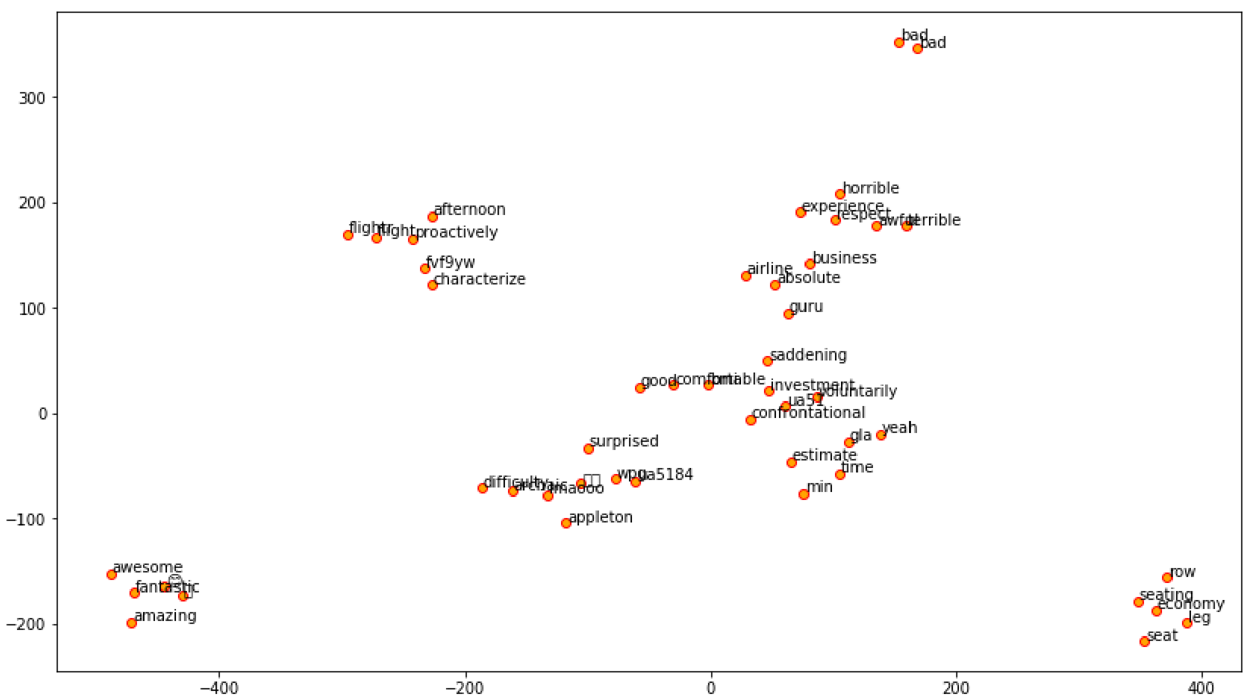
```
In [48]: import matplotlib.pyplot as plt
         from sklearn.manifold import TSNE

         words = sum([[k] + v for k, v in similar_words.items()], [])
         wvs = w2v_model.wv[words]

         tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
         np.set_printoptions(suppress=True)
         T = tsne.fit_transform(wvs)
         labels = words

         plt.figure(figsize=(14, 8))
         plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
         for label, x, y in zip(labels, T[:, 0], T[:, 1]):
             plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 55357 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 56397 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:176: Ru
ntimeWarning: Glyph 128077 missing from current font.
  font.load_char(ord(s), flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 56842 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 10548 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:180: Ru
ntimeWarning: Glyph 10548 missing from current font.
  font.set_text(s, 0, flags=flags)
```



Word2Vec Word Embedding Model

Using Pre-Trained Model from Google

Subet of Google's word2vec model for the 500,000 most frequent words

For our final model, instead of training our own word2vec model, we instead download Google's pre-trained word2vec model. Google has trained this model on articles from its news service and it includes 300 dimensions in almost 3.7 GB. This is not a manageable size, but we still want to load Google's model to compare to our custom-trained models.

