AI Capstone Project - Domain: E-Commerce

Description Problem Statement

- Amazon is an online shopping website that now caters to millions of people everywhere. Over 34,000 consumer reviews for Amazon brand products like Kindle, Fire TV Stick and more are provided.
- The dataset has attributes like brand, categories, primary categories, reviews.title, reviews.text, and the sentiment. Sentiment is a categorical variable with three levels "Positive", "Negative", and "Neutral". For a given unseen data, the sentiment needs to be predicted.
- You are required to predict Sentiment or Satisfaction of a purchase based on multiple features and review text.

Project Task : Week 1

1. Perform EDA on dataset

In [1]:
```
# Importing required libraries
# Basic libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import re
import seaborn as sns
import tensorflow as tf
from bs4 import BeautifulSoup
from collections import Counter, defaultdict
import warnings
warnings.filterwarnings('ignore')

# Sklearn libraries
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression, SGDClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.pipeline import Pipeline
from sklearn.metrics import precision_score, recall_score, confusion_matrix, f1_sc
from sklearn.dummy import DummyClassifier
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer, Tfid

# Ovesampling libraries
from imblearn.over_sampling import RandomOverSampler

# NLP libraries
from wordcloud import WordCloud, STOPWORDS
import nltk
from nltk.corpus import stopwords, wordnet
from nltk.tokenize import RegexpTokenizer
from nltk import word_tokenize, sent_tokenize, pos_tag
from nltk.stem import WordNetLemmatizer
from nltk.stem.porter import PorterStemmer
nltk.download('stopwords')
nltk.download('wordnet')
```

```
nltk.download('omw-1.4')

from gensim import corpora
from gensim.models import Word2Vec
from gensim.models.keyedvectors import KeyedVectors
from gensim.models import LdaModel

# Deeplearning libraries and modules
import keras.backend as kb
from keras.preprocessing import sequence
from keras.models import Sequential
from keras.layers.core import Dense, Activation, Dropout, Lambda
from keras.layers.embeddings import Embedding
from keras.layers.recurrent import LSTM, GRU, SimpleRNN
from keras.layers.convolutional import Convolution1D
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.activations import softmax

from keras.utils import np_utils
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

In [2]:

```
train_data = pd.read_csv('train_data.csv')
test_data = pd.read_csv('test_data.csv')
test_data_hidden = pd.read_csv('test_data_hidden.csv')
```

In [3]:

```
train_data.head()
```

Out[3]:

	name	brand	categories	primaryCategories	reviews.date	reviews.text
0	All-New Fire HD 8 Tablet, 8" HD Display, Wi-Fi...	Amazon	Electronics,iPad & Tablets,All Tablets,Fire Ta...	Electronics	2016-12-26T00:00:00.000Z	Purchased on Black FridayPros - Great Price (e...
1	Amazon - Echo Plus w/ Built-In Hub - Silver	Amazon	Amazon Echo,Smart Home,Networking,Home & Tools...	Electronics,Hardware	2018-01-17T00:00:00.000Z	I purchased two Amazon in Echo Plus and two do...
2	Amazon Echo Show Alexa-enabled Bluetooth Speak...	Amazon	Amazon Echo,Virtual Assistant Speakers,Electro...	Electronics,Hardware	2017-12-20T00:00:00.000Z	Just an average Alexa option. Does show a few ...
3	Fire HD 10 Tablet, 10.1 HD Display, Wi-Fi, 16 ...	Amazon	eBook Readers,Fire Tablets,Electronics Feature...	Office Supplies,Electronics	2017-08-04T00:00:00.000Z	very good product. Exactly what I wanted, and ...
4	Brand New Amazon Kindle Fire 16gb 7" Ips Displ...	Amazon	Computers/Tablets & Networking,Tablets & eBook...	Electronics	2017-01-23T00:00:00.000Z	This is the 3rd one I've purchased. I've bough...

Seeing what a positive, neutral and negative looks like & checking class counts for each one

In [4]:

```
# Storing review categories in to separate variables
positive = train_data[train_data['sentiment']=="Positive"].iloc[:,[5,6,7]]
neutral = train_data[train_data['sentiment']=="Neutral"].iloc[:,[5,6,7]]
negative = train_data[train_data['sentiment']=="Negative"].iloc[:,[5,6,7]]
```

In [5]:

```
# Looking at what a reviews of different categories look like
print("Positive\nValue counts: {} \nSample: {}".format(positive.shape[0],positive[
print("\nNeutral\nValue counts: {} \nSample: {}".format(neutral.shape[0],neutral['
print("\nNegative\nValue counts: {} \nSample: {}".format(negative.shape[0],negativ
```

```
Positive
Value counts: 3749
Sample: Purchased on Black FridayPros - Great Price (even off sale)Very powerful a
nd fast with quad core processors Amazing soundWell builtCons -Amazon ads, Amazon
need this to subsidize the tablet and will remove the adds if you pay them $15.Ina
bility to access other apps except the ones from Amazon. There is a way which I wa
s able to accomplish to add the Google Play storeNet this is a great tablet for th
e money

Neutral
Value counts: 158
Sample: Just an average Alexa option. Does show a few things on screen but still l
imited.

Negative
Value counts: 93
Sample: was cheap, can not run chrome stuff, returned to store.
```

In [6]:
```
# Keeping only required features
train_data_new = train_data[['sentiment', 'reviews.text']]
```

In [7]:
```
train_data_new.columns
```

Out[7]:
```
Index(['sentiment', 'reviews.text'], dtype='object')
```

In [8]:
```
# Resetting the index
train_data_new.index = pd.Series(list(range(train_data_new.shape[0])))
```

In [9]:
```
train_data_new.shape
```

Out[9]:
```
(4000, 2)
```

In [10]:
```
# Initializing modules
wordnetlemmatizer = WordNetLemmatizer()
tokenizer = RegexpTokenizer(r'[a-z]+') # Selecting only text
stop_words = set(stopwords.words('english'))
```

In [11]:
```
import string
# Defining a text preprocessing function
def preprocess_text(document):
  document = document.lower()
  words = tokenizer.tokenize(document)
  words = [w for w in words if not w in stop_words]

  #Lemmatizing
  for pos in [wordnet.NOUN, wordnet.ADV, wordnet.ADJ, wordnet.VERB]:
    words = [wordnetlemmatizer.lemmatize(x,pos) for x in words]

  return ' '.join(words)
```

In [12]:
```
train_data_new['processed_review'] = train_data_new['reviews.text'].apply(preproce
```

In [13]:
```
train_data_psd = train_data_new[['sentiment', 'processed_review']]
```

In [14]:
```
def preprocess_text2(data2):
    #Remove Punctuation Logic
    import string
    removePunctuation = [char for char in data2 if char not in string.punctuation]
    #Join Chars to form sentences
    sentenceWithoutPunctuations = ''.join(removePunctuation)
    words = sentenceWithoutPunctuations.split()
```

```
    #StopwordRemoval
    from nltk.corpus import stopwords
    removeStopwords = [word for word in words if word.lower() not in stopwords.word

    return removeStopwords
```

In [15]:
```
train_data_psd['processed_review'].apply(preprocess_text2)
```

Out[15]:
```
0       [purchase, black, fridaypros, great, price, ev...
1       [purchase, two, amazon, echo, plus, two, dot, ...
2       [average, alexa, option, show, thing, screen, ...
3             [good, product, exactly, want, good, price]
4       [rd, one, purchase, buy, one, niece, case, com...
                              ...
3995    [fun, family, play, may, get, bore, newness, w...
3996    [love, kindle, great, product, reduce, eye, st...
3997    [look, blutooth, speaker, use, phone, want, wo...
3998    [second, amazon, fire, tablet, purchase, time,...
3999                   [satisfy, tablet, fast, efficient]
Name: processed_review, Length: 4000, dtype: object
```

In [16]:
```
train_data_psd.groupby('sentiment').describe()
```

Out[16]:

	processed_review			
	count	unique	top	freq
sentiment				
Negative	93	78	last model kindle hdx terrible purchase model ...	3
Neutral	158	145	average alexa option show thing screen still l...	2
Positive	3749	3372	buy kindle yr old granddaughter christmas husb...	4

Converting the reviews in to TF-IDF score

In [17]:
```
bow = CountVectorizer(analyzer=preprocess_text2).fit(train_data_psd['processed_rev
reviews_bow = bow.transform(train_data_psd['processed_review'])
print(bow.vocabulary)
tfidf_init = TfidfTransformer().fit(reviews_bow)
tfidf_data = tfidf_init.transform(reviews_bow)
tfidf_data.shape
```

```
None
(4000, 3408)
```

Out[17]:

Running Multinomial Naivae Bayes Classifier

In [18]:
```
# Running Multinomial NaiveBayes classifer on transformed data
nb_classifier = MultinomialNB()
nb_classifier.fit(tfidf_data, train_data_psd['sentiment'])
```

Out[18]:
```
MultinomialNB()
```

In [19]:
```
sample_review = "This is a worst product. I don't prefer buying it next time."
prep_review = preprocess_text2(sample_review)
bow_review = bow.transform(prep_review)
tfidf_review = tfidf_init.transform(bow_review)
```

