Midterm 02 - STAT440

Joseph Sepich (jps6444)

10/23/2020

```
set.seed(42)
```

Problem 1 Nelder-Mead

Part a

There are two steps here we need to justify. First there is an equality, then there is an approximation. The equality is simply an expansion of the Kenel Density Estimate function $\hat{f}_\sigma(x)$. Since $\frac{1}{n}$ is a constant and we can sum integrals we pull those two terms out of the integral to get that middle function.

The approximation is done via our Monte Carlo integration. The integral we have in the summation is actually in the form of an expectation $\int g(x)f(x)dx$. In our case the $g(x) = k_\sigma(x, X_i)$ and the f(x) is our true density. Since we have samples $\{X_i\}_{i=1}^n$ from the true density $f(x)$ we can therefore use these samples in our Monte Carlo integration.

In case I did not explain enough the actual integral estimation term is the $\frac{1}{n-1}\Sigma_{j \neq i} k_\sigma(X_j, X_i)$, which is merely the expected value of our KED functoin by summing up the sample points and dividing by the sample size, which is follows the expected value equation. The beginning part of the term $(\frac{1}{n}\Sigma_{i=1}^n)$ comes from the first step mentioned above expanding $\hat{f}_\sigma(x)$.

Part b

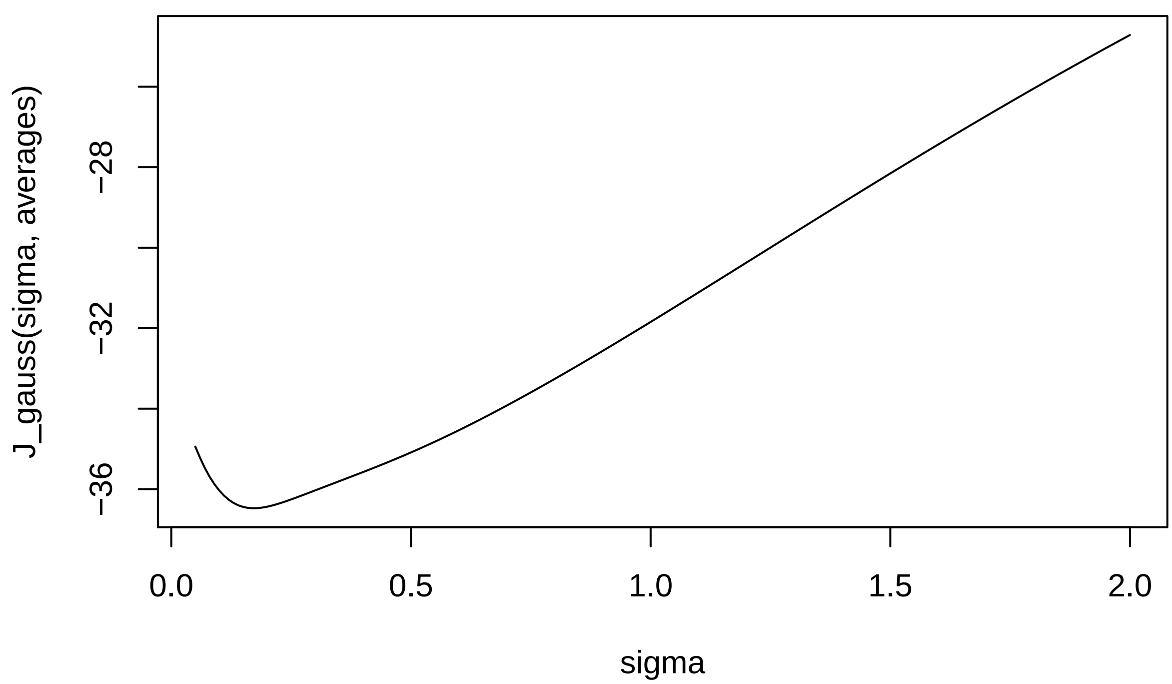
```
# gaussian kernel
k_gauss <- function(x, x_prime, sigma=1) {
    dnorm(x, mean=x_prime, sd=sigma)
}

# error function J
J_gauss <- function(sigma, X) {
    n <- length(X)
    first_term <- 0
    for (i in 1:n) {
        for (j in 1:n) {
            first_term <- first_term + k_gauss(X[i], X[j], sqrt(2)*sigma)
        }
    }
    first_term <- first_term * (1 / n^2)
    second_term <- 0
    for (i in 1:n) {
        inner <- 0
        for (j in 1:n) {
            if (j != i) {
                second_term <- second_term + k_gauss(X[j], X[i], sigma)
            }
```

```r
        }
        second_term <- second_term + (1 / (n - 1)) * inner
    }
    second_term <- second_term * (2 / n)
    return(first_term - second_term)
}
```

```r
# load dataset
scores <- data.matrix(read.csv('./data/score.csv'))
averages <- rowMeans(scores) / 10
averages[0:10]
```

```
##  [1] 9.04 9.00 8.86 7.92 7.82 7.80 7.62 7.62 7.54 7.30
```

```r
sigma <- seq(0.05, 2, 0.01)
plot(sigma, J_gauss(sigma, averages), type="l")
```



