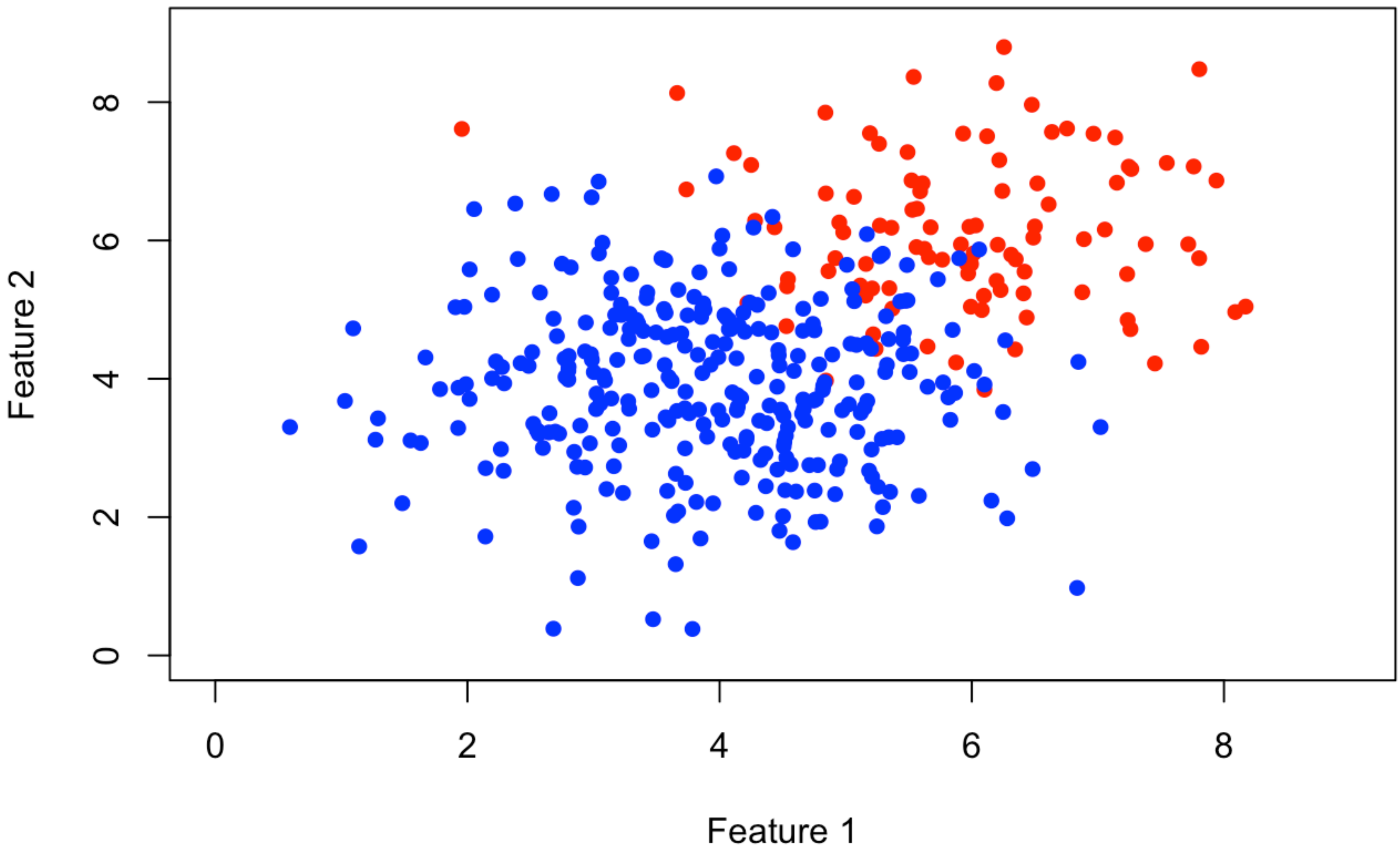
tutorial5

Likun Cui

04 September, 2018

Partition the data

```
# create positive class sample with 2 descriptive features set.seed(3)
f1 <- rnorm(100, mean=6, sd = 1.2)
set.seed(4)
f2 <- rnorm(100, mean=6, sd = 1.2)
P.data <- cbind(f1, f2)
# create positive class sample with 2 descriptive features set.seed(7)
f1 <- rnorm(300, mean=4, sd = 1.2)
set.seed(8)
f2 <- rnorm(300, mean=4, sd = 1.2)
N.data <- cbind(f1, f2)
# combine all samples
data.mat <- data.frame(rbind(P.data, N.data), Class=rep(c(1, 0), time=c(nrow(P.dat
a), nrow(N.data))))
