Sentiment Analysis on Amazon Product Reviews

March 21, 2020

Part 1. Data Exploration

[82]:
```
import pandas as pd
import numpy as np

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

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import BernoulliNB, MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn import metrics
from sklearn.metrics import roc_auc_score, accuracy_score
from sklearn.pipeline import Pipeline

from bs4 import BeautifulSoup
import re
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk import sent_tokenize, word_tokenize, pos_tag

import logging
from gensim.models import word2vec
from gensim.models import Word2Vec
from gensim.models.keyedvectors import KeyedVectors

from keras.preprocessing import sequence
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, Lambda
from keras.layers.embeddings import Embedding
```

```python
from keras.layers.recurrent import LSTM, SimpleRNN, GRU
from keras.preprocessing.text import Tokenizer
from collections import defaultdict
from keras.layers.convolutional import Convolution1D
from keras import backend as K
from keras.layers.embeddings import Embedding
#import gensim
```

[56]:
```python
# Load csv file
df = pd.read_csv('/home/sakil/Desktop/Online_Product_Analytics/
↪Amazon_Unlocked_Mobile.csv')
df.head()
```

[56]:
```
                                        Product Name Brand Name   Price  \
0  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...    Samsung  199.99
1  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...    Samsung  199.99
2  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...    Samsung  199.99
3  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...    Samsung  199.99
4  "CLEAR CLEAN ESN" Sprint EPIC 4G Galaxy SPH-D7...    Samsung  199.99

   Rating                                            Reviews  Review Votes
0       5  I feel so LUCKY to have found this used (phone...           1.0
1       4  nice phone, nice up grade from my pantach revu...           0.0
2       5                                       Very pleased           0.0
3       4  It works good but it goes slow sometimes but i...           0.0
4       4  Great phone to replace my lost phone. The only...           0.0
```

Data Exploration

[57]:
```python
print("Summary statistics of numerical features : \n", df.describe())

print("\nTotal number of reviews: ",len(df))
print("\nTotal number of brands: ", len(list(set(df['Brand Name']))))
print("\nTotal number of unique products: ", len(list(set(df['Product Name']))))

print("\nPercentage of reviews with neutral sentiment : {:.2f}%"\
      .format(df[df['Rating']==3]["Reviews"].count()/len(df)*100))
print("\nPercentage of reviews with positive sentiment : {:.2f}%"\
      .format(df[df['Rating']>3]["Reviews"].count()/len(df)*100))
print("\nPercentage of reviews with negative sentiment : {:.2f}%"\
      .format(df[df['Rating']<3]["Reviews"].count()/len(df)*100))
```

```
Summary statistics of numerical features :
                Price         Rating   Review Votes
count  407907.000000  413840.000000  401544.000000
mean      226.867155       3.819578       1.507237
std       273.006259       1.548216       9.163853
min         1.730000       1.000000       0.000000
25%        79.990000       3.000000       0.000000
50%       144.710000       5.000000       0.000000
```

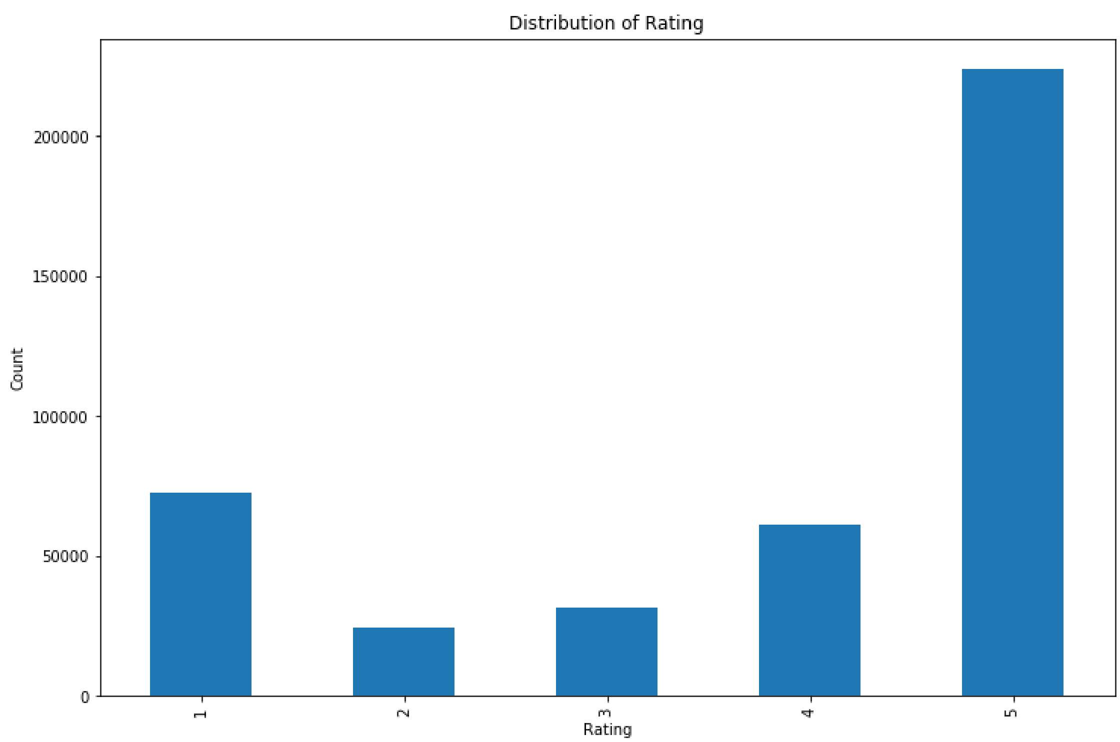
```
75%        269.990000       5.000000       1.000000
max       2598.000000       5.000000     645.000000

Total number of reviews:  413840

Total number of brands:  385

Total number of unique products:  4410

Percentage of reviews with neutral sentiment : 7.68%

Percentage of reviews with positive sentiment : 68.86%

Percentage of reviews with negative sentiment : 23.45%
```

Data Visualization

[59]:

