Statistical Learning, Fall 2014

Assignment 2: Classification

The purpose of this exercise is to compare the Naive Bayes (NB) model to Logistic Regression (LR) on data with categorical attributes. We will first train/fit a NB and a LR model on some data set and then compare their performance by looking how well they predict in terms of 0/1-loss (misclassification rate, error percentage). We will do this on a real-world data set, the *congressional voting records*. The data set can be downloaded from the UCI Machine Learning repository, `http://archive.ics.uci.edu/ml/datasets.html`.

Data. The goal is to predict whether a member of US Congress is democrat or republican, depending on his/her voting record in 1984. All attributes are categorical. The data set has *missing values*. You can choose an arbitrary standard statistical algorithm (for instance the na.roughfix() function from randomForest library) or invent your own method for dealing with them. Note, however, that you need to stick to the same method for both NB and LR and you are required to explain whatever method you pick in a few sentences in the report.

Methods. Naive Bayes maximum likelihood estimation has been implemented in the *e1071* package of the TU Vienna. You will need to install this package to use Naive Bayes. See the manual pages for examples of its use. Our course book (Hastie, Tibshirani, Friedman) is rather short on Naive Bayes; consult wikipedia and/or other web sources if you need more information. Logistic regression conditional likelihood maximization can be performed via the *glm* (generalized linear model)-command. For an example of its use, check out the mini-tutorial *Logistic Regression in R* by Christopher Manning, available from `http://nlp.stanford.edu/manning/courses/ling289/logistic.pdf`.

Assignment. Please do all the following experiments and summarize the results in a written report that includes the required graph and answers to all of the questions below.

Experiments. Randomly split the data in two equal parts; the first part constitutes the training data, the second part is the test data. Suppose that the training data consists of say N vectors. Now, do the following for each $n = 1 \dots N$:

1. Train a NB model on the first n data vectors in the training data. Do this (a) with Laplace smoothing, assuming $\alpha = 1$, i.e. one "fake" observation per each class. This means that, for each class y (e.g. $y \in \{\text{democrat}, \text{republican}\}$) and each feature (attribute) X_j , the probability that X_j has value k given that the class is y is set to $(n_{k,y} + \alpha)/(n_y + \alpha K_j))$, where $n_{k,y}$ is the number of data vectors where the feature has value k and the class is y, n_y is the number of data vectors with class y, and K_j is the number of values that feature X_j can take. Do this also (b) with Laplace smoothing for a value of α close to 0, say 0.1 (so that the estimated parameters are very close to maximum likelihood) (c) also use $\alpha = 10$. Now use the three inferred models to predict the test set and measure loss using the 0/1-loss function.

2. Train a LR model on the first n samples in the training set. Use the inferred model to do prediction on the test set with the 0/1-loss.

3. Train a LR model on the first n samples in the training set, regressing only on the first 2 attributes. Use the inferred model to predict the test set and evaluate it with the 0/1-loss.

As a result of this procedure, you should have obtained $N \times 5$ 0/1-errors: for each sample size of the training set, you learned 5 predictors (3 NB, 2 LR) and tested them on the full test set.

Now make a plot of these five 0/1-errors as a function of $n = 1 \dots N$. Include the plot in your report and analyze its meaning. To get a smooth plot, you may want to repeat the whole procedure many (say $M = 10$) times and average the results; each time, you start with a random permutation of all the samples, putting the first half into training and second half into test set. Then you learn and predict for n from 1 to N, ending up with a new $N \times 5$ matrix. Therefore, you will obtain M of these matrices. Make a new final $N \times 5$ matrix with, for all i, j, entry (i, j) equal to the average of all the (i, j)-entries of the M matrices you constructed. If this takes too much computation time, you make skip some values of n and interpolate in the graph.

Is one of the five methods the overall winner? Are some of the methods generally undesirable? Is what you see in accordance with what you would have expected from the theoretical insights you have learned in the course? If so, explain. If not, what is different from what you expected? In what way does overfitting play a role in the results? (hint: also consider the training error for the logistic regression model, i.e. the 0/1-loss you make if you predict the training set rather than the test set with the model fit on the training set). In what way does the training error differ for the various versions of NB and LR? How extreme (close to 0 or 1) are the predictive probabilities based on NB and LR?

General Remarks

1. Note that the corresponding *predict*-functions in R work differently for NB and LR. Check out page 6 of Mannings tutorial to get an idea about how to compute the 0/1-performance and how to use *predict* with LR. Check out the *e1071* documentation to find out how to use *predict* with NB. Finally, recall that the first column in the dataset should not be treated as a feature, e.g. you may use:
   `NBtest <- predict(NB, test[,-1], type='class')`.

2. For **bonus points**, you may want to add a final curve in the graph based on using penalized rather than standard logistic regression. This cannot be implemented with the standard *glm*-function though and you will need to install additional package(s).

3. Please submit your R code and report (in pdf format) to `tim@timvanerven.nl`.