Tutorial 1: Kernel Density Estimation: Lab

The command `density` is contained in the basic `stats` package and does not need further installation. Computation of UCV, BCV, SJ window width estimators are found in the **MASS** package:

```
> install.packages("MASS")
```

if you haven't already done this.

Exercise 1: Old Faithful Geyser Data The Old Faithful geyser data comes with the R installation. The various window widths are computed as follows (activate **MASS** first).

```
ucv(geyser$duration)
bcv(geyser$duration)
width.SJ(geyser$duration)
width.SJ(geyser$duration, method = "dpi")
```

Here 'dpi' stands for 'direct plug in'. The default method is 'ste' which stands for 'solve the equation'.

To get a density estimate plot for `geyser$durations`, use (for example):

```
d <- density(geyser$duration, ucv(geyser$duration))
plot(d)
```

- Make a histogram of the `duration` variable.
- Now try kernel density estimation, using a gaussian kernel. Which of the methods (UCV, BCV, Sheather-Jones plug in) gives the largest magnitude of the left hand mode compared with the right?

Exercise 2: Claw Density This is a good practical test of the efficiency of density estimators. We generate random numbers from a density with spikes - and find out the size of sample that is necessary to detect the spikes.

Generate n observations from the *claw density*

$$p(x) = 0.5N(0,1) + (0.1)\sum_{k=0}^{4} N\left(\frac{k}{2} - 1, (0.1)^2\right)$$

and estimate that density using a kernel density estimator. Take $n = 100, 200, 300$ and 1000. Compare performances of UCV, BCV, SJPI window-width estimators for each simulation. Which window-width estimation best fits the claws?

Notes `rnorm` generates samples of normal observations (check it using help). `sample` generates a random sample from a defined set of values. Check the syntax and see how to generate numbers from $0, 1, 2, 3, 4, 5$ with probabilities $0.1, 0.1, 0.1, 0.1, 0.1, 0.5$ respectively. The `ifelse` command may be useful for generating from the claw density.

R Commands Recall that normal random numbers are generated by `rnorm`; look it up by typing `?rnorm`. You can (for example) generate 50 observations from a $N(0, 1)$ and put them into column x by typing

```
x<-rnorm(50,0,1)
```

For the exercise, generate n random numbers from $\{0.1, 2, 3, 4, 5\}$ with probabilities

$$(0.1, 0.1, 0.1, 0.1, 0.1, 0.5).$$

This may be done as follows:

```
> x<-c(0,1,2,3,4,5)
> y<-sample(x,50,replace=TRUE,c(0.1,0.1,0.1,0.1,0.1,0.5))
```

`x` is the vector of values to be sampled from, we want a sample size of 50, the probabilities are listed in the last argument (look up `?sample`). One way to get the claw density is:

```
> z<-rnorm(50,0,1)
> z1<-ifelse(y<5,(y/2)-1+0.1*z,z)
```

Here `z` is a sample of 50 $N(0, 1)$ observations; the `ifelse` gives two options. Firstly, the random number for y is $0, 1, 2, 3$ or 4 on the 'if' side and if $y = 5$ on the 'else' side.
For $k = 0, 1, 2, 3, 4$, you generate an observation from $N(\frac{k}{2} - 1, (0.1)^2)$ (standard deviation is 0.1) and for $k = 5$ you generate an observation from $N(0, 1)$.

Exercise 3: Hidalgo Stamp Data Apply the kernel density estimator with kernel corresponding to Rosenblatt's density estimator to the 1872 Hialgo Stamp Data. (This is the rectangular kernel; $K(x) = \mathbf{1}_{[-\frac{1}{2},\frac{1}{2}]}(x)$). The data is found in `Hidalgo1872.xls` in the course data directory. Alternatively, it is in the `Hidalgo1872.rda` file, which is an R file.

What do you notice about the smoothness of the resulting density estimate?