Part c

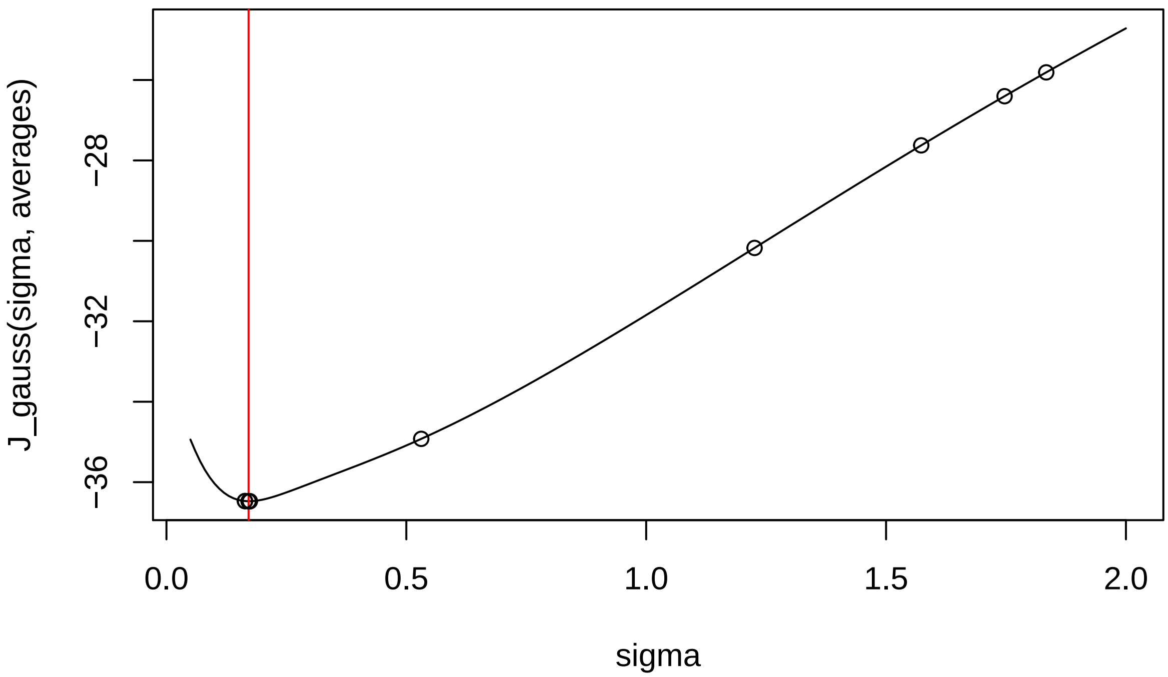
The error curve above is a convex function along this curve. You can draw a line between any two points on this interval that will always be above the line. A minimiziation algorithm should find the minimum here, since there are no local extrema to draw the algorithm away from the global minimum.

Part d

```r
my_loss <- function(sigma) {
    J_gauss(sigma, averages)
}

nm_gauss_kde <- function(par=c(0.05, 2), fn = my_loss, return_points=FALSE) {
    # parameters
    alpha <- 1
    gamma <- 2
    rho <- 0.5
    sigma <- 0.5

    # termination criteria
    max_term <- 100
    sd_threshold <- 0.001

    dim <- 2
    points <- runif(dim, min=par[1], max=par[2])

    point_mat <- NA

    iter <- 1
    while(sd(points) >= sd_threshold & iter <= max_term) {
        # Order points
        points <- points[order(fn(points))]
        if (return_points) {
            if (iter == 1) {
                point_mat <- matrix(points, nrow=1, ncol=dim)
            } else {
                point_mat <- rbind(point_mat, matrix(points, nrow=1, ncol=dim))
            }
        }

        # Compute centroid - this is merely the first point (2 points only)
        centroid <- points[1:dim-1] / (dim-1)

        # Reflect about centroid
        reflected_point <- centroid +  alpha * (centroid - points[dim])
        if (reflected_point < 0) {
            reflected_point <- -1 * reflected_point
        }
        val_r <- fn(reflected_point)
        # don't need to check reflected
        # criteria; if better than x_n but not better than x_1; isn't possible
        # since x_1 = x_n here

        # Expand step
        if (val_r < fn(points[1])) {
            expanded <- centroid + gamma * (reflected_point - centroid)
            if (expanded < 0) {
                expanded <- -1*expanded
            }
```

