k-NN Classification

Predicting using nearest neighbors

k Nearest Neighbours

- k is the number of nearest neighbours the classifier will use to make it's prediction

- Instance based/Memory based learning which means that it chooses to memorize the training instances

- Non parametric which means it make no explicit assumptions of the relationship between the dependent variable and the independent variable

k-NN Algorithm

The k-NN algorithm gets its name from the fact that it uses information about the test's k-nearest neighbours to classify it.

- k-NN algorithm treats the features as coordinates in a multidimensional feature space.
- After choosing k, for each unlabeled record in the test dataset, k-NN identifies k records in the training data that are nearest in similarity.
- The unlabeled test instance is assigned the class of the majority of the k-nearest neighbors.
- The most common measure of similarity used is Euclidean distance, but there are other measures that can be used including the Manhattan, Chebyshev and Hamming distance.

k Nearest Neighbour

Euclidean distance which is given by

$$d(x, x') = \sqrt{(x_1 - x'_1)^2 + (x_2 - x'_2)^2 + \ldots + (x_n - x'_n)^2}$$

which is the straight line distance between any two points in space x and x' having the coordinates (x_1, x_2, …,x_n) and (x'_1, x'_2, …,x'_n) in a n-dimensional hyperplane.

k Nearest Neighbour

Features (j) →

← Training Data (i)

	x_{ij}	…	x_{ip}	y
x_{ij}	1			1
	2			2
	3			3
	…			…
x_{nj}	n			n

distance	result
$d(z,x_{1j})$	r1
$d(z,x_{2j})$	r2
$d(z,x_{3j})$	r3
…	…
$d(z,x_{nj})$	rn

k Nearest Neighbour

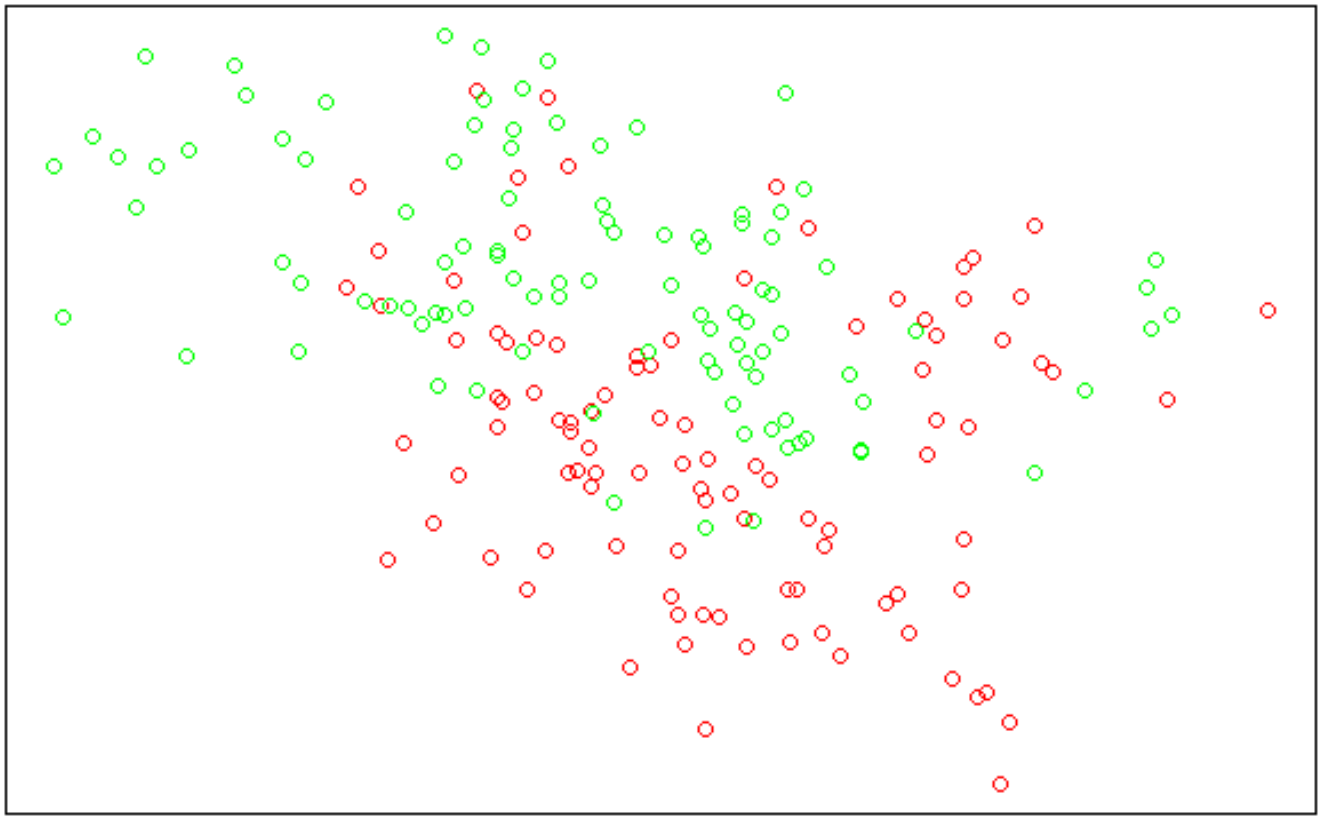
From the final output in the result vector, we choose the k values based on it's distance in the distance vector.