Exercise 4: U.S. Highway Data The `ushighways` data (found in `ushighways.txt` in the course data directory) consists of the approximate length (in miles) of all 212 U.S. 3-digit interstate highways (spurs and connectors). Compare the kernel density estimates for these data using UCV, BCV and SJPI window-width estimators.

Exercise 5: Kernel regression In the standard formulation of the regression problem, the mean of a response variable Y is related to one or more covariates $X_1, \ldots, X_d$ through some function $f(.)$, which needs to be estimated. That is,

$$Y = f(X) + \epsilon$$

where f is a systematic component and ϵ a random error.

Generating Data We generate data in which the underlying regression function is given by

$$f(x) = 3\sin(2x) + 10(x-5)\mathbf{1}(x > 5),$$

where $\mathbf{1}$ is the indicator function. This is a sine function, with a kink at the point $x = 5$, after which a steep linear component is added.

We take the *error* or *residual* distribution as:

$$T + (G-1)(X-5)^2 + 3)$$

where $T \sim t_3$ and $G \sim \Gamma(2,2)$. We sample 5000 pairs where $\frac{X}{10} \sim \text{Beta}(2,2)$.

This can be done using the following code. The package **FKSUM** is very useful for projection pursuit. Install this package and see what it does. In particular, look up the `fk_sum` command. Here `fk` stands for *fast kernel*.

```
library("FKSUM")
set.seed(1)
n <- 5000
x <- rbeta(n, 2, 2) * 10
fx <- 3 * sin(2 * x) + 10 * (x > 5) * (x - 5)
y <- fx + rt(n, 3) + (rgamma(n, 2, 2) - 1) * ((x - 5)^2 + 3)
xeval <- seq(0, 10, length = 1000)
par(mfrow = c(1, 2))
for (h in c(0.25, 0.05)) {
    plot(x, y, xlab = "x", ylab = "y")
    fhat <- fk_sum(x, y, h, x_eval = xeval) / fk_sum(x, rep(1, n),
                                                     h, x_eval =
xeval)
      lines(xeval, fhat, col = 2, lwd = 2)
      lines(xeval, 3 * sin(2 * xeval) + 10 * (xeval > 5) * (xeval -
5),
```

```
            col = 3, lwd = 2)
}
```

The true function is shown in green, and the estimates in red. The larger bandwidth oversmooths and does not accurately capture the local extrema of the function (left). On the other hand, the smaller bandwidth is accurate over much of the range, but exhibits increased variation in the tails,
Now we apply the `fk_regression` function and see what it does. We take two choices of bandwidth - both of which give better estimators. The routine used is the Nadaraya-Watson estimator.

```
par(mfrow = c(1, 2))
plot(fk_regression(x, y, type = "NW"))
lines(xeval, 3 * sin(2 * xeval) + 10 * (xeval > 5) * (xeval - 5),
        +           col = 3, lwd = 2)
plot(fk_regression(x, y, type = "NW", h = "cv"))
lines(xeval, 3 * sin(2 * xeval) + 10 * (xeval > 5) * (xeval - 5),
        + col = 3, lwd = 2)
```

Exercise 6: Projection Pursuit, Components of a Density We start with an $n \times d$ data matrix $\mathbf{x}$, which consists of n independent instantiations of a d-variate random vector. We'll take $d = 4$ and $n = 2000$.

Recall the idea of *projection pursuit* is to find interesting directions, were 'interest' is measured in terms of a *projection index* $I(p)$.

We'll use the implementation in the package **FKSUM** where FK stands for 'fast kernel'. The projection index here will be entropy (so we're looking for directions as close as possible to Gaussian - which maximises entropy).

We take successive rays orthogonal to each other.

