Aprendizagem Aplicada à Segurança

(Mestrado em Cibersegurança-DETI-UA)

Lecture 2
Supervised learning – Classification

Petia Georgieva
(petia@ua.pt)

DETI/IEETA – UA

OUTLINE

- **Logistic Regression (logit model)**
- **Support Vector machines (SVM)**
- **K- Nearest-Neighbor (k-NN)**
- **Decision Tree (DT)**

Classification –

LOGISTIC REGRESSION (LOGIT)

Binary vs Multiclass Classification

Email: Spam /Not Spam ?
Tumour: Malignant /Benign ?
Online Transactions: Fraudulent (Yes /No) ?

Binary classification:
y = 1: “positive class” (e.g. malignant tumour)
y = 0: “negative class” (e.g. benign tumour)

Find a model h(x) that outputs values between 0 and 1
0<=h(x)<=1
if h(x) >=0.5, predict “y=1”
if h(x) <0.5, predict “y=0”

Multiclass classification (*K* classes) => y= {0, 1, 2,..}
Build *K* binary classifiers, for each classifier one of the classes has label 1 all other classes take label 0 .

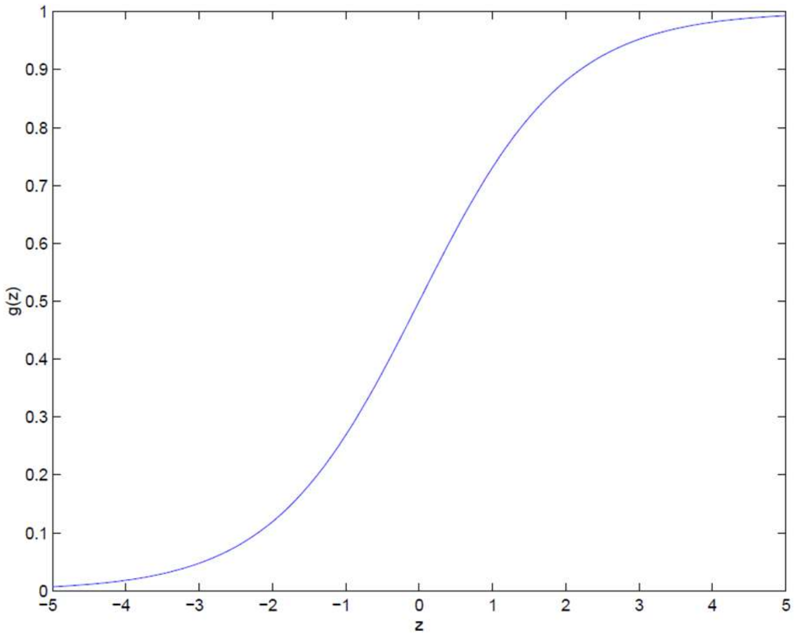
Logistic Regression

Given labelled data of *m* examples, *n* features
Labels {0,1} => binary classification
x –vector of features; θ – vector of model parameters;
h(x) – logistic (sigmoid function) model – logit model

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}} = \frac{1}{1+e^{-z}} = g(\theta^T x) = g(z)$$

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots.\theta_n x_n$$

Logistic (sigmoid) function



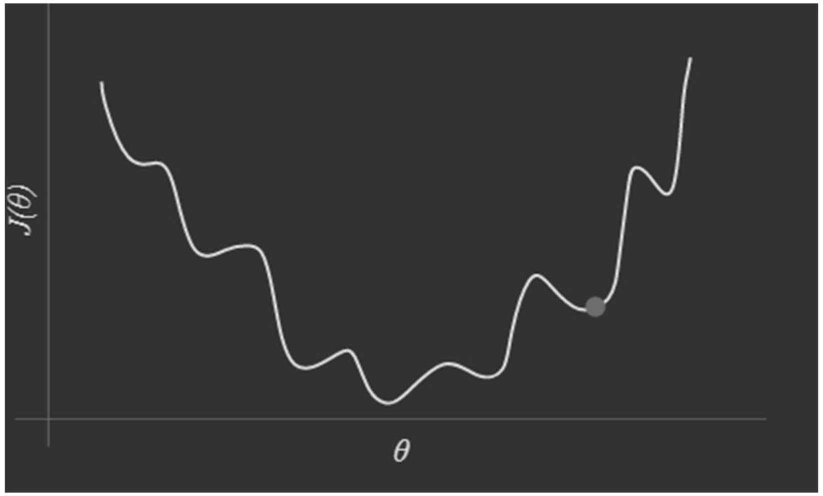
Logistic Regression Cost Function

Linear regression model => $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots\theta_n x_n = \vec{\theta}^T \vec{x}$

Lin Regr. cost (loss) function (MSE) => $J = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Nonlinear logit model => $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

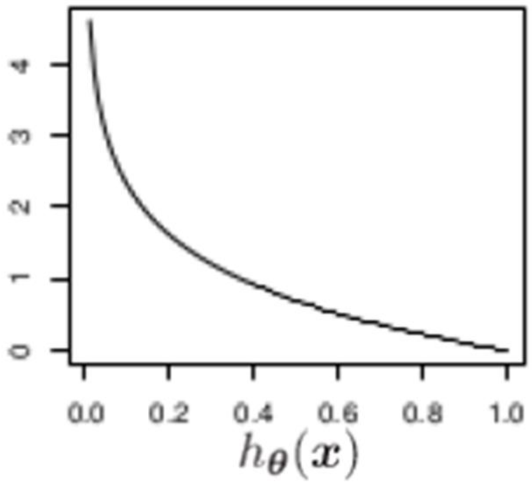
If we use the same cost function as with linear regression, but now we have the nonlinear logit model, *J(θ)* will be a non-convex function (has many local minima)=> **not efficient for optimization !**



Logistic Regression Cost Function

$$\text{cost}\,(h_{\boldsymbol{\theta}}(\boldsymbol{x}),\ y) = \begin{cases} -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if}\quad y = 1 \\ -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x})) & \text{if}\quad y = 0 \end{cases}$$

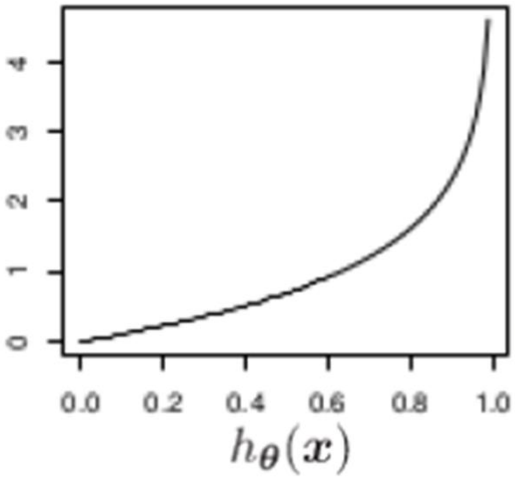
if $\quad y = 1$



if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 1$
then cost $= 0$

if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \to 0$
then cost $\to \infty$
predicted
$\text{prob}(y = 1 \,|\, \boldsymbol{x}; \boldsymbol{\theta}) = 0$
but $y = 1$

if $\quad y = 0$



if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = 0$
then cost $= 0$

if $h_{\boldsymbol{\theta}}(\boldsymbol{x}) \to 1$
then cost $\to \infty$
predicted
$\text{prob}(y = 0 \,|\, \boldsymbol{x}; \boldsymbol{\theta}) = 0$
but $y = 0$

Logistic regression cost function combined into one expression :
(also known as binary Cross-Entropy or Log Loss function)

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left[-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right]$$

Logit with gradient descent learning

Inicialize model parameters *(e.g. θ =0)*

Repeat until J converge {

Compute Logit Model prediction =>
(different from linear regression model)

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

Compute Logit cost function =>
(different from linear regression cost function)

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Goal =>

$$\min_\theta J(\theta)$$

Compute cost function gradients =>
(same as linear regression gradients)

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Update parameters =>
(same as linear regression parameter update)

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

}

Optimization algorithms

Gradient descent (learned in class) -
updates the parameters in direction in which the gradient decreases most rapidly.

2. Other optimization algorithms

- Conjugate gradient
- AdaGrad, RMSProp
- Stochastic gradient descent with momentum
- ADAM (combination of RMSProp and stochastic optimization)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno)
- Quasi-Newton methods (approximate the second derivative)

Characteristics

- Adaptive learning rate (alfa);
- Often faster than gradient descent; better convergence;
- Approximate (estimate) the true gradient over a mini-batch and not over the whole data;
- More complex algorithms

Logistic regression - example

Ex.: We have applicant's scores on two admission exams in the univ (these are the features x_1 and x_2) and we know the final decision (admitted or not admitted – the labels y). Build a logistic regression model to tell what are the chances for new candidates to be admitted into the university if we know their exam scores.

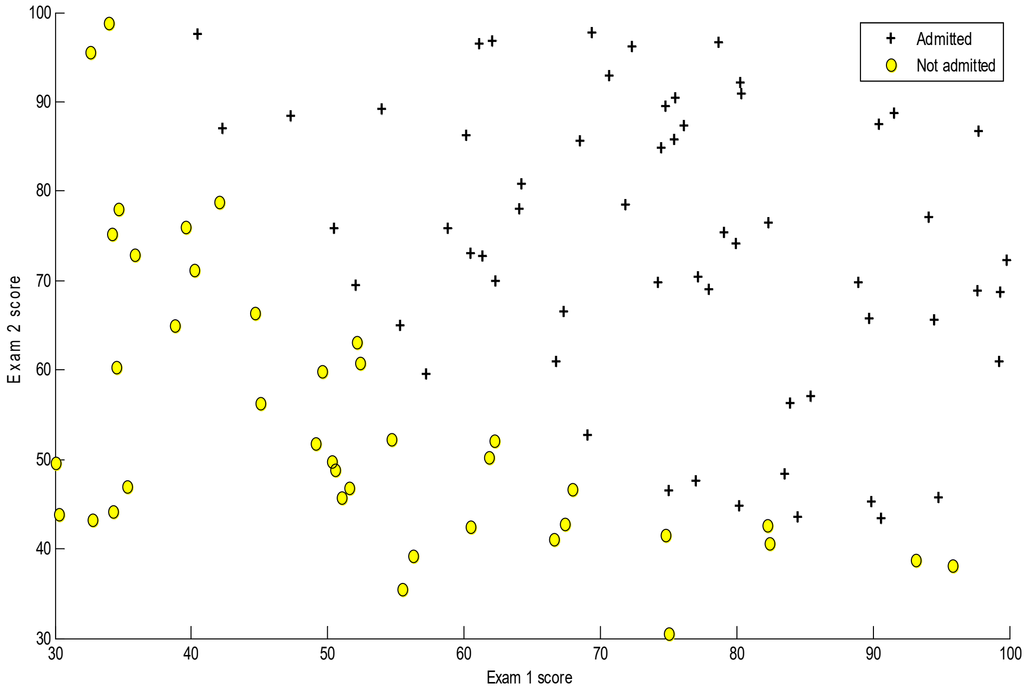


Fig. 1 Training Data

Logistic regression - example

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0 \Rightarrow \text{decision boundary}$$

$$\text{if} \quad z > 0 \Rightarrow g(z) > 0.5 \Rightarrow \text{predict class} = 1$$

$$g(z) = \frac{1}{1+e^{-z}}$$

$$\text{if} \quad z < 0 \Rightarrow g(z) < 0.5 \Rightarrow \text{predict class} = 0$$

