Department of Statistics
Lund University

Data Mining
Fall

Project 2: Splines, generalized additive models, classification trees

Generalized additive model, B-splines

Perform all requested task. Your work will be monitored and the credit for it will be given based on your in-lab activities. Some useful R-code that can help in completing Project 2 can be found at CANVAS.

Part One – Smooth spline fitting a generalized additive model with one predictor

We consider the following `Wage` data set taken from the simpler version of the main textbook: *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani. The purpose in analyzing this data set is to examine a number of factors that relate to wages for a group of males from the Atlantic region of the United States. In particular, we wish to understand the association between an employees age and education, as well as the calendar year, on his wage. It has been preliminarily observed that age increases with age until about 60 years of age, at which point it begins to decline. There is also a slow but steady increase of approximately \$10, 000 in the average wage between 2003 and 2009 and, on average, wage increases with the level of education.

This time we will download and use the R-package that has been created to accompany that textbook. There is an extensive collection of R-packages that can be utilized when working with R, if you are interested, see the R-Project website for more details.

1. Load the package *"ISLR"* to R system (see the R-code that can be fetched **here**). Activate `Wage` data file. Check what variable it contains and what is the size of this data set.

2. Plot `wage` variable vs. `age` variable and comment what kind of features this data set presents.

3. Download and activate the package `"splines"` and perform the following sequence of commands

```
agelims=range(age)
age.grid=seq(from=agelims[1],to=agelims[2])

fit=lm(wage~bs(age,knots=c(25,40,60)),data=Wage)
pred=predict(fit,newdata=list(age=age.grid),se=T)
plot(age,wage,col="gray")
lines(age.grid,pred$fit,lwd=2)
```

```
lines(age.grid,pred$fit+2*pred$se,lty="dashed")
lines(age.grid,pred$fit-2*pred$se,lty="dashed")
```

Explain what tasks have been performed.

4. In the presented approach we have specified explicitly knots. Alternatively, one can consider the degree of freedom as well as using the higher order than cubic splines and we can even use different splines than the B-splines (the normal splines in the code below). The following code is performing these tasks

```
dim(bs(age,knots=c(25,40,60)))
dim(bs(age,df=6))

attr(bs(age,df=6),"knots")

fit2=lm(wage~ns(age,df=4),data=Wage)

pred2=predict(fit2,newdata=list(age=age.grid),se=T)

lines(age.grid, pred2$fit,col="red",lwd=2)
```

Compare fits using degrees of freedom versus specified knots. What knots have been chosen when the degree of freedom method was applied?

5. Next we turn to the smooth splines.

```
title("Smoothing Spline")
fit=smooth.spline(age,wage,df=16)
fit2=smooth.spline(age,wage,cv=TRUE)
fit2$df
lines(fit,col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
legend("topright",legend=c("16 DF","6.8 DF"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```

Notice that in the first call to `smooth.spline()`, we specified `df=16`. The function then determines which value of λ leads to 16 degrees of freedom. In the second call to `smooth.spline()`, we select the smoothness level by cross-validation; this results in a value of λ that yields 6.8 degrees of freedom. Judging from the fit which of these two methods you prefer and why?

Part Two – Smooth spline fitting a generalized additive model

This time we fit the response based on all three predictors: `age`, `year`, `education`. For the first two we use the smoothed splines f_1 and f_2 with four and five degrees of freedom, respectively, while for the third predictor which is categorical we use piecewise constant function. We use uploaded software with `gam()` function in R to fit GAMs using smoothing splines, via backfitting as described in the lecture. This method fits a model involving multiple predictors by repeatedly updating the fit for each predictor in turn, holding the others fixed.

1. Install "gem" R-package and learn about gem() function using the help features of R. What has happened by executing the following command?

   ```
   gam.m3=gam(wage~s(year,4)+s(age,5)+education,data=Wage)
   ```

   Use summary(gam.m3) and try to explain as much as you can understand from the resulting listing.

2. The following code show how one can make predictions based on the model

   ```
   preds=predict(gam.m3,newdata=Wage)

   dev.off()