As indicated in the lecture, we start by 'whitening' the data matrix (i.e. make it mean 0 with covariance matrix I by:

$$\mathbf{x} \leftarrow (\mathbf{x} - \underline{\widehat{\mu}}\mathbf{1}')\widehat{\Lambda}^{-1/2}\widehat{U}$$

where $\underline{\widehat{\mu}}$ denotes the column averages, $\widehat{\Lambda}$ is the diagonal matrix containing the estimated eigenvalues of the covariance matrix (listed from highest to lowest) and $\widehat{U}$ is the orthonormal matrix of eigenvectors of the covariance matrix.

The routine we follow does not use the iterative fitting sequence described in the lectures. Rather, the one-dimensional marginals along a direction $\mathbf{p} = \mathbf{x}w$ (where w is a unit d-vector) is:

$$f_{\mathbf{p}}(x) = \frac{1}{nh}\sum_{j=1}^{n} K\left(\frac{p_i - x}{h}\right)$$

for $x \in \mathbb{R}$, $p_i = \sum_{j=1}^{d} \mathbf{x}_{ij} w_j$.

The aim is to find a suitable $d \times d'$ projection matrix $\mathbf{W}$, where d' represents the number of components of interest. The first component maximises the entropy.

Subsequent components maximise the entropy subject to the constraint of being orthogonal to the components that have already been computed. We restrict the projection matrix $\mathbf{W}$ to matrices of the form $\mathbf{W} = \Lambda^{-1/2} U Q$ where Λ is diagonal, with the eigenvalues of the sample covariance matrix (so that $\Lambda = I$ after whitening), U is a $d \times d\prime$ matrix whose columns are the first d' eigenvectors of C_Z (so that $U = I$ after whitening) and Q is $d' \times d'$ orthonormal. Effectively, after whitening, we look for $\mathbf{W}$ in the space of orthonormal matrices.

The **ProDenICA** package conveniently gives functions for simulating data from some standard densities and also computes the *Amari* distance.

We consider the following formulation of *projection pursuit*. Let $\mathbf{x}$ be an $n \times d$ data matrix, which is considered to be n instantiations of a random d-vector. We are trying to find the *projection matrix* $\mathbf{W}$

Generate a set of data with independent components. Next, apply a randomly generated linear transformation to the data and apply various ICA (independent component analysis) methods.

Sample 2000 points with 4 components. Within each experiment, take a random combination of the 18 densities produced by the `rjordan()` function, which are listed according to the letters `a` to `r`.

The following produces a data set where,

- Firstly, we generate 2000 independent random 4-vectors. Each component from the 4-vector comes from a different probability distribution. These are the 4 components which we try to estimate. This is the 2000×4 matrix X.
- We then randomly mix them, using a 4×4 mixing matrix, which we denote R. to obtain $Xx = XR$. The matrix Xx is our data matrix. The independent *components* are in the matrix X.
- We now try to *recover* these components.

We use the function `fk_ICA()` (fast kernel Independent Component Analysis) to estimate the independent components.

```
library(ProDenICA)
set.seed(1)
X <- matrix(0, 2000, 4)
```

```
densities <- sample(letters[1:18], 4)
for (i in 1:4) X[,i] <- rjordan(densities[i], 2000)
R <- mixmat(4)
Xx <- X %*% R
model <- fk_ICA(Xx, 4)
par(mfrow = c(4, 3), mar = c(0, 0, 3, 0))
for (i in 1:4) {
    plot(fk_density(Xx[,i]), main = paste("Mixed component", i),
           xaxt = "n", yaxt = "n")
    plot(fk_density(model$S[,i]), main = paste("Estimated component",
i),
           xaxt = "n", yaxt = "n")
    plot(fk_density(X[,i]), main = paste("True component", i),
           xaxt = "n", yaxt = "n")
}
```

You should find that under 'mixed component' there doesn't seem to be much that is discernible. The observations for 'mixed component' are the mixtures.

You should also find that the four *estimated* components do correspond quite well to the four *true components*, although they won't appear in the order listed (so that, for example, the first estimated component may correspond to the third true component).

Find out which densities are available under `rjordan`; you can try this with several choices of input densities, do a mixture and see how well the routine recovers the true components.