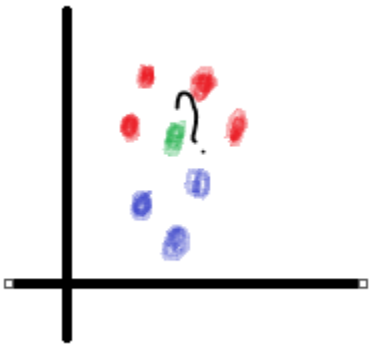
K - Nearest -Neighbor Classification

K- nearest neighbor gets its name since the method uses information about an example's k-nearest neighbors to classify unlabeled examples. For instance lets follow the below diagram to understand the underlying theory.



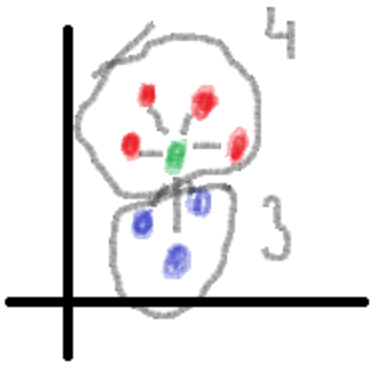
-Now in this diagram we see there are 3 red points and 3 blue points,

-We need to predict in which group does the green point lies. For which we use K-NN algorithm.

-KNN algorithm is used to classify unlabeled examples.

-Mostly used in classification problem.

-Number of nearest neighbors used i.e “k”.



- KNN algorithm treats features as coordinates in multidimensional plane.

- to locate the nearest neighbors we need a distance function to calculate the distance between the co-ordinates.

- by default: Euclidean distance, then voting is done to decide which class the unlabeled variable belongs.

- bias variance: balance between over fitting and underfitting.

- k-large reduces impact caused by noisy data.

- k-nn~lazy algorithm: depends on training instances, instance based learning.

**Euclidean distance :

$$\text{dist}(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2}$$

Breast Cancer Diagnosis with help of KNN Algorithm:

Machine Learning automates the diagnosis of cancer cells whether it is malignant of benign.

Breast cancer data: 569 samples of cancer biopsies. M – malignant, B- benign.

Data source: http://archive.ics.uci.edu/ml. (University of Wisconsin)

Exploring and Preparing the data:

Here is a brief screen shot of the data set: wbcd

K - Nearest -Neighbor Classification

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
1	842302	M	17.990	10.38	122.80	1001.0	0.11840	0.27760
2	842517	M	20.570	17.77	132.90	1326.0	0.08474	0.07864
3	84300903	M	19.690	21.25	130.00	1203.0	0.10960	0.15990
4	84348301	M	11.420	20.38	77.58	386.1	0.14250	0.28390
5	84358402	M	20.290	14.34	135.10	1297.0	0.10030	0.13280
6	843786	M	12.450	15.70	82.57	477.1	0.12780	0.17000
7	844359	M	18.250	19.98	119.60	1040.0	0.09463	0.10900
8	84458202	M	13.710	20.83	90.20	577.9	0.11890	0.16450
9	844981	M	13.000	21.82	87.50	519.8	0.12730	0.19320

View of the data : to verify whether 569 samples are present or not / 32 features are there.

```
> str(wbcd) #view of the data : 569 obs-rows/32-cols
'data.frame':   569 obs. of  32 variables:
 $ id                     : int  842302 842517 84300903 84348301 84358402 843786 844359 84458202 844981 84501001 ...
 $ 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 ...
 $ texture_se             : num  0.905 0.734 0.787 1.156 0.781 ...
 $ perimeter_se           : num  8.59 3.4 4.58 3.44 5.44 ...
 $ area_se                : num  153.4 74.1 94 27.2 94.4 ...
 $ smoothness_se          : num  0.0064 0.00522 0.00615 0.00911 0.01149 ...
 $ compactness_se         : num  0.049 0.0131 0.0401 0.0746 0.0246 ...
 $ concavity_se           : num  0.0537 0.0186 0.0383 0.0566 0.0569 ...
 $ concave.points_se      : num  0.0159 0.0134 0.0206 0.0187 0.0188 ...
 $ symmetry_se            : num  0.03 0.0139 0.0225 0.0596 0.0176 ...
 $ fractal_dimension_se   : num  0.00619 0.00353 0.00457 0.00921 0.00511 ...
 $ radius_worst           : num  25.4 25 23.6 14.9 22.5 ...
 $ texture_worst          : num  17.3 23.4 25.5 26.5 16.7 ...
 $ perimeter_worst        : num  184.6 158.8 152.5 98.9 152.2 ...
 $ area_worst             : num  2019 1956 1709 568 1575 ...
 $ smoothness_worst       : num  0.162 0.124 0.144 0.21 0.137 ...
 $ compactness_worst      : num  0.666 0.187 0.424 0.866 0.205 ...
 $ concavity_worst        : num  0.712 0.242 0.45 0.687 0.4 ...
 $ concave.points_worst   : num  0.265 0.186 0.243 0.258 0.163 ...
 $ symmetry_worst         : num  0.46 0.275 0.361 0.664 0.236 ...
 $ fractal_dimension_worst: num  0.1189 0.089 0.0876 0.173 0.0768 ...
```

```
> table(wbcd$diagnosis) #B->357,M->212

  B   M
357 212
```

357 samples are benign

212 samples are malignant

Proportion table gives the percentage of Benign vs. Malignant

```
> round(prop.table(table(wbcd$diagnosis)) * 100, digits = 1)

   Benign Malignant
     62.7      37.3
```

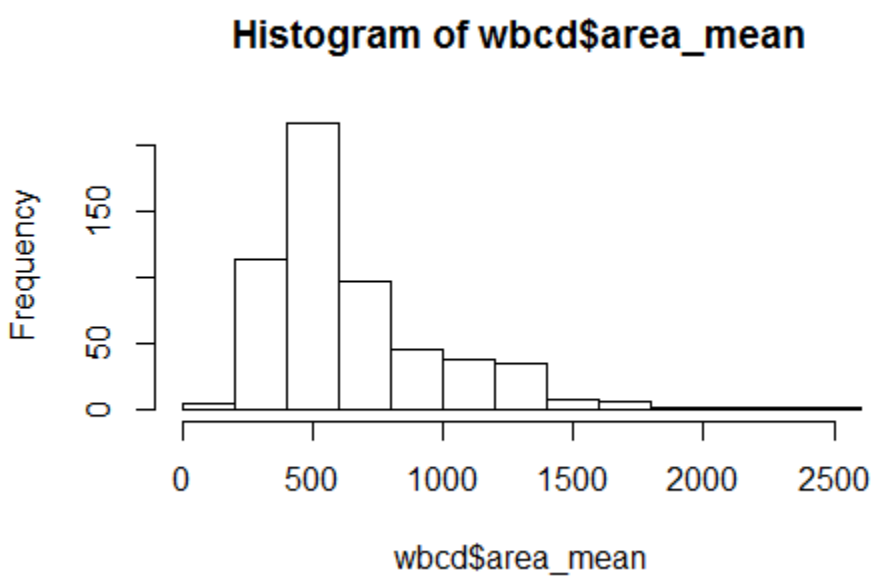
Diagnosis column is the label, other than this column all other columns are numeric, and we delete id column since it has no significance in the operation.

Taking summary of the following columns for instance gives us an insight which column will play the major role when the algorithm will be working~ since K-NN algorithm is based on distance measure, if we check the smoothness column min value is : .05 and max is .16, whereas for area : min 143 and ranges upto max: 2501 so since the range is wide it will have a higher impact over the algorithm.

K - Nearest -Neighbor Classification

```
> summary(wbcd[c("radius_mean", "area_mean", "smoothness_mean")])
  radius_mean       area_mean      smoothness_mean
 Min.   : 6.981   Min.   : 143.5   Min.   :0.05263
 1st Qu.:11.700   1st Qu.: 420.3   1st Qu.:0.08637
 Median :13.370   Median : 551.1   Median :0.09587
 Mean   :14.127   Mean   : 654.9   Mean   :0.09636
 3rd Qu.:15.780   3rd Qu.: 782.7   3rd Qu.:0.10530
 Max.   :28.110   Max.   :2501.0   Max.   :0.16340
```

The histogram shows mean of area_mean lies between around 500-600(most of the values are present)



Time to rescale, so that comparing the features become easy and they will lie within 0-1.

Note: The traditional method of rescaling features for k-NN is min-max normalization. This process transforms a feature such that all its values fall in a range between 0 and 1. The formula for normalizing a feature is as follows:

$$X_{new} = \frac{X - \min(X)}{\max(X) - \min(X)}$$

```
> normalize <- function(x){
+
+   return((x - min(x))/(max(x) - min(x)))
+ }
> normalize(c(1, 2, 3, 4, 5))
[1] 0.00 0.25 0.50 0.75 1.00
>
```

K - Nearest -Neighbor Classification

Our function works perfectly as the test shows. Now we normalize all the columns we have from column 2:31 we apply the normalization function.We remove the id column, since it is not relevant, so our first column is diagnosis. **The lapply() function takes a list and applies a specified function to each list element.** As a data frame is a list of equal-length vectors, we can use lapply() to apply **normalize()** to each feature in the data frame. The final step is to convert the list returned by lapply() to a data frame, using the as.data.frame() function.

Normalized columns.

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
1	0.52103744	0.02265810	0.54598853	0.36373277	0.5937528	0.79203730
2	0.64314449	0.27257355	0.61578329	0.50159067	0.2898799	0.18176799
3	0.60149557	0.39026040	0.59574321	0.44941676	0.5143089	0.43101650
4	0.21009040	0.36083869	0.23350149	0.10290562	0.8113208	0.81136127
5	0.62989256	0.15657761	0.63098611	0.48928950	0.4303512	0.34789277
6	0.25883856	0.20257017	0.26798424	0.14150583	0.6786133	0.46199620
7	0.53334280	0.34731146	0.52387534	0.38027572	0.3791640	0.27489111
8	0.31847224	0.37605681	0.32071039	0.18426299	0.5982667	0.44512607
9	0.28486914	0.40953669	0.30205238	0.15961824	0.6740995	0.53315748

```
> wbcd_n <- as.data.frame(lapply(wbcd[2:31], normalize))
> View(wbcd_n)
> summary(wbcd_n$area_mean)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.0000  0.1174  0.1729  0.2169  0.2711  1.0000 
>
```

Checking normalization was applied correctly. The dataset wbcd_n has 30 columns.

DataSet Preparation : Creating training and test dataset

There are 569 biopsy samples, we use 469 samples for training and 100 samples for test.

```
wbcd_test     100 obs. of 30 variables
wbcd_train    469 obs. of 30 variables
```

It is not interesting to predict what we know. Performance measures we obtain during the training may mislead as we do not know the extent to which cases have been overfitted or how well the learner will generalize to unseen cases. A more interesting question is how well our learner performs on a dataset of unlabeled data.
In the absence of such data, **we can simulate this scenario by dividing our data into two portions: a training dataset that will be used to build the k-NN model and a test dataset that will be used to estimate the predictive accuracy of the model.** We will use the first 469 records for the training dataset and the remaining 100 to simulate new patients.

K - Nearest -Neighbor Classification

Below we create label for training and test set.

```
wbcd_test_lab… chr [1:100] "B" "B" "B" "B" "B" "B" "B"…
wbcd_train_la… chr [1:469] "M" "M" "M" "M" "M" "M" "M"…
```

Training Model on the data

To classify our test instances, we will use a k-NN implementation from the class package, which provides a set of basic R functions for classification.

Install the package class. The knn() function in the class package provides a standard, classic implementation of the k-NN algorithm. For each instance in the test data, the function will identify the k-Nearest Neighbors, using Euclidean distance, where k is a user-specified number. The test instance is classified by taking a "vote" among the k-Nearest Neighbors—specifically, this involves assigning the class of the majority of the k neighbors.
Note: A tie vote is broken at random.

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)
```

Why k=21?

Training data includes **469 instances, we might try k = 21**, an odd number roughly equal to the square root of 469. With a two-category outcome, **using an odd number eliminates the chance of ending with a tie vote**.

K - Nearest -Neighbor Classification

```
#training model on the data

install.packages("class")
library(class)

wbcd_test_pred <- knn(train = wbcd_train, test = wbcd_test,
                      cl = wbcd_train_labels, k = 21)
```

```
> wbcd_test_pred
  [1] B B B B B B B B B B M B B B B B B B M B B B B M B B B B B M M B M B M B B B B B
 [41] M B B M B B B M M B B B M B B B B B B B B B B B M B M M B B B B B B B B B B B B
 [81] B B B B B B B B B B B B B M M M M M M B
Levels: B M
>
```

Performance Evaluation of the Model:

Installing : gmodel

```
install.packages("gmodels")
library(gmodels)
```

- **Confusion Matrix**

True Negative	**False Positive (False Alarm)**
False Negative	**True Positive**

The cell percentages in the table indicate the proportion of values that fall into four categories:

The **top-left cell indicates the true negative results**.

True Negative : 77 #benign mass

The **bottom-right cell indicates the true positive results**.

K - Nearest -Neighbor Classification

True Positive : 21 # malignant mass

The **lower-left cell are false negative results.**

False Negative : 2 #are malignant by nature but proved benign.

Note: Errors in this direction could be extremely costly as they might lead a patient to believe that she is cancer-free, but in reality, the disease may continue to spread.

The **top-right cell would contain the false positive results.**

False Positive : 0 #are begnign by nature but proved malignant **#false alarm**

```
> CrossTable(x = wbcd_test_labels, y = wbcd_test_pred,
+            prop.chisq=FALSE)

   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|

Total Observations in Table:  100

                 | wbcd_test_pred
wbcd_test_labels |         B |         M | Row Total |
-----------------|-----------|-----------|-----------|
               B |        77 |         0 |        77 |
                 |     1.000 |     0.000 |     0.770 |
                 |     0.975 |     0.000 |           |
                 |     0.770 |     0.000 |           |
-----------------|-----------|-----------|-----------|
               M |         2 |        21 |        23 |
                 |     0.087 |     0.913 |     0.230 |
                 |     0.025 |     1.000 |           |
                 |     0.020 |     0.210 |           |
-----------------|-----------|-----------|-----------|
    Column Total |        79 |        21 |       100 |
                 |     0.790 |     0.210 |           |
-----------------|-----------|-----------|-----------|
```

2 out of 100, or 2 percent of masses were incorrectly classified by the k-NN approach. While 98 percent accuracy seems impressive.

K - Nearest -Neighbor Classification

Improving Model Performance:

To standardize a vector, we can use the R's built-in **scale()** function, which, by default, rescales values using the **z-score standardization**. The scale() function offers the additional benefit that it can be applied directly to a data frame, so we can avoid the use of the lapply() function. To create a z-score standardized version of the wbcd data, we can use the following command:

z-transformed data `#wbcd_z <- as.data.frame(scale(wbcd[-1]))`

	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean
1	1.09609953	-2.071512e+00	1.26881726	0.983509520	1.56708746	3.280628064
2	1.82821197	-3.533215e-01	1.68447255	1.907030269	-0.82623545	-0.486643478
3	1.57849920	4.557859e-01	1.56512598	1.557513185	0.94138212	1.051999895
4	-0.76823332	2.535091e-01	-0.59216612	-0.763791736	3.28066684	3.399917422
5	1.74875791	-1.150804e+00	1.77501133	1.824623802	0.28012535	0.538866307
6	-0.47595587	-8.346009e-01	-0.38680772	-0.505205927	2.23545452	1.243241565
7	1.16987830	1.605082e-01	1.13712450	1.094332010	-0.12302797	0.088217620
8	-0.11841259	3.581350e-01	-0.07280278	-0.218772414	1.60263890	1.139100062
9	-0.31988539	5.883121e-01	-0.18391855	-0.383869508	2.19990308	1.682529360
10	-0.47311823	1.104467e+00	-0.32919213	-0.508615849	1.58130803	2.561104951
11	0.53708344	9.184652e-01	0.44162208	0.406095932	-1.01679116	-0.712914562

```
> summary(wbcd_z$area_mean)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
-1.4530 -0.6666 -0.2949  0.0000  0.3632  5.2460 
>
```

A z-transformed data always have mean =0,and the range is very compact -3 to 3.

Why use Z-score standardization?

Although normalization is traditionally used for k-NN classification, it may not always be the most appropriate way to rescale features. Since the z-score standardized values have no predefined minimum and maximum, extreme values are not compressed towards the center. One might suspect that with a malignant tumor, we might see some very extreme outliers as the tumors grow uncontrollably.It might, therefore, be reasonable to allow the outliers to be weighted more heavily in the distance calculation. Let's see whether z-score standardization can improve our predictive accuracy.

We create – train, test dataset and then as before we use knn to predict the class label.

K - Nearest -Neighbor Classification

```
> wbcd_z_train <- wbcd_z[1:469, ]
> wbcd_z_test <- wbcd_z[470:569, ]
> wbcd_z_train_labels <- wbcd[1:469, 1]
> wbcd_z_test_labels <- wbcd[470:569, 1]
> wbcd_z_test_pred <- knn(train = wbcd_z_train, test = wbcd_z_test,
+                         cl = wbcd_z_train_labels, k = 21)
> wbcd_z_test_pred
  [1] B B B B B B B B B B M B B B B B B B M B B B B M B B B B B M M B M B M B B B B B
 [41] M B B M B B B M M B B B M B B B B B B B B B B B M B M M B B B B B B B B B B B B
 [81] B B B B B B B B B B B B B M M M M M M B
Levels: B M
>
```

Unfortunately using z-transform didn't change anything.Still there are 2 false negatives.

```
> CrossTable(x = wbcd_z_test_labels, y = wbcd_z_test_pred,
+            prop.chisq = FALSE)


   Cell Contents
|-------------------------|
|                       N |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|


Total Observations in Table:  100


                    | wbcd_z_test_pred
wbcd_z_test_labels |         B |         M | Row Total |
-------------------|-----------|-----------|-----------|
                 B |        77 |         0 |        77 |
                   |     1.000 |     0.000 |     0.770 |
                   |     0.975 |     0.000 |           |
                   |     0.770 |     0.000 |           |
-------------------|-----------|-----------|-----------|
                 M |         2 |        21 |        23 |
                   |     0.087 |     0.913 |     0.230 |
                   |     0.025 |     1.000 |           |
                   |     0.020 |     0.210 |           |
-------------------|-----------|-----------|-----------|
      Column Total |        79 |        21 |       100 |
                   |     0.790 |     0.210 |           |
-------------------|-----------|-----------|-----------|
```

Summary : k-NN does not do any learning. It simply stores the training data. Unlabeled test examples are then matched to the most similar records in the training set using a distance function, unlabeled example is assigned the label of its neighbors. Though knn is a very simple classfier it classified malignant against benign 98% correctly, in both cases.