Nearest Neighbor Classifier

Ishwar K Sethi

Now, we begin to look at some popular classification models. These models have different characteristics and generally offer tradeoffs in terms of accuracy, ease of design, and interpretability. In the end, every data miner begins to use one or two of these models as his/her favorite model.

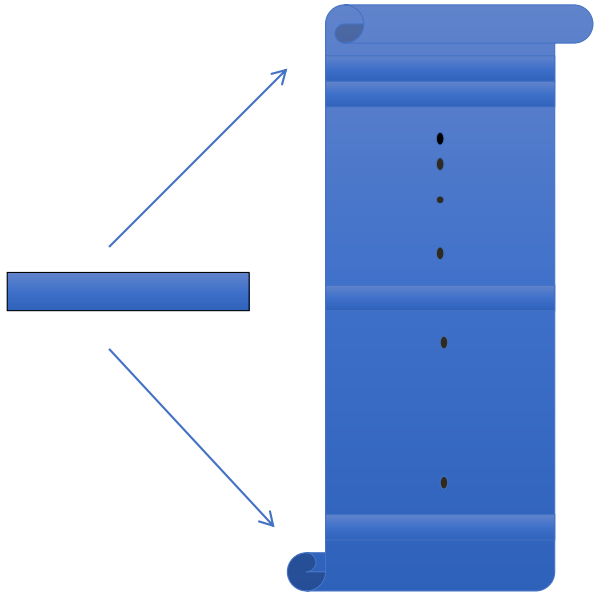

Our first Classifier, NN Classifier, is loved by lazy data miners.
Why?
Because its design doesn't require any effort.

Nearest Neighbor Classifier

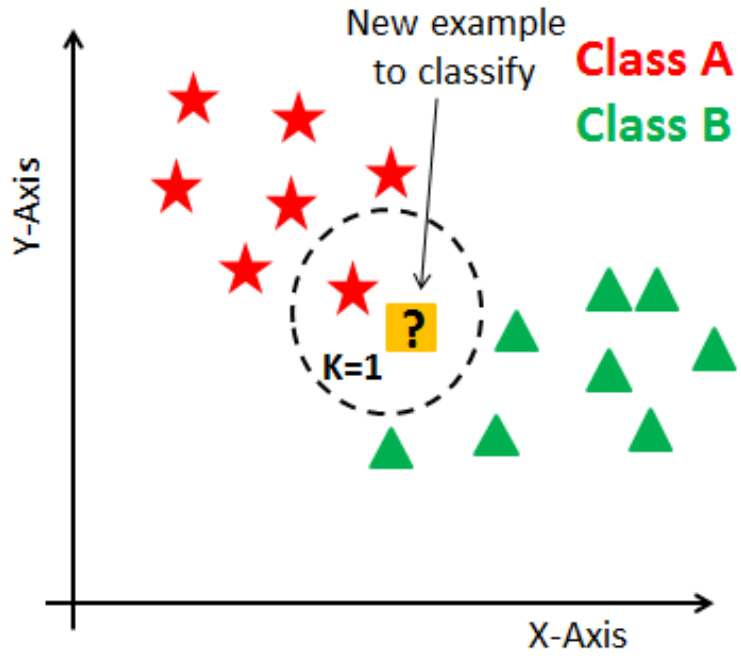
- Also known as
  - Memory-Based Reasoning (MBR) method
  - Instance-Based Classification
- Nonparametric method
  - No assumption about data distribution is made
- Many variations:
  - k-NN method
  - Weighted k-NN method
  - Edited NN method
  - Locally adaptive NN method

Basics of NN Approach

- Store your training examples with labels in a database
- Find the best match (nearest neighbor) for a given test example by searching through training examples
- The class label given to test case is that of the best matching example
  - If it walks like a duck, quacks like a duck, then it's probably a duck