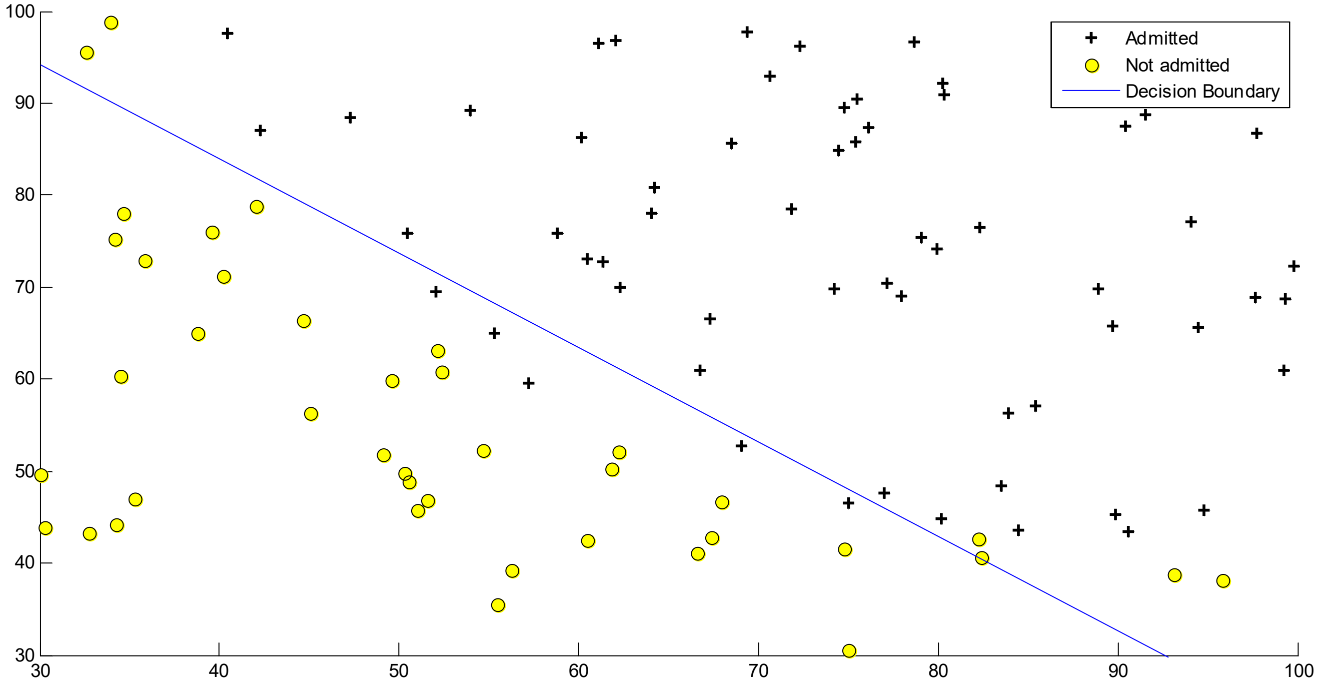
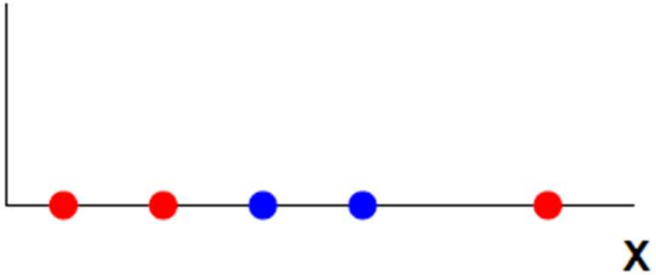


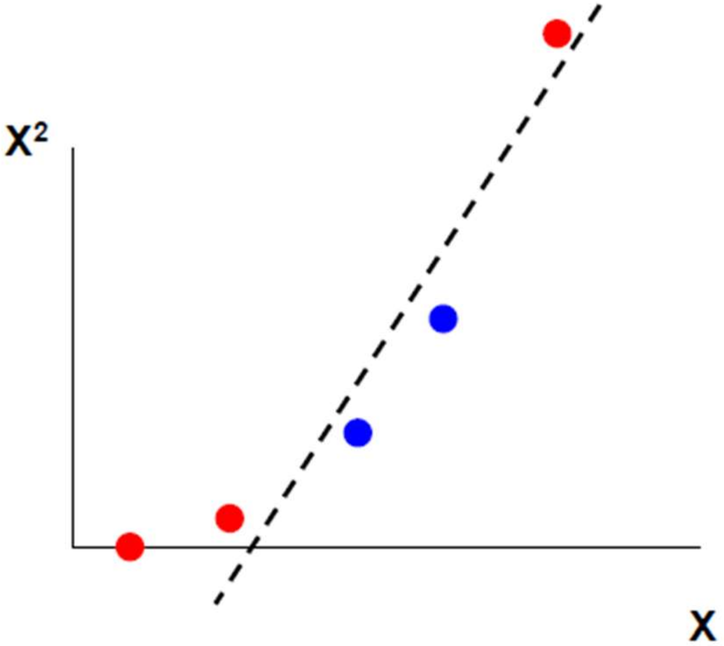
Fig. Training data and linear decision boundary with the optimized parameters (θ)

Nonlinearly Separable Data

Linear classifier cannot classify these examples.



And now ?



$$z = \theta^T x = \theta_0 + \theta_1 x + \theta_2 x^2 = 0 =>$$

Nonlinear decision boundary (in the original feature space x)

Linear decision boundary (in the extended feature space x, x^2)

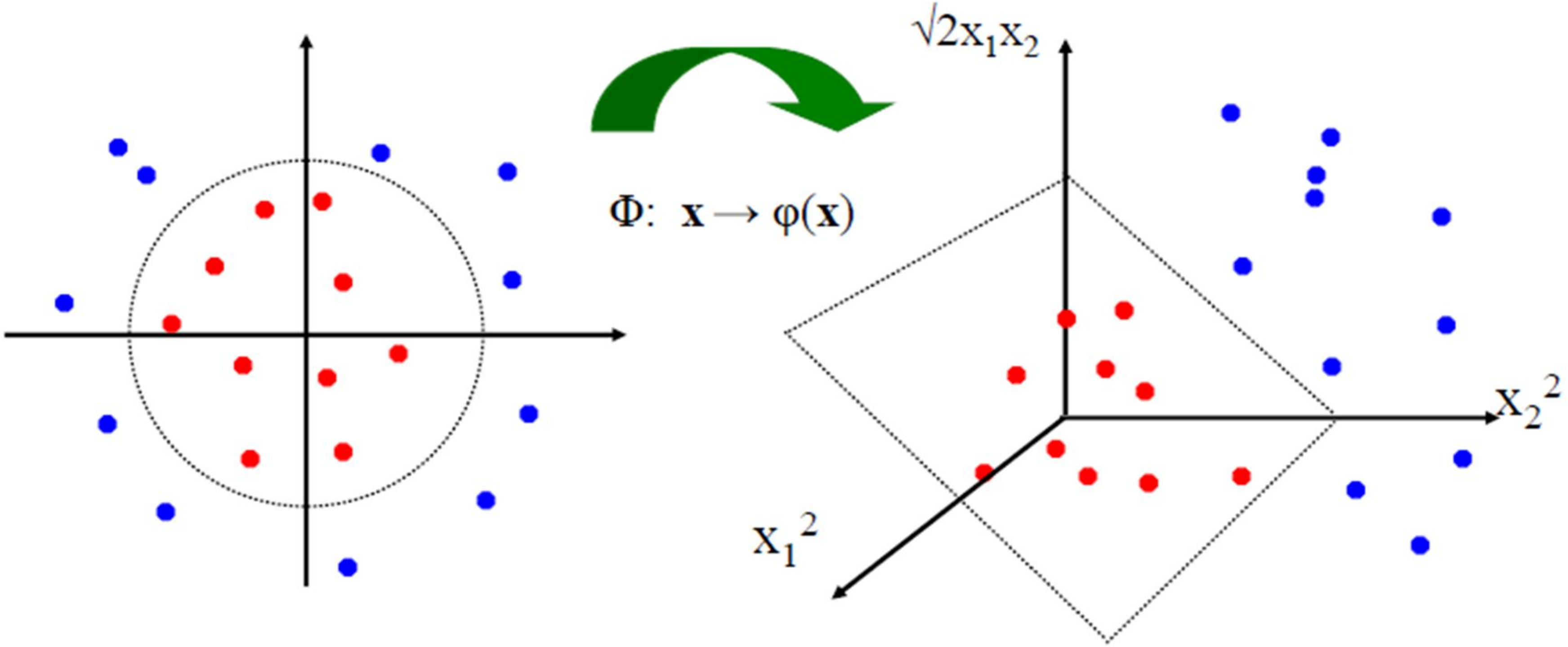
$$if \quad z > 0 => g(z) > 0.5 => \text{predict class} = 1$$

$$if \quad z < 0 => g(z) < 0.5 => \text{predict class} = 0$$

Nonlinearly Separable Data

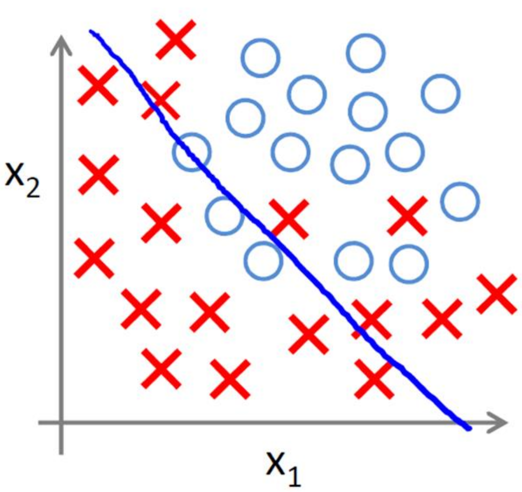
- The original input space (x) can be mapped to some higher-dimensional feature space ($\varphi(\mathbf{x})$)where the training set is separable:

$x=(x_1,x_2)$ $\varphi(\mathbf{x}) =(x_1{}^2,x_2{}^2,\sqrt{2}x_1x_{2)}$

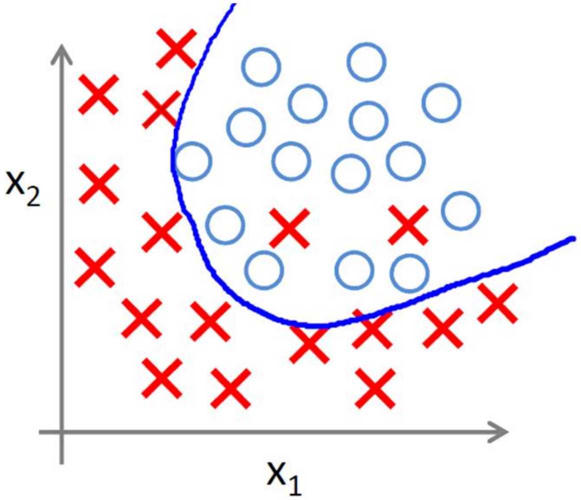


Overfitting problem

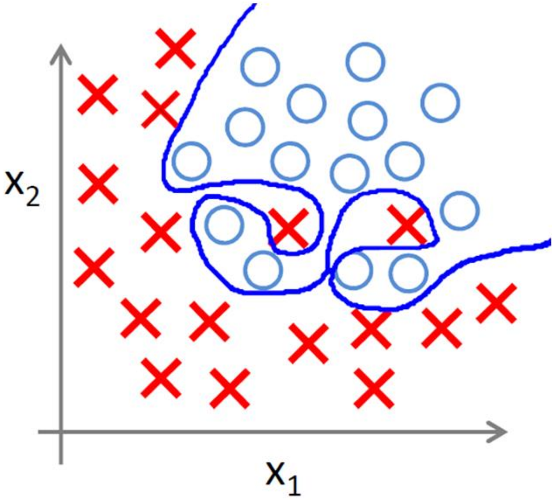
Overfitting: If we have too many features, the learned model may fit the training data very well but fail to generalize to new examples.



underfit- high bias model



ok model



overfit – high variance

Regularization

Regularization to prevent overfitting.

1 Ridge Regression

— Keep all the features, but reduces the magnitude of θ.

— Works well when each of the features contributes a bit to predict y.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

2 Lasso Regression

- May shrink some coefficients of θ to exactly zero.
- Serve as a feature selection tools (reduces the number of features).