To make this manageable, when loading the model we limit it to the first 500,000 rows. Google's model is organized with the most frequent words at the beginning of the dataset so this should be an effective reduction.

We save this reduced model to use in our benchmarking.

In [49]:
```
from gensim.models import KeyedVectors

w2v_model = KeyedVectors.load_word2vec_format(r'C:\Users\Patrick\gensim-data\word2vec-google-news-300
\word2vec-google-news-300\GoogleNews-vectors-negative300.bin', binary=True, limit=500000)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                  for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}
similar_words
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\ipykernel_launcher.py:7: DeprecationWarnin
g: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  import sys
```

Out[49]:
```
{'flight': ['flights', 'plane', 'Flight', 'airplane', 'takeoff'],
 'airline': ['airlines', 'Airlines', 'Airline', 'Airways', 'Lufthansa'],
 'good': ['great', 'bad', 'terrific', 'decent', 'nice'],
 'bad': ['good', 'terrible', 'horrible', 'Bad', 'lousy'],
 'time': ['day', 'moment', 'days', 'period', 'periods'],
 'seat': ['seats', 'Seat', 'Seats', 'seat_vacated', 'seated'],
 'amazing': ['incredible',
  'awesome',
  'unbelievable',
  'fantastic',
  'phenomenal'],
 'experience': ['experiences',
  'expertise',
  'expereince',
  'experince',
  'knowledge']}
```

In [50]:
```
from gensim.models import KeyedVectors

#export model
#w2v_model.wv.save_word2vec_format(fname='gensim_word2vec_google_subset.bin', binary=True)
```

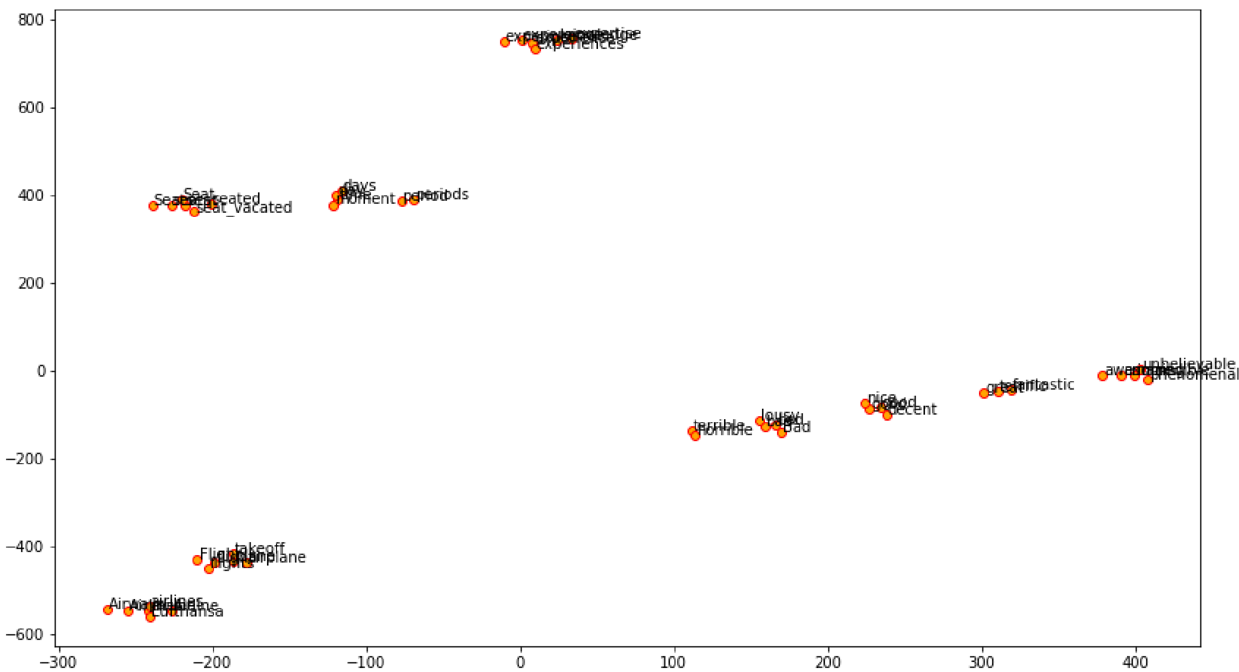
In [51]:
```python
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\ipykernel_launcher.py:5: DeprecationWarnin
g: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  """
```



