Problem Definition:

The key problem is classifying opinions on E commerce websites as positive or negative opinions. I am focusing on classifying the reviews given to the Video Games on Amazon as 'Like' or 'Don't Like' based on sentiment analysis algorithms.

1. Data Selection and Sampling

Data Source:

I have obtained the dataset from http://snap.stanford.edu/data/web-Amazon-links.html The dataset consists of "Video Game Reviews"

Instances: 463,669
Number of Attributes: 9

Following is an example of the single Data instance:

product/productId: B000068VC4
product/title: Virtual Resort: Spring Break
product/price: 4.67
review/userId: A3B8G17N9VEMJM
review/profileName: S. Cates "picklelips"
review/helpfulness: 2/2
review/score: 5.0
review/time: 1052352000
review/summary: This game is one of the best!
review/text: This game is totally awesome! Not only is it fun, well made, good graphics, and funny, it is also challenging and it keeps you wanting more! The scenario's are very challenging, and the people are funny. My only complaint is that they didn't put more cool scenarios on it! Then I could play this game forever!

1.1 Data Preparation:

The original data base was very large for our project. So following measures were taken to reduce the data base size.

1. First using the attribute "review/time" I sampled reviews from 2006 to 2013.

2. Then I removed all attributes from Instances except for "review/text" and "review/score" since other factors do not contribute in deciding the sentiments and will mostly act as noise.

3.Then I removed the instance with review less than 50 words since too shorter reviews mostly are not written seriously and are a result of impulse response rather than a true reaction/sentiment.

4. Then the attribute "review/score" was replaced by attribute "polarity". All the instances that had review score more than 3 were given polarity as 'Good' and for review score 3 and less polarity was 'Bad'.

This way I had 37,380 instances.

5. Out of these 37,380 reviews, 26,921 had 'Like' polarity and only 10,459 had 'Don't Like' polarity. So I randomly sampled 10,459 instances out of 26,921 with 'Like' polarity to balance the dataset.

All the above steps were done using Python.

6. Then I removed all the special characters and duplicate instances from the Review Attribute.

Finally, I had total 20918 instances (10459 'Like' + 10459 'Don't Like') with 2 attributes, review/text and polarity. Polarity is our class attribute.

1.2 Data Sampling:

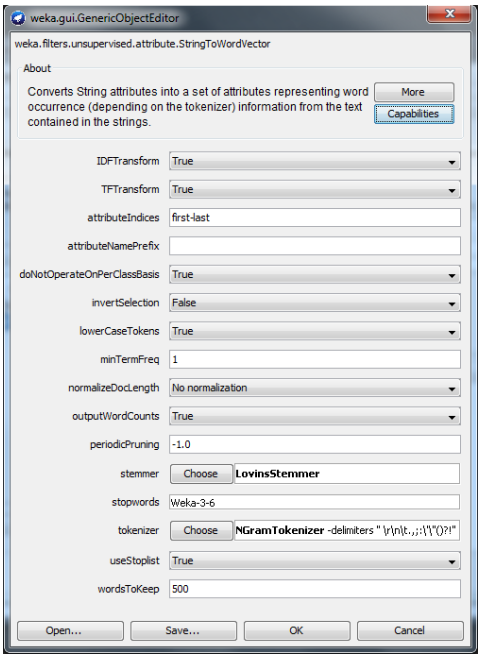
I used 75% of the above dataset for training and 25% for testing using Unsupervised Resample. Training set had 15,688 instances and Test set had 5230 instances.

Detailed results and snapshots are attached separately in Results folder.

2. Data Preprocessing

Bag of Words:

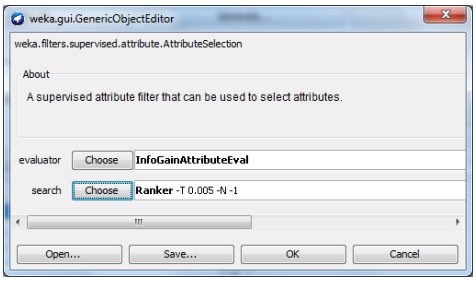
In pre-processing, first I used weka's "StringToWordVector" to create bag of words. Using this filter I created 200 bag of words.
Following is a screenshot for the parameters that were selected for this filter.