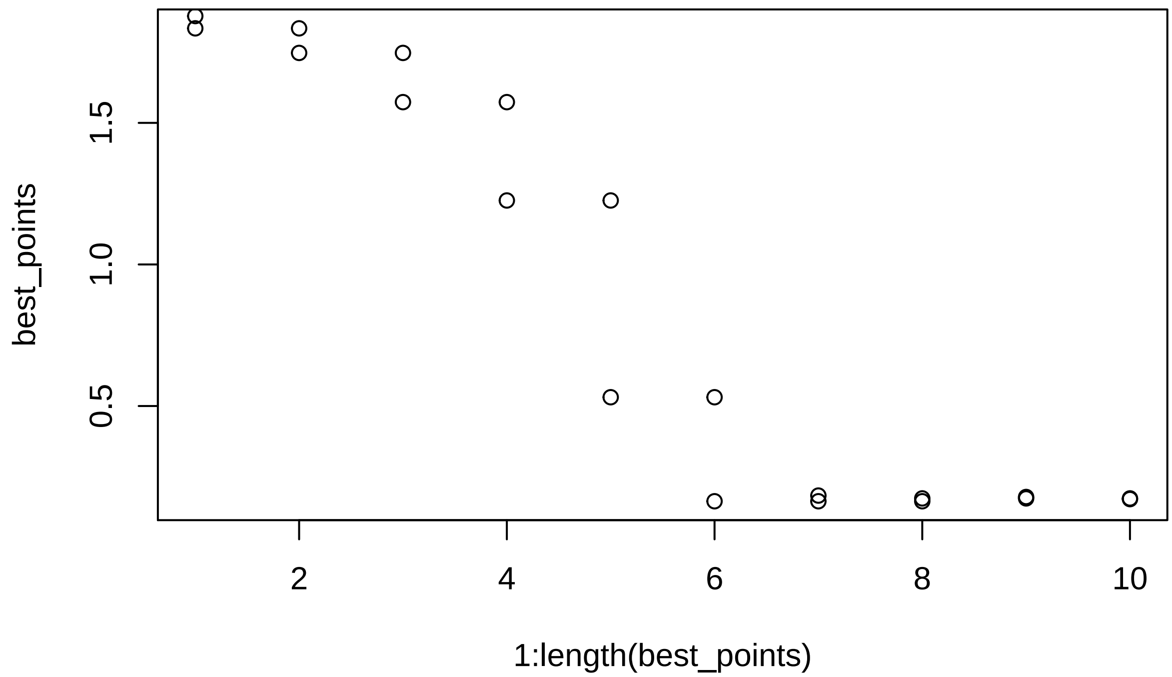
```r
            if (fn(expanded) < val_r) {
                points[dim] <- expanded
            } else {
                points[dim] <- reflected_point
            }
        } else {
            contracted <- centroid + rho * (reflected_point - centroid)
            if (contracted < 0) {
                contracted <- -1*contracted
            }
            if (fn(contracted) < fn(points[dim])) {
                # contract
                points[dim] <- contracted
            } else {
                # shrink
                points <- points[1] + sigma * (points - points[1])
            }
        }
        # next iteration
        iter <- iter + 1
    }
    if (return_points) {
        return(point_mat)
    }
    return(points[1])
}
```

```r
points <- nm_gauss_kde(return_points = TRUE)
best_points <- points[,1]
plot(sigma, J_gauss(sigma, averages), type="l")
abline(v=points[length(best_points),1], col="red")
points(best_points, J_gauss(best_points, averages))
```



Part e

```
plot(1:length(best_points), best_points)
points(1:length(best_points), points[,2])
```



In the first portion, from steps 1 through about 5, the step is almost always a reflect and expand. During thi part the algorithm is walking down the large slope to the right of the minimum. After this the reflect and contract happens, so we do not go past 0.05, and then the algorithm pretty much converged through shrinking or some step that is too small to distinguish on this graph.