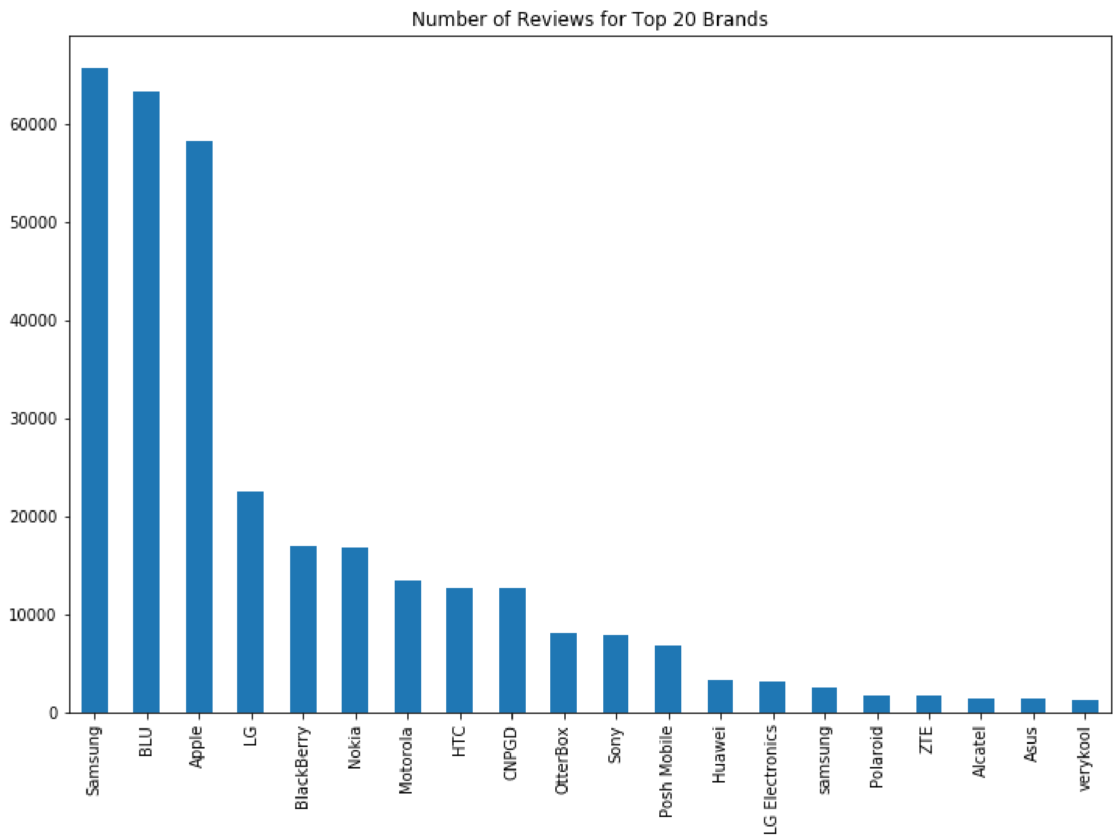
```python
# Plot distribution of rating
plt.figure(figsize=(12,8))
# sns.countplot(df['Rating'])
df['Rating'].value_counts().sort_index().plot(kind='bar')
plt.title('Distribution of Rating')
plt.xlabel('Rating')
plt.ylabel('Count')
```

[59]: Text(0, 0.5, 'Count')



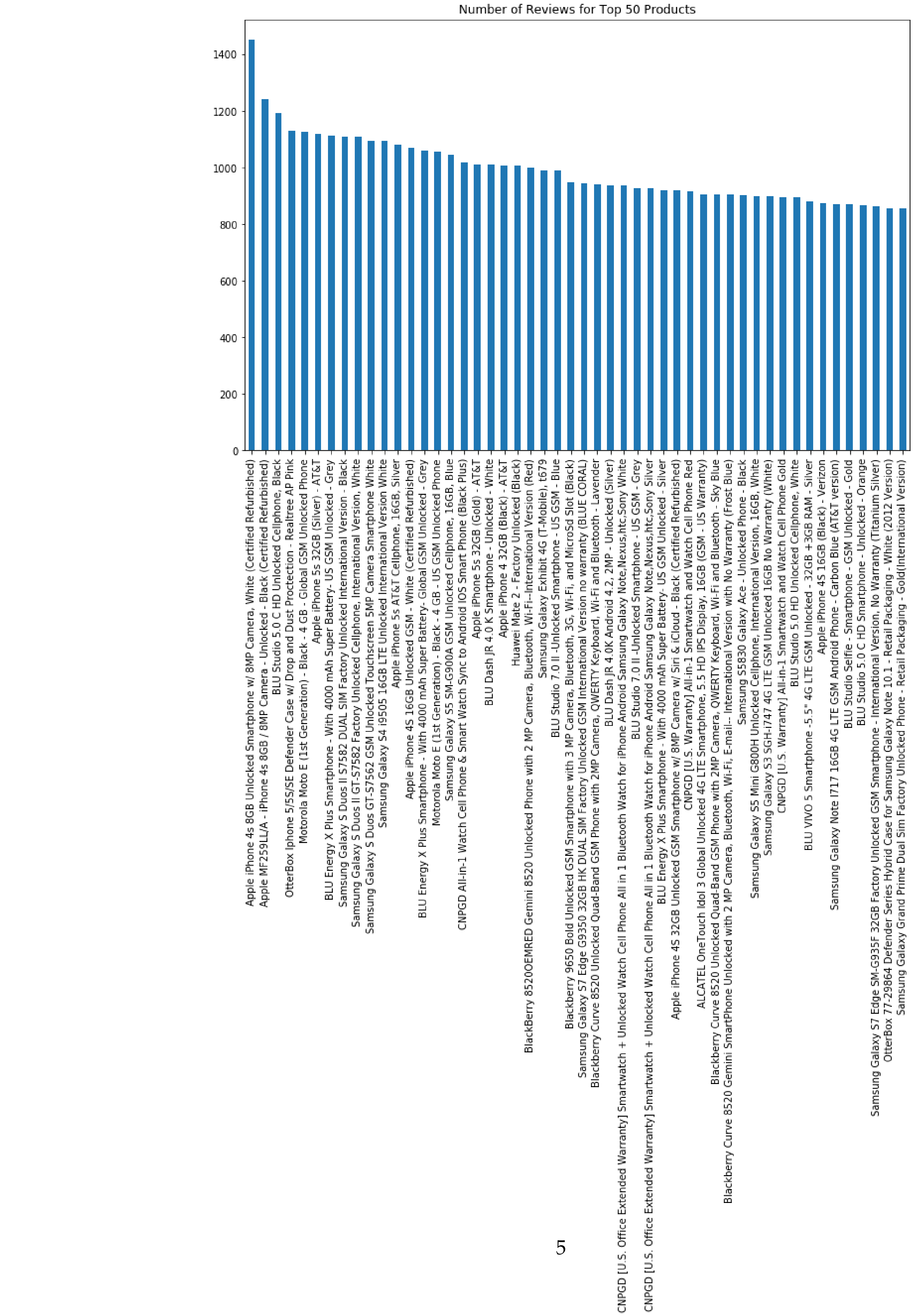
[60]:
```
# Plot number of reviews for top 20 brands
brands = df["Brand Name"].value_counts()
# brands.count()
plt.figure(figsize=(12,8))
brands[:20].plot(kind='bar')
plt.title("Number of Reviews for Top 20 Brands")
```

