Final Project Data Mining

Jobaidul Alam Boni

Introduction

Text analysis is one of the great achievements of Artificial Intelligence and in last few years it has become an important and most popular from of data mining. Text data analysis has become an important part in every analytical fields from social media analytics to fraud detection to spam identification even to the analysing traffic congestion to disaster analysis. In this project we will try to understand how different data mining techniques can help to classify text of a big set of data.

Board Game Geek Rating Predictor

Board Game Geek is an online platform that discusses content, idea and thoughts of different board games around the world. It has more than two million registered users and one of the largest hub of this kind. BGG includes reviews, ratings, images, play-aids, translations, and session reports from board game enthusiast around the world. Based on all those discussion we will try to predict the rating of the games based on thier reviews and thoughts. We have collected the data of 15 millions reviews from https://www.kaggle.com/jvanelteren/boardgamegeek-reviews (https://www.kaggle.com/jvanelteren/boardgamegeek-reviews). We will try different data minining techniques and will try to find out which techniques will provide the best

Data Description

Data consist of two sets of data set we will read each data set and will try to understand different data set

In [1]:
```
import pandas as pd
import numpy as np
df=pd.read_csv('E:/data mining/bgg-15m-reviews.csv')
df
```

Out[1]:

	Unnamed: 0	user	rating	comment	ID	name
0	0	Torsten	10.0	NaN	30549	Pandemic
1	1	mitnachtKAUBO-I	10.0	Hands down my favorite new game of BGG CON 200...	30549	Pandemic
2	2	avlawn	10.0	I tend to either love or easily tire of co-op ...	30549	Pandemic
3	3	Mike Mayer	10.0	NaN	30549	Pandemic
4	4	Mease19	10.0	This is an amazing co-op game. I play mostly ...	30549	Pandemic
...	...	...	...	...	...	...
15823264	15823264	Fafhrd65	8.0	Turn based preview looks very promising. The g...	281515	Company of Heroes
15823265	15823265	PlatinumOh	8.0	KS	281515	Company of Heroes
15823266	15823266	BunkerBill	7.0	NaN	281515	Company of Heroes
15823267	15823267	Hattori Hanzo	6.0	NaN	281515	Company of Heroes
15823268	15823268	Richie2000	1.0	NaN	281515	Company of Heroes

15823269 rows × 6 columns

In [2]:
```
info=pd.read_csv('E:/data mining/games_detailed_info.csv')
info
```

Out[2]:

	Unnamed: 0	type	id	thumbnail	i
0	0	boardgame	30549	https://cf.geekdo-images.com/thumb/img/HEKrtpT...	https://cf.ge images.com/original/img/
1	1	boardgame	822	https://cf.geekdo-images.com/thumb/img/kqE4YJS...	https://cf.ge images.com/original/img/o
2	2	boardgame	13	https://cf.geekdo-images.com/thumb/img/g8LvJsd...	https://cf.ge images.com/original/img/A
3	3	boardgame	68448	https://cf.geekdo-images.com/thumb/img/Grz-qM9...	https://cf.ge images.com/original/img/3I
4	4	boardgame	36218	https://cf.geekdo-images.com/thumb/img/iPITR5c...	https://cf.ge images.com/original/img/oN
...	...	...	...	...	
19225	19225	boardgame	246345	https://cf.geekdo-images.com/thumb/img/EMAGbmG...	https://cf.ge images.com/original/img/L
19226	19226	boardgame	195623	https://cf.geekdo-images.com/thumb/img/y2EuGgf...	https://cf.ge images.com/original/img/o
19227	19227	boardgame	235943	https://cf.geekdo-images.com/thumb/img/LUyoD4j...	https://cf.ge images.com/original/img/o
19228	19228	boardgame	284862	https://cf.geekdo-images.com/thumb/img/b_ckSKO...	https://cf.ge images.com/original/img/V
19229	19229	boardgame	281515	https://cf.geekdo-images.com/thumb/img/QnvFHRq...	https://cf.ge images.com/original/img/p

19230 rows × 56 columns

We have 15823269 reviews of 19230 games. But there are some ratings that does not have any comments or reviews. So we will remove those comments.

In [3]:
```
df1=df[~df.comment.str.contains("NaN",na=True)]
df1
```

Out[3]:

	Unnamed: 0	user	rating	comment	ID	name
1	1	mitnachtKAUBO-I	10.0	Hands down my favorite new game of BGG CON 200...	30549	Pandemic
2	2	avlawn	10.0	I tend to either love or easily tire of co-op ...	30549	Pandemic
4	4	Mease19	10.0	This is an amazing co-op game. I play mostly ...	30549	Pandemic
5	5	cfarrell	10.0	Hey! I can finally rate this game I've been pl...	30549	Pandemic
8	8	gregd	10.0	Love it- great fun with my son. 2 plays so far...	30549	Pandemic
...	...	...	...	...	...	...
15823242	15823242	ChiefMe	10.0	KS Collector's Bundle with a friend of mine	281515	Company of Heroes
15823247	15823247	Mukaz	10.0	Belekokio Gerumo...	281515	Company of Heroes
15823253	15823253	jpaquila	10.0	Excelente!! lo mejor que probé.	281515	Company of Heroes
15823264	15823264	Fafhrd65	8.0	Turn based preview looks very promising. The g...	281515	Company of Heroes
15823265	15823265	PlatinumOh	8.0	KS	281515	Company of Heroes

2995022 rows × 6 columns

After removing the empty rows we have 2995022 reviews and ratings. Next we will shuffle the data so that we have a good homogenous mix of data

In [4]:
```
from sklearn.utils import shuffle
df1=shuffle(df1,random_state=0)
df1
```