Problem 2 Permutation Test

```
weather <- read.csv('./data/Szeged_Weather_Summary.csv')
weather[1:5,]
```

```
##   Year Month Temperature ApparentTemp  Humidity WindSpeed WindBearing
## 1 2006     1 -1.67428315    -4.170818 0.8346505  8.902067    160.7769
## 2 2006     2 -0.06128472    -2.986136 0.8433929 10.958278    197.8914
## 3 2006     3  4.53346792     1.940272 0.7786541 14.416716    195.0619
## 4 2006     4 12.62587191    12.083819 0.7295278 10.926802    192.1569
## 5 2006     5 15.66531511    15.555600 0.7209677 10.201839    208.9583
##   Visibility  Pressure
## 1   7.900988 1021.1935
## 2   7.421214  995.2108
## 3   9.577225  976.3727
## 4  10.626760 1013.4965
## 5  11.748066 1016.6247
```

Part a

Null Hypothesis:

$$H_0 : \rho = 0$$

Alternative Hypothesis:

$$H_A : \rho \neq 0$$

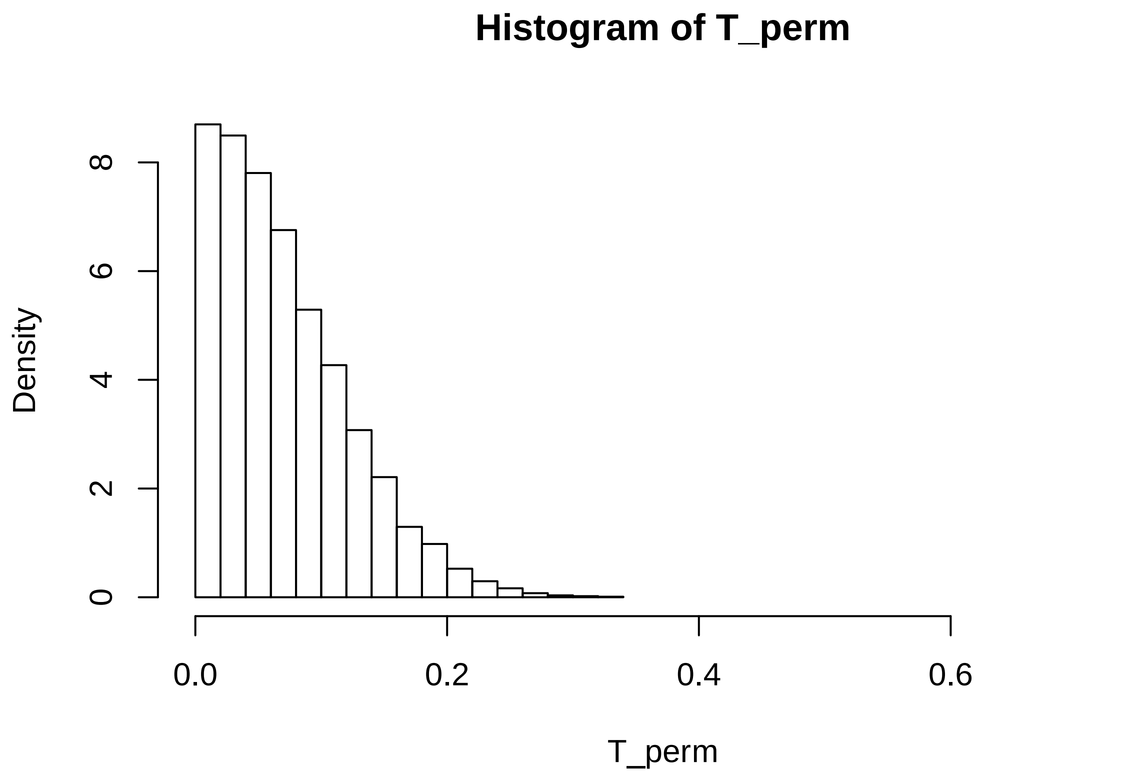
We can use the following test statistic:

$$T = |r - \rho_0| = |r|$$

This test statistic we are using here translates to the absolute value of the sample correlation coefficient between Temperature and Humidity. Our null hypothesis states that ρ_0 is zero, which is why we only need $|r|$. Our extreme set would be test statisics that are greater than our observed test statistic T_{obs} where $T_{obs} = |r_{obs}|$. Explicitly our extreme set is $T \geq T_{obs}$. We can calculate our P value by calculating the proportion of test statistics that fall within our extreme set.

Part b

```r
K <- 10000

T_obs <- abs(cor(weather$Temperature, weather$Humidity))

T_perm <- vector('numeric', K)
temp_df <- weather
for (i in 1:K) {
    temp_df$Humidity <- sample(weather$Humidity)
    T_perm[i] <- abs(cor(temp_df$Temperature, temp_df$Humidity))
}

x_end <- max(c(T_obs, max(T_perm)))
hist(T_perm, freq=FALSE, xlim = c(0,x_end))
abline(v=T_obs, col='red')
```




```
print(mean(T_perm >= T_obs))
```

```
## [1] 0
```

We would reject the null hypothesis here with a p-value of zero in our permutation test.

Problem 3 Cross-Validation

```
# size of data
n <- nrow(weather)
# error caches
error1 <- numeric(n)
error2 <- numeric(n)
error3 <- numeric(n)

for (i in 1:n) {
    # create models
    m1_i <- lm(Temperature~Humidity, data=weather[-i,])
    m2_i <- lm(Temperature~WindSpeed, data=weather[-i,])
    m3_i <- lm(Temperature~Humidity+WindSpeed, data=weather[-i,])

    # predicted values
```

```
    y1 <- predict(m1_i, weather[i,])
    y2 <- predict(m2_i, weather[i,])
    y3 <- predict(m3_i, weather[i,])

    # compute squared error
    error1[i] <- (weather$Temperature[i]- y1)^2
    error2[i] <- (weather$Temperature[i]- y2)^2
    error3[i] <- (weather$Temperature[i]- y3)^2
}

print(mean(error1))
```

```
## [1] 29.74187
```

```
print(mean(error2))
```

```
## [1] 52.57002
```

```
print(mean(error3))
```

```
## [1] 21.29755
```

The linear regression model that had the minimal error through the leave-one-out cross-validation was using both Humidity and WindSpeed to predict Temperature. Since the model with just Humidity did much better than just WindSpeed we can also infer that Humidity is likely more correlated with Temperature than WindSpeed is.

Problem 4 Bootstrap and Regression

Here we use a significance level of $\alpha = 0.5$ to construct a $1 - \alpha$ confidence interval for our simple linear regression β parameters via non-parametric bootstrap sampling. We will be using the model that includes both Humidity and WindSpeed to determine temperature.

```
alpha <- 0.05

# model
# temp = b_1 + b_2 * humid + b_3 * wind
# fit regression - calculate estimates
model <- lm(Temperature~Humidity+WindSpeed, data=weather)
beta_hat <- model$coefficients
epsilon_hat <- model$residuals
model$coefficients
```

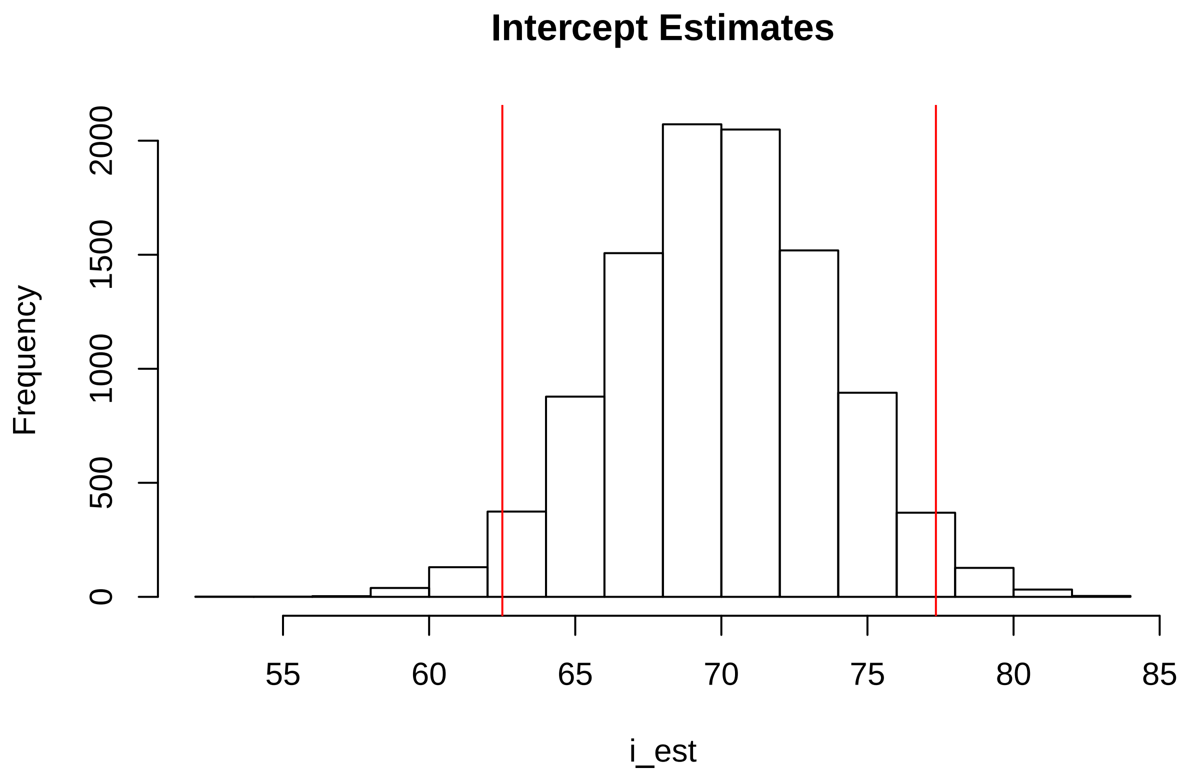
```
## (Intercept)    Humidity   WindSpeed
##   69.956939  -55.547540   -1.592721
```

```r
# bootstrap
K <- 10000
n <- nrow(weather)

beta_boot <- matrix(nrow=K,ncol=length(beta_hat))
for (i in 1:K) {
    humid_boot_k <- sample(weather$Humidity, n, replace=TRUE)
    wind_boot_k <- sample(weather$WindSpeed, n, replace=TRUE)
    epsilon_boot_k <- sample(epsilon_hat, n, replace=TRUE)

    y_k <- beta_hat[1] +
        beta_hat[2]*humid_boot_k +
        beta_hat[3]*wind_boot_k +
        epsilon_boot_k
    beta_boot[i,] <- lm(y_k~humid_boot_k+wind_boot_k)$coefficients
}
```

```r
# calculating confidence intervals
i_est <- beta_boot[,1]
interval <- quantile(i_est, probs=c(alpha/2, 1-alpha/2))
print(interval)
```

```
##     2.5%    97.5%
## 62.50525 77.34226
```

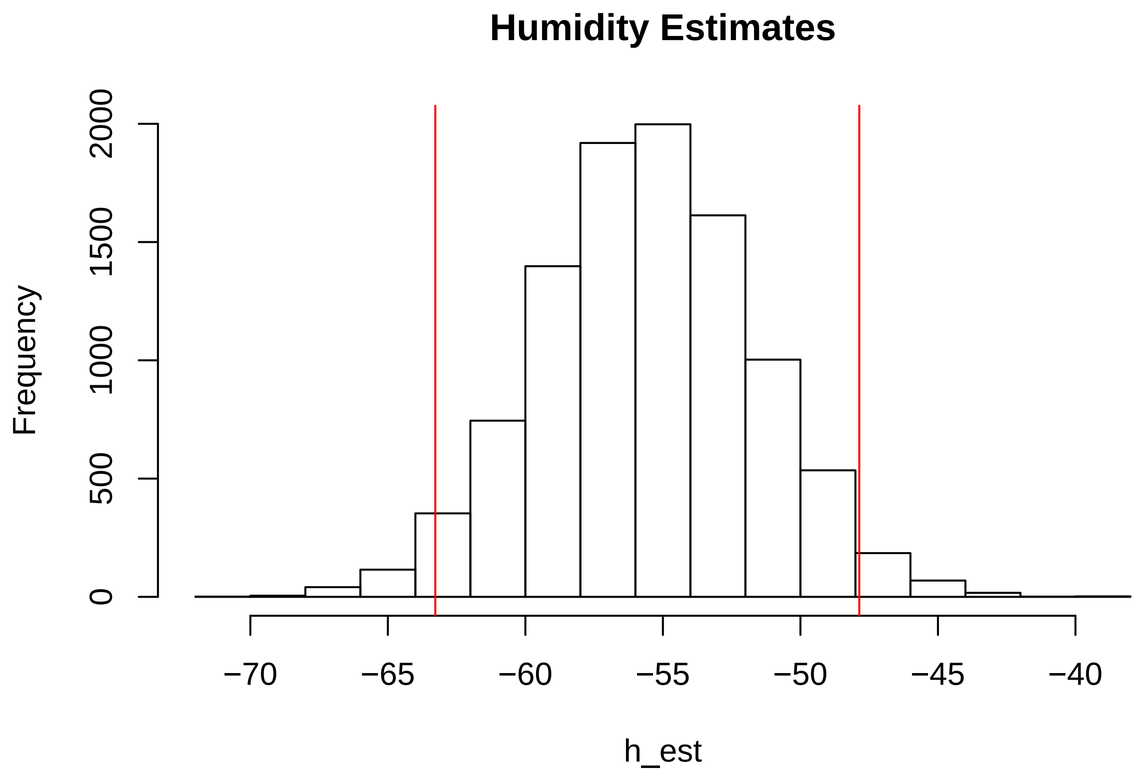
```r
hist(i_est, main="Intercept Estimates")
abline(v=interval, col='red')
```




```r
h_est <- beta_boot[,2]
interval <- quantile(h_est, probs=c(alpha/2, 1-alpha/2))
print(interval)
```

```
##      2.5%     97.5%
## -63.27531 -47.85855
```

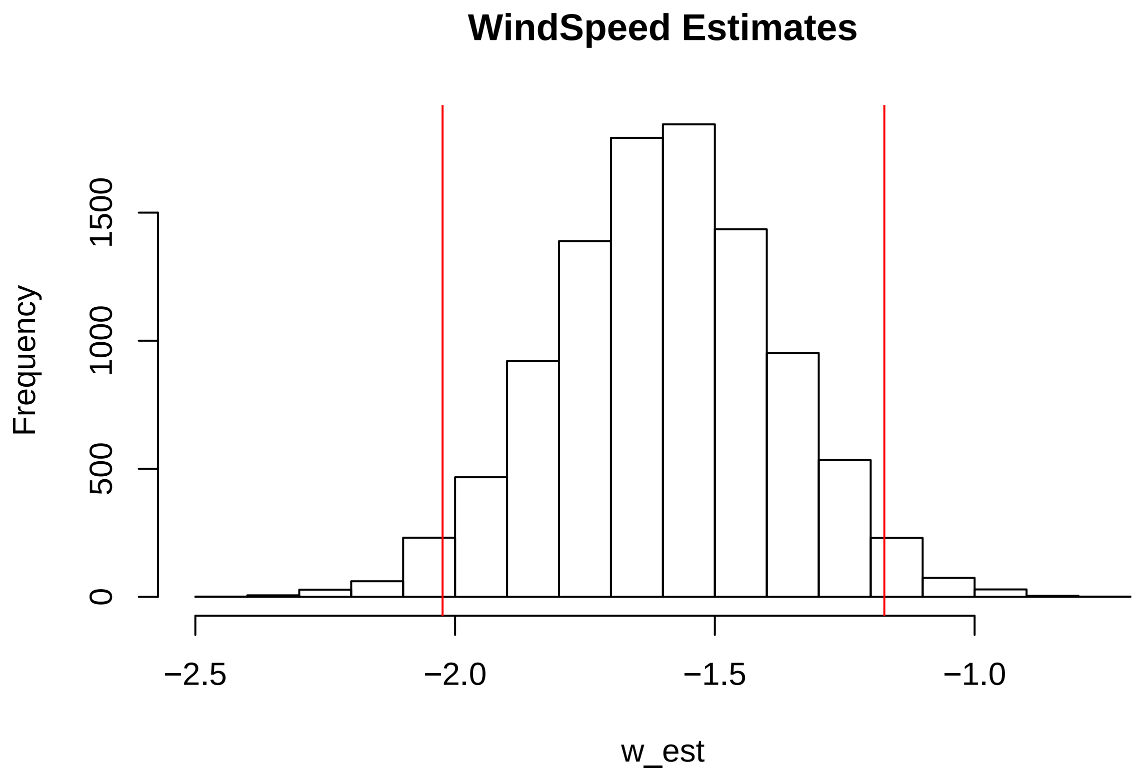
```r
hist(h_est, main="Humidity Estimates")
abline(v=interval, col='red')
```




```r
w_est <- beta_boot[,3]
interval <- quantile(w_est, probs=c(alpha/2, 1-alpha/2))
print(interval)
```

```
##      2.5%     97.5%
## -2.024092 -1.173746
```

```r
hist(w_est, main="WindSpeed Estimates")
abline(v=interval, col='red')
```



Via our bootstrap sampling we are stating that our 95% confidence interval for the intercept estimate in $\text{Temp} = \beta_1 + \beta_2\text{Humid} + \beta_3\text{WindSpeed}$ is from 62.51 to 77.34. The β_2 parameter interval is from -63.28 to -47.86 and the β_3 parameter interval is from -2.02 to -1.17.

Concept Map

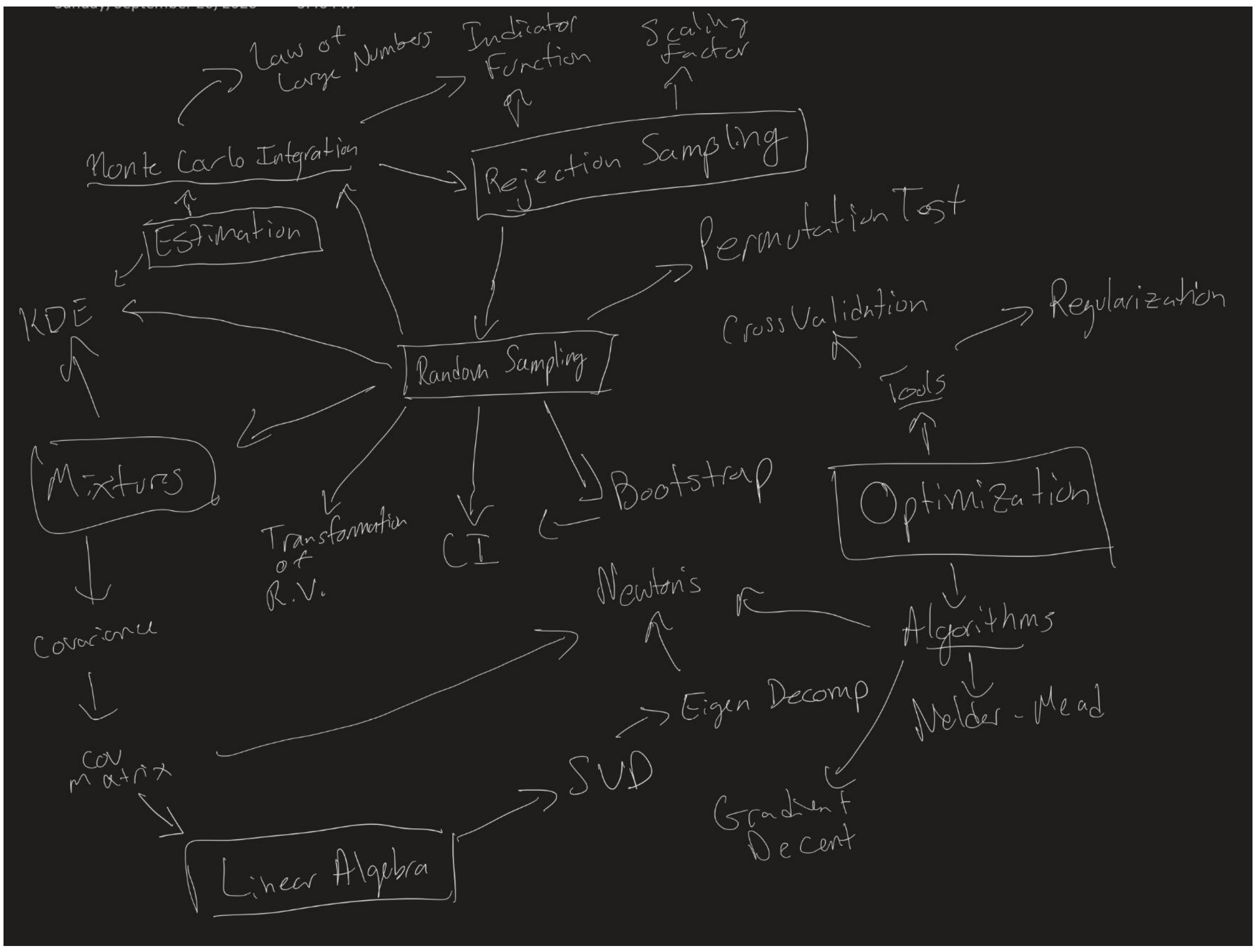


Figure 1: Concept Map