3. Feature Selection:

To create two Sets of attributes I selected Uni-gram in one set of attributes and for the second set I selected uni-gram and bi-gram both.
F1: contained only Uni-Gram
F2: contained Uni-gram and Bi-gram

"Lovins stemmer" was used as stemming method. To reduce the number of attributes I used the attribute selector " infogainAttribute Eval" which uses ranker as a search. The threshold for the information gain value was selected as 0.005.



After using attribute selector and manually removing the unwanted word attributes I were left with 36 **attributes** excluding class attribute for **F1** and **49 attributes** excluding class attribute for **F2**.

```
Search Method:
       Attribute ranking.
       Threshold for discarding attributes:   0.005

Attribute Evaluator (supervised, Class (nominal): 1 Polarity):
       Information Gain Ranking Filter

Ranked attributes:
 0.021      406 great
 0.01974    958 wast
 0.01945    538 lov
 0.01636    984 worst
 0.01525    951 wa
 0.01383    668 poor
 0.01383    451 hor
 0.01305    580 mone
 0.01162     87 awesom
 0.01113    262 disappoint
 0.01018    886 ter
 0.00998    981 work
 0.0091     123 bor
 0.00909     91 bad
 0.00883     37 addict
 0.00853    338 fantas
 0.00829     55 amaz
 0.00826    727 rent
 0.00819    376 frustr
 0.0076     750 rpg
 0.00734    983 wors
 0.00732    737 return
 0.00699    756 sam
 0.0069     381 gam
 0.00665    928 unfortun
 0.00648    603 noth
 0.00603    687 produc
 0.00582    858 stupid
```

4. Data Mining (Machine Learning)

I are using Naive Bayes , Decision Tree (through J48 and Random Forest), Back Propagation, and Support Vector Machine- (through Sequential Minimal Optimization-SMO) to classify the data and compare the observations.

4.1 F1 Attribute Set

The methods have been ranked according to their accuracy.

#1 Back Propagation

Using Back Propagation I got the best accuracy of **73.4557%** on Test Set using the parameters: Decay = True , Learning rate = 0.9, Momentum = 0.3, reset = True,
Stopping Criteria : Training Time = 500

Results:

Scheme:weka.classifiers.functions.MultilayerPerceptron -L 0.9 -M 0.3 -N 500 -V 0 -S 0 -E 20 -H a -D

=== Summary ===
Correctly Classified Instances 3841 73.4557 %
Incorrectly Classified Instances 1388 26.5443 %

=== Confusion Matrix ===

a	b	<-- classified as	
2051	566	a	= Like
822	1790	b	= NotLike

#2 Support Vector Machine - SMO

I got the accuracy of **73.4175%** on Test Set using the parameters: reduced ErrorPruning = False , UnPruned = False , useLaplace = False

Result:
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

=== Summary ===

Correctly Classified Instances	3839	73.4175 %
Incorrectly Classified Instances	1390	26.5825 %

=== Confusion Matrix ===

a	b	<-- classified as	
1972	645	a	= Like
745	1867	b	= NotLike

#3 Naive Bayes

Naive Bayes ranked 3rd. I got the best accuracy of **72.9011%** on Training set using the parameters UseKernelEstimator = False , UseSupervisedDiscretization = True

Result:
Scheme:weka.classifiers.bayes.NaiveBayes -D

=== Summary ===

Correctly Classified Instances	3812	72.9011 %
Incorrectly Classified Instances	1417	27.0989 %

=== Confusion Matrix ===

a	b	<-- classified as	
2094	523	a	= Like
894	1718	b	= NotLike

#4 Random Forest

Random Forest ranked 4th. I got the accuracy of **71.964%** on Test Set using the parameters maxDepth = 17 , numFeatures = 5 , numTrees = 100 , seed = 1

Result:
Scheme:weka.classifiers.trees.RandomForest -I 100 -K 5 -S 1 -depth 17
=== Summary ===

Correctly Classified Instances	3763	71.964 %
Incorrectly Classified Instances	1466	28.036 %

=== Confusion Matrix ===

a	b	<-- classified as	
1855	762	a	= Like
704	1908	b	= NotLike

#5 J48 Algorithm

J48 ranked last. I got the accuracy of **70.5297%** on Training set using the parameters reduced ErrorPruning = False , UnPruned = False, useLaplace = False

Result:
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

=== Summary ===

Correctly Classified Instances	3688	70.5297 %
Incorrectly Classified Instances	1541	29.4703 %

=== Confusion Matrix ===

a	b	<-- classified as	
1933	684	a	= Like
857	1755	b	= NotLike

4.2 F2 Attribute Set

#1 Support Vector Machine - SMO

I got the best accuracy of **75.1769%** on Test Set using the parameters: reduced ErrorPruning = False , UnPruned = False , useLaplace = False

Result:
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

=== Summary ===
Correctly Classified Instances 3931 75.1769 %
Incorrectly Classified Instances 1298 24.8231 %

=== Confusion Matrix ===
a b <-- classified as
1972 645 | a = Like
653 1959 | b = NotLike

#2 Back Propagation

Using Back Propagation I got the best accuracy of **75.0813%** on Test Set using the parameters: Decay = True , Learning rate = 0.9, Momentum = 0.3, reset = True,
Stopping Criteria : Training Time = 500

Results:

weka.classifiers.functions.MultilayerPerceptron -L 0.9 -M 0.3 -N 500 -V 0 -S 0 -E 20 -H a -D

=== Summary ===
Correctly Classified Instances 3926 75.0813 %
Incorrectly Classified Instances 1303 24.9187 %

=== Confusion Matrix ===
a b <-- classified as
2029 588 | a = Like
715 1897 | b = NotLike

#3 Naive Bayes

Naive Bayes ranked 3rd. I got the accuracy of **73.647%** on test set using the parameters UseKernelEstimator = False, UseSupervised Discretization = True

Result:
Scheme:weka.classifiers.bayes.NaiveBayes -D

=== Summary ===
Correctly Classified Instances 3851 73.647 %
Incorrectly Classified Instances 1378 26.353 %

=== Confusion Matrix ===
a b <-- classified as
2011 606 | a = Like
772 1840 | b = NotLike

#4 Random Forest

Random Forest ranked 4th. I got the accuracy of **72.3083%** on Test Set using the parameters
maxDepth = 17 , numFeatures = 5 , numTrees = 100 , seed = 1

Result:
Scheme:weka.classifiers.trees.RandomForest -I 100 -K 5 -S 1 -depth 17
=== Summary ===

Correctly Classified Instances 3781 72.3083 %
Incorrectly Classified Instances 1448 27.6917 %

=== Confusion Matrix ===

a b <-- classified as
1738 879 | a = Like
569 2043 | b = NotLike

#5 J48 Algorithm

J48 ranked last. I got the accuracy of **70.5297%** on Training set using the parameters reduced ErrorPruning = False , UnPruned = False, useLaplace = False

Result:
Scheme:weka.classifiers.trees.J48 -C 0.25 -M 2

=== Summary ===
Correctly Classified Instances 3712 70.9887 %
Incorrectly Classified Instances 1517 29.0113 %

=== Confusion Matrix ===
a b <-- classified as
1918 699 | a = Like
818 1794 | b = NotLike

5. Analysis

F1:

Rank	*Algorithm*	*Accuracy*
1.	*Back Propagation*	***73.455%***
2.	*SMO*	***73.417%***
3.	*Naive Bayes*	***72.901%***
4.	*Random Forest*	***71.964%***
5.	*J48*	***70.529%***

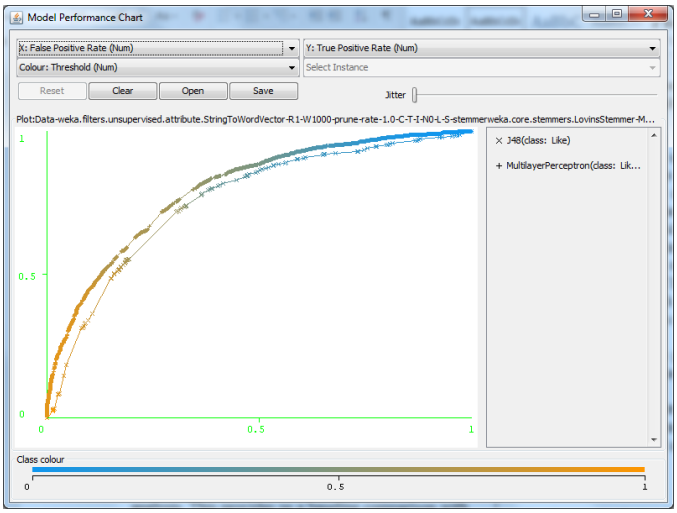
F2:

Rank	*Algorithm*	*Accuracy*
1.	*SMO*	***75.176%***
2.	*Back Propagation*	***75.081%***
3.	*Naive Bayes*	***73.647%***
4.	*Random Forest*	***72.308%***
5.	*J48*	***70.529%***

Based on above Accuracy results and other measures such as precision, recall and F-measue I can conclude that Support Vector Machines and Back Propogation is better in our case for Unigrams and Bigrams. Below is the Model performace chart (ROC curve) between Back Propogation and Decision Tree(J48).

ROC: ROC curve, is a graphical plot that illustrates the performance of a binary classifier system as its discrimination threshold is varied. Below curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

I can clearly see that Back Propogation performed better as compared to J48 in classifcaion task.



6. TextToSentiment Library

Apart from the standard algorithms described in last section, I used yet another source for sentiment analysis. This provides us a baseline comparison with the learning algorithms.

I used TextToSentiment library of the http://www.datasciencetoolkit.org/ to to do sentiment analysis based on English word dictionary. This library tries to guess whether the text represents a roughly positive or negative comment, and returns a score between -5 and 5. The lowest score indicates that there were words associated with unhappiness or dissatisfaction, a score of zero either means that no charged words were found, or that the positive and negative words balanced themselves out, and a high score shows that there were one or more 'good' words that occur in pleased, happy comments.

For example, "Great trip. Thanks for the warm welcome, Dallas and Fort Worth" gets a score of +2 because "great" has a weight of +3, and "warm" gives +1, so the average is +2. It's using a very simple algorithm, based around Finn Arup's annotated list of words.

I got very deviated results as compared to our learning based classification algorithm. I ran it on the whole dataset and it classified reviews as :

Positive-15644(scores1 to 5)
Negative-5274(scores 0 to -5)

But our original dataset was balanced with 10,459 positive and equal number of negative reviews.

One interesting case seen in sentiment analysis through text is with sarcasm. Since sarcasm uses positive words whereas its sentiment being negative, it becomes inherently difficult to detect and hence, classify it. Same is the inverse case, that is, a text which uses words which sound negative when taken a separate word at a time, but are totally opposite in meaning when forms a sentence. As this library is based on dictionary of English words so it can easily wrongly classify such reviews. For example this review got classified as negative(-3).

"I wanted to play this game so badly that i skipped my work. It kept me busy the whole day".

This is very different from learning based classification algorithms where I give the training data and build a model and then validate

7. Conclusions

Sentiment Analysis although a widely studied and researched area, remain challenging till date. The most popular and efficient algorithm struggle to hit a very good accuracy. However, the accuracy not only depends on the algorithm, but also the type of text and its quality. **Hence, it becomes very necessary to refine and preprocess the data.** A better data and wise selection of parameters is necessary, specially when dataset is huge. This improves the quality of data and hence increases the effieciency of algorithms. A detailed analysis of the data fed to the algorithm and it's capability to perform best on a certain type of text, and selection of right parameters is very important.

As can be seen in the result, better results were obtained when a feature selection was done and set of 2 attributes(F2) was fed to the classifiers

There were few advantages and disadvantages of each algorithm observed as I moved along the project. They are listed as below:

Naive Bayes: Naiv Bayes is better with smaller data than it is with big one. It was observed when I trained this with a small sample of 20% of the dataset. The accuracies were higher than it was with full datase. Its main disadvantage is that it can't learn interactions between features. For instance, it can't learn that a person loves games from EA sports and also loves FPS games, however, that person hates FPS games made by EA sports.

SVMs: They had the highest accuracy. They are anyway reputed to deliver a good performance in text classification problems where very high-dimensional spaces are the norm. However, they are memory-intensive and hard to interpret.

Decision Trees: The best part about decision tree is that they are easy to interpret and explain. They easily handle feature interactions and they're non-parametric, so they take care of outliers better. However, the biggest disadvantage is that they easily overfit.