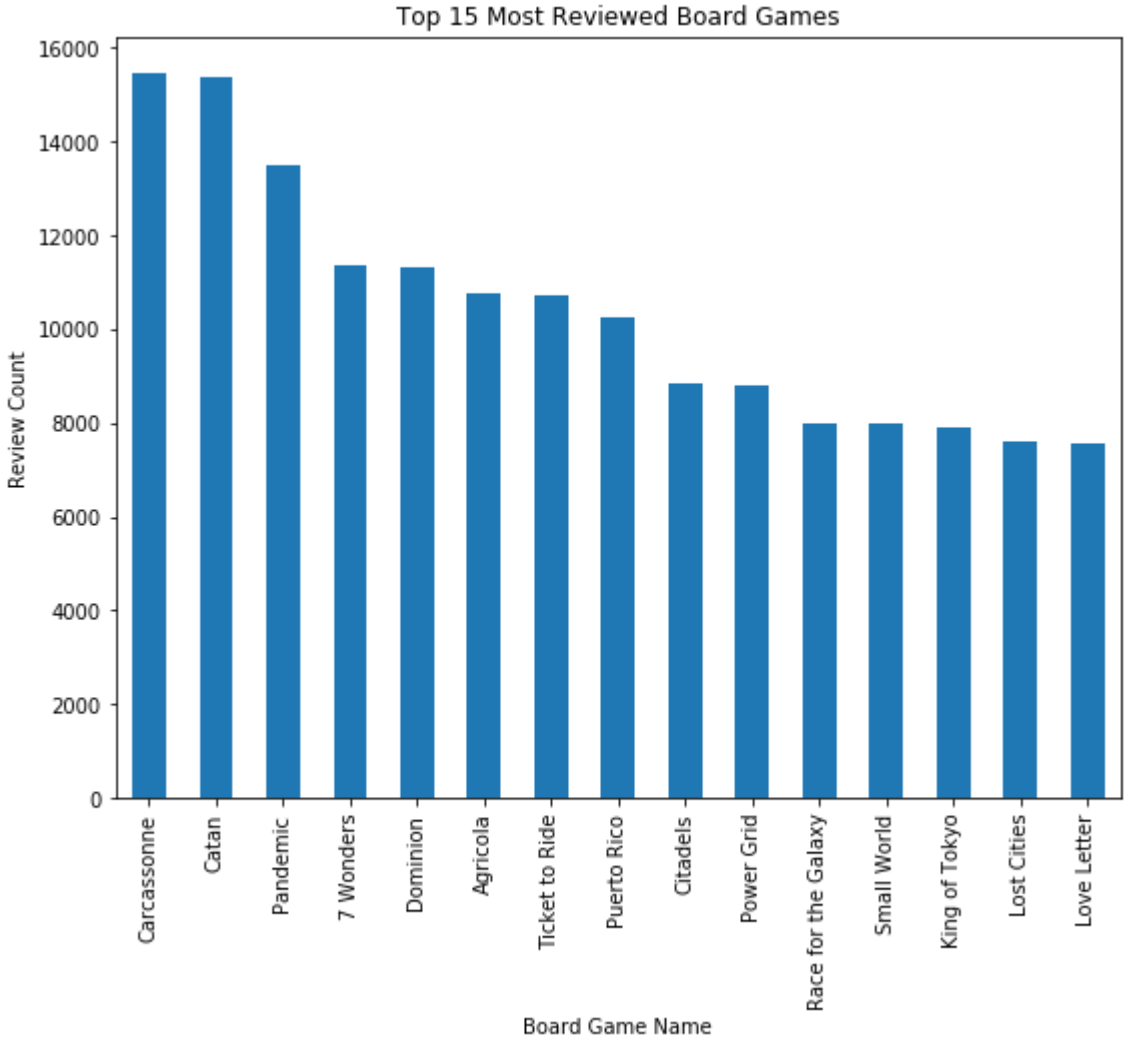
Out[4]:

	Unnamed: 0	user	rating	comment	ID	name
40545	40545	chessduffer	10.0	Believe the Hype, This game is completely awes...	31260	Agricola
11354135	11354135	seaapple	6.5	Fun card game that plays like a board game. F...	72	Verräter
866088	866088	ingmen	9.0	Light family game with a deep gameplay.	230802	Azul
5392690	5392690	Phoenixio	9.0	Calgary	176396	Quadropolis
6860438	6860438	Sops	8.0	When this is great, this is really really grea...	27708	1960: The Making of the President
...	...	...	...	...	...	...
12793936	12793936	_The_Inquiry_	3.0	Prior to 2020: 1 play	22889	Pictureka!
6256575	6256575	Khexhu	5.0	Prefer the 1942 edition greatly over this one.	98	Axis & Allies
12637155	12637155	billymoustache	8.0	Solo'd	245961	Fleet: The Dice Game
9142174	9142174	R3DSH1FT	5.0	Once the novelty wears off there's not a whole...	72321	The Networks
1979742	1979742	wmshub	7.0	Nice two player card game. Reminds me a bit of...	54043	Jaipur

2995022 rows × 6 columns

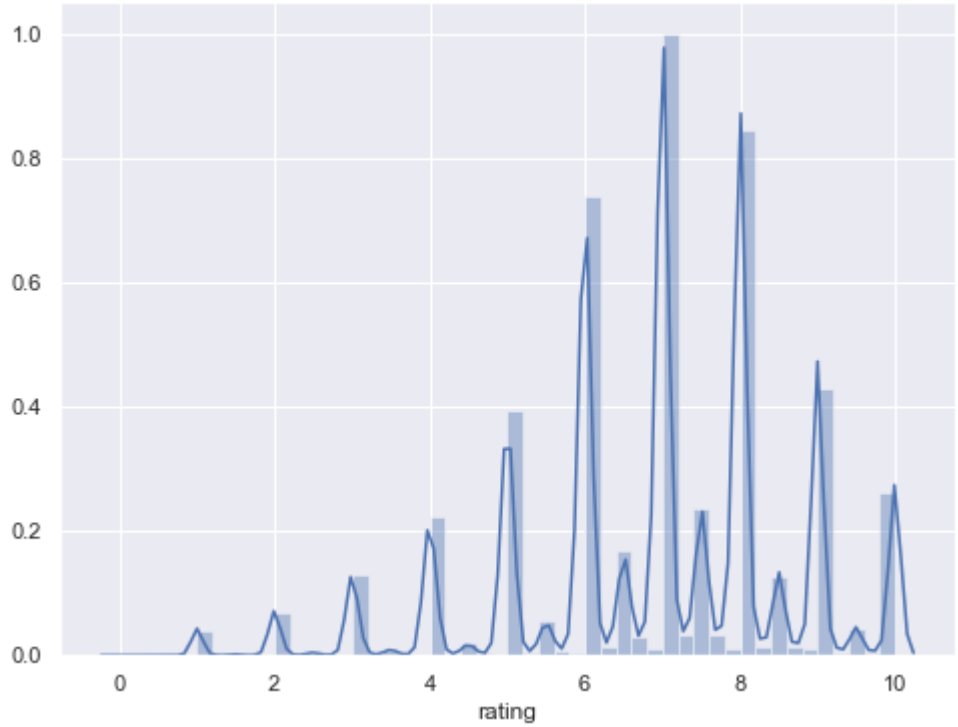
In [5]:
```
import matplotlib.pyplot as plt
```

In [6]:
```
plt.figure(figsize=(9, 7))
df.loc[df['comment'].notna()]['name'].value_counts()[:15].plot(kind='bar')
plt.xlabel('Board Game Name')
plt.ylabel('Review Count')
plt.title('Top 15 Most Reviewed Board Games')
plt.show()
```



In [7]:
```
import seaborn as sns
plt.figure(figsize=(8, 6))
sns.set(color_codes="True")
sns.distplot(df1["rating"])
```

Out[7]: <matplotlib.axes._subplots.AxesSubplot at 0x1c483855308>



From the two figure we can see the most reviewed board game are- Carcassonne, Catan, Pandemic, 7 Wonders and Dominion. 2nd figure shows that the most common ratings are in between 6 and 9