The final output is calculated as an arithmetic mean across the result values of the k nearest neighbours.

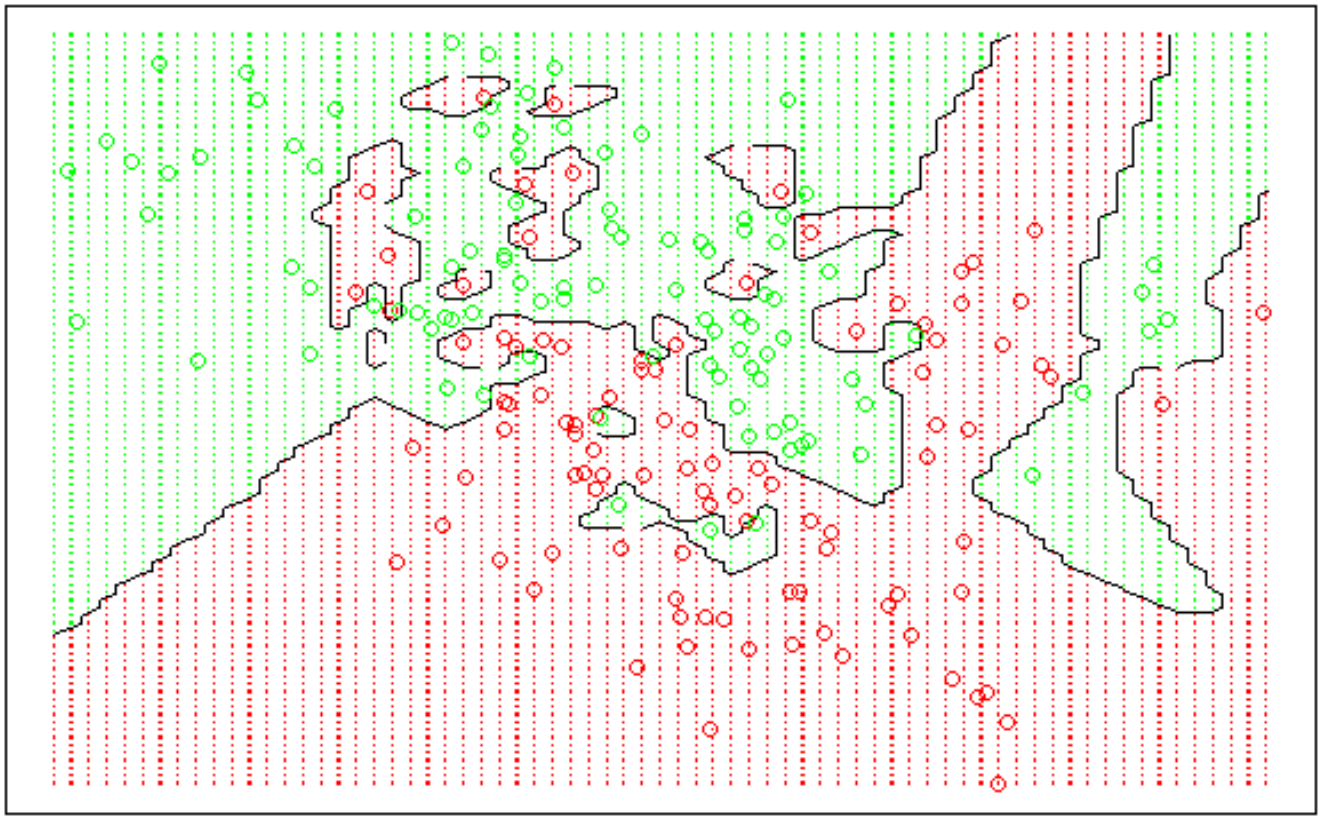
$$\bar{r} = \frac{1}{k} \sum_{i=1}^{k} r_i$$

