Introduction to Data Science CS61 June 12 - July 12, 2018

Dr. Ash Pahwa

Lesson 7: Classification - kNN

Lesson 7.1: kNN Modeling Method

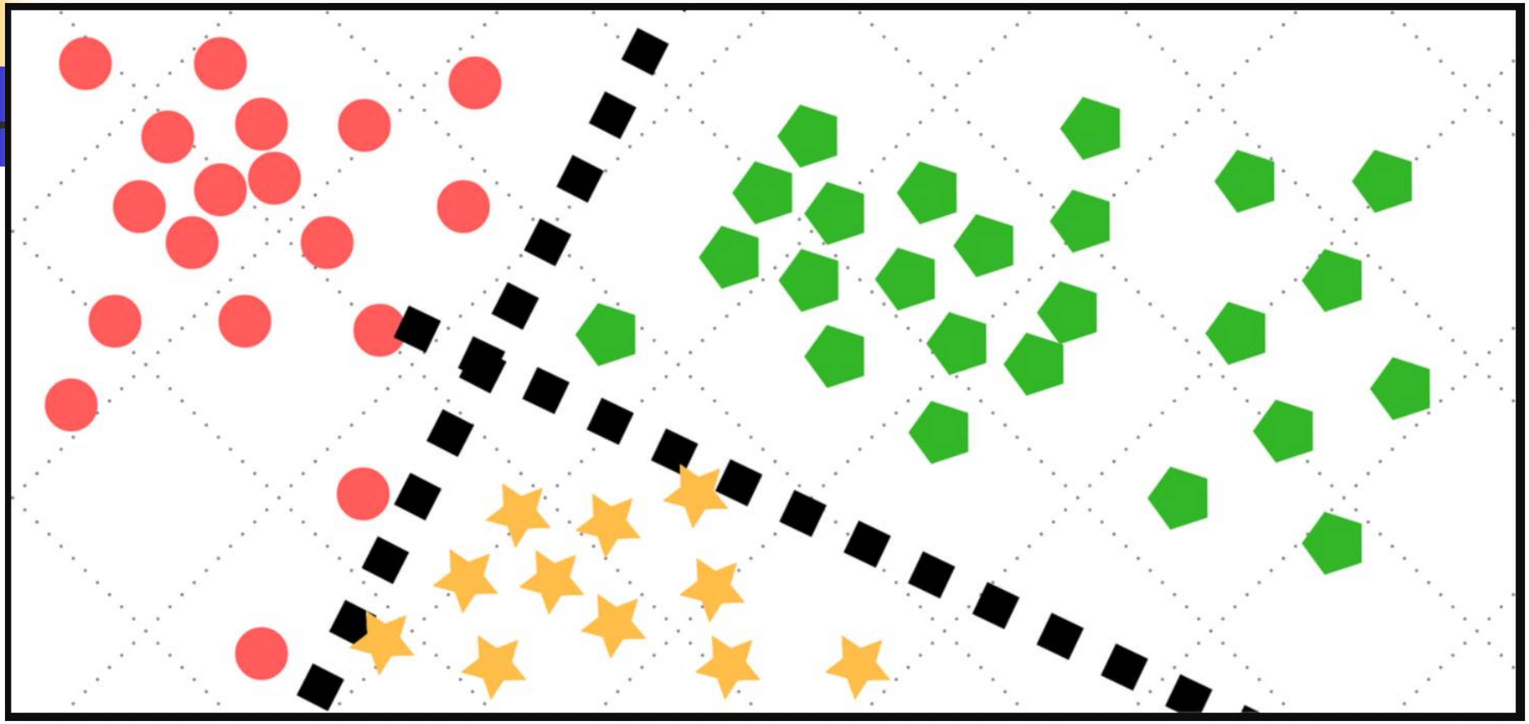
k Nearest Neighbor

Outline

- Similarity Based Learning
- The Bayes Classifier
- ‘k’ Nearest Neighbor (kNN) Model
- kNN Model Assessment
- Data Normalization:
  Standardization & Scaling
- kNN in R
  - Example 1: Dataset = Food, No Package
  - Example 2: Dataset = Food, Package = Class
  - Example 3: Dataset = Iris, Package = Class
- kNN in Python
  - Example 4: Dataset = Food, No Package
  - Example 5: Dataset = Food, Package = Scikit-Learn

Similarity Based Learning

Classification



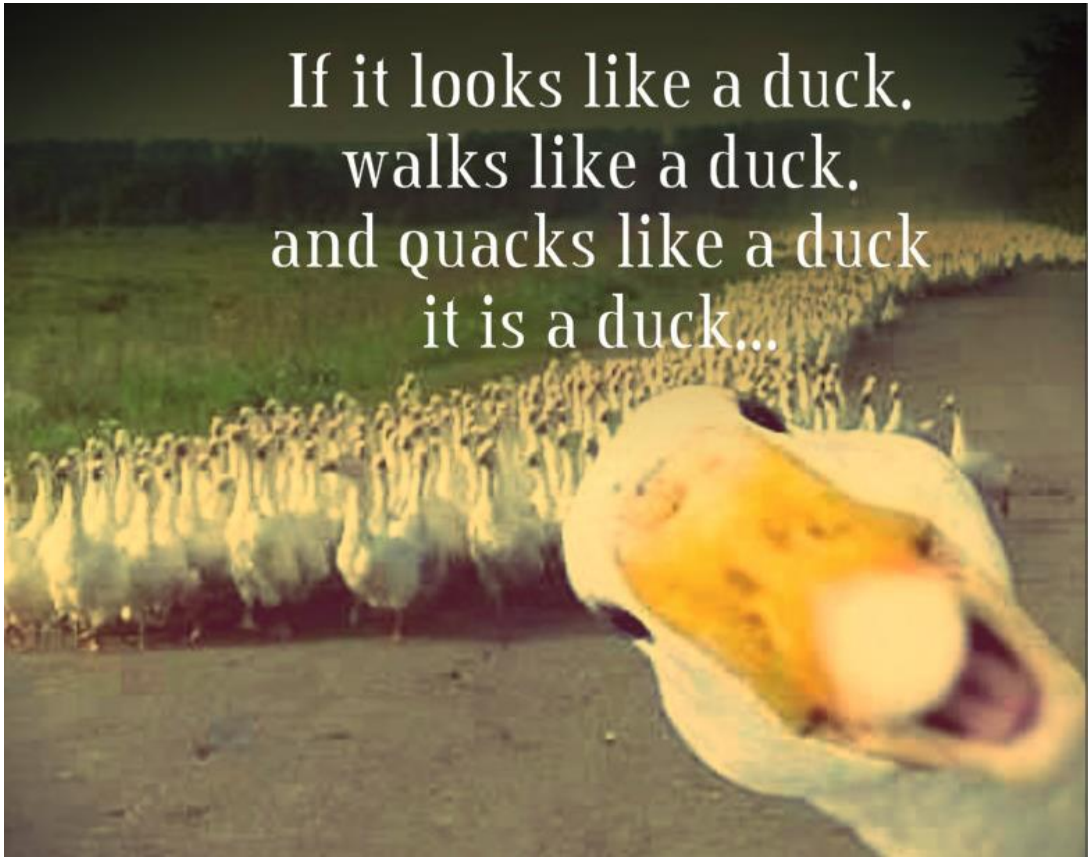
Characteristics:

- Color: Red, Green, Yellow
- Shape: Circle, Pentagon, Star

A new object is given to us:

- **Determine which class does it belong to?**
- **Compute the boundaries between classes**