In [ ]:

General Testing

In [52]:
```python
#check if using GPU
import tensorflow as tf
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")
```

```
Default GPU Device: /device:GPU:0
```

In [ ]:
```
#test execution on GPU
import tensorflow as tf
with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)

with tf.Session() as sess:
    print (sess.run(c))
```

Reference:

Much of the example code is credited to Dipanjan (DJ) Sarkar at:

[https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41 (https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41)](https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41)

and

[https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa (https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa)](https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa)

5. Feature Engineering, Feature Selection, and Baseline Benchmark

In [1]:
```
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import warnings
import numpy as np

warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)
```

In [2]:
```
#na_filter set to False as otherwise empty strings are interpreted as NaN
df_tweets_cleaned = pd.read_csv('..\data\Tweets_cleaned.csv', encoding='utf-8', na_filter= False)
```

Feature Engineering

Let's generate some features we could possibly use. Some features, such as `emojis_flag` , `emoticons_flag` , and `hashtags_flag` are already generated. Below are some of the features we are engineering/generating:

1. `emojis_num` denotes the number of emojis used in a tweet.
2. `emoitcons_num` denotes the number of emoticons used in a tweet.
3. `hashtag_num` denotes the number of hashtags used in a tweet.
4. `numbers_flag` denotes whether the tweet contains numbers or not (either in Arabic or English)
5. `numbers_num` denotes the number of times a tweet contains numbers We noticed that numbers were used in quite a few negative tweets, such as hours, time, dollars, flight numbers, etc. This is why we are generating a binary flag, as well as a numeric count of numbers used in a tweet.
6. `char_length_original` denotes the length of the the user's original tweet. This includes everything (@ mentions, RT retweets, hyperlinks, etc.)
7. `char_length_user` denotes the length of the user's cleaned tweet. The length will be based off the column `text_cleaned` We also noticed that negative tweets were, on average, longer than positive tweets in terms of character length.
8. `mentions_num` denotes the number of mentions a tweet has (@ mentions)
9. `retweet_flag` denotes if the user's tweet retweeted a tweet (normally the retweet is one of an airline, rarely another user). No need to create a count for retweets in a user's tweet because it's always 1.
10. `http_flag` denotes if the user's tweet has a HTTP link. No need to create a count for http links in a user's tweet because it's always 1 too.

The True/Flase `_flag` will need to be converted into binary flags instead (i.e. True/False into 1/0).

Any of the `_num` columns will likely need to be scaled to a scale from 0 to 1.

We will also need to vectorize the words in the tweets. To do so, there are several ways of doing so. We could use `word2vec` , `emoji2vec` , or a combination of both of them called `phrase2vec` .

Lastly, we will need to convert airline_sentiment into 0 or 1. In this situation, because we care about classifying negative sentiment tweets, and not really care about whether it's positive or neutral, we decided to group the positive and neutral tweets as `non-negative` . All `non-negative` tweets are class `0` , whereas all `negative` tweets are class `1` .

Generate columns `emojis_num` , `emoticons_num` , and `hashtag_num`

Generate basic features such as `emojis_num` , `emoticons_num` , `hashtag_num` from already developed columns.