Choosing a appropriate k determines how well the model will generalize to the test data. The balance between overfitting and underfitting the training data has to be kept in mind.

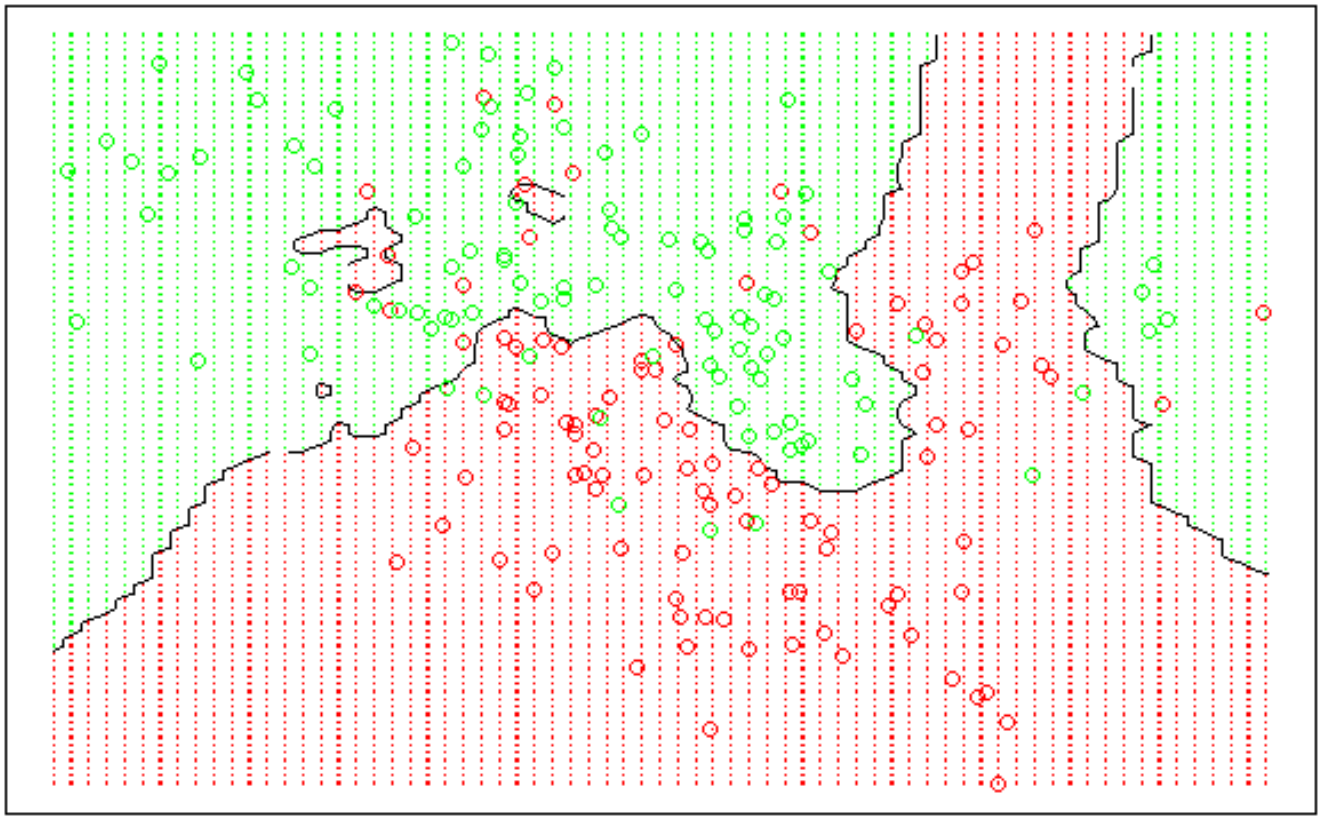
Distribution of the 2 classes



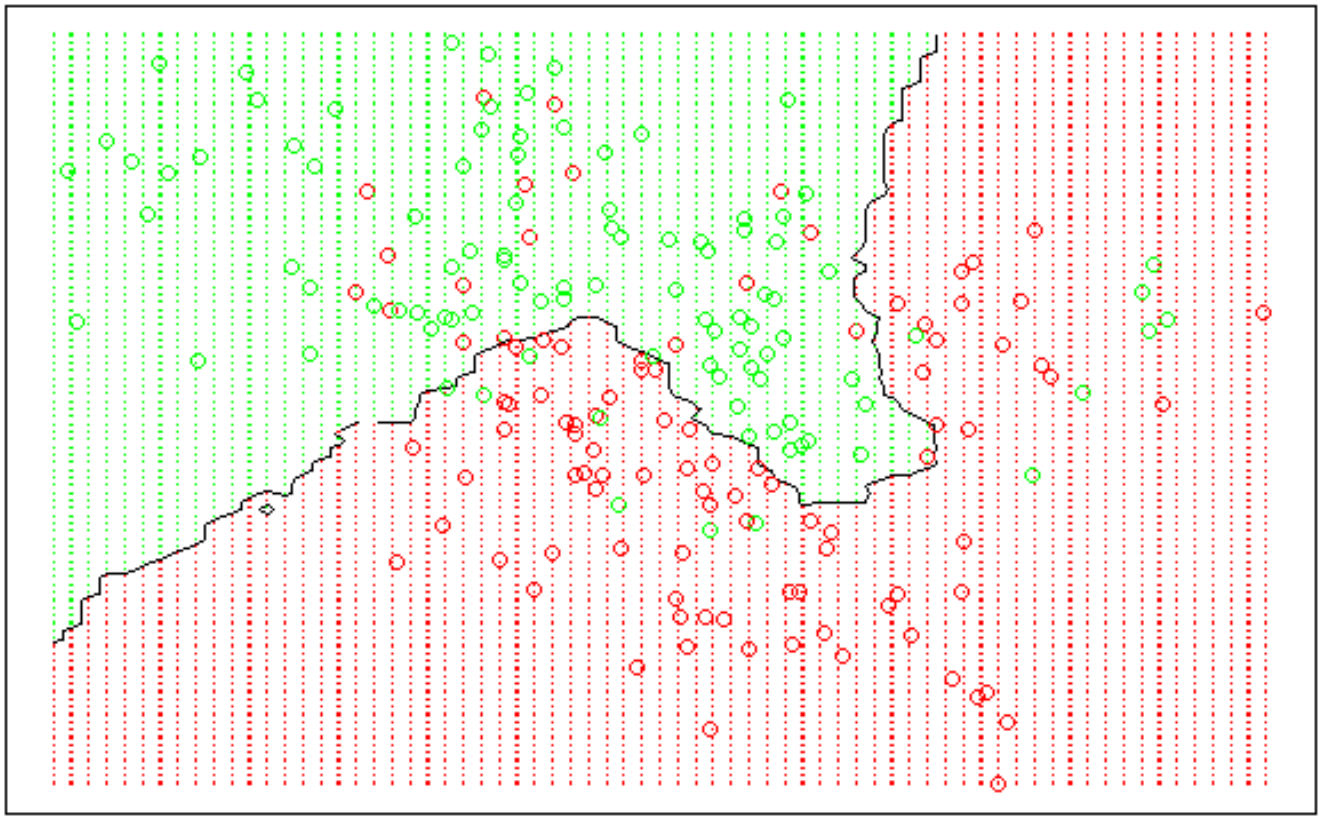
1-Nearest Neighbour



5-Nearest Neighbours



15-Nearest Neighbours



Example: Wisconsin Breast Cancer

We're going to reuse the Wisconsin Breast Cancer dataset in this exercise

```
> library(data.table)
> dt = fread("..\wisc_bc_data.csv")
> str(dt)

Classes 'data.table' and 'data.frame': 569 obs. of 32 variables:
$ id : int 842302 842517 84300903 84348301 84358402 ...
$ diagnosis : chr "M" "M" "M" "M" ...
$ radius_mean : num 18 20.6 19.7 11.4 20.3 ...
$ texture_mean : num 10.4 17.8 21.2 20.4 14.3 ...
$ perimeter_mean : num 122.8 132.9 130 77.6 135.1 ...
$ area_mean : num 1001 1326 1203 386 1297 ...
$ smoothness_mean : num 0.1184 0.0847 0.1096 0.1425 0.1003 ...
$ compactness_mean : num 0.2776 0.0786 0.1599 0.2839 0.1328 ...
$ concavity_mean : num 0.3001 0.0869 0.1974 0.2414 0.198 ...
$ concave points_mean : num 0.1471 0.0702 0.1279 0.1052 0.1043 ...
$ symmetry_mean : num 0.242 0.181 0.207 0.26 0.181 ...
$ fractal_dimension_mean : num 0.0787 0.0567 0.06 0.0974 0.0588 ...
$ radius_se : num 1.095 0.543 0.746 0.496 0.757 ... 0.1189 0.089 0.0876 0.173 0.0768 ...
...
 - attr(*, ".internal.selfref")=<externalptr>
```

Data Transformation

We don’t use the first column, which is a record identifier.

```
> dt = dt[, 1 := NULL]
```

We can rename the values of the column diagnosis

```
> dt[diagnosis == “M”, diagnosis := “Malignant”]
> dt[diagnosis == “B”, diagnosis := “Benign”]
> dt[, .N, by = .(diagnosis)]

          Diagnosis    N
1:        Malignant  212
2:           Benign  357
```

Data Transformation

Recall the distance calculation for k-NN is heavily dependent upon the measurement scale of the input features. Therefore we have to normalize our data.

```
> scale_0_1 = function(x) {
          return ((x – min(x)) / (max(x) – min(x)))
}
```

Let's test our normalize function before we use it on our data

```
> scale_0_1(c(1,2,3,4,5))
[1] 0.00 0.25 0.50 0.75 1.00
> scale_0_1(c(10,20,30,40,50))
[1] 0.00 0.25 0.50 0.75 1.00
```

Since it works as intended above, we can use it to normalize our data

```
> dt_n = dt[, lapply(.SD, scale_0_1), .SDcols =! "diagnosis"]
```

Data Preparation

Now we need to further prep our data to segment them into a training set and a test set.

```
> dt_train = dt_n[1:469]
> dt_train_labels = dt[1:469, .(diagnosis)]

> dt_test = dt_n[470:569]
> dt_test_labels = dt[470:569, .(diagnosis)]
```

Training the model

kNN classification syntax

using the `knn()` function in the `class` package

Building the classifier and making predictions:

```
p <- knn(train, test, class, k)
```

- `train` is a data frame containing numeric training data
- `test` is a data frame containing numeric test data
- `class` is a factor vector with the class for each row in the training data
- `k` is an integer indicating the number of nearest neighbors

The function returns a factor vector of predicted classes for each row in the test data frame.

Example:

```
wbcd_pred <- knn(train = wbcd_train, test = wbcd_test,
                 cl = wbcd_train_labels, k = 3)
```

Training the model

Therefore with the syntax of the knn function, we can build our model

```
> install.packages("class")
> library(class)
> modelk13 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k=13)
```

Next we want to evaluate our model performance

```
> CrossTable(x = dt_test_labels$diagnosis, y = modelk13, prop.chisq = FALSE)

                              | modelk13
dt_test_labels$diagnosis      |   Benign | Malignant | Row Total |
   ---------------------      |----------|-----------|-----------|
                       Benign |       76 |         1 |        77 |
                              |    0.987 |     0.013 |     0.770 |
                              |    0.974 |     0.045 |           |
                              |    0.760 |     0.010 |           |
  ---------------------       |----------|-----------|-----------|
                    Malignant |        2 |        21 |        23 |
                              |    0.087 |     0.913 |     0.230 |
                              |    0.026 |     0.955 |           |
                              |    0.020 |     0.210 |           |
  ---------------------       |----------|-----------|-----------|
                 Column Total |       78 |        22 |       100 |
                              |    0.780 |     0.220 |           |
  ---------------------       |----------|-----------|-----------|
```

Improving performance

We can try a different data transformation to improve the performance of the model, z-score standardization (X-mean(X))/sd(X)

```
> dt_z = dt[, lapply(.SD, scale), .SDcols =! "diagnosis"]
> dt_train_z = dt_z[1:469]
> dt_test_z = dt_z[470:569]
>
> modelk13_z = knn(train = dt_train_z, test = dt_test_z, cl = dt_train_labels$diagnosis,
                       k = 13)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk13_z, prop.chisq = FALSE)

                                | modelk13_z
dt_test_labels$diagnosis        |    Benign | Malignant | Row Total |
   ------------------------     |-----------|-----------|-----------|
                     Benign     |        76 |         1 |        77 |
                                |     0.987 |     0.013 |     0.770 |
                                |     0.987 |     0.043 |           |
                                |     0.760 |     0.010 |           |
   ------------------------     |-----------|-----------|-----------|
                  Malignant     |         1 |        22 |        23 |
                                |     0.043 |     0.957 |     0.230 |
                                |     0.013 |     0.957 |           |
                                |     0.010 |     0.220 |           |
   ------------------------     |-----------|-----------|-----------|
               Column Total     |        77 |        23 |       100 |
                                |     0.770 |     0.230 |           |
   ------------------------     |-----------|-----------|-----------|
```

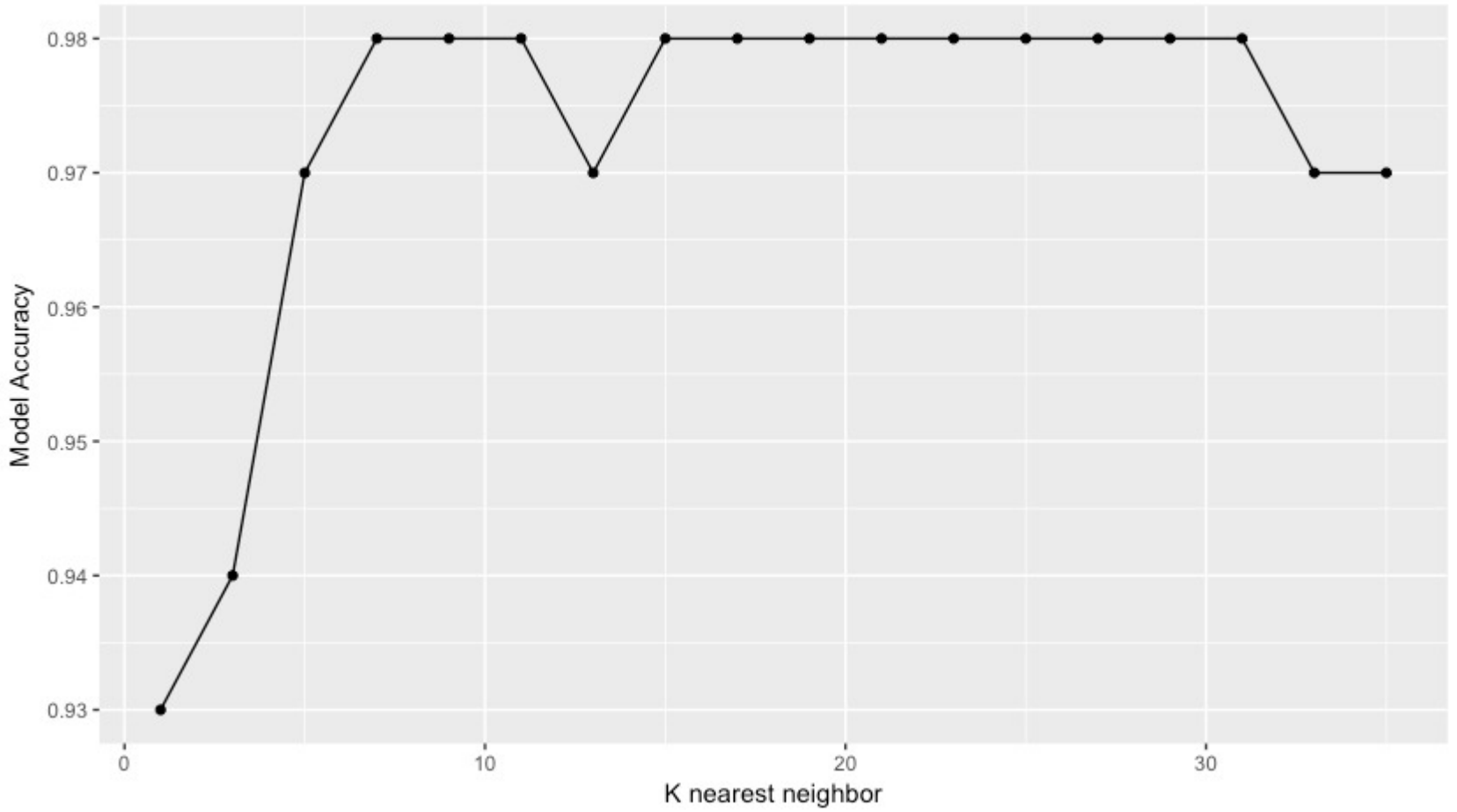
Improving performance

We can also try different values of k, to check which ones gives a better accuracy.

```
> modelk1 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k = 1)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk1, prop.chisq = FALSE)
...
>
> modelk5 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k = 5)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk5, prop.chisq = FALSE)
...
>
> modelk11 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k=11)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk11, prop.chisq = FALSE)
...
>
> modelk15 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k=15)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk15, prop.chisq = FALSE)
...
>
> modelk21 = knn(train = dt_train, test = dt_test, cl = dt_train_labels$diagnosis, k=21)
> CrossTable(x = dt_test_labels$diagnosis, y = modelk21, prop.chisq = FALSE)
...
```

Testing a set of ODD values for k

Improving performance



Curse of Dimensionality

When the number of p is large, there tends to be deterioration in the performance of kNN and other local approaches that perform prediction using only observations that are near the test observations for which a prediction must be made.

To understand this, let's assume we have a set of observations X that is uniformly distributed on [0,1]. We have only one feature, i.e.

p = 1 feature, X

We want to predict a test observation's response using only observations that are within 10% of the range of X closest to that test observation, for instance X = 0.4, the range will be [0.35, 0.45]

which is

$$= \frac{1}{10} \times 100\%$$

10% of the space or available observations.

Curse of Dimensionality

Now let's take a similar example, but with 2 dimensions

p = 2 dimensions, X_1 and X_2

- (X_1, X_2) are uniformly distributed on [0,1] x [0,1]
- 10% of the range of X_1, for example X_1 = 0.2, therefore [0.15, 0.25]
- 10% of the range of X_2, for example X_2 = 0.7, therefore [0.65, 0.75]

The cube which results from the ranges,

$$= \frac{1}{10} \times \frac{1}{10} \times 100\%$$

1% of the space or available observations.

Curse of Dimensionality

Let's try to compute the percentage of the available observations for p = 100

- $(X_1, X_2, \ldots, X_{100})$ are uniformly distributed on [0,1] x [0,1] x …. x [0,1]
- 10% of the range of each X, for that test observation

The hypercube which results from the ranges, make up

$$= \frac{1}{10} \times \frac{1}{10} \times \cdots \times \frac{1}{10} \times 100\% \; = 0.1^{100} \times 100\%$$

1^{-98}% of the space or available observations.

Therefore as p increase linearly, observations that are geometrically near decrease exponentially, which means that there are very few training observations NEAR any given test observation!!

Summary

Advantages

- Non parametric which means it makes no explicit assumptions of the relationship between the dependent variable and the independent variable.
- **No training phase since it keeps the training data.**
- Simple and effective.

Disadvantages

- It is a lazy algorithm and is computationally inefficient, since it stores all the training data.
- **Cannot work with high dimensional data, which means p has to be a small number.**
- **Slow classification phase.**