   plot(1:length(preds),wage-preds)
   ```

   Explain what each line of the codes is performing. Comment on what you see in the final plot.

Part Three – Discriminating using classification tree

We will work in this example with the data set Carseats that is included in the library ISLR (see the associated R-code to see how to make these data accessible in R).

Here is a brief description of the data set (available also by executing the command: help(Carseats). The data provides information related to sales of child car seats at 400 different stores. The data is kept in the data frame format (see R manual if you want to learn more about this format, however to complete this project you will not use anything more than common sense). The data frame contains 400 observations on the following 11 variables.

'Sales' Unit sales (in thousands) at each location

'CompPrice' Price charged by competitor at each location

'Income' Community income level (in thousands of dollars)

'Advertising' Local advertising budget for company at each location (in thousands of dollars)

'Population' Population size in region (in thousands)

'Price' Price company charges for car seats at each site

'ShelveLoc' A factor with levels 'Bad', 'Good' and 'Medium' indicating the quality of the shelving location for the car seats at each site

'Age' Average age of the local population

'Education' Education level at each location

'Urban' A factor with levels 'No' and 'Yes' to indicate whether the store is in an urban or rural location

'US' A factor with levels 'No' and 'Yes' to indicate whether the store is in the US or not

We use classification trees to analyze the Carseats data set.

1. In these data, Sales is a continuous variable, and so we begin by recoding it as a binary variable. Use the `ifelse()` function (you can use the command `help()` to find out more about this function) to create a variable, called `High`, which takes on a value of `Yes` if the Sales variable exceeds 8, and takes on a value of `No` otherwise.

2. Use the `data.frame()` function to merge `High` with the rest of the Carseats data.

3. Download and activate the package `"tree"`. Read about the facilities of this package **here**. Use R-help feature to get some general information about the function `tree()` – do not need to read all the details.

4. Build the binary tree for the classification variable `High`. Do not include the sales variable as the input variable! – Why? Note in the code how you exclude the variable. The `summary()` function gives you some information on the built tree. Check which variables are used as internal nodes in the tree, the number of terminal nodes, and the (training) error rate. What is the training error rate? Please, experiment with the way explanatory variables (features) can be specified in function `tree()`.

5. One of the most attractive properties of trees is that they can be graphically displayed. Display the tree structure, and the node labels. Take the 2nd, 10th, and 50th store and classify their sales as `High` or `Not High` using the obtained regression tree. Compare this to the actual value of *High* for these stores. Comment about effectiveness of the qualifier. Suppose that you have a new store that reports the following values: `98,60,5,80,100,"Bad",35,6,"No","Yes"`. What would be the prediction of the sales for this store?

6. Type the name of the obtained tree object and check that in the printout you have obtained: the split criterion (e.g. Price < 92.5), the number of observations in that branch, the deviance (for the moment, we do not discuss this parameter), the overall prediction for the branch (Yes or No), and the fraction of observations in that branch that take on values of Yes and No. Branches that lead to terminal nodes are indicated using asterisks. Relate the obtained printout to the tree graph obtained earlier? Would you be able to plot the tree if only this information is available to you?

7. In order to properly evaluate the performance of a classification tree on these data, we must estimate the test error rather than simply computing the training error. We split the observations into a training set and a test set, build the tree using the training set, and evaluate its performance on the test data. The `predict()` function can be used for this purpose. What is the prediction rate for the test data? How it compares with the rate for the training data? Is it surprising?

8. Run resampling of the testing data (**do not use `set.seed(2)` function in the loop!**) to obtain a number of estimate of the prediction rate and average them. Is the average prediction rate comparable with the original one? What is the variability of the prediction rate?

9. Next, we consider pruning the tree. Use help facility in R to check on `prune.misclass` function. In particular, note the following parameters (in the notation of the lecture α is represented here by `k`).

`k`: cost-complexity parameter defining either a specific subtree of 'tree' ('k' a scalar) or the (optional) sequence of subtrees minimizing the cost-complexity measure ('k' a vector). If missing, 'k' is determined algorithmically.

`best`: integer requesting the size (i.e. number of terminal nodes) of a specific subtree in the cost-complexity sequence to be returned. This is an alternative way to select a subtree to supplying a scalar cost-complexity parameter 'k'. If there is no tree in the sequence of the requested size, the next largest is returned.

Consider a simple 3 nod classification tree and check its performance (classification rate).

10. We check now if cross-validation might lead to improved results. The function `cv.tree()` performs cross-validation in order to determine the optimal level of tree complexity; cost complexity pruning is used in order to select a sequence of trees for consideration. The `cv.tree()` function reports the number of terminal nodes of each tree considered (size) as well as the corresponding error rate and the value of the cost-complexity parameter used (`k`, which corresponds to α). Note that, despite the name, `dev` corresponds to the cross-validation error rate in this instance. What are conclusion about the optimal size for the tree?

11. Plot the error rate as a function of both the subgraph size and α. Are these graphs somehow related? How?

12. Obtain the pruned tree based on the results of your cross-validation study.

13. Run the classification based on the test data using the pruned tree and compare the results to the original results. Is the new tree easier to interpret? Is the new tree as effective as the non-pruned one?

Part Four – regression tree

We will work in this example with the data set `VolatilityData.csv` available in our Data repository in CANVAS. We perform the steps of analysis as outlined in the introductory lecture.

1. Read the data to R.

2. Devide data into

   - Training sample - the one on which we will learn something: 50% of the data.
   - Validating sample - the one on which the choice of the method will be validated: 25% of the data.
   - Testing sample - the one on which the chosen method will be evaluated: 25% of the data.

3. Make a simple linear regression fit based on the training sample. Evaluate the prediction mean squared error (PMSE) of the fit.

4. Evaluate the fit using the validating sample by computing the prediction mean squared error obtained for this sample. Compare the two PMSEs. Which one is bigger? Is it expected?

5. Plot the fit and comment on the found relation between the volatility and returns.

6. Using the package `tree` build a rich regression tree over the data. Plot the tree.

7. Plot the regression fit over the data. Does it provide a good fit? Compute the PMSE of this fit.

8. Compute the PMSE using the validating sample. Compare all the PMSEs.

9. Next, we consider pruning the tree. Perform the trimming using `prune.tree` function in the package. Choose the smallest standard deviation (PMSE) version of the trimmed trees and plot the resulting tree.

10. Plot the regression tree fit over the data. Does it seems to be more reasonable?

11. Compare the PMSE with the previous ones.

12. Regress the data against the quadratic function (parabol). Do the same steps as in the case of regressing against the linear function (line). Is the fit better?

13. Use the testing sample to evaluate PMSE for all three fit to the data (linear, regression tree, quadratic). Which one preforms the best?