plot(subset(data.mat, Class==1)[,-3], col="red", pch=16, ylim=c(0, 9), xlim=c(0, 9
), xlab="Feature 1", ylab="Feature 2")
points(subset(data.mat, Class==0)[,-3], col="blue", pch=16)
```




```
dim(N.data)
```

```
## [1] 300   2
```

```
sub<-sample(1:nrow(N.data),round(nrow(N.data)*0.8))
length(sub)
```

```
## [1] 240
```

```
data_train<-N.data[sub,]
data_test<-N.data[-sub,]
dim(data_train)
```

```
## [1] 240   2
```

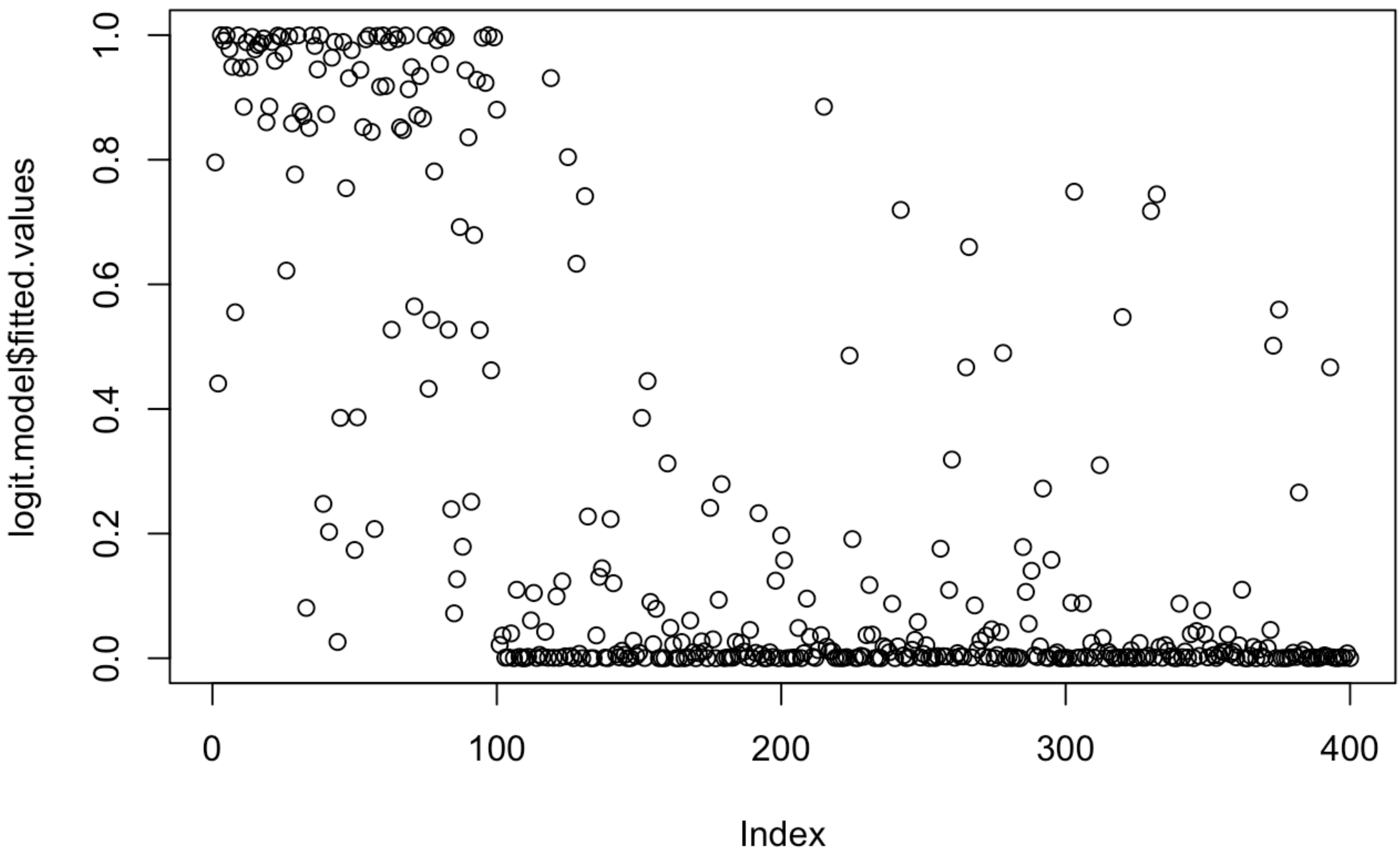
```
dim(data_test)
```

```
## [1] 60  2
```

```
#data_train
#print(data)
#data.mat
#data_train.mat
```