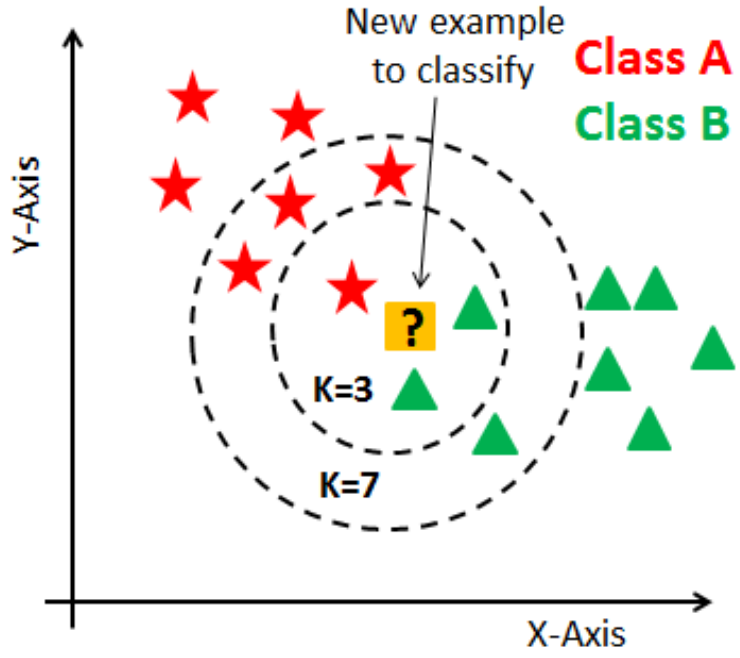


NN Illustration



NN classifier is also known as k-NN classifier where k determines the number of neighbors voting to classify a new/test example. In this example, k = 1.

k-NN Illustration



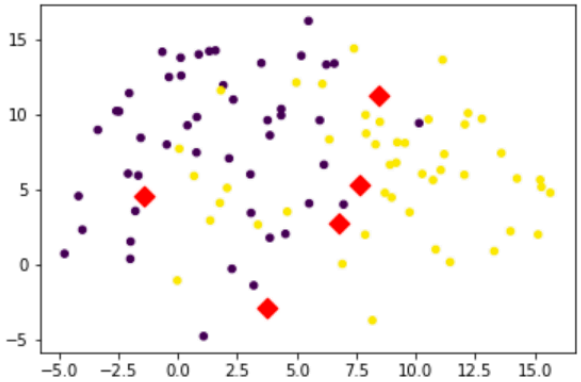
With k = 3, the new example is classified as of class B while k = 7 leads to classify it as of class A.

```
## K Nearest Neighbor Illustration
```

```
import matplotlib.pyplot as plt
import numpy as np
```

```
from sklearn.datasets import make_blobs
X, y = make_blobs(n_samples = 100, centers=2, cluster_std = 4.0, random_state=20)# Generates data with labels
```

```
from sklearn.neighbors import KNeighborsClassifier
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.05, random_state=42)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20)
plt.scatter(X_test[:, 0], X_test[:, 1], c="red",marker = "D", s=80)
plt.show()
```



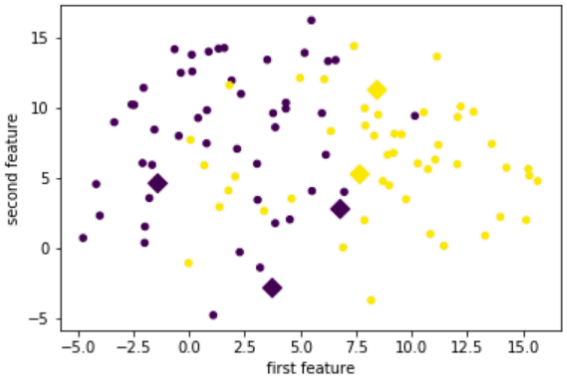
In this example, we generate 100 training examples in 2 dimensions with 2 class labels and then try to classify some test examples (red diamonds)

```
k_value = 1
knn = KNeighborsClassifier(k_value)
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=1, p=2,
           weights='uniform')
```

```
result = knn.predict(X_test)
```

```
marker = np.append(y_train,result, axis=0)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20)
plt.scatter(X_test[:, 0], X_test[:, 1], c=result,marker = "D", s=80)
plt.xlabel("first feature")
plt.ylabel("second feature")
plt.show()
```




```
print("accuracy: {:.2f}".format(knn.score(X_test, y_test)))
y_pred = knn.predict(X_test)
```

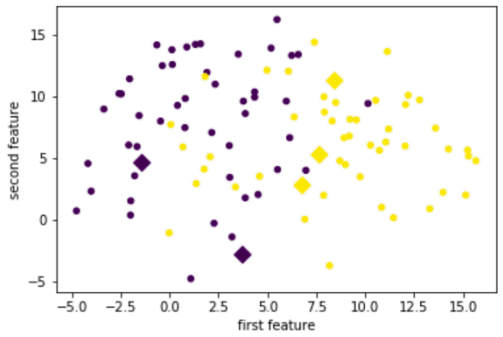
```
accuracy: 0.80
```

```
k_value = 7
knn = KNeighborsClassifier(k_value)
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=7, p=2,
           weights='uniform')
```

```
result = knn.predict(X_test)
```

```
marker = np.append(y_train,result, axis=0)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20)
plt.scatter(X_test[:, 0], X_test[:, 1], c=result,marker = "D", s=80)
plt.xlabel("first feature")
plt.ylabel("second feature")
plt.show()
```

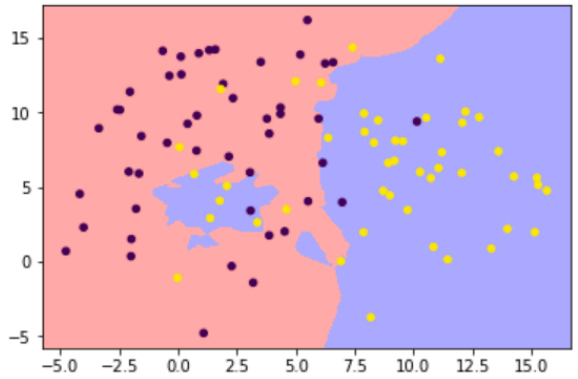



```
print("accuracy: {:.2f}".format(knn.score(X_test, y_test)))
y_pred = knn.predict(X_test)
```

```
accuracy: 1.00
```

```
# Plot the decision boundary
from matplotlib.colors import ListedColormap
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
# Plot the decision boundary. For that, we will assign a color to each
    # point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.02),
np.arange(y_min, y_max, 0.02))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20)
plt.show()
```

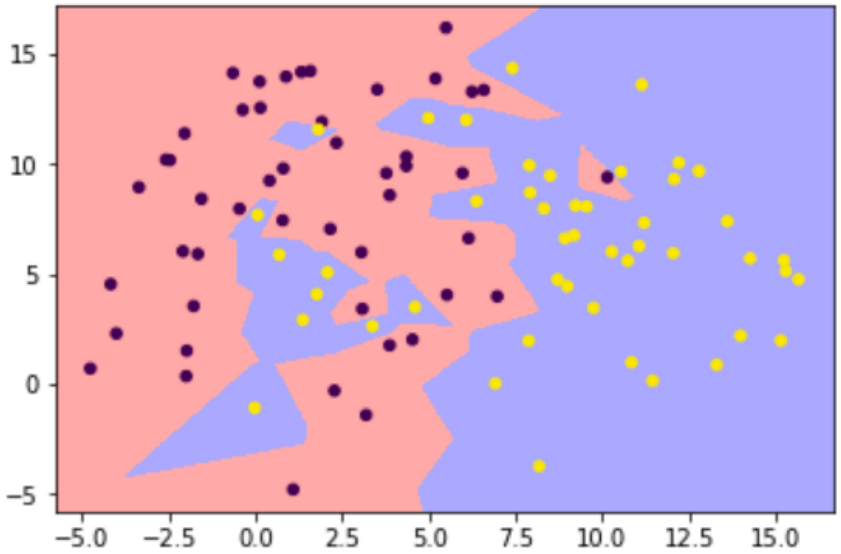


K = 7

This figure shows how the entire two-dimensional space has been partitioned into regions of two classes.

Boundary with k = 1 on next slide

Classification Boundary with k = 1



The strength of this approach is that it produces complex classification boundaries/models

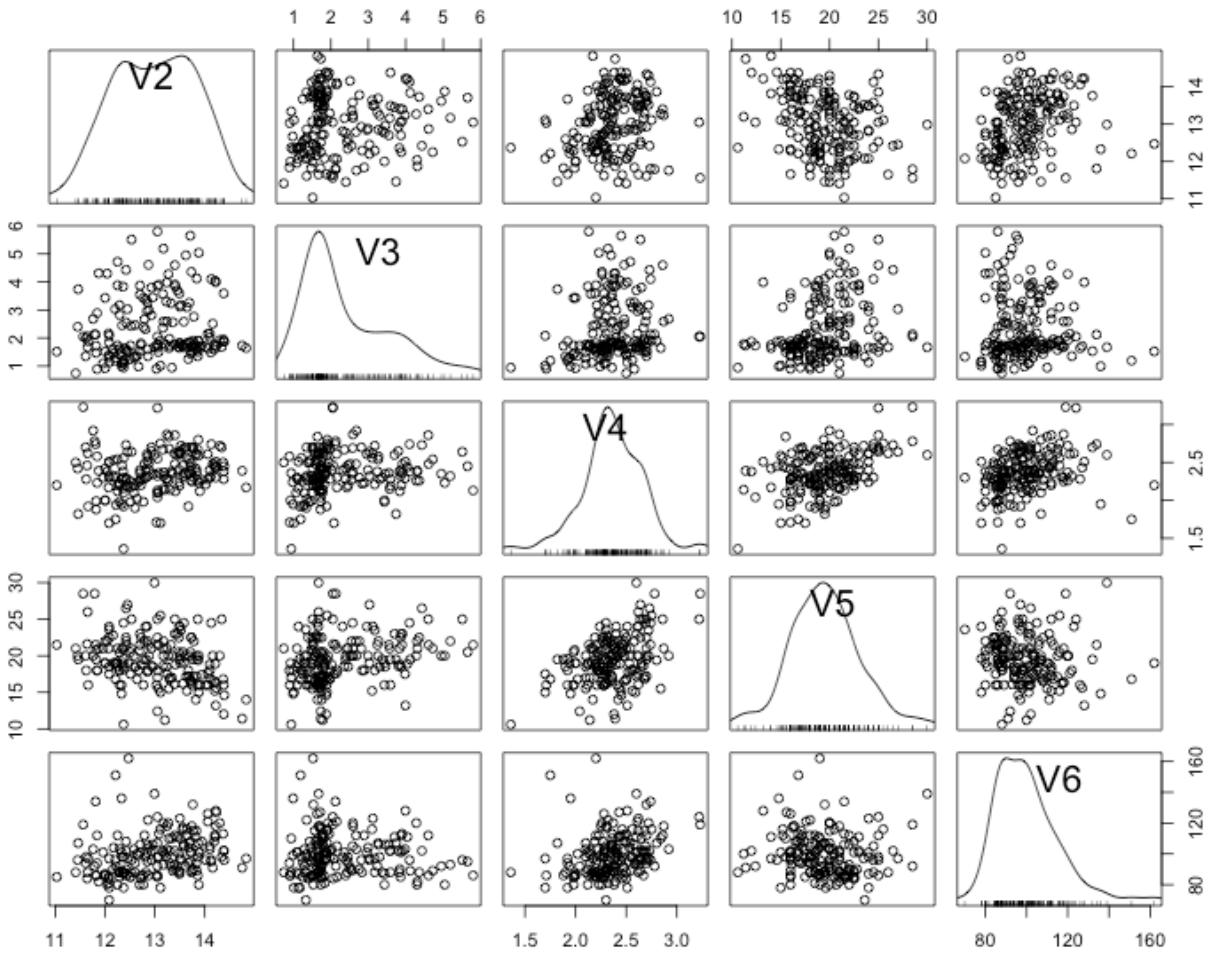
Wine Data Example for k-NN

3 Classes of Wine, 13 features (chemical composition), 178 examples. This demo is using R.

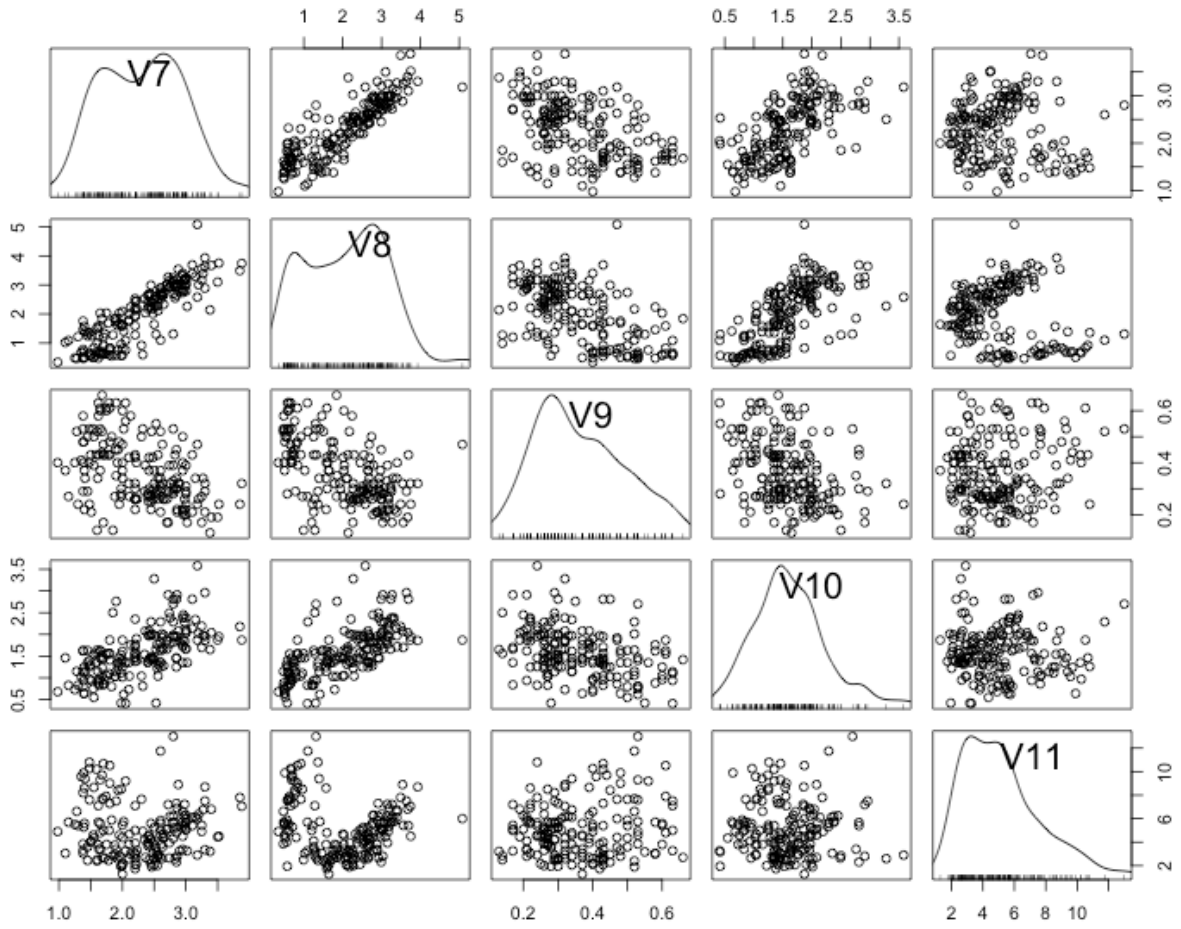
```
# KNN Demo
# Using Wine data; 3 classes, 13 features, each feature representing concentration of
different chemicals, 178 samples.
wine <- read.table("http://archive.ics.uci.edu/ml/machine-learning-
databases/wine/wine.data",sep=",")
# We are using sep="," to tell R that columns are separated by commas
```

Getting Scatter Plots

```
#Get scatter plots
#One common way of plotting multivariate data is to make a “matrix scatterplot”,
#showing each pair of variables plotted against
#each other. We can use the “scatterplotMatrix()” function from the “car” R package to
#do this.
#To use this function, we first need to install the “car” R package
Install.packages(“car”)
library("car")
scatterplotMatrix(wine[2:6],smoother="FALSE",reg.line="FALSE")
# This will plot pairs of features 2 to 6.
```



Scatter plot of wine data, features 2 - 6




```
> scatterplotMatrix(wine[7:11],smoother="FALSE",reg.line="FALSE")
```


```
> plot(wine$V4, wine$V5)
> text(wine$V4, wine$V5, wine$V1, cex=0.7, pos=4, col="red")
```

Shows data distribution with class labels using features 4 and 5. Shows data from classes 2 and 3 are more mixed up.

Overall feature means & sd

```
> sapply(wine[2:14],mean)
         V2          V3          V4          V5          V6          V7          V8
V9         V10         V11         V12
 13.0006180   2.3363483   2.3665169  19.4949438  99.7415730   2.2951124   2.0292697
0.3618539   1.5908989   5.0580899   0.9574494
         V13         V14
  2.6116854 746.8932584
> # mean values are calculated
> sapply(wine[2:14],sd)
         V2          V3          V4          V5          V6          V7          V8
V9         V10         V11         V12
  0.8118265   1.1171461   0.2743440   3.3395638  14.2824835   0.6258510   0.9988587
0.1244533   0.5723589   2.3182859   0.2285716
         V13         V14
  0.7099904 314.9074743
```

```
> # Mean and variance calculation for each class
> cultivar2wine <- wine[wine$V1=="2",]
> # This extracts all class 2 samples
> sapply(cultivar2wine[2:14],mean)
       V2         V3         V4         V5         V6         V7         V8         V9
V10        V11        V12        V13
 12.278732   1.932676   2.244789  20.238028  94.549296   2.258873   2.080845   0.363662
1.630282   3.086620   1.056282   2.785352
       V14
519.507042
> # Similarly, we can extract class 1 and 3 means and variances.
> sapply(cultivar2wine[2:14],sd)
         V2          V3          V4          V5          V6          V7          V8
V9         V10         V11         V12
  0.5379642   1.0155687   0.3154673   3.3497704  16.7534975   0.5453611   0.7057008
0.1239613   0.6020678   0.9249293   0.2029368
        V13         V14
  0.4965735 157.2112204
```

```
> wine1 <- wine[wine$V1=="1",]
> wine2 <- wine[wine$V1==”2",]
> wine3 <- wine[wine$V1=="3",]
> train <- rbind(wine1[1:50,2:14,1], wine2[1:60,2:14,2], wine3[1:40,2:14,3])
> test <- rbind(wine1[51:59,2:14,1], wine2[61:71,2:14,2], wine3[41:48,2:14,3])
> c1 <- factor(c(rep(1,50), rep(2,60), rep(3,40))
+ )
```

Separate out data from three classes. Create training and test set for using KNN classifier

```
> library(class)
> knn(train, test, c1, k = 3, prob=FALSE)
 [1] 1 1 1 1 1 1 1 1 1 2 3 2 2 2 2 2 2 2 2 2 3 2 3 3 1 1 1 3
Levels: 1 2 3
> knn(train, test, c1, k = 5, prob=FALSE)
 [1] 1 1 1 1 1 1 1 1 1 3 3 2 2 2 2 2 2 2 2 3 3 2 3 3 1 1 1 2
Levels: 1 2 3
> knn(train, test, c1, k = 1, prob=FALSE)
 [1] 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 2 2 2 2 3 3 2 2 3 3 1 1 3
Levels: 1 2 3
```

Misclassified examples are marked in red

```
> # Leave one out crossvalidation
> train <- rbind(wine1[,2:14,1], wine2[,2:14,2], wine3[,2:14,3])
> cl <-factor(c(rep(1,59), rep(2, 71), rep(3, 48)))
> knn.cv(train, cl,k = 3, prob=FALSE)
```

Leave one out method of crossvalidation rotates through the training data by using every example as a test example once.

```
 [1] 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 3 1 1 3 3 1 1 1 1 1 1 1 1 1 2 1 1 2 1 1 1 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 2 3 2 2 2
 [67] 2 2 3 2 1 2 3 1 1 2 2 3 1 2 2 3 2 3 3 2 2 2 2 3 2 2 2 2 2 1 3 2 3 2 3 2 2 2 2 2 2 2 2
2 3 2 3 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 3 3
[133] 2 3 3 2 2 3 2 3 3 1 3 3 2 3 2 3 3 3 3 2 2 2 3 3 2 2 3 3 2 1 3 1 3 3 3 2 3 3 3 2 3 3 3
1 1 3
```

Confusion Matrix

	Wine 1	Wine 2	Wine 3
Wine 1	51	3	5
Wine 2	5	50	16
Wine 3	5	14	29

KNN with Normalized Features

```
> standardisedconcentrations <- as.data.frame(scale(wine[2:14]))
> swine<- standardisedconcentrations
> dim(swine)
[1] 178  13
> train <- rbind(swine[1:59,,1], swine[60:130,,2],swine[131:178,,3])
> knn.cv(train, cl,k = 3, prob=FALSE)

 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 3 2 2 2 1
  [67] 2 2 2 2 2 1 2 1 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 3 2 2 1 2 2 2 2 2 2 2 3 3
 [133] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3
```

	Wine 1	Wine 2	Wine 3
Wine 1	59	0	0
Wine 2	2	65	4
Wine 3	0	0	48

Normalization offers better results because features with large values are not allowed to dominate the distance calculations.

K-NN: Strengths and Weaknesses

- No training or model building effort is required
- No assumptions about how the data is distributed
- Choice of k determines accuracy
  - For a small training set, k should be taken as 1. For large training set, k is generally taken as an odd value such as 7 or 9 or 11 etc. Larger k values yield smother decision boundaries
- Can be computationally expensive with large number of attributes and records
  - Several efficient algorithms for fast k-NN search, for example k-d tree

Further Reading

- You can read more about k-NN examples at

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html