# Applied Data Mining Homework 2

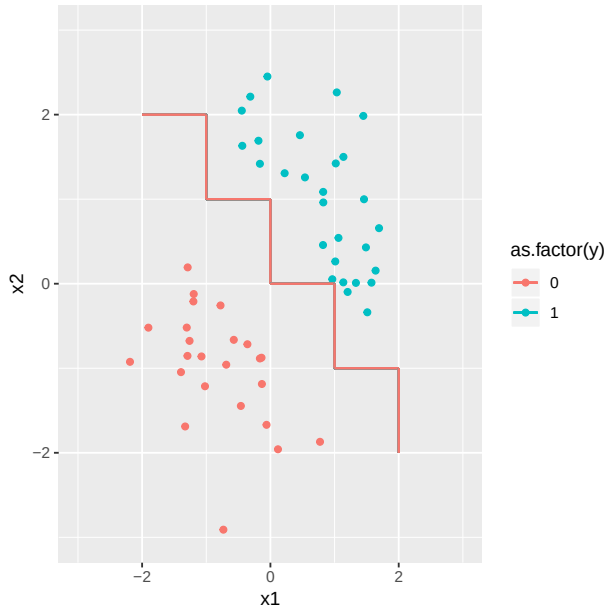
*Xun Zhao, xz2827*

## Problem 1: Trees

### 1.

Yes. But it needs the tree to be very high than some simple tree classifiers.

As the tree classifier boundary is made of some line segments that are parallel with x-axis or y-axis, we can draw a zigzag curve along the linear boundary. Thus, the sloping linear boundary becomes many segments, and every segment is a dicision in tree classifier.




### 2.

No, the Bayes-Optimal is defined as:

$$f(\vec{x}) = \underset{y}{arg\,max} P(\mathbf{Y} = y|\mathbf{X} = \vec{x})$$

And the risk is defined as:

$$R(f) = \sum_{+1,-1} \int L(y,\, f(\vec{x}))P(\vec{x},\, y)d\vec{x}$$

Thus, the risk $R(f)$ depends on the possible data distribution instead of only existing data points. Moreover, the Bayes-optimal classifier is defined by the possible distribution, so it can always minimize the risk under certain data model. However, the tree classifier cannot fit the linear boundary, which causes the misclassification compared with Bayes-optimal. As the Bayes-optimal has the lowest risk, the risk of tree classifier will always differ from Bayes-optimal, unless the Bayes-optimal's boundary is axis-parallel.

### 3.

$$g(\vec{x}) = \begin{cases} f_1(\vec{x}) > 0 & \begin{cases} f_2(\vec{x}) > 0 & class\, A \\ f_2(\vec{x}) < 0 & class\, B \end{cases} \\ f_1(\vec{x}) < 0 & \begin{cases} f_3(\vec{x}) > 0 & class\, B \\ f_3(\vec{x}) < 0 & class\, C \end{cases} \end{cases}$$

## Problem 2: 10-fold Cross Validation

### 1.

10 folds means that we devide the data set $\chi$ in to 10 parts with the same size $\frac{||\chi||}{10}$.

$$\chi = \bigcup_{i=1}^{10} \chi_i$$

$$\text{if } i \neq j,\ \chi_i \cap \chi_j = \varnothing$$

$$||\chi_i|| = \frac{||\chi||}{10}$$

### 2.

Because k-NN algorithm actually is not a learning algorithm with model parameter, the training data can be used only for classification. Thus, we do not have to choose the best model from a set.

Every time we use a new fold, namely, different cross validation set $\chi_{cv}$, we apply the classification function $f$ on $\chi_{cv}$ based on training set $\chi_{tr} = \chi - \chi_{cv}$ and culculated the err.

The training set is used as model (or classification function).

$$\hat{y} = f(\vec{x};\, k,\, \chi_{training}), \quad k,\, \chi_{training} \text{ are fixed}$$

The cross validation set is used as a criterion to quantify how well the model can generalize.

$$R(f) = \sum_{\vec{x} \text{ in } \chi_{cv}} l[y,\ f(\vec{x};\ k,\ \chi_{training})]$$

### 3.

As the risk function $R(k)$ given below, we compare the risk (for all folds and for all cross validation data points).

$$R(k) = \frac{1}{10} \sum_{i=1}^{10} \frac{1}{||\chi_i||} \sum_{j=1}^{||\chi_i||} l[y_{ij},\ f(\vec{x}_{ij};\ k,\ \chi - \chi_i)]$$

### 4.

We can simply choose $k$ that minimize the risk $R(k)$.

$$k = \underset{k \in \{1,3,5,7,9\}}{arg\,min}\ R(k)$$

### 5.

The most obvious disadvantage of k-NN algorithm is that it has to store all the training data, and iterate all the training data when classifying a new data point. Thus, it is time-consuming and occupy a lot of storage.

## Problem 3: Cross validating a nearest neighbor classifier

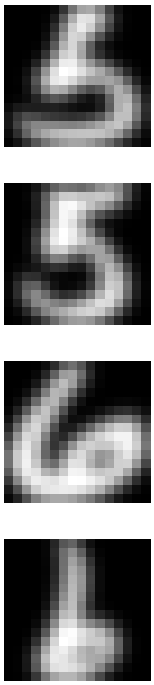
### 1. Read data from files

```
pixels = as.matrix(read.table('../data/uspsdata.txt'))
labels = as.matrix(read.table('../data/uspscl.txt'))
nums = nrow(pixels)
```

### 2. Plot first 4 digits

```
image.print <- function(x)
{
x.matrix <- matrix(x, 16, 16, byrow = FALSE)
```

```
x.matrix.rotated <- t(apply(x.matrix, 1, rev))
image(x.matrix.rotated, axes = FALSE, col = grey(seq(0, 1, length.out = 256)))
}
for(i in 1:4){
    image.print(pixels[i,])
}
```



## 3. Devide data into 3 parts

```
p.tr = pixels[1:round(nums * 0.6),]
l.tr = labels[1:round(nums * 0.6),]
p.cv = pixels[(round(nums * 0.6) + 1):round(nums * 0.8),]
l.cv = labels[(round(nums * 0.6) + 1):round(nums * 0.8),]
p.ts = pixels[(round(nums * 0.8) + 1):nums,]
l.ts = labels[(round(nums * 0.8) + 1):nums,]
```

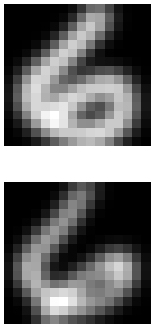
## 4. Train with $k = 1$

```
library(class)
estim = knn(p.tr, p.ts, cl = l.tr, k = 1)
err.rate = sum(estim != l.ts) / length(estim)
```

The test error is:

```
## [1] "err.rate = 0.05"
```

The misclassified digits are:



## 5. Optimaize $k$

```
k.seq = seq(1, 13, 2)
err.rates = sapply(
    k.seq,
    function(k){
        estim = knn(p.tr, p.ts, cl = l.tr, k = k)
        sum(estim != l.ts) / length(estim)
})
print(rbind(k.seq, err.rates))
```

```
##           [,1]  [,2] [,3]  [,4]  [,5]   [,6]   [,7]
## k.seq     1.00 3.000    5 7.000 9.000 11.000 13.000
## err.rates 0.05 0.025    0 0.025 0.025  0.025  0.025
```

The best $k$, with least error rate on test set is:

```
## [1] "k.optim = 5"
```

```
estim = knn(rbind(p.tr, p.ts), p.cv, cl = c(l.tr, l.ts), k = k.optim)
err.rate = sum(estim != l.cv) / length(estim)
```

Trained with training set and test set, the model's error rate on validation set is:

```
## [1] "err.rate = 0"
```