In [3]:
```
#creates emojis_num column
def create_emojis_num(df):
    df['emojis_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'emojis_flag']:
            tweet_emojis = df.at[i, 'emojis']
            #strip brackets, quote, and spaces
            tweet_emojis_list = list(tweet_emojis.strip('[]').replace("\'", "").strip().split(","))
            emoji_counter = 0

            for emoji in tweet_emojis_list:
                emoji_counter = emoji_counter + 1

            df.at[i, 'emojis_num'] = emoji_counter
        else:
            df.at[i, 'emojis_num'] = 0

    return df

#creates emoticons_num column
def create_emoticons_num(df):
    df['emoticons_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'emoticons_flag']:
            tweet_emoticons = df.at[i, 'emoticons']
            #strip brackets, quote, and spaces
            tweet_emoticons_list = list(tweet_emoticons.strip('[]').replace("\'", "").strip().split(
","))
            emoticons_counter = 0

            for emoticon in tweet_emoticons_list:
                emoticons_counter = emoticons_counter + 1

            df.at[i, 'emoticons_num'] = emoticons_counter
        else:
            df.at[i, 'emoticons_num'] = 0

    return df

#creates hashtag_num column
def create_hashtags_num (df):
    df['hashtags_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'hashtags_flag']:
            tweet_hashtags = df.at[i, 'hashtags']
            #strip brackets, quote, and spaces
            tweet_hashtags_list = list(tweet_hashtags.strip('[]').replace("\'", "").strip().split(","
))
            hashtags_counter = 0

            for hashtag in tweet_hashtags_list:
                hashtags_counter = hashtags_counter + 1

            df.at[i, 'hashtags_num'] = hashtags_counter
        else:
            df.at[i, 'hashtags_num'] = 0

    return df
```

In [4]:
```
df_tweets_cleaned = create_emojis_num(df_tweets_cleaned)
df_tweets_cleaned = create_emoticons_num(df_tweets_cleaned)
df_tweets_cleaned = create_hashtags_num(df_tweets_cleaned)
```

In [5]:
```
#df_tweets_cleaned.loc[df_tweets_cleaned['hashtags_flag'] == True]
```

Generate columns `numbers_flag` , `numbers_num`

Generate a binary flag and a count of how many times numbers were used in a tweet. Numbers can either be numeric, or in English. English numbers are sometimes considered stop words by Spacy (e.g. "twelve" is a stop word in tweet `568911315026063361` , but "thirty" is not for some reason in tweet `568237684277141504`), and were removed in `lemmas_list` , so we generate the numbers features from column `text_cleaned_no_abbreviations` . We will use Spacy model to help us determine which token are like numbers, using `like_num` .

In [6]:
```
df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 568911315026063361]
```

Out[6]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline
2767	568911315026063361	negative	1.0	Late Flight	1.0	United

In [7]:
```
#load spacy model
import spacy

nlp = spacy.load('en_core_web_md')
```

In [8]:
```
#this function will create the columns numbers_flag and numbers_num
def create_numbers_columns(df):
    df['numbers_flag'] = False
    df['numbers_num'] = 0

    for i, row in df.iterrows():
        if i % 1000 == 0:
            print('at row number: ' + str(i))

        text = df.at[i, 'text_cleaned_no_abbreviations']
        #print(type(text))

        like_num_count = 0

        #tokenize text into list of tokens
        #print(text)

        token_list = nlp(text)

        #iterate through our tokens and count the number of nums
        for token in token_list:
            #print(token)
            if token.like_num:
                like_num_count = like_num_count + 1

        #at the end, we set our new columns
        if like_num_count != 0:
            df.at[i, 'numbers_flag'] = True
            df.at[i, 'numbers_num'] = like_num_count

    return df
```

In [9]:
```
#Sanity check

create_numbers_columns(df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 568911315026063361])
#create_numbers_columns(df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 570093964059156481])
```

Out[9]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline
2767	568911315026063361	negative	1.0	Late Flight	1.0	United