Similarity Based Learning

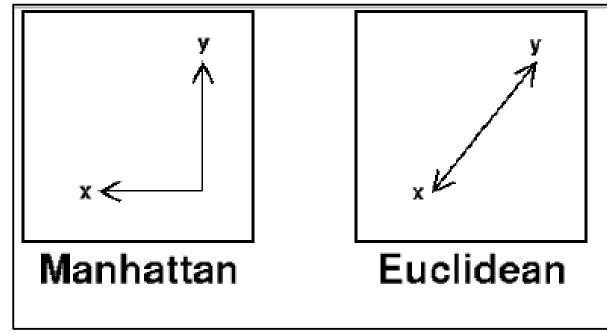


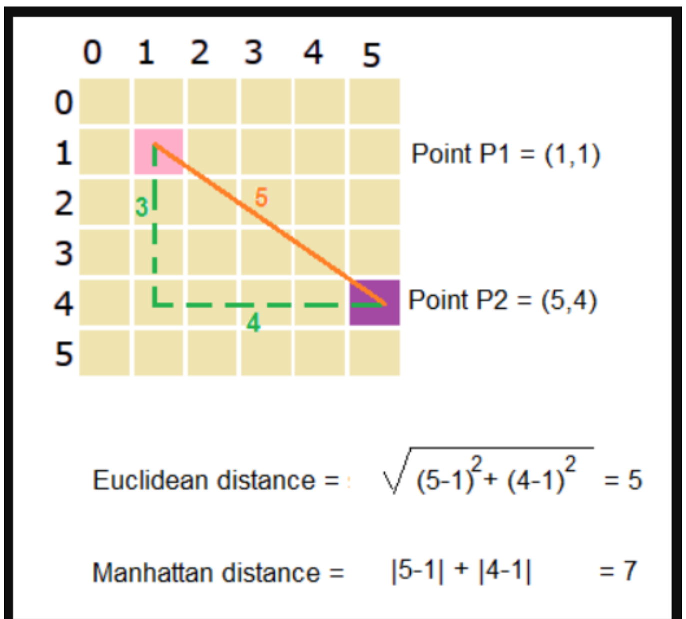
Similarity Based Learning

- Compute the distance matrices between objects

$$Euclidean\ distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$Manhattan\ distance = |x_2 - x_1| + |y_2 - y_1|$$

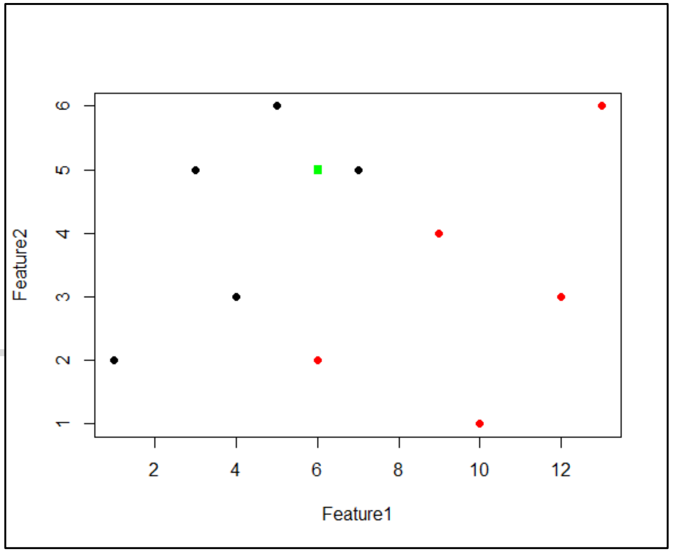




The Bayes Classifier

Based on “Thomas Bayes” Theorem

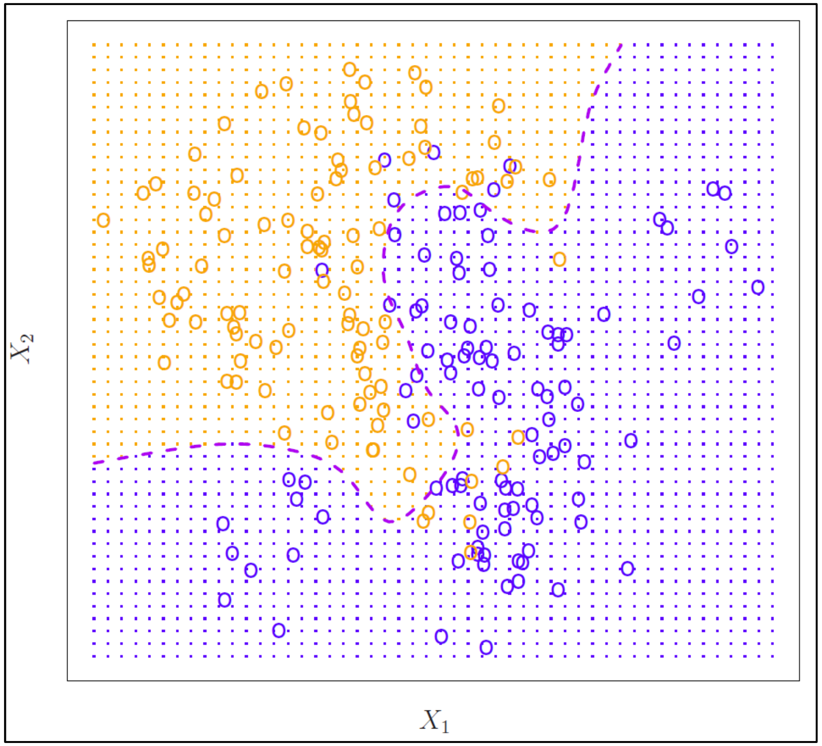
Classification



- Assumption#1: 2 classes $y = \{0,1\}$
- Assumption#2: Data distribution is Gaussian (Normal Distribution)
- Compute the following
  - Given the point x_0 what is the probability that belong to class 0
    - $P(y = 0|x = x_0)$
  - Given the point x_0 what is the probability that belong to class 1
    - $P(y = 1|x = x_0)$
  - $P(y = 0|x = x_0) + P(y = 1|x = x_0) = 1$
- $If\ P(y = 0|x = x_0) > P(y = 1|x = x_0)$
  - The new object belongs to class 0
- $If\ P(y = 0|x = x_0) < P(y = 1|x = x_0)$
  - The new object belongs to class 1

The Bayes Classifier Decision Boundary

- At decision boundary both the probabilities would be the same
- $P(y = 0|x = x_0) = P(y = 1|x = x_0)$



k Nearest Neighbor (kNN)

kNN Model

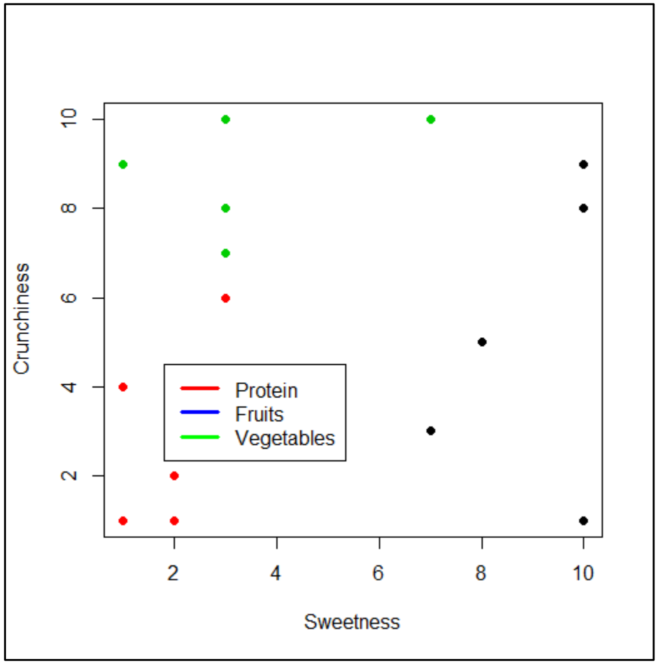
- Parametric or Non-Parametric
  - Non-Parametric Model
    - No assumption is made about the model parameters when we start
- Flexibility vs Ability of Interpret Results
  - Flexible = Function of ‘k’
    - When the value of ‘k’ is low = more flexible
    - When the value of ‘k’ is high = less flexible
  - Ability to interpret results = low
- Supervised or Unsupervised
  - Supervised Model
    - Response variable is needed
- Regression vs Classification
  - Classification