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log\left(h_\theta\left(x^{(i)}\right)\right) - \left(1-y^{(i)}\right)\log\left(1-h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\left|\theta_j\right|$$

Regularized Logistic Regression

Unregularized Logit cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Regularized Logit cost function (ridge regression)

λ is the regularization parameter (hyper-parameter) that needs to be selected

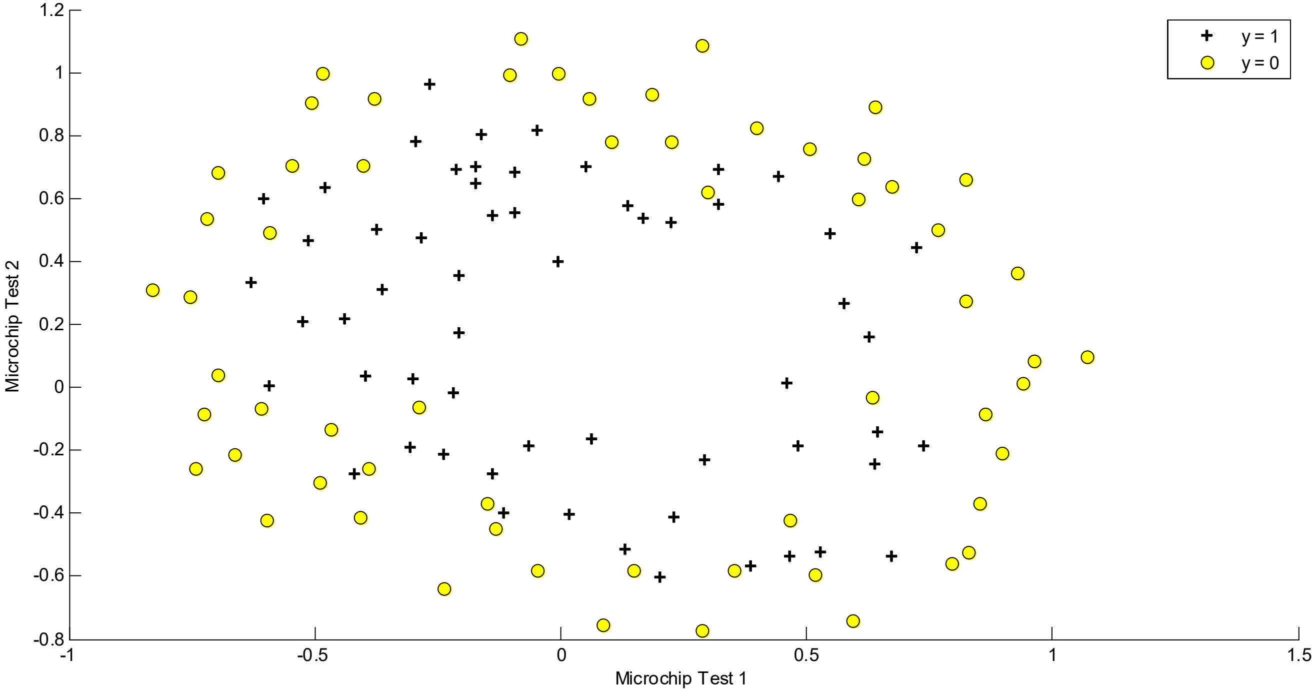
$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right] \boxed{+ \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2}$$

Small λ => lower bias, higher variance

High λ => higher bias, lower variance

Regularized Log Reg -example

Predict whether microchips from a fabrication plant passes quality assurance (QA). During QA, each microchip goes through various tests to ensure it is functioning correctly. Suppose we have the test results for some microchips on two different tests. From these two tests, we would like to determine whether the microchips should be accepted (y=1) or rejected (y=0).



Regularized Log Reg -example

Dataset is not linearly separable => logistic regression will only be able to find a linear decision boundary. One way to fit the data better is to create more features. For example add polynomial terms of x1 and x2 .

$$\text{mapFeature}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_1 x_2 \\ x_2^2 \\ x_1^3 \\ \vdots \\ x_1 x_2^5 \\ x_2^6 \end{bmatrix}$$

NONLINEAR decision boundary $\Rightarrow$

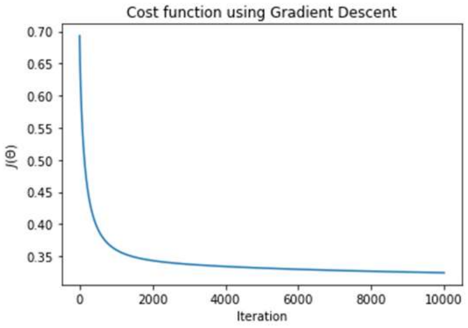
$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_1 x_2 + \theta_3 x_1^2 + \theta_4 x_1 x_2 + \ldots\ldots.\theta_{28} x_2^6 = 0$$

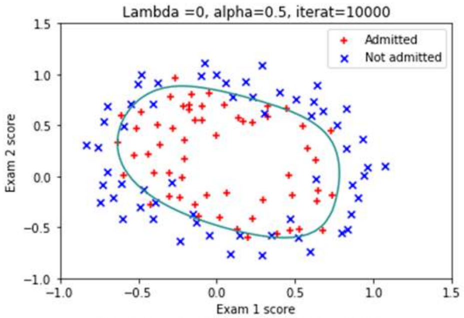
$$if \quad z > 0 => g(z) > 0.5 => \text{predict class} = 1$$

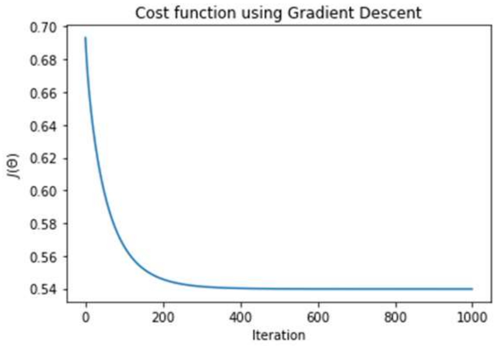
$$if \quad z < 0 => g(z) < 0.5 => \text{predict class} = 0$$

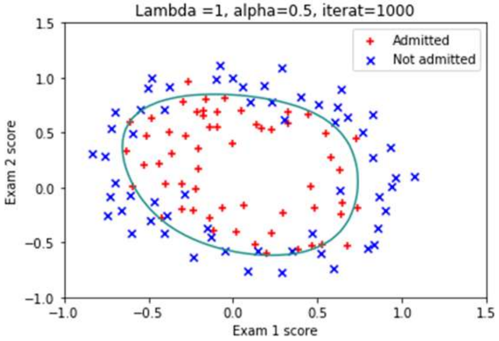
Regularized Log Reg -example

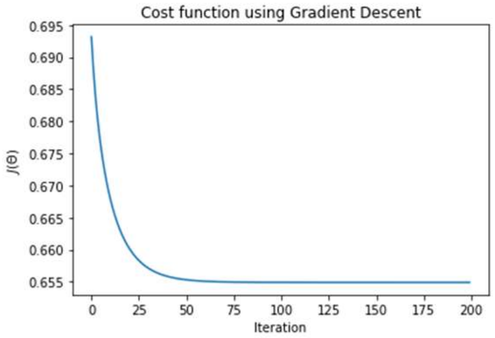
Accuracy on training data: :84.75% (λ =0) | 83.90 % (λ =1) | 71.2 % (λ =10))

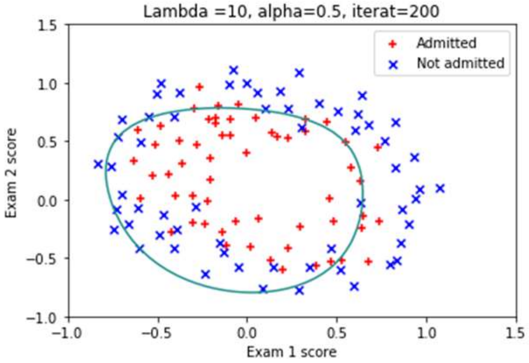








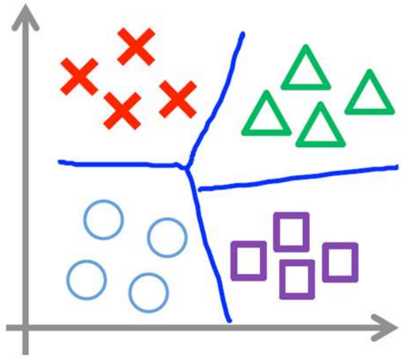




Multiclass Classification

Exs. *Email division* (work, friends, family, hobby)

Medical diagnosis (not ill, cold, flu) ; *Weather* (sunny, cloudy, rain, snow)



One-versus-all strategy :

For K classes train K binary classifiers:

```
for c=1:K
    Make y_binary=1 (only for examples of class c)
         y_binary=0 (for examples of all other classes)
    theta=Train classifier with training data X and output y_binary.
    Save the learned parameters of all classifiers in one matrix
    where each raw is the learned paramenters of one classifier:
   theta_all(c,:)=theta
end
```

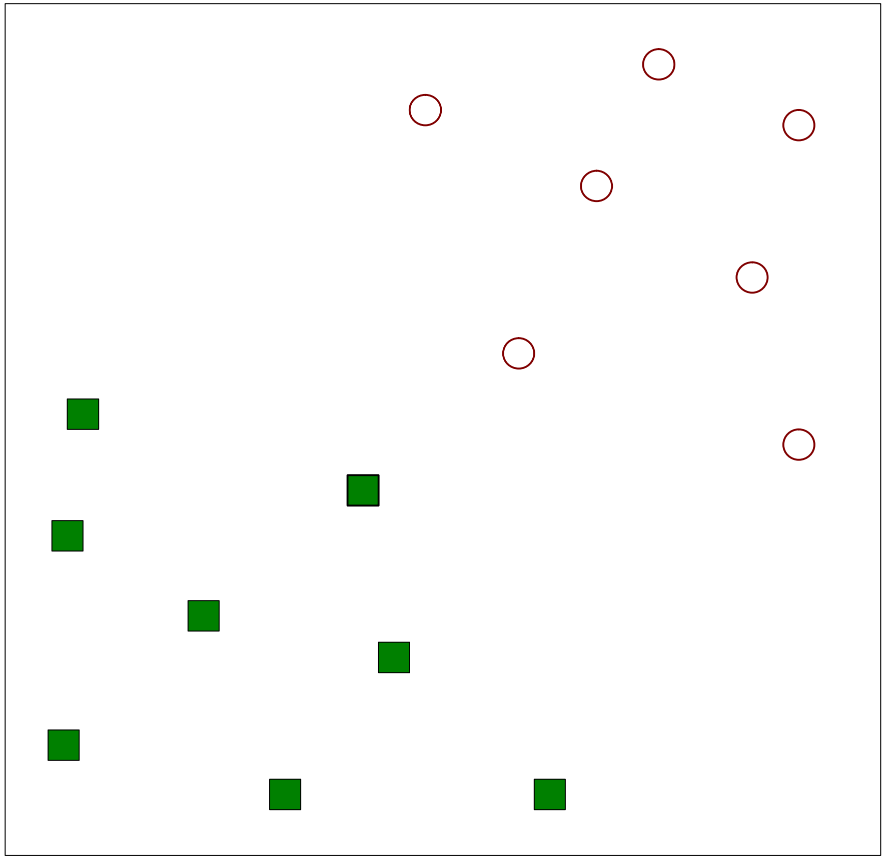
New example: winner-takes-all strategy, the binary classifier with the highest output score assigns the class.

