Classification II

3. Bayes Classification Methods:

The **Bayes classifier** is a statistical method which based on the Bayes theorem.

- It assigns each observation to the most likely class, given its predictor values.
- It is know also as **Bayes Classifier**.

Bayes' Theorem:


Thomas Bayes
1702 - 1761

Bayes' theorem is named after Thomas Bayes (1701?–1761), who first used conditional probability and did early work in probability and decision theory during the 18th century.

The conditional probability is a measure of the probability of an event occurring given that another event has occurred.

If the event of interest is A and the event B is known or assumed to have occurred, 'the conditional probability of A given B', is usually written as P(A | B), where

$$P(A \mid B) = \frac{P(A \cap B)}{P(B)}$$

Example:

- The probability that any given person has a cough on any given day may be only 5%.
- If we know or assume that the person has a cold, then they are much more likely to be coughing.
- The conditional probability that someone coughing is unwell might be 75%, then:

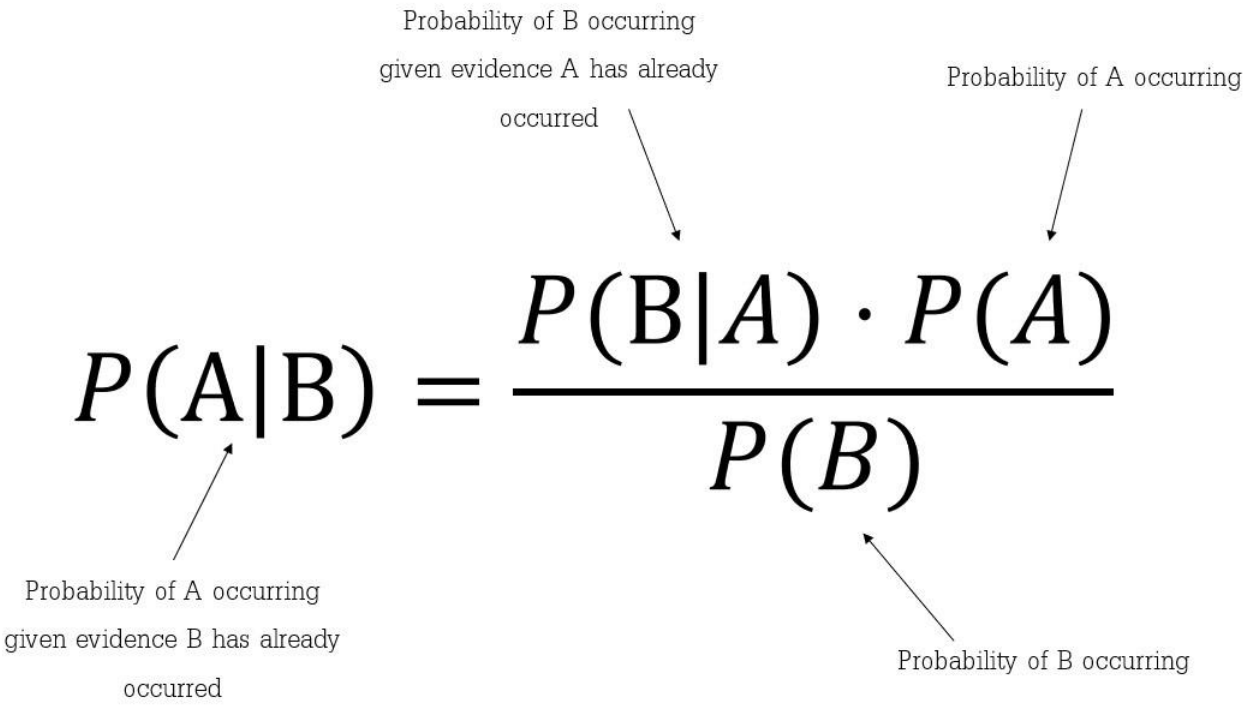
$$P(\text{Cough}) = 5\%$$

$$P(\text{Sick} \mid \text{Cough}) = 75\%$$

Bayes' Theorem:

Bayes' theorem is a way to figure out conditional probability.

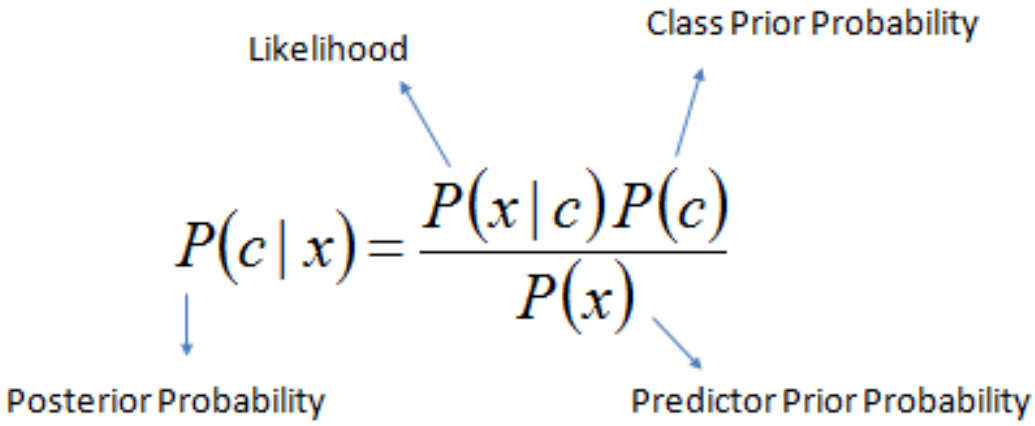
- Bayes' Theorem allows us to update predicted probabilities of an event by incorporating new information.



4. Naïve Bayes classifiers:

based on applying Bayes' theorem with strong (naïve) independence assumptions between the attributes.

- It is called Naive because the algorithm makes a very strong assumption about the data having attributes independent of each other while in reality, they may be dependent in some way.



$$P(c \mid X) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Example 1: (Titanic - Categorical)

Using *Titanic* (available in r). The format of the dataset differs from that we used last lecture. There are 32 observations which represent all possible combinations of Class, Sex, Age and Survived with their frequency (frequency table).

1. Use view() function to vies your data.

```
> View(Titanic)
```

	Class	Sex	Age	Survived	Freq
1	1st	Male	Child	No	0
2	2nd	Male	Child	No	0
3	3rd	Male	Child	No	35
4	Crew	Male	Child	No	0

- Freq = 0 (frequency) means that there is no value with the same combination (1st class, male, child, and dead).
- 3rd row shows that 35 passengers on titanic were 3rd class ticket holders, male, child, and dead.

Example 1: (Titanic)

2. We need to expand the table into individual rows. Let's create a repeating sequence of rows based on the frequencies in the dataset.

3. Show the dataset dimension and head.

```
> Titanic_df=as.data.frame(Titanic)
> repeating_sequence=rep.int(seq_len(nrow(Titanic_df)), Titanic_df$Freq)
> Titanic_dataset=Titanic_df[repeating_sequence,]
> Titanic_dataset$Freq=NULL
```

```
> dim(Titanic_dataset)
[1] 2201    4
> head(Titanic_dataset)
    Class  Sex   Age Survived
3     3rd Male Child       No
3.1   3rd Male Child       No
3.2   3rd Male Child       No
3.3   3rd Male Child       No
3.4   3rd Male Child       No
3.5   3rd Male Child       No
```

Example 1: (Titanic)

4. Install the e1071 package.

5. Check the description of naiveBayes() function.

Note: the naiveBayes() function has 3 different format.

```
install.packages("e1071")
library("e1071")
```

```
> ?naiveBayes
```

Example 1: (Titanic)

6. Use the Naive Bayes classifier to fit the model.

Note: Y denotes to the response attribute (survived).

The model creates the conditional probability for each feature separately, and the a-priori probabilities which indicates the distribution of our data.

```
> Naive_Model=naiveBayes(Survived ~., data=Titanic_dataset)
> Naive_Model

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
      No      Yes
0.676965 0.323035

Conditional probabilities:
     Class
Y           1st        2nd        3rd       Crew
  No  0.08187919 0.11208054 0.35436242 0.45167785
  Yes 0.28551336 0.16596343 0.25035162 0.29817159

     Sex
Y          Male     Female
  No  0.91543624 0.08456376
  Yes 0.51617440 0.48382560

     Age
Y          Child      Adult
  No  0.03489933 0.96510067
  Yes 0.08016878 0.91983122
```

Example 1: (Titanic)

7. Find the contingency and probability contingency table. Interpret the results.

```
> Pred = predict(Naive_Model, Titanic_dataset)
> table(Pred, Titanic_dataset$Survived)

Pred    No  Yes
  No  1364  362
  Yes  126  349
> prop.table = prop.table(table(Pred, Titanic_dataset$Survived))
> prop.table

Pred           No        Yes
  No  0.61971831 0.16447070
  Yes 0.05724671 0.15856429
```

- We are able to classify 1364 out of 1490 “No” cases correctly (61.97%) and 349 out of 711 “Yes” cases correctly (15.86).

Example 1: (Titanic)

8. Find the overall accuracy.

```
> mean(Pred == Titanic_dataset$Survived)
[1] 0.7782826
```

- Which is the percentage of the cases that classified correctly.

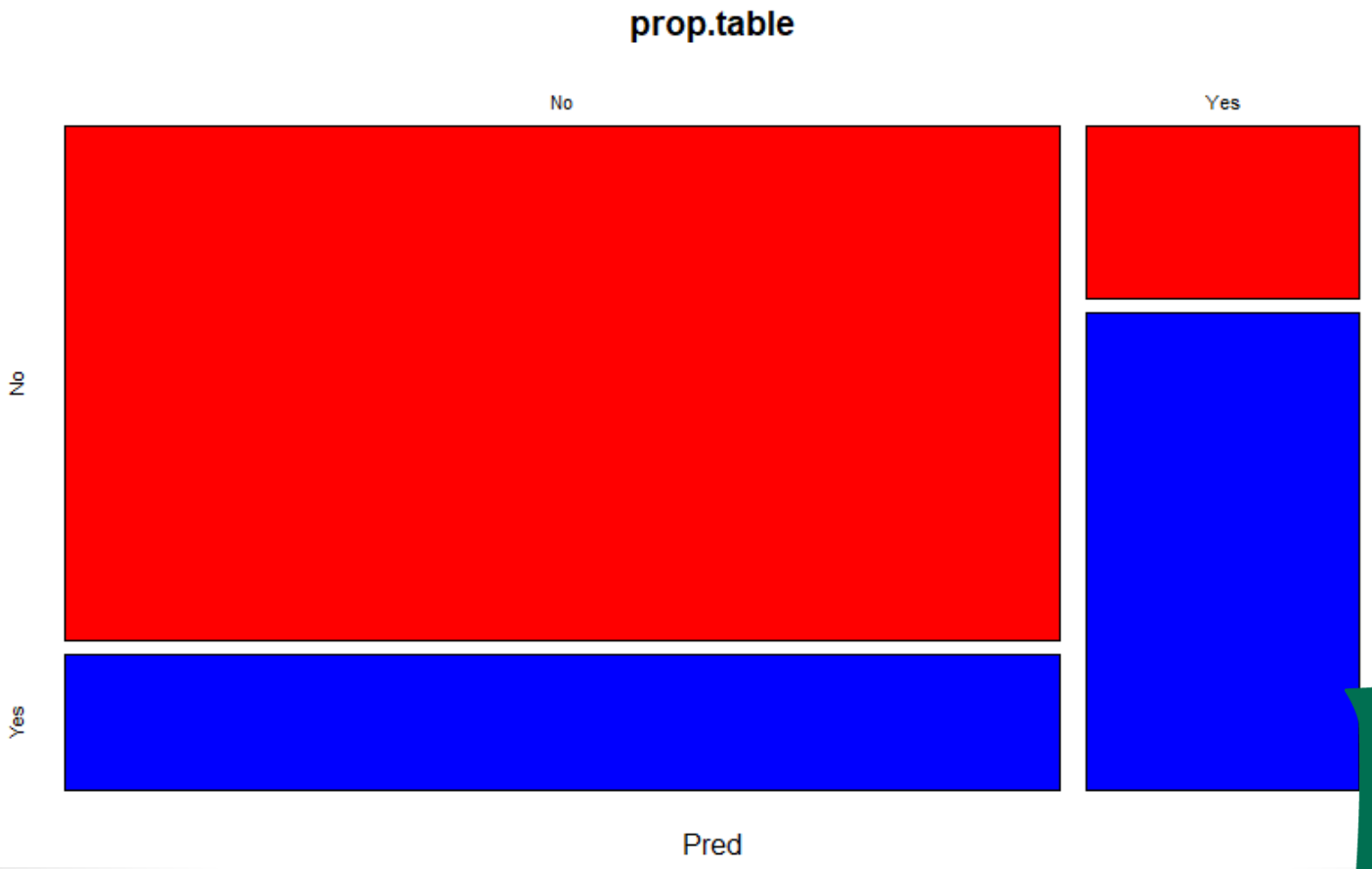
0.61971831 + 0.15856429 = 0.7782826

Pred	No	Yes
No	0.61971831	0.16447070
Yes	0.05724671	0.15856429

Example 1: (Titanic)

9. Present the probability contingency table using mosaic plot.

```
> plot(prop.table, col=c('red','blue'))
```



Example 1:

10. Split the data into two groups training data (80%) and test data (20%). Redo the procedure and compare the results.

```
> smp_size = floor(0.80 * nrow(Titanic_dataset))
> index = sample(seq_len(nrow(Titanic_dataset)), size = smp_size)
> train = Titanic_dataset[index, ]
> test = Titanic_dataset[-index, ]
> Naive_Model1=naiveBayes(Survived ~., data=train)
> Pred1 = predict(Naive_Model1, test)
> prop.table1 = prop.table(table(Pred1, test$Survived))
> prop.table1

Pred1          No        Yes
  No   0.60770975 0.19501134
  Yes  0.05442177 0.14285714
> mean(Pred1 == test$Survived)
[1] 0.7505669
```

Note: the training and test data are random samples, so if you redo the same process, you will get a different results sometimes.

Example 2: (Iris - Quantitative)

In the *iris* dataset in lab 4, there are 5 attributes, the first two are the Sepal's length and width, Petal's length and width, and the plant species.

1. Use the Naive Bayes classifier for the Sepal's length and width only. (80% / 20%)

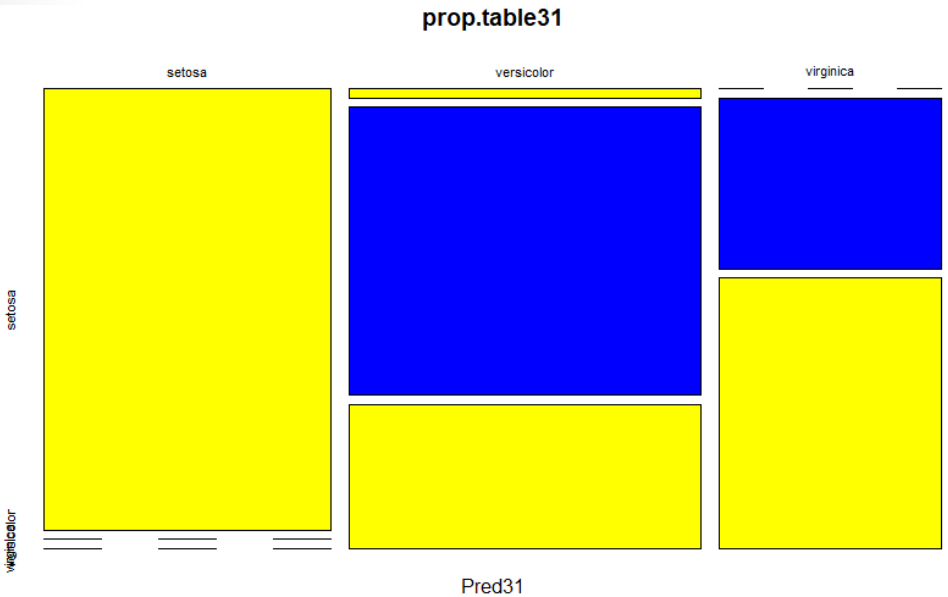
```
> smp_size = floor(0.80 * nrow(iris))
> index = sample(seq_len(nrow(iris)), size = smp_size)
> train1 = iris[index, ]
> test1 = iris[-index, ]
> Naive_Model3=naiveBayes(train1[,1:2], train1[,5])
> Pred3 = predict(Naive_Model3, test1[,-5])
> table(Pred3, test1[,5])

Pred3        setosa versicolor virginica
  setosa          9          0         0
  versicolor      0          5         4
  virginica       0          1        11
```

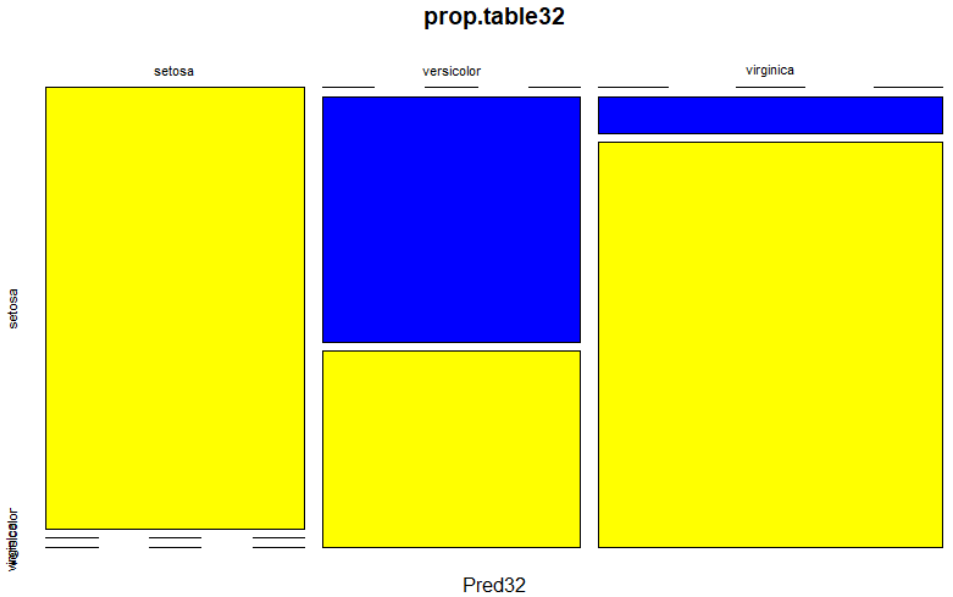
Example 2: (Iris - Quantitative)

2. Find the and probability contingency table of the training and test data. Compare the results.

```
> Pred31 = predict(Naive_Model3, train1[,-5])
> prop.table31 = prop.table(table(Pred31, train1[,5]))
> prop.table31

Pred31            setosa  versicolor   virginica
  setosa     0.333333333 0.000000000 0.000000000
  versicolor 0.008333333 0.266666667 0.133333333
  virginica  0.000000000 0.100000000 0.158333333
```

```
> Pred32 = predict(Naive_Model3, test1[,-5])
> prop.table32 = prop.table(table(Pred32, test1[,5]))
> prop.table32

Pred32            setosa versicolor  virginica
  setosa     0.30000000 0.00000000 0.00000000
  versicolor 0.00000000 0.16666667 0.13333333
  virginica  0.00000000 0.03333333 0.36666667
```





Advantages of the Naïve :

1. Easy to implement.
2. Requires less training data to estimate the test data.
3. Less sensitive to missing data.

Disadvantages of the Naïve:

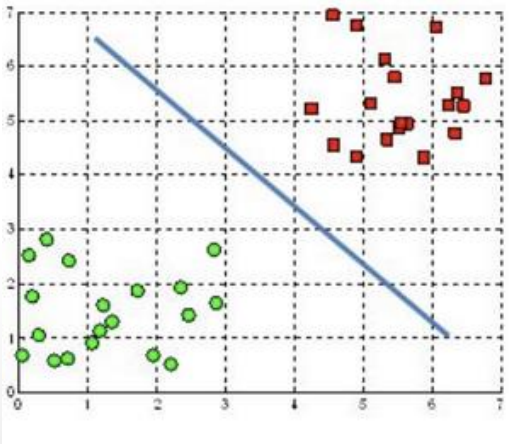
1. Makes a very strong assumption of independent attributes.
2. If categorical variable has a category in test data set, which was not observed in training data set, then model will assign a 0 (zero) probability and will be unable to make a prediction.
3. Very sensitive to the form of input data.

5. Support Vector Machine (SVM):

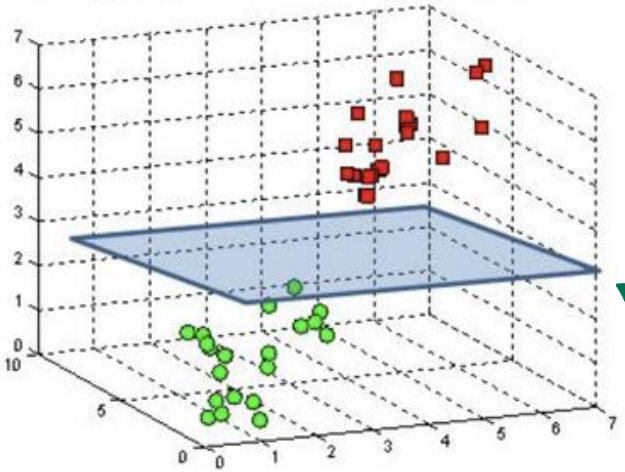
Support vector machines are a set of methods used for classification, regression and outliers detection. However, it is mostly used in classification problems.

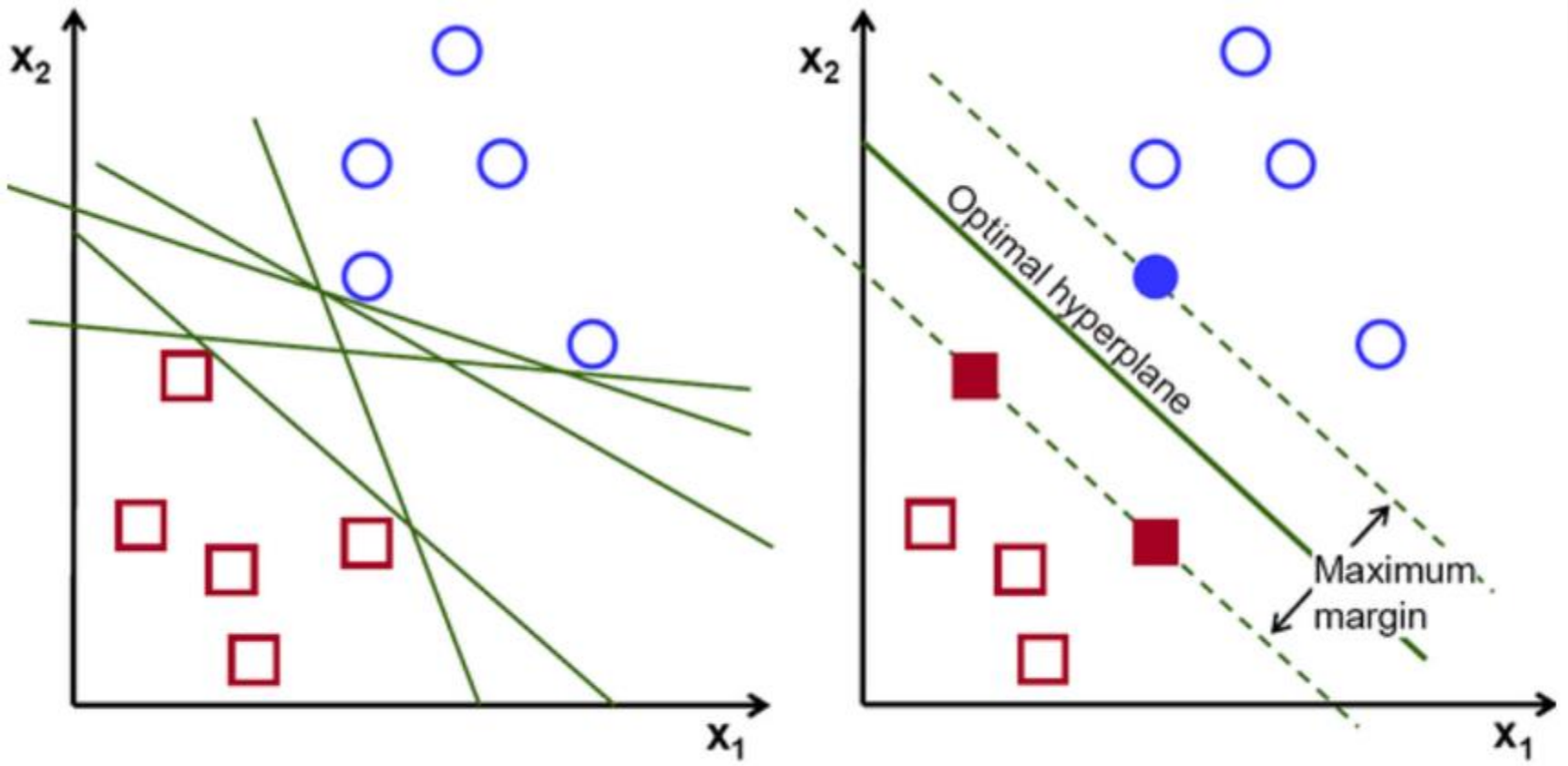
- We plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyperplane that differentiates the two classes very well.

A hyperplane in $\mathbb{R}^2$ is a line



A hyperplane in $\mathbb{R}^3$ is a plane

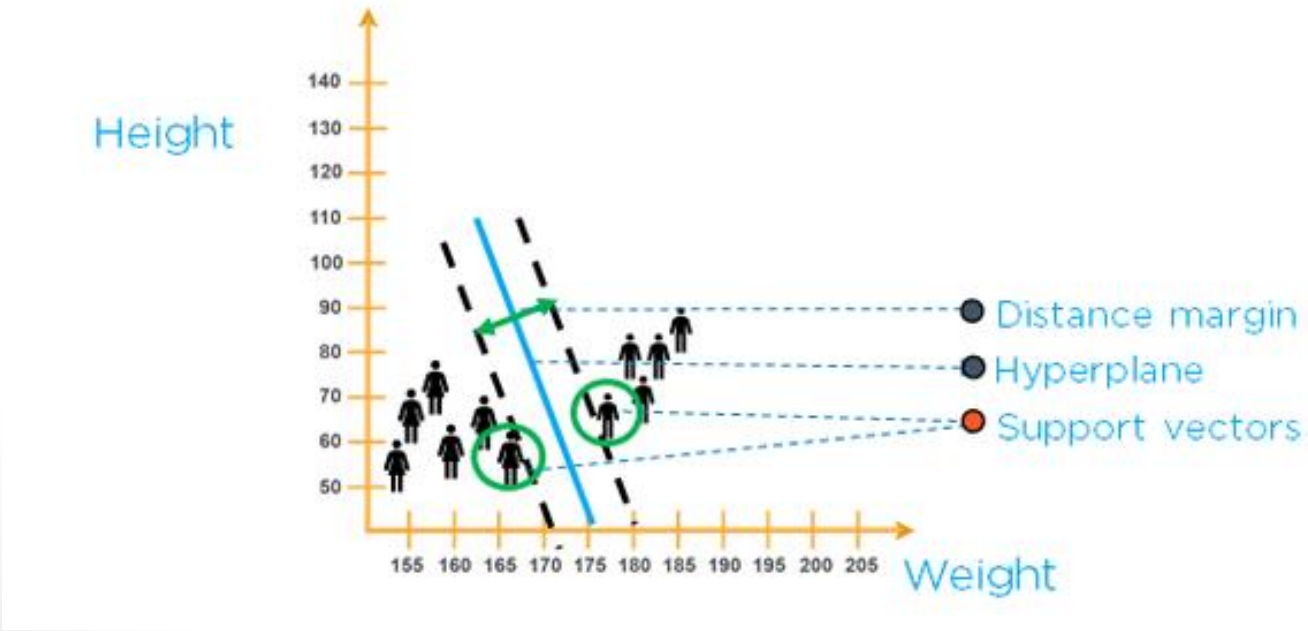




(Source: https://towardsdatascience.com/support-vector-machine-vs-logistic-regression-94cc2975433f)

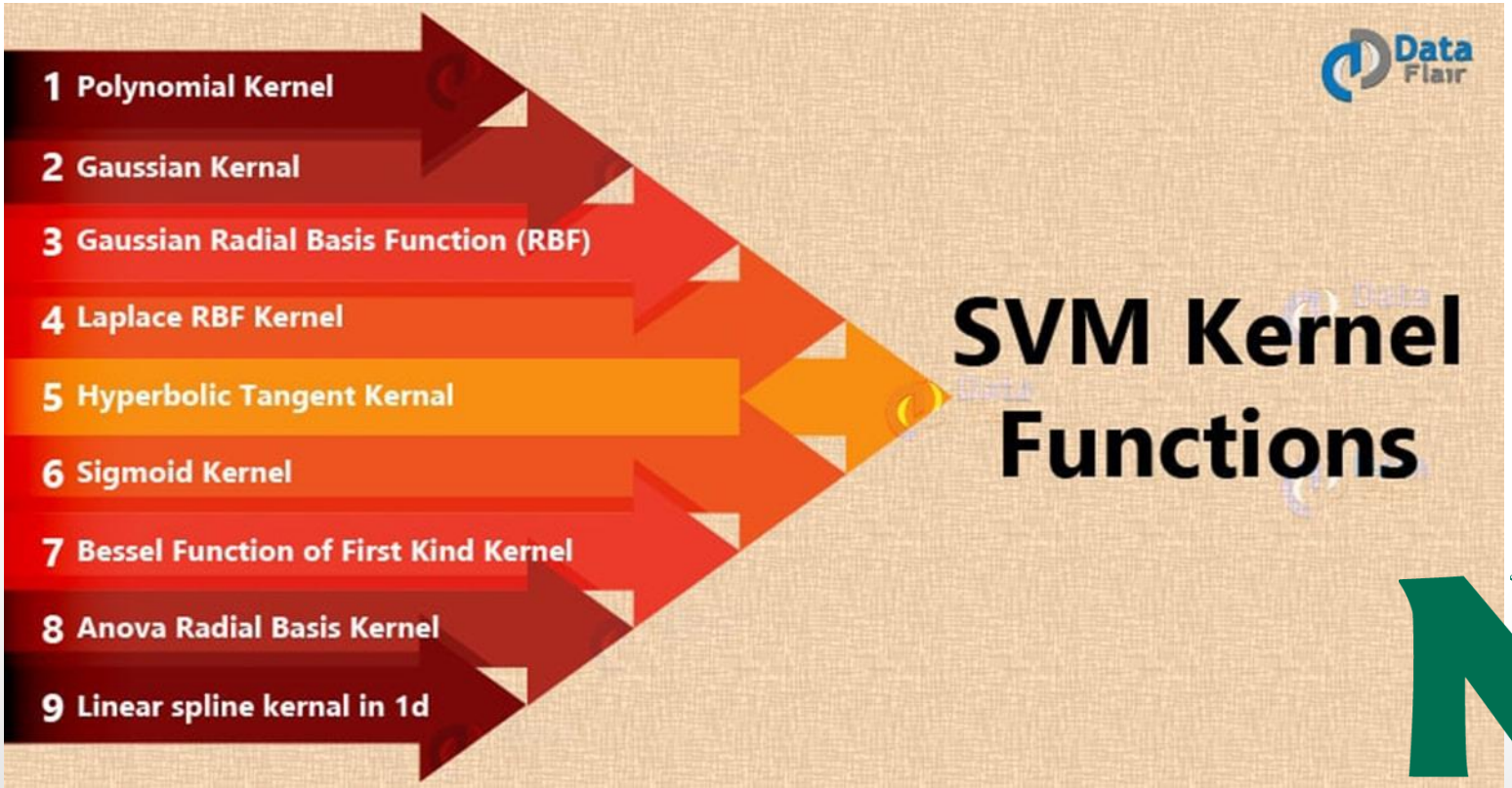
Support Vectors:

Support Vector Machines are an optimization problem. They are attempting to find a hyperplane that divides the two classes with the largest margin. The support vectors are the points which fall within this margin. It's easiest to understand if you build it up from simple to more complex.



SVM Kernel Functions:

SVM algorithms use a set of mathematical functions that are defined as the kernel. The function of kernel is to take data as input and transform it into the required form. Different SVM algorithms use different types of kernel functions.



Example 3: (Titanic)

1. Split the data into two groups training data (80%) and test data (20%).

```
> dim(Titanic_dataset)
[1] 2201    4
> smp_size = floor(0.90 * nrow(Titanic_dataset))
> index = sample(seq_len(nrow(Titanic_dataset)), size = smp_size)
> SVM_train1 = Titanic_dataset[index, ]
> dim(SVM_train1)
[1] 1980    4
> SVM_test1 = Titanic_dataset[-index, ]
> dim(SVM_test1)
[1] 221   4
```

Example 3: (Titanic)

2. Use the SVM classifier to fit the model using linear kernel function.

```
> SVM_Model=svm(Survived ~., data=SVM_train1, kernel = "linear")
> SVM_Model

Call:
svm(formula = Survived ~ ., data = SVM_train1, kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  893
```

Example 3: (Titanic)

3. Find the contingency and probability contingency table.

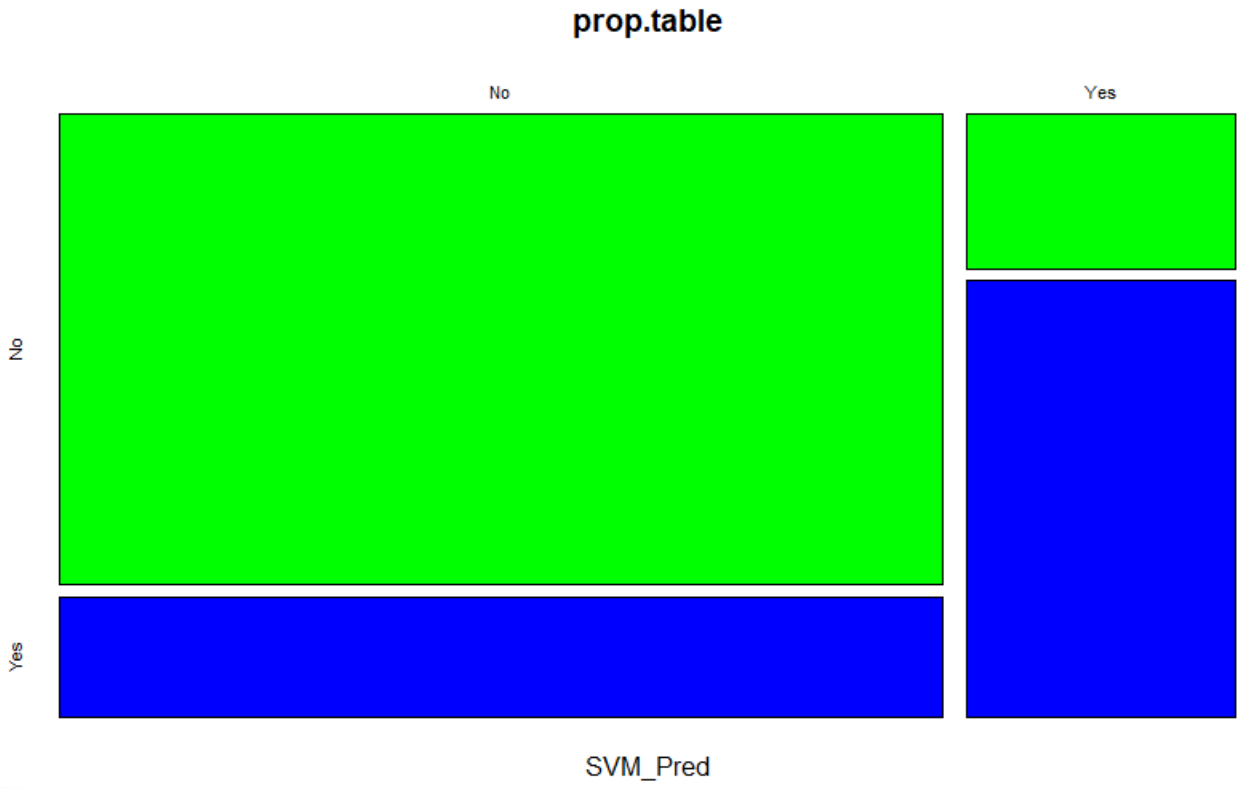
```
> SVM_Pred = predict(SVM_Model, SVM_test1)
> table(SVM_Pred, SVM_test1$Survived)

SVM_Pred  No Yes
     No  136  37
     Yes  12  36
```

```
> prop.table(table(SVM_Pred, SVM_test1$Survived))

SVM_Pred          No        Yes
     No  0.61538462 0.16742081
     Yes 0.05429864 0.16289593
```

Example 3: (Titanic)

4. Find the overall accuracy.

5. Present the probability contingency table using mosaic plot.



Example 3: (Titanic)

4. Compare the overall accuracy for the different kernel functions.

```
> SVM_Model=svm(Survived ~., data=SVM_train1, kernel = "linear")
> SVM_Pred = predict(SVM_Model, SVM_test1)
> mean(SVM_Pred == SVM_test1$Survived)
[1] 0.7782805
```

```
> SVM_Model=svm(Survived ~., data=SVM_train1, kernel = "polynomial")
> SVM_Pred = predict(SVM_Model, SVM_test1)
> mean(SVM_Pred == SVM_test1$Survived)
[1] 0.7647059
```

```
> SVM_Model=svm(Survived ~., data=SVM_train1, kernel = "radial")
> SVM_Pred = predict(SVM_Model, SVM_test1)
> mean(SVM_Pred == SVM_test1$Survived)
[1] 0.7737557
```

```
> SVM_Model=svm(Survived ~., data=SVM_train1, kernel = "sigmoid")
> SVM_Pred = predict(SVM_Model, SVM_test1)
> mean(SVM_Pred == SVM_test1$Survived)
[1] 0.7873303
```

Example 4: (Iris)

1. Split the dataset (*iris*) into two groups training data (80%) and test data (20%).

2. Use the SVM classifier to fit the model using linear kernel function.

```
> smp_size = floor(0.80 * nrow(iris))
> index = sample(seq_len(nrow(iris)), size = smp_size)
> SVM_train2 = iris[index, ]
> SVM_test2 = iris[-index, ]
> SVM_Model2=svm(SVM_train2[,1:4], SVM_train2[,5], kernel = "linear")
> SVM_Model2

Call:
svm.default(x = SVM_train2[, 1:4], y = SVM_train2[, 5], kernel = "linear")


Parameters:
   SVM-Type:  C-classification
 SVM-Kernel:  linear
       cost:  1

Number of Support Vectors:  25
```

Example 4: (Iris)

3. Find the contingency and probability contingency table.

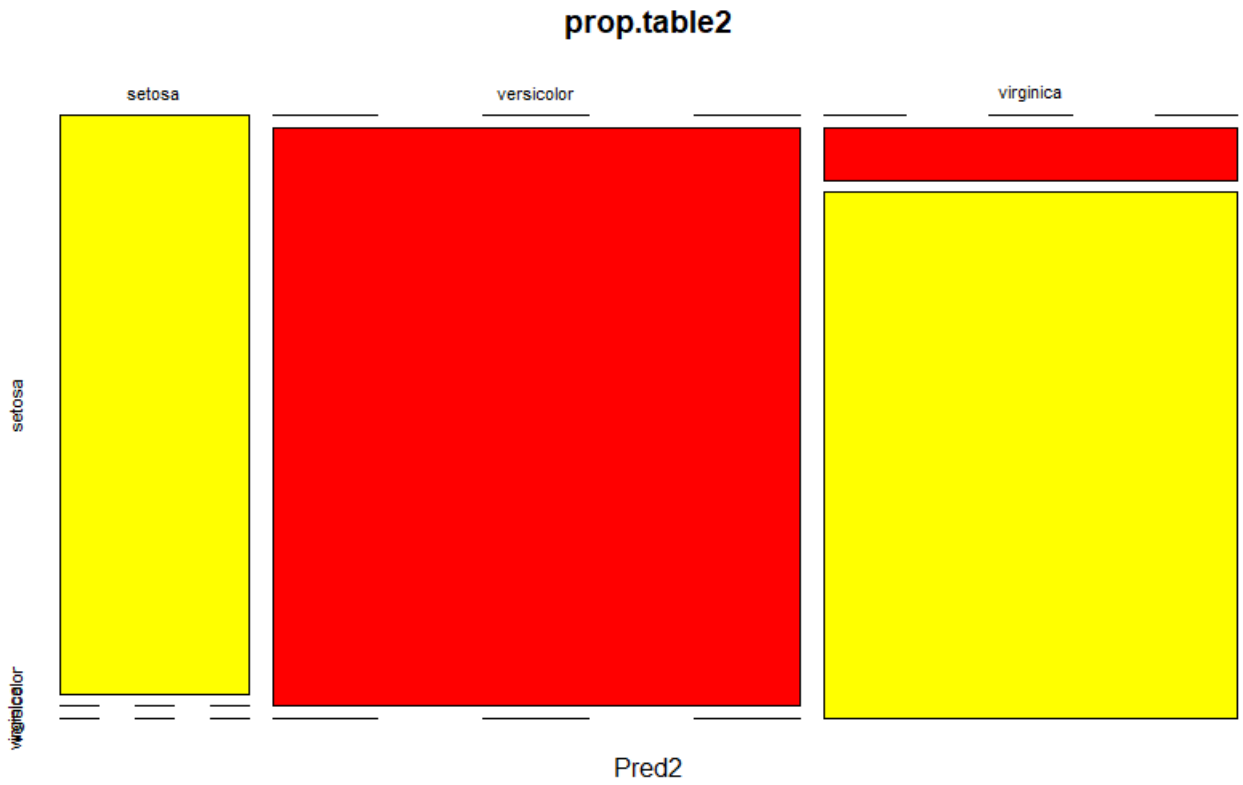
```
> table(Pred2, SVM_test2[,5])

Pred2        setosa versicolor virginica
  setosa          5          0         0
  versicolor      0         14         0
  virginica       0          1        10
>
> prop.table2 = prop.table(table(Pred2, SVM_test2[,5]))
> prop.table2

Pred2             setosa versicolor  virginica
  setosa     0.16666667 0.00000000 0.00000000
  versicolor 0.00000000 0.46666667 0.00000000
  virginica  0.00000000 0.03333333 0.33333333
```

Example 4: (Iris)

4. Find the overall accuracy.

5. Present the probability contingency table using mosaic plot.

```
> mean(Pred2 == SVM_test2[,5])
[1] 0.9666667
>
> plot(prop.table2, col=c('yellow','red'))
```



Example 4: (Iris)

6. Compare the overall accuracy for the different kernel functions.

```
> SVM_Model2=svm(SVM_train2[,1:4], SVM_train2[,5], kernel = "linear")
> Pred2 = predict(SVM_Model2, SVM_test2[,-5])
> mean(Pred2 == SVM_test2[,5])
[1] 0.9666667
```

```
> SVM_Model2=svm(SVM_train2[,1:4], SVM_train2[,5], kernel = "polynomial")
> Pred2 = predict(SVM_Model2, SVM_test2[,-5])
> mean(Pred2 == SVM_test2[,5])
[1] 0.9333333
```

```
> SVM_Model2=svm(SVM_train2[,1:4], SVM_train2[,5], kernel = "radial")
> Pred2 = predict(SVM_Model2, SVM_test2[,-5])
> mean(Pred2 == SVM_test2[,5])
[1] 0.9333333
```

```
> SVM_Model2=svm(SVM_train2[,1:4], SVM_train2[,5], kernel = "sigmoid")
> Pred2 = predict(SVM_Model2, SVM_test2[,-5])
> mean(Pred2 == SVM_test2[,5])
[1] 0.8666667
```

Advantages of the SVM:

1. Very good when we have no idea on the data.
2. More effective in high dimensional spaces.
3. Effective in cases where number of dimensions is greater than the number of samples.

Disadvantages of the SVM:

1. Choosing an appropriate Kernel function is difficult.
2. Requires attributes Scaling.
3. Long training time for large datasets.
4. Difficult to understand and interpret the final model.