Next we will divide the data set as follows- Train data set=60% Development data set =20% Test Data set=20%

In [8]:
```
print('Train Data Set ')
df1_train=df1[:1797013]
df1_train
```

```
Train Data Set
```

Out[8]:

	Unnamed: 0	user	rating	comment	ID	name
40545	40545	chessduffer	10.0	Believe the Hype, This game is completely awes...	31260	Agricola
11354135	11354135	seaapple	6.5	Fun card game that plays like a board game. F...	72	Verräter
866088	866088	ingmen	9.0	Light family game with a deep gameplay.	230802	Azul
5392690	5392690	Phoenixio	9.0	Calgary	176396	Quadropolis
6860438	6860438	Sops	8.0	When this is great, this is really really grea...	27708	1960: The Making of the President
...	...	...	...	...	...	...
11507198	11507198	ravenme	8.0	Good game, one of the best worker placement ga...	173442	Empires: Age of Discovery
4330642	4330642	MikeTuna	8.0	This is a game that I feel like I'm trying rea...	154203	Imperial Settlers
7447203	7447203	gumbomasta	9.3	What a wild and wooly experience.	105551	Archipelago
361422	361422	goldeneyeace	9.0	A deceptively simple 2-player game, which expe...	54043	Jaipur
1637414	1637414	h_zeera	7.0	The teach is a bit long, but the theme and mec...	158899	Colt Express

1797013 rows × 6 columns

In [9]:
```
print('Development Data Set')
df1_dev=df1[1797013:2396017]
df1_dev
```

```
Development Data Set
```

Out[9]:

	Unnamed: 0	user	rating	comment	ID	name
3946069	3946069	bigmac33070	9.5	Just epic in its scope, which I adore on so ma...	12493	Twilight Imperium (Third Edition)
7907007	7907007	pavin	6.0	Jeu du "trou-de-cul"	929	The Great Dalmuti
9476696	9476696	JohnRayJr	6.0	Middle Earth Quest seems to follow a recent (?...	31563	Middle-Earth Quest
8703813	8703813	ansonvergens	9.0	Greek Edition	194880	Dream Home
6732511	6732511	fraludico	7.0	Italian edition.	140	Pit
...	...	...	...	...	...	...
14915469	14915469	stefanmiker	6.8	A Great Boardgame about the American Civil War...	14701	The Price of Freedom: The American Civil War 1...
10064164	10064164	alexgrant	7.5	A little unsure after the first game; enjoyabl...	141517	A Study in Emerald
10087684	10087684	Roarket	7.0	Components a little fiddly to setup at start	217085	Unearth
15816828	15816828	David Ells	8.0	Kickstarter!	171500	El Dorado Canyon
5263014	5263014	denngeki	6.0	2-4/4	2653	Survive: Escape from Atlantis!

599004 rows × 6 columns

In [10]:
```
print('Test Data Set')
df1_test=df1[2396017:]
df1_test
```

```
Test Data Set
```

Out[10]:

	Unnamed: 0	user	rating	comment	ID	name
10257598	10257598	Large Deviator	9.0	For details, look at my review: http://www.boa...	43022	Yomi
13993328	13993328	oskari	5.0	A boring deduction game. Way too little hidden...	756	Black Vienna
13246020	13246020	zonk67	9.0	Update: I stand corrected. 2nd edition is the ...	221769	Bios: Megafauna (Second Edition)
15078008	15078008	Kaixo	4.0	Mass market nonsense. Theme implementation is...	232944	Where in the World is Carmen Sandiego? Card Game
3471856	3471856	alarii	2.0	Solo Rating !	167791	Terraforming Mars
...	...	...	...	...	...	...
12793936	12793936	_The_Inquiry_	3.0	Prior to 2020: 1 play	22889	Pictureka!
6256575	6256575	Khexhu	5.0	Prefer the 1942 edition greatly over this one.	98	Axis & Allies
12637155	12637155	billymoustache	8.0	Solo'd	245961	Fleet: The Dice Game
9142174	9142174	R3DSH1FT	5.0	Once the novelty wears off there's not a whole...	72321	The Networks
1979742	1979742	wmshub	7.0	Nice two player card game. Reminds me a bit of...	54043	Jaipur

599005 rows × 6 columns

Data Pre-processing

We will do multiple steps to preprocess the text data.

Lowercasing: It is one of effective prepocessing way of the text data. It helps to provide consistent data for different reviews and will help to provide more accurate data set.

Stop words removal: Stop words is a set of of common words that are very common in almost all type of sentences. One of the example of stop words are- “a”, “the”, “is”, “are” and etc. Stop words often provide misleading results and is a very important tool for text data analysis.

Remove Punctuations: Removing different punctuations i.e. http//, @, different symbols(",',?,!) will also provide more accurate results

Tokenization: Tokenization is the process of tokenizing or splitting a string, text into a list of tokens. One can think of token as parts like a word is a token in a sentence, and a sentence is a token in a paragraph. It helps to calculate the frequency of words

In [11]:
```
pip install wordcloud
```

```
Requirement already satisfied: wordcloud in c:\users\jxb5105\appdata\local\co
ntinuum\anaconda3\lib\site-packages (1.8.1)
Requirement already satisfied: matplotlib in c:\users\jxb5105\appdata\local\c
ontinuum\anaconda3\lib\site-packages (from wordcloud) (3.1.1)
Requirement already satisfied: pillow in c:\users\jxb5105\appdata\local\conti
nuum\anaconda3\lib\site-packages (from wordcloud) (6.2.0)
Requirement already satisfied: numpy>=1.6.1 in c:\users\jxb5105\appdata\local
\continuum\anaconda3\lib\site-packages (from wordcloud) (1.16.5)
Requirement already satisfied: cycler>=0.10 in c:\users\jxb5105\appdata\local
\continuum\anaconda3\lib\site-packages (from matplotlib->wordcloud) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\jxb5105\appdata
\local\continuum\anaconda3\lib\site-packages (from matplotlib->wordcloud) (1.
1.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in
c:\users\jxb5105\appdata\local\continuum\anaconda3\lib\site-packages (from ma
tplotlib->wordcloud) (2.4.2)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\jxb5105\appda
ta\local\continuum\anaconda3\lib\site-packages (from matplotlib->wordcloud)
(2.8.0)
Requirement already satisfied: six in c:\users\jxb5105\appdata\local\continuu
m\anaconda3\lib\site-packages (from cycler>=0.10->matplotlib->wordcloud) (1.1
2.0)
Requirement already satisfied: setuptools in c:\users\jxb5105\appdata\local\c
ontinuum\anaconda3\lib\site-packages (from kiwisolver>=1.0.1->matplotlib->wor
dcloud) (41.4.0)
Note: you may need to restart the kernel to use updated packages.
```

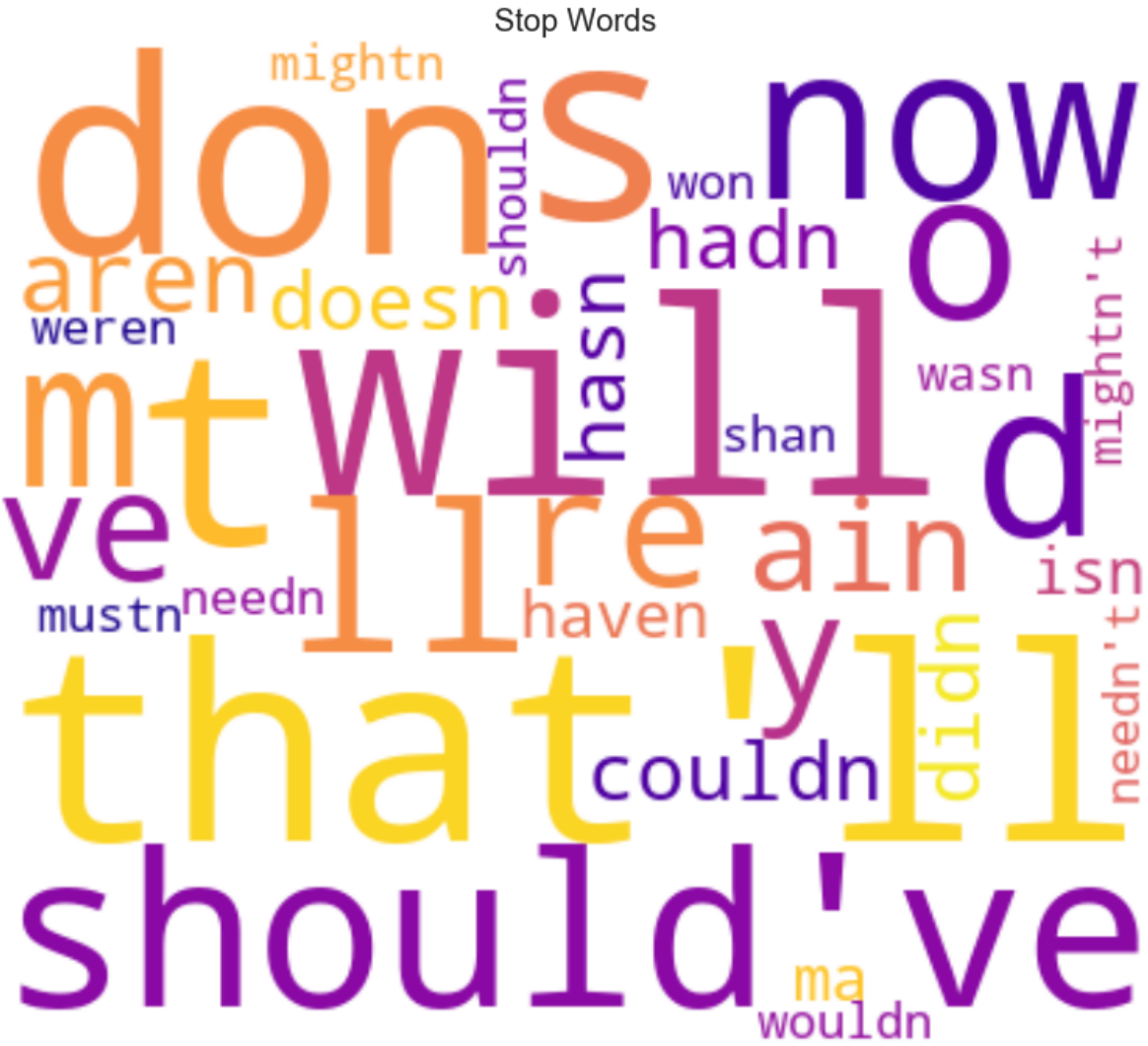
In [12]:
```
import nltk
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from wordcloud import WordCloud,STOPWORDS
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import word_tokenize,sent_tokenize
from bs4 import BeautifulSoup
import re,string,unicodedata
from sklearn.metrics import classification_report,confusion_matrix,accuracy_sc
ore
from sklearn.model_selection import train_test_split
from string import punctuation

import matplotlib.pyplot as plt
import seaborn as sns
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\jxb5105\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[12]: True

In [21]:
```
stop_word = stopwords.words('english')
new=' '.join([str(elem) for elem in stop_word])
from wordcloud import WordCloud
import matplotlib.pyplot as plt
wc = WordCloud(width=400, height=350,colormap='plasma',background_color='whit
e').generate(new)
plt.figure(figsize=(15,12))
plt.imshow(wc, interpolation='bilinear')
plt.title('Stop Words', fontsize=20)
plt.axis('off');
plt.show()
```



In [14]:
```
def clean(text):
    text = re.sub(r'http\S+', " ", text)
    text = re.sub(r'@\w+',' ',text)
    text = re.sub(r'#\w+', ' ', text)
    text = re.sub(r'\d+', ' ', text)
    text = re.sub('r<.*?>',' ', text)
    text = text.split()
    text = " ".join([word for word in text if not word in stop_word])


    return text
```

In [15]:
```
train=df1_train['comment'].str.lower()
train
```

Out[15]:
```
40545       believe the hype, this game is completely awes...
11354135    fun card game that plays like a board game.  f...
866088                light family game with a deep gameplay.
5392690                                                calgary
6860438     when this is great, this is really really grea...
                                  ...
11507198    good game, one of the best worker placement ga...
4330642     this is a game that i feel like i'm trying rea...
7447203                     what a wild and wooly experience.
361422      a deceptively simple 2-player game, which expe...
1637414     the teach is a bit long, but the theme and mec...
Name: comment, Length: 1797013, dtype: object
```

In [16]:
```
train_new=train.apply(lambda x:clean(x))
train_new
```

Out[16]:
```
40545       believe hype, game completely awesome. reminds...
11354135    fun card game plays like board game. feels lik...
866088                       light family game deep gameplay.
5392690                                                calgary
6860438     great, really really great! card mechanic to-i...
                                  ...
11507198    good game, one best worker placement games the...
4330642     game feel like i'm trying really hard like. th...
7447203                                 wild wooly experience.
361422      deceptively simple -player game, expertly side...
1637414     teach bit long, theme mechanics pair well usua...
Name: comment, Length: 1797013, dtype: object
```

In [17]:
```
def tokenize_remove_punctuations(x):                # Tokenize the text into t
okens and count their frequencies
    wordfreq = {}
    for txt in x:
        tokens = nltk.RegexpTokenizer(r"\w+").tokenize(txt)     # we could use
.lower().split()
        for token in tokens:                                    # and remove t
he punctuations
            if token not in wordfreq.keys():
                wordfreq[token] = 1
            else:
                wordfreq[token] += 1

    return wordfreq
def sort_words_freq(wordfreq_X):                    # Sort words frequency in
 reverse order
    wordfreq_sorted = dict(sorted(wordfreq_X.items(), key=lambda x: x[1], reve
rse=True))
    return wordfreq_sorted
def delete_threshold(wordfreq_sorted):
    delete = []
    for key, val in wordfreq_sorted.items():
        if val < 1000 :
            delete.append(key)

    for i in delete:
        del wordfreq_sorted[i]
    return   wordfreq_sorted
```

In [18]:
```
wordfreq_X = tokenize_remove_punctuations(train_new)
wordfreq_X2 = sort_words_freq(wordfreq_X)
wordfreq_X3=delete_threshold(wordfreq_X2)
```

In [24]:
```
words=[k  for  k in  wordfreq_X3.keys()]
word=words[:50]
new=' '.join([str(elem) for elem in word])
from wordcloud import WordCloud
import matplotlib.pyplot as plt
wc = WordCloud(width=400, height=350,colormap='plasma',background_color='whit
e').generate(new)
plt.figure(figsize=(15,12))
plt.imshow(wc, interpolation='bilinear')
plt.title('Common Words', fontsize=20)
plt.axis('off');
plt.show()
```

In [30]:
```
x_train=train_new
y_train=df1_train['rating']
```

In [25]:
```
dev=df1_dev['comment'].str.lower()
dev_new=dev.apply(lambda x:clean(x))
x_dev=dev_new
y_dev=df1_dev['rating']
print(x_dev)
print(y_dev)
```

```
3946069     epic scope, adore many levels. really enjoy ra...
7907007                                  jeu du "trou-de-cul"
9476696     middle earth quest seems follow recent (?) tre...
8703813                                        greek edition
6732511                                     italian edition.
                                  ...
14915469    great boardgame american civil war!!!incredibl...
10064164    little unsure first game; enjoyable however. d...
10087684                 components little fiddly setup start
15816828                                       kickstarter!
5263014                                                 - /
Name: comment, Length: 599004, dtype: object
3946069     9.5
7907007     6.0
9476696     6.0
8703813     9.0
6732511     7.0
           ...
14915469    6.8
10064164    7.5
10087684    7.0
15816828    8.0
5263014     6.0
Name: rating, Length: 599004, dtype: float64
```

In [26]:
```
test=df1_test['comment'].str.lower()
test_new=test.apply(lambda x:clean(x))
x_test=test_new
y_test=df1_test['rating']
print(x_test)
print(y_test)
```

```
10257598                                 details, look review:
13993328    boring deduction game. way little hidden infor...
13246020    update: stand corrected. nd edition better ove...
15078008    mass market nonsense. theme implementation dec...
3471856                                          solo rating !
                                ...
12793936                                          prior : play
6256575                          prefer edition greatly one.
12637155                                                solo'd
9142174     novelty wears there's whole lot going games fe...
1979742     nice two player card game. reminds bit traders...
Name: comment, Length: 599005, dtype: object
10257598    9.0
13993328    5.0
13246020    9.0
15078008    4.0
3471856     2.0
           ...
12793936    3.0
6256575     5.0
12637155    8.0
9142174     5.0
1979742     7.0
Name: rating, Length: 599005, dtype: float64
```

Model Selection for Text Classification

Naive Bayes Classifier

Naive Bayes methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of conditional independence between every pair of features given the value of the class variable. Bayes' theorem states the following relationship, given class variable y and dependent feature vector x_1 through x_n, :

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) P(x_1, \dots, x_n \mid y)}{P(x_1, \dots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i | y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i | y),$$

for all i, this relationship is simplified to

$$P(y \mid x_1, \dots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \dots, x_n)}$$

Since $P(x_1, \dots, x_n)$ is constant given the input, we can use the following classification rule:

$$P(y \mid x_1, \dots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

and we can use Maximum A Posteriori (MAP) estimation to estimate $P(y)$ and $P(x_i \mid y)$; the former is then the relative frequency of class y in the training set.

The different naive Bayes classifiers differ mainly by the assumptions they make regarding the distribution of $P(x_i \mid y)$.

Multinomial Naive Bayes

`MultinomialNB` implements the naive Bayes algorithm for multinomially distributed data, and is one of the two classic naive Bayes variants used in text classification (where the data are typically represented as word vector counts, although tf-idf vectors are also known to work well in practice). The distribution is parametrized by vectors $\theta_y = (\theta_{y1}, \ldots, \theta_{yn})$ for each class y, where n is the number of features (in text classification, the size of the vocabulary) and θ_{yi} is the probability $P(x_i \mid y)$ of feature i appearing in a sample belonging to class y.

The parameters θ_y is estimated by a smoothed version of maximum likelihood, i.e. relative frequency counting:

$$\hat{\theta}_{yi} = \frac{N_{yi} + \alpha}{N_y + \alpha n}$$

where $N_{yi} = \sum_{x \in T} x_i$ is the number of times feature i appears in a sample of class y in the training set T, and $N_y = \sum_{i=1}^{n} N_{yi}$ is the total count of all features for class y.

The smoothing priors $\alpha \geq 0$ accounts for features not present in the learning samples and prevents zero probabilities in further computations. Setting $\alpha = 1$ is called Laplace smoothing, while $\alpha < 1$ is called Lidstone smoothing.

Complement Naive Bayes

`ComplementNB` implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets. Specifically, CNB uses statistics from the *complement* of each class to compute the model's weights. The inventors of CNB show empirically that the parameter estimates for CNB are more stable than those for MNB. Further, CNB regularly outperforms MNB (often by a considerable margin) on text classification tasks. The procedure for calculating the weights is as follows:

$$\hat{\theta}_{ci} = \frac{\alpha_i + \sum_{j:y_j \neq c} d_{ij}}{\alpha + \sum_{j:y_j \neq c} \sum_k d_{kj}}$$

$$w_{ci} = \log \hat{\theta}_{ci}$$

$$w_{ci} = \frac{w_{ci}}{\sum_j |w_{cj}|}$$

where the summations are over all documents j not in class c, d_{ij} is either the count or tf-idf value of term i in document j, α_i is a smoothing hyperparameter like that found in MNB, and $\alpha = \sum_i \alpha_i$. The second normalization addresses the tendency for longer documents to dominate parameter estimates in MNB. The classification rule is:

$$\hat{c} = \arg\min_c \sum_i t_i w_{ci}$$

i.e., a document is assigned to the class that is the *poorest* complement match.

Bernoulli Naive Bayes

`BernoulliNB` implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions; i.e., there may be multiple features but each one is assumed to be a binary-valued (Bernoulli, boolean) variable. Therefore, this class requires samples to be represented as binary-valued feature vectors; if handed any other kind of data, a `BernoulliNB` instance may binarize its input (depending on the `binarize` parameter).

The decision rule for Bernoulli naive Bayes is based on

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

which differs from multinomial NB's rule in that it explicitly penalizes the non-occurrence of a feature i that is an indicator for class y, where the multinomial variant would simply ignore a non-occurring feature.

In the case of text classification, word occurrence vectors (rather than word count vectors) may be used to train and use this classifier. `BernoulliNB` might perform better on some datasets, especially those with shorter documents. It is advisable to evaluate both models, if time permits.

In [27]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer,
TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import ComplementNB
from sklearn.naive_bayes import BernoulliNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
from math import sqrt
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split, cross_val_score, cross_v
al_predict
from sklearn.ensemble import VotingClassifier
sns.set(color_codes=True)
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import KFold, cross_val_score, train_test_split
import random
from sklearn.metrics import accuracy_score
from collections import Counter
from sklearn.metrics import accuracy_score
```

At first we will use Multinomial Naive Bayes model with different values of alpha and will check accuracy using development data set

Alpha=1

In [31]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=1)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy1=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n Test accuracy is',(accuracy1))
MSE1=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE1))
```

```
 Test accuracy is 28.264919766812906

 Mean Squared Error 2.8898271130075925
```

Alpha=2

In [25]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=2)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy2=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy2))
MSE2=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE2))
```

```
****Test accuracy is 27.531535682566393

 Mean Squared Error 3.0038230128680277
```

Alpha=3

In [26]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=3)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy3=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy3))
MSE3=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE3))
```

```
****Test accuracy is 27.196646433078914

 Mean Squared Error 3.055039699234062
```

Alpha=0.1

In [27]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=0.1)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy4=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy4))
MSE4=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE4))
```

```
****Test accuracy is 29.870418227591134

 Mean Squared Error 2.6779654226015186
```

Alpha=0.01

In [28]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=0.01)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy5=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy5))
MSE5=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE5))
```

```
****Test accuracy is 30.18777837877543

 Mean Squared Error 2.614611922457947
```

Alpha=0.001

In [29]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=0.001)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy6=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy6))
MSE6=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE6))
```

```
****Test accuracy is 30.17893035772716

 Mean Squared Error 2.637055846037756
```

Highest Accuracy we found using Multinomial Naive Bayes with an alpha =0.01

Now we will check Complement Naive Bayes model

Alpha=1

In [30]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=1)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy7=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy7))
MSE7=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE7))
```

```
****Test accuracy is 29.041375349747245

 Mean Squared Error 4.373144419736763
```

Alpha=2

In [31]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=2)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy8=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy8))
MSE8=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE8))
```

```
****Test accuracy is 29.534360371550108

 Mean Squared Error 4.261423963779874
```

Alpha =3

In [32]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=3)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy9=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy9))
MSE9=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE9))
```

```
****Test accuracy is 29.45122236245501

 Mean Squared Error 4.249492490868175
```

Alpha=0.1

In [33]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=0.1)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy10=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(
100)
print('\n****Test accuracy is',(accuracy10))
MSE10=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE10))
```

```
****Test accuracy is 27.23771460624637

 Mean Squared Error 4.844814058002951
```

Alpha=0.01

In [34]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=0.01)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy11=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(
100)
print('\n****Test accuracy is',(accuracy11))
MSE11=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE11))
```

```
****Test accuracy is 26.889803740876523

 Mean Squared Error 4.952601318188193
```

Alpha=0.001

In [35]:
```
model_cm = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', ComplementNB(alpha=0.001)) ])
model_cm.fit(x_train,y_train.astype('int'))
y_pred=model_cm.predict(x_dev)
accuracy12=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(
100)
print('\n****Test accuracy is',(accuracy12))
MSE12=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE12))
```

```
****Test accuracy is 26.829703975265605

 Mean Squared Error 4.967287363690393
```

Highest Accuracy we found using Complement Naive Bayes with an alpha =2

Now we will check Bernoulli Naive Bayes model

In [38]:
```
model_bb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', BernoulliNB()) ])
model_bb.fit(x_train,y_train.astype('int'))
y_pred=model_bb.predict(x_dev)
accuracy13=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(
100)
print('\n****Test accuracy is',(accuracy13))
MSE13=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE13))
```

```
****Test accuracy is 29.502807994604375

 Mean Squared Error 2.8365186209107116
```

Among all the Naive Bayes model with different hyperparameter we found the best model to be Multinomial Naive Bayes with an alpha =0.01 and accuracy of 30.18777837877543 %

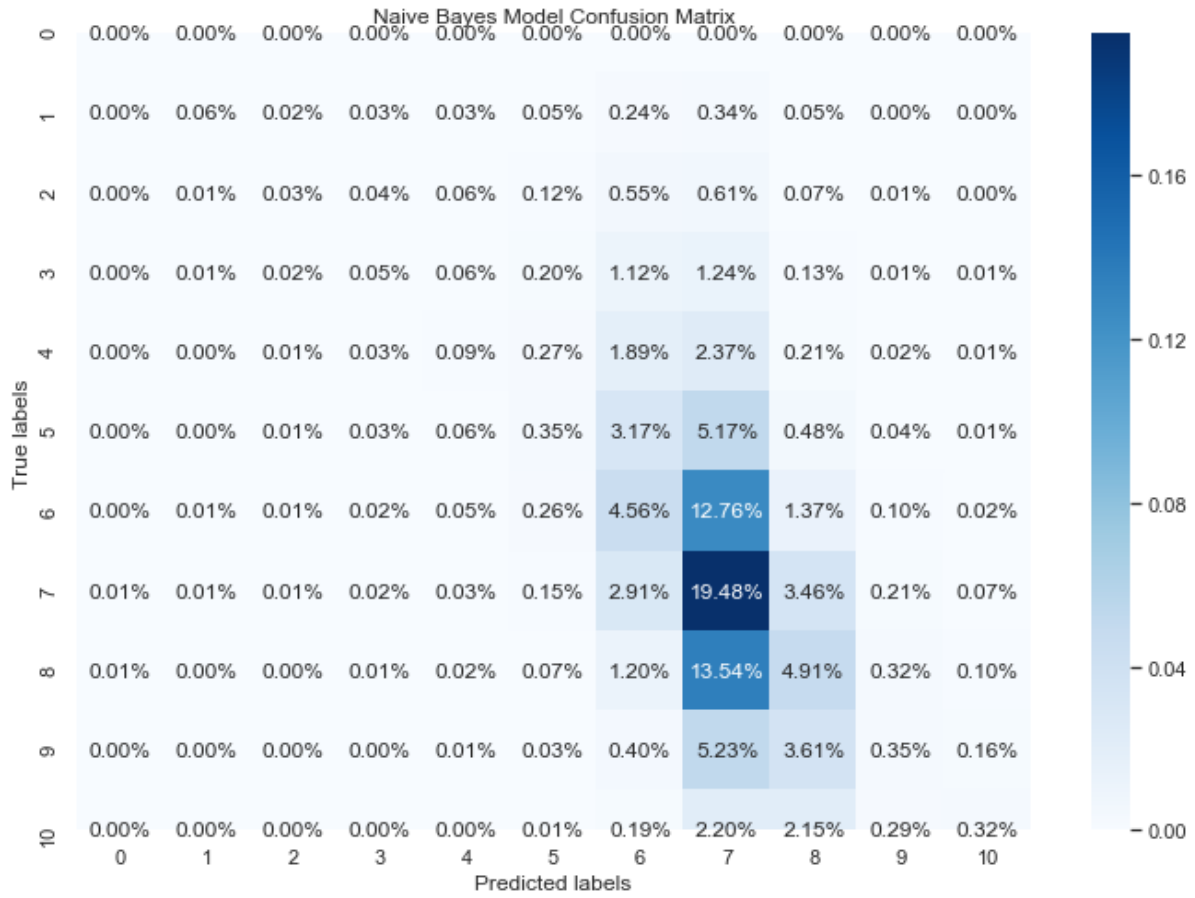
Confusion Matrix, Accuracy and MSE for Naive Bayers Model

In [42]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=0.01)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_dev)
accuracy5=accuracy_score(y_dev.astype('int'),y_pred, normalize=True) * float(1
00)
print('\n****Test accuracy is',(accuracy5))
MSE5=mean_squared_error(y_dev.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE5))
cm=confusion_matrix(y_dev.astype('int'),y_pred)
fig= plt.figure(figsize=(9,6))
ax = fig.add_axes([1,1,1,1])
sns.heatmap(cm/np.sum(cm), annot=True,ax = ax,fmt='0.2%', cmap='Blues')
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Naive Bayes Model Confusion Matrix')
```

```
****Test accuracy is 30.18777837877543

 Mean Squared Error 2.614611922457947
```

Out[42]: Text(0.5, 1, 'Naive Bayes Model Confusion Matrix')



Support Vector Machine

Support Vector Machine Algorithm requires longer time. That is why we use 10000 train data and 2,000 development data.

Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outliers detection.

The advantages of support vector machines are:

- Effective in high dimensional spaces.
- Still effective in cases where number of dimensions is greater than the number of samples.
- Uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.
- Versatile: different Kernel functions can be specified for the decision function. Common kernels are provided, but it is also possible to specify custom kernels.

The disadvantages of support vector machines include:

- If the number of features is much greater than the number of samples, avoid over-fitting in choosing Kernel functions and regularization term is crucial.
- SVMs do not directly provide probability estimates, these are calculated using an expensive five-fold cross-validation (see Scores and probabilities, below).

In [44]:
```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer,
TfidfTransformer
from sklearn.svm import LinearSVC
from sklearn import svm, linear_model
```

In [45]:
```
df2_train=df1_train[:10000]
df2_dev=df1_dev[:2000]
train2=df2_train['comment'].str.lower()
train_new2=train2.apply(lambda x:clean(x))
x_train2=train_new2
y_train2=df2_train['rating']
dev2=df2_dev['comment'].str.lower()
dev_new2=dev2.apply(lambda x:clean(x))
x_dev2=dev_new2
y_dev2=df2_dev['rating']
```

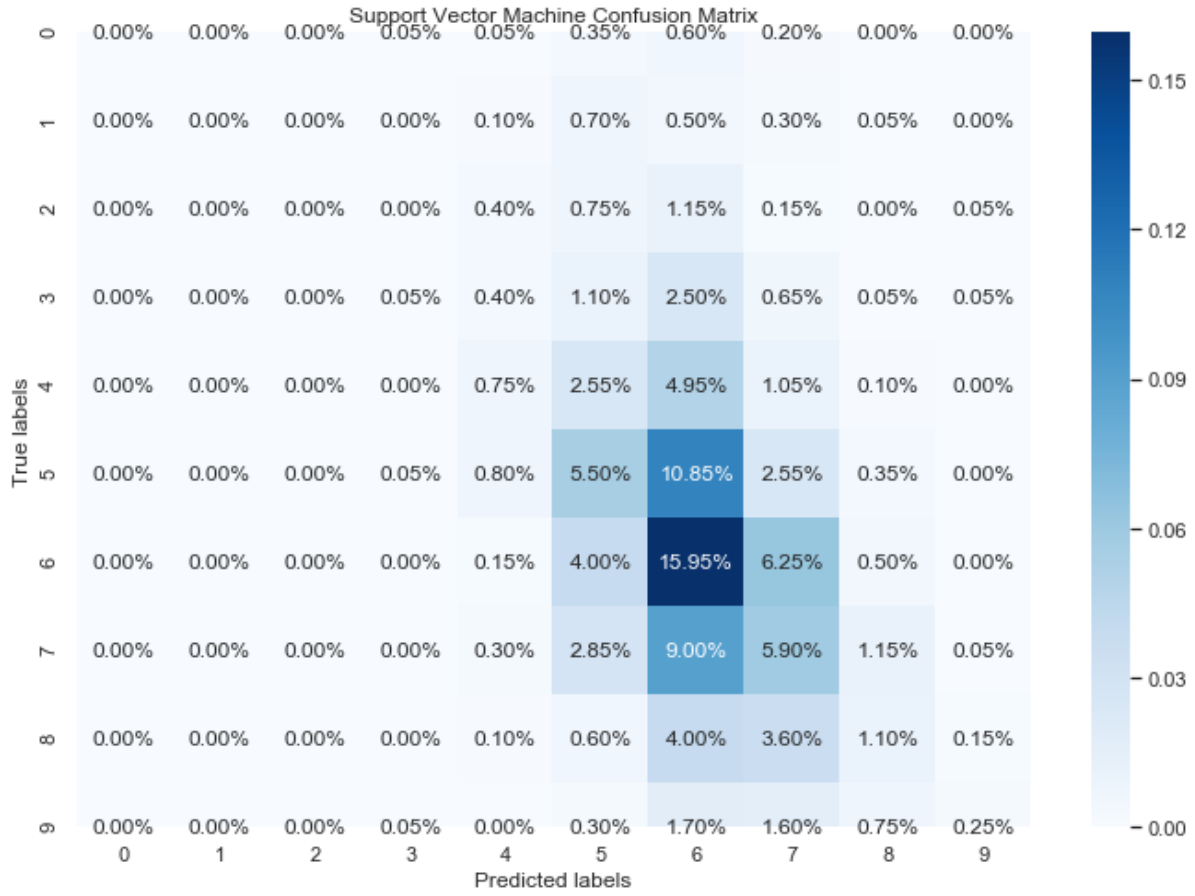
In [46]:
```
model_svc = make_pipeline(TfidfVectorizer(ngram_range=(1,3)), svm.SVC(kernel=
"linear",probability=True))
model_svc.fit(x_train2, y_train2.astype('int'))
y_pred = model_svc.predict(x_dev2)
accuracy14=accuracy_score(y_dev2.astype('int'),y_pred, normalize=True) * float
(100)
print('\n****Test accuracy is',(accuracy14))
MSE14=mean_squared_error(y_dev2.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE14))
```

```
****Test accuracy is 29.5

 Mean Squared Error 3.0465
```

In [47]:
```
cm=confusion_matrix(y_dev2.astype('int'),y_pred)
fig= plt.figure(figsize=(9,6))
ax = fig.add_axes([1,1,1,1])
sns.heatmap(cm/np.sum(cm), annot=True,ax = ax,fmt='0.2%', cmap='Blues')
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Support Vector Machine Confusion Matrix')
```

Out[47]: Text(0.5, 1, 'Support Vector Machine Confusion Matrix')



Best model we have is the Multinomial Naive Bayes with an alpha =0.01

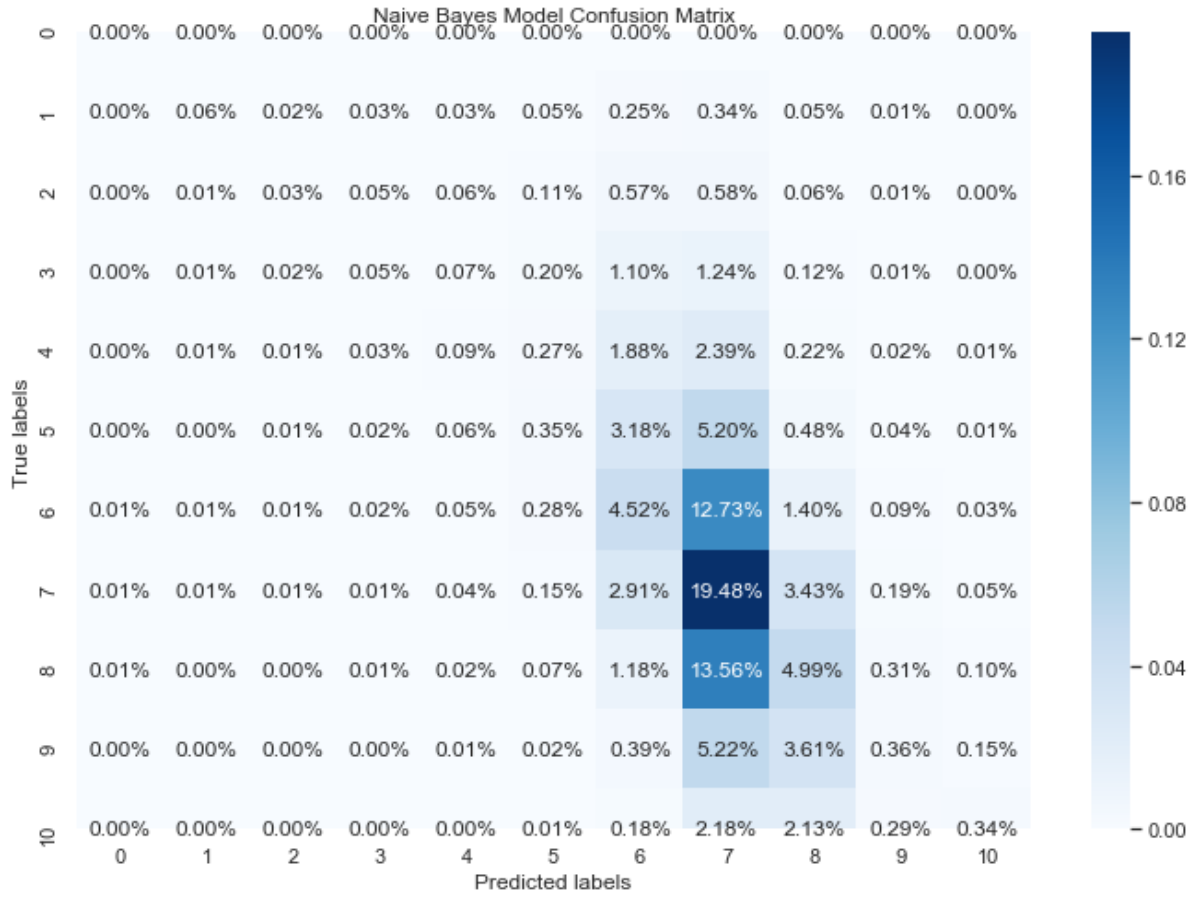
Final Model Accuracy with test data set

In [33]:
```
model_mb = Pipeline([
    ('count_vectorizer', CountVectorizer(lowercase = True, stop_words = stopwo
rds.words('english'))),
    ('tfidf_transformer',  TfidfTransformer()), #weighs terms by importance to
help with feature selection
    ('classifier', MultinomialNB(alpha=0.01)) ])
model_mb.fit(x_train,y_train.astype('int'))
y_pred=model_mb.predict(x_test)
accuracy5=accuracy_score(y_test.astype('int'),y_pred, normalize=True) * float(
100)
print('\n****Test accuracy is',(accuracy5))
MSE5=mean_squared_error(y_test.astype('int'),y_pred)
print('\n Mean Squared Error', (MSE5))
cm=confusion_matrix(y_test.astype('int'),y_pred)
fig= plt.figure(figsize=(9,6))
ax = fig.add_axes([1,1,1,1])
sns.heatmap(cm/np.sum(cm), annot=True,ax = ax,fmt='0.2%', cmap='Blues')
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Naive Bayes Model Confusion Matrix')
```

```
****Test accuracy is 30.27220140065609

 Mean Squared Error 2.601590971694727
```

Out[33]: Text(0.5, 1, 'Naive Bayes Model Confusion Matrix')



Resulting accuracy is still low. One of the main reason of low accuracy might be the huge data set. Also if there was enough time to complete SVC with more data set there is a possibility of higher accuracy

Challenges and Improvements

One of the main challenges of this project was the big amount of data. It really took a long time run because of the 15 million data. Due to this longer time period data has to confinded into reasonable amount for support vector machine. There wasn't enought time to run more models than the Naive Bayes model and Support Vector Machine. Althrough we tried different naive bayes model type and calculated accuracy with different hyper parameter. As we used a lot of data as a train set for naive bayes model it gives us a more generate idea. Also as we shuffled the data at the beginning, it gave us a good homogenous mix of results.

References:

1. https://scikit-learn.org/stable/modules/naive_bayes.html (https://scikit-learn.org/stable/modules/naive_bayes.html)
2. https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html (https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)
3. https://scikit-learn.org/stable/modules/svm.html (https://scikit-learn.org/stable/modules/svm.html)
4. https://www.geeksforgeeks.org/python-program-to-convert-a-list-to-string/ (https://www.geeksforgeeks.org/python-program-to-convert-a-list-to-string/)
5. https://www.geeksforgeeks.org/generating-word-cloud-python/ (https://www.geeksforgeeks.org/generating-word-cloud-python/)

In [ ]: