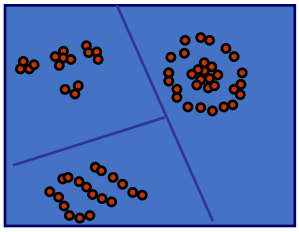
Building Classification Models



Classification Vs. Clustering

- Building a classifier requires marked or labeled training data. The number of classes, i.e. the number of distinct labels, is known beforehand. In a way, *the training data for classification comes with known answers*. Thus, the process is often termed supervised learning.
- In clustering, we seek a grouping in data without often knowing how many groups are there and how the examples in each group look like. Thus, it is often termed unsupervised learning.

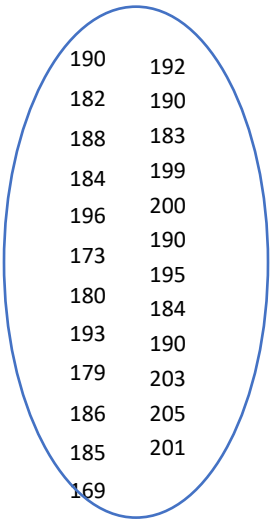
What is Classification?

190	F
182	F
188	F
184	F
196	F
173	F
180	F
193	F
179	F
186	F
185	F
169	F

192	M
190	M
183	M
199	M
200	M
190	M
195	M
184	M
190	M
203	M
205	M
201	M

Given the measurements for several male and female basketball players, can we predict the gender of the player whose height is 194 ?

What is Clustering?

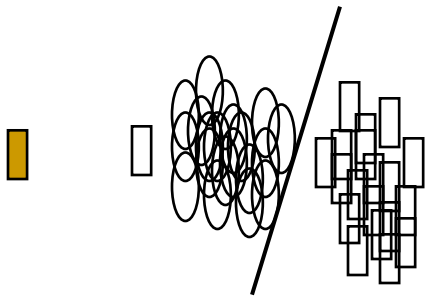


Given a collection of height measurements, form two groups

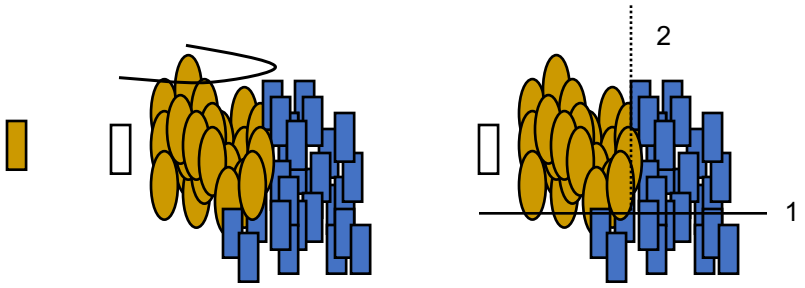
Classifier Types

- Linear vs. nonlinear
  - A linear classifier separates examples from different classes by using linear decision functions
  - A nonlinear classifier uses nonlinear functions and is thus more powerful in achieving separation
- One shot vs. multi-stage classification
  - How the classification decision is arrived at?

Linear & Nonlinear Classification



Single Stage Vs. Multi-Stage Classification



Hard Vs. Soft Classification

- Hard classification implies that the label(s) assigned to a document/object carry equal weight
- Soft classification implies some kind of ordering among the labels assigned to a document/object. Soft classifiers are often called fuzzy classifiers

Binary Vs. Multi-way Classification

- Binary classification implies only two classes or categories. For example, spam vs. non-spam email
- Multi-way classification implies more than two categories. For example, Reuters collection has 135 topic categories.
  - A M-way multi-class problem can be converted to M binary classification problems

Single Vs. Multi-label Classification

- A single label classifier assigns only one class/category label to an input pattern. The traditional classifiers are single label classifiers
- A multi-label classifier assigns more than one label to an input, for example a student classification system might label a student as *academic high achiever*, *athletic*, and *popular*. Many information retrieval applications need multiple labels to stored artifact

Generative Vs. Discriminative Vs. Geometric Models

- In generative models, we assume a functional form for conditional probabilities and use data to estimate these probabilities and apply Bayes rule to estimate posteriori probabilities
- In discriminative models, we try to estimate directly the posteriori probabilities or learn to map input directly to class labels
- In geometric models, we try to learn a geometric function that can separate examples from different classes. The method doesn't use any probabilistic concepts

Walk Through An Example: Flower Classification

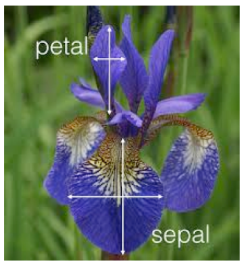
- Build a classification model (smartphone app) to differentiate between two classes of flower

Iris Virginica

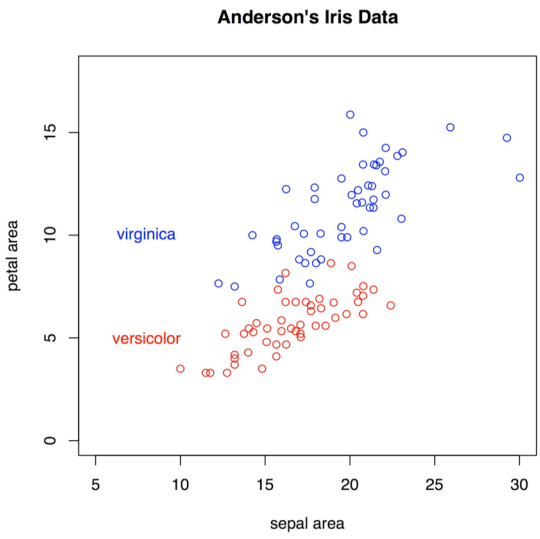
Iris Versicolor

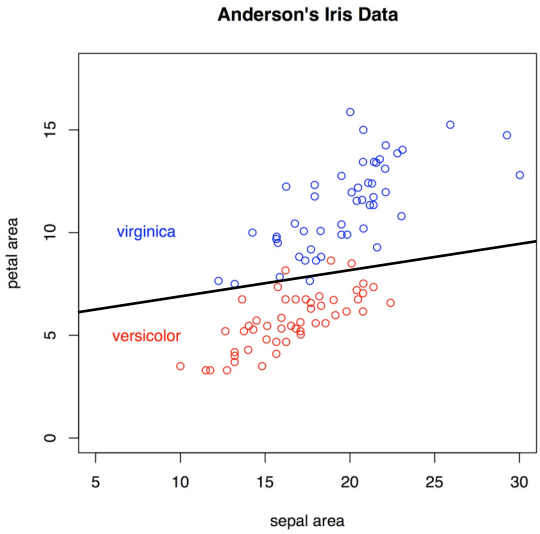
How Do We Go About It?

- Collect a large number of both types of flowers with the help of an expert
- Measure some attributes that can help differentiate between the two types of flowers. Let those attributes be petal area and sepal area.

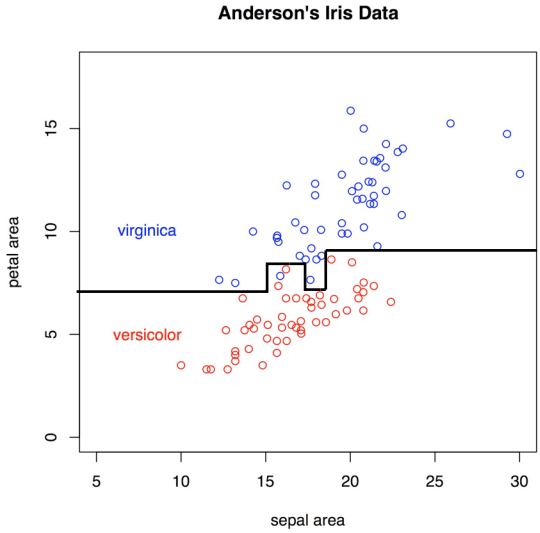


Scatter plot of 100 examples of flowers

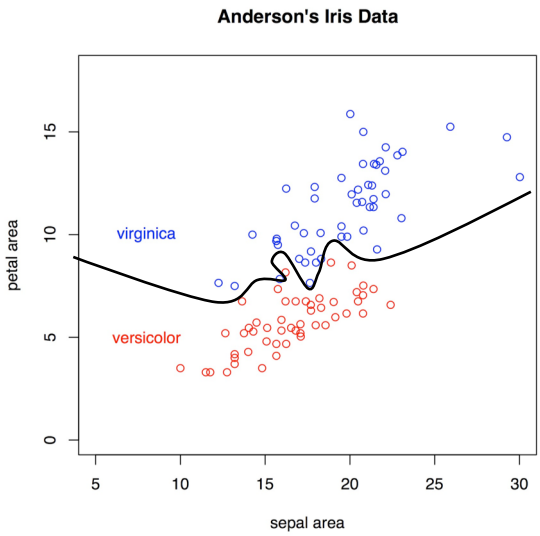




We can separate the flower types using the linear boundary shown above. The parameters of the line represent the learned classification model.



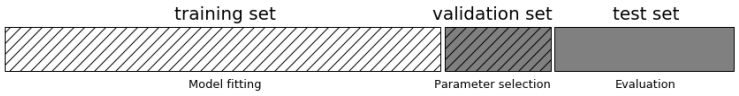
Another possible boundary. This boundary cannot be expressed via an equation. However, a tree structure can be used to express this boundary. Note, this boundary does better prediction of the collected data

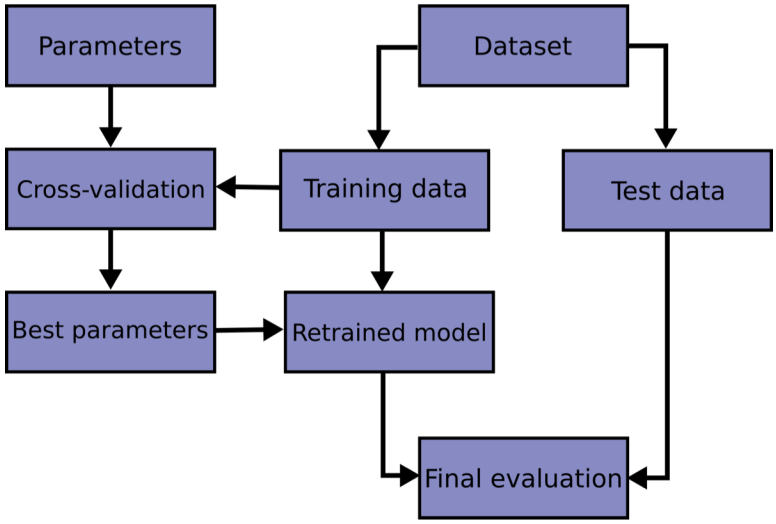


Yet another possible boundary. This boundary does prediction without any error. Is this a better boundary?

Model Complexity

- There are tradeoffs between the complexity of models and their performance in the field. A good design (model choice) weighs these tradeoffs.
- A good design should avoid overfitting. How?
  - Divide the entire data into three sets
    - Training set (about 70% of the total data). Use this set to build the model
    - Test set (about 20% of the total data). Use this set to estimate the model accuracy after deployment
    - Validation set (remaining 10% of the total data). Use this set to determine the appropriate settings for free parameters of the model. May not be required in some cases.





Measuring Classifier Performance

	Correct category is c_1	Correct category is c_2
Assigned category is c_1	a	b
Assigned category is c_2	c	d

- Accuracy = (a + d)/(a + b + c + d)
- Precision(c_1) = a/(a + b); Precision(c_2) = d/(c + d)
- Recall(c_1) = a/(a + c); Recall(c_2) = d/(b + d)

Measuring Performance

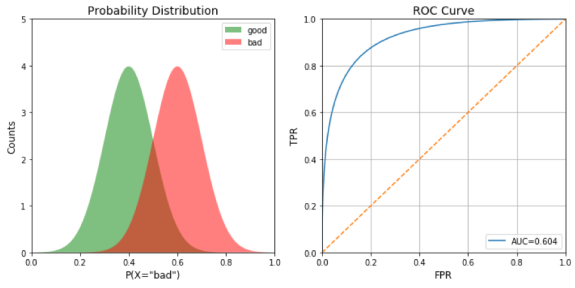
- True Positive: Correctly identified as relevant ("a" from prev. slide)
- True Negative: Correctly identified as not relevant ("d")
- False Positive: Incorrectly labeled as relevant ("b")
- False Negative: Incorrectly labeled as not relevant ("c')

Cat vs. No Cat

True Positive | True Negative | False Negative | False Positive

ROC (Receiver Operating Characteristics) Curve AUC (Area under the ROC Curve)

- True Positive Rate (TPR) = TP/(TP+FN) = a/(a+c)
- False Positive Rate (FPR) = FP/(FP+TN) = b/(b+d)
- An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.



Confusion Matrix for M Classes

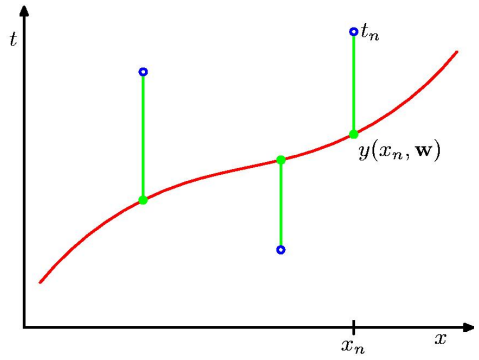
Predicted class

Actual class	Sit	Stand	Walk Jog	Ascend	Descend	Cycle	**Class-specific recall**
Sit	*3202*	*2*	*0*	*0*	*0*	*14*	**0.99**
Stand	*7*	*3191*	*2*	*7*	*0*	*0*	**0.99**
Walk	*0*	*0*	*10647*	*74*	*0*	*0*	**0.99**
Ascend	*0*	*0*	*34*	*500*	*15*	*1*	**0.90**
Descend	*0*	*0*	*41*	*60*	*405*	*0*	**0.80**
Cycle	*146*	*3*	*0*	*0*	*0*	*2539*	**0.94**
Class-specific precision	**0.95**	**1.00**	**0.99**	**0.78**	**0.96**	**0.99**	**0.98**

Accuracy?

- The previous definition of accuracy can be misleading when you have uneven representation from two classes. For example, consider a 2-class problem with 90 examples from class 1 and 10 from class 2.
  - Class predictive model 1 classifies 75 of class 1 examples and 3 of class 2 examples correctly
  - Another model classifies 60 of class 1 examples and 7 of class 2 examples correctly
  - Which model is a better predictor?
- What happens when different mistakes do not cost the same?

Sum-of-Squares Error for Regression Models

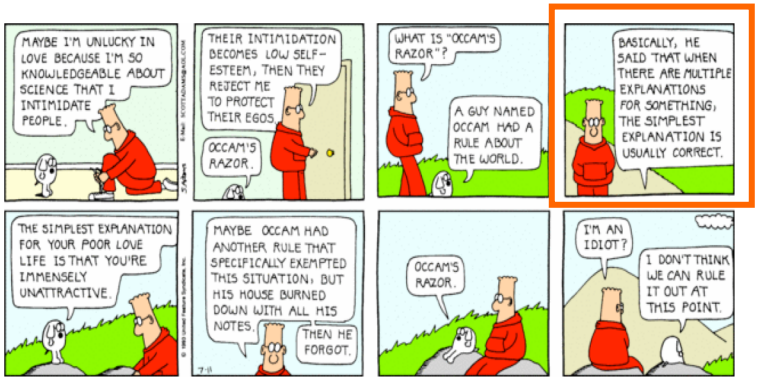


For regression model, the error is measured by taking the square of the difference between the predicted output value and the target value for each training (test) example and adding this number over all examples as shown

Bias and Variance

- Bias: expected difference between model's prediction and truth
- Variance: how much the model differs among training sets
- Model Scenarios
  - High Bias: Model makes inaccurate predictions on training data
  - High Variance: Model does not generalize to new datasets
  - Low Bias: Model makes accurate predictions on training data
  - Low Variance: Model generalizes to new datasets

The Guiding Principle for Model Selection: Occam's Razor

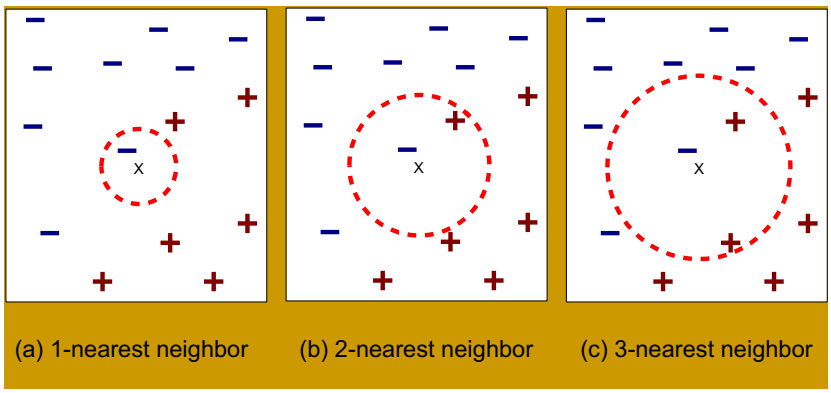


Our First Classifier

It's a lazy data miner's classifier

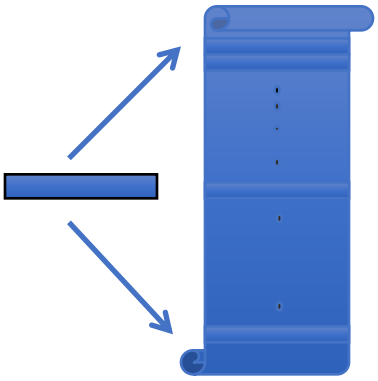
Nearest Neighbor Classifier

- 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 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

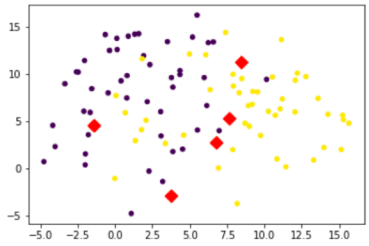


```
## K Nearest Neighbor Illustration
```

```
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()
```

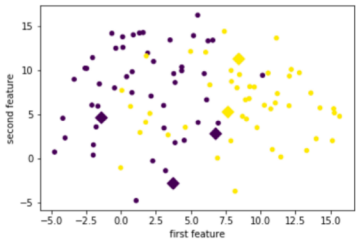


```
k_value = 1
knn = KNeighborsClassifier(k_value)
knn.fit(X_train,y_train)
```

```
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)
```

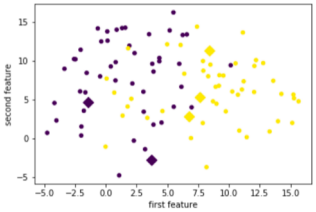
```
accuracy: 0.80
```

```
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)
```

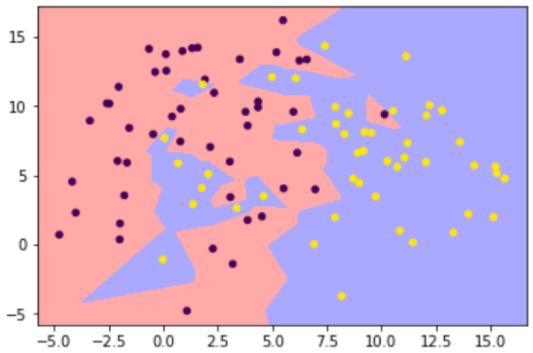
```
marker = np.append(y_train,result, axis=0)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, s=20)
plt.scatter(X_test[:, 0], X_test[:, 1], c=result,marker = "D", s=80)
plt.xlabel("first feature")
plt.ylabel("second feature")
plt.show()
```




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

```
accuracy: 1.00
```

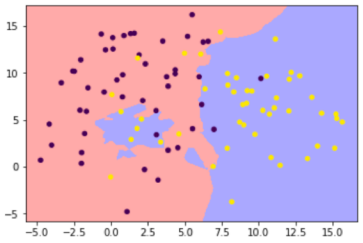
An Example of 1-NN Classification Model



The strength of this approach is that it can produce complex classification models


```
# 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

K-NN: Strengths and Weaknesses

- No training or model building effort is required
- No assumptions about how the data is distributed
- Can be computationally expensive with large number of attributes and records
  - Several efficient algorithms for fast k-NN search, for example k-d tree

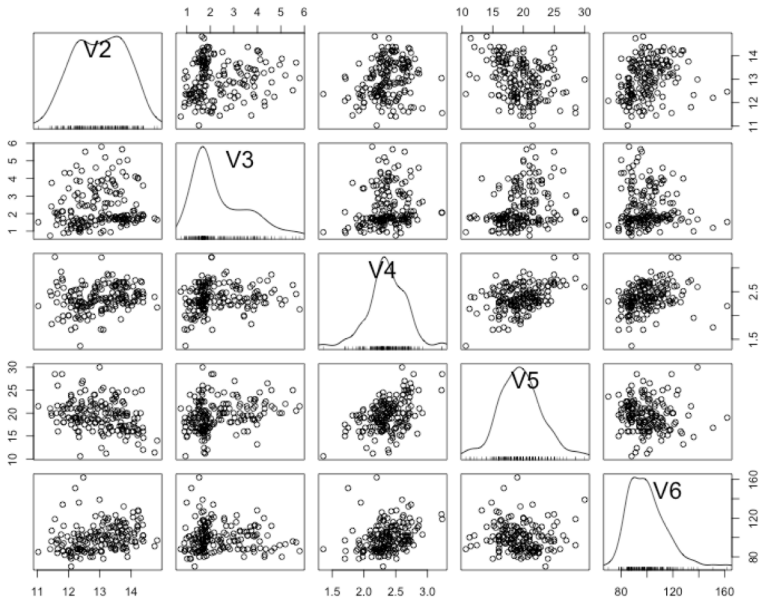
Wine Data Example for KNN

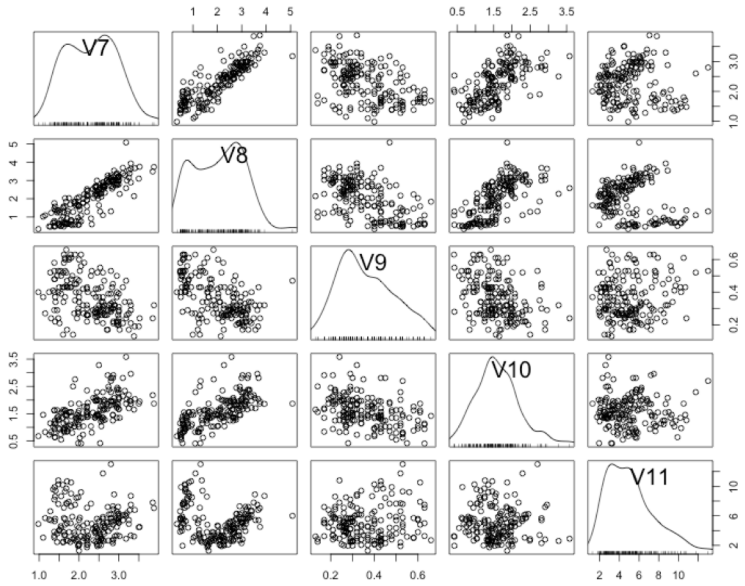
3 Classes of Wine, 13 features (chemical composition), 178 examples

```
# 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.
```






```
> 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")
```

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 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 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 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)
```

```
 [1] 1 1 1 1 3 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 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 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 2 3 2 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 2 2 3 2 2 1 2 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 3 3
```

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