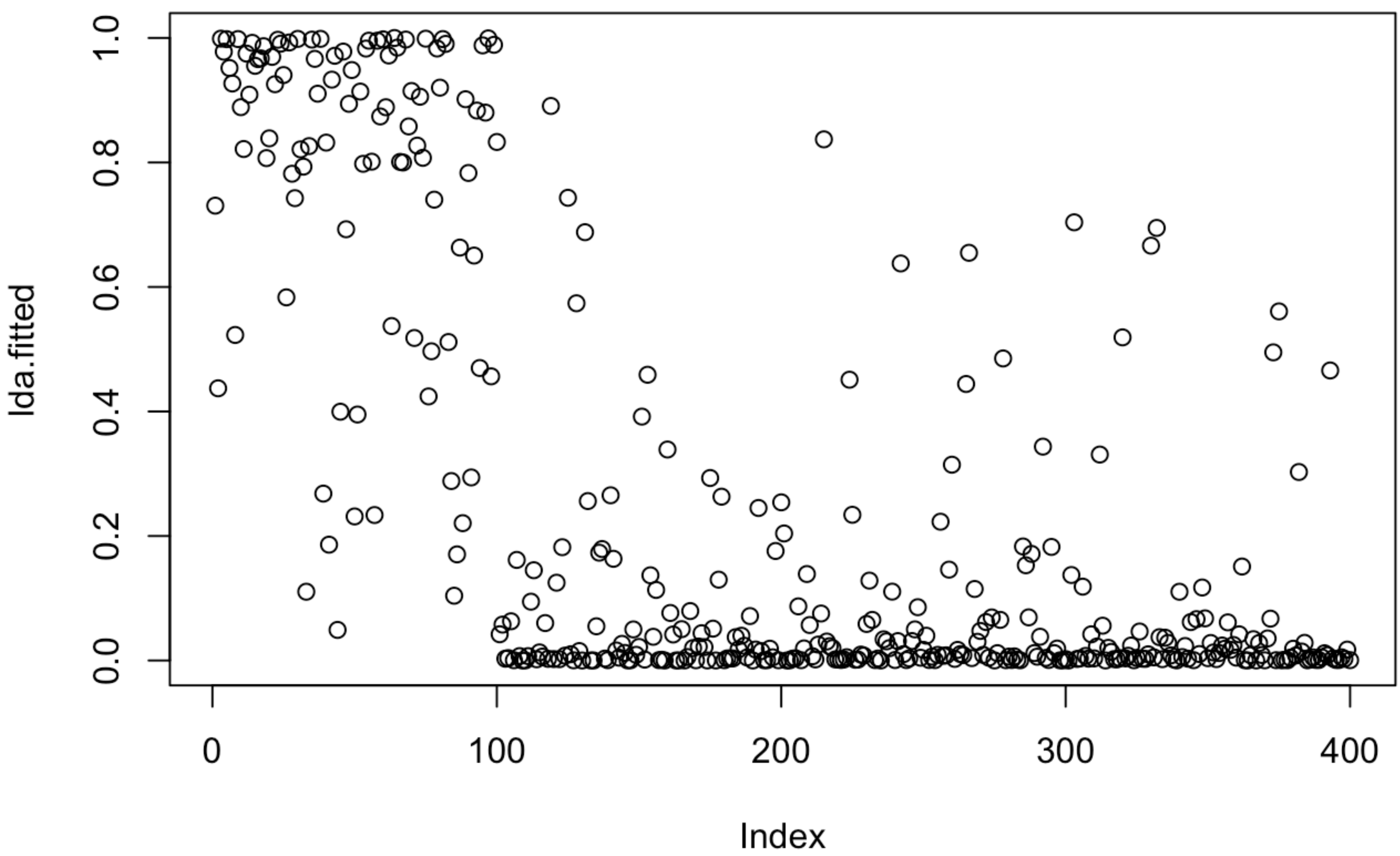
Train a Logistic Regression

```
# train a logistic regression model
logit.model <- glm(Class~., family=binomial(link='logit'), data=data.mat)
# plot fitted values from logistic regression model
plot(logit.model$fitted.values)
```



Train an LDA

```
library(MASS)
# train an LDA model
lda.model <- lda(Class~., data=data.mat)
lda.fitted <- predict(lda.model, data.mat)$posterior[,"1"]
# plot fitted values from LDA model
plot(lda.fitted)
```




```
# use fitted value to classify samples
lda.decision <- ifelse(lda.fitted > 0.5, 1, 0)
# calculate classification accuracy (in percentage %)
sum(lda.decision == data.mat$Class) / nrow(data.mat) * 100
```

```
## [1] 92.5
```

Train an KNN

```
library(class)
# a knn with k=1
knn.model1 <- knn(train=data.mat[,-3], test=data.mat[,-3], cl=data.mat[,3], k=1)
# a knn with k=5
knn.model2 <- knn(train=data.mat[,-3], test=data.mat[,-3], cl=data.mat[,3], k=5)
# a knn with k=50
knn.model3 <- knn(train=data.mat[,-3], test=data.mat[,-3], cl=data.mat[,3], k=50)

# calculate classification accuracy
sum(knn.model1 == data.mat$Class) / nrow(data.mat) * 100
```

```
## [1] 100
```

```
sum(knn.model2 == data.mat$Class) / nrow(data.mat) * 100
```

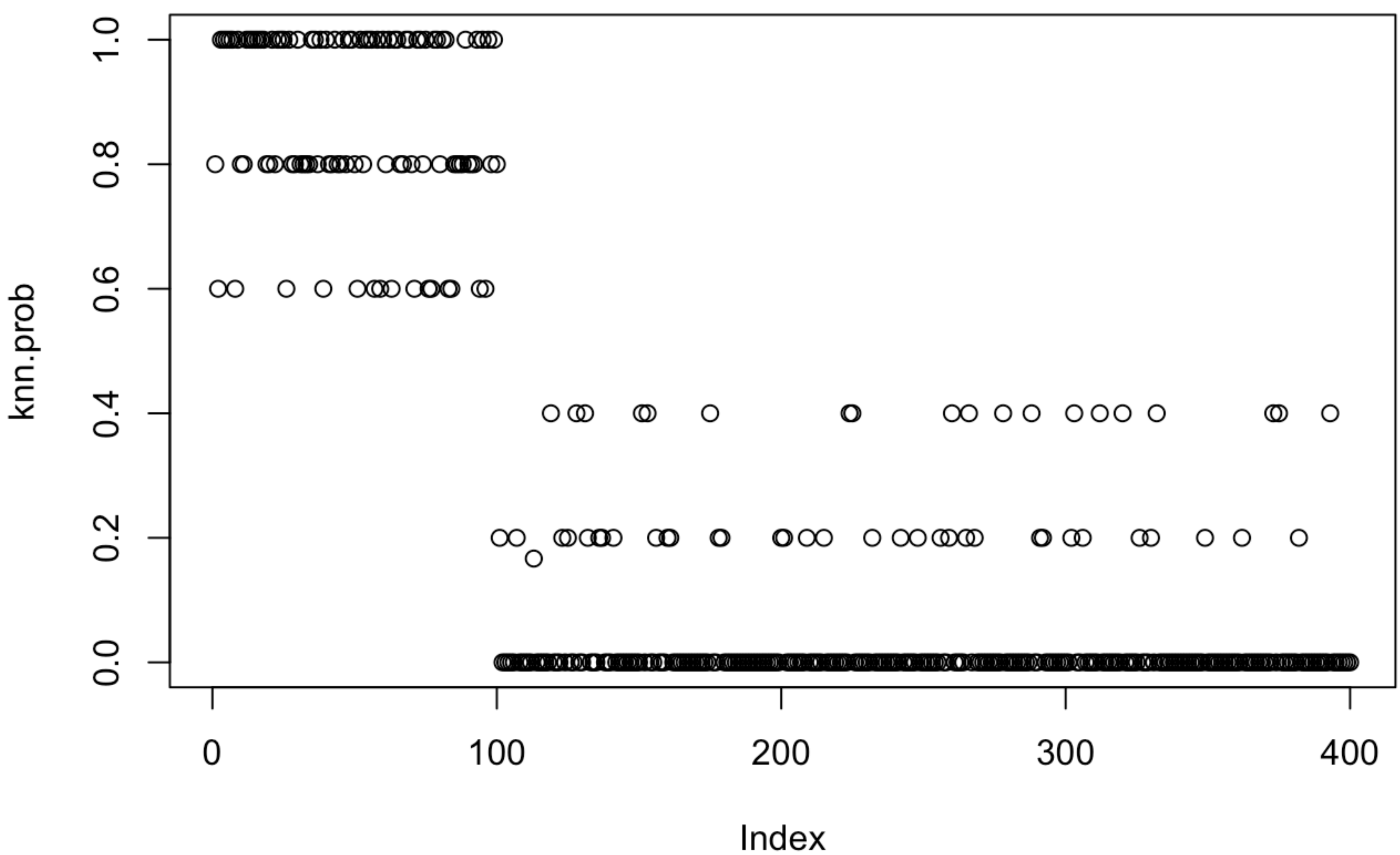
```
## [1] 93.5
```

```
sum(knn.model3 == data.mat$Class) / nrow(data.mat) * 100
```

```
## [1] 92.5
```

```
# apply knn and enable calculation of prediction probability
knn.model <- knn(train=data.mat[,-3], test=data.mat[,-3], cl=data.mat[,3], k=5, pr
ob=TRUE)
# extract prediction probability from the prediction model
knn.prob <- attr(knn.model, "prob")
isNegativeSample <- data.mat[,3] != 1
knn.prob[isNegativeSample] <-  1 - knn.prob[isNegativeSample]

plot(knn.prob)
```

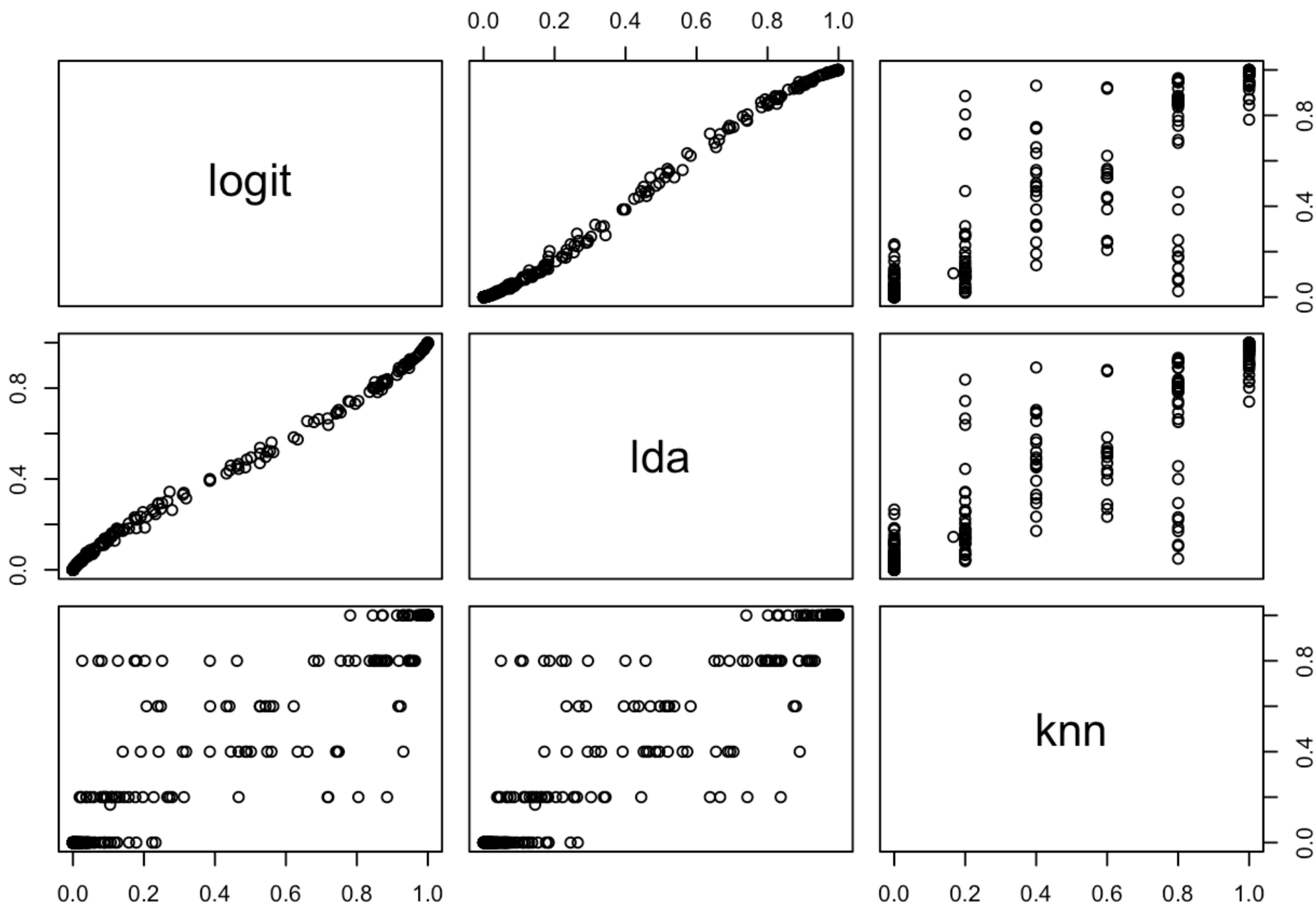


Compare their performance

```
# combine classification results into a data frame
classifications <- data.frame(logit=logit.model$fitted.values, lda=lda.fitted, knn
=knn.prob)
# calculate correlation
cor(classifications)
```

```
##           logit       lda       knn
## logit 1.0000000 0.9986088 0.9284626
## lda   0.9986088 1.0000000 0.9339678
## knn   0.9284626 0.9339678 1.0000000
```

```
# create pairwise scatter plot
pairs(classifications)
```



identify optimal k value by minimising classification error on test set.

```
library(caret)
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
set.seed(1)
inTrain <- createDataPartition(data.mat$Class, p = .8)[[1]]
dataTrain <- data.mat[ inTrain, ]
dataTest  <- data.mat[-inTrain, ]
set.seed(1)
KNN1 <- train(Class~f1,
                     data = dataTrain,
                     method = "knn",
                     trControl = trainControl(method = "repeatedcv",
                                              repeats = 5))
```

```
## Warning in train.default(x, y, weights = w, ...): You are trying to do
## regression and your outcome only has two possible values Are you trying to
## do classification? If so, use a 2 level factor as your outcome column.
```

```
## Print diagnostic and summary information and statistics fo the model
KNN1
```

```
## k-Nearest Neighbors
##
## 320 samples
##   1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 288, 288, 288, 288, 288, 288, ...
## Resampling results across tuning parameters:
##
##   k  RMSE       Rsquared   MAE
##   5  0.3351509  0.4249579  0.1982500
##   7  0.3242995  0.4516295  0.1987500
##   9  0.3272536  0.4417977  0.2040833
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```
print("Therefore, k=9 is the best choice.")
```

```
## [1] "Therefore, k=9 is the best choice."
```

Now we used test set to select optimal k, is it still valid to use this test set to evaluate the performance of our optimised kNN classifier?

Why or why not?

```
print("No. Since the chosen k would be perfectly predicting the test set, a new da
ta set should be developed to evaluate the performance.")
```

```
## [1] "No. Since the chosen k would be perfectly predicting the test set, a new d
ata set should be developed to evaluate the performance."
```