Data Mining
Text Classification
Naive Bayes

Ad Feelders

Universiteit Utrecht

Text Mining

- Text Mining is data mining applied to text data.
- Often uses well-known data mining algorithms.
- Text data requires substantial pre-processing.
- This typically results in a large number of attributes (for example, the size of the vocabulary/dictionary).

Text Classification

- Predict the class(es) of text documents.
- Can be single-label or multi-label.
- Multi-label classification is often performed by building multiple binary classifiers (one for each possible class).
- Examples of text classification:
    - topics of news articles,
    - spam/no spam for e-mail messages,
    - sentiment analysis (e.g. positive/negative review),
    - opinion spam (e.g. fake reviews),
    - music genre from song lyrics

Is this Rap, Blues, Metal, or Country?

```
 Blasting our way through the boundaries of Hell
 No one can stop us tonight
 We take on the world with hatred inside
 Mayhem the reason we fight
 Surviving the slaughters and killing we've lost
 Then we return from the dead
 Attacking once more now with twice as much strength
 We conquer then move on ahead

[Chorus:]
 Evil
 My words defy
 Evil
 Has no disguise
 Evil
 Will take your soul
 Evil
 My wrath unfolds

 Satan our master in evil mayhem
 Guides us with every first step
 Our axes are growing with power and fury
 Soon there'll be nothingness left
 Midnight has come and the leathers strapped on
 Evil is at our command
 We clash with God's angel and conquer new souls
 Consuming all that we can
```

Probabilistic Classifier

A probabilistic classifier assigns a probability to each class. In case a class prediction is required we typically predict the class with highest probability:

$$\hat{c} = \arg\max_{c \in C} P(c \mid d) = \arg\max_{c \in C} \frac{P(d \mid c)P(c)}{P(d)}$$

where d is a document, and C is the set of all possible class labels.

Since $P(d) = \sum_{c \in C} P(c, d)$ is the same for all classes, we can ignore the denominator if we only need to know the most likely class:

$$\hat{c} = \arg\max_{c \in C} P(c \mid d) = \arg\max_{c \in C} P(d \mid c)P(c)$$

Naive Bayes

Represent document as set of features:

$$\hat{c} = \arg\max_{c \in C} P(d \mid c)P(c) = \arg\max_{c \in C} P(x_1, \ldots, x_m \mid c)P(c)$$

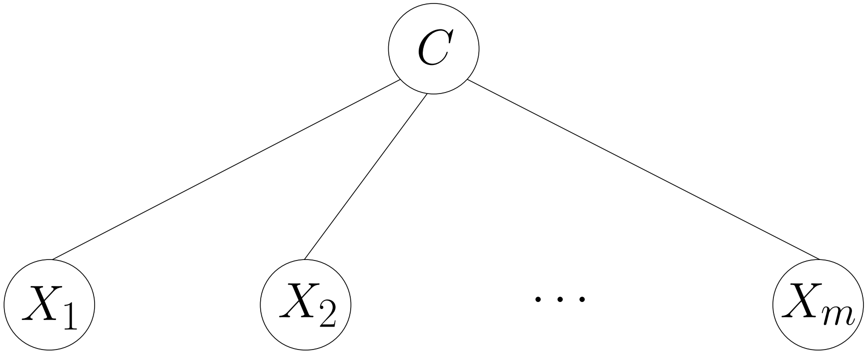
Naive Bayes assumption:

$$P(x_1, \ldots, x_m \mid c) = P(x_1 \mid c)P(x_2 \mid c) \cdot \ldots \cdot P(x_m \mid c)$$

The features are assumed to be independent within each class (avoiding the curse of dimensionality).

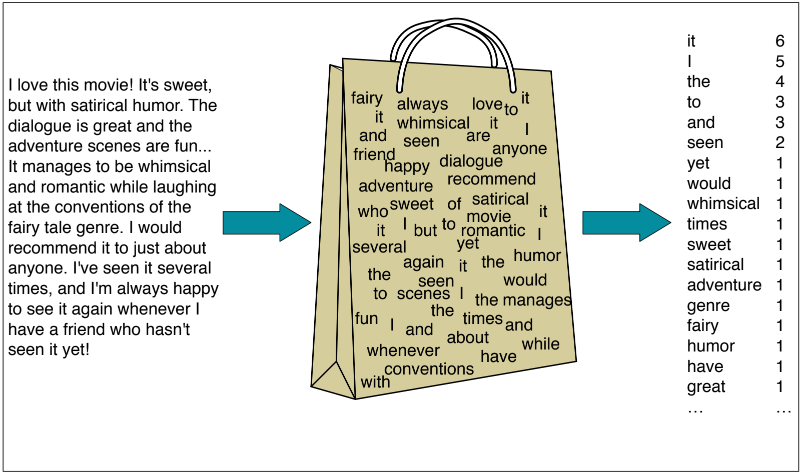
$$c_{\text{NB}} = \arg\max_{c \in C} P(c) \prod_{i=1}^{m} P(x_i \mid c)$$

Independence Graph of Naive Bayes



The features are independent given the class label.

Bag Of Words Representation of a Document



I love this movie! It's sweet, but with satirical humor. The dialogue is great and the adventure scenes are fun... It manages to be whimsical and romantic while laughing at the conventions of the fairy tale genre. I would recommend it to just about anyone. I've seen it several times, and I'm always happy to see it again whenever I have a friend who hasn't seen it yet!

it	6
I	5
the	4
to	3
and	3
seen	2
yet	1
would	1
whimsical	1
times	1
sweet	1
satirical	1
adventure	1
genre	1
fairy	1
humor	1
have	1
great	1
...	...

Figure 6.1 Intuition of the multinomial naive Bayes classifier applied to a movie review. The position of the words is ignored (the *bag of words* assumption) and we make use of the frequency of each word.

Bag Of Words Representation of a Document

Not matter, the order and position do.

Multinomial Naive Bayes for Text

Represent document d as a sequence of words: $d = \langle w_1, w_2, \ldots, w_n \rangle$.

$$c_{\text{NB}} = \arg\max_{c \in C} P(c) \prod_{k=1}^{n} P(w_k \mid c)$$

Notice that $P(w \mid c)$ is independent of word position or word order, so d is truly represented as a bag-of-words. Taking the log we obtain:

$$c_{\text{NB}} = \arg\max_{c \in C} \log P(c) + \sum_{k=1}^{n} \log P(w_k \mid c)$$

By the way, why is it allowed to take the logarithm?

Multinomial Naive Bayes for Text

Consider the text (perhaps after some pre-processing)

```
catch as catch can
```

We have $d = \langle \texttt{catch}, \texttt{as}, \texttt{catch}, \texttt{can} \rangle$, with $w_1 = \texttt{catch}$, $w_2 = \texttt{as}$, $w_3 = \texttt{catch}$, and $w_4 = \texttt{can}$. Suppose we have two classes, say $C = \{+, -\}$, then for this document:

$$\begin{aligned}
c_{\text{NB}} &= \arg\max_{c \in \{+,-\}} \log P(c) + \log P(\texttt{catch} \mid c) + \log P(\texttt{as} \mid c) \\
&\quad + \log P(\texttt{catch} \mid c) + \log P(\texttt{can} \mid c) \\
&= \arg\max_{c \in \{+,-\}} \log P(c) + 2 \log P(\texttt{catch} \mid c) + \log P(\texttt{as} \mid c) \\
&\quad + \log P(\texttt{can} \mid c)
\end{aligned}$$

Training Multinomial Naive Bayes

Class priors:

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

Word probabilities within each class:

$$\hat{P}(w_i \mid c) = \frac{\text{count}(w_i, c)}{\sum_{w_j \in V} \text{count}(w_j, c)} \quad \text{for all } w_i \in V,$$

where V (for Vocabulary) denotes the collection of all words that occur in the training corpus (after possibly extensive pre-processing).

count(w, c) is the number of times word $w \in V$ occurs in a document of class $c \in C$.

Interpretation of word probabilities

Word probabilities within each class:

$$\hat{P}(w_i \mid c) = \frac{\text{count}(w_i, c)}{\sum_{w_j \in V} \text{count}(w_j, c)} \quad \text{for all } w_i \in V$$

Interpretation: if we draw a word at random from a document of class c, the probability that we draw w_i is $\hat{P}(w_i \mid c)$.

Verify that

$$\sum_{w_i \in V} \hat{P}(w_i \mid c) = 1,$$

as required.

Training Multinomial Naive Bayes: Smoothing

Perform *smoothing* to avoid zero probability estimates.

Word probabilities within each class with Laplace smoothing are:

$$\hat{P}(w_i \mid c) = \frac{\text{count}(w_i, c) + 1}{\sum_{w_j \in V}(\text{count}(w_j, c) + 1)} = \frac{\text{count}(w_i, c) + 1}{\sum_{w_j \in V} \text{count}(w_j, c) + |V|}$$

Verify that again

$$\sum_{w_i \in V} \hat{P}(w_i \mid c) = 1,$$

as required.

The $+1$ is also called a pseudo-count: pretend you already observed one occurrence of each word in each class.

Worked Example: Sentiment of Movie Reviews

	Cat	Documents
Training	-	just plain boring
	-	entirely predictable and lacks energy
	-	no surprises and very few laughs
	+	very powerful
	+	the most fun film of the summer
Test	?	predictable with no fun

The Vocabulary

The vocabulary consists of all words that occur in the training documents:

1. just
2. plain
3. boring
4. entirely
5. predictable
6. and
7. lacks
8. energy
9. no
10. surprises
11. very
12. few
13. laughs
14. powerful
15. the
16. most
17. fun
18. film
19. of
20. summer

Class Prior Probabilities

Recall that:

$$\hat{P}(c) = \frac{N_c}{N_{doc}}$$

So we get:

$$\hat{P}(+) = \frac{2}{5} \qquad \hat{P}(-) = \frac{3}{5}$$

Word Conditional Probabilities

To classify the test example, we need the following probability estimates:

$$\hat{P}(\textit{predictable} \mid -) = \frac{1+1}{14+20} = \frac{1}{17} \qquad \hat{P}(\textit{predictable} \mid +) = \frac{0+1}{9+20} = \frac{1}{29}$$

$$\hat{P}(\textit{no} \mid -) = \frac{1+1}{14+20} = \frac{1}{17} \qquad \hat{P}(\textit{no} \mid +) = \frac{0+1}{9+20} = \frac{1}{29}$$

$$\hat{P}(\textit{fun} \mid -) = \frac{0+1}{14+20} = \frac{1}{34} \qquad \hat{P}(\textit{fun} \mid +) = \frac{1+1}{9+20} = \frac{2}{29}$$

Note that the word “with” in the test document is ignored because it doesn't occur in the training corpus.

Classification:

$$\hat{P}(-)\hat{P}(\text{predictable no fun} \mid -) = \frac{3}{5} \times \frac{1}{17} \times \frac{1}{17} \times \frac{1}{34} = \frac{3}{49,130}$$

$$\hat{P}(+)\hat{P}(\text{predictable no fun} \mid +) = \frac{2}{5} \times \frac{1}{29} \times \frac{1}{29} \times \frac{2}{29} = \frac{4}{121,945}$$

The model predicts class *negative* for the test review.

Why smoothing?

If we don't use smoothing, the estimates are:

$$\hat{P}(\mathit{predictable} \mid -) = \frac{1}{14} \qquad \hat{P}(\mathit{predictable} \mid +) = \frac{0}{9} = 0$$

$$\hat{P}(\mathit{no} \mid -) = \frac{1}{14} \qquad \hat{P}(\mathit{no} \mid +) = \frac{0}{9} = 0$$

$$\hat{P}(\mathit{fun} \mid -) = \frac{0}{14} = 0 \qquad \hat{P}(\mathit{fun} \mid +) = \frac{1}{9}$$

Classification:

$$\hat{P}(-)\hat{P}(\text{predictable no fun} \mid -) = \frac{3}{5} \times \frac{1}{14} \times \frac{1}{14} \times 0 = 0$$

$$\hat{P}(+)\hat{P}(\text{predictable no fun} \mid +) = \frac{2}{5} \times 0 \times 0 \times \frac{1}{9} = 0$$

Both classes have estimated probability undefined! (division by zero)

Multinomial Naive Bayes: Training

```
TrainMultinomialNB($C, D$)
 1  $V \leftarrow$ ExtractVocabulary($D$)
 2  $N_{doc} \leftarrow$ CountDocs($D$)
 3  for each $c \in C$
 4  do $N_c \leftarrow$ CountDocsInClass($D, c$)
 5     $prior[c] \leftarrow N_c / N_{doc}$
 6     $text_c \leftarrow$ ConcatenateTextOfAllDocsInClass($D, c$)
 7     for each $w \in V$
 8     do $\text{count}_{cw} \leftarrow$ CountWordOccurrence($text_c, w$)
 9     for each $w \in V$
10     do $condprob[w][c] \leftarrow \frac{\text{count}_{cw}+1}{\sum_{w'}(\text{count}_{cw'}+1)}$
11  return $V, prior, condprob$
```

Multinomial Naive Bayes: Prediction

Predict the class of a document d.

```
ApplyMultinomialNB($C, V, prior, condprob, d$)
1  $W \leftarrow$ ExtractWordOccurrencesFromDoc($V, d$)
2  for each $c \in C$
3  do $score[c] \leftarrow \log prior[c]$
4      for each $w \in W$
5      do $score[c] += \log condprob[w][c]$
6  return $\arg\max_{c \in C} score[c]$
```

Violation of Naive Bayes independence assumptions

The multinomial naive Bayes model makes two kinds of independence assumptions:

1. Word occurrences are independent within each class:

$$P(\langle w_1, \ldots, w_n \rangle | c) = \prod_{k=1}^{n} P(W_k = w_k | c)$$

2. Positional independence: $P(W_{k_1} = w | c) = P(W_{k_2} = w | c)$

These independence assumptions do not really hold for documents written in natural language.

How can naive Bayes get away with such *heroic* assumptions?

Why does Naive Bayes work?

- Naive Bayes can work well even though independence assumptions are *badly* violated.
- Example:

	c_1	c_2	predicted
true probability $P(c\|d)$	0.6	0.4	c_1
$\hat{P}(c) \prod \hat{P}(w_k\|c)$	0.00099	0.00001	
NB estimate $\hat{P}(c\|d)$	0.99	0.01	c_1

- Double counting of evidence causes underestimation (0.01) and overestimation (0.99).
- Classification is about predicting the correct class, *not* about accurate estimation.

Double counting of evidence

- Suppose the words *special* and *effects* always occur together in a movie review: either both occur in the review, or neither occurs.
- The independence assumption is badly violated!
- Let $P(special\ \ effects \mid pos) = 0.01$ and $P(special\ \ effects \mid neg) = 0.001$.
- Evidence in favor of the positive class when *special effects* occurs in a review, because probability for the positive class is 10 times as big as for the negative class.
- But naive Bayes will count this evidence twice, namely when is sees *special* and when it sees *effects*.

Naive Bayes is not so naive

- Probability estimates may be way off, but that doesn't have to hurt classification performance (much).
- Requires the estimation of relatively few parameters, which may be beneficial if you have a small training set.
- Fast, low storage requirements

Feature Selection

The vocabulary of a training corpus may be huge, but not all words will be good class predictors.

How can we reduce the number of features?

- Feature utility measures:
    - Frequency – exclude sparse terms.
    - Mutual information – select the terms that have the highest mutual information with the class label.
    - Chi-square test of independence between term and class label.
- Sort features by utility and select top k.
- Can we miss good sets of features this way?

Entropy

Entropy is the average amount of information generated by observing the value of a random variable:

$$H(X) = \sum_x P(x) \log_2 \frac{1}{P(x)} = -\sum_x P(x) \log_2 P(x)$$

We can also interpret it as a measure of the uncertainty about the value of X prior to observation.

Compare the weather forecast in the Netherlands ($P(sunny) = 0.5$, $P(rain) = 0.5$):

$$P(sunny) \log_2 \frac{1}{P(sunny)} + P(rain) \log_2 \frac{1}{P(rain)} = 0.5 \log_2 2 + 0.5 \log_2 2 = 1 \;\; bit.$$

On the Canary islands ($P(sunny) = 0.9$, $P(rain) = 0.1$):

$$0.9 \log_2 1.11 + 0.1 \log_2 10 = 0.47 \;\; bits.$$

Conditional Entropy

Conditional entropy:

$$H(X \mid Y) = \sum_{x,y} P(x,y) \log_2 \frac{1}{P(x \mid y)} = -\sum_{x,y} P(x,y) \log_2 P(x \mid y)$$

- Measure of the uncertainty about the value of X after observing the value of Y.
- If X and Y are independent, then $H(X) = H(X \mid Y)$.
- Example: gender and eye color.

Mutual Information

For random variables X and Y, their mutual information is given by

$$I(X;Y) = H(X) - H(X \mid Y) = \sum_x \sum_y P(x,y) \log_2 \frac{P(x,y)}{P(x)P(y)}$$

- Mutual information measures the reduction in uncertainty about X achieved by observing the value of Y (and vice versa).
- If X and Y are independent, then for all x, y we have $P(x,y) = P(x)P(y)$, so $I(X;Y) = 0$.
- Otherwise $I(X;Y)$ is a positive quantity, and the larger its value the stronger the association.

Estimated Mutual Information

To estimate $I(X;Y)$ from data we compute

$$I(X;Y) = \sum_x \sum_y \hat{P}(x,y) \log_2 \frac{\hat{P}(x,y)}{\hat{P}(x)\hat{P}(y)},$$

where

$$\hat{P}(x,y) = \frac{n(x,y)}{N} \qquad \hat{P}(x) = \frac{n(x)}{N},$$

and $n(x,y)$ denotes the number of records with $X = x$ and $Y = y$.
Plugging-in these estimates we get:

$$\begin{aligned} I(X;Y) &= \sum_x \sum_y \frac{n(x,y)}{N} \log_2 \frac{n(x,y)/N}{(n(x)/N)(n(y)/N)} \\ &= \sum_x \sum_y \frac{n(x,y)}{N} \log_2 \frac{N \times n(x,y)}{n(x) \times n(y)} \end{aligned}$$

Estimated Mutual Information

Mutual information between occurrence of the word "bad" in a movie review and class (negative/positive review):

bad/class	0	1	Total
0	5243	7080	12323
1	2757	920	3677
Total	8000	8000	16000

$$\begin{aligned} I(\text{bad};\text{class}) &= \frac{5243}{16000}\log_2\frac{16000\times 5243}{12323\times 8000} + \frac{7080}{16000}\log_2\frac{16000\times 7080}{12323\times 8000} \\ &+ \frac{2757}{16000}\log_2\frac{16000\times 2757}{3677\times 8000} + \frac{920}{16000}\log_2\frac{16000\times 920}{3677\times 8000} \\ &\approx 0.056 \end{aligned}$$

Fun fact: the estimated mutual information is equal to the deviance of the independence model divided by $2N$ (if we take the log with base 2 in computing the deviance).

Movie Reviews: IMDB Review Dataset

- Collection of 50,000 reviews from IMDB, allowing no more than 30 reviews per movie.
- Contains an even number of positive and negative reviews, so random guessing yields 50% accuracy.
- Considers only highly polarized reviews. A negative review has a score ≤ 4 out of 10, and a positive review has a score ≥ 7 out of 10.
- Neutral reviews are not included in the dataset.

Andrew L. Maas et al., *Learning Word Vectors for Sentiment Analysis*, Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 142–150,2011.

Data available at:

`http://ai.stanford.edu/~amaas/data/sentiment/`

Analysis of Movie Reviews in R

```
# load the tm package
> library(tm)
# Read in the data using UTF-8 encoding
> reviews.neg <- VCorpus(DirSource("D:/MovieReviews/train/neg",
                         encoding="UTF-8"))
> reviews.pos <- VCorpus(DirSource("D:/MovieReviews/train/pos",
                         encoding="UTF-8"))
# Join negative and positive reviews into a single Corpus
> reviews.all <- c(reviews.neg,reviews.pos)
# create label vector (0=negative, 1=positive)
> labels <- c(rep(0,12500),rep(1,12500))
> reviews.all
<<VCorpus>>
Metadata:  corpus specific: 0, document level (indexed): 0
Content:  documents: 25000
```

Analysis of Movie Reviews

The first review before pre-processing:

```
> as.character(reviews.all[[1]])
[1] "Story of a man who has unnatural feelings for a pig.
Starts out with a opening scene that is a terrific example
of absurd comedy. A formal orchestra audience is turned into
an insane, violent mob by the crazy chantings of it's singers.
Unfortunately it stays absurd the WHOLE time with no
general narrative eventually making it just too off putting.
Even those from the era should be turned off.
The cryptic dialogue would make Shakespeare seem easy to a
third grader. On a technical level it's better than you might
think with some good cinematography by future great Vilmos Zsigmond.
Future stars Sally Kirkland and Frederic Forrest can be seen briefly."
```

Analysis of Movie Reviews: Pre-Processing

```
# Remove punctuation marks (comma's, etc.)
> reviews.all <- tm_map(reviews.all,removePunctuation)
# Make all letters lower case
> reviews.all <- tm_map(reviews.all,content_transformer(tolower))
# Remove stopwords
> reviews.all <- tm_map(reviews.all, removeWords,
                        stopwords("english"))
# Remove numbers
> reviews.all <- tm_map(reviews.all,removeNumbers)
# Remove excess whitespace
> reviews.all <- tm_map(reviews.all,stripWhitespace)
```

Not done: stemming, part-of-speech tagging, ...

Analysis of Movie Reviews

The first review after pre-processing:

```
> as.character(reviews.all[[1]])
[1] "story man unnatural feelings pig starts opening scene terrific
example absurd comedy formal orchestra audience turned insane violent
mob crazy chantings singers unfortunately stays absurd whole time
general narrative eventually making just putting even era turned
cryptic dialogue make shakespeare seem easy third grader technical
level better might think good cinematography future great vilmos
zsigmond future stars sally kirkland frederic forrest can seen briefly"
```

Analysis of Movie Reviews

```
# draw training sample (stratified)
# draw 8000 negative reviews at random
> index.neg <- sample(12500,8000)
# draw 8000 positive reviews at random
> index.pos <- 12500+sample(12500,8000)
> index.train <- c(index.neg,index.pos)

# create document-term matrix from training corpus
> train.dtm <- DocumentTermMatrix(reviews.all[index.train])
> dim(train.dtm)
[1] 16000 92819
```

We've got 92,819 features. Perhaps this is a bit too much.

```
# remove terms that occur in less than 5% of the documents
# (so-called sparse terms)

> train.dtm <- removeSparseTerms(train.dtm,0.95)
> dim(train.dtm)
[1] 16000   306
```

Analysis of Movie Reviews

```
# view a small part of the document-term matrix
> inspect(train.dtm[100:110,80:85])

<<DocumentTermMatrix (documents: 11, terms: 6)>>
Non-/sparse entries: 7/59
Sparsity           : 89%
Maximal term length: 6
Weighting          : term frequency (tf)
Sample             :
             Terms
Docs           family fan far father feel felt
  10099_1.txt       0   0   1      0    0    0
  1033_4.txt        0   0   0      0    1    0
  10718_4.txt       0   0   0      0    0    0
  11182_3.txt       0   0   0      0    0    0
  11861_4.txt       1   0   0      0    0    0
  3014_4.txt        0   1   0      0    1    2
  315_1.txt         0   0   0      0    0    0
  6482_2.txt        0   0   0      0    1    0
  9577_1.txt        0   0   0      0    0    0
  9674_3.txt        0   0   0      0    0    0
```

Multinomial naive Bayes in R: Training

```
> train.mnb
function (dtm,labels)
{
call <- match.call()
V <- ncol(dtm)
N <- nrow(dtm)
prior <- table(labels)/N
labelnames <- names(prior)
nclass <- length(prior)
cond.probs <- matrix(nrow=V,ncol=nclass)
dimnames(cond.probs)[[1]] <- dimnames(dtm)[[2]]
dimnames(cond.probs)[[2]] <- labelnames
index <- list(length=nclass)
for(j in 1:nclass){
 index[[j]] <- c(1:N)[labels == labelnames[j]]
}

for(i in 1:V){
  for(j in 1:nclass){
    cond.probs[i,j] <- (sum(dtm[index[[j]],i])+1)/(sum(dtm[index[[j]],])+V)
  }
}
list(call=call,prior=prior,cond.probs=cond.probs)}
```

Multinomial naive Bayes in R: Prediction

```
> predict.mnb
function (model,dtm)
{
classlabels <- dimnames(model$cond.probs)[[2]]
logprobs <- dtm %*% log(model$cond.probs)
N <- nrow(dtm)
nclass <- ncol(model$cond.probs)
logprobs <- logprobs+matrix(nrow=N,ncol=nclass,log(model$prior),byrow=T)
classlabels[max.col(logprobs)]
}
```

Application of Multinomial naive Bayes to Movie Reviews

```
# Train multinomial naive Bayes model

> reviews.mnb <- train.mnb(as.matrix(train.dtm),labels[index.train])

# create document term matrix for test set
# we only extract words from the training vocabulary!

> test.dtm <- DocumentTermMatrix(reviews.all[-index.train],
               list(dictionary=dimnames(train.dtm)[[2]]))
> dim(test.dtm)
[1] 9000  306

> reviews.mnb.pred <- predict.mnb(reviews.mnb,as.matrix(test.dtm))
> table(reviews.mnb.pred,labels[-index.train])

reviews.mnb.pred    0    1
               0 3473  849
               1 1027 3651

# compute accuracy on test set: about 79% correct
> (3473+3651)/9000
[1] 0.7915556
```

Feature Selection with Mutual Information

The top-10 features (terms) according to mutual information are:

term	MI(term, class)
bad	0.056
worst	0.052
waste	0.035
awful	0.032
great	0.028
terrible	0.020
excellent	0.020
wonderful	0.018
boring	0.018
stupid	0.018

Computing Mutual Information

```
# load library "entropy"
> library(entropy)
# convert document term matrix to binary (term present/absent)
> train.dtm.bin <- as.matrix(train.dtm)>0

# compute mutual information of each term with class label
> train.mi <- apply(as.matrix(train.dtm.bin),2,
    function(x,y){mi.plugin(table(x,y)/length(y),unit="log2")},
                  labels[index.train])

# sort the indices from high to low mutual information
> train.mi.order <- order(train.mi,decreasing=T)

# show the five terms with highest mutual information
> train.mi[train.mi.order[1:5]]
       bad      worst      waste      awful      great
0.05568853 0.05161474 0.03456289 0.03168221 0.02807607
```

Using the top-50 features

```
# train on the 50 best features
> revs.mnb.top50 <- train.mnb(as.matrix(train.dtm)[,train.mi.order[1:50]],
                              labels[index.train])
# predict on the test set
> revs.mnb.top50.pred <- predict.mnb(revs.mnb.top50,
    as.matrix(test.dtm)[,train.mi.order[1:50]])

# show the confusion matrix
> table(revs.mnb.top50.pred,labels[-index.train])

revs.mnb.top50.pred    0    1
                  0 3429  996
                  1 1071 3504

# accuracy is a bit worse compared to using all features
> (3429+3504)/9000
[1] 0.7703333
```

Feature score in model

$$\text{score}(c, d) = \log \hat{P}(c) + \sum_{k=1}^{n} \log \hat{P}(w_k \mid c)$$

Score difference of word w_k in favour of positive class is

$$\log \hat{P}(w_k \mid \text{pos}) - \log \hat{P}(w_k \mid \text{neg})$$

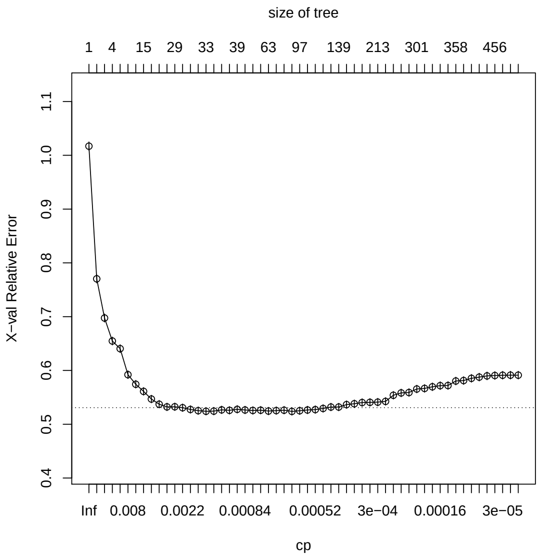
The top-20 feature score differences sorted by absolute value are:

word	score diff.	word	score diff.
waste	−2.71	poor	−1.24
awful	−2.28	loved	1.18
worst	−2.17	beautiful	0.93
terrible	−1.72	minutes	−0.91
wonderful	1.61	great	0.90
stupid	−1.55	money	−0.83
boring	−1.49	nothing	−0.81
excellent	1.41	best	0.76
bad	−1.33	performances	0.76
perfect	1.33	script	−0.74

Classification Trees

```
# load the required packages
> library(rpart)
> library(rpart.plot)
# grow the tree
> reviews.rpart <- rpart(label~.,
    data=data.frame(as.matrix(train.dtm),
    label=labels[index.train]),cp=0,method="class")
# plot cv-error of pruning sequence
> plotcp(reviews.rpart)
```

Cross-Validation Error of Pruning Sequence

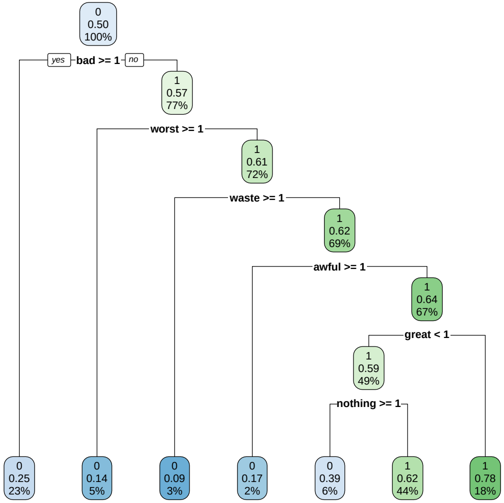


Classification Trees

```
# simple tree for plotting
> reviews.rpart.pruned <- prune(reviews.rpart,cp=5.0000e-03)
> rpart.plot(reviews.rpart.pruned)
# tree with lowest cv error
> reviews.rpart.pruned <- prune(reviews.rpart,cp=5.833333e-04)
# make predictions on the test set
> reviews.rpart.pred <- predict(reviews.rpart.pruned,
    newdata=data.frame(as.matrix(test.dtm)),type="class")
# show confusion matrix
> table(reviews.rpart.pred,labels[-index.train])
reviews.rpart.pred    0    1
                 0 3150 1021
                 1 1350 3479

# accuracy is worse than naive Bayes!
> (3150+3479)/9000
[1] 0.7365556
```

The Simple Tree



Random Forests

```
# load the required packages
> library(randomForest)

# train random forest with default settings: 500 trees and mtry = 17
> reviews.rf <- randomForest(as.factor(label)~.,
     data=data.frame(as.matrix(train.dtm),label=labels[index.train]))

# make predictions
> reviews.rf.pred <- predict(reviews.rf,newdata=data.frame(as.matrix(test.dtm)))

# show confusion matrix
> table(reviews.rf.pred,labels[-index.train])

reviews.rf.pred    0    1
              0 3483  824
              1 1017 3676

# compute accuracy: only slightly better than naive Bayes!
> (3483+3676)/9000
[1] 0.7954444
```