Projektaufgaben Block 2

Carlo Michaelis, 573479; David Hinrichs, 572347; Lukas Ruff, 572521

06 Dezember 2016

1 Nichtparametrisches Testen

1.1 Zwillingsstudie

Um zu testen, ob der Kindergartenbesuch einen signifikanten Einfluss auf die sozialen Fähigkeiten eines Kindes hat, führen wir einen zweiseitigen t-Test und einen zweiseitigen Wilcoxon-Vorzeichen-Test, jeweils zum Signifikanzniveau $\alpha = 0.05$, durch:

```
# Enter data
x <- c(82,69,73,43,58,56,76,65)
y <- c(63,42,74,37,51,43,80,62)

# Two-sided t-test
t.test(x, y, alternative = "two.sided", mu = 0, conf.level = 0.95, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  x and y
## t = 2.3791, df = 7, p-value = 0.04895
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   0.05320077 17.44679923
## sample estimates:
## mean of the differences
##                    8.75
```

```
# t.test(x-y, alternative = "two.sided", mu = 0, conf.level = 0.95)  # alternative

# Two-sided Wilcoxon signed rank test
wilcox.test(x, y, alternative = "two.sided", mu = 0, conf.level = 0.95,
            paired = TRUE, conf.int = TRUE)
```

```
##
##  Wilcoxon signed rank test
##
## data:  x and y
## V = 32, p-value = 0.05469
## alternative hypothesis: true location shift is not equal to 0
## 95 percent confidence interval:
##  -0.5 19.0
## sample estimates:
## (pseudo)median
##           7.75
```

```
# wilcox.test(x-y, alternative = "two.sided", mu = 0, conf.level = 0.95,
#              conf.int = TRUE)  # alternative
```

Wir können sehen, dass der t-Test die Nullhypothese ablehnt und somit einen signifikanten Einfluss feststellt. Der Wilcoxon-Vorzeichen-Test verwirft die Nullhypothese dagegen nicht. Durch die Normalverteilungsannahme $X_i - Y_i \sim N(0, \sigma^2)$, die für gegebenes Sample in Frage gestellt werden kann, besitzt der t-Test eine größere Power. Der nichtparametrische Wilcoxon-Vorzeichen-Test benötigt hingegen keine Verteilungsannahme, besitzt jedoch eine kleinere Power.

1.2 t-Test vs. Wilcoxon-Vorzeichen-Test

```r
fnTestPowerMC <- function(fnError, n = 30, theta = 0, alpha = 0.05, nSim = 10^4,
                          ...) {
  # This function estimates the probability of rejecting the null hypothesis of a
  # t-test and a Wilcoxon signed rank test using Monte Carlo simulations of iid
  # random variables X_i = \theta + \epsilon_i.
  #
  # Args:
  #   fnError:  Function which generates random samples from a symmetric error
  #             distribution \epsilon
  #   n:        Number of random samples used in tests
  #   theta:    True parameter
  #   alpha:    Significance level used in tests
  #   nSim:     Number of MC simulations of size n
  #   ...:      Further arguments to be passed to fnError
  #
  # Returns:
  #   A list containing the following elements:
  #     $TProb:       MC estimation of rejection probability for the t-test
  #     $WilcoxProb:  MC estimation of rejection probability for the Wilcoxon
  #                   signed rank test

  # Perform MC simulation
  matX <- matrix(theta + fnError(n*nSim, ...), ncol = n)

  # Define sub-functions which only return p-values from the two tests
  fnPvalT <- function(x, theta) {
    return(t.test(x, mu = theta)$p.value)
  }
  fnPvalWilcox <- function(x, theta) {
    return(wilcox.test(x, mu = theta)$p.value)
  }

  # Perform nSim number of tests with sample size n for each of the two tests
  vecPvalT <- apply(matX, 1, fnPvalT, theta = theta)
  vecPvalWilcox <- apply(matX, 1, fnPvalWilcox, theta = theta)

  # Compute and return estimations of rejection probabilities
  result <- list()
  result$TProb <- mean(vecPvalT < alpha)
  result$WilcoxProb <- mean(vecPvalWilcox < alpha)
  return(result)
}
```

```r
# Set seed
set.seed(42)
```

```
# Normal errors
fnTestPowerMC(fnError = rnorm, nSim = 10^5, theta = 0)
```

```
## $TProb
## [1] 0.04999
##
## $WilcoxProb
## [1] 0.04981
```

```
# Cauchy errors (t-distribution with df = 1)
fnTestPowerMC(fnError = rt, nSim = 10^5, df = 1)
```

```
## $TProb
## [1] 0.02034
##
## $WilcoxProb
## [1] 0.04947
```

```
# Uniform errors
fnTestPowerMC(fnError = runif, nSim = 10^5, min = -1, max = 1)
```

```
## $TProb
## [1] 0.05177
##
## $WilcoxProb
## [1] 0.05073
```

