Classification

- k-nearest neighbor classifier
- Naïve Bayes
- Logistic Regression
- Support Vector Machines

Classification Analysis
Applications of Gradient Descent, Lagrange and KKT are countless. I am sure that each of you will have to use them some day (if you stay in engineering)

Notes on Optimization
Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point. If instead one takes steps proportional to the positive of the gradient, one approaches a local maximum of that function; the procedure is then known as gradient ascent.
Unconstrained Optimization Using R

\[ f = \text{function}(x)\{(x[1] - 5)^2 + (x[2] - 6)^2\} \]
\[ \text{initial}\_x = c(10, 11) \]
\[ x\_optimal = \text{optim}(\text{initial}\_x, f, \text{method}="CG") \]
\[ x\_min = x\_optimal\$par \]
\[ x\_min \]
Lagrange Method

- In **mathematical optimization**, the **method of Lagrange multipliers** (named after **Joseph Louis Lagrange**) is a strategy for finding the local maxima and minima of a **function** subject to **equality constraints**.
- maximize $f(x, y)$ subject to $g(x, y) = c$.
- We need both $f$ and $g$ to have continuous first **partial derivatives**. We introduce a new variable ($\lambda$) called a Lagrange multiplier and study the Lagrange function (or Lagrangian) defined by

$$\Lambda(x, y, \lambda) = f(x, y) + \lambda \cdot (g(x, y) - c)$$

- where the $\lambda$ term may be either added or subtracted.
Constrained Linear Programming Using R

Maximize expected return: \( f(x_1, x_2, x_3) = x_1 \times 5\% + x_2 \times 4\% + x_3 \times 6\% \)

Subjected to constraints:
10\% < x_1, x_2, x_3 < 100\%
\( x_1 + x_2 + x_3 = 1 \)
\( x_3 < x_1 + x_2 \)
\( x_1 < 2 \times x_2 \)

library(lpSolve)
library(lpSolveAPI)

# Set the number of vars
model <- make.lp(0, 3)

# Define the object function: for Minimize, use -ve
set.objfn(model, c(-0.05, -0.04, -0.06))

# Add the constraints
add.constraint(model, c(1, 1, 1), "=" , 1)
add.constraint(model, c(1, 1, -1), ">" , 0)
add.constraint(model, c(1, -2, 0), "<" , 0)

# Set the upper and lower bounds
set.bounds(model, lower=c(0.1, 0.1, 0.1), upper=c(1, 1, 1))

# Compute the optimized model
solve(model)

# Get the value of the optimized parameters
get.variables(model)

# Get the value of the objective function
get.objective(model)

# Get the value of the constraint
get.constraints(model)
In mathematical optimization, the Karush–Kuhn–Tucker (KKT) conditions (also known as the Kuhn–Tucker conditions) are first order necessary conditions for a solution in nonlinear programming to be optimal, provided that some regularity conditions are satisfied. Allowing inequality constraints, the KKT approach to nonlinear programming generalizes the method of Lagrange multipliers, which allows only equality constraints. The system of equations corresponding to the KKT conditions is usually not solved directly, except in the few special cases where a closed-form solution can be derived analytically. In general, many optimization algorithms can be interpreted as methods for numerically solving the KKT system of equations.
Minimize quadratic objective function:

\[ f(x_1, x_2) = c_1 x_1^2 + c_2 x_1 x_2 + c_2 x_2^2 - (d_1 x_1 + d_2 x_2) \]

Subject to constraints

\[ a_{11} x_1 + a_{12} x_2 = b_1 \]
\[ a_{21} x_1 + a_{22} x_2 = b_2 \]
\[ a_{31} x_1 + a_{32} x_2 = b_3 \]
\[ a_{41} x_1 + a_{42} x_2 \geq b_4 \]
\[ a_{51} x_1 + a_{52} x_2 \geq b_5 \]

library(quadprog)
mu_return_vector = c(0.05, 0.04, 0.06)
sigma = matrix(c(0.01, 0.002, 0.005,
                 0.002, 0.008, 0.006,
                 0.005, 0.006, 0.012),
                nrow=3, ncol=3)
D.Matrix = 2*sigma
d.Vector = rep(0, 3)
A.Equality = matrix(c(1,1,1), ncol=1)
A.Matrix = cbind(A.Equality, mu_return_vector, diag(3))
b.Vector = c(1, 0.052, rep(0, 3))
out = solve.QP(Dmat=D.Matrix, dvec=d.Vector,
               Amat=A.Matrix, bvec=b.Vector,
               meq=1)
out$solution
out$value
NEAREST NEIGHBOR CLASSIFICATION
Illustrating Classification Task

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>55K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>80K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>67K</td>
<td>?</td>
</tr>
</tbody>
</table>

Training Set

Learning algorithm

Induction

Apply Model

Deduction

Test Set
## Instance-Based Classifiers

### Set of Stored Cases

<table>
<thead>
<tr>
<th>Atr1</th>
<th>........</th>
<th>AtrN</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>C</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
</tbody>
</table>

- Store the training records
- Use training records to predict the class label of unseen cases

### Unseen Case

<table>
<thead>
<tr>
<th>Atr1</th>
<th>........</th>
<th>AtrN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Instance Based Classifiers

- **Examples:**
  - **Rote-learner**
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - **Nearest neighbor classifier**
    - Uses k “closest” points (nearest neighbors) for performing classification
Nearest Neighbor Classifiers

- **Basic idea:**
  - “If it walks like a duck, quacks like a duck, then it’s probably a duck”

  ![Diagram](https://via.placeholder.com/150)

  - Training Records
  - Compute Distance
  - Test Record
  - Choose k of the “nearest” records
Nearest-Neighbor Classifiers

- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of $k$, the number of nearest neighbors to retrieve

- To classify an unknown record:
  1. Compute distance to other training records
  2. Identify $k$ nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)
Definition of Nearest Neighbor

K-nearest neighbors of a record $x$ are data points that have the $k$ smallest distance to $x$
1 nearest-neighbor

Voronoi Diagram defines the classification boundary

The area takes the class of the green point
Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance

\[ d(p, q) = \sqrt{\sum_{i} (p_i - q_i)^2} \]

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, \( w = \frac{1}{d^2} \)
Choosing the value of $k$:

- If $k$ is too small, sensitive to noise points
- If $k$ is too large, neighborhood may include points from other classes
Scaling issues
- Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes

Example:
- height of a person may vary from 1.5m to 1.8m
- weight of a person may vary from 90lb to 300lb
- income of a person may vary from $10K to $1M
Problem with Euclidean measure:
- High dimensional data
  - curse of dimensionality
- Can produce counter-intuitive results

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
\end{array}
\]

\[d = 1.4142\] vs \[d = 1.4142\]

Solution: Normalize the vectors to unit length
k-NN classifiers are lazy learners
- It does not build models explicitly
- Unlike eager learners such as decision trees

Classifying unknown records are relatively expensive
- Naïve algorithm: O(n)
- Need for structures to retrieve nearest neighbors fast.
  - The Nearest Neighbor Search problem.
Nearest Neighbor Search

- Two-dimensional \textit{kd}-trees
  - A data structure for answering nearest neighbor queries in $\mathbb{R}^2$

- \textit{kd}-tree construction algorithm
  - Select the $x$ or $y$ dimension (alternating between the two)
  - Partition the space into two with a line passing from the median point
  - Repeat recursively in the two partitions as long as there are enough points
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees
Nearest Neighbor Search

2-dimensional kd-trees

region(u) – all the black points in the subtree of u
Nearest Neighbor Search

2-dimensional kd-trees

- A binary tree:
  - Size $O(n)$
  - Depth $O(\log n)$
  - Construction time $O(n\log n)$
  - Query time: worst case $O(n)$, but for many cases $O(\log n)$

Generalizes to d dimensions

- Example of Binary Space Partitioning
install.packages(c("class"))
library(class)
train = rbind(iris3[1:25,,1], iris3[1:25,,2], iris3[1:25,,3])
test = rbind(iris3[26:50,,1], iris3[26:50,,2], iris3[26:50,,3])
cl = factor(c(rep("s",25), rep("c",25), rep("v",25)))
knn(train, test, cl, k = 3, prob=TRUE)
attributes(.Last.value)
SUPPORT VECTOR MACHINES
Find a linear hyperplane (decision boundary) that will separate the data
Support Vector Machines

- One Possible Solution
Support Vector Machines

- Another possible solution
Support Vector Machines

- Other possible solutions
Support Vector Machines

- Which one is better? B1 or B2?
- How do you define better?
- Find hyperplane maximizes the margin => B1 is better than B2
Support Vector Machines

\[ \mathbf{w} \cdot \mathbf{x} + b = 0 \]

\[ \mathbf{w} \cdot \mathbf{x} + b = -1 \]

\[ f(\mathbf{x}) = \begin{cases} 
1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \geq 1 \\
-1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b \leq -1
\end{cases} \]

Margin = \frac{b_{12}}{||\mathbf{w}||}
Support Vector Machines

- We want to maximize:
  \[ \text{Margin} = \frac{2}{\| \mathbf{w} \|^2} \]

- Which is equivalent to minimizing:
  \[ L(w) = \frac{|| \mathbf{w} ||^2}{2} \]

- But subjected to the following constraints:
  \[
  \begin{align*}
  \mathbf{w} \cdot \mathbf{x}_i + b &\geq 1 \text{ if } y_i = 1 \\
  \mathbf{w} \cdot \mathbf{x}_i + b &\leq -1 \text{ if } y_i = -1
  \end{align*}
  \]

- This is a constrained optimization problem
  - Numerical approaches to solve it (e.g., quadratic programming)
What if the problem is not linearly separable?
Support Vector Machines

- What if the problem is not linearly separable?
What if the problem is not linearly separable?

- Introduce slack variables
  - Need to minimize:
    
    \[ L(w) = \frac{\|w\|^2}{2} + C \left( \sum_{i=1}^{N} \xi_i \right) \]

- Subject to:
  
  \[ w \cdot x_i + b \geq 1 - \xi_i \text{ if } y_i = 1 \]
  
  \[ w \cdot x_i + b \leq -1 + \xi_i \text{ if } y_i = -1 \]
What if decision boundary is not linear?
Nonlinear Support Vector Machines

- Transform data into higher dimensional space

Use the Kernel Trick

\[ (x_1 + x_2)^4 \]
Kernel Functions

input space

future space
Kernel Functions
Kernel Functions

A kernel is a function $k(\mathbf{x}_i, \mathbf{x}_j)$ that given two vectors in input space, returns the dot product of their images in feature space $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$

So by computing the dot product directly using a kernel function, one avoid the mapping $\phi(\mathbf{x})$. This is desirable because $Z$ has possibly infinite dimensions and $\phi(\mathbf{x})$ can be tricky or impossible to compute. Using a kernel function, one need not explicitly know what $\phi(\mathbf{x})$ is. By using a kernel function, a SVM that operates in infinite dimensional space can be constructed.
Kernel Functions

Polynomial kernel

\[ K(x_i, x_j) = (1 + x_i \cdot x_j)^d \]

Radial basis kernel

Commonly used radial basis kernel is Gaussian kernel:

\[ K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right) \]

Sigmoid kernel:

\[ K(x, y) = \tanh(\kappa x^T y + \theta) \]

With user defined parameter \( \kappa \) and \( \theta \).
# Example 1: XOR Function
x = array(data = c(0,0,1,1,0,1,0,1),dim=c(4,2))
y = factor(c(1,-1,-1,1))
model = svm(x,y)
predict(model,x)

# Example 2: IRIS dataset
data(iris)
attach(iris)
x = subset(iris, select = -Species)
y = Species
model = svm(x, y)
print(model)
summary(model)
pred <- predict(model, x)
table(pred, y)
LOGISTIC REGRESSION
Classification via regression

- Instead of predicting the class of an record we want to predict the probability of the class given the record.
- The problem of predicting continuous values is called regression problem.
- General approach: find a continuous function that models the continuous points.
Example: Linear regression

- Given a dataset of the form 
  \{(x_1, y_1), ..., (x_n, y_n)\} find a linear function that given the vector \(x_i\) predicts the \(y_i\) value as \(y'_i = w^T x_i\)
  - Find a vector of weights \(w\) that minimizes the sum of square errors
  \[
  \sum_i (y'_i - y_i)^2
  \]
  - Several techniques for solving the problem.
Assume a linear classification boundary \( w \cdot x > 0 \)

For the positive class the bigger the value of \( w \cdot x \), the further the point is from the classification boundary, the higher our certainty for the membership to the positive class

- Define \( P(C_+ | x) \) as an increasing function of \( w \cdot x \)

For the negative class the smaller the value of \( w \cdot x \), the further the point is from the classification boundary, the higher our certainty for the membership to the negative class

- Define \( P(C_- | x) \) as a decreasing function of \( w \cdot x \)
Logistic Regression

The logistic function

\[ f(t) = \frac{1}{1 + e^{-t}} \]

\[ P(C_+ | x) = \frac{1}{1 + e^{-w \cdot x}} \]

\[ P(C_- | x) = \frac{e^{-w \cdot x}}{1 + e^{-w \cdot x}} \]

\[ \log \frac{P(C_+ | x)}{P(C_- | x)} = w \cdot x \]

Linear regression on the log-odds ratio

Logistic Regression: Find the vector \( w \) that maximizes the probability of the observed data.
Logistic Regression

- Produces a probability estimate for the class membership which is often very useful.
- The weights can be useful for understanding the feature importance.
- Works for relatively large datasets
- Fast to apply.
# Example 1: Linear Regression
\[ x = c(1, 2, 3, 4) \]
\[ y = 2x + 3 \]
\[ \text{lm}(y \sim x) \]

# Example 2: Logistic Regression
```r
data(iris)
iris.sub <- subset(iris, Species%in%c("versicolor","virginica"))
model <- glm(Species ~ Sepal.Length + Sepal.Width, 
data = iris.sub,
             family = binomial(link = "logit"))
hist(predict(model), type = "response")
```
NAÏVE BAYES CLASSIFIER
Bayes Classifier

- A probabilistic framework for solving classification problems
- \( A, C \) random variables
- **Joint** probability: \( \text{Pr}(A=a, C=c) \)
- **Conditional** probability: \( \text{Pr}(C=c \mid A=a) \)
- Relationship between joint and conditional probability distributions
  \[ \text{Pr}(C, A) = \text{Pr}(C \mid A) \times \text{Pr}(A) = \text{Pr}(A \mid C) \times \text{Pr}(C) \]

- **Bayes Theorem:**
  \[ P(C \mid A) = \frac{P(A \mid C)P(C)}{P(A)} \]
Bayesian Classifiers

Consider each attribute and class label as random variables

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Evade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Evade C
Event space: \{Yes, No\}
P(C) = (0.3, 0.7)

Refund A_1
Event space: \{Yes, No\}
P(A_1) = (0.3, 0.7)

Martial Status A_2
Event space: \{Single, Married, Divorced\}
P(A_2) = (0.4, 0.4, 0.2)

Taxable Income A_3
Event space: R
P(A_3) \sim \text{Normal}(\mu, \sigma)
Bayesian Classifiers

- Given a record $X$ over attributes $(A_1, A_2, \ldots, A_n)$
  - E.g., $X = (\text{‘Yes’}, \text{‘Single’}, 125K)$

- The goal is to predict class $C$
  - Specifically, we want to find the value $c$ of $C$ that maximizes $P(C=c| X)$
    - Maximum Aposteriori Probability estimate

- Can we estimate $P(C| X)$ directly from data?
  - This means that we estimate the probability for all possible values of the class variable.
Bayesian Classifiers

- **Approach:**
  - compute the posterior probability \( P(C \mid A_1, A_2, \ldots, A_n) \) for all values of \( C \) using the Bayes theorem

\[
P(C \mid A_1 A_2 \ldots A_n) = \frac{P(A_1 A_2 \ldots A_n \mid C) P(C)}{P(A_1 A_2 \ldots A_n)}
\]

- Choose value of \( C \) that maximizes \( P(C \mid A_1, A_2, \ldots, A_n) \)

- Equivalent to choosing value of \( C \) that maximizes \( P(A_1, A_2, \ldots, A_n \mid C) P(C) \)

- How to estimate \( P(A_1, A_2, \ldots, A_n \mid C) \)?
Naïve Bayes Classifier

- Assume independence among attributes $A_i$ when class is given:
  - $P(A_1, A_2, \ldots, A_n|C) = P(A_1|C) P(A_2|C) \cdots P(A_n|C)$

- We can estimate $P(A_i|C)$ for all values of $A_i$ and $C$.

- New point $X$ is classified to class $c$ if
  $$P(C = c|X) = P(C = c) \prod_i P(A_i|c)$$
  is maximum over all possible values of $C$. 
## How to Estimate Probabilities from Data?

- **Class Prior Probability:**
  \[ P(C = c) = \frac{N_c}{N} \]
  
  e.g., \( P(C = \text{No}) = \frac{7}{10}, \quad P(C = \text{Yes}) = \frac{3}{10} \)

- **For discrete attributes:**
  \[ P(A_i = a | C = c) = \frac{N_{a,c}}{N_c} \]
  
  where \( N_{a,c} \) is number of instances having attribute \( A_i = a \) and belongs to class \( c \)

- **Examples:**
  - \( P(\text{Status}=\text{Married}|\text{No}) = \frac{4}{7} \)
  - \( P(\text{Refund}=\text{Yes}|\text{Yes}) = 0 \)

<table>
<thead>
<tr>
<th>Tid</th>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Evade</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Single</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Married</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Single</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Married</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Divorced</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Married</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Divorced</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Single</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Married</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Single</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>
For **continuous** attributes:

- **Discretize** the range into bins
  - one ordinal attribute per bin
  - violates independence assumption
- **Two-way split**: \( (A < v) \) or \( (A > v) \)
  - choose only one of the two splits as new attribute

**Probability density estimation:**

- Assume attribute follows a **normal distribution**
- Use data to estimate parameters of distribution (i.e., **mean** \( \mu \) and **standard deviation** \( \sigma \))
- Once probability distribution is known, we can use it to estimate the conditional probability \( P(A_i|c) \)
How to Estimate Probabilities from Data?

- Normal distribution:
  \[
P(A_i = a \mid c_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} e^{-\frac{(a - \mu_{ij})^2}{2\sigma_{ij}^2}}
\]
  - One for each \((a_i, c_i)\) pair
  - For (Income, Class=No):
    - If Class=No
      - sample mean = 110
      - sample variance = 2975

\[
P(\text{Income} = 120 \mid \text{No}) = \frac{1}{\sqrt{2\pi(54.54)}} e^{-\frac{(120-110)^2}{2(2975)}} = 0.0072
\]
Creating a Naïve Bayes Classifier, essentially means to compute **counts**: 

Total number of records: \( N = 10 \)

- **Class No:**
  - Number of records: 7
  - Attribute Refund:
    - Yes: 3
    - No: 4
  - Attribute Marital Status:
    - Single: 2
    - Divorced: 1
    - Married: 4
  - Attribute Income:
    - mean: 110
    - variance: 2975

- **Class Yes:**
  - Number of records: 3
  - Attribute Refund:
    - Yes: 0
    - No: 3
  - Attribute Marital Status:
    - Single: 2
    - Divorced: 1
    - Married: 0
  - Attribute Income:
    - mean: 90
    - variance: 25
Example of Naïve Bayes Classifier

Given a Test Record:

\[ X = (\text{Refund} = \text{No}, \text{Married}, \text{Income} = 120\text{K}) \]

naive Bayes Classifier:

\[
\begin{align*}
P(\text{Refund}=\text{Yes}|\text{No}) &= 3/7 \\
P(\text{Refund}=\text{No}|\text{No}) &= 4/7 \\
P(\text{Refund}=\text{Yes}|\text{Yes}) &= 0 \\
P(\text{Refund}=\text{No}|\text{Yes}) &= 1 \\

P(\text{Marital Status}=\text{Single}|\text{No}) &= 2/7 \\
P(\text{Marital Status}=\text{Divorced}|\text{No}) &= 1/7 \\
P(\text{Marital Status}=\text{Married}|\text{No}) &= 4/7 \\
P(\text{Marital Status}=\text{Single}|\text{Yes}) &= 2/7 \\
P(\text{Marital Status}=\text{Divorced}|\text{Yes}) &= 1/7 \\
P(\text{Marital Status}=\text{Married}|\text{Yes}) &= 0
\end{align*}
\]

For taxable income:

If class=No: sample mean=110 \\
                  sample variance=2975

If class=Yes: sample mean=90 \\
               sample variance=25

\[
\begin{align*}
P(X|\text{Class}=\text{No}) &= P(\text{Refund}=\text{No}|\text{Class}=\text{No}) \\
&\quad \times P(\text{Married}|\text{Class}=\text{No}) \\
&\quad \times P(\text{Income}=120\text{K}|\text{Class}=\text{No}) \\
&= \frac{4}{7} \times \frac{4}{7} \times 0.0072 = 0.0024
\end{align*}
\]

\[
\begin{align*}
P(X|\text{Class}=\text{Yes}) &= P(\text{Refund}=\text{No}|\text{Class}=\text{Yes}) \\
&\quad \times P(\text{Married}|\text{Class}=\text{Yes}) \\
&\quad \times P(\text{Income}=120\text{K}|\text{Class}=\text{Yes}) \\
&= 1 \times 0 \times 1.2 \times 10^{-9} = 0
\end{align*}
\]

\[
P(\text{No}) = 0.3, \quad P(\text{Yes}) = 0.7
\]

Since \( P(X|\text{No})P(\text{No}) > P(X|\text{Yes})P(\text{Yes}) \)

Therefore \( P(\text{No}|X) > P(\text{Yes}|X) \)

\[ \Rightarrow \text{Class} = \text{No} \]
Naïve Bayes Classifier

- If one of the conditional probability is zero, then the entire expression becomes zero.
- Probability estimation:

Original: \( P(A_i = a \mid C = c) = \frac{N_{ac}}{N_c} \)

Laplace: \( P(A_i = a \mid C = c) = \frac{N_{ac} + 1}{N_c + N_i} \)  

m-estimate: \( P(A_i = a \mid C = c) = \frac{N_{ac} + mp}{N_c + m} \)

- \( N_i \): number of attribute values for attribute \( A_i \)
- \( p \): prior probability
- \( m \): parameter
Example of Naïve Bayes Classifier

Given a Test Record:

\[ X = (\text{Refund} = \text{No}, \text{Married}, \text{Income} = 120K) \]

**naive Bayes Classifier:**

- \[ P(\text{Refund} = \text{Yes} | \text{No}) = 4/9 \]
- \[ P(\text{Refund} = \text{No} | \text{No}) = 5/9 \]
- \[ P(\text{Refund} = \text{Yes} | \text{Yes}) = 1/5 \]
- \[ P(\text{Refund} = \text{No} | \text{Yes}) = 4/5 \]

- \[ P(\text{Marital Status} = \text{Single} | \text{No}) = 3/10 \]
- \[ P(\text{Marital Status} = \text{Divorced} | \text{No}) = 2/10 \]
- \[ P(\text{Marital Status} = \text{Married} | \text{No}) = 5/10 \]
- \[ P(\text{Marital Status} = \text{Single} | \text{Yes}) = 3/6 \]
- \[ P(\text{Marital Status} = \text{Divorced} | \text{Yes}) = 2/6 \]
- \[ P(\text{Marital Status} = \text{Married} | \text{Yes}) = 1/6 \]

For taxable income:
- If class=No: sample mean=110 sample variance=2975
- If class=Yes: sample mean=90 sample variance=25

- \[ P(X | \text{Class} = \text{No}) = P(\text{Refund} = \text{No} | \text{Class} = \text{No}) \times P(\text{Married} | \text{Class} = \text{No}) \times P(\text{Income} = 120K | \text{Class} = \text{No}) \]
  \[ = \frac{5}{9} \times \frac{5}{10} \times 0.0072 \]

- \[ P(X | \text{Class} = \text{Yes}) = P(\text{Refund} = \text{No} | \text{Class} = \text{Yes}) \times P(\text{Married} | \text{Class} = \text{Yes}) \times P(\text{Income} = 120K | \text{Class} = \text{Yes}) \]
  \[ = \frac{4}{5} \times \frac{1}{6} \times 1.2 \times 10^{-9} \]

\[ P(\text{No}) = 0.7, \quad P(\text{Yes}) = 0.3 \]

Since \[ P(X | \text{No})P(\text{No}) > P(X | \text{Yes})P(\text{Yes}) \]

Therefore \[ P(\text{No} | X) > P(\text{Yes} | X) \]

\[ => \text{Class} = \text{No} \]
Implementation details

- Computing the conditional probabilities involves multiplication of many very small numbers
  - Numbers get very close to zero, and there is a danger of numeric instability
- We can deal with this by computing the logarithm of the conditional probability

\[
\log P(C|A) \sim \log P(A|C) + \log P(A) = \sum_i \log(A_i|C) + \log P(A)
\]
Naïve Bayes is commonly used for text classification.

For a document \( d = (t_1, ..., t_k) \)

\[
P(c|d) = P(c) \prod_{t_i \in d} P(t_i|c)
\]

- \( P(t_i|c) \) = Fraction of terms from all documents in \( c \) that are \( t_i \).

- Easy to implement and works relatively well
- Limitation: Hard to incorporate additional features (beyond words).
**TrainMultinomialNB(C, D)**

1. \( V \leftarrow \text{EXTRACTVOCABULARY}(D) \)
2. \( N \leftarrow \text{COUNTDOCS}(D) \)
3. \textbf{for each} \( c \in C \)
4. \( \textbf{do} \ N_c \leftarrow \text{COUNTDOCSINCLASS}(D, c) \)
5. \( \text{prior}[c] \leftarrow N_c / N \)
6. \( \text{text}_c \leftarrow \text{CONCATENATETEXTOFALLDOCSINCLASS}(D, c) \)
7. \textbf{for each} \( t \in V \)
8. \( \textbf{do} \ T_{ct} \leftarrow \text{COUNTTOKENSOFTERM}(\text{text}_c, t) \)
9. \textbf{for each} \( t \in V \)
10. \( \textbf{do} \ \text{condprob}[t][c] \leftarrow \frac{T_{ct} + 1}{\sum_{t'}(T_{t'c} + 1)} \)
11. \textbf{return} \( V, \text{prior}, \text{condprob} \)

**ApplyMultinomialNB(C, V, prior, condprob, d)**

1. \( W \leftarrow \text{EXTRACTTOKENSFROMDOC}(V, d) \)
2. \textbf{for each} \( c \in C \)
3. \( \textbf{do} \ \text{score}[c] \leftarrow \log \text{prior}[c] \)
4. \textbf{for each} \( t \in W \)
5. \( \textbf{do} \ \text{score}[c] += \log \text{condprob}[t][c] \)
6. \textbf{return} \ \arg \max_{c \in C} \text{score}[c] \)

---

**Figure 13.2** Naive Bayes algorithm (multinomial model): Training and testing.
Naïve Bayes (Summary)

- Robust to isolated noise points
- Handle missing values by ignoring the instance during probability estimate calculations
- Robust to irrelevant attributes
- Independence assumption may not hold for some attributes
  - Use other techniques such as Bayesian Belief Networks (BBN)
- Naïve Bayes can produce a probability estimate, but it is usually a very biased one
  - Logistic Regression is better for obtaining probabilities.
install.packages(e1071)
library(e1071)
classifier<-naiveBayes(iris[,1:4], iris[,5])
table(predict(classifier, iris[, -5]), iris[, 5], dnn=list('predicted', 'actual'))
classifier$apriori
classifier$tables$Petal.Length
Generative vs Discriminative models

- Naïve Bayes is a type of a generative model
  - Generative process:
    - First pick the category of the record
    - Then given the category, generate the attribute values from the distribution of the category
  - Conditional independence given C

- We use the training data to learn the distribution of the values in a class
Logistic Regression and SVM are **discriminative models**

- The goal is to find the boundary that discriminates between the two classes from the training data.

In order to classify the language of a document, you can

- Either learn the two languages and find which is more likely to have generated the words you see.
- Or learn what differentiates the two languages.