```
prediction = nb_classifier.predict(tfidf_review[0])
print(prediction)
```

```
['Positive']
```

Since the dataset has class imbalances problem, we can see that even a bad review is classified as positive

Project task week 2

1. Tackling class imbalance problem

In [20]:
```
train_data_psd.columns
```

Out[20]:
```
Index(['sentiment', 'processed_review'], dtype='object')
```

In [21]:
```
X = train_data_psd.drop('sentiment', axis=1)
y = train_data_psd['sentiment']
print("X shape: ", X.shape)
print("y shape: ", y.shape)
```

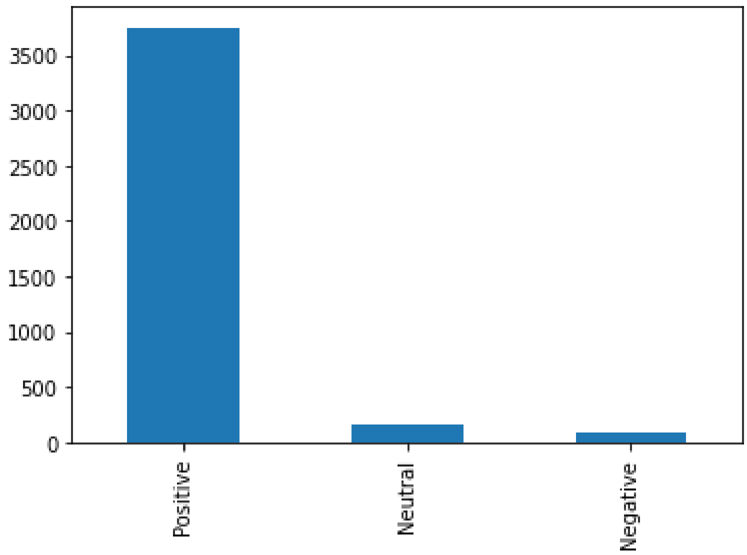
```
X shape:  (4000, 1)
y shape:  (4000,)
```

In [22]:
```
train_data_psd.sentiment.value_counts()
```

Out[22]:
```
Positive    3749
Neutral      158
Negative      93
Name: sentiment, dtype: int64
```

In [23]:
```
train_data_psd['sentiment'].value_counts().plot(kind='bar')
```

Out[23]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6d6d1fd0>
```



Oversampling the dataset using RandomOverSampler

In [24]:
```
# Over sampling the dataset using randomoversampler to tackle imbalance problem
ros = RandomOverSampler(random_state=1)
X_res, y_res = ros.fit_resample(X, y)
```

In [25]:
```
Counter(y_res)
```

Out[25]:
```
Counter({'Negative': 3749, 'Neutral': 3749, 'Positive': 3749})
```

In [26]:
```
print("Before sampling: ", Counter(y))
print("After sampling: ", Counter(y_res))
```

```
Before sampling:  Counter({'Positive': 3749, 'Neutral': 158, 'Negative': 93})
After sampling:  Counter({'Positive': 3749, 'Neutral': 3749, 'Negative': 3749})
```

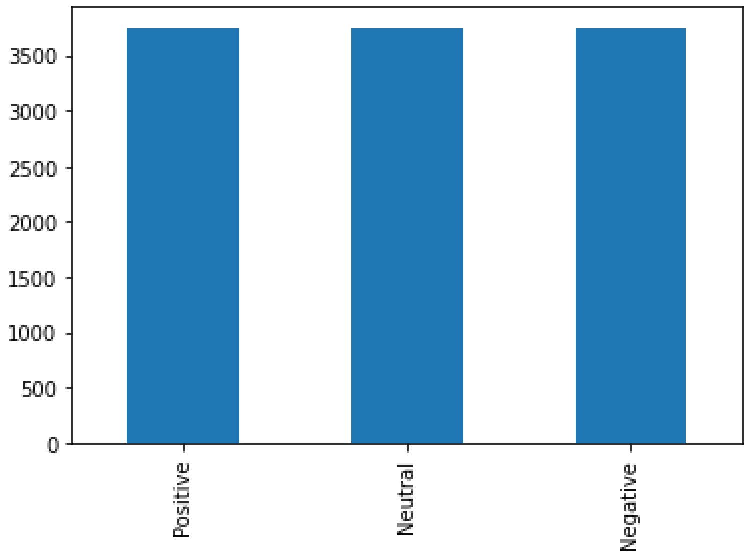
In [27]:
```
final_df = pd.concat([X_res, y_res], axis=1)
final_df.head()
```

Out[27]:

	processed_review	sentiment
0	purchase black fridaypros great price even sal...	Positive
1	purchase two amazon echo plus two dot plus fou...	Positive
2	average alexa option show thing screen still l...	Neutral
3	good product exactly want good price	Positive
4	rd one purchase buy one niece case compare one...	Positive

In [28]:
```
final_df['sentiment'].value_counts().plot(kind='bar')
```

Out[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2e6d616bd0>
```



we can see in the above figure that that dataset samples are balanced

In [29]:
```
final_df.isnull().sum()
```

Out[29]:
```
processed_review    0
sentiment           0
dtype: int64
```

In [30]:
```
final_df.shape
```

Out[30]:
```
(11247, 2)
```

In [31]:
```
# Applying sampling on whole dataset
final_data = final_df.sample(frac=0.1, random_state=1)
final_data.head()
```

Out[31]:

	processed_review	sentiment
674	would take charge good thing try wrap christmas	Negative
7149	buy kindle year old grand daughter buy warrant...	Negative
625	best purchase mad tablet daughter love	Positive
4139	first tablet kindle curious update version dis...	Negative
10290	think well mother law play game facebook next ...	Neutral

In [32]:

```
# Train test and split data
X_train, X_test, y_train, y_test = train_test_split(final_data['processed_review']
```

In [33]:

```
print("X train shape: ", X_train.shape)
print("X test shape: ", X_test.shape)
print(X_train.iloc[5])
```

```
X train shape:  (1012,)
X test shape:  (113,)
god tablet camera could little well
```

In [34]:

```
X_train
```

Out[34]:

```
217              give gift kid autism help development lot
4772                  screen dark cannot adjust brightness
4171     last model kindle hdx terrible purchase model ...
177                                             work great
6492           touch load content bad respond quickly slow
                               ...
1703     family love boy love ask alexa tell joke make ...
6817     proprietary apps daughter like could install b...
10049    cool device main issue play playlist store son...
2653                              box easy set even easy use
10094                         disappoint mirror display mode
Name: processed_review, Length: 1012, dtype: object
```

Project task week 3

Model selection

1. Apply multiclass SVM's and neural nets
2. Use possible ensemble techniques (XG Boost, Oversampled Multinomial NB)
3. Assign a score to the sentence sentiment- feature engineer a new variable called sentiment score

In [35]:

```
# Cleaning the text
def cleanText(raw_text, remove_stopwords=False, stemming=False, split_text=False):
  # Convert raw reviews into cleaned reviews
  # Select letters only
  text = BeautifulSoup(raw_text, 'lxml').get_text()
  letters_only = re.sub("[^a-zA-Z]"," ", text)
  words = letters_only.lower().split()

  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
```

```python
        # stemmer = PorterStemmer()
        stemmer = SnowballStemmer('english')
        words = [stemmer.stem(w) for w in words]

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

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

In [36]:
```python
# cleaning the text
X_train_cleaned = []
X_test_cleaned = []

for d in X_train:
  X_train_cleaned.append(cleanText(d))

print("X train cleaned sample: ", X_train_cleaned[5])

for d in X_test:
  X_test_cleaned.append(cleanText(d))
print("X test cleaned sample: ", X_test_cleaned[5])
```

```
X train cleaned sample:  god tablet camera could little well
X test cleaned sample:  great go companion avid reader easy load book connect prim
e
```

In [37]:
```python
# Fit and transform the training data in to a vectorizer
countVect = CountVectorizer()
X_train_cv = countVect.fit_transform(X_train_cleaned)

print('Number of features: ', len(countVect.get_feature_names()))
print('Feature samples: ', countVect.get_feature_names()[:5])

# Initialize and fit a MultinommialNB classifer
mnb_class = MultinomialNB()
mnb_class.fit(X_train_cv, y_train)
```

```
Number of features:  1588
Feature samples:  ['ability', 'able', 'absolute', 'absolutely', 'access']
```

Out[37]:
```
MultinomialNB()
```

In [38]:
```python
def model_evaluation(predictions):
  print("Accuracy of the model: {:.4f}".format(accuracy_score(y_test, predictions)
  print("Classification reports: \n", classification_report(y_test, predictions))
  print("Confusion matrix: \n", confusion_matrix(y_test, predictions))
```

In [39]:
```python
predictions = mnb_class.predict(countVect.transform(X_test_cleaned))
model_evaluation(predictions)
```

```
Accuracy of the model: 0.8673
Classification reports:
               precision    recall  f1-score   support

    Negative       0.94      0.83      0.88        41
     Neutral       0.81      0.94      0.87        36
    Positive       0.86      0.83      0.85        36

    accuracy                           0.87       113
   macro avg       0.87      0.87      0.87       113
weighted avg       0.87      0.87      0.87       113

Confusion matrix:
 [[34  3  4]
 [ 1 34  1]
 [ 1  5 30]]
```

In [40]:

```
# Tdidf vectorizer with logistic regression
tfidf = TfidfVectorizer(min_df=5)
X_train_tfidf = tfidf.fit_transform(X_train)

print('Number of features: ', len(tfidf.get_feature_names()))
print('Feature samples: ', tfidf.get_feature_names()[:5])

# Initialize and fit a MultinommialNB classifer
log_regr = LogisticRegression()
log_regr.fit(X_train_tfidf, y_train)
```

```
Number of features:  664
Feature samples:  ['able', 'absolutely', 'access', 'account', 'activate']
```

Out[40]:

```
LogisticRegression()
```

In [41]:

```
predictions = log_regr.predict(tfidf.transform(X_test_cleaned))
model_evaluation(predictions)
```

```
Accuracy of the model: 0.9115
Classification reports:
               precision    recall  f1-score   support

    Negative       0.93      0.95      0.94        41
     Neutral       0.91      0.89      0.90        36
    Positive       0.89      0.89      0.89        36

    accuracy                           0.91       113
   macro avg       0.91      0.91      0.91       113
weighted avg       0.91      0.91      0.91       113

Confusion matrix:
 [[39  1  1]
 [ 1 32  3]
 [ 2  2 32]]
```

In [42]:

```
# Tfidf vectorizer using SGD classifer
SGDclass = SGDClassifier()
SGDclass.fit(X_train_tfidf, y_train)
```

Out[42]:

```
SGDClassifier()
```

In [43]:

```
predictions = SGDclass.predict(tfidf.transform(X_test_cleaned))
model_evaluation(predictions)
```

```
Accuracy of the model: 0.9204
Classification reports:
               precision    recall  f1-score   support

    Negative       0.89      0.98      0.93        41
     Neutral       0.90      0.97      0.93        36
    Positive       1.00      0.81      0.89        36

    accuracy                           0.92       113
   macro avg       0.93      0.92      0.92       113
weighted avg       0.93      0.92      0.92       113

Confusion matrix:
 [[40  1  0]
 [ 1 35  0]
 [ 4  3 29]]
```

In [44]:
```
# Taking look of top 10 features with smallest and largest coefficents
feature_names = np.array(tfidf.get_feature_names())
sorted_coef = np.argsort(SGDclass.coef_[0])
print("Top 10 features with largest coefficients:\n", feature_names[sorted_coef[:1
print("Top 10 features with smallest coefficients:\n", feature_names[sorted_coef[:
```

```
Top 10 features with largest coefficients:
 ['easy' 'love' 'starter' 'great' 'command' 'account' 'hook' 'show' 'hd'
 'affordable']
Top 10 features with smallest coefficients:
 ['terrible' 'return' 'update' 'poor' 'minute' 'bad' 'exchange' 'bridge'
 'youtube' 'protective']
```

In [45]:
```
# Using XGBoost classifier
XGBclass = XGBClassifier()
XGBclass.fit(X_train_tfidf, y_train)
```

Out[45]:
```
XGBClassifier(objective='multi:softprob')
```

In [46]:
```
predictions = SGDclass.predict(tfidf.transform(X_test_cleaned))
model_evaluation(predictions)
```

```
Accuracy of the model: 0.9204
Classification reports:
               precision    recall  f1-score   support

    Negative       0.89      0.98      0.93        41
     Neutral       0.90      0.97      0.93        36
    Positive       1.00      0.81      0.89        36

    accuracy                           0.92       113
   macro avg       0.93      0.92      0.92       113
weighted avg       0.93      0.92      0.92       113

Confusion matrix:
 [[40  1  0]
 [ 1 35  0]
 [ 4  3 29]]
```

In [47]:
```
# Using the Pipeline and GridSearchCV
estimators = [("tfidf", TfidfVectorizer()), ("lr", LogisticRegression())]
model = Pipeline(estimators)


# Defining parameters to tune
params = {"lr__C":[0.1, 1, 10],
          "tfidf__min_df": [1, 3],
```

```
            "tfidf__max_features": [1000, None],
            "tfidf__ngram_range": [(1,1), (1,2)],
            "tfidf__stop_words": [None, "english"]}

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

Out[47]:

```
GridSearchCV(estimator=Pipeline(steps=[('tfidf', TfidfVectorizer()),
                                       ('lr', LogisticRegression())]),
             n_jobs=-1,
             param_grid={'lr__C': [0.1, 1, 10],
                         'tfidf__max_features': [1000, None],
                         'tfidf__min_df': [1, 3],
                         'tfidf__ngram_range': [(1, 1), (1, 2)],
                         'tfidf__stop_words': [None, 'english']},
             scoring='accuracy')
```

In [48]:

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

```
Accuracy of the model: 0.9646
Classification reports:
               precision    recall  f1-score   support

    Negative       1.00      0.98      0.99        41
     Neutral       0.90      1.00      0.95        36
    Positive       1.00      0.92      0.96        36

    accuracy                           0.96       113
   macro avg       0.97      0.96      0.96       113
weighted avg       0.97      0.96      0.96       113

Confusion matrix:
 [[40  1  0]
 [ 0 36  0]
 [ 0  3 33]]
```

In [49]:

```
# Word2Vec
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[49]:

```
True
```

In [50]:

```
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')

def parse_sentence(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

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

print('Parsed sentences in training set: ', len(sentences))
print('Parsed sentense sample: ', sentences[10])
```

```
Parsed sentences in training set:  1012
Parsed sentense sample:  ['mom', 'love', 'kindle', 'fire', 'hd', 'first', 'kindl
e', 'time', 'upgrade', 'great', 'buy', 'birthday', 'present']
```

Creating a vocabulary list using Word2Vec model

In [51]:
```python
# Fitting parsed sentences to Word2Vec model
num_features = 300
min_word_count = 10
num_workers = 4
context = 10
downsampling  = 1e-3

w2v = Word2Vec(sentences, workers=4, size=300, min_count=10,window=10,sample=1e-3)
w2v.init_sims(replace=True)
w2v.save('w2v1')

print('Words ini vocabulary list: ', len(w2v.wv.index2word))
print('First 10 words: ', w2v.wv.index2word[:10])
```

```
Words ini vocabulary list:  414
First 10 words:  ['tablet', 'use', 'buy', 'great', 'get', 'work', 'kindle', 'amazo
n', 'one', 'love']
```

Average feature vectors

In [52]:
```python
# Transfoming training data inoto feature vectors
def makeFeatureVectors(reviews, model, num_features):
  featureVec = np.zeros((num_features,), dtype='float32')
  nwords=0.0
  index2word_set = set(model.wv.index2word)
  isZeroVec = True
  for word in reviews:
    if word in index2word_set:
      nwords = nwords + 1.0
      featureVec = np.add(featureVec, model[word])
      isZeroVec = False
  if isZeroVec == False:
    featureVec = np.divide(featureVec, nwords)
  return featureVec

def getAvgFeatureVectors(reviews, model, num_features):
  counter = 0
  reviewFeatureVectors = np.zeros((len(reviews), num_features), dtype='float32')
  for review in reviews:
    reviewFeatureVectors[counter] = makeFeatureVectors(review, model, num_features
    counter = counter + 1
  return reviewFeatureVectors
```

In [53]:
```python
# Getting feature vectors for training set
trainVector = getAvgFeatureVectors(X_train, w2v, num_features)
# Getting feature vectors for validation set
testVector = getAvgFeatureVectors(X_test, w2v, num_features)
```

In [54]:
```python
print("Training set : %d feature vectors with %d dimensions" %trainVector.shape)
print("Validation set : %d feature vectors with %d dimensions" %testVector.shape)
```

```
Training set : 1012 feature vectors with 300 dimensions
Validation set : 113 feature vectors with 300 dimensions
```

Random forest classifier

In [55]:
```
rfc = RandomForestClassifier(n_estimators=100)
rfc.fit(trainVector, y_train)
predictions = rfc.predict(testVector)
model_evaluation(predictions)
```

```
Accuracy of the model: 0.4513
Classification reports:
               precision    recall  f1-score   support

    Negative       0.54      0.61      0.57        41
     Neutral       0.41      0.42      0.41        36
    Positive       0.37      0.31      0.33        36

    accuracy                           0.45       113
   macro avg       0.44      0.44      0.44       113
weighted avg       0.44      0.45      0.45       113

Confusion matrix:
 [[25  8  8]
 [10 15 11]
 [11 14 11]]
```

Project task week 4

Applying LSTM

1. Use LSTM for previous problem
2. Compare the accuracy of neural nets with traditional ML based algorithms
3. Find the best setting LSTM (neural net) & GRU that can best classify the reviewsas positive, negative and neutral (Use GridSearchCV & RandomSearch)

Applying LSTM

In [56]:
```
df = final_df.sample(frac=0.1,random_state=1)
# dropping missing values
df.dropna(inplace=True)
# Convert sentiments by replacing with numbers
df.sentiment.replace(('Positive', 'Negative', 'Neutral'),(1,0,2), inplace=True)
df.shape
```

Out[56]:
```
(1125, 2)
```

In [57]:
```
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(df['processed_review'],df['sen
```

In [58]:
```
# Vectorizing X_train and X_test to 2D tensor
tokenizer = Tokenizer(nb_words=20000)
tokenizer.fit_on_texts(X_train)

# Converting in to sequences
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=100)
X_test_seq = sequence.pad_sequences(sequences_test, maxlen=100)

# One hot encoding
```

```
y_train_seq = np_utils.to_categorical(y_train, 3)
y_test_seq = np_utils.to_categorical(y_test, 3)
```

In [59]:
```
print("X_train shape: {}, y_train shape: {}".format(X_train_seq.shape, y_train_seq
print("X_test shape: {}, y_test shape: {}".format(X_test_seq.shape, y_test_seq.sha
```

```
X_train shape: (1012, 100), y_train shape: (1012, 3)
X_test shape: (113, 100), y_test shape: (113, 3)
```

In [60]:
```
# Building an LSTM model
lstm_model = Sequential()
lstm_model.add(Embedding(20000, 128))
lstm_model.add(LSTM(128, dropout=0.2))
lstm_model.add(Dense(3))
lstm_model.add(Activation('softmax'))
lstm_model.summary()

# Compiling model
lstm_model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accurac
lstm_model.fit(X_train_seq, y_train_seq, batch_size=32,epochs=5)

score = lstm_model.evaluate(X_test_seq, y_test_seq, batch_size=32)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, None, 128)         2560000

 lstm (LSTM)                 (None, 128)               131584

 dense (Dense)               (None, 3)                 387

 activation (Activation)     (None, 3)                 0

=================================================================
Total params: 2,691,971
Trainable params: 2,691,971
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
32/32 [==============================] - 14s 327ms/step - loss: 0.6424 - accuracy:
0.4625
Epoch 2/5
32/32 [==============================] - 10s 325ms/step - loss: 0.4879 - accuracy:
0.7075
Epoch 3/5
32/32 [==============================] - 10s 321ms/step - loss: 0.2779 - accuracy:
0.8458
Epoch 4/5
32/32 [==============================] - 10s 323ms/step - loss: 0.1869 - accuracy:
0.9328
Epoch 5/5
32/32 [==============================] - 10s 323ms/step - loss: 0.0927 - accuracy:
0.9536
4/4 [==============================] - 1s 34ms/step - loss: 0.1602 - accuracy: 0.9
292
Test loss:  0.16015471518039703
Test accuracy:  0.9292035102844238
```

LSTM with word2vec embedding

In [61]:
```python
# Loading prebuilt Word2Vector model
w2v = Word2Vec.load("w2v1")

# Getting Word2Vector embedding matrix
embedding_matrix = w2v.wv.syn0
```

In [62]:
```python
print("Embedding matrix: ", embedding_matrix.shape)
```

```
Embedding matrix:  (414, 300)
```

In [63]:
```python
top_words = embedding_matrix.shape[0]

# Vectorizing X_train and X_test to 2D tensor
tokenizer = Tokenizer(nb_words=top_words)
tokenizer.fit_on_texts(X_train)

# Converting in to sequences
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=100)
X_test_seq = sequence.pad_sequences(sequences_test, maxlen=100)

# One hot encoding
y_train_seq = np_utils.to_categorical(y_train, 3)
y_test_seq = np_utils.to_categorical(y_test, 3)
```

In [64]:
```python
print("X_train shape: {}, y_train shape: {}".format(X_train_seq.shape, y_train_seq
print("X_test shape: {}, y_test shape: {}".format(X_test_seq.shape, y_test_seq.sha
```

```
X_train shape: (1012, 100), y_train shape: (1012, 3)
X_test shape: (113, 100), y_test shape: (113, 3)
```

In [65]:
```python
embedding_layer = Embedding(embedding_matrix.shape[0], embedding_matrix.shape[1], w

# Constructing LSTM with embedding model
lstm_model2 = Sequential()
lstm_model2.add(embedding_layer)
lstm_model2.add(LSTM(128, dropout=0.2))
lstm_model2.add(Dense(3))
lstm_model2.add(Activation('softmax'))
lstm_model2.summary()

# Compiling model
lstm_model2.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accura
lstm_model2.fit(X_train_seq, y_train_seq, batch_size=32, epochs=5)

score = lstm_model2.evaluate(X_test_seq, y_test_seq, batch_size=32)
print("Test loss: ", score[0])
print("Test accuracy: ", score[1])
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 embedding_1 (Embedding)     (None, None, 300)         124200    
                                                                 
 lstm_1 (LSTM)               (None, 128)               219648    
                                                                 
 dense_1 (Dense)             (None, 3)                 387       
                                                                 
 activation_1 (Activation)   (None, 3)                 0         
                                                                 
=================================================================
Total params: 344,235
Trainable params: 344,235
Non-trainable params: 0
_________________________________________________________________
Epoch 1/5
32/32 [==============================] - 17s 428ms/step - loss: 0.6337 - accuracy: 
0.4012
Epoch 2/5
32/32 [==============================] - 13s 398ms/step - loss: 0.4993 - accuracy: 
0.6907
Epoch 3/5
32/32 [==============================] - 15s 456ms/step - loss: 0.3762 - accuracy: 
0.7678
Epoch 4/5
32/32 [==============================] - 13s 401ms/step - loss: 0.2538 - accuracy: 
0.8607
Epoch 5/5
32/32 [==============================] - 14s 434ms/step - loss: 0.1720 - accuracy: 
0.9130
4/4 [==============================] - 1s 50ms/step - loss: 0.2189 - accuracy: 0.8
673
Test loss:  0.218874201178550 72
Test accuracy:  0.8672566413879395
```

Optional tasks

Clustering similar reviews

1. Cluster similar reviews
2. Perform topic modelling using LDA & NMF

Topic modelling

Using LDA

In [66]:
```
doc_complete = train_data_psd["processed_review"].tolist()
doc_clean = [cleanText(doc).split() for doc in doc_complete]
```

In [67]:
```
dictionary = corpora.Dictionary(doc_clean)
print(dictionary)
```

```
Dictionary(3416 unique tokens: ['able', 'access', 'accomplish', 'ad', 'add']...)
```

In [68]:
```
doc_term_matrix = [dictionary.doc2bow(doc) for doc in doc_clean]
```

In [69]:
```
from gensim.models import LdaModel
```

```
n_topics = 9
ldamodel = LdaModel(doc_term_matrix, num_topics=n_topics, id2word=dictionary, pass
topics = ldamodel.show_topics()
for topic in topics:
  print(topic, "\n")
```

```
(0, '0.040*"kindle" + 0.022*"battery" + 0.022*"charge" + 0.021*"read" + 0.016*"las
t" + 0.016*"light" + 0.015*"would" + 0.015*"easy" + 0.014*"life" + 0.014*"much"')

(1, '0.044*"tablet" + 0.042*"great" + 0.038*"use" + 0.035*"good" + 0.030*"price" +
0.028*"easy" + 0.027*"product" + 0.024*"work" + 0.015*"amazon" + 0.015*"need"')

(2, '0.032*"kindle" + 0.018*"book" + 0.018*"fire" + 0.018*"read" + 0.018*"screen"
+ 0.015*"use" + 0.013*"tablet" + 0.012*"one" + 0.012*"like" + 0.012*"amazon"')

(3, '0.050*"great" + 0.041*"tablet" + 0.041*"read" + 0.024*"book" + 0.024*"price"
+ 0.017*"size" + 0.017*"use" + 0.015*"screen" + 0.015*"get" + 0.015*"perfect"')

(4, '0.030*"alexa" + 0.029*"echo" + 0.028*"great" + 0.023*"light" + 0.022*"home" +
0.019*"music" + 0.018*"smart" + 0.017*"love" + 0.017*"plus" + 0.015*"use"')

(5, '0.052*"love" + 0.048*"tablet" + 0.047*"kid" + 0.037*"old" + 0.037*"year" + 0.
029*"buy" + 0.026*"game" + 0.022*"play" + 0.021*"great" + 0.020*"use"')

(6, '0.057*"buy" + 0.051*"love" + 0.038*"gift" + 0.037*"one" + 0.028*"get" + 0.022
*"recommend" + 0.021*"purchase" + 0.021*"would" + 0.019*"great" + 0.018*"produc
t"')

(7, '0.026*"love" + 0.022*"use" + 0.020*"echo" + 0.018*"sound" + 0.017*"one" + 0.0
16*"tap" + 0.014*"alexa" + 0.013*"music" + 0.013*"buy" + 0.012*"speaker"')

(8, '0.044*"show" + 0.036*"echo" + 0.024*"video" + 0.020*"screen" + 0.019*"amazon"
+ 0.016*"see" + 0.015*"alexa" + 0.015*"like" + 0.014*"device" + 0.013*"use"')
```

In [70]:

```
word_dict = {}
for i in range(n_topics):
  words = ldamodel.show_topic(i, topn=20)
  word_dict["Topic #" + "{}".format(i)] = [i[0] for i in words]
```

In [71]:

```
topics_df = pd.DataFrame(word_dict)
topics_df
```

Out[71]:

	Topic #0	Topic #1	Topic #2	Topic #3	Topic #4	Topic #5	Topic #6	Topic #7	Topic #8
0	kindle	tablet	kindle	great	alexa	love	buy	love	show
1	battery	great	book	tablet	echo	tablet	love	use	echo
2	charge	use	fire	read	great	kid	gift	echo	video
3	read	good	read	book	light	old	one	sound	screen
4	last	price	screen	price	home	year	get	one	amazon
5	light	easy	use	size	music	buy	recommend	tap	see
6	would	product	tablet	use	smart	game	purchase	alexa	alexa
7	easy	work	one	screen	love	play	would	music	like
8	life	amazon	like	get	plus	great	great	buy	device
9	much	need	amazon	perfect	use	use	product	speaker	use
10	long	apps	love	game	set	easy	christmas	get	music
11	well	quality	new	movie	easy	child	best	purchase	great
12	make	want	device	good	amazon	apps	wife	great	play
13	model	would	well	nice	ask	time	tablet	much	sound
14	go	well	small	love	control	son	use	well	well
15	fire	excellent	get	need	fun	grandson	fire	house	also
16	book	device	buy	work	product	daughter	happy	work	call
17	like	love	light	easy	device	learn	daughter	wifi	love
18	buy	set	size	kindle	work	granddaughter	son	day	camera
19	time	like	much	watch	turn	purchase	family	easy	dot