In [10]:
```
df_tweets_cleaned = create_numbers_columns(df_tweets_cleaned)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [11]:
```
#df_tweets_cleaned.loc[df_tweets_cleaned['numbers_flag'] == True]
```

Generate columns `char_length_original` , `char_length_user`

Generate columns with the number of characters in original tweet, and cleaned tweet from column `text_cleaned` .

In [12]:
```
#this function will create the columns numbers_flag and numbers_num
def create_char_length_columns(df):
    df['char_length_original'] = 0
    df['char_length_user'] = 0

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        cleaned_text = df.at[i, 'text_cleaned_no_abbreviations']

        df.at[i, 'char_length_original'] = len(text)
        df.at[i, 'char_length_user'] = len(cleaned_text)

    return df
```

In [13]:
```
df_tweets_cleaned = create_char_length_columns(df_tweets_cleaned)
```

Generate columns `mentions_num` , `retweet_flag` , and `http_flag`

Generate columns `mentions_num` : number of mentions in a tweet, `retweet_flag` : whether a tweet has a retweet, and `http_flag` : whether a tweet has a http link.

In [14]:
```
import re

#this function will create mentions_num column
def create_mentions_num(df):
    df['mentions_num'] = 0

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'\@[\w\d]*'

        regex_hits_list = re.findall(regex_to_find, text)
        df.at[i, 'mentions_num'] = len(regex_hits_list)

    return df

#this function will create retweet_flag column
def create_retweet_flag(df):
    df['retweet_flag'] = False

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'RT \@.*'

        regex_hits_list = re.findall(regex_to_find, text)
        if (len(regex_hits_list) != 0):
            df.at[i, 'retweet_flag'] = True

    return df

#this function will create http_flag column
def create_http_flag(df):
    df['http_flag'] = False

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'https*://[^\s]*'

        regex_hits_list = re.findall(regex_to_find, text)
        if (len(regex_hits_list) != 0):
            df.at[i, 'http_flag'] = True

    return df
```

In [15]:
```
df_tweets_cleaned = create_mentions_num(df_tweets_cleaned)
df_tweets_cleaned = create_retweet_flag(df_tweets_cleaned)
df_tweets_cleaned = create_http_flag(df_tweets_cleaned)
```

Scale numeric columns

The numeric columns will likely need to be scaled to a scale from 0 to 1. For columns `char_length_original` and `char_length_user` we will use normal MinMaxScaler because there aren't any big outliers, but for the other columns we will use RobustScaler as there are outliers.

In [16]:
```
df_tweets_cleaned[['emojis_num', 'emoticons_num', 'hashtags_num', 'numbers_num', 'char_length_origina
l', 'char_length_user', 'mentions_num']].describe()
```

Out[16]:

	emojis_num	emoticons_num	hashtags_num	numbers_num	char_length_original	char_length_user	mentions_num
count	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000
mean	0.066667	0.019262	0.238525	0.429372	103.822063	88.186066	1.132719
std	0.612111	0.140400	0.654195	0.741321	36.277339	36.834301	0.410359
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	1.000000
25%	0.000000	0.000000	0.000000	0.000000	77.000000	59.000000	1.000000
50%	0.000000	0.000000	0.000000	0.000000	114.000000	96.000000	1.000000
75%	0.000000	0.000000	0.000000	1.000000	136.000000	121.000000	1.000000
max	40.000000	3.000000	8.000000	7.000000	186.000000	177.000000	6.000000

In [17]:
```
from sklearn.preprocessing import MinMaxScaler, RobustScaler

minMaxScaler = MinMaxScaler()
df_tweets_cleaned['char_length_original_scaled'] = 0
df_tweets_cleaned['char_length_user_scaled'] = 0
df_tweets_cleaned[['char_length_original_scaled', 'char_length_user_scaled']] = \
    minMaxScaler.fit_transform(df_tweets_cleaned[['char_length_original', 'char_length_user']])

robustScaler = RobustScaler()
df_tweets_cleaned['emojis_num_scaled'] = 0
df_tweets_cleaned['emoticons_num_scaled'] = 0
df_tweets_cleaned['hashtags_num_scaled'] = 0
df_tweets_cleaned['numbers_num_scaled'] = 0
df_tweets_cleaned['mentions_num_scaled'] = 0
df_tweets_cleaned[['emojis_num_scaled', 'emoticons_num_scaled', 'hashtags_num_scaled', 'numbers_num_sc
aled', 'mentions_num_scaled']] = \
    minMaxScaler.fit_transform(df_tweets_cleaned[['emojis_num', 'emoticons_num', 'hashtags_num', 'numb
ers_num', 'mentions_num']])
```

Convert binary True/False columns to 1s/0s

We should convert binary True/False columns to 1s/0s.

In [18]:
```
df_tweets_cleaned['emojis_flag'] = df_tweets_cleaned['emojis_flag'].astype(int)
df_tweets_cleaned['emoticons_flag'] = df_tweets_cleaned['emoticons_flag'].astype(int)
df_tweets_cleaned['hashtags_flag'] = df_tweets_cleaned['hashtags_flag'].astype(int)
df_tweets_cleaned['numbers_flag'] = df_tweets_cleaned['numbers_flag'].astype(int)
df_tweets_cleaned['retweet_flag'] = df_tweets_cleaned['retweet_flag'].astype(int)
df_tweets_cleaned['http_flag'] = df_tweets_cleaned['http_flag'].astype(int)
```

Group the positive and neutral

As stated before, our goal is to predict negative sentiment tweets; we don't particularly care if the tweets are positive or neutral, to us they are the same thing: not negative. Therefore, we merge the positive and neutral classes to 0s, and rename negative label to 1s.

In [19]:
```
df_tweets_cleaned['binary_response_variable'] = False

df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'neutral', 'binary_response_variable'] =
False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'positive', 'binary_response_variable'] =
False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'negative', 'binary_response_variable'] =
True
```

In [20]:
```
df_tweets_cleaned.binary_response_variable
```

Out[20]:
```
0        False
1        False
2        False
3         True
4         True
         ...
14635    False
14636     True
14637    False
14638     True
14639    False
Name: binary_response_variable, Length: 14640, dtype: bool
```

In [21]:
```
df_tweets_cleaned.binary_response_variable = df_tweets_cleaned.binary_response_variable.astype(int)
```

In [22]:
```
df_tweets_cleaned
```

Out[22]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airlir
0	570306133677760513	neutral	1.0000			Virg Ameri
1	570301130888122368	positive	0.3486		0.0	Virg Ameri
2	570301083672813571	neutral	0.6837			Virg Ameri
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virg Ameri
4	570300817074462722	negative	1.0000	Can't Tell	1.0	Virg Ameri
...	...	...	...	...	...	
14635	569587686496825344	positive	0.3487		0.0	America
14636	569587371693355008	negative	1.0000	Customer Service Issue	1.0	America
14637	569587242672398336	neutral	1.0000			America
14638	569587188687634433	negative	1.0000	Customer Service Issue	0.6659	America
14639	569587140490866689	neutral	0.6771		0.0	America

14640 rows × 39 columns

Get rid of all other columns

We only need the `binary_response_variable`, `_flag` columns, `_scaled` columns, and `lemmas_list` column (this will be vectorized using our models).

In [23]:
```
df_tweets_cleaned.columns
```

Out[23]:
```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline', 'text',
       'text_cleaned', 'text_cleaned_time_removed', 'emojis_flag', 'emojis',
       'emoticons_flag', 'emoticons', 'text_cleaned_without_emojis_emoticons',
       'hashtags', 'text_cleaned_without_emojis_emoticons_hashtags',
       'hashtags_flag', 'text_cleaned_lower_case',
       'text_cleaned_no_abbreviations', 'text_list_no_stop_words',
       'lemmas_list', 'emojis_num', 'emoticons_num', 'hashtags_num',
       'numbers_flag', 'numbers_num', 'char_length_original',
       'char_length_user', 'mentions_num', 'retweet_flag', 'http_flag',
       'char_length_original_scaled', 'char_length_user_scaled',
       'emojis_num_scaled', 'emoticons_num_scaled', 'hashtags_num_scaled',
       'numbers_num_scaled', 'mentions_num_scaled',
       'binary_response_variable'],
      dtype='object')
```

In [24]:
```
df_tweets_cleaned = df_tweets_cleaned[[
    'binary_response_variable',
    'emojis_flag',
    'emoticons_flag',
    'hashtags_flag',
    'numbers_flag',
    'retweet_flag',
    'http_flag',
    'char_length_original_scaled',
    'char_length_user_scaled',
    'emojis_num_scaled',
    'emoticons_num_scaled',
    'hashtags_num_scaled',
    'numbers_num_scaled',
    'mentions_num_scaled',
    'lemmas_list'
]]
```

Baseline Benchmark with trained word2vec gensim model

We do a baseline benchmark with Patrick's trained word2vec gensim model, located at `..\milestone1\Patrick\gensim_word2vec_trained.bin` of this repo.

Split into train and test datasets

We need to split our dataframe into train and test datasets/dataframes. We don't need a validation set for now, but we will split into one later for our machine learning model tuning. For now, 70%/30% for train/test seems good enough. We will split based on column `binary_response_variable`.

In [25]:
```
from sklearn.model_selection import train_test_split

X = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns != 'binary_response_variable']
Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_variable']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1, stratify=Y)
```

Generate Tweet vectors

In [26]:
```
import gensim

custom_w2v_gensim_model_path = '..\milestone1\Patrick\gensim_word2vec_trained.bin'
w2v_gensim_model = gensim.models.KeyedVectors.load_word2vec_format(custom_w2v_gensim_model_path, binar
y=True)
```

In [27]:
```
#calculates a vector for a given Tweet
def calculate_average_tweet_vector(tweet, w2v_model, num_dimensions):
    tokens = tweet.split(' ')

    tweet_vector = np.zeros(num_dimensions, np.float32)
    actual_token_count = 0

    for token in  tokens:
        if token in w2v_model.wv.vocab:
            actual_token_count = actual_token_count + 1
            tweet_vector = np.add(tweet_vector, w2v_model[token])

    tweet_vector = np.divide(tweet_vector, actual_token_count)

    return tweet_vector
```

In [28]:
```
#Sanity check
#calculate_average_tweet_vector("arrangement reimburse rental", w2v_gensim_model, w2v_gensim_model.wv.
vector_size)
```

In [29]:
```
num_dimensions = w2v_gensim_model.wv.vector_size
X_train['tweet_vector'] = X_train.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text,
w2v_gensim_model, num_dimensions))
X_test['tweet_vector'] = X_test.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text, w2
v_gensim_model, num_dimensions))
```

In [30]:
```
df_train_tweet_vector = pd.DataFrame(list(X_train['tweet_vector']), index=X_train.index)
df_test_tweet_vector = pd.DataFrame(list(X_test['tweet_vector']), index=X_test.index)
```

In [31]:
```
#after creating new dataframes above, any zeroes are converted to null...we need to convert them back
 to zeroes...
df_train_tweet_vector = df_train_tweet_vector.fillna(0)
df_test_tweet_vector = df_test_tweet_vector.fillna(0)
```

In [32]:
```
X_train_combined = pd.concat([df_train_tweet_vector, X_train], axis=1)
X_test_combined = pd.concat([df_test_tweet_vector, X_test], axis=1)
```

Remove `tweet_vector` and `lemmas_list` column

In [33]:
```
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'tweet_vector']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'tweet_vector']
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'lemmas_list']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'lemmas_list']
```

In [34]:
```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(C=100)
logmodel.fit(X_train_combined, Y_train)
predictions = logmodel.predict(X_test_combined)
```