Classification –

SUPPORT VECTOR MACHINES (SVM)

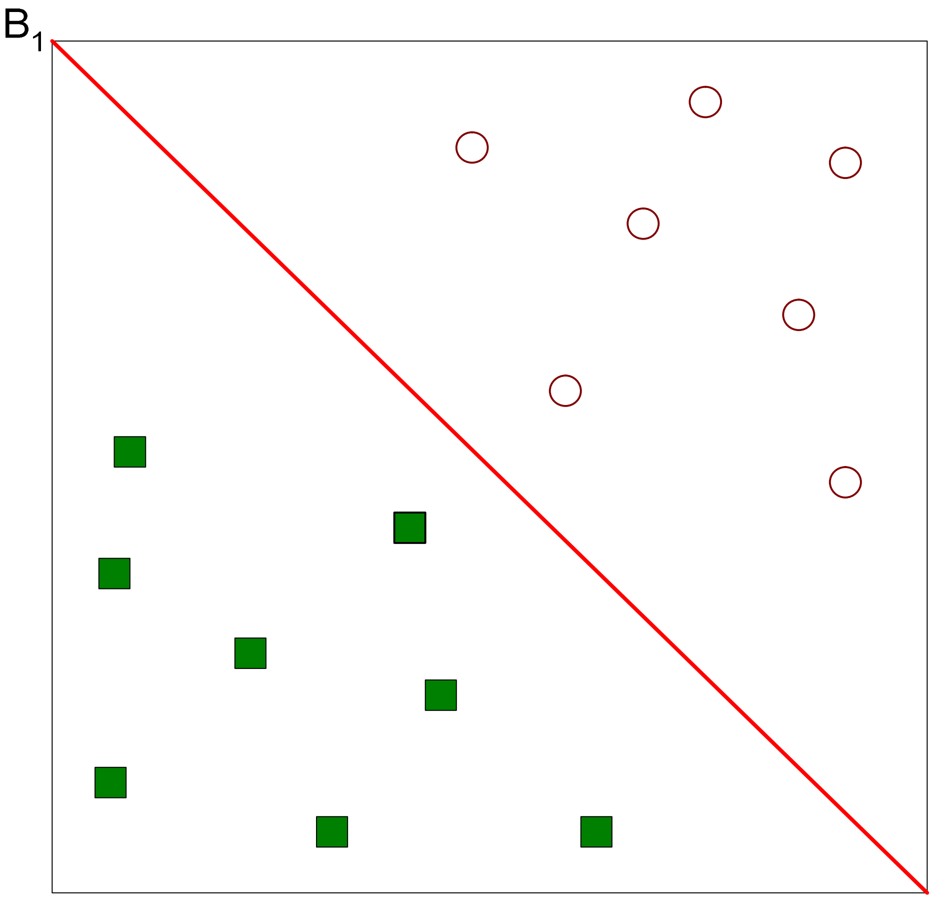
Proposed by Vladimir N. Vapnik and Alexey Chervonenkis, 1963

Linearly separable classes



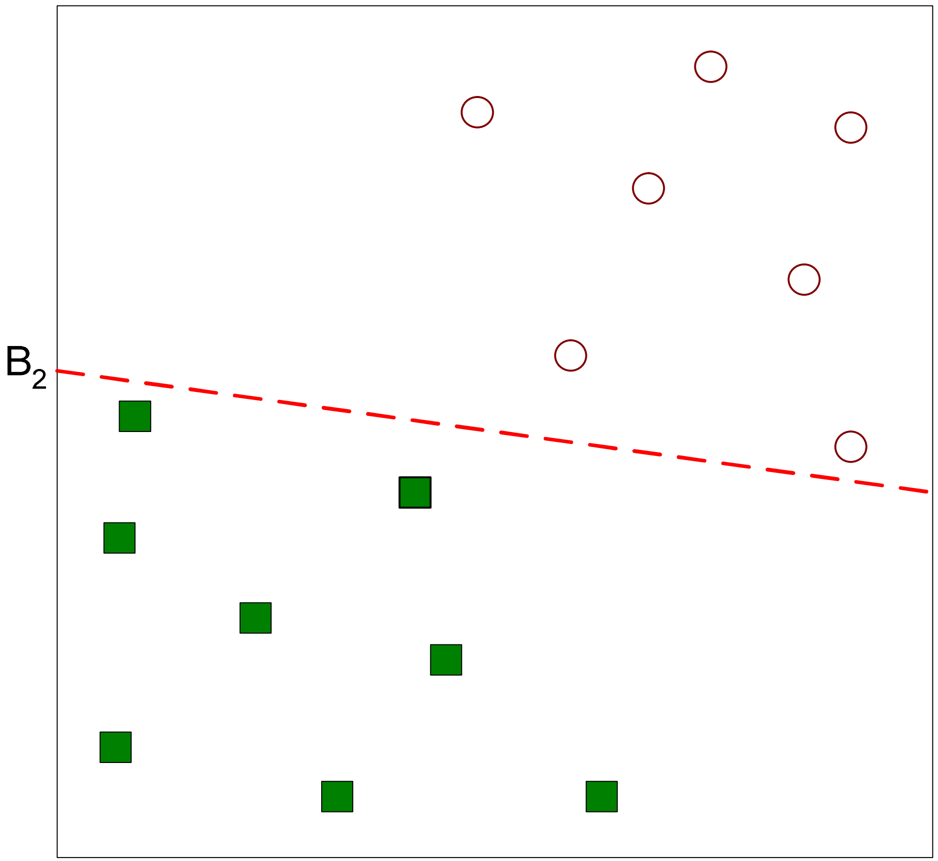
Find a decision boundary to separate data

Linearly separable classes



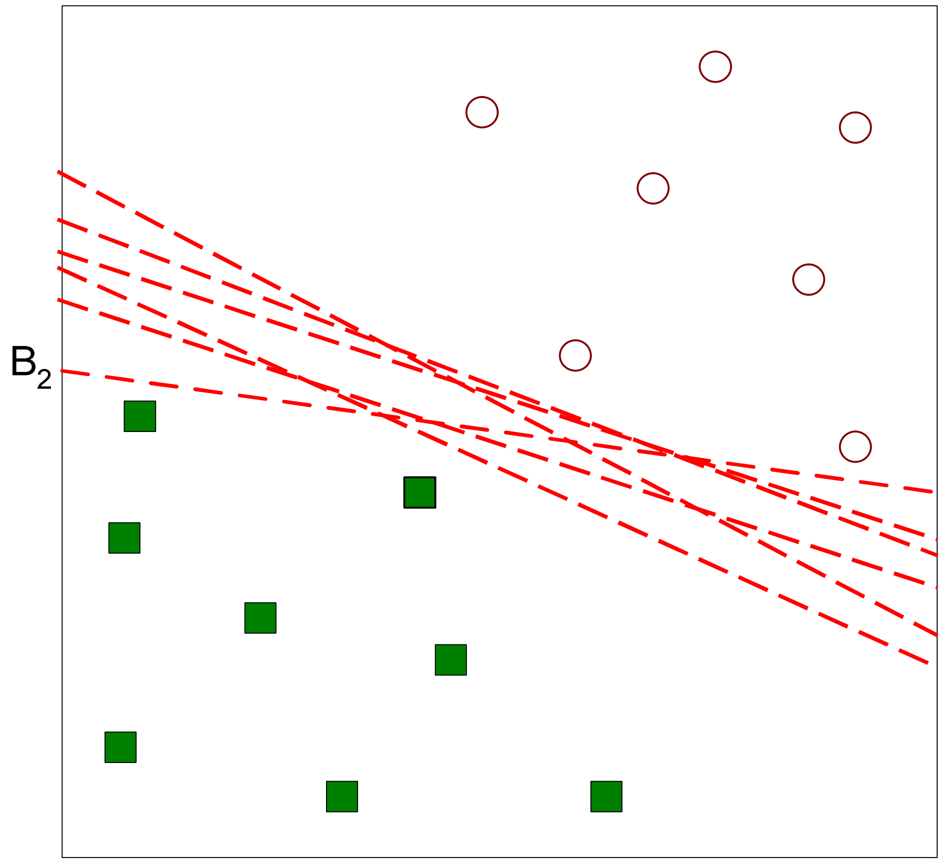
One Possible Solution

Linearly separable classes



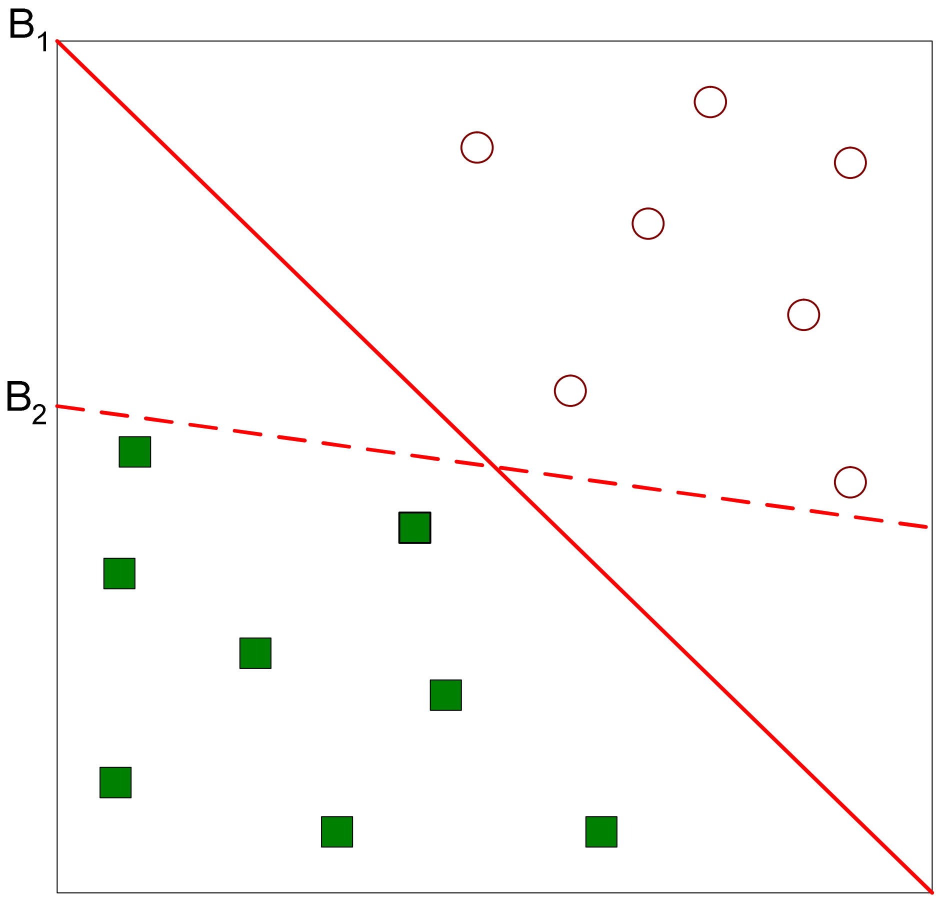
Another possible solution

Linearly separable classes



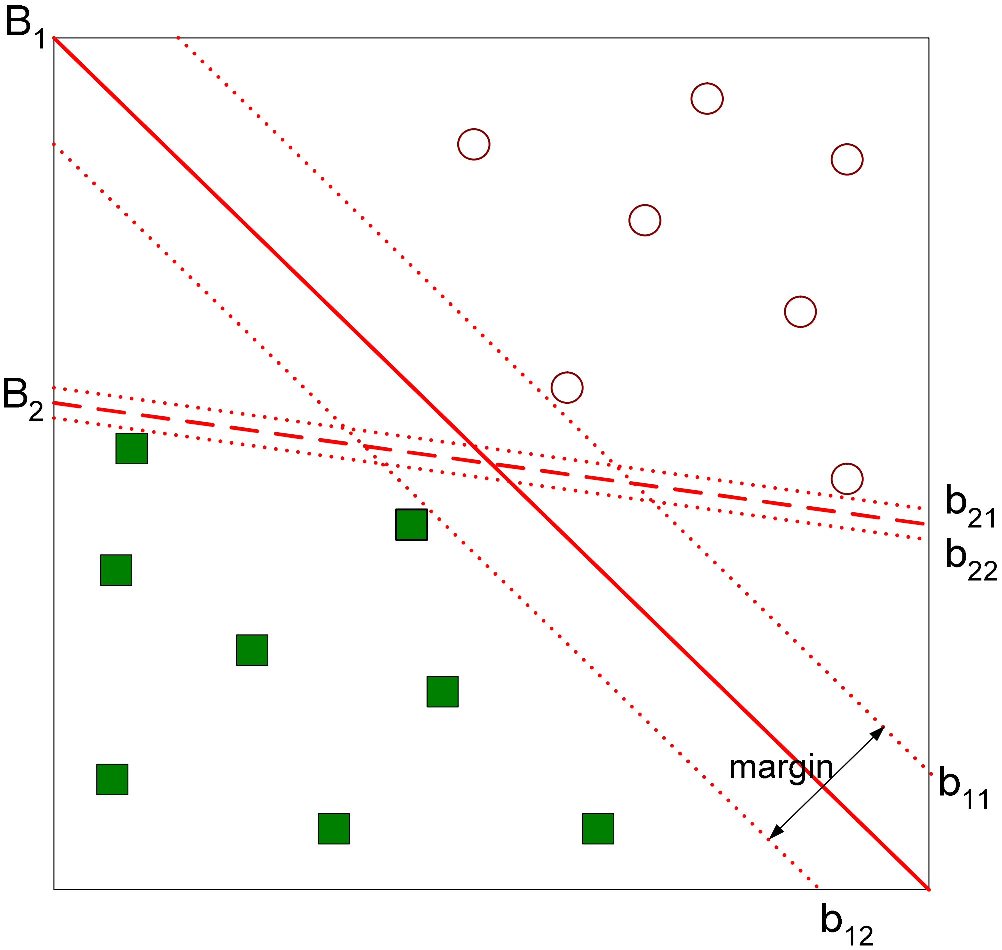
Many possible solutions

Linearly separable classes



Which one is better? B1 or B2?

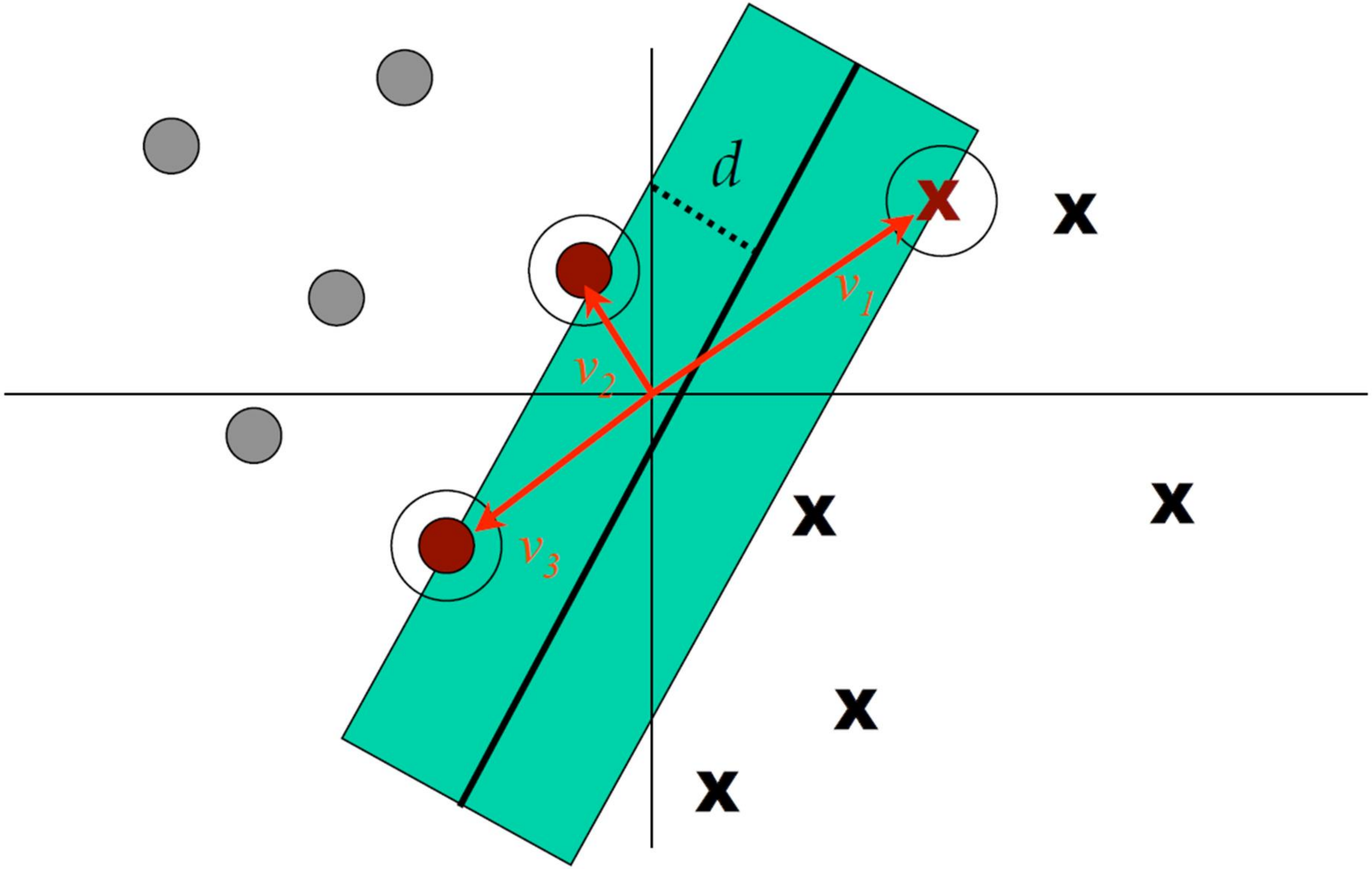
SVM - Large margin classifier



Find a boundary that maximizes the margin => B1 is better than B2

SUPPORT VECTORS (v1,v2,v3)

Only the closest points (support vectors) from each class are used to decide which is the optimum (the largest) margin between the classes.

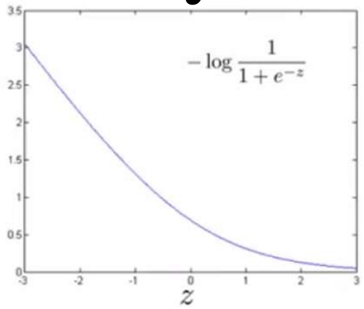


SVM cost function

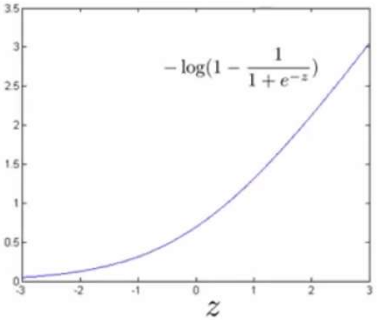
Regularized LogReg cost function:

$$\min_{\theta} \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)})))\right)\right]+\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

for y=1

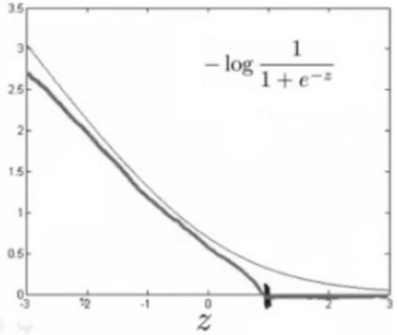


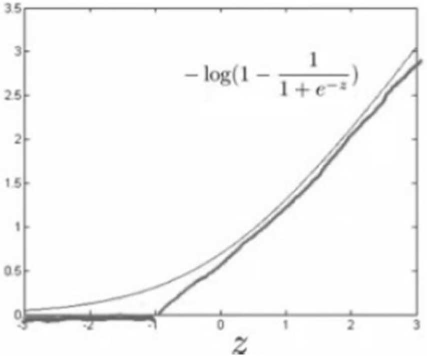
for y=0



Regularized SVM cost function (Modification of Logit cost function. ***cost0*** & ***cost1*** are assimptotic safety margins with computational advantages)

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)})+(1-y^{(i)})cost_0(\theta^T x^{(i)})\right]+\frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$





$$z=\theta^T x$$

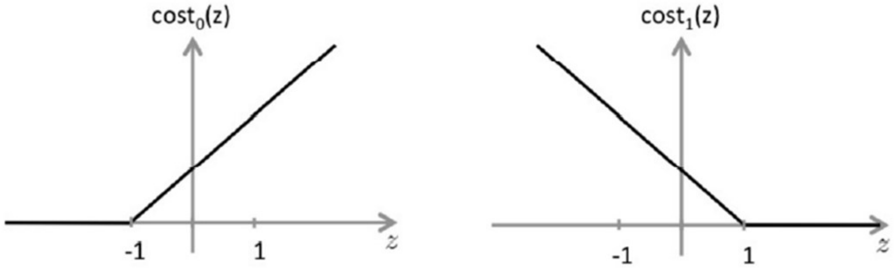
SVM cost function

Regularized LogReg cost function:

$$\min_{\theta} \frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)}\left(-\log h_\theta(x^{(i)})\right) + (1-y^{(i)})\left((-\log(1-h_\theta(x^{(i)}))\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Regularized SVM cost function

$$\min_{\theta} C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$

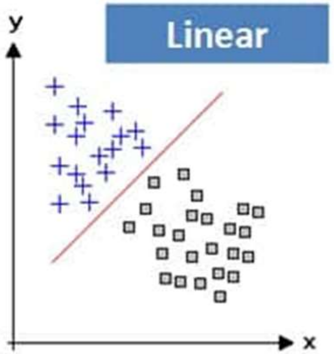


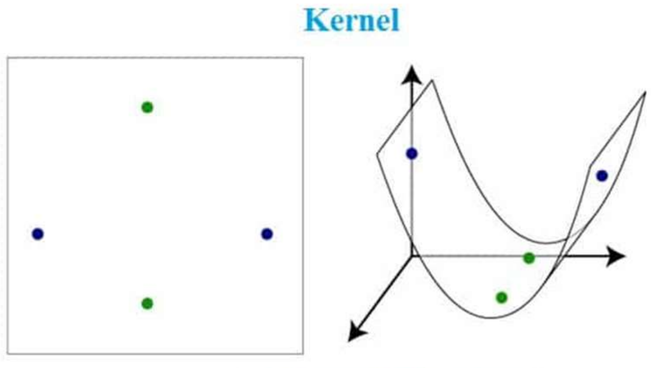
$z = \theta^T x$

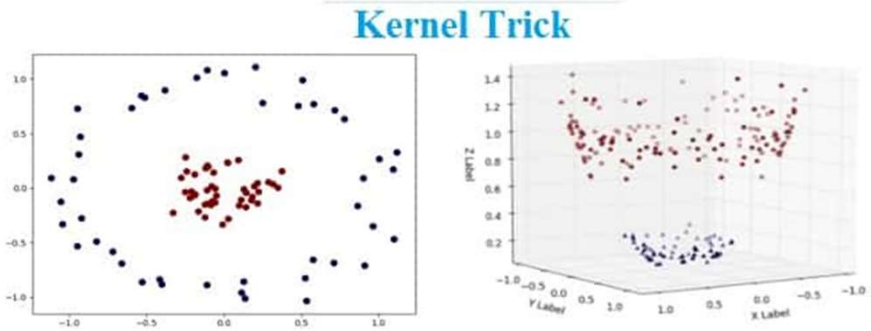
Different way of parameterization: C is equivalent to $1/\lambda$.

C >0 - parameter that controls the penalty for misclassified training examples.
Increase C more importance to training data fitting.
Decrease C – more importance to generalization properties (combat overfitting).

Nonlinearly separable data – kernel SVM







Kernel: function which maps a lower-dimensional data into higher dimensional data.

Tipical Kernels:

- Polynomial Kernel - adding extra polynomial terms
- Gaussian Radial Basis Function (RBF) kernel – <u>the most used kernel</u>
- Laplace RBF kernel
- Hyperbolic tangent kernel
- Sigmoid kernel, etc.

Nonlinear SVM – Gaussian RBF Kernel

$$k(x_i, x_j) = e^{\left(-\gamma \left\| x^{(i)} - x^{(j)} \right\|^2\right)}, \quad \gamma > 0, \gamma = 1/2\sigma^2, \quad \sigma - \text{variance}$$

The RBF kernel is a metric of similarity between examples, $x^{(i)}$ and $x^{(j)}$
Substitute the original features with similarity features (kernels).

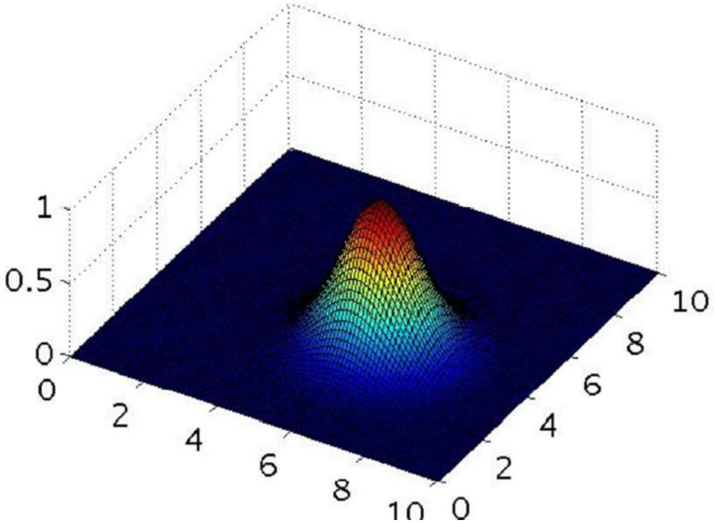
Note: the original (n+1 dimensional) feature vector is substituted by the new (m+1 dimensional) similarity feature vector.

m –number of examples, **m>>n !!!**

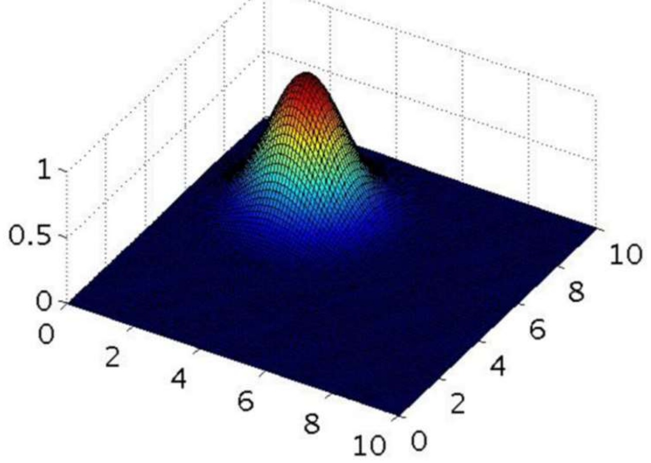
Gaussian RBF Kernel – Parameter σ

$$k(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}, \quad \gamma = \frac{1}{2\sigma^2} > 0$$

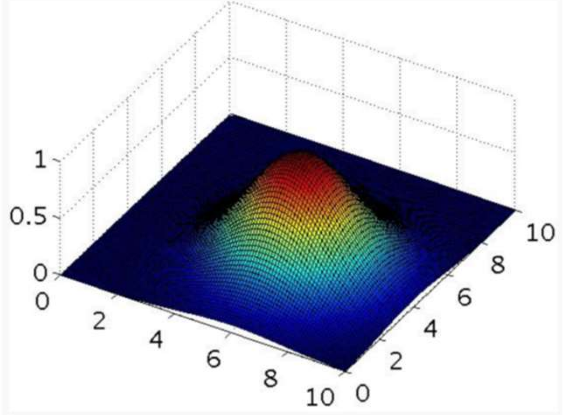
σ=1



σ=0.5



σ=1.5



σ determines how fast the similarity metric decreases to 0 as the examples go away of each other.

Large σ : kernels vary more smoothly (combat overfitting)

Small σ : kernels vary less smoothly (more importance to training data fitting)

SVM parameters

How to **choose hyper-parameter C:**

Large C: lower bias, high variance (equivalent to small regular. param. λ)

Small C: higher bias, lower variance (equivalent to large regular. param. λ)

How to **choose hyper-parameter** σ**:**

Large σ**:** features vary more smoothly. Higher bias, lower variance

Small σ**:** features vary less smoothly. Lower bias, higher variance

SVM implementation

Use SVM software packages to solve SVM optimization !!!

In Python, use Scikit-learn (sklearn) machine learning library and

Import SVC (Support Vector Classification):

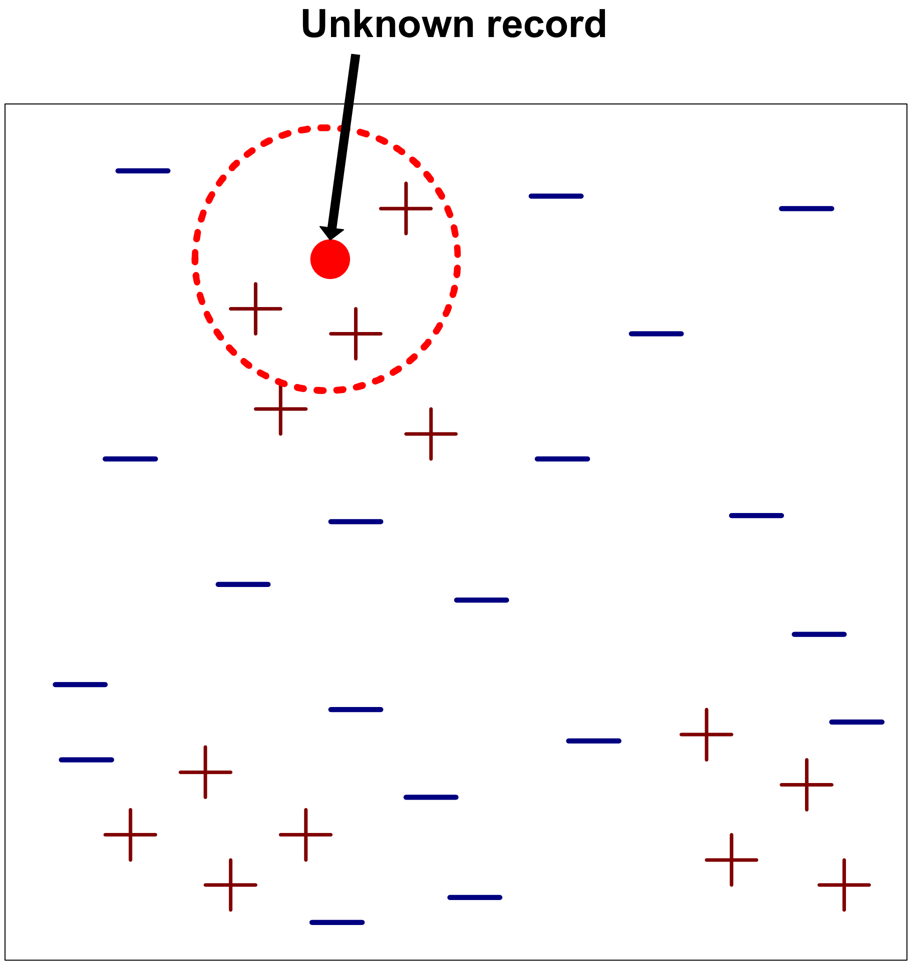
from sklearn.svm import SVC
classifier = SVC(kernel="rbf",gamma =?)

"rbf“ (Radial Basis Function) corresponds to the Gaussian kernel.
gamma = 1/σ.

Classification –

K- Nearest-Neighbor (k-NN)

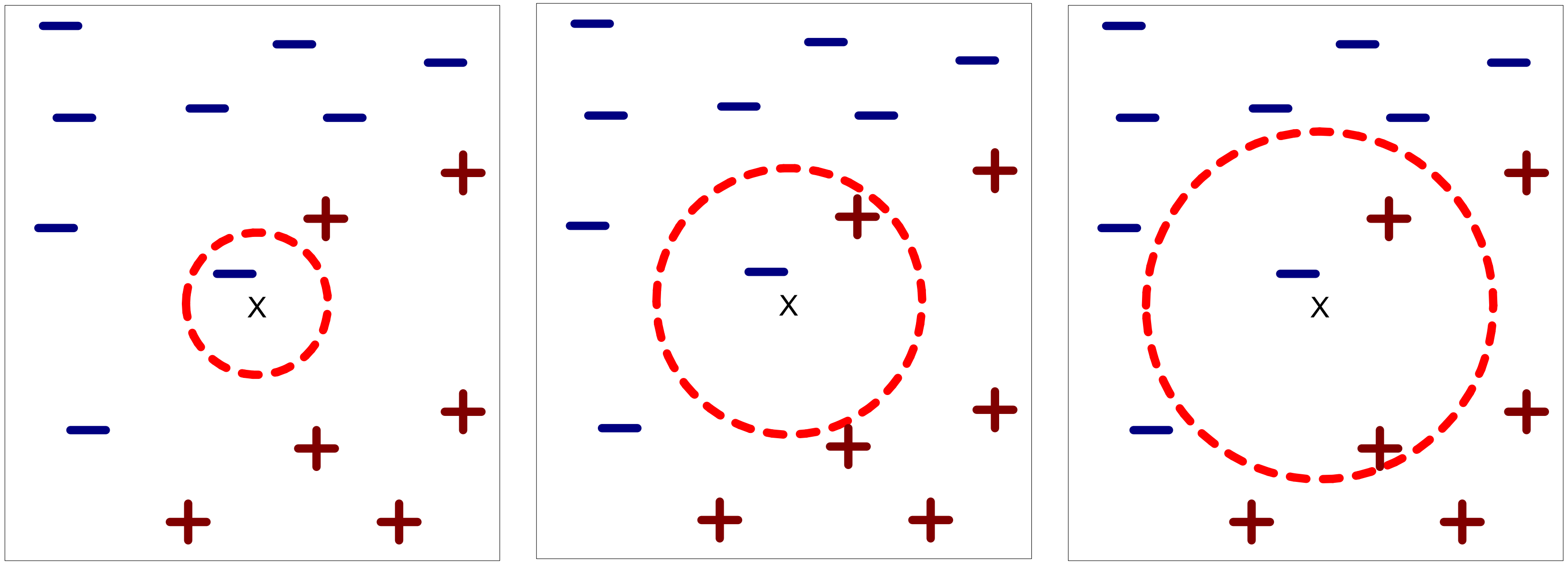
K- Nearest-Neighbor (k-NN) Classifier



- KNN requires:
  - Set of labeled records.
  - Measure to compute distance (similarity) between records.
  - *K* is the number of nearest neighbors (the closest points).

- To classify a new (unlabeled) record:
  - Compute its distance to all labeled records.
  - Identify *k* nearest neighbors.
  - The class label of the new record is the label of the majority of the nearest neighbors.

K-NN- choice of k



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K- Nearest Neighbors of the new point x are the points that have the smallest distance to x

Lab work - Spam Classification

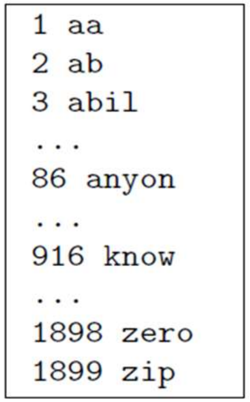
- Labelled data set : SpamAssassin Public Corpus

- Convert the email into a binary feature vector:

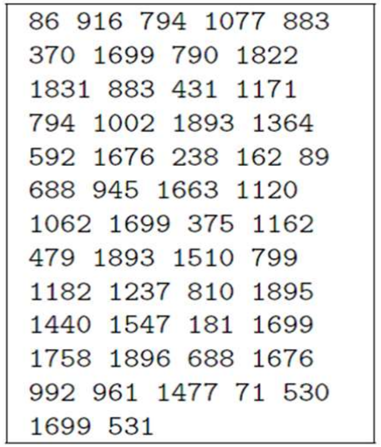
- Clean (remove slash , dots, coms)
- Tokenize (parse) into words
- Count the word frequency
- Create dictionary with most frequent words (e.g. 10000 to 50000 words) - **Feature space.**
- Substitute the words with the dictionary indices.
- Extract binary features from emails - binary (sparse) feature for an email corresponds to whether the i-th word in the dictionary occurs in the email.

- **Apply classifier** (e.g. SVM)

Dictionary => Email with dictionary indices => binary features



```
1 aa
2 ab
3 abil
...
86 anyon
...
916 know
...
1898 zero
1899 zip
```



```
86 916 794 1077 883
370 1699 790 1822
1831 883 431 1171
794 1002 1893 1364
592 1676 238 162 89
688 945 1663 1120
1062 1699 375 1162
479 1893 1510 799
1182 1237 810 1895
1440 1547 181 1699
1758 1896 688 1676
992 961 1477 71 530
1699 531
```

$$x = \begin{bmatrix} 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^n$$

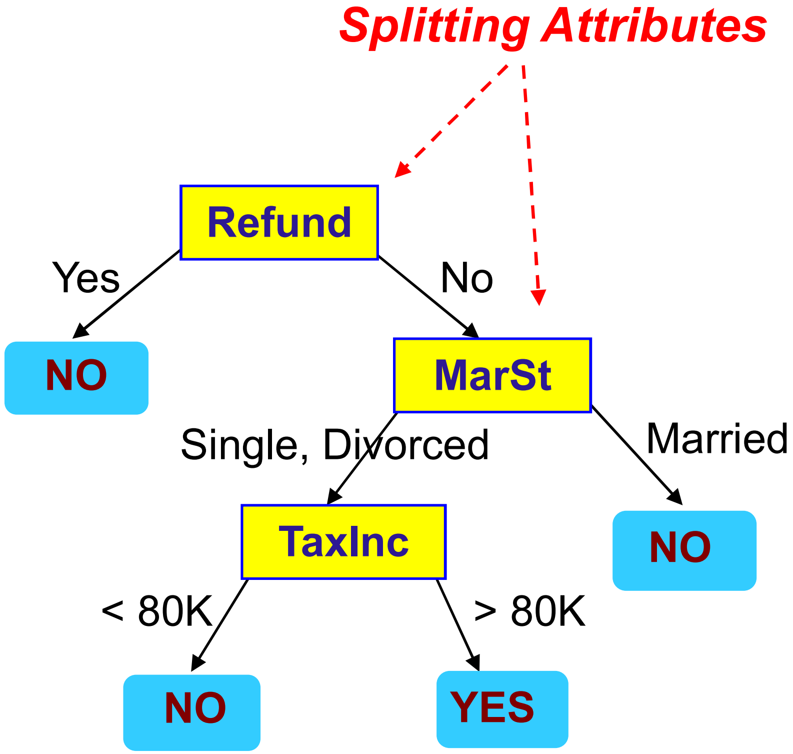
Classification –

Decision Trees

Classification by Decision Tree – model 1

	categorical	categorical	continuous	class
Tid	**Refund**	**Marital Status**	**Taxable Income**	**Cheat**
1	Yes	Single	125K	**No**
2	No	Married	100K	**No**
3	No	Single	70K	**No**
4	Yes	Married	120K	**No**
5	No	Divorced	95K	**Yes**
6	No	Married	60K	**No**
7	Yes	Divorced	220K	**No**
8	No	Single	85K	**Yes**
9	No	Married	75K	**No**
10	No	Single	90K	**Yes**

Training Data

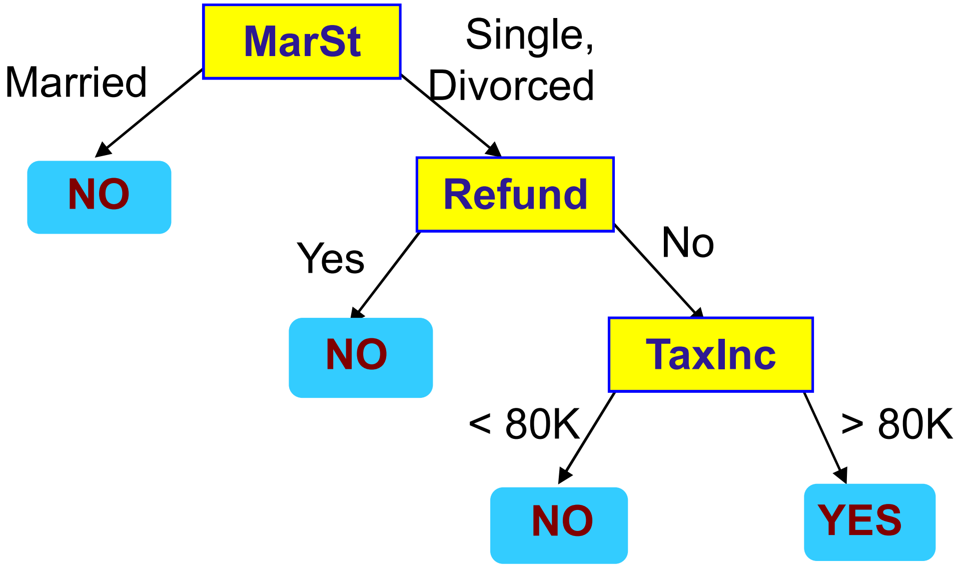


Model: Decision Tree

Classification by Decision Tree – model 2

	categorical	categorical	continuous	class
Tid	Refund	Marital Status	Taxable Income	Cheat
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Training Data



There could be more than one tree that fits the same data!

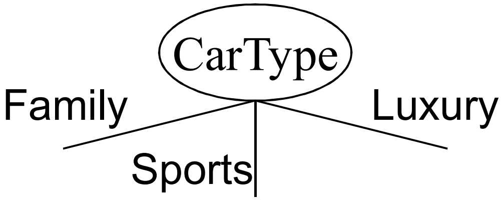
Decision Tree

Decision Tree has 2 basic type of nodes:

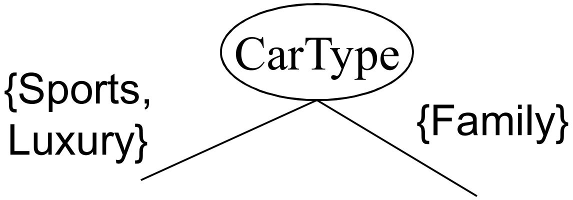
- **Leaf node** - a class label, determined by majority vote of training examples reaching that leaf.
- **Node** is a question on one feature. It branches out according to the answers.
    - **root node**: the first (top) node of the tree
    - **child node**: the next node to a current node

Splitting of Categorical Features

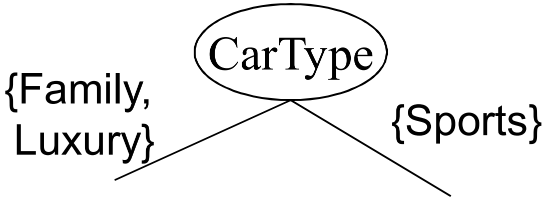
- Multi-way split: Use as many partitions as distinct values.



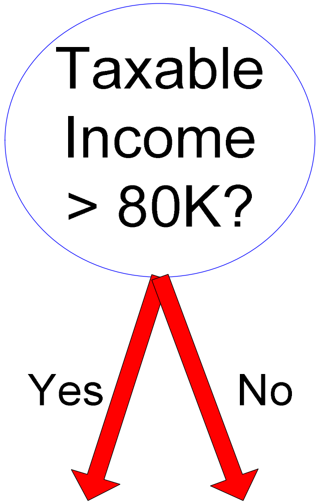
- Binary split: Divides values into two subsets.



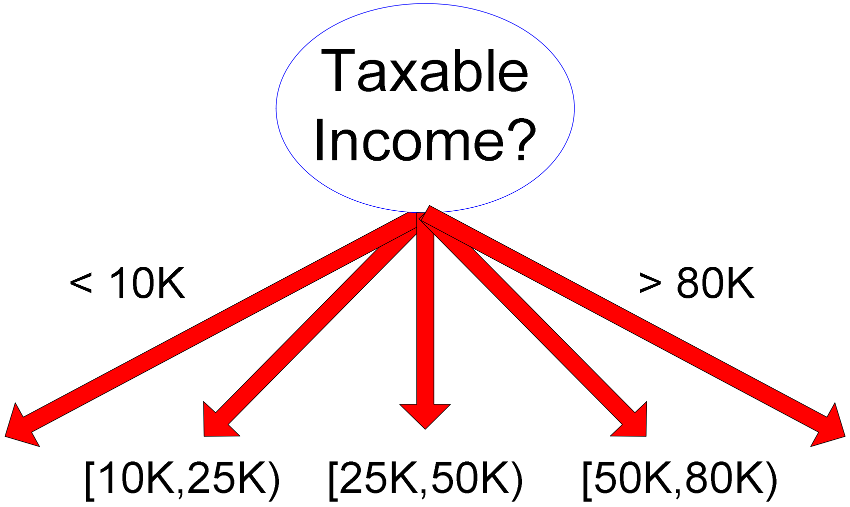
OR



Splitting of Continuous Features



(i) Binary split



(ii) Multi-way split

How to determine the best split ?

We want:

- To get the smallest tree
- Pure leaf nodes, i.e. all examples having (almost) the same class (ex. C0 or C1).
- Choose the feature that produces the "purest" (the most homogeneous) nodes
- We need a measure of node impurity (homogeneity).

C0: 5 C1: 5	C0: 9 C1: 1
Non-homogeneous, **High degree of impurity**	**Homogeneous,** **Low degree of impurity**

Measures of Node Impurity

- **Gini Index** at a given node:

$$\boxed{GINI(node) = 1 - \sum_{Class_j} [p(Class_j \mid node)]^2}$$

- **Classification error** at a given node:

$$\boxed{Error(node) = 1 - \max_{Class_j} p(Class_j \mid node)}$$

- **Entropy (H)** at a given node:

$$\boxed{H(node) = -\sum_{Class_j} p(Class_j \mid node) \log p(Class_j \mid node)}$$

p (*Class_j* / *node*): probability of *Class_j* at a given *node*

Computing GINI - example

$$GINI(node) = 1 - \sum_{Class_j} [p(Class_j \mid node)]^2$$

(*p* - probability)

C1	**0**
C2	**6**

p(C1) = 0/6 = 0   p(C2) = 6/6 = 1

Gini = 1 - $p(C1)^2$ - $p(C2)^2$ = 1 - 0 - 1 = 0

C1	**1**
C2	**5**

p(C1) = 1/6   p(C2) = 5/6

Gini = 1 – $(1/6)^2$ – $(5/6)^2$ = 0.278

C1	**2**
C2	**4**

p(C1) = 2/6   p(C2) = 4/6

Gini = 1 – $(2/6)^2$ – $(4/6)^2$ = 0.444

Computing Classification Error - example

$$\boxed{Error(node) = 1 - \max_{Class_j} p(Class_j \mid node)}$$

C1	**0**
C2	**6**

p(C1) = 0/6 = 0   p(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

p(C1) = 1/6   p(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

C1	**2**
C2	**4**

p(C1) = 2/6   p(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

Entropy

p(head)=0.5
p(tail)=0.5
H=1

p(head)=0.51
p(tail)=0.49
H=0.9997

- Entropy is a probabilistic measure of information uncertainty. In DTree we use it to measure the purity of a node.

- The node is pure if it has only examples that belongs to one class => entropy H = 0, no uncertainty (this is what we want !)

- If the node has equal number of examples of all classes => entropy reaches maximum H =1, the result is very uncertain.

Computing Entropy - example

$$H(node) = -\sum_{Class_j} p(Class_j \mid node)\log p(Class_j \mid node)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0   P(C2) = 6/6 = 1

H = – 0 log(0) – 1 log(1) = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6   P(C2) = 5/6

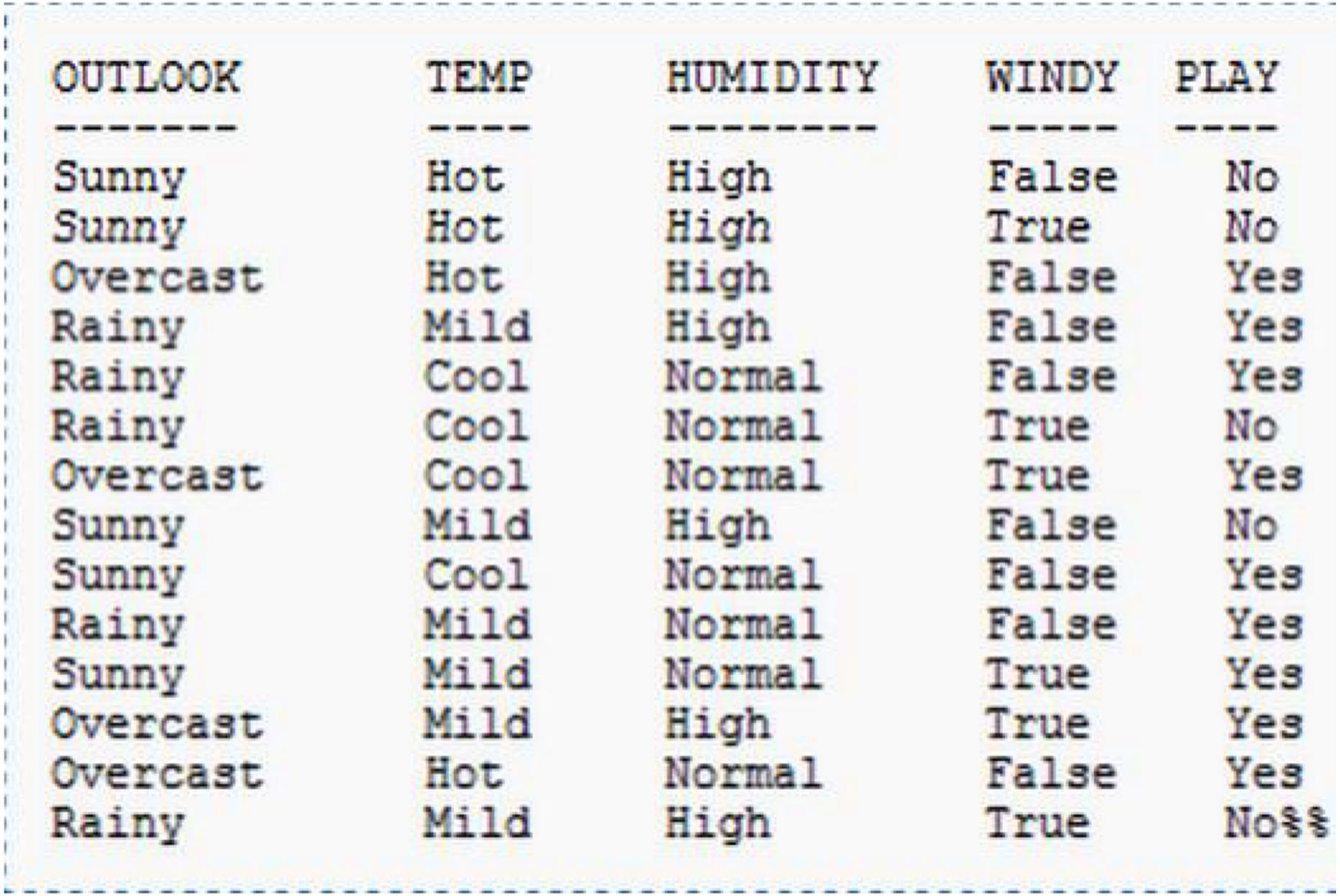
H = – (1/6) $\log_2$(1/6) – (5/6) $\log_2$(1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6   P(C2) = 4/6

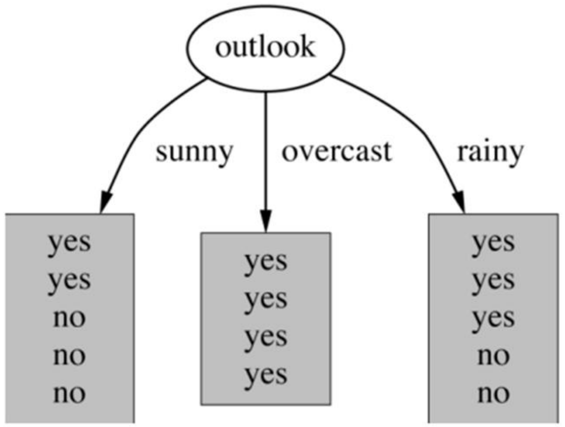
H = – (2/6) $\log_2$(2/6) – (4/6) $\log_2$(4/6) = 0.92

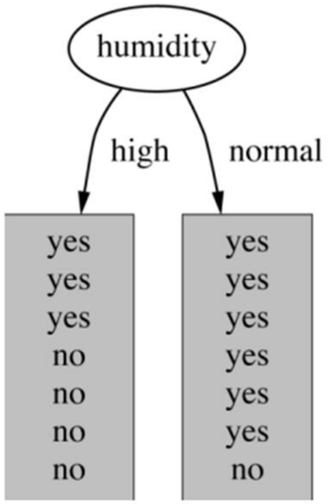
Example – Weather data (Play golf or Not)

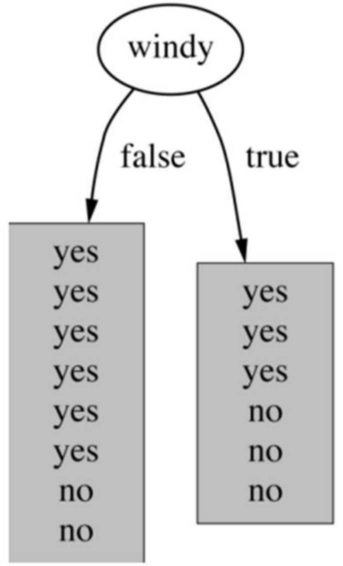


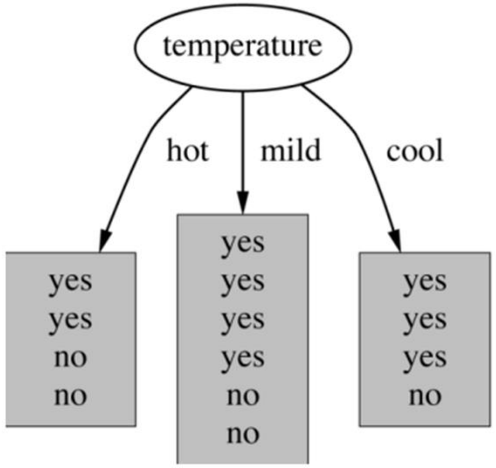
```
OUTLOOK        TEMP      HUMIDITY      WINDY  PLAY
--------       ----      ---------     -----  ----
Sunny          Hot       High          False    No
Sunny          Hot       High          True     No
Overcast       Hot       High          False    Yes
Rainy          Mild      High          False    Yes
Rainy          Cool      Normal        False    Yes
Rainy          Cool      Normal        True     No
Overcast       Cool      Normal        True     Yes
Sunny          Mild      High          False    No
Sunny          Cool      Normal        False    Yes
Rainy          Mild      Normal        False    Yes
Sunny          Mild      Normal        True     Yes
Overcast       Mild      High          True     Yes
Overcast       Hot       Normal        False    Yes
Rainy          Mild      High          True     No%%
```

Which feature to select as root node?





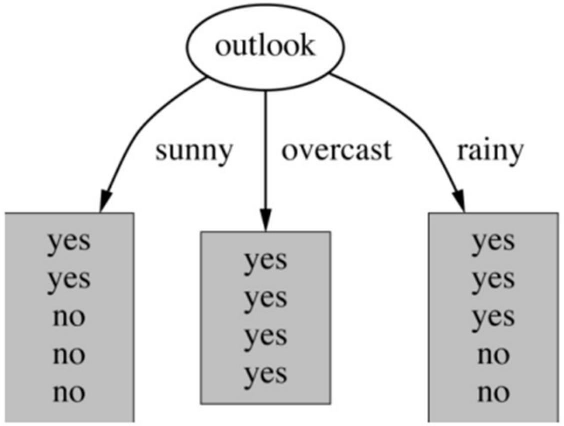




Information Gain

Information Gain = Entropy (H) before split - Entropy (H) after split at one node (one feature). How much uncertainty was reduced after splitting on a given feature.

$$H(node) = -\sum_{Class_j} p(Class_j \mid node) \log p(Class_j \mid node)$$



Feature OUTLOOK:

H(before split)=-[(9/14)*log$_2$ (9/14)+(5/14)*log$_2$ (5/14)] = 0.9403
(14 example: 9 ex. class Yes ; 5 ex. class No)

H(Sunny)=-[(2/5)*log$_2$ (2/5)+(3/5)*log$_2$ (3/5)]
(5 ex. with Outlook=Sunny: 2 ex. class Yes; 3 ex. class No)

H(Overcast)=-[4/4*log$_2$ (4/4)]
(4 ex. with Outlook=Overcast - 4 ex. class Yes ; 0 ex. class No)

H(Rainy)=-((2/5)*log$_2$ (2/5)+(3/5)*log$_2$ (3/5))
(5 ex. with Outlook=Rainy - 3 ex. class Yes; 2 ex. class No)

H(after split) =(5/14)*H(Sunny)+(4/14)*H(Overcast)+(5/14)*H(Rainy) =0.6935

Information Gain=H(before split) –H (after split) = 0.9403 - 0.6935=0.2467

Building a Decision Tree - example

Feature OUTLOOK: Information Gain = 0.2467

Feature TEMPERATURE : Information Gain = 0.029

Feature HUMIDITY: Information Gain = 0.157

Feature WINDY: Information Gain = 0.048

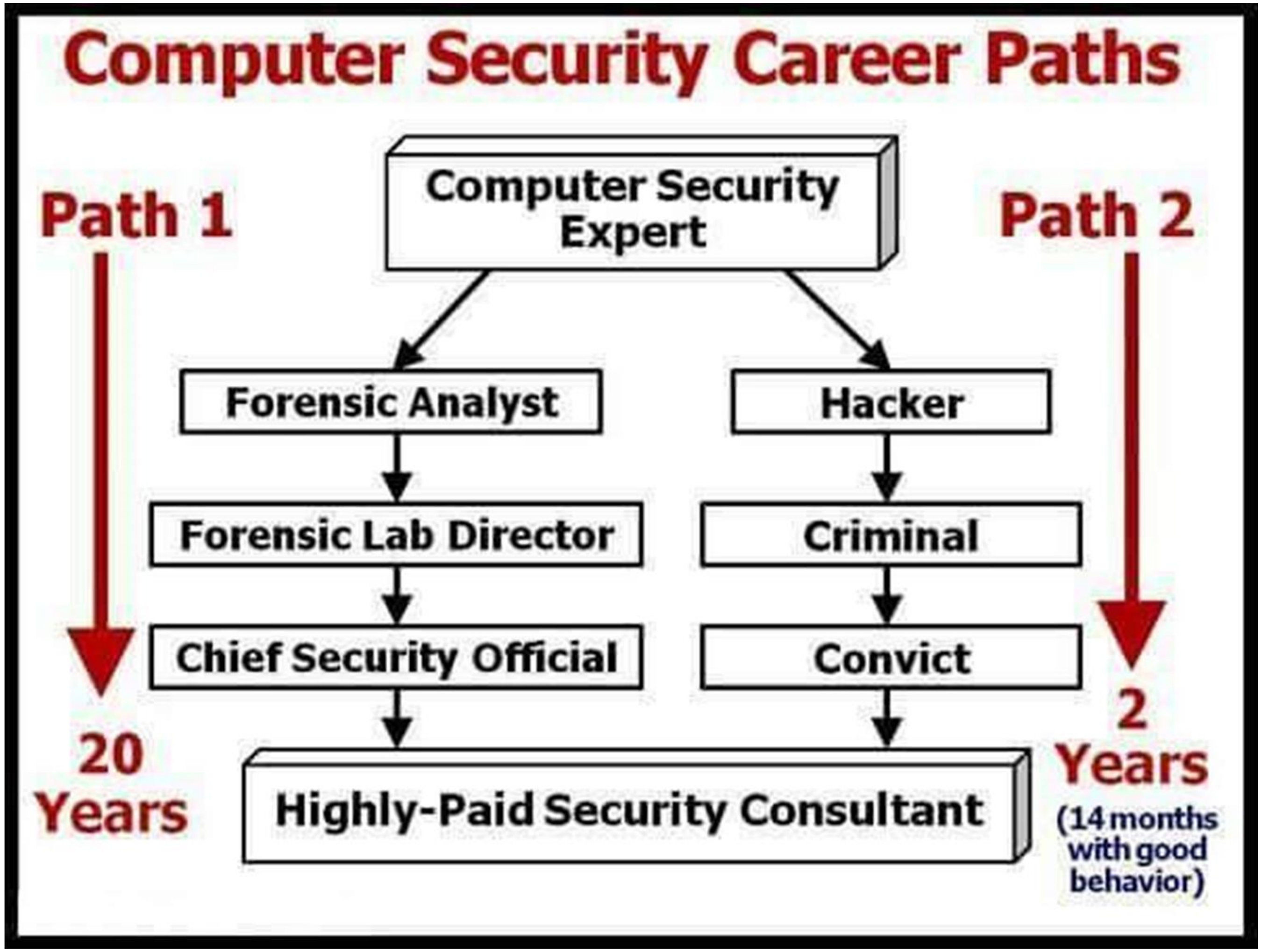
Choose feature that maximizes the information gain (minimizes the node impurity) !

Decision: Choose OUTLOOK as the root node.

This procedure is repeated for each node.

Splitting stops when data cannot be split any further or

the class is defined by the majority of elements of a subset.


Computer Security Career Paths
Path 1
Path 2
Computer Security Expert
Forensic Analyst
Forensic Lab Director
Chief Security Official
Hacker
Criminal
Convict
Highly-Paid Security Consultant
20 Years
2 Years
(14 months with good behavior)