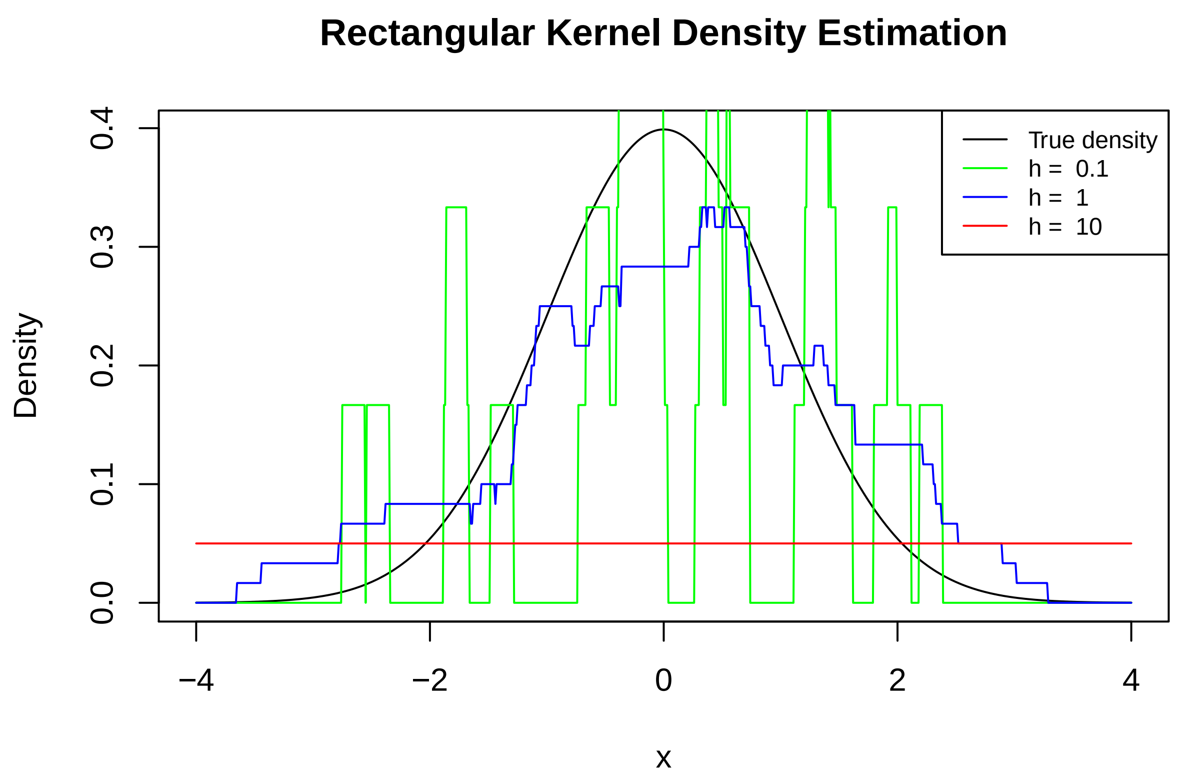
2 Dichteschätzung

2.1 Kerndichteschätzer

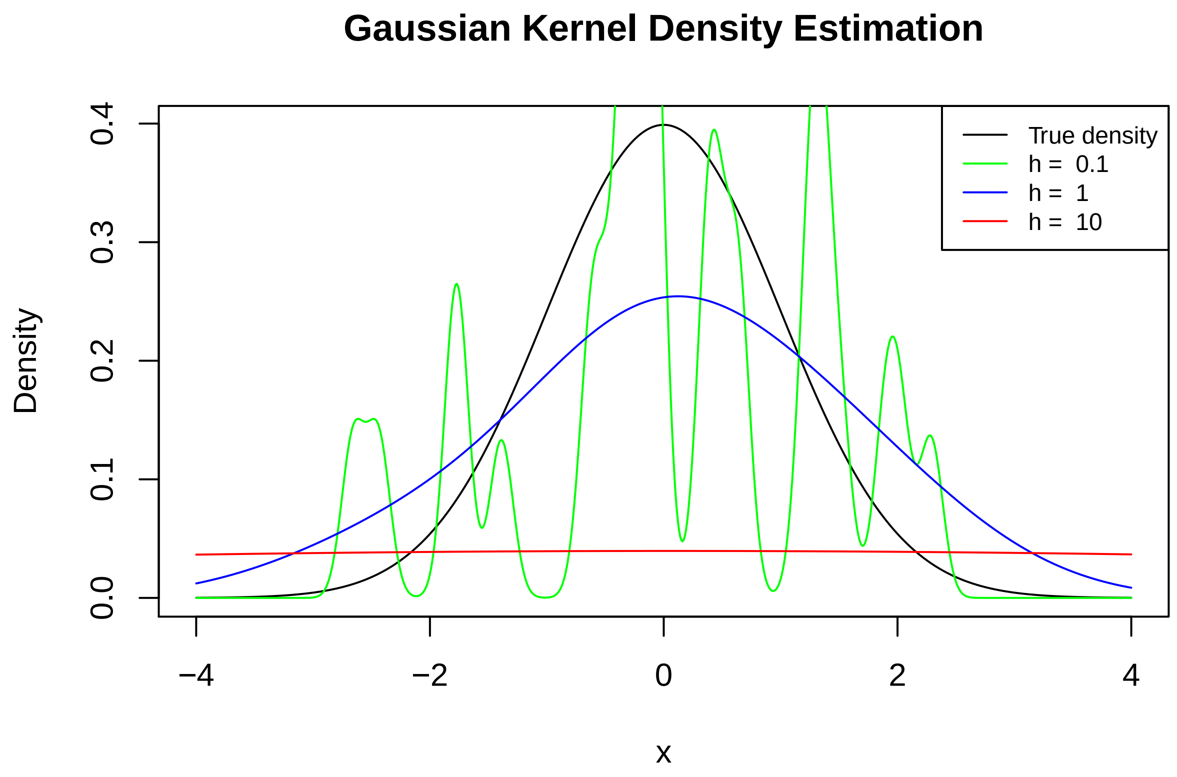
```
fnKernelDensityEst <- function(x, X, K, h) {
  # This function computes the kernel density estimates for a given sample X and
  # kernel K with bandwidth h at points x.
  #
  # Args:
  #   x: Points for which the kernel density estimates are computed
  #   X: Data sample on which the kernel density estimation is fitted
  #   K: Kernel function to be used for smoothing
  #   h: Smoothing Bandwidth
  #
  # Returns:
  #   A vector of the kernel density estimates at the points x

  # Get number of points and sample size
  nPts <- length(x)
  n <- length(X)

  # Compute and return the estimated kernel density values
  matKernel <- K((matrix(rep(X, nPts), ncol = n, byrow = TRUE) - x) / h)
  return((1/(n*h)) * apply(matKernel, 1, sum))
}
```

```r
# Define different smoothing kernels

fnRectangularKernel <- function(x) {
  return(0.5 * (abs(x) <= 1))
}

fnGaussianKernel <- function(x) {
  return((1/sqrt(2*pi)) * exp((-0.5) * x^2))
}

fnEpanechnikovKernel <- function(x) {
  return(0.75 * (1 - x^2) * (abs(x) <= 1))
}
```

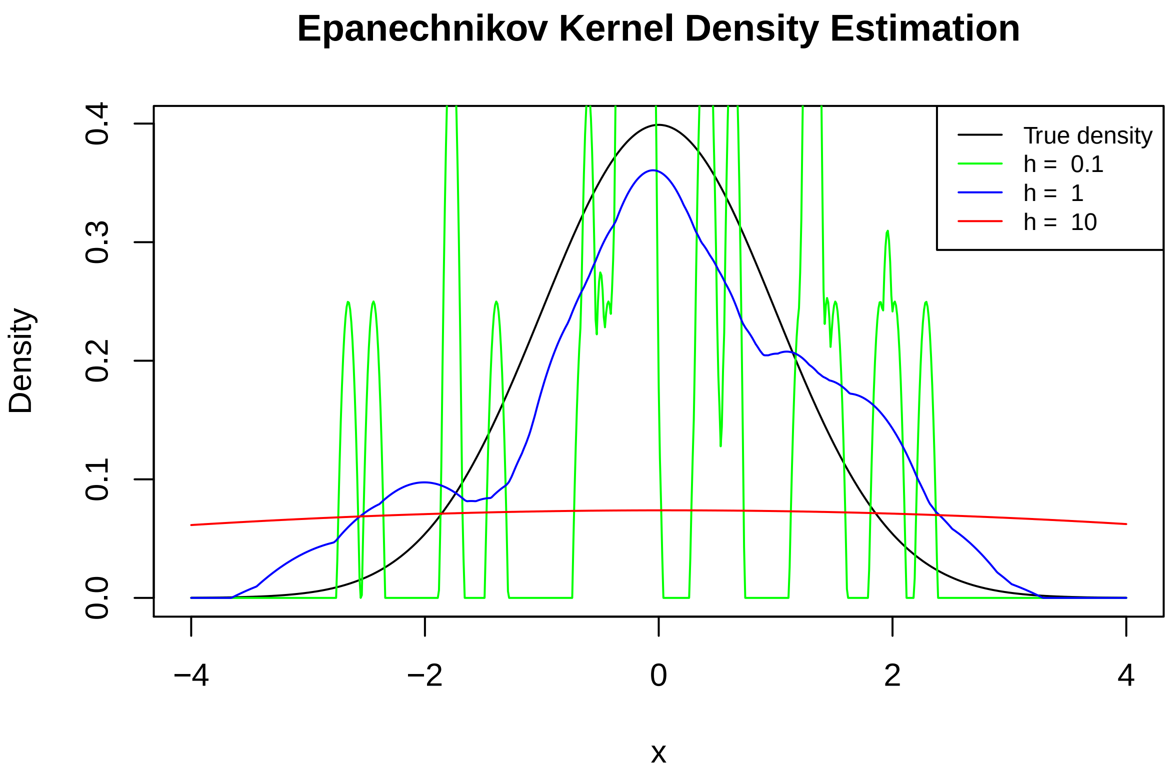
```r
# Test 1: Standard Normal ######################################################

# Sample size and generate sample
set.seed(42)
n <- 30
X <- rnorm(n)

# Define points to estimate kernel density on
x <- seq(-4, 4, 0.01)

# Set different bandwidths
h <- c(0.1, 1, 10)

# Rectangular kernel density estimation
plot(x, dnorm(x), type = "l", main = "Rectangular Kernel Density Estimation",
     xlab = "x", ylab = "Density")
lines(x, fnKernelDensityEst(x, X, K = fnRectangularKernel, h[1]), col = "green")
lines(x, fnKernelDensityEst(x, X, K = fnRectangularKernel, h[2]), col = "blue")
lines(x, fnKernelDensityEst(x, X, K = fnRectangularKernel, h[3]), col = "red")
legend("topright", legend = c("True density", paste("h = ", h[1]),
                              paste("h = ", h[2]), paste("h = ", h[3])),
       col = c("black", "green", "blue", "red"), lty = 1, cex = 0.75)
```



```
# Gaussian kernel density estimation
plot(x, dnorm(x), type = "l", main = "Gaussian Kernel Density Estimation",
     xlab = "x", ylab = "Density")
lines(x, fnKernelDensityEst(x, X, K = fnGaussianKernel, h[1]), col = "green")
lines(x, fnKernelDensityEst(x, X, K = fnGaussianKernel, h[2]), col = "blue")
lines(x, fnKernelDensityEst(x, X, K = fnGaussianKernel, h[3]), col = "red")
legend("topright", legend = c("True density", paste("h = ", h[1]),
                              paste("h = ", h[2]), paste("h = ", h[3])),
       col = c("black", "green", "blue", "red"), lty = 1, cex = 0.75)
```



```
# Epanechnikov kernel density estimation
plot(x, dnorm(x), type = "l", main = "Epanechnikov Kernel Density Estimation",
     xlab = "x", ylab = "Density")
lines(x, fnKernelDensityEst(x, X, K = fnEpanechnikovKernel, h[1]), col = "green")
lines(x, fnKernelDensityEst(x, X, K = fnEpanechnikovKernel, h[2]), col = "blue")
lines(x, fnKernelDensityEst(x, X, K = fnEpanechnikovKernel, h[3]), col = "red")
legend("topright", legend = c("True density", paste("h = ", h[1]),
                              paste("h = ", h[2]), paste("h = ", h[3])),
       col = c("black", "green", "blue", "red"), lty = 1, cex = 0.75)
```



2.2 Kreuzvalidierung zur Bandweitenwahl

3 Bildentrauschen