[60]: Text(0.5, 1.0, 'Number of Reviews for Top 20 Brands')



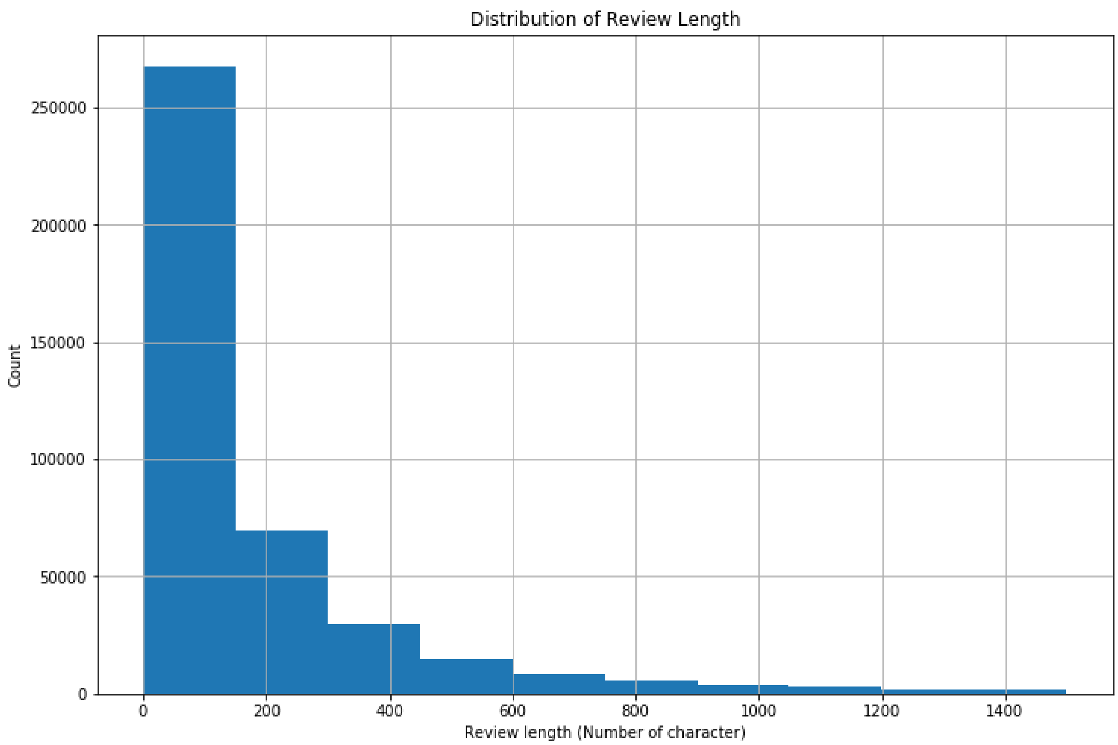
[61]:
```
# Plot number of reviews for top 50 products
products = df["Product Name"].value_counts()
plt.figure(figsize=(12,8))
products[:50].plot(kind='bar')
plt.title("Number of Reviews for Top 50 Products")
```

[61]: Text(0.5, 1.0, 'Number of Reviews for Top 50 Products')


Number of Reviews for Top 50 Products
0
200
400
600
800
1000
1200
1400

[62]:
```
# Plot distribution of review length
review_length = df["Reviews"].dropna().map(lambda x: len(x))
plt.figure(figsize=(12,8))
review_length.loc[review_length < 1500].hist()
plt.title("Distribution of Review Length")
plt.xlabel('Review length (Number of character)')
plt.ylabel('Count')
```

[62]: Text(0, 0.5, 'Count')



Part 2. Data Preparation For illustrative purpose, I use only 10% of the data in this project. To simply the problem, I only consider reviews with positive sentiment (rating = 4, 5) and negative sentiment (rating = 1, 2), and drop reviews with neutral sentiment (rating 3).

Prepare Data

[63]:
```
df = df.sample(frac=0.1, random_state=0) #uncomment to use full set of data

# Drop missing values
df.dropna(inplace=True)

# Remove any 'neutral' ratings equal to 3
df = df[df['Rating'] != 3]
```

```
# Encode 4s and 5s as 1 (positive sentiment) and 1s and 2s as 0 (negative␣
↪sentiment)
df['Sentiment'] = np.where(df['Rating'] > 3, 1, 0)
df.head()
```

[63]:
```
                                              Product Name Brand Name    Price  \
134801  BLU Studio 5.0 C HD - Unlocked Cell Phones - R...        BLU  2000.00
123493                         Blu LIFE 8 Unlocked (Pink)        BLU   199.98
335592  Samsung Galaxy S Duos II S7582 DUAL SIM Factor...    Samsung   299.99
246353  Motorola Droid 2 A955 Verizon Phone 5MP Cam, W...   Motorola    82.00
273324  Nokia Lumia 920 32GB Unlocked GSM 4G LTE Windo...      Nokia   149.35

        Rating                                            Reviews  \
134801       5  For the price I paid for this devices, its fan...
123493       5  love love love it...good buy...recommend to a...
335592       4                                               Good
246353       1  Not good. Returned first phone and they sent m...
273324       4  Met expectations! I'm very satisfied!Even arri...

        Review Votes  Sentiment
134801           0.0          1
123493           0.0          1
335592           0.0          1
246353           0.0          0
273324           1.0          1
```

