16096156

STAT0030: ICA3

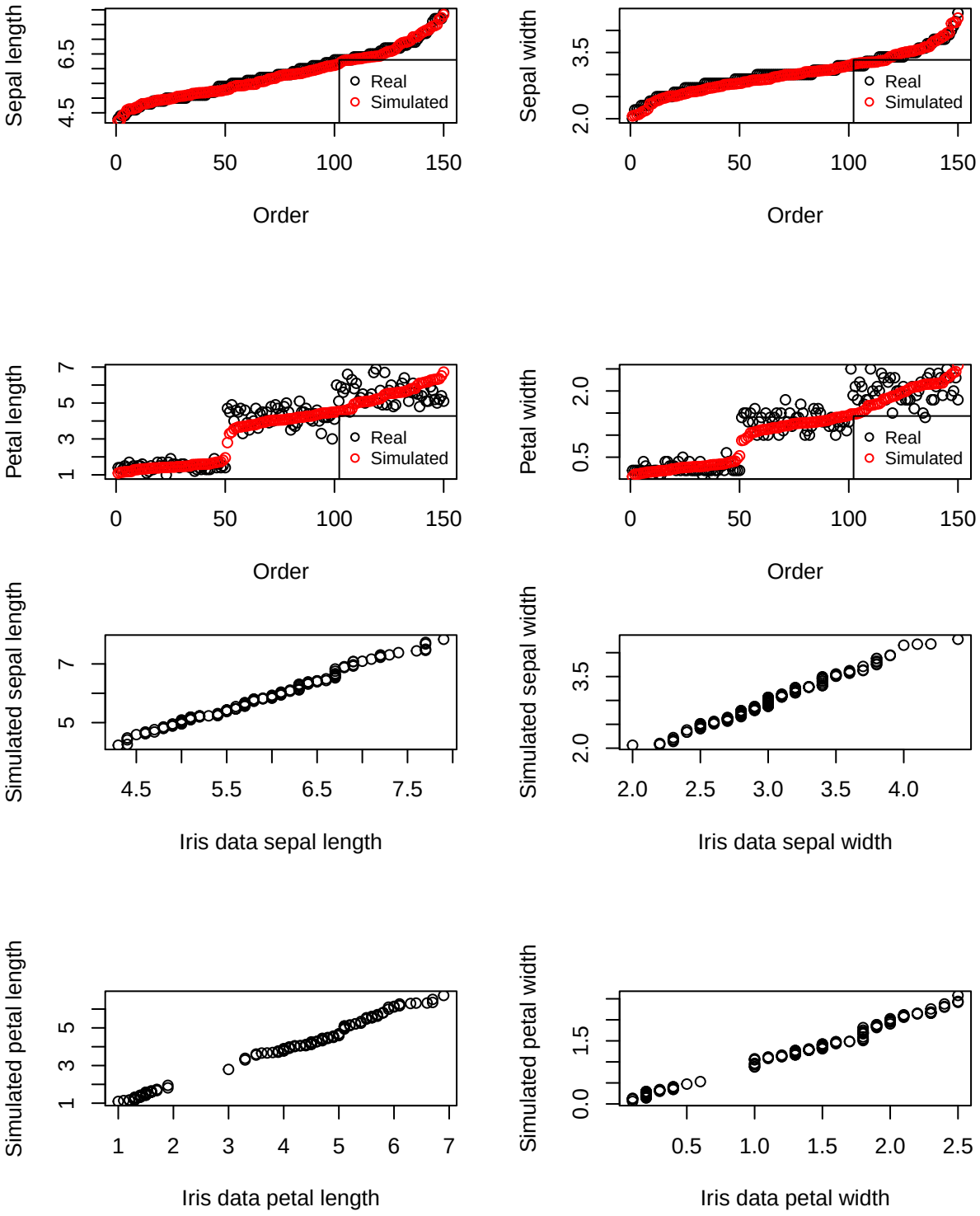
```r
# First, read the dataset
data(iris)
```

Q1.

```r
# Set the seed to ensure that the code is reproducible
set.seed(12345)

# Rename the data
data <- iris[, -5]

# Load the necessary library
library(EMMIXmfa)

# a)

# Run the MFA model with K = 3, M=2, nkmeans = 3, nrandom = 0
K = 3
M = 2
mfa_model <- mfa(data, g = K, q = M, nkmeans = 3, nrandom = 0, sigma_type = "unique",
    D_type = "common")

n.cols <- ncol(data)
n.rows <- nrow(data)

# We first declare X as an empty matrix, which will be filled with the values of
# the simulated data.
X <- matrix(0, nrow = n.rows, ncol = n.cols)

# Then we also sample Z from K = {1, 2, 3}, with probabilities equal to those
# generated by the MFA model.
p <- mfa_model$pivec
Z <- sample(x = seq(1, K), size = 150, replace = TRUE, prob = p)

# Write a function that uses the output of the MFA model to simulate the iris
# dataset Inputs: N, output of MFA model Output: N X J matrix of our samples
simulate_mfa <- function(N, mfa_model) {
    b <- mfa_model$B
    D <- mfa_model$D
    var <- diag(mfa_model$D)
    mu <- mfa_model$mu
    J <- ncol(D)
    for (i in seq(1, N)) {
        Y <- rnorm(M, mean = 0, sd = 1)
```

```
        b_j <- b[, , Z[i]]
        mu_j <- mu[, Z[i]]
        error <- rnorm(J, mean = 0, sd = sqrt(var))
        X[i, ] <- t(b_j %*% Y) + mu_j + error
    }
    return(X)
}

N <- n.rows
data_sim <- simulate_mfa(N, mfa_model)
data_sim <- as.data.frame(data_sim)

# b)

# In what follows, we will assess how well the simulated data matches the initial
# data, by looking at both scatter plots of the two sets of variables (sorted in
# ascending order by index) and at Q-Q plots.

par(mfrow = c(2, 2))

plot(sort(data$Sepal.Length), ylab = "Sepal length", xlab = "Order")
points(sort(data_sim$V1), col = "red")
legend("bottomright", legend = c("Real", "Simulated"), col = c("black", "red"), pch = c(1,
    1), cex = 0.8)
plot(sort(data$Sepal.Width), ylab = "Sepal width", xlab = "Order")
points(sort(data_sim$V2), col = "red")
legend("bottomright", legend = c("Real", "Simulated"), col = c("black", "red"), pch = c(1,
    1), cex = 0.8)
plot(data$Petal.Length, ylab = "Petal length", xlab = "Order")
points(sort(data_sim$V3), col = "red")
legend("bottomright", legend = c("Real", "Simulated"), col = c("black", "red"), pch = c(1,
    1), cex = 0.8)
plot(data$Petal.Width, ylab = "Petal width", xlab = "Order")
points(sort(data_sim$V4), col = "red")
legend("bottomright", legend = c("Real", "Simulated"), col = c("black", "red"), pch = c(1,
    1), cex = 0.8)

qqplot(data$Sepal.Length, data_sim$V1, xlab = "Iris data sepal length", ylab = "Simulated sepal length")
qqplot(data$Sepal.Width, data_sim$V2, xlab = "Iris data sepal width", ylab = "Simulated sepal width")
qqplot(data$Petal.Length, data_sim$V3, xlab = "Iris data petal length", ylab = "Simulated petal length")
qqplot(data$Petal.Width, data_sim$V4, xlab = "Iris data petal width", ylab = "Simulated petal width")
```

```
##   |                                                                      |
```



Q1.b. As can be seen from the scatterplots above, the simulated data appears to mimic the real data well, particularly in the case of Sepal length and Sepal width. In the other two cases, the simulated data appears to be more concentrated (less spread out) than the initial data. The Q-Q plots, which aim to establish whether the two sets of variables follow the same distribution, tell a consistent story to that of the scatter plots. Once again, in the case of Sepal length and width, the Q-Q plots suggests that the two datasets are very close to each other, with very few outliers at the extreme ends of the distributions. Similarly, the distributions of the two datasets recording Petal length and width appear to be identical, with only a few

outliers in the middle quantiles.

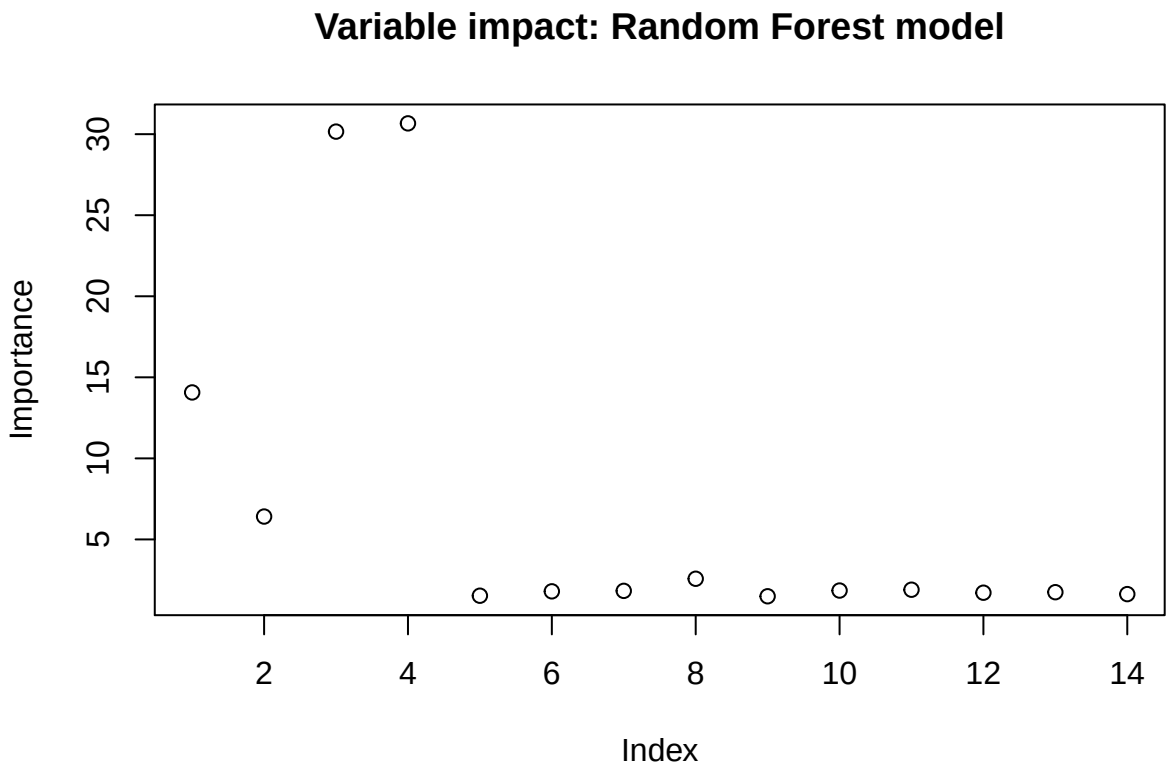
Q1.c.There are a number of techniques for selecting the right model given a particular dataset, in terms of setting the appropriate M and K parameters. The majority of these methods asses a measure of distance or spread between the points that make up a cluster, in order to establish whether the current clustering setting is appropriate or not. This deals with both the choice of M and K. The Silhouette Method is an example of such a model selection tool, which calculates the average distance of one point to all other points within its cluster and compares that to the minimum average distance of the same point to points within other clusters. The silhouette coefficient is the difference between the latter and the former, normalized by the maximum between the two. Naturally, the best clustering would be one that maximizes the silhouette coefficient. Another example of such a method is the Bayesian Information Criteria (BIC), which attempt to maximize a model's likelihood function, whilst penalizing for the model's complexity. The model with the lowest BIC is desirable. One way to optimize M and K for any of these methods is to perform cross-validation across different values of the two parameters and to select the values that render the model with the most desirable outcome (lowest BIC, for example).

Q2.

```
# Set the seed to ensure that the code is reproducible
set.seed(12345)

# Load the necessary library
library(randomForest)

# Re-declare the data and the hyperparameters
data = iris[, -5]
K = 3
M = 2

# a) We first use the function predict on the MFA model with K=3, M=2, nkmeans =
# 3, nrandom = 0, in order to predict the cluster assignments
preds_mfa <- predict(mfa(data, g = K, q = M, nkmeans = 3, nrandom = 0, sigma_type = "unique",
    D_type = "common"), Y = data)

# We then run the kmeans algorithm, with 3 centroids, and we save its cluster
# assignments in a separate variable
kmeanscl <- kmeans(data, centers = 3)
preds_kmeans <- kmeanscl$cluster

# We use the Adjusted Rand Index (ARI) function to compare the accuracy of the
# two prediction methods above
index_comp <- ari(preds_mfa, preds_kmeans)
index_mfa <- ari(preds_mfa, iris[, 5])
index_kmeans <- ari(preds_kmeans, iris[, 5])


# b) The comments on this question are at the end of the code chunk.

# c)

# We first generate a matrix of ten columns of independent standard Gaussians
gauss <- matrix(0, nrow = 150, ncol = 10)
for (i in seq(1:10)) {
    gauss[, i] = t(rnorm(150, mean = 0, sd = 1))
}
```

```r
# Then we add this matrix to the iris dataset, forming iris_noisy. We construct a
# random forest on iris_noisy to predict the labels
iris_noisy = cbind(iris[, -5], gauss)
iris_rf <- randomForest(x = iris_noisy, y = iris$Species, keep.forest = TRUE)

rf_oob <- iris_rf$err.rate[iris_rf$ntree]  #i-th element of the error rate vector is the
# OOB error rate for all trees up to the i-th, therefore we want to choose the
# OOB error rate corresponding to the last tree.
rf_labels <- iris_rf$predicted
rf_assess <- ari(rf_labels, iris$Species)  #we also run ari on the predicted labels of the RF

# We also run MFA on this modified dataset in order to get its prediction and
# assess its accuracy
model <- mfa(iris_noisy, g = K, q = M, nkmeans = 3, nrandom = 0, sigma_type = "unique",
    D_type = "common")
mfa_labels <- predict(model, Y = iris_noisy)
mfa_assess <- ari(mfa_labels, iris$Species)

plot(importance(iris_rf), main = "Variable impact: Random Forest model", ylab = "Importance",
    xlab = "Index")
```

```
##   |                                                                      |
##   |                                                                      |
```



Q2.b. The two models do not seem to have a perfect overlap in classifying the data into clusters. In fact, the MFA model appears to perform better when compared to the true labels of the dataset, with an overlap of 94%, whilst the kmeans model only achieves an overlap of 73% relative to the true labels. The kmeans algorithm uses Euclidean distance as a means of determining a point's proximity to a nearby center, which

can make this model sensitive to outliers ('Pattern Recognition and Machine Learning', Christopher Bishop). As can be seen from the plots of the iris data, all four dimensions contain outliers, which could be one of the reasons why the kmeans model performs poorly relative to the MFA model. Additionally, kmeans does not account for any uncertainty in the data, whereas the MFA model does, through incorporating probability distributions in the generation of data. One way to improve the performance of kmeans is to run many iterations of the algorithm, starting from different initializations each time, and keeping the one with the lowest squared error. This procedure can be more costly and time-consuming, however these challenges can be overcome by splitting existing data into a number of datasets, instead of using new datasets for each iteration. This method is likely to succeed in improving the performance of kmeans as it tests different starting points and chooses the one that generates the best result for that particular dataset (initialization being one of the main factors that impacts the performance of kmeans).

Q2.c. The Random Forest exhibits a very low out-of-bag error of 4%, consistent with its high ARI of roughly 89%. This suggests that the model fits the data well and is able to generate labels that are close to the true labels of the data. On the other hand, the MFA model does not compare well, with an ARI of just 53%. One explanation for the superior performance of the Random Forest model can be inferred from its importance plot shown above. It is clear that the variables with the most impact on the model's predictions are the first four variables, namely the ones that are also present in the original iris dataset. On the other hand, minimal importance is allocated to the randomly generated variables, which have no connection to the data labels and have been created artificially. Additionally, Random Forests represent an averaging of a number of decision trees, which helps reduce overfitting and ensures that the model is able to generalize well in testing. This is particularly relevant for this type of data, where 10 out of 14 dimensions are generated at random and represent just noise in the clustering process; in this case, the Random Forest ensures that the noise does not have an oversized impact on the classification task at hand.

Q3.

```
# a) Setting the seed to ensure that code is reproducible
set.seed(12345)

# Define a function g that calculates the density function for a Gaussian
# multivariate distribution
g <- function(x, mu, covar) {
    k <- length(x)
    c <- exp(-1/2 * t(x - mu) %*% solve(covar) %*% (x - mu))/sqrt((2 * pi)^k * det(covar))
    return(c)
}

# Define the desired function mfa_impute Inputs: data, output of MFA model
# Outputs: E_X (N x J x K), P_Z (N x K)
mfa_impute = function(dat, mfa_model) {

    # Declare the dimensions
    N <- nrow(dat)
    J <- ncol(dat)
    K <- mfa_model$g

    # Declare the matrices that will store the desired data
    E_X <- array(rep(1, N * J * K), dim = c(N, J, K))
    P_Z <- array(rep(1, N * K), dim = c(N, K))
    gauss <- matrix(1, nrow = N, ncol = K)  #matrix that would include all of the
    # N*K multivariate Gaussian densities (each a scalar)

    # Loop over each observation
    for (k in seq(1, K)) {
```

```
        for (i in seq(1, N)) {
            # Construct the data
            oi <- which(!is.na(dat[i, ]))
            xi_oi <- dat[i, oi]
            xi_oi <- as.numeric(xi_oi)

            # Retrieve the generic parameters from the MFA model
            Z_i <- k
            mu <- mfa_model$mu[, Z_i]
            covar <- mfa_model$B[, , Z_i] %*% t(mfa_model$B[, , Z_i]) + mfa_model$D

            # Retrieve the oi subsets from the generic parameters
            mu_oi <- mu[oi]
            cov_joi <- covar[, oi]
            cov_oioi <- covar[oi, oi]

            # Calculate the expectation at the current Z_i
            E_X[i, , k] <- mu + (cov_joi %*% solve(cov_oioi)) %*% (xi_oi - mu_oi)

            # Calculate pi * g for the current i and each k
            gauss[i, Z_i] <- mfa_model$pivec[Z_i] * g(xi_oi, mu_oi, cov_oioi)

        }
    }

    # Divide current pi * g by the sum of all (pi *g), summed over all K's i.e.
    # across the row
    P_Z <- gauss/rowSums(gauss)
    return(list(E_X = E_X, P_Z = P_Z))
}

# b)

# Partition iris into two datasets, iris1 and iris2
size_1 <- 100
rows <- nrow(iris)
sample_1 <- sample(rows, size = size_1, replace = FALSE)
iris1 <- iris[sample_1, ]
iris2 <- iris[-sample_1, ]

# Run an MFA model on iris1
mfa_model <- mfa(iris1[, -5], g = K, q = M, nkmeans = 3, nrandom = 0, sigma_type = "unique",
    D_type = "common")

# Modify iris2 to choose uniformly at random one column to be set to NA (for each
# of its rows)
iris3 <- iris2[, -5]
ncols <- ncol(iris3)
# Sample uniformly at random indices of the columns to be set to NA
index <- sample(ncols, size = 50, replace = TRUE)
for (i in seq(1, 50)) {
    iris3[i, index[i]] <- NA
}
```

```r
# Run mfa_impute on the newly created iris3 dataset
iris3_imputed <- mfa_impute(iris3, mfa_model)

# c)

# Compute Z_hat as the maximum on each row of Z values
P_Z <- iris3_imputed$P_Z
Z_hat <- max.col(P_Z)


# Predict Z_hat2 based on the MFA model and the iris2 data
Z_hat2 <- predict(mfa_model, Y = iris2[, -5])

# Use the function ari to compare how well the output of mfa_impute and the
# prediction based on the MFA model fit the actual labels of the data
comp_1 <- ari(Z_hat, iris2[, 5])
comp_2 <- ari(Z_hat2, iris2[, 5])

cat(sprintf("The overlap in classification between the output of mfa_impute and
            the true labels is %s",
    comp_1 * 100), "%", "\n")
cat(sprintf("The overlap in classification between the prediction generated by
            the MFA model and the true labels is %s",
    comp_2 * 100), "%", "\n")

# d)

N <- nrow(iris3)
J <- ncol(iris3)
K <- 3

# We first retrieve P_Z and E_X from the MFA model ran on iris3
P_Z <- iris3_imputed$P_Z
E_X <- iris3_imputed$E_X

# We calculate below E(Xij given xioi) according to the formula
E_XX <- array(rep(1, N * J * K), dim = c(N, J, K))
for (k in seq(1, K)) {
    E_XX[, , k] <- P_Z[, k] * E_X[, , k]
}
E_XX <- rowSums(E_XX, dims = 2)

# We then proceed to estimate the missing entries in iris3 as their corresponding
# values in E_XX. We also include below the true labels (true_labels) for the
# missing values according to the initial dataset.
EX_na <- array(rep(1, N))
true_labels <- array(rep(1, N))
for (i in seq(1, N)) {
    EX_na[i] <- E_XX[i, index[i]]
    true_labels[i] <- iris2[i, index[i]]
}
MSE_1 <- sum((EX_na - true_labels)^2)/N  #mean squared error of the difference
# between the estimated missing values and the true labels.
```

```r
# Lastly, we also estimate the missing values as the average of the observed
# values in that particular column (i.e. exclude all NA's)
EX_emp <- array(rep(1, N))
for (i in seq(1, N)) {
    for (j in seq(1, J)) {
        if (index[i] == j) {
            EX_emp[i] = mean(na.omit(iris3[, j]))
        }
    }
}
MSE_2 <- sum((EX_emp - true_labels)^2)/N  #mean squared error of the difference
# between the estimated missing values calculated as averages of the observed
# values and the true labels.

cat("The mean squared error between the estimate of the missing labels using
    conditional expectations and the true labels is: ",
    MSE_1, "\n")
cat("The mean squared error between the estimate of the missing labels using the
    mean of the observed data and the true labels is: ",
    MSE_2, "\n")
```

```
##   |                                                                      |
## The overlap in classification between the output of mfa_impute and
##             the true labels is 77.0161204906052 %
## The overlap in classification between the prediction generated by
##             the MFA model and the true labels is 76.8645441142425 %
## The mean squared error between the estimate of the missing labels using
##     conditional expectations and the true labels is:  0.08041128
## The mean squared error between the estimate of the missing labels using the
##     mean of the observed data and the true labels is:  0.7209251
```

Q3.c. The mfa_impute function appears to do a slightly better job at estimating the true labels of the data than the prediction of the MFA model. The difference is minimal, however. This means that the MFA method, once trained on complete data, does a good predictive job even in the case of missing test data, which can be really valuable in practice, where complete data is often not feasible, too costly, etc.

Q3.d. The mean squared error of the estimates of the missing data, as calculated using conditional expectations, is relatively small in comparison to that of the estimates calculated using the means of the observed data. This is not surprising, given that, while at its core the former method also uses the values of the observed data to impute the missing data, it does so by also taking into account how the observed data fits into the different clusters. Furthermore, it utilizes observed data related to the same observation (i.e. row data) as the missing value, making it more likely to provide an 'informed' estimate for the missing value.