Pros and Cons of kNN

Pros	Cons
Simple and Effective	Does not produce a model, limiting the ability to understand how the features are related to the class
Makes no assumption about the underlying data distribution Non-parametric	Requires selection of an appropriate value of ‘k’

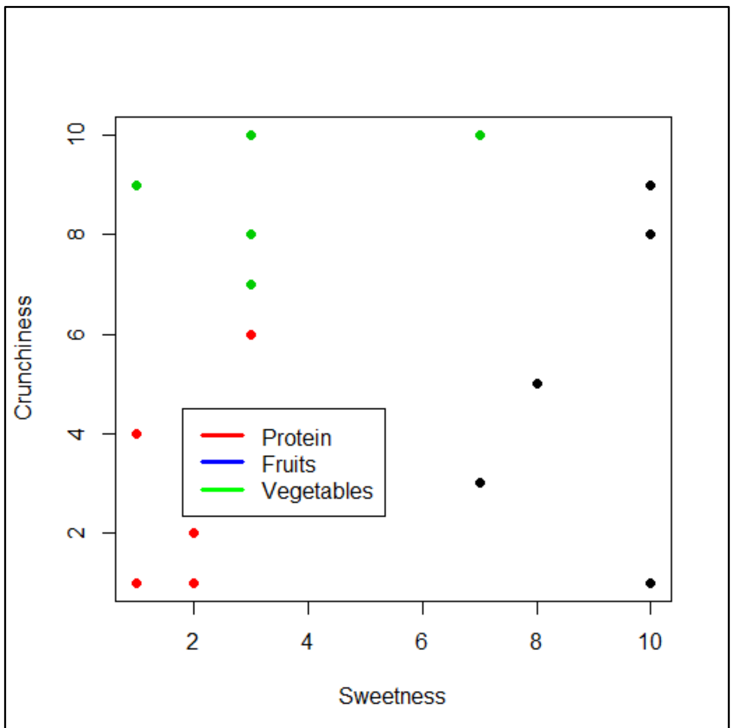
Example

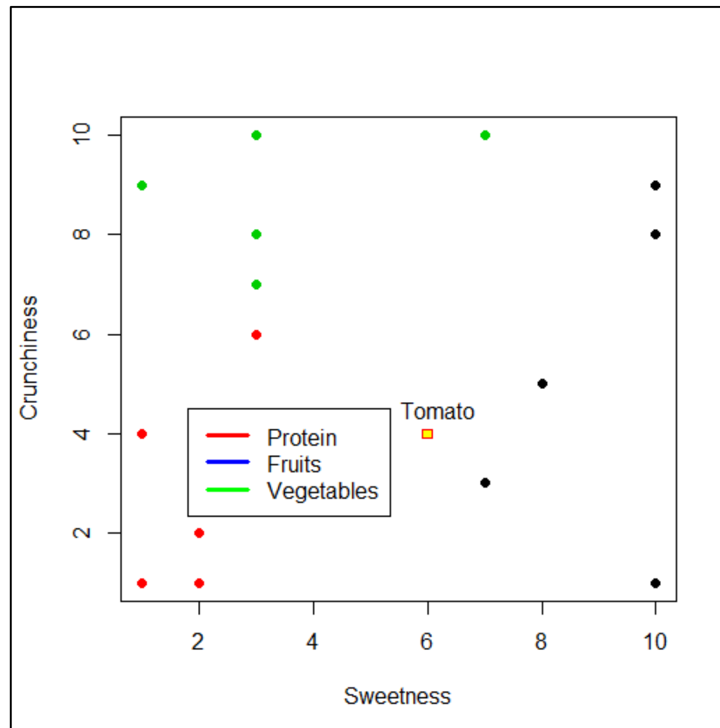
#	Item	Sweetness	Crunchiness	Food Type
1	Apple	10	9	Fruit
2	Bacon	1	4	Protein
3	Banana	10	1	Fruit
4	Carrot	7	10	Vegetable
5	Celery	3	10	Vegetable
6	Cheese	1	1	Protein
7	Grape	8	5	Fruit
8	Green bean	3	7	Vegetable
9	Nuts	3	6	Protein
10	Orange	7	3	Fruit
11	Lettuce	1	9	Vegetable
12	Cucumber	3	8	Vegetable
13	Shrimp	2	2	Protein
14	Fish	2	1	Protein
15	Pear	10	8	Fruit



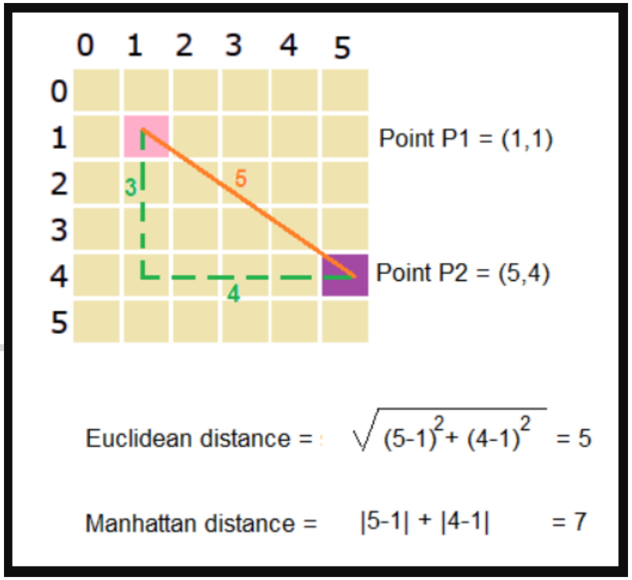
Predict
Tomato: Fruit or Vegetable?

- Tomato
  - Sweetness = 6
  - Crunchiness = 4





Distance Function
* Euclidean
* Manhattan



- **Test data and Training Dataset with 2 dimensions**
  - $Test\ Case:\ Point\ p = (p_1, p_2)$
  - $Training\ Data:\ Point\ q_1 = (q_{11}, q_{12})$
  - $Training\ Data:\ Point\ q_2 = (q_{21}, q_{22})$
  - $\ldots$
  - $Training\ Data:\ Point\ q_n = (q_{n1}, q_{n2})$
- $\boldsymbol{Euclidean\ Distance}\ between\ p\ and\ q_1 = \sqrt{(p_1 - q_{11})^2 + (p_2 - q_{12})^2}$
- --
- Distance formula with k dimensions
  - $\boldsymbol{Euclidean\ Distance}\ between\ p\ and\ q_1 = \sqrt{\sum_1^k (p_i - q_{1i})}$
  - $\boldsymbol{Manhattan\ Distance}\ between\ p\ and\ q_1 = \sum_1^k |p_i - q_{1i}|$

Euclidean Distance from Tomato

Distance between Apple and Tomato

$$Euclidean\ distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$= \sqrt{(10-6)^2 + (9-4)^2} = \sqrt{4^2 + 5^2} = \sqrt{41} = 6.4$$

Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato
1	Apple	10	9	Fruit	6.4
2	Bacon	1	4	Protein	5.0
3	Banana	10	1	Fruit	5.0
4	Carrot	7	10	Vegetable	6.1
5	Celery	3	10	Vegetable	6.7
6	Cheese	1	1	Protein	5.8
7	Grape	8	5	Fruit	2.2
8	Green bean	3	7	Vegetable	4.2
9	Nuts	3	6	Protein	3.6
10	Orange	7	3	Fruit	1.4
11	Lettuce	1	9	Vegetable	7.1
12	Cucumber	3	8	Vegetable	5.0
13	Shrimp	2	2	Protein	4.5
14	Fish	2	1	Protein	5.0
15	Pear	10	8	Fruit	5.7

Test Data

	Tomato	6	4		

Sorted Distance

Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato
10	Orange	7	3	Fruit	1.4
7	Grape	8	5	Fruit	2.2
9	Nuts	3	6	Protein	3.6
8	Green bean	3	7	Vegetable	4.2
13	Shrimp	2	2	Protein	4.5
2	Bacon	1	4	Protein	5.0
3	Banana	10	1	Fruit	5.0
12	Cucumber	3	8	Vegetable	5.0
14	Fish	2	1	Protein	5.0
15	Pear	10	8	Fruit	5.7
6	Cheese	1	1	Protein	5.8
4	Carrot	7	10	Vegetable	6.1
1	Apple	10	9	Fruit	6.4
5	Celery	3	10	Vegetable	6.7
11	Lettuce	1	9	Vegetable	7.1

K=1

Pick the top 1 entry

Votes: Fruit = 1

Final Result = Tomato is a Fruit

Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato
10	Orange	7	3	Fruit	1.4
7	Grape	8	5	Fruit	2.2
9	Nuts	3	6	Protein	3.6
8	Green bean	3	7	Vegetable	4.2
13	Shrimp	2	2	Protein	4.5
2	Bacon	1	4	Protein	5.0
3	Banana	10	1	Fruit	5.0
12	Cucumber	3	8	Vegetable	5.0
14	Fish	2	1	Protein	5.0
15	Pear	10	8	Fruit	5.7
6	Cheese	1	1	Protein	5.8
4	Carrot	7	10	Vegetable	6.1
1	Apple	10	9	Fruit	6.4
5	Celery	3	10	Vegetable	6.7
11	Lettuce	1	9	Vegetable	7.1

K=3

Pick the top 3 entries

Votes: Fruit = 2, Protein = 1

Final Result = Tomato is a Fruit

Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato
10	Orange	7	3	Fruit	1.4
7	Grape	8	5	Fruit	2.2
9	Nuts	3	6	Protein	3.6
8	Green bean	3	7	Vegetable	4.2
13	Shrimp	2	2	Protein	4.5
2	Bacon	1	4	Protein	5.0
3	Banana	10	1	Fruit	5.0
12	Cucumber	3	8	Vegetable	5.0
14	Fish	2	1	Protein	5.0
15	Pear	10	8	Fruit	5.7
6	Cheese	1	1	Protein	5.8
4	Carrot	7	10	Vegetable	6.1
1	Apple	10	9	Fruit	6.4
5	Celery	3	10	Vegetable	6.7
11	Lettuce	1	9	Vegetable	7.1

K=9

Pick the top 9 entries

Votes: Fruit = 3, Protein = 4, Vegetable = 2

Final Result = Tomato is a Protein

Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato
10	Orange	7	3	Fruit	1.4
7	Grape	8	5	Fruit	2.2
9	Nuts	3	6	Protein	3.6
8	Green bean	3	7	Vegetable	4.2
13	Shrimp	2	2	Protein	4.5
2	Bacon	1	4	Protein	5.0
3	Banana	10	1	Fruit	5.0
12	Cucumber	3	8	Vegetable	5.0
14	Fish	2	1	Protein	5.0
15	Pear	10	8	Fruit	5.7
6	Cheese	1	1	Protein	5.8
4	Carrot	7	10	Vegetable	6.1
1	Apple	10	9	Fruit	6.4
5	Celery	3	10	Vegetable	6.7
11	Lettuce	1	9	Vegetable	7.1

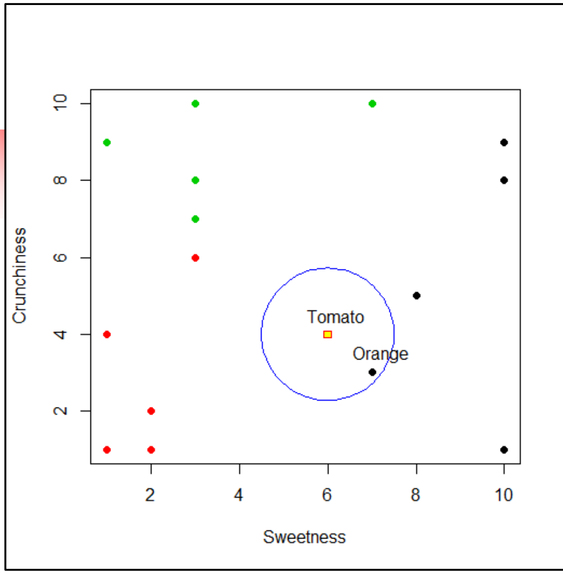
Results: Count the Votes

- k=1, Fruit (Fruit = 1)
- k=3, Fruit (Fruit = 2, Protein = 1)
- k=5, Fruit or Protein (Fruit = 2, Protein = 2, Vegetable = 1)
- k=9, Protein (Fruit = 3, Protein = 4, Vegetable = 2)

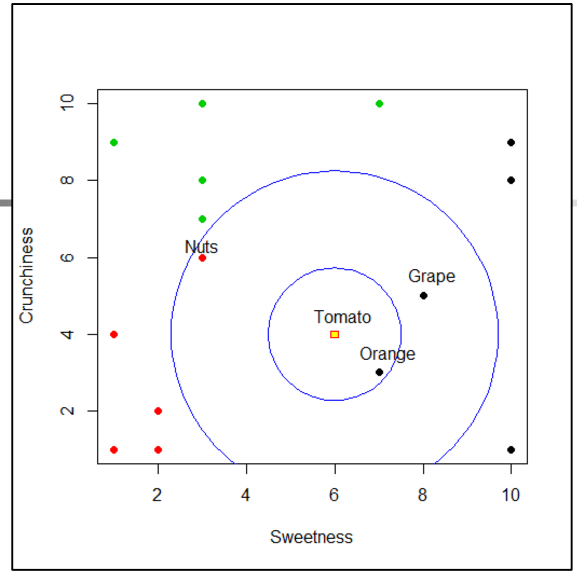
Training Data

#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato	k=1, #votes	k=3, #votes	k=5, #votes
10	Orange	7	3	Fruit	1.4	Fruit=1		
7	Grape	8	5	Fruit	2.2			
9	Nuts	3	6	Protein	3.6		Fruit=2, Protein=1	
8	Green bean	3	7	Vegetable	4.2			
13	Shrimp	2	2	Protein	4.5			Fruit =2. Protein=2, Vegetable=1
2	Bacon	1	4	Protein	5.0			
3	Banana	10	1	Fruit	5.0			
12	Cucumber	3	8	Vegetable	5.0			
14	Fish	2	1	Protein	5.0			
15	Pear	10	8	Fruit	5.7			
6	Cheese	1	1	Protein	5.8			
4	Carrot	7	10	Vegetable	6.1			
1	Apple	10	9	Fruit	6.4			
5	Celery	3	10	Vegetable	6.7			
11	Lettuce	1	9	Vegetable	7.1			

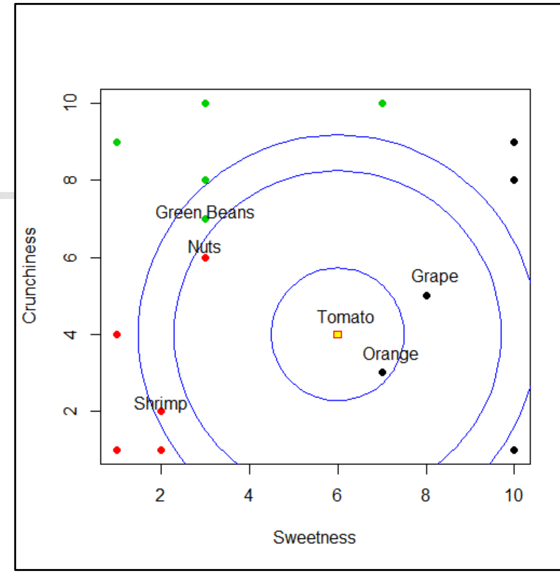
Results: Visualization



K=1



K=3



K=5

Training Data

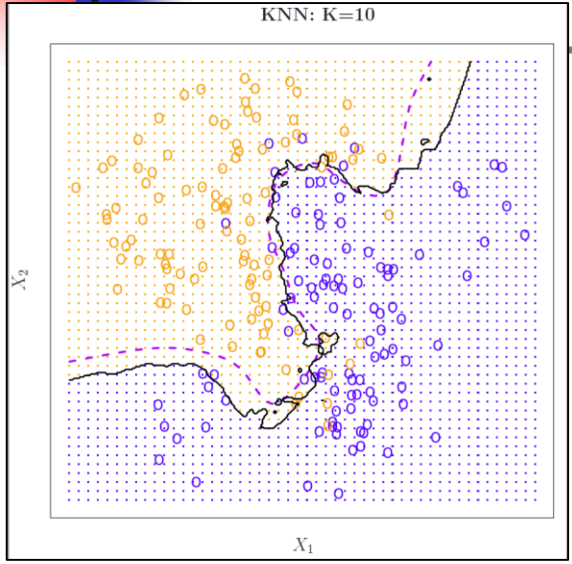
#	Item	Sweetness	Crunchiness	Food Type	Euclidean Distance From Tomato	k=1, #votes	k=3, #votes	k=5, #votes
10	Orange	7	3	Fruit	1.4	Fruit=1		
7	Grape	8	5	Fruit	2.2			
9	Nuts	3	6	Protein	3.6		Fruit=2, Protein=1	
8	Green bean	3	7	Vegetable	4.2			
13	Shrimp	2	2	Protein	4.5			Fruit =2. Protein=2, Vegetable=1
2	Bacon	1	4	Protein	5.0			
3	Banana	10	1	Fruit	5.0			
12	Cucumber	3	8	Vegetable	5.0			
14	Fish	2	1	Protein	5.0			
15	Pear	10	8	Fruit	5.7			
6	Cheese	1	1	Protein	5.8			
4	Carrot	7	10	Vegetable	6.1			
1	Apple	10	9	Fruit	6.4			
5	Celery	3	10	Vegetable	6.7			
11	Lettuce	1	9	Vegetable	7.1			

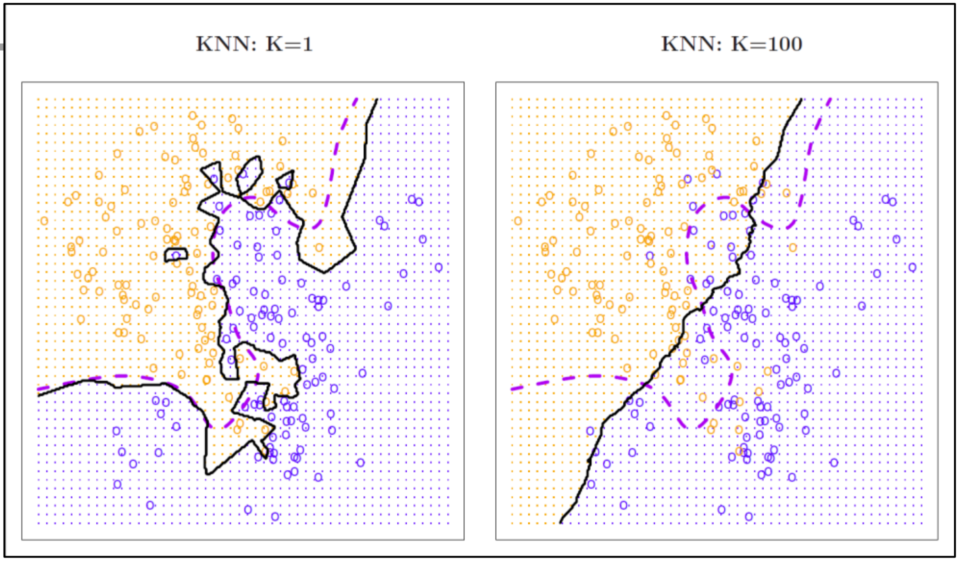
kNN Model Assessment

How to decide the value of ‘k’?

What Should be the Value of 'k'?

Bayes Decision Boundary: Purple dashed line





kNN: k=10: Black Line

kNN: k=1: Black Line

kNN: k=100: Black Line

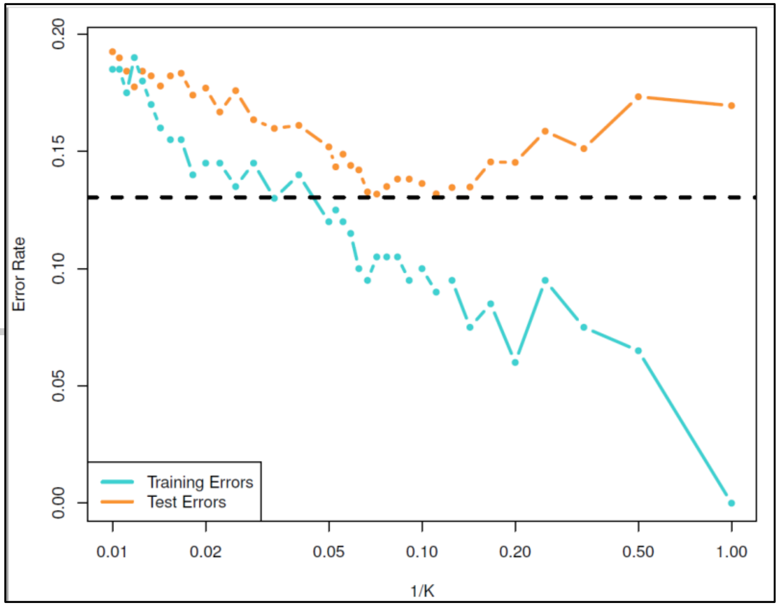
- If the natural boundary is non-linear
  - Lower value of 'k' is better
- If the natural boundary is linear
  - Higher value of 'k' is better

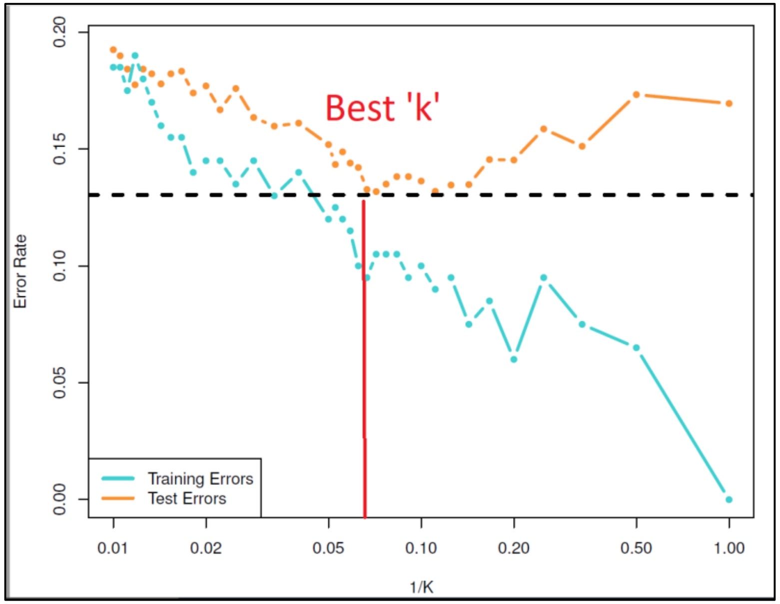
Bias Variance Tradeoff

- If 'k' = 1
  - Decision boundary is overly flexible
  - Model finds patterns in the data that don't correspond to the Bayes boundary
  - Low bias but high variance
- If 'k' = 100 (large)
  - Every training instance is represented in the final vote –
  - All testing data will be classified as the class with majority votes
  - Decision boundary becomes close to linear
  - Model becomes less flexible
  - Low variance but high bias

Testing Error as a Function of ‘k’

- Best ‘k’
  - 1/k = 0.07
  - ‘k’=14





Data Normalization: Standardization & Scaling

In kNN Modeling we usually either Standardize or Scale the data

Data Standardization & Scaling

- Suppose we have 2 data items
  - Height: varies from 4 – 7 feet
  - Net Worth: $10,000 - $100B
- If we use both the variables in a model
  - Net Worth will dominate because it contains large values
- Solution
  - Standardize
  - Scale

Data Standardization and Scaling

- Standardization Data Variation
  - -3 to +3

$$z = \frac{\text{Data Value - Mean}}{\text{Standard Deviation}} = \frac{y - \mu}{\sigma}$$

- Scaling Data Variation
  - 0 to 1

$$y_i^j = \frac{x_i^j - min_j}{max_j - min_j}$$

Example

```
> normalize = function(x) {
+   return(  (x-min(x))/(max(x)-min(x)))}
> data = c(124,3,311,341,298,136,23,75,5,51,822,364,663,444,999)
> (standard.data = scale(data))
              [,1]
 [1,] -0.603086904
 [2,] -0.994156118
 [3,]  0.001292791
 [4,]  0.098252100
 [5,] -0.040722910
 [6,] -0.564303180
 [7,] -0.929516578
 [8,] -0.761453776
 [9,] -0.987692164
[10,] -0.839021223
[11,]  1.652833026
[12,]  0.172587571
[13,]  1.138948687
[14,]  0.431145729
[15,]  2.224892951
attr(,"scaled:center")
[1] 310.6
attr(,"scaled:scale")
[1] 309.4081
> (normalized.data = normalize(data))
 [1] 0.121485944 0.000000000 0.309236948 0.339357430 0.296184739
0.133534137 0.020080321 0.072289157 0.002008032 0.048192771 0.822289157
[12] 0.362449799 0.662650602 0.442771084 1.000000000
>
```

	A	B	C	D	E	F
1		#	Data	Standardization		Scaling
2		1	124	-0.60		0.12
3		2	3	-0.99		0.00
4		3	311	0.00		0.31
5		4	341	0.10		0.34
6		5	298	-0.04		0.30
7		6	136	-0.56		0.13
8		7	23	-0.93		0.02
9		8	75	-0.76		0.07
10		9	5	-0.99		0.00
11		10	51	-0.84		0.05
12		11	822	1.65		0.82
13		12	364	0.17		0.36
14		13	663	1.14		0.66
15		14	444	0.43		0.44
16		15	999	2.22		1
17						
18		Mean	310.60		Minimum	3
19		StdDev	309.41		Maximum	999
20						

Building kNN Model in R

Example 1

Dataset = Food

Method: Build the kNN algorithm using R

Example 1: Food Dataset
Read the Dataset

```
> # import the CSV file
> food <- read.csv("03 food16.csv")
> food
   X.        Item Sweetness Crunchiness Food.Type
1   1       Apple        10           9     Fruit
2   2       Bacon         1           4   Protein
3   3      Banana        10           1     Fruit
4   4      Carrot         7          10 Vegetable
5   5      Celery         3          10 Vegetable
6   6      Cheese         1           1   Protein
7   7       Grape         8           5     Fruit
8   8 Green bean          3           7 Vegetable
9   9        Nuts         3           6   Protein
10 10      Orange         7           3     Fruit
11 11     Lettuce         1           9 Vegetable
12 12    Cucumber         3           8 Vegetable
13 13      Shrimp         2           2   Protein
14 14        Fish         2           1   Protein
15 15        Pear        10           8     Fruit
16 16      Tomato         6           4          
> table(food$Food.Type)

              Fruit   Protein Vegetable 
        1         5         5         5 
> 
```

Example 1: Food Dataset Separate Train and Test Data

```
> trainData <- food[1:15,3:5]
> trainData
   Sweetness Crunchiness Food.Type
1         10           9     Fruit
2          1           4   Protein
3         10           1     Fruit
4          7          10 Vegetable
5          3          10 Vegetable
6          1           1   Protein
7          8           5     Fruit
8          3           7 Vegetable
9          3           6   Protein
10         7           3     Fruit
11         1           9 Vegetable
12         3           8 Vegetable
13         2           2   Protein
14         2           1   Protein
15        10           8     Fruit
> testData <- food[16:16,3:5]
> testData
   Sweetness Crunchiness Food.Type
16         6           4
>
```

Example 1: Food Dataset
Compute the Distance : Sort

```
> test.count = 1
> sum = rep(0,dim(trainData)[1])
> for ( i in 1:dim(trainData)[1]  )  {
+   sum[i] = sum[i] + (trainData$Sweetness[i] - testData$Sweetness[test.count])^2
+   sum[i] = sum[i] + (trainData$Crunchiness[i] - testData$Crunchiness[test.count])^2
+ }
> sum
 [1] 41 25 25 37 45 34  5 18 13  2 50 25 20 25 32
> (trainData$dist = sqrt(sum))
 [1] 6.403124 5.000000 5.000000 6.082763 6.708204 5.830952 2.236068 4.242641 3.605551
1.414214 7.071068 5.000000 4.472136 5.000000 5.656854
> (trainData_Sorted = trainData[order(trainData$dist),])
   Sweetness Crunchiness Food.Type     dist
10         7           3     Fruit 1.414214
7          8           5     Fruit 2.236068
9          3           6   Protein 3.605551
8          3           7 Vegetable 4.242641
13         2           2   Protein 4.472136
2          1           4   Protein 5.000000
3         10           1     Fruit 5.000000
12         3           8 Vegetable 5.000000
14         2           1   Protein 5.000000
15        10           8     Fruit 5.656854
6          1           1   Protein 5.830952
4          7          10 Vegetable 6.082763
1         10           9     Fruit 6.403124
5          3          10 Vegetable 6.708204
11         1           9 Vegetable 7.071068
>
```

Example 1: Food Dataset Retrieve the Nearest Neighbor

- k=1, Fruit (Fruit = 1)
- k=3, Fruit (Fruit = 2, Protein = 1)

```
> ################################################
> k = 1
> (nearestNeighbour = as.character(trainData_Sorted$Food.Type[1:k]))
[1] "Fruit"
> table(nearestNeighbour)
nearestNeighbour
Fruit
    1
> ###############################################
> k = 3
> (nearestNeighbour = as.character(trainData_Sorted$Food.Type[1:k]))
[1] "Fruit"   "Fruit"   "Protein"
> table(nearestNeighbour)
nearestNeighbour
  Fruit Protein
      2       1
```

Example 1: Food Dataset
Retrieve the Nearest Neighbor

- k=5, Fruit or Protein (Fruit = 2, Protein = 2, Vegetable = 1)
- k=9, Protein (Fruit = 3, Protein = 4, Vegetable = 2)

```
> ################################################
> k = 5
> (nearestNeighbour = as.character(trainData_Sorted$Food.Type[1:k]))
[1] "Fruit"     "Fruit"     "Protein"   "Vegetable" "Protein"
> table(nearestNeighbour)
nearestNeighbour
    Fruit   Protein Vegetable
        2         2         1
> ################################################
> k = 9
> (nearestNeighbour = as.character(trainData_Sorted$Food.Type[1:k]))
[1] "Fruit"     "Fruit"     "Protein"   "Vegetable" "Protein"   "Protein"   "Fruit"
"Vegetable" "Protein"
> table(nearestNeighbour)
nearestNeighbour
    Fruit   Protein Vegetable
        3         4         2
```

Example 2

Dataset = Food

Package: R/Class

Example 1: Food Dataset Package: Class

```
> ##############################################################
> # import the CSV file
> food <- read.csv("03 food16.csv")
> food
   X.        Item Sweetness Crunchiness Food.Type
1   1       Apple        10           9     Fruit
2   2       Bacon         1           4   Protein
3   3      Banana        10           1     Fruit
4   4      Carrot         7          10 Vegetable
5   5      Celery         3          10 Vegetable
6   6      Cheese         1           1   Protein
7   7       Grape         8           5     Fruit
8   8  Green bean         3           7 Vegetable
9   9        Nuts         3           6   Protein
10 10      Orange         7           3     Fruit
11 11     Lettuce         1           9 Vegetable
12 12    Cucumber         3           8 Vegetable
13 13      Shrimp         2           2   Protein
14 14        Fish         2           1   Protein
15 15        Pear        10           8     Fruit
16 16      Tomato         6           4
> table(food$Food.Type)

              Fruit   Protein Vegetable
        1         5         5         5
```

Example 1: Food Dataset

```
> train <- food[1:15,3:4]
> train
   Sweetness Crunchiness
1         10           9
2          1           4
3         10           1
4          7          10
5          3          10
6          1           1
7          8           5
8          3           7
9          3           6
10         7           3
11         1           9
12         3           8
13         2           2
14         2           1
15        10           8
> test <- food[16:16,3:4]
> test
   Sweetness Crunchiness
16         6           4
>
```

Use ‘Class’ Library: kNN Function

- k=1, Fruit (Fruit = 1)
- k=3, Fruit (Fruit = 2, Protein = 1)
- k=5, Fruit or Protein (Fruit = 2, Protein = 2, Vegetable = 1)
- k=9, Protein (Fruit = 3, Protein = 4, Vegetable = 2)

```
> # load the "class" library
> library(class)
> ##########################################
> test_pred <- knn(train = train, test = test,
+                       cl = train_labels, k = 1)
> test_pred
[1] Fruit
Levels:  Fruit Protein Vegetable
> ##########################################
> test_pred <- knn(train = train, test = test,
+                  cl = train_labels, k = 3)
> test_pred
[1] Fruit
Levels:  Fruit Protein Vegetable
> ##########################################
> test_pred <- knn(train = train, test = test,
+                  cl = train_labels, k = 5)
> test_pred
[1] Fruit
Levels:  Fruit Protein Vegetable
> ##########################################
> test_pred <- knn(train = train, test = test,
+                  cl = train_labels, k = 9)
> test_pred
[1] Protein
Levels:
```

Example 3

Dataset = Iris

Package: R/Class

Iris Dataset

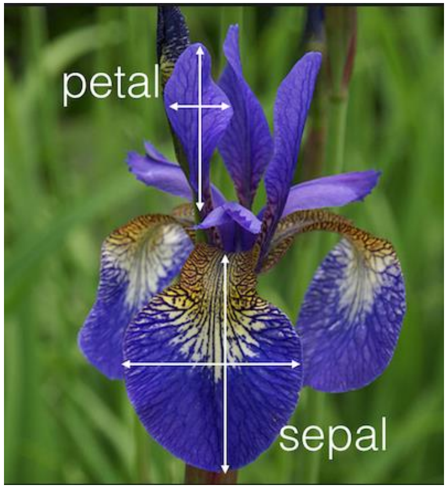
- Edgar Anderson's Iris Data
- Iris Species
  - Setosa, Versicolor, Virginica
- Sepal + Petal length and width in centimeters
- 150 records (50 flowers from each 3 species)

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Iris Virginica

Iris Versicolor

Iris Setosa



Iris Dataset

```
> dim(iris)
[1] 150   5
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species  
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500                  
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
> names(iris)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
> str(iris)
'data.frame':       150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1
...
> table(iris$Species)

    setosa versicolor  virginica 
        50         50         50 
>
```

Shuffle the Dataset

```
> ####################################
> # 1. Mix all the rows
> # Like shuffle deck of cards
> #
> set.seed(9850)
> gp = runif(nrow(iris)) # Generate 150 random numbers uniformaly distributed
> iris = iris[order(gp),]
> head(iris,10)
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
103          7.1         3.0          5.9         2.1  virginica
20           5.1         3.8          1.5         0.3     setosa
63           6.0         2.2          4.0         1.0 versicolor
17           5.4         3.9          1.3         0.4     setosa
83           5.8         2.7          3.9         1.2 versicolor
53           6.9         3.1          4.9         1.5 versicolor
118          7.7         3.8          6.7         2.2  virginica
91           5.5         2.6          4.4         1.2 versicolor
80           5.7         2.6          3.5         1.0 versicolor
43           4.4         3.2          1.3         0.2     setosa
>
```

Normalize the dataset (from 0 – 1)

```
> ######################################
> # 2. Normalize all numbers from 0 - 1
> #
> summary(iris[,c(1,2,3,4)])
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width   
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100  
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300  
 Median :5.800   Median :3.000   Median :4.350   Median :1.300  
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199  
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800  
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500  
> 
> normalize = function(x) {
+   return(  (x-min(x))/(max(x)-min(x)))}
> 
> iris_n = as.data.frame(lapply(iris[,c(1,2,3,4)],normalize))
> summary(iris_n)
  Sepal.Length     Sepal.Width      Petal.Length     Petal.Width     
 Min.   :0.0000   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000  
 1st Qu.:0.2222   1st Qu.:0.3333   1st Qu.:0.1017   1st Qu.:0.08333  
 Median :0.4167   Median :0.4167   Median :0.5678   Median :0.50000  
 Mean   :0.4287   Mean   :0.4406   Mean   :0.4675   Mean   :0.45806  
 3rd Qu.:0.5833   3rd Qu.:0.5417   3rd Qu.:0.6949   3rd Qu.:0.70833  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000  
> 
```

Build Training and Test Dataset

```
> #######################################
> # 3. Build Training and Test dataset
> #
> iris_train = iris_n[1:129,]
> iris_test = iris_n[130:150,]
> iris_training_target = iris[1:129,5]
> iris_test_target = iris[130:150,5]
>
```

Train the Model using Training Dataset, Predict Response Variable of the Test Dataset Package "class" has kNN module

```
> ########################################
> # call the kNN function
> #
> # The value of 'k' should be sqrt of observation
> # The value of 'k' should be an odd number
> # If a voting tie occurs, it can be resolved
> #
> # Observations = 150
> # k = 13
> #
> library(class)
>
> k = 13
> m1 <- knn(train=iris_train, test=iris_test, cl=iris_training_target, k=k)
> (t = table(iris_test_target,m1))
                m1
iris_test_target setosa versicolor virginica
      setosa          7          0         0
      versicolor      0          3         2
      virginica       0          0         9
>
> ##################################################
> # Compute Accuracy of Prediction
> (accuracy = sum(diag(t))/sum(t)*100)
[1] 90.47619
>
```

Example 4

Dataset = Food

Method: Build the kNN algorithm using Python

Read Data Set

```
import numpy as np
import pandas as pd
from collections import Counter
################################################
# Read Food Dataset
train = pd.read_csv('03 food1-15.csv')
test = pd.read_csv('05 food16.csv')
train
Out[9]:
     #        Item  Sweetness  Crunchiness   FoodType
0    1       Apple         10            9      Fruit
1    2       Bacon          1            4    Protein
2    3      Banana         10            1      Fruit
3    4      Carrot          7           10  Vegetable
4    5      Celery          3           10  Vegetable
5    6      Cheese          1            1    Protein
6    7       Grape          8            5      Fruit
7    8  Green bean          3            7  Vegetable
8    9        Nuts          3            6    Protein
9   10      Orange          7            3      Fruit
10  11     Lettuce          1            9  Vegetable
11  12    Cucumber          3            8  Vegetable
12  13      Shrimp          2            2    Protein
13  14        Fish          2            1    Protein
14  15        Pear         10            8      Fruit
test
Out[10]:
   #    Item  Sweetness  Crunchiness  FoodType
0  1  Tomato          6            4       NaN
```

Compute the Distance

```
################################################
# Compute the distance
# from Test object to all the Train's objects
#
trainC = train.shape[0]
print(trainC)
15
sum = np.zeros(trainC)

for i in range (0, trainC):
    sum[i] = sum[i] + (train.Sweetness[i] - test.Sweetness[0])**2
    sum[i] = sum[i] + (train.Crunchiness[i] - test.Crunchiness[0])**2



distance = np.sqrt(sum)

print(sum)
[ 41.  25.  25.  37.  45.  34.   5.  18.  13.   2.  50.  25.  20.  25.  32.]

print(distance)
[ 6.40312424  5.          5.          6.08276253  6.70820393  5.83095189
  2.23606798  4.24264069  3.60555128  1.41421356  7.07106781  5.
  4.47213595  5.          5.65685425]
```

Compute the Distance

```
train['dist'] = distance

print(train)
     #       Item  Sweetness  Crunchiness   FoodType      dist
0    1      Apple         10            9      Fruit  6.403124
1    2      Bacon          1            4    Protein  5.000000
2    3     Banana         10            1      Fruit  5.000000
3    4     Carrot          7           10  Vegetable  6.082763
4    5     Celery          3           10  Vegetable  6.708204
5    6     Cheese          1            1    Protein  5.830952
6    7      Grape          8            5      Fruit  2.236068
7    8 Green bean          3            7  Vegetable  4.242641
8    9       Nuts          3            6    Protein  3.605551
9   10     Orange          7            3      Fruit  1.414214
10  11    Lettuce          1            9  Vegetable  7.071068
11  12   Cucumber          3            8  Vegetable  5.000000
12  13     Shrimp          2            2    Protein  4.472136
13  14       Fish          2            1    Protein  5.000000
14  15       Pear         10            8      Fruit  5.656854
```

Sort the Distances

```
##########################################
# Sort the dataset by distance
#
trainSorted = train.sort_values(['dist'])

print(trainSorted)
     #       Item  Sweetness  Crunchiness   FoodType      dist
9   10     Orange          7            3      Fruit  1.414214
6    7      Grape          8            5      Fruit  2.236068
8    9       Nuts          3            6    Protein  3.605551
7    8 Green bean          3            7  Vegetable  4.242641
12  13     Shrimp          2            2    Protein  4.472136
1    2      Bacon          1            4    Protein  5.000000
2    3     Banana         10            1      Fruit  5.000000
11  12   Cucumber          3            8  Vegetable  5.000000
13  14       Fish          2            1    Protein  5.000000
14  15       Pear         10            8      Fruit  5.656854
5    6     Cheese          1            1    Protein  5.830952
3    4     Carrot          7           10  Vegetable  6.082763
0    1      Apple         10            9      Fruit  6.403124
4    5     Celery          3           10  Vegetable  6.708204
10  11    Lettuce          1            9  Vegetable  7.071068
```

Find the Nearest Neighbors

- k=1, Fruit (Fruit = 1)
- k=3, Fruit (Fruit = 2, Protein = 1)

```
#############################################
# Find the nearest neighbor
#
k = 1
nearestNeighbor = trainSorted.FoodType[0:k]
print(nearestNeighbor)
9    Fruit
Name: FoodType, dtype: object

Counter(nearestNeighbor)
Out[36]: Counter({'Fruit': 1})

k = 3
nearestNeighbor = trainSorted.FoodType[0:k]
print(nearestNeighbor)
9      Fruit
6      Fruit
8    Protein
Name: FoodType, dtype: object

Counter(nearestNeighbor)
Out[40]: Counter({'Fruit': 2, 'Protein': 1})
```

Example 5

Dataset = Food

Package: Python/Scikit-Learn

Example 4
Read Food Dataset

```
import numpy as np
import pandas as pd
from sklearn import neighbors

df = pd.read_csv('04 food16.csv')
df
Out[5]:
      #       Item  Sweetness  Crunchiness  Food Type
0     1      Apple         10            9      Fruit
1     2      Bacon          1            4    Protein
2     3     Banana         10            1      Fruit
3     4     Carrot          7           10  Vegetable
4     5     Celery          3           10  Vegetable
5     6     Cheese          1            1    Protein
6     7      Grape          8            5      Fruit
7     8  Green bean         3            7  Vegetable
8     9       Nuts          3            6    Protein
9    10     Orange          7            3      Fruit
10   11    Lettuce          1            9  Vegetable
11   12   Cucumber          3            8  Vegetable
12   13     Shrimp          2            2    Protein
13   14       Fish          2            1    Protein
14   15       Pear         10            8      Fruit
15   16     Tomato          6            4        NaN
```

Example 4
Split data into Train + Test

```
########################################
# Split data into train + test
#
X = np.array(df[['Sweetness','Crunchiness']])
y = df['Food Type']

X_train = X[0:15,]
X_train
Out[15]:
array([[10,  9],
       [ 1,  4],
       [10,  1],
       [ 7, 10],
       [ 3, 10],
       [ 1,  1],
       [ 8,  5],
       [ 3,  7],
       [ 3,  6],
       [ 7,  3],
       [ 1,  9],
       [ 3,  8],
       [ 2,  2],
       [ 2,  1],
       [10,  8]], dtype=int64)
```

```
y_train = y[0:15]
y_train
Out[17]:
0         Fruit
1       Protein
2         Fruit
3     Vegetable
4     Vegetable
5       Protein
6         Fruit
7     Vegetable
8       Protein
9         Fruit
10    Vegetable
11    Vegetable
12      Protein
13      Protein
14        Fruit
Name: Food Type, dtype: object

######################################
X_test = X[15:16,]
X_test
Out[20]: array([[6, 4]], dtype=int64)
```

Use 'Scikit-Learn' Package: kNeighborsClassifier Function

- k=1, Fruit (Fruit = 1)
- k=3, Fruit (Fruit = 2, Protein = 1)

```
clf = neighbors.KNeighborsClassifier(n_neighbors=1)
clf.fit(X_train, y_train)
Out[22]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')

clf.predict(X_test)
Out[23]: array(['Fruit'], dtype=object)

######################################################
clf = neighbors.KNeighborsClassifier(n_neighbors=3)
clf.fit(X_train, y_train)
Out[26]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=3, p=2,
           weights='uniform')

clf.predict(X_test)
Out[27]: array(['Fruit'], dtype=object)
```

Use 'Scikit-Learn' Package: kNeighborsClassifier Function

- k=5, Fruit or Protein (Fruit = 2, Protein = 2, Vegetable = 1)
- k=9, Protein (Fruit = 3, Protein = 4, Vegetable = 2)

```
clf = neighbors.KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
Out[29]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')

clf.predict(X_test)
Out[30]: array(['Fruit'], dtype=object)

####################################################
clf = neighbors.KNeighborsClassifier(n_neighbors=9)
clf.fit(X_train, y_train)
Out[33]:
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=9, p=2,
           weights='uniform')

clf.predict(X_test)
Out[34]: array(['Protein'], dtype=object)
```

Summary

- Similarity Based Learning
- The Bayes Classifier
- ‘k’ Nearest Neighbor (kNN) Model
- kNN Model Assessment
- Data Normalization:
  Standardization & Scaling
- kNN in R
  - Example 1: Dataset = Food, No Package
  - Example 2: Dataset = Food, Package = Class
  - Example 3: Dataset = Iris, Package = Class
- kNN in Python
  - Example 4: Dataset = Food, No Package
  - Example 5: Dataset = Food, Package = Scikit-Learn