Learning to Classify Text via R

Paul Hofmarcher, Thomas Rusch, Wilhelm Geiger

Institute for Statistics and Mathematics
WU Vienna University of Economics and Business

Goal of this Chapter

- How can we identify particular features of language data that are salient for classifying it?
- How can we construct models of language that can be used to perform language processing tasks automatically?
- What can we learn about language form these models?
- *Tools:* Decision Tree, naive Bayes classifiers, maximum entropy classifiers.

Supervised Classification

- Choosing the correct class label for a given input.
- Classifier is *supervised* if it is build based on a training corpus containing the correct label for each input.
- Creating a classifier deciding which *features* of the input are relevant.

Generative vs. Conditional Classifiers

- Generative Classifiers predict $P(input, label)$ (e.g. Naive Bayes)
- Conditional Classifiers predict $P(label|input)$ (e.g. Maximum Entropy)

Questions that these classifiers may answer

1. What is the most likely label for an input? (G,C)
2. How likely is a given label fo a given input? (G,C)
3. What is the most likely input value? (G)
4. How likely is a given input value? (G)
5. How likely is a given input value with a given label? (G)
6. What is the most likely label for an input that might have one of two values, but we won't know? (G)

Evaluation - General

- We need to find a way to decide how well our classification model works
- Classic evaluation approach: training set, test set, evaluation set (distinct and non-overlapping)
- We know the correct labels and can assess performance of our classification model
- Evaluation set should be balanced as to which labels occur and quite general
- Size of evaluation set can be important as well
- Special cases: Cross validation and bootstrap sampling

Features

- Means something different than we would think
- We distinguish between the label l and property some f_i of input x
- f_i is called "feature" (e.g. end with letter "a")
- We need to define a combination of labels and properties ("joint-feature")

$$g(x,l) = \begin{cases} 1 & \text{if } f_i = a, (i = 1, \ldots, k) \\ 0 & \text{otherwise} \end{cases}$$

Methods - Naive Bayes Classifiers I

- Every feature gets a say in determining the label of an input value
  - Assumes independence of features ("naive")
  - Value gets label $l, l = 1, \ldots, L$ and then the k features are generated according to that label
  - We want to maximize $P(l|f_1, \ldots, f_k) \propto P(f_1, \ldots, f_k, l)$
  - Bayes theorem states that $P(f_1, \ldots, f_k, l) = P(l) \times P(f_1, \ldots, f_k|l)$
  - Under the independence assumption $P(f_1, \ldots, f_k|l) = \prod_i P(f_i|l)$
  - Choose label as max $P(f_1, \ldots, f_k, l)$
  - Calculation of $P(f_i|l)$ should be calculated via smoothing techniques
  - Non-binary features can be binned or expanded to Dummy coding or metric values can be regressed

Gender Identification I

- Creating Classifier, deciding which *features* of the input are relevant, e.g., just looking at the final letter.

  ```
  > feature_last_letter <- function(x,number=1)
  {unlist(lapply(strsplit(x,""), tail,number))}
  ```

- Split data into *training, dev, prediction* set.
- Training set is used to train via *naive Bayes* classifier—`library("e1071")`.
- dev-set is used for error analysis (contains 600 names).
- prediction set for label forecasting.

Gender Identification II

```
> bayes <- naiveBayes(sex~letter, data=test_data)
> sex_predict <- predict(bayes,dev_data[,-1])
```

- check classification hit-ratio and misclassified names (149 names).
- refine extractor function to improve hit-ratio, based on misclassifications observed in *dev* set.

Gender Identification III

```
> 1-length(which(check!=sex_predict))/length(dev_index)
[1] 0.7516667
```

```
> missclass_names
        names   letter  sex       predicted_sex
1480    hunter       r   f             m
1086     karen       n   f             m
1721 jacquelyn       n   f             m
376      kelly       y   m             f
367       kane       e   m             f
1207    eileen       n   f             m
```

Gender Identification IV

```
> table(twoletters)
 ah  al  am  an  ar  at  ay  be  by  ce  de  dy  ee  el  en  er
  6   1   1   9   1   1   2   1   1   4   1   5   1   4  10   6
 et  ex  ey  ge  he  id  ie  in  is  ke  ly  my  nd  ne  ni  nn
  1   1   6   2   1   1   4   5   1   1   1   2   2   6   2   6
 on  ow  oy  pe  re  ri  ry  se  te  th  ty  ue  us  vi  yn
  6   1   1   1   4   2   3   2   5   2   1   2   1   1  12
```

- names ending in `yn` tend to be female, despite the fact that names ending in `n` tend to be male.
- refine extractor function to improve hit-ratio, based on misclassifications observed in *dev* set.

Document Classification I

- Pang Lee MovieReviews Corpus tm.corpus.MovieReviews
- Define feature extractor, so classifier will know which words it should pay attention.
- 1000 positive and 1000 negative movie reviews.
- Limit number of features (words) to 2000 and define feature extractor that simply checks whether word is in document or not.
- For counting the words, we can use `DTM` from tm, maybe "`Binary(DTM)`".

Document Classification II

- instead of `nltk.NaiveBayesClassifier`, `train` from caret, or `naiveBayes??`

```
tail(pos_neg_ratios, 9) ## positive keywords
 friendship    surround   courag    paxton    howard
        6.0         6.0      6.0       6.0       7.0
    segment      balanc    castl    outstand
        7.0         7.0      11.0      22.0


head(pos_neg_ratios, 8) ## negative keywords
patch    schumach   idiot    lame   seagal
0.1250   0.1250    0.1363   0.1500  0.2000
    failur       worst    stupid
    0.2000      0.2200    0.2439
```

Methods - Decision trees I

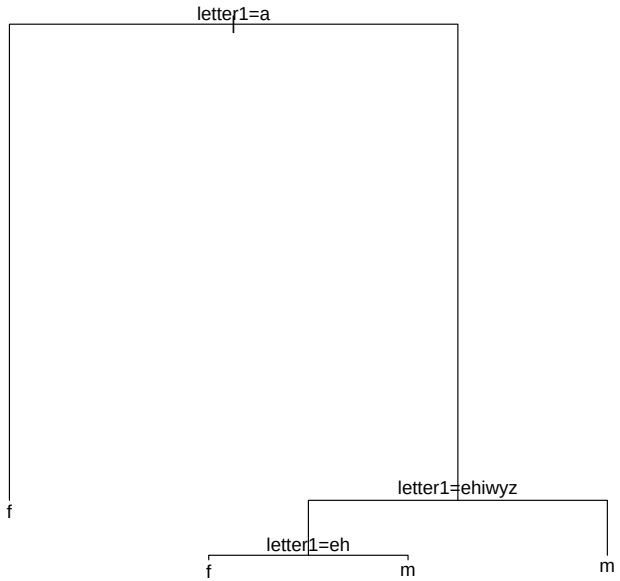
- Trees are a simple and intuitive form of additive models
- They partition the feature space into disjoint rectangles and fit a constant in every partition
- All observations in a terminal node are classified to belong to the majority category
- Binary Trees: Idea and Visualization
  - One starts with the whole feature space of inputs
  - That one is split into two regions based on a split point of a *single* input
  - The split point and variable is chosen to optimize a fit criterion
  - The procedure is then repeated recursively in each region until a stopping rule applies

Methods - Decision trees II

- Possible objective functions
  - Gini index
  - Deviance
  - misclassification error
  - Entropy or information gain
- Avoid overfitting via Pruning
  - CV based pruning (misclassification rate, deviance, entropy or Gini index)
  - Cost-complexity pruning

Methods - Decision trees III

```
> t1 <- rpart(sex ~ letter1, data = trainData, control = rpart.control(cp = 0))
> plot(t1)
> text(t1)
```



Methods - Maximum Entropy Classifiers

- Look for a set of parameters $\{\theta_j\}$ that maximize total likelihood of the training corpus, where $P_\theta(x,l) \propto P_\theta(l|x) = \frac{1}{C(\theta,l)} exp(\sum_j \theta_j g_j(x,l))$
- Features can interact
- Iterative Optimization (BFGS or CG)
- Similar to Naive Bayes, but all feature-label combinations (joint-features) can be used (e.g. one label many features or vice versa)
- ME models tries to capture frequencies of individual joint-features without making strong assumptions
- Maximum entropy principle: from all candidate distributions (i.e. reflect our knowledge) of joint-features choose the distribution with highest entropy $H = \sum_l P(l) \times \log_2 P(l)$
- Statisticians call this a log-linear model

Part-of-Speech Tagging I

- Choose a part-of-speech tag for a word by looking at the internal makeup of the word.
- in R openNLP provides functions for part-of-speech tagging.

```
> sentence <- "They may have thought it was the end
of the crisis.
It turned out to be more like the beginning."

> tagPOS(sentence, language="en")
[1] "They/PRP may/MD have/VB thought/VBN it/PRP was/VBD
the/DT end/NN of/IN the/DT crisis./NN It/PRP turned/VBD
out/RP to/TO be/VB more/JJR like/IN the/DT beginning./NN"
```

Part-of-Speech Tagging II

- Train a classifier to work out which suffixes are most informative.
- Train *Decision Tree* classifier, based on the suffixes.
  - "," → ","
  - "the" → "Determiner"
  - "s" → "verb'
    - "is" → "BEZ"
  - "if not"→ "noun"

Part-of-Speech Tagging – Exploiting Context

- When tagging the word *fly*, knowing the previous word was *a*, *fly* should be classified as a noun not as a verb.
- E.g., classifier should learn that a word is likely to be a noun if it comes after the word *large*.
- Simple classifiers always treat each input as independent from all other inputs.

Part-of-Speech Tagging – Sequence Classification

- To captue dependencies between related classification tasks –*joint classifier*. They choose an appropriate labeling for a collection of related inputs.
- *Greedy sequence classification* is to find the most likely class label for the first input. Given this class label find the best label for the next input...(This approach was taken by the bigram tagger (Section 5.5))
  - Feature extractor function requires `history` argument.
  - Each tag in histopry corresponds with a word in the sentence.
  - Training is done with annotated corpus.

Other Methods for Sequence Classification

- Note: We commit every decision that we make. E.g., if we label a word as a noun, but later find evidence that it should be a verb, we can't change it.
- One solution is to adopt a transformed strategy instead. These classifiers work by creating initial assignment of labels of inputs and then iteratively refining the assignment *Brill tagger*.
- Another solution is to assign scores to all possible sequences of part-of-speech tags and choose the sequence with the highest overall score *Hidden Markov Models* $\rightarrow$ number of possible tag sequences is quite large.
- Maximum Entropy Markov Models, Linear-Chain Conditional Random Field Models.

Sentence Segementation

- Classification task for punctuation.
- Whenever we encounter a symbol that could possibly and a sentence we have to decide whether it terminates the preceding sentence.
- openNLP provides the function `sentDetect`

```
> sentence <- "They may have thought it was the end of the crisis.
 It turned out to be more like the beginning."
> sentDetect(sentence)
[1] "They may have thought it was the end of the crisis. "
[2] "It turned out to be more like the beginning."
```

Sentence Segmentation

`sentDetect` for dialog act types:

```
> dialog <- "A: How did you get that horrible swelling on your nose?
 B: I bent down to smell a brose.
 A: There isn't a B in rose.
 B: There was in this one!"
> sentDetect(dialog)
[1] "A: How did you get that horrible swelling on your nose?\n"
[2] "B: I bent down to smell a brose.\n"
[3] "A: There isn't a B in rose.\n"
[4] "B: There was in this one!"
```

Sentence Segmentation

`sentDetect` for dialog act types:

```
> dialog <- "A: How did you get that horrible swelling on your nose?
 B: I bent down to smell a brose...
 A: There isn't a B in rose.
 B: There was in this one!"
> sentDetect(dialog)
[1] "A: How did you get that horrible swelling on your nose?\n"
[2] "B: I bent down to smell a brose...\nA: There
    isn't a B in rose.\n"
[3] "B: There was in this one!"
```

Sentence Segmentation

```
> tokenize(sentence)
 [1] "They"     " "        "may"      " "        "have"     " "
 [7] "thought"  " "        "it"       " "        "was"      " "
[13] "the"      " "        "end"      " "        "of"       " "
[19] "the"      " "        "crisis"   "."        " "        "It"
[25] " "        "turned"   " "        "out"      " "        "to"
[31] " "        "be"       " "        "more"     " "        "like"
[37] " "        "the"      " "        "beginning" "."
```

Evaluation - Cross validation

- Especially useful if only limited amount of data is available
- Idea
  - Divide original data set A into N subsets (folds) $A_i, \ldots, A_n$ with $\bigcup_i A_i = A$ and $\bigcap_i A_i = \emptyset$
  - For each $i, (i = 1, \ldots, N)$ train the model on all $A \setminus A_i$
  - Evaluate model on the set A_i
  - Combine the results for each of the N evaluation sets

Evaluation - Bootstrap Sampling

- Training set: Sample n rows from a $n \times k$ data myslide with replacement
- Evaluation set: All rows that were not in the training set ("out-of-bag")
- Train the model on the training set
- Evaluate it on the evaluation set
- Repeat as often as desired (e.g. 10 times)

Evaluation - Assessing performance I

Confusion matrix: Matrix that displays how often label i was predicted as label $j (i, j = 1, \ldots, k)$

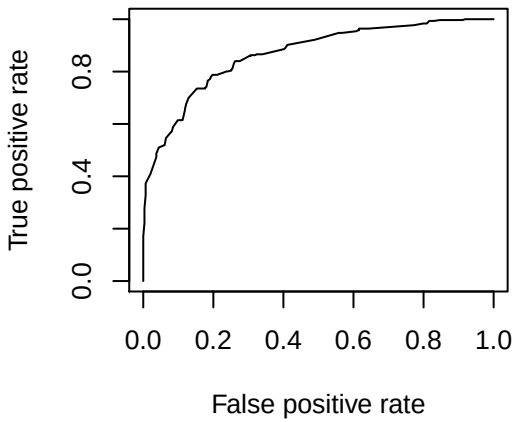
```
gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.

gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.
```

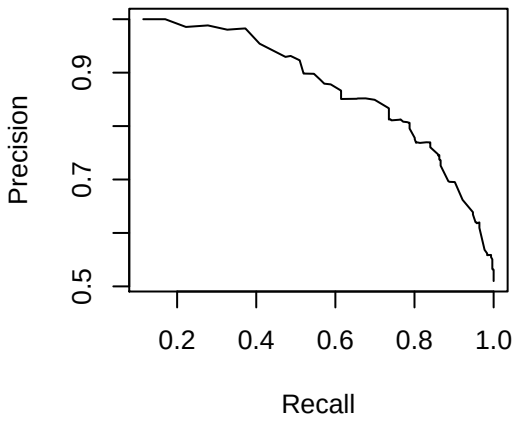
```
> confusionMatrix(sexPredictNB, check)
Confusion Matrix and Statistics

          Reference
Prediction   f   m
         f 225  44
         m  81 250

               Accuracy : 0.7917
                 95% CI : (0.7569, 0.8235)
    No Information Rate : 0.51
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.5842
 Mcnemar's Test P-Value : 0.001282

            Sensitivity : 0.7353
            Specificity : 0.8503
         Pos Pred Value : 0.8364
         Neg Pred Value : 0.7553
             Prevalence : 0.5100
         Detection Rate : 0.3750
   Detection Prevalence : 0.4483

       'Positive' Class : f
```

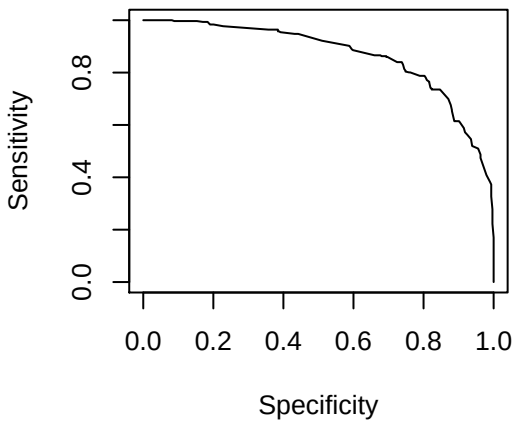
Evaluation - Assessing performance II

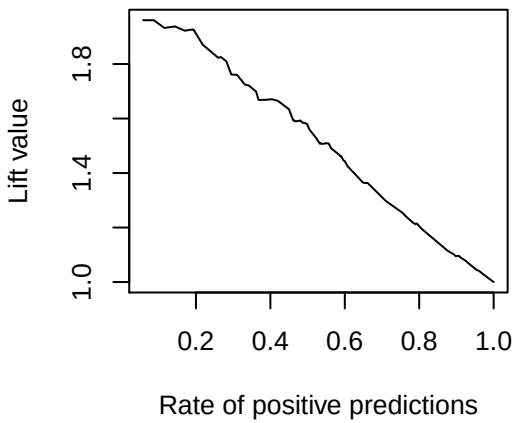
Performance visualisation

- ROC curve: true positive vs. false positive rate
- Precision/recall graph: Precision vs. recall
- Sensitivity/specifity plot: Sensitivity vs. specificity
- Lift charts: Lift vs. rate of positive predictions


True positive rate
0.0
0.4
0.8
0.0
0.2
0.4
0.6
0.8
1.0
False positive rate


Precision
0.5
0.7
0.9
0.2
0.4
0.6
0.8
1.0
Recall


Sensitivity
0.0
0.4
0.8
0.0
0.2
0.4
0.6
0.8
1.0
Specificity


Lift value
1.0
1.4
1.8
0.2
0.4
0.6
0.8
1.0
Rate of positive predictions

Evaluation - Assessing performance III

Some performance measures

- Accuracy. $P(\hat{Y} = Y)$
- Error rate. $P(\hat{Y} \neq Y)$
- False positive rate (fallout) $P(\hat{Y} = \oplus | Y = \ominus)$
- True positive rate (recall, sensitivity). $P(\hat{Y} = \oplus | Y = \oplus)$
- False negative rate (miss). $P(\hat{Y} = \ominus | Y = \oplus)$
- True negative rate (specificity). $P(\hat{Y} = \ominus | Y = \ominus)$
- Positive predictive value (precision). $P(Y = \oplus | \hat{Y} = \oplus)$
- Negative predictive value. $P(Y = \ominus | \hat{Y} = \ominus)$
- Prediction-conditioned fallout. $P(Y = \ominus | \hat{Y} = \oplus)$
- Prediction-conditioned miss. $P(Y = \oplus | \hat{Y} = \ominus)$

Evaluation - Assessing performance III

More performance measures

- Rate of positive predictions. $P(\hat{Y} = \oplus)$
- Rate of negative predictions. $P(\hat{Y} = \ominus)$
- Phi correlation coefficient (Matthews correlation). $\frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FN) \cdot (TN+FP) \cdot (TP+FP) \cdot (TN+FN)}}$
- Mutual information. $I(\hat{Y}, Y) := H(Y) - H(Y|\hat{Y})$, where H is the (conditional) entropy.
- Chi square test statistic.
- Odds ratio. $\frac{TP \cdot TN}{FN \cdot FP}$
- Lift value. $\frac{P(\hat{Y}=\oplus | Y=\oplus)}{P(\hat{Y}=\oplus)}$

Evaluation - Assessing performance IV

Behold! we found some more performance measures

- Precision-recall break-even point. The cutoff(s) where precision and recall are equal.
- Calibration error (absolute difference between predicted confidence and actual reliability).
- Mean cross-entropy $MXE := -\frac{1}{P+N}(\sum_{y_i=\oplus} ln(\widehat{y}_i) + \sum_{y_i=\ominus} ln(1-\widehat{y}_i))$
- Root-mean-squared error $RMSE := \sqrt{\frac{1}{P+N}\sum_i (y_i - \widehat{y}_i)^2}$
- SAR = 1/3 * (Accuracy + Area under the ROC curve + Root mean-squared error)
- Expected cost
- Cost of a classifier when class-conditional misclassification costs are explicit

Methods - Other

- Possibly many more
- Highly predictive or modern methods
- Improve POS with them (e.g. random forests)
- R infrastructure is clearly an advantage here

tm.classify plugin I

- Feature extractor
  - generic, object-oriented, flexible (i.e. work on letter, word, token, sentence, context, sequence, textual basis etc.)
  - `fe_control=list(...)`
  - to work with tm objects (vcorpus, dtm, etc.)
  - outputs inputs for the classification function
  - ...

tm.classify plugin II

- Classification
  - Uses the feature extractor output object
  - Exploit the R infrastructure
  - Support for lists of training and test sets and many classification methods
  - We thought of a simplified version of `train()` from "caret"
  - outputs training and test predictions, classification model etc.
  - `predict()` method for new data
  - ...

tm.classify plugin III

- Utility functions
  - Support for CV and BS sampling
  - Variable importance
  - Visualisation
  - Preformance measures
  - Reuse ROCR and caret functions
  - ...
- POS taggers perhaps?