```
In [35]: from sklearn.metrics import classification_report
         print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.71      0.75      1639
           1       0.84      0.89      0.86      2753

   micro avg       0.82      0.82      0.82      4392
   macro avg       0.81      0.80      0.81      4392
weighted avg       0.82      0.82      0.82      4392
```

Baseline Benchmark with Google word2vec gensim model subset

We do a baseline benchmark with Google's word2vec gensim model that is subset with our word vectors as well, located at `..\milestone1\Patrick\gensim_word2vec_google_subset.bin` , which can be unzipped from `..\milestone1\Patrick\gensim_word2vec_google_subset.bin` of this repo.

Split into train and test datasets

We need to split our dataframe into train and test datasets/dataframes. We don't need a validation set for now, but we will split into one later for our machine learning model tuning. For now, 70%/30% for train/test seems good enough. We will split based on column `binary_response_variable` .

```
In [36]: from sklearn.model_selection import train_test_split

         X = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns != 'binary_response_variable']
         Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_variable']

         X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1, stratify=Y)
```

Generate Tweet vectors

```
In [37]: import gensim

         custom_w2v_gensim_model_path = '..\milestone1\Patrick\gensim_word2vec_google_subset.bin'
         w2v_gensim_model = gensim.models.KeyedVectors.load_word2vec_format(custom_w2v_gensim_model_path, binar
         y=True)
```

```
In [38]: #calculates a vector for a given Tweet
         def calculate_average_tweet_vector(tweet, w2v_model, num_dimensions):
             tokens = tweet.split(' ')

             tweet_vector = np.zeros(num_dimensions, np.float32)
             actual_token_count = 0

             for token in  tokens:
                 if token in w2v_model.wv.vocab:
                     actual_token_count = actual_token_count + 1
                     tweet_vector = np.add(tweet_vector, w2v_model[token])

             tweet_vector = np.divide(tweet_vector, actual_token_count)

             return tweet_vector
```

```
In [39]: #Sanity check
         #calculate_average_tweet_vector("arrangement reimburse rental", w2v_gensim_model, w2v_gensim_model.wv.
         vector_size)
```

In [40]:
```
num_dimensions = w2v_gensim_model.wv.vector_size
X_train['tweet_vector'] = X_train.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text,
w2v_gensim_model, num_dimensions))
X_test['tweet_vector'] = X_test.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text, w2
v_gensim_model, num_dimensions))
```

In [41]:
```
df_train_tweet_vector = pd.DataFrame(list(X_train['tweet_vector']), index=X_train.index)
df_test_tweet_vector = pd.DataFrame(list(X_test['tweet_vector']), index=X_test.index)
```

In [42]:
```
#after creating new dataframes above, any zeroes are converted to null...we need to convert them back
 to zeroes...
df_train_tweet_vector = df_train_tweet_vector.fillna(0)
df_test_tweet_vector = df_test_tweet_vector.fillna(0)
```

In [43]:
```
X_train_combined = pd.concat([df_train_tweet_vector, X_train], axis=1)
X_test_combined = pd.concat([df_test_tweet_vector, X_test], axis=1)
```

Remove tweet_vector and lemmas_list column

In [44]:
```
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'tweet_vector']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'tweet_vector']
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'lemmas_list']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'lemmas_list']
```

In [45]:
```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(C=100)
logmodel.fit(X_train_combined, Y_train)
predictions = logmodel.predict(X_test_combined)
```

In [46]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.70      0.74      1639
           1       0.83      0.89      0.86      2753

   micro avg       0.82      0.82      0.82      4392
   macro avg       0.81      0.80      0.80      4392
weighted avg       0.82      0.82      0.82      4392
```

In [47]:
```
predictions
```

Out[47]:
```
array([1, 1, 1, ..., 0, 1, 0])
```

In [ ]: