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.

In [ ]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns


import nltk
from nltk.corpus import stopwords
from nltk.classify import SklearnClassifier

from wordcloud import WordCloud,STOPWORDS
import matplotlib.pyplot as plt
%matplotlib inline
from subprocess import check_output


from sklearn.feature_extraction.text import CountVectorizer
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
```

In [ ]:

```
train = pd.read_csv("train_data.csv")
test = pd.read_csv("test_data.csv")
```

In [ ]:

```
train.head()
```

In [ ]:

```
test.head()
```

In [ ]:

```
train.count()
```

In [ ]:

```
#train = train.append(test, ignore_index=True)
test.count()
```

In [ ]:

```
train.duplicated().sum()
```

There are 58 duplicates, let's drop the duplicate values

In [ ]:

```
train = train.drop_duplicates().reset_index(drop=True)
```

In [ ]:

```
train.info()
```

In [ ]:

```
train.dtypes
```

In [ ]:

```
train.describe()
```

Lets Visualize with the class imbalance thing !

Basic EDA of trainig data set

In [ ]:

```
sns.countplot(y=train.sentiment);
```

In [ ]:

```
sns.countplot( train['sentiment']);
```

Class Imbalance Problem

In [ ]:

```
train.sentiment.value_counts()
```

In [ ]:

```
# NA data
train.isnull().sum()
```

In [ ]:

```
test.isnull().sum()
```

We should rename the column to avoid errors

In [ ]:

```
train.columns
```

In [ ]:

```
train.rename(columns = {'reviews.text':'reviews_text', 'reviews.title':'reviews_
title','reviews.date':'reviews_date'}, inplace = True)
```

In [ ]:

```
train.columns
```

In [ ]:

```
train['sentiment'].value_counts().plot(kind='pie', autopct= '%1.0f%%')
```

Here I am dropping the Neutral Sentiment as My goal is to work on differentiate the positive and negative

In [ ]:

```
train = train[train.sentiment != "Neutral"]
```

Now we are ready for a WordCloud visualization which shows only the most emphatic words of the Positive and Negative sentiment.

In [ ]:

```
train_pos = train[ train['sentiment'] == 'Positive']
train_pos = train_pos['reviews_text']
train_neg = train[ train['sentiment'] == 'Negative']
train_neg = train_neg['reviews_text']

def wordcloud_draw(data, color = 'black'):
    words = ' '.join(data)
    cleaned_word = " ".join([word for word in words.split()
                             if 'http' not in word
                                 and not word.startswith('@')
                                 and not word.startswith('#')
                                 and word != 'RT'
                             ])
    wordcloud = WordCloud(stopwords=STOPWORDS,
                      background_color=color,
                      width=2500,
                      height=2000
                     ).generate(cleaned_word)
    plt.figure(1,figsize=(13, 13))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()

print("Positive words")
wordcloud_draw(train_pos,'white')
print("Negative words")
wordcloud_draw(train_neg)
```

Data Preprocessing Part

Stopword Removal using NLTK

After the vizualization, we need to remove the hashtags, mentions, links and stopwords from the training set.

Stop Word: Stop Words are words which do not contain important significance to be used in Search Queries. Usually these words are filtered out from search queries because they return vast amount of unnecessary information. (the, for, this etc.)

In [ ]:

```
def remove_non_ascii(words):
    """Remove non-ASCII characters from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'ignore')
.decode('utf-8', 'ignore')
        new_words.append(new_word)
    return new_words

def to_lowercase(words):
    """Convert all characters to lowercase from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    """Remove punctuation from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def remove_numbers(words):
    """Remove all interger occurrences in list of tokenized words with textual r
epresentation"""
    new_words = []
    for word in words:
        new_word = re.sub("\d+", "", word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words

def stem_words(words):
    """Stem words in list of tokenized words"""
    stemmer = LancasterStemmer()
    stems = []
    for word in words:
        stem = stemmer.stem(word)
        stems.append(stem)
    return stems

def lemmatize_verbs(words):
    """Lemmatize verbs in list of tokenized words"""
    lemmatizer = WordNetLemmatizer()
    lemmas = []
    for word in words:
        lemma = lemmatizer.lemmatize(word, pos='v')
```

```
        lemmas.append(lemma)
    return lemmas

def normalize(words):
    words = remove_non_ascii(words)
    words = to_lowercase(words)
    words = remove_punctuation(words)
    words = remove_numbers(words)
#     words = remove_stopwords(words)
    return words
```

In [ ]:

```
# First step - tokenizing phrases
train['reviews_text'] = train['reviews_text'].apply(nltk.word_tokenize)

# Second step - passing through prep functions
#train['reviews_text'] = train['reviews_text'].apply(normalize)
train['reviews_text'].head()
```

Let's visualize the most common keywords

In [ ]:

```
# Most common keywords
plt.figure(figsize=(10,6))
sns.countplot(y=train.reviews_title, order = train.reviews_title.value_counts().
iloc[:20].index)
plt.title('Top 20 keywords')
plt.show()
# train.keyword.value_counts().head(10)
```

In [ ]:

```

```