Introduction to Machine Learning (BCS-41)

By
Sushil Kumar Saroj
Assistant Professor

Department of Computer Science & Engineering
Madan Mohan Malaviya University of Technology Gorakhpur
(UP State Govt. University)

Email: skscs@mmmut.ac.in

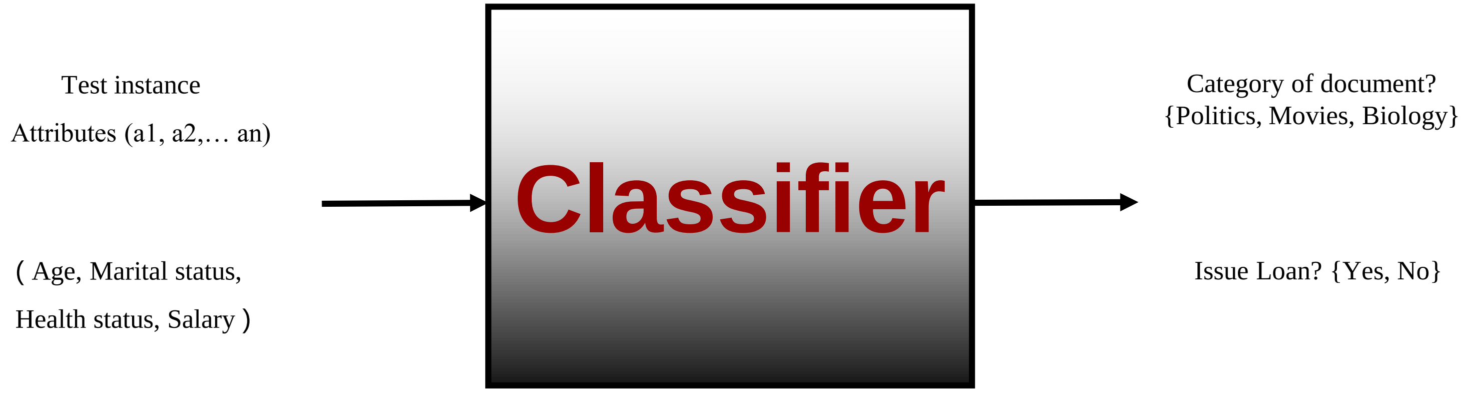
Syllabus

Unit-II

LINEAR MODELS- Linear Classification–Univariate Linear Regression–Multivariate Linear Regression–Regularized Regression– Logistic Regression–Perceptron–Multilayer Neural Networks –Learning Neural Networks Structures – Support Vector Machines–Soft Margin SVM–Going Beyond Linearity – Generalization and Over Fitting – Regularization–Validation

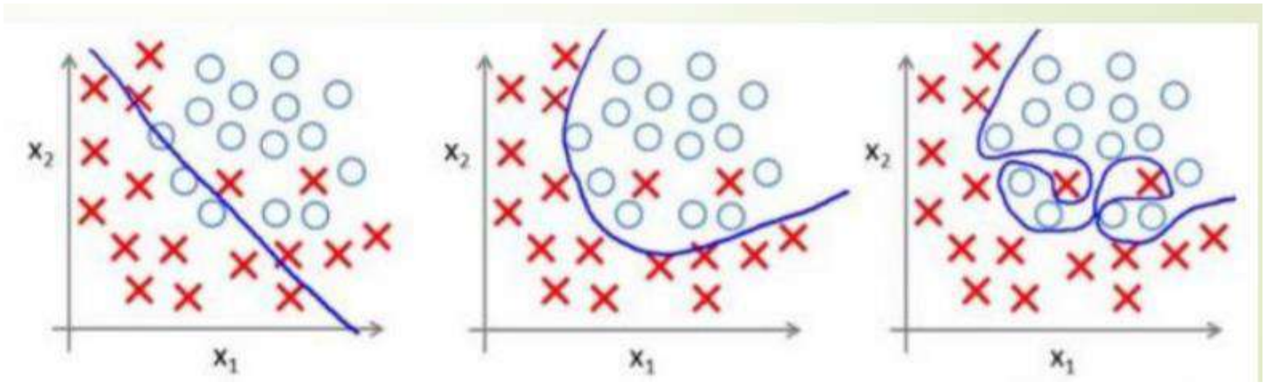
Classification

- A machine learning task that deals with identifying the class to which an instance belongs
- A classifier performs classification
- Classification is an example of pattern recognition

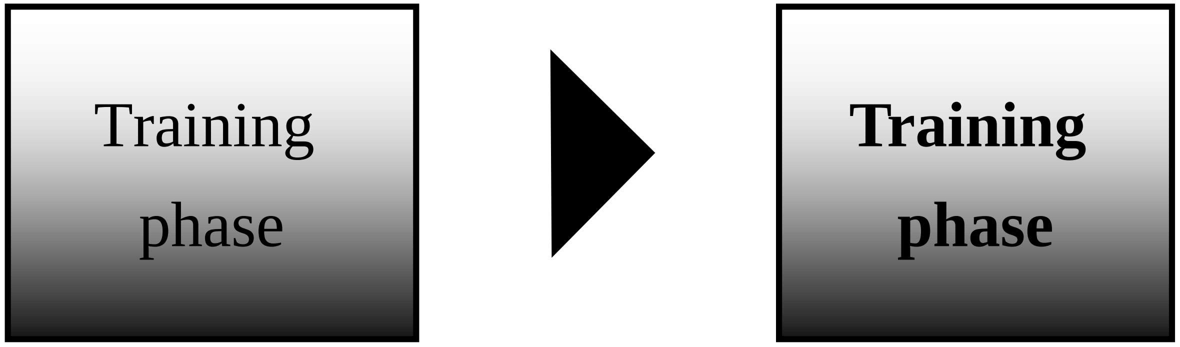


Linear Classification

- A classification algorithm (Classifier) that makes its classification based on a linear predictor function combining a set of weights with the feature vector



Classification learning



Learning the classifier from the available data ‘Training set’ (Labeled)

Learning the classifier from the available data ‘Training set’ (Labeled)

Different approaches

- Explicitly creating the discriminