CSCI5180 - Techniques for Data Mining

Project - Sentiment Analysis

This assignment contains 15 pages

Hoang Phuong Thai, s1155001492

Ying Ting Chung, s1008630404

Matti Jauhiainen, s1155001293

December 20, 2010

Chinese University of Hong Kong
Department of Computer Science and Engineering
Rm1028 Ho Sin-Hang Engineering Building, Shatin N.T. Hong Kong
Telephone (852) 26098440, Fax (852) 26035024
dept@cse.cuhk.edu.hk
www.cse.cuhk.edu.hk

We declare that the assignment here submitted is original except for source material explicitly acknowledged, and that the same or closely related material has not been previously submitted for another course. We also acknowledge that we are aware of University policy and regulations on honesty in academic work, and of the disciplinary guidelines and procedures applicable to breaches of such policy and regulations, as contained in the website.

Contents

1 Introduction **4**

2 Preprocessing **5**
2.1 Retrieving data 5
2.2 Cleaning data 6
2.3 Weka 7

3 Analysis **8**
3.1 Data 8
3.2 Attribute selection of candidate words 8

4 Discussion **13**
4.1 Model selection 13
4.2 Results 13
4.3 Interpretation 13
4.4 Future work 14

5 Conclusion **15**

A Appendix **16**
A.1 Running the shell scripts 16
A.2 imdb.sh 16
A.3 looppages.sh 16
A.4 getreviews.sh 16
A.5 clean.sh 16
A.6 preprocessor.java 17
A.7 wordoccurences.sh 17
A.8 J48 Decision Tree 17

1 Introduction

The motivation for this project is to predict a user rating given a text based on the emotional context of negative or positive words. Conceptually given a string of text one could predict whether the content would infer positive or negative emphasis. However, this is a very complex topic that a lot of companies are willing to give huge amounts of research hours to support this technology. Why is it important to infer negative and positive texts one may ask? Companies such as *Amazon*[1], which is an online store can use the data from user reviews and the respective rating to certain products to infer whether the product is something useful for other users.

In this project we have chosen to focus on movie reviews and ratings from *imdb.com*[2] that stores all movie information and user reviews. *IMDb* has stored billions of user reviews and ratings and by extracting the user reviews along with the ratings one could use the reviews and predict the rating.

This report is divided in 3 main parts - namely *preprocessing*, *analysis* and *discussion*. In the preprocessing section we will describe how the data set was retrieved, how it was cleaned and finally converted to an *.arff* file, which is a compatible file format for the tool Weka. In the Analysis section we are concerned about the actual data set and how it is interpreted in Weka and further adjustments to improve the classification process. The discussion section will include justifications of the choices we made for the classification, the approach of the decision tree and the results of the classification.

Phuong, *Ying* and *Matti*
December 20, 2010

[1] http://amazon.com
[2] http://imdb.com

2 Preprocessing

This section will discuss the approach of retrieving and cleaning the data set which was taken from *IMDb*[3]. Finally, the data set has been converted to an *.arff* file which is the file extension for the Weka tool we have used for classification in this project.

Before retrieving the data it was important to figure out what data would make sense to retrieve. Essentially the most vital attributes were a user rating n $\{1 \leq n \leq 10 | n \in \mathbb{N}\}$ and a review bound to this rating n, where a review t is a long string of text $\{t \in alphabet : [a - zA - Z0 - 9]\}$. However, a review may contain a lot of rubbish words or in other words noise to the classification process - hence it was necessary to eliminate this noise. In these next subsections the process of data retrieval, cleaning and finally converting to a Weka compatible file is discussed.

2.1 Retrieving data

The data retrieval has been executed by using *shell scripts* and *Java* implementations. Every movie on IMDb has a unique id - one could be tempted to loop through all id's, but since IMDb holds 95% of all movies made our approach had to be tweaked. Using IMDb's search engine and retrieving a list of all movies that had more than 1000 user reviews would be a more appropriate way to retrieve data, since some movies might not have any user reviews. This data retrieval process can be divided in three steps. Hence 3 shell scripts were created. All scripts may be found in the appendix A.

imdb.sh :
: This shell scripts essentially creates a search looking for movies that has more than 1000 user reviews. Each search is then being saved in a *.html* file. Each html file contains all the unique id's of every movie. The shell script can be found in the appendix A.2.

looppages.sh :
: Now that all html pages has been saved looppages.sh handles each html page saved by imdb.sh to extract all unique id's and saves them in a file. The script can be found in appendix A.3

getreviews.sh :
: Finally, this scripts handles the actual retrieval of the user reviews - each review is being checked whether a rating is attached to it otherwise the review will be skipped. Reviews with no rating is considered noise and vice versa. There are more than 1300 movie titles created by the looppages.sh. We found it sufficient to take only the first 10 reviews of each movie which would give a data set of more than 13000 records and furthermore avoid too biased records - it might be the case that a movie is very good which

[3] http:www.imdb.com

would in this case only give good user reviews and ratings. The script can be found in appendix A.4

Retrieving the data set took approximately one hour and created one data set file of 16 mega bytes. As one may expect there were a lot of rubbish words in each record that should be removed to make a better classification.

2.2 Cleaning data

The data that the shell scripts generated were raw data with noise. Unnecessary words had to be removed to create a more precise classification of the records. Ambiguous words, filling words and names should be eliminated - emotional words were only of interest. One more shell script were created to print out the occurrences of all words. The table 1 shows the first few lines of the script *wordoccurences.sh* (appendix A.7) output.

Number of occurrences	Word
144655	the
85009	and
71638	to
70766	a
69071	of
52134	is
39977	in
34675	that
32271	I
30685	it
25182	this
23149	for
21318	with
20179	was
20128	as
18836	from
16881	The
16344	but
15147	his
14953	you

Table 1: First few lines of the word occurrences of the over 13000 record data set

It is evident from table 1 that the data set contains words that does not contribute to the classification. All words that did not have an emotional impact were removed. However, we are *aware* of *negation*, *double negation*, *ambiguity* and *irony* humans tend to express in spoken and written language. This is a very complex problem that is out of scope in this project, but essentially a very

advanced processing of each sentence in each review should be considered to identify these features. Since only emotional words are considered individually it may not give a very precise classification, but nevertheless can be used.

It may be the case that a user review states *"I didn't like the movie"*, which is clearly a negative emphasis but is interpreted and classified as positive in our case. As stated earlier this problem is out of scope of this project but worth having in mind.

2.3 Weka

To make the data set Weka compatible some minor regular expressions had to be implemented in a shell script and a Java implementation. Inserting a comma on each line to separate the attributes and adding quotation marks on each review to denote a string. These two implementations can be referred to A.5 and A.6 in appendix.

3 Analysis

In this section an analysis on the classification will be given clarifying the results and which compromises we had to take in order to minimize the processing time.

3.1 Data

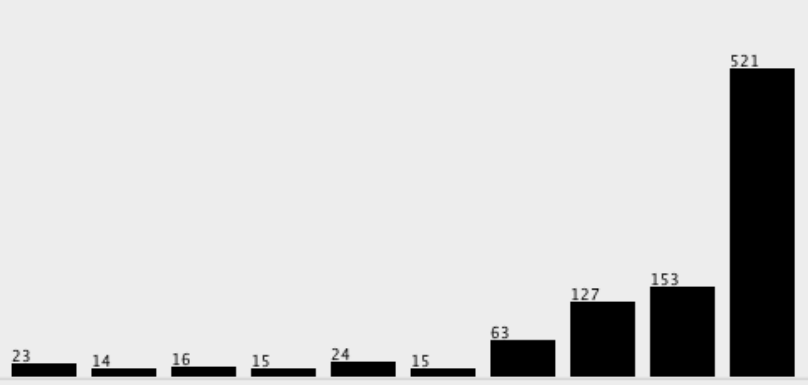

Figure 1: Discretize Ratings

Now that the data is cleaned the input vector of dimension p is defined as $x \in \mathbb{R}^p$ consisting of $\vec{x}_i = (n_1, n_2, .., n_p)$, where $\{0 \leq n_p, n_p \in \mathbb{N}\}$ over the alphabet $\{t \in alphabet : [a - zA - Z0 - 9]\}$. The output is $\{1 \leq n \leq 10, n \in \mathbb{N}\}$ based on the text t.

Figure 1 shows the rating being discretized and figure 2 showing the frequency relatively to the ratings. One may quickly notice that the ratings are biased towards $7 \leq n$, which essentially results in a biased classification. A way to avoid this is the notion of under sampling the majority class to eliminate the imbalanced data set. This approach is in fact a noise reducing method such that flattening the data set to have roughly an even class distribution gives a better classification.

Table 2 shows the frequency for the whole data set of each input vector x. Looking closer at the table it is clear that the contribution of rating $n = 10$ is 50% which is 40% above the limit. The desired distribution would be a 10% representation of each class.

3.2 Attribute selection of candidate words

The selection of candidate words is in essence a further preprocessing, since selecting exactly those words that impacts the classification. Noisy data in the

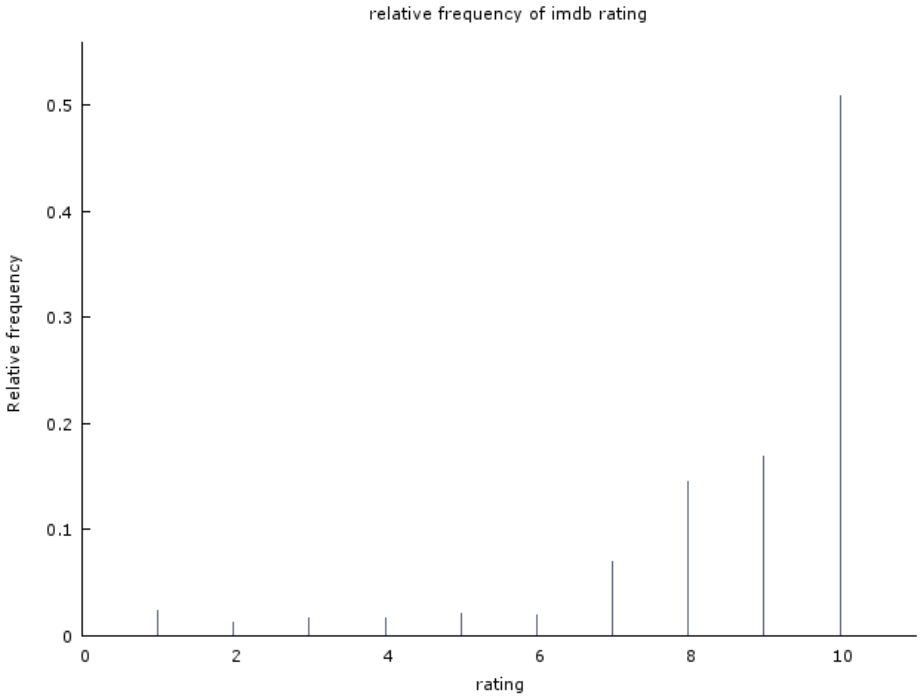

Figure 2: Relative Frequency for rating

Rating	Frequency	Relatively	Accumulated
1	221	2.28%	2.28%
2	119	1.23%	3.50%
3	153	1.58%	5.08%
4	153	1.58%	6.65%
5	204	2.10%	8.76%
6	187	1.93%	10.68%
7	680	7.01%	17.69%
8	1411	14.54%	32.22%
9	1644	16.94%	49.16%
10	4935	50.84%	100.00%

Table 2: Frequency distribution for rating, obs 1-9707

raw data set is present and choosing candidates is a very important process.

Figure 3 shows the class distribution over frequencies of the word 'excellent'. Each colour in a bar denotes a discretized rating class. For movie reviews in which 'excellent' appears once, the movie has over 50% of chance to be rated

Rating	Frequency
Mean	8.6397
Median	10.000
Minimum	1.0000
Maximum	10.000
Standard deviation	2.0817
C.V.	0.24095
Skewness	-2.0587
Ex. Kurtosis	4.0055

Table 3: Summary statistics, using the observations 1 - 9707 for the variable 'rating' (9707 valid observations)

10. By seeing whether high rating or low rating dominates, it helps deciding whether the word is positive or negative respectively. By comparing the class distribution over different word frequencies, it helps attribute selection. Suppose some word making no difference to rating distributions either its presence or absence. We could assume such word giving no information to our classification task.

Table 4 shows the 43 emotional word candidates that occurred more than *100 times* in the data set. The word candidates are divided in two word features of positive and negative, respectively. A positive word occurrence would have a positive impact on the rating increasing it and a negative word would impact the rating by decreasing the value.

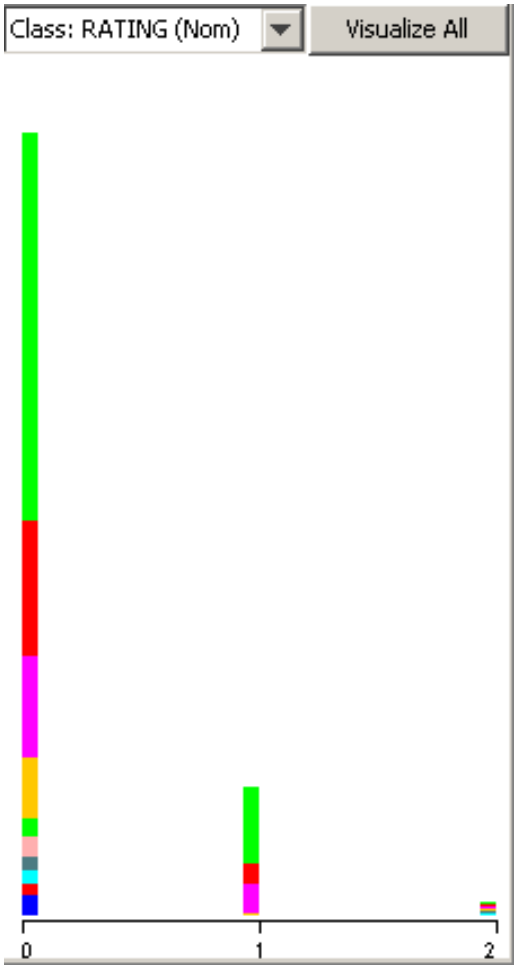

Figure 3: Frequency of excellent distributed over ratings

Positive word candidates	Negative word candidates
amazing	annoy
beautiful	bad
clever	bizarre
cool	boring
enjoy	brutal
excellent	crazy
fantastic	cry
fine	disappointed
funny	disassociated
hilarious	disturbing
humor	dry
incredible	fake
intellectual	gay
laugh	hurt
likeable	inhuman
love	irreverent
realistic	scared
romantic	stupid
smart	uncomfortable
moody	weak
talented	wrong
terrific	

Table 4: Word candidates 22 positive words, 21 negative words

4 Discussion

This section includes the selection of our model, results, our interpretation of the results and finally future work.

4.1 Model selection

In essence, decision tree classification is a recursive partitioning of feature space into class regions. Partitioning is done along coordinate axes so decision boundary is rectilinear. In our task, input attributes and target variable are all nominal.

Decision tree has also been known for its interpretability. Any path from the parent node down to decision node is just a conjunctive conditional statement. Natural language is relatively harder to be represented and interpreted in machines. Interpretability could give human trainer an edge to understand and improve the text mining process.

4.2 Results

The word candidate list has been reduced significantly, however this has been a compromise that we have taken to reduce the processing time and tree complexity in Weka. Initially a word candidate list consisting of 60 positive and 60 negative words gave an accuracy of 99% with a cross validation of *10 folds*. The compromise of decreasing the tree complexity and faster processing receiving a worse accuracy versus a complex tree structure and a time consuming processing to get a better accuracy. A reduction of approximately 60% on the number of input words gave an accuracy of 80% with cross validation of 10 folds which is acceptable.

# input words	# Leaves	Size of tree	Accuracy	Avg. F-Measure
120	239	477	99.41%	99.40%
78	251	501	96.38%	96.40%
68	245	489	93.89%	93.80%
63	227	453	90.14%	90%
49	199	397	84.74%	84.20%
46	176	351	82.34%	81.60%
43	167	333	80.96%	80.00%
39	152	303	76.60%	75.40%

Table 5: Tree complexity

4.3 Interpretation

The number of English words are huge and we only selected 43 out of dictionary. Naturally, input word vectors are very sparse. Substantial amount of entries are

zeros; Frequencies of a particular word, like 'love', could appear as many as 6 times in one single review. But generally variance of frequencies is small.
Due to sparse data, we actually speculate that still more training examples could be included. Sampling size of current project might be prone to over-fitting. In fact, we are uncertain with the F-measures obtained in Table 5. Therefore we also evaluated the model using tree complexity. Not only is this a complexity-consistency tradeoff, but a simpler hypothesis would generalize better as well.

4.4 Future work

In order to get better results with fewest attributes, we sought to modify current approach in some other ways.

1. We experimented with other tree induction algorithms: SimpleCART, ID3, RandomForest. Surprisingly they all gave worse result than J48.
2. We tried marking the occurrence of word (presence as 1, absence as 0) instead of counting frequencies. Result deteriorated as well.
3. Realizing this is an imbalanced dataset, we applied random under-sampling, i.e. deleting roughly 3000 records with rating of 10. Result was unsatisfactory.

Whether to apply over-sampling to minority classes or under-sampling to majority classes, and type of sampling (random, focused, criteria customized to problem domain) is still a big subject to be understood.

5 Conclusion

The subject of *sentimental analysis* is very complex and in the field of *machine learning*. The hardest part in this matter is the fact that each word individually can not be interpreted directly. Hence the notion of negation, double negation, metaphors and irony used in text does give an immense impact on the total meaning of a sentence. In this project we have excluded this fact and focused on individual emotional words, which in a sense can be used. However, this is not a very precise approach on a higher level of classification.

During this process the decision of taking compromises has been made on reducing the amount of the individual emotional words to reduce the processing time of a very complex decision tree (J48). The cost of this compromise is a lower accuracy since some important words were eliminated. However, an accuracy of roughly 80% is acceptable in the context of ignoring the fact of the complexity of sentences.

Furthermore, different algorithms were applied on the data set such as *SimpleCART*, *ID3*, and *RandomForest* with no significant improvements. Lastly, the notion of *under-sampling* were applied to reduced the majority class but with no significant improvements.

This is a project that definitely needs further research, our approach is moving in the right direction, however other methods should be applied to optimize the classification. One could apply the Naive Bayes model to handle this problem.

A Appendix

A.1 Running the shell scripts

1. Use the *imdb.sh* to get the search result pages with links movie pages.
2. Use *looppages.sh* to go trough the search result pages for links to user review pages. Links are saved to *booyahlist.txt* (the list is removed in the beginning of the script so make backups.)
3. Use *getreviews.sh* to construct the big file *lesscrap* with the movie reviews and ratings.
4. Get word occurrences with *wordoccurences.sh*.

A.2 imdb.sh

```
#!/bin/bash
#Script gets the pages with links to movies with more than 1000 votes
#in descending order by the release date. Pages are saved as page101.html etc...
#Modify the link to change the search, the variable index points to the result
#page number.

for index in 1 101 201 301 401 501 601 701 801 901 1001 1101 1201 1301 1401 1501 1601;
do
   curl -o page$index.html http://www.imdb.com/search/title?num_votes=1000,&sort=year,desc&start=$index&view=simple
# curl http://www.imdb.com/search/title?num_votes=1000,&sort=year,desc&start=$index&view=simple > searchRes$index.html
done
```

A.3 looppages.sh

```
#Loops through html pages in folder and gets the imdb link strings.
#The links are saved in the thebooyahlist.txt

for page in `ls | grep .html`;
do
   cat $page | sed 's/tt[0-9][0-9]*/BOOYAH&HAYOOB/g' | grep BOOYAH | sed 's/.*BOOYAH//' | sed 's/HAYOOB.*//' | uniq >>
done
```

A.4 getreviews.sh

```

rm lesscrap
for link in `cat thebooyahlist.txt`;
do
# $page = http://www.imdb.com/title/$link/usercomments?start=0
# echo $page | grep alt=\"[0-9][0-9]*\/10\"␣|␣sed␣'s/.*alt=\"//'␣|␣sed␣'s/[0-9][0-9]*\/10/&BOOYAH/'␣|␣sed␣'s/BOOYAH.*/
# echo $page | tr '\n' '␣' | sed -e 's/\<\/p\>.\<p\>/BOOYAH/g' | sed 's/\<\/p\>\<p\>/BOOYAH/g' | sed 's/BOOYAH/\    /g'|

# curl http://www.imdb.com/title/$link/usercomments?start=0 | grep alt=\"[0-9][0-9]*\/10\" | sed 's/.*alt=\"//' | sed
   curl http://www.imdb.com/title/$link/usercomments?start=0 | tr '\n' ' ' | sed -e 's/\<img width=\"102/BOOYAH/g'| sed

# curl http://www.imdb.com/title/$link/usercomments?start=0 | tr '\n' ' ' | sed -e 's/\<\/p\>.\<p\>/BOOYAH/g'| sed 's/
done




#cat dump | sed -e 's/\<\/p\>.\<p\>/BOOYAH/g'| sed 's/\<\/p\>\<p\>/BOOYAH/g' | sed 's/BOOYAH/\    /g'| tr '    ' '\n' >
#cat dump3 | sed -e 's/<[a-zA-Z\/][^>]*//g' > notags
#cat notags | sed -e 's/[^a-zA-Z␣\n]//g' > justascii
#cat justascii | sed -e 's/Was␣the␣above␣review␣useful␣to␣you.*//g' > lesscrap
```

A.5 clean.sh

```
#cat lesscrap2 | sed -e 's/[␣][0-9]+/␣/g' > test

cat lesscrap2 | sed 's/\([␣][0-9]*[␣]\)/\1,"/' | sed 's/$/"/'> test
```

A.6 preprocessor.java

```
1
2  import java.io.*;
3
4  public class preprocessor {
5
6    /* Removing all lines in the file specified by path "inFile",
7     * which matches the specified "line",
8     * and output a new pre-processed file to path "outFile"
9    */
10   public void removeLines(String inFile, String outFile, String line) throws IOException {
11
12     BufferedReader br = new BufferedReader(new FileReader(inFile));
13     BufferedWriter bw = new BufferedWriter(new FileWriter(outFile));
14
15     String inputLine;
16
17     while((inputLine = br.readLine()) != null)  {
18       if(!inputLine.trim().equalsIgnoreCase(line))  {
19         bw.write(inputLine);
20         bw.newLine();
21       }
22     }
23
24     br.close();
25     bw.close();
26   }
27 }
```

A.7 wordoccurences.sh

```
1  #Counts the wordoccurances in file.
2
3  perl -0777 -lape's/\s+/\n/g' FILENAME | sort | uniq -c | sort -nr > wordoccurences.txt
```

A.8 J48 Decision Tree

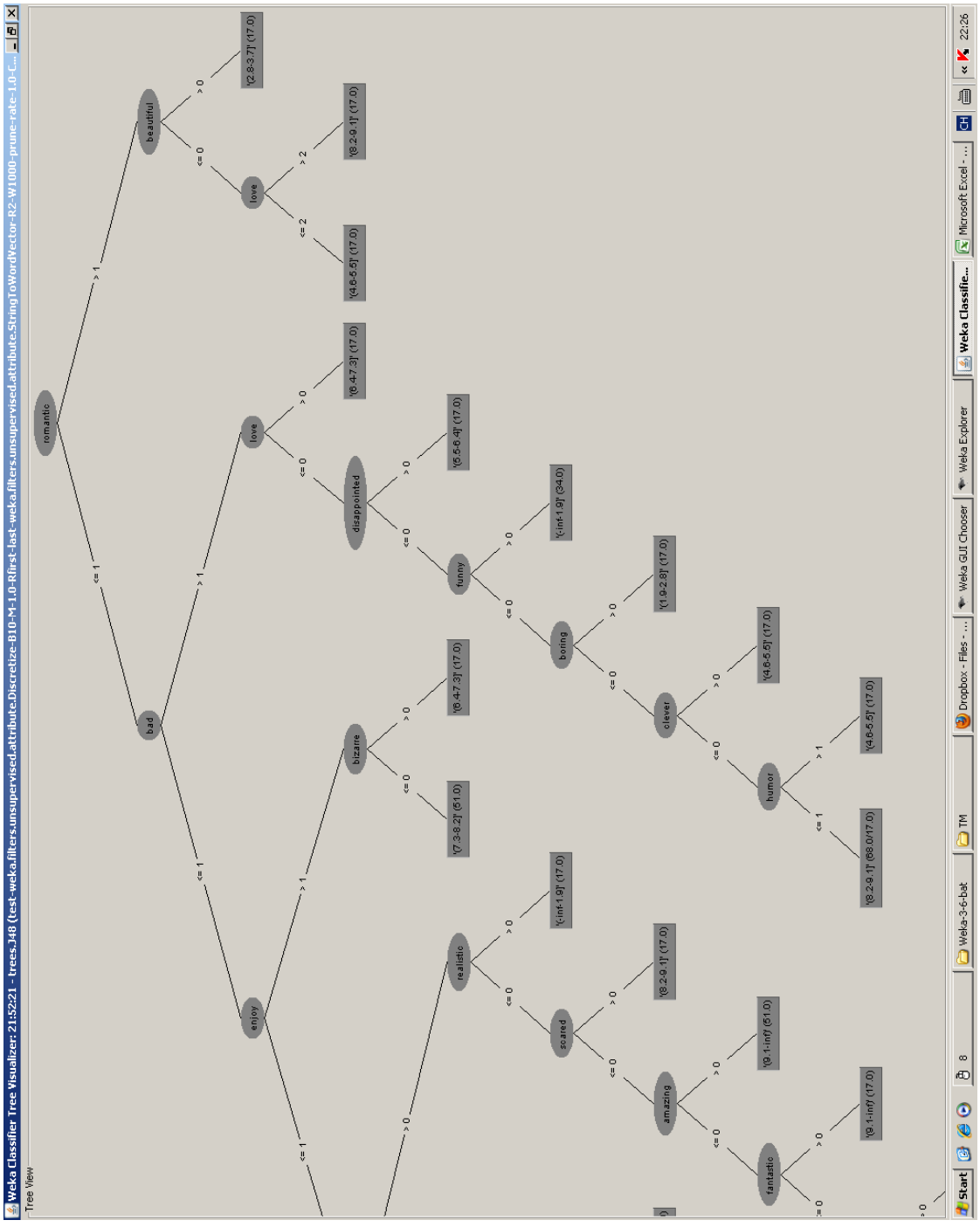

Figure 4: J48 Decision Tree