Train Test Split

[64]:
```
# Split data into training set and validation
X_train, X_test, y_train, y_test = train_test_split(df['Reviews'],␣
↪df['Sentiment'], \
                                                    test_size=0.1,␣
↪random_state=0)

print('Load %d training examples and %d validation examples. \n' %(X_train.
↪shape[0],X_test.shape[0]))
print('Show a review in the training set : \n', X_train.iloc[10])
```

```
Load 27799 training examples and 3089 validation examples.

Show a review in the training set :
 good product and fast shipping. thank you.
```

1 Part 3. Bag of Words

The goal of this project is to classify the reviews into positive and negative sentiment. There are two main steps involved. First, we need to find a word embedding to convert a text into a numerical representation. Second, we fit the numerical representations of text to machine learning algorithms or deep learning architectures. One common approach of word embedding is frequency based embedding such as Bag of Words (BoW) model. BoW model learns a vocubulary list from a given corpus and represents each document based on some counting methods of words. In this part, we will explore the model performance of using BoW with supervised learning algorithms. Here's the workflow in this part. Step 1 : Preprocess raw reviews to cleaned reviews Step 2 : Create BoW using CountVectorizer / Tfidfvectorizer in sklearn Step 3 : Transform review text to numerical representations (feature vectors) Step 4 : Fit feature vectors to supervised learning algorithm (eg. Naive Bayes, Logistic regression, etc.) Step 5 : Improve the model performance by GridSearch

2 Text Preprocessing

The following text preprocessing are implemented to convert raw reviews to cleaned review, so that it will be easier for us to do feature extraction in the next step: remove html tags using BeautifulSoup

remove non-character such as digits and symbols

convert to lower case

remove stop words such as "the" and "and" if needed

convert to root words by stemming if needed

[65]:
```
def cleanText(raw_text, remove_stopwords=False, stemming=False,␣
 ↪split_text=False, \
             ):
    '''
    Convert a raw review to a cleaned review
    '''
    text = BeautifulSoup(raw_text, 'lxml').get_text()  #remove html
    letters_only = re.sub("[^a-zA-Z]", " ", text)  # remove non-character
    words = letters_only.lower().split() # convert to lower case

    if remove_stopwords: # remove stopword
        stops = set(stopwords.words("english"))
        words = [w for w in words if not w in stops]

    if stemming==True: # stemming
#         stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

    if split_text==True:  # split text
        return (words)

    return( " ".join(words))
```

[66]:
```
# Preprocess text data in training set and validation set
X_train_cleaned = []
X_test_cleaned = []
```

```
for d in X_train:
    X_train_cleaned.append(cleanText(d))
print('Show a cleaned review in the training set : \n',  X_train_cleaned[10])

for d in X_test:
    X_test_cleaned.append(cleanText(d))
```

```
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "http://www.amazon.com/gp/product/B013YDFH3Y?redirect=true&ref_=cm_
cr_ryp_prd_ttl_sol_0" looks like a URL. Beautiful Soup is not an HTTP client.
You should probably use an HTTP client like requests to get the document behind
the URL, and feed that document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "https://www.amazon.com/dp/B00K15KRV6/ref=cm_cr_ryp_prd_ttl_sol_22"
looks like a URL. Beautiful Soup is not an HTTP client. You should probably use
an HTTP client like requests to get the document behind the URL, and feed that
document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "http://www.amazon.com/gp/product/B0193D539M?redirect=true&ref_=cm_
cr_ryp_prd_ttl_sol_0" looks like a URL. Beautiful Soup is not an HTTP client.
You should probably use an HTTP client like requests to get the document behind
the URL, and feed that document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup

Show a cleaned review in the training set :
 good product and fast shipping thank you
```

3 CountVectorizer with Mulinomial Naive Bayes (Benchmark Model)

Now we have cleaned reviews, the next step is to convert the reviews into numerical representations for machine learning algorithm.

In sklearn library, we can use CountVectorizer which implements both tokenization and occurrence counting in a single class. The output is a sparse matrix representation of a document.

[67]:
```
# Fit and transform the training data to a document-term matrix using␣
 ↪CountVectorizer
countVect = CountVectorizer()
X_train_countVect = countVect.fit_transform(X_train_cleaned)
print("Number of features : %d \n" %len(countVect.get_feature_names())) #6378
print("Show some feature names : \n", countVect.get_feature_names()[::1000])


# Train MultinomialNB classifier
mnb = MultinomialNB()
mnb.fit(X_train_countVect, y_train)
```

```
Number of features : 19607

Show some feature names :
 ['aa', 'areable', 'boot', 'clean', 'crushing', 'distortions', 'excatly',
'frills', 'heart', 'inverter', 'lolit', 'movie', 'over', 'predictable',
'reconnecting', 'scaling', 'soldto', 'tapped', 'ubuntu', 'wedges']
```

[67]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

[68]:
```python
def modelEvaluation(predictions):
    '''
    Print model evaluation to predicted result
    '''
    print ("\nAccuracy on validation set: {:.4f}".format(accuracy_score(y_test,␣
 ↪predictions)))
    print("\nAUC score : {:.4f}".format(roc_auc_score(y_test, predictions)))
    print("\nClassification report : \n", metrics.classification_report(y_test,␣
 ↪predictions))
    print("\nConfusion Matrix : \n", metrics.confusion_matrix(y_test,␣
 ↪predictions))
```

[69]:
```python
# Evaluate the model on validaton set
predictions = mnb.predict(countVect.transform(X_test_cleaned))
modelEvaluation(predictions)
```

```
Accuracy on validation set: 0.9184

AUC score : 0.8790

Classification report :
               precision    recall  f1-score   support

           0       0.87      0.80      0.83       778
           1       0.93      0.96      0.95      2311

    accuracy                           0.92      3089
   macro avg       0.90      0.88      0.89      3089
weighted avg       0.92      0.92      0.92      3089


Confusion Matrix :
 [[ 622  156]
 [  96 2215]]
```

4 TfidfVectorizer with Logistic Regression

Some words might frequently appear but have little meaningful information about the sentiment of a particular review. Instead of using occurance counting, we can use tf-idf transform to scale down the impact of frequently appeared words in a given corpus.

In sklearn library, we can use TfidfVectorizer which implements both tokenization and tf-idf weighted counting in a single class.

[70]:
```
# Fit and transform the training data to a document-term matrix using␣
 ↪TfidfVectorizer
tfidf = TfidfVectorizer(min_df=5) #minimum document frequency of 5
X_train_tfidf = tfidf.fit_transform(X_train)
print("Number of features : %d \n" %len(tfidf.get_feature_names())) #1722
print("Show some feature names : \n", tfidf.get_feature_names()[::1000])

# Logistic Regression
lr = LogisticRegression()
lr.fit(X_train_tfidf, y_train)
```

```
Number of features : 5987

Show some feature names :
 ['00', 'changing', 'fall', 'letting', 'primarily', 'stars']

/home/sakil/anaconda/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)
```

[70]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

[71]:
```
# Look at the top 10 features with smallest and the largest coefficients
feature_names = np.array(tfidf.get_feature_names())
sorted_coef_index = lr.coef_[0].argsort()
print('\nTop 10 features with smallest coefficients :\n{}\n'.
 ↪format(feature_names[sorted_coef_index[:10]]))
print('Top 10 features with largest coefficients : \n{}'.
 ↪format(feature_names[sorted_coef_index[:-11:-1]]))
```

```
Top 10 features with smallest coefficients :
['not' 'return' 'disappointed' 'waste' 'horrible' 'worst' 'poor' 'slow'
 'stopped' 'doesn']
```

```
Top 10 features with largest coefficients :
['great' 'love' 'excellent' 'perfect' 'good' 'easy' 'best' 'far' 'amazing'
 'awesome']
```

[72]:
```
# Evaluate on the validaton set
predictions = lr.predict(tfidf.transform(X_test_cleaned))
modelEvaluation(predictions)
```

```
Accuracy on validation set: 0.9310

AUC score : 0.8985

Classification report :
              precision    recall  f1-score   support

           0       0.89      0.83      0.86       778
           1       0.94      0.96      0.95      2311

    accuracy                           0.93      3089
   macro avg       0.92      0.90      0.91      3089
weighted avg       0.93      0.93      0.93      3089


Confusion Matrix :
 [[ 648  130]
 [  83 2228]]
```

5 Pipeline and GridSearch

In sklearn library, we can build a pipeline to streamline the workflow and use GridSearch on the pipeline model to implement hyper-parameter tuning for both vectorizer and classifier in one go.

[73]:
```
# Building a pipeline
estimators = [("tfidf", TfidfVectorizer()), ("lr", LogisticRegression())]
model = Pipeline(estimators)


# Grid search
params = {"lr__C":[0.1, 1, 10], #regularization param of logistic regression
          "tfidf__min_df": [1, 3], #min count of words
          "tfidf__max_features": [1000, None], #max features
          "tfidf__ngram_range": [(1,1), (1,2)], #1-grams or 2-grams
          "tfidf__stop_words": [None, "english"]} #use stopwords or don't

grid = GridSearchCV(estimator=model, param_grid=params, scoring="accuracy",␣
 ↪n_jobs=-1)
grid.fit(X_train_cleaned, y_train)
```

```
print("The best paramenter set is : \n", grid.best_params_)


# Evaluate on the validaton set
predictions = grid.predict(X_test_cleaned)
modelEvaluation(predictions)
```

```
/home/sakil/anaconda/lib/python3.7/site-
packages/sklearn/model_selection/_split.py:1978: FutureWarning: The default
value of cv will change from 3 to 5 in version 0.22. Specify it explicitly to
silence this warning.
  warnings.warn(CV_WARNING, FutureWarning)
/home/sakil/anaconda/lib/python3.7/site-
packages/sklearn/linear_model/logistic.py:432: FutureWarning: Default solver
will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
  FutureWarning)

The best paramenter set is :
 {'lr__C': 10, 'tfidf__max_features': None, 'tfidf__min_df': 1,
'tfidf__ngram_range': (1, 2), 'tfidf__stop_words': None}

Accuracy on validation set: 0.9563

AUC score : 0.9375

Classification report :
               precision    recall  f1-score   support

           0       0.92      0.90      0.91       778
           1       0.97      0.98      0.97      2311

    accuracy                           0.96      3089
   macro avg       0.95      0.94      0.94      3089
weighted avg       0.96      0.96      0.96      3089


Confusion Matrix :
 [[ 700   78]
 [  57 2254]]
```

6 Part 4. Word2Vec

Another common approach of word embedding is prediction based embedding, such as Word2Vec model. In gist, Word2Vec is a combination of two techniques: Continuous Bag of Words (CBoW) and skip-gram model. Both are shallow neural networks which learn weights for word vector representations.

In this part, we will train Word2Vec model to create our own word vector representations using gensim library. Then we fit the feature vectors of the reviews to Random Forest Classifier. Here's the workflow of this part. Step 1 : Parse review text to sentences (Word2Vec model takes a list of sentences as inputs)

Step 2 : Create volcabulary list using Word2Vec model

Step 3 : Transform each review into numerical representation by computing average feature vectors of words therein

Step 4 : Fit the average feature vectors to Random Forest Classifier

7 Parsing Review into Sentences

Word2Vec model takes a list of sentences as inputs and outputs word vector representations for words in the vocalbulary list created. Before we train the Word2Vec model, we have to parse reviews in the training set into sentences.

[74]:
```
# Split review text into parsed sentences uisng NLTK's punkt tokenizer
# nltk.download()
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def parseSent(review, tokenizer, remove_stopwords=False):
    '''
    Parse text into sentences
    '''
    raw_sentences = tokenizer.tokenize(review.strip())
    sentences = []
    for raw_sentence in raw_sentences:
        if len(raw_sentence) > 0:
            sentences.append(cleanText(raw_sentence, remove_stopwords,␣
 ↪split_text=True))
    return sentences


# Parse each review in the training set into sentences
sentences = []
for review in X_train_cleaned:
    sentences += parseSent(review, tokenizer)

print('%d parsed sentence in the training set\n'  %len(sentences))
print('Show a parsed sentence in the training set : \n',  sentences[10])
```

```
27768 parsed sentence in the training set

Show a parsed sentence in the training set :
 ['good', 'product', 'and', 'fast', 'shipping', 'thank', 'you']
```

8 Creating Volcabulary List usinhg Word2Vec Model

Now we have a set of cleaned and parsed sentences from the training data, we can train our own word evctor representations by sepcifiying the embedding dimension (= length of feature vector).

[75]:
```
# Fit parsed sentences to Word2Vec model
# logging.basicConfig(format='%(asctime)s : %(levelname)s :␣
↪%(message)s',level=logging.INFO)

num_features = 300  #embedding dimension
min_word_count = 10
num_workers = 4
context = 10
downsampling = 1e-3

print("Training Word2Vec model ...\n")
w2v = Word2Vec(sentences, workers=num_workers, size=num_features, min_count =␣
↪min_word_count,\
                 window = context, sample = downsampling)
w2v.init_sims(replace=True)
w2v.save("w2v_300features_10minwordcounts_10context") #save trained word2vec␣
↪model

print("Number of words in the vocabulary list : %d \n" %len(w2v.wv.index2word))␣
↪#4016
print("Show first 10 words in the vocalbulary list  vocabulary list: \n", w2v.
↪wv.index2word[0:10])
```

```
Training Word2Vec model ...

Number of words in the vocabulary list : 4016

Show first 10 words in the vocalbulary list  vocabulary list:
 ['the', 'i', 'it', 'and', 'phone', 'a', 'to', 'is', 'this', 'for']
```

9 Averaging Feature Vectors

Now we have created a volcabulary list of words, with each word having a word representation (ie. feature vector of dim 300).

To find a numerical representation for a review, we run through each word in a review text. For words appear in the volcabulary list, we compute the average feature vectors of all those words. The average feature vector is the numerical represenation of the review.

[95]:
```
# Transfrom the training data into feature vectors

def makeFeatureVec(review, model, num_features):
    '''
```

```
    Transform a review to a feature vector by averaging feature vectors of␣
 ↪words
    appeared in that review and in the volcabulary list created
    '''
    featureVec = np.zeros((num_features,),dtype="float32")
    nwords = 0.
    index2word_set = set(model.wv.index2word) #index2word is the volcabulary␣
 ↪list of the Word2Vec model
    isZeroVec = True
    for word in review:
        if word in index2word_set:
            nwords = nwords + 1.
            featureVec = np.add(featureVec, model[word])
            isZeroVec = False
    if isZeroVec == False:
        featureVec = np.divide(featureVec, nwords)
    return featureVec


def getAvgFeatureVecs(reviews, model, num_features):
    '''
    Transform all reviews to feature vectors using makeFeatureVec()
    '''
    counter = 0
    reviewFeatureVecs = np.zeros((len(reviews),num_features),dtype="float32")
    for review in reviews:
        reviewFeatureVecs[counter] = makeFeatureVec(review, model,num_features)
        counter = counter + 1
    return reviewFeatureVecs
```

[98]:
```
# Get feature vectors for training set
X_train_cleaned = []
for review in X_train:
    X_train_cleaned.append(cleanText(review, remove_stopwords=True,␣
 ↪split_text=True))
trainVector = getAvgFeatureVecs(X_train_cleaned, w2v, num_features)
print("Training set : %d feature vectors with %d dimensions" %trainVector.shape)


# Get feature vectors for validation set
X_test_cleaned = []
for review in X_test:
    X_test_cleaned.append(cleanText(review, remove_stopwords=True,␣
 ↪split_text=True))
testVector = getAvgFeatureVecs(X_test_cleaned, w2v, num_features)
print("Validation set : %d feature vectors with %d dimensions" %testVector.
 ↪shape)
```

```
#import nltk
#nltk.download('stopwords')
# debugging
# print("Checkinf for NaN and Inf")
# print("np.inf=", np.where(np.isnan(trainVector)))
# print("is.inf=", np.where(np.isinf(trainVector)))
# print("np.max=", np.max(abs(trainVector)))
```

```
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "http://www.amazon.com/gp/product/B013YDFH3Y?redirect=true&ref_=cm_
cr_ryp_prd_ttl_sol_0" looks like a URL. Beautiful Soup is not an HTTP client.
You should probably use an HTTP client like requests to get the document behind
the URL, and feed that document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "https://www.amazon.com/dp/B00K15KRV6/ref=cm_cr_ryp_prd_ttl_sol_22"
looks like a URL. Beautiful Soup is not an HTTP client. You should probably use
an HTTP client like requests to get the document behind the URL, and feed that
document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
/home/sakil/anaconda/lib/python3.7/site-packages/bs4/__init__.py:375:
UserWarning: "http://www.amazon.com/gp/product/B0193D539M?redirect=true&ref_=cm_
cr_ryp_prd_ttl_sol_0" looks like a URL. Beautiful Soup is not an HTTP client.
You should probably use an HTTP client like requests to get the document behind
the URL, and feed that document to Beautiful Soup.
  ' that document to Beautiful Soup.' % decoded_markup
/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:15:
DeprecationWarning: Call to deprecated `__getitem__` (Method will be removed in
4.0.0, use self.wv.__getitem__() instead).
  from ipykernel import kernelapp as app

Training set : 27799 feature vectors with 300 dimensions
Validation set : 3089 feature vectors with 300 dimensions
```

[99]:
```
# Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100)
rf.fit(trainVector, y_train)
predictions = rf.predict(testVector)
modelEvaluation(predictions)
```

```
Accuracy on validation set: 0.9285

AUC score : 0.8972

Classification report :
```

```
              precision    recall  f1-score   support

           0       0.88      0.83      0.85       778
           1       0.95      0.96      0.95      2311

    accuracy                           0.93      3089
   macro avg       0.91      0.90      0.90      3089
weighted avg       0.93      0.93      0.93      3089


Confusion Matrix :
 [[ 649  129]
 [  92 2219]]
```

10 Part 5. LSTM

Long Short Term Memory networks (LSTM) are a special kind of Recurrent Neural Networks (RNN), capable of learning long-term dependencies. LSTM can be very usefull in text mining problems since it involves dependencies in the sentences which can be caught in the "memory" of the LSTM.

In this part, we train a simple LSTM and a LSTM with Word2Vec embedding to classify the reviews into positive and negative sentiment using Keras libarary.

11 Simple LSTM

We need to preprocess the text data to 2D tensor before we fit into a simple LSTM. First, we tokenize the corpus by only considering top words (top_words = 20000), and transform reviews to numerical sequences using the trained tokenizer. Next, we make sure that all numerical sequences have the same length (maxlen=100) for modeling, by truncating long reviews and pad shorter reviews with zero values.

To construct a simple LSTM, we use embedding class in Keras to construct the first layer. This embedding layer converts numerical sequence of words into a word embedding. We should note that the embedding class provides a convenient way to map discrete words into a continuous vector space, but it does not take the semantic similarity of the words into account. The next layer is the LSTM layer with 128 memory units. Finally, we use a dense output layer with a single neuron and a sigmoid activation function to make 0 or 1 predictions for the two classes (positive sentiment and negative sentiment). Since it is a binary classification problem, log loss is used as the loss function (binary_crossentropy in Keras). ADAM optimization algorithm is used.

Here's the workflow in this part.

Step 1 : Prepare X_train and X_test to 2D tensor

Step 2 : Train a simple LSTM (embeddign layer => LSTM layer => dense layer)

Step 3 : Compile and fit the model using log loss function and ADAM optimizer

[85]:
```
top_words = 20000
maxlen = 100
batch_size = 32
nb_classes = 2
```

```
nb_epoch = 3


# Vectorize X_train and X_test to 2D tensor
tokenizer = Tokenizer(nb_words=top_words) #only consider top 20000 words in the␣
↪corpse
tokenizer.fit_on_texts(X_train)
# tokenizer.word_index #access word-to-index dictionary of trained tokenizer

sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)

X_train_seq = sequence.pad_sequences(sequences_train, maxlen=maxlen)
X_test_seq = sequence.pad_sequences(sequences_test, maxlen=maxlen)


# one-hot encoding of y_train and y_test
y_train_seq = np_utils.to_categorical(y_train, nb_classes)
y_test_seq = np_utils.to_categorical(y_test, nb_classes)

print('X_train shape:', X_train_seq.shape) #(27799, 100)
print('X_test shape:', X_test_seq.shape) #(3089, 100)
print('y_train shape:', y_train_seq.shape) #(27799, 2)
print('y_test shape:', y_test_seq.shape) #(3089, 2)
```

```
/home/sakil/anaconda/lib/python3.7/site-
packages/keras_preprocessing/text.py:178: UserWarning: The `nb_words` argument
in `Tokenizer` has been renamed `num_words`.
  warnings.warn('The `nb_words` argument in `Tokenizer` '

X_train shape: (27799, 100)
X_test shape: (3089, 100)
y_train shape: (27799, 2)
y_test shape: (3089, 2)
```

[86]:
```
# Construct a simple LSTM
model1 = Sequential()
model1.add(Embedding(top_words, 128, dropout=0.2))
model1.add(LSTM(128, dropout_W=0.2, dropout_U=0.2))
model1.add(Dense(nb_classes))
model1.add(Activation('softmax'))
model1.summary()

# Compile LSTM
model1.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

```
model1.fit(X_train_seq, y_train_seq, batch_size=batch_size, nb_epoch=nb_epoch,␣
 ↪verbose=1)

# Model evaluation
score = model1.evaluate(X_test_seq, y_test_seq, batch_size=batch_size)
print('Test loss : {:.4f}'.format(score[0]))
print('Test accuracy : {:.4f}'.format(score[1]))
```

```
/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:3:
UserWarning: The `dropout` argument is no longer support in `Embedding`. You can
apply a `keras.layers.SpatialDropout1D` layer right after the `Embedding` layer
to get the same behavior.
  This is separate from the ipykernel package so we can avoid doing imports
until
/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:4:
UserWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(128, dropout=0.2,
recurrent_dropout=0.2)`
  after removing the cwd from sys.path.

Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_2 (Embedding)      (None, None, 128)         2560000
_________________________________________________________________
lstm_2 (LSTM)                (None, 128)               131584
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 258
_________________________________________________________________
activation_2 (Activation)    (None, 2)                 0
=================================================================
Total params: 2,691,842
Trainable params: 2,691,842
Non-trainable params: 0
_________________________________________________________________

/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:14:
UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.


Epoch 1/3
27799/27799 [==============================] - 279s 10ms/step - loss: 0.2728 -
accuracy: 0.8874
Epoch 2/3
27799/27799 [==============================] - 297s 11ms/step - loss: 0.1518 -
accuracy: 0.9458
Epoch 3/3
```

```
27799/27799 [==============================] - 293s 11ms/step - loss: 0.1135 -
accuracy: 0.9614
3089/3089 [==============================] - 10s 3ms/step
Test loss : 0.1640
Test accuracy : 0.9411
```

[87]:
```
# get weight matrix of the embedding layer
model1.layers[0].get_weights()[0] # weight matrix of the embedding layer,␣
 ↪word-by-dim matrix
print("Size of weight matrix in the embedding layer : ", \
      model1.layers[0].get_weights()[0].shape) #(20000, 128)

# get weight matrix of the hidden layer
print("Size of weight matrix in the hidden layer : ", \
      model1.layers[1].get_weights()[0].shape) #(128, 512)  weight dim of LSTM␣
 ↪- w

# get weight matrix of the output layer
print("Size of weight matrix in the output layer : ", \
      model1.layers[2].get_weights()[0].shape) #(128, 2) weight dim of dense␣
 ↪layer
```

```
Size of weight matrix in the embedding layer :  (20000, 128)
Size of weight matrix in the hidden layer :  (128, 512)
Size of weight matrix in the output layer :  (128, 2)
```

12 LSTM with Word2Vec Embedding

In the simple LSTM model constructed above, the embedding class in Keras comes in handy to converts numerical sequence of words into a word embedding, but it does not take the semantic similarity of the words into account. The model assigns random weights to the embedding layer and learn the embeddings by minimizing the global error of the network.

Instead of using random weights, we can use pretrained word embeddings to initialize the weight of an embedding layer. In this part, we use the Word2Vec embedding trained in Part 4 to intialize the weights of embedding layer in LSTM. Step 1 : Load pretrained word embedding model

Step 2 : Construct embedding layer using embedding matrix as weights

Step 3 : Train a LSTM with Word2Vec embedding (embeddign layer => LSTM layer => dense layer)

Step 4 : Compile and fit the model using log loss function and ADAM optimizer

[88]:
```
# Load trained Word2Vec model
w2v = Word2Vec.load("w2v_300features_10minwordcounts_10context")



# Get Word2Vec embedding matrix
embedding_matrix = w2v.wv.syn0  # embedding matrix, type = numpy.ndarray
```

```
print("Shape of embedding matrix : ", embedding_matrix.shape) #(4016, 300) =␣
↪(volcabulary size, embedding dimension)
# w2v.wv.syn0[0] #feature vector of the first word in the volcabulary list
```

```
Shape of embedding matrix :  (4016, 300)
```

```
/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:6:
DeprecationWarning: Call to deprecated `syn0` (Attribute will be removed in
4.0.0, use self.vectors instead).
```

[89]:

```
top_words = embedding_matrix.shape[0] #4016
maxlen = 100
batch_size = 32
nb_classes = 2
nb_epoch = 3


# Vectorize X_train and X_test to 2D tensor
tokenizer = Tokenizer(nb_words=top_words) #only consider top 20000 words in the␣
↪corpse
tokenizer.fit_on_texts(X_train)
# tokenizer.word_index #access word-to-index dictionary of trained tokenizer

sequences_train = tokenizer.texts_to_sequences(X_train)
sequences_test = tokenizer.texts_to_sequences(X_test)

X_train_seq = sequence.pad_sequences(sequences_train, maxlen=maxlen)
X_test_seq = sequence.pad_sequences(sequences_test, maxlen=maxlen)


# one-hot encoding of y_train and y_test
y_train_seq = np_utils.to_categorical(y_train, nb_classes)
y_test_seq = np_utils.to_categorical(y_test, nb_classes)

print('X_train shape:', X_train_seq.shape) #(27799, 100)
print('X_test shape:', X_test_seq.shape) #(3089, 100)
print('y_train shape:', y_train_seq.shape) #(27799, 2)
print('y_test shape:', y_test_seq.shape) #(3089, 2)
```

```
/home/sakil/anaconda/lib/python3.7/site-
packages/keras_preprocessing/text.py:178: UserWarning: The `nb_words` argument
in `Tokenizer` has been renamed `num_words`.
  warnings.warn('The `nb_words` argument in `Tokenizer` '
```

```
X_train shape: (27799, 100)
X_test shape: (3089, 100)
```

```
y_train shape: (27799, 2)
y_test shape: (3089, 2)
```

[90]:
```python
# Construct Word2Vec embedding layer
embedding_layer = Embedding(embedding_matrix.shape[0], #4016
                            embedding_matrix.shape[1], #300
                            weights=[embedding_matrix])


# Construct LSTM with Word2Vec embedding
model2 = Sequential()
model2.add(embedding_layer)
model2.add(LSTM(128, dropout_W=0.2, dropout_U=0.2))
model2.add(Dense(nb_classes))
model2.add(Activation('softmax'))
model2.summary()

# Compile model
model2.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])

model2.fit(X_train_seq, y_train_seq, batch_size=batch_size, nb_epoch=nb_epoch,␣
 ↪verbose=1)


# Model evaluation
score = model2.evaluate(X_test_seq, y_test_seq, batch_size=batch_size)
print('Test loss : {:.4f}'.format(score[0]))
print('Test accuracy : {:.4f}'.format(score[1]))
```

```
/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:10:
UserWarning: Update your `LSTM` call to the Keras 2 API: `LSTM(128, dropout=0.2,
recurrent_dropout=0.2)`
  # Remove the CWD from sys.path while we load stuff.

Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_3 (Embedding)      (None, None, 300)         1204800
_________________________________________________________________
lstm_3 (LSTM)                (None, 128)               219648
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
_________________________________________________________________
activation_3 (Activation)    (None, 2)                 0
=================================================================
```

```
Total params: 1,424,706
Trainable params: 1,424,706
Non-trainable params: 0
_________________________________________________________________

/home/sakil/anaconda/lib/python3.7/site-packages/ipykernel_launcher.py:20:
UserWarning: The `nb_epoch` argument in `fit` has been renamed `epochs`.

Epoch 1/3
27799/27799 [==============================] - 338s 12ms/step - loss: 0.2812 -
accuracy: 0.8801
Epoch 2/3
27799/27799 [==============================] - 345s 12ms/step - loss: 0.1753 -
accuracy: 0.9346
Epoch 3/3
27799/27799 [==============================] - 333s 12ms/step - loss: 0.1328 -
accuracy: 0.9507
3089/3089 [==============================] - 11s 4ms/step
Test loss : 0.1740
Test accuracy : 0.9359
```

[91]:

```
# get weight matrix of the embedding layer
print("Size of weight matrix in the embedding layer : ", \
      model2.layers[0].get_weights()[0].shape) #(20000, 128)

# get weight matrix of the hidden layer
print("Size of weight matrix in the hidden layer : ", \
      model2.layers[1].get_weights()[0].shape) #(128, 512)  weight dim of LSTM␣
 ↪- w

# get weight matrix of the output layer
print("Size of weight matrix in the output layer : ", \
      model2.layers[2].get_weights()[0].shape) #(128, 2) weight dim of dense␣
 ↪layer
```

```
Size of weight matrix in the embedding layer :  (4016, 300)
Size of weight matrix in the hidden layer :  (300, 512)
Size of weight matrix in the output layer :  (128, 2)
```

13 Part 6. Word Cloud

In this part, we create word clouds for positive sentiment reviews and negative sentiment reviews of a selected brand, to get an intuition of words frequently appear in different sentiments.

[100]:

```
def create_word_cloud(brand, sentiment):
    try:
        df_brand = df.loc[df['Brand Name'].isin([brand])]
        df_brand_sample = df_brand.sample(frac=0.1)
```

```
        word_cloud_collection = ''

        if sentiment == 1:
            df_reviews =␣
↪df_brand_sample[df_brand_sample["Sentiment"]==1]["Reviews"]

        if sentiment == 0:
            df_reviews =␣
↪df_brand_sample[df_brand_sample["Sentiment"]==0]["Reviews"]

        for val in df_reviews.str.lower():
            tokens = nltk.word_tokenize(val)
            tokens = [word for word in tokens if word not in stopwords.
↪words('english')]
            for words in tokens:
                word_cloud_collection = word_cloud_collection + words + ' '

        wordcloud = WordCloud(max_font_size=50, width=500, height=300).
↪generate(word_cloud_collection)
        plt.figure(figsize=(30,30))
        plt.imshow(wordcloud)
        plt.axis("off")
        plt.show()
    except:
        pass
```

[101]:
```
create_word_cloud(brand='Apple', sentiment=1)
```

[102]:
```
create_word_cloud(brand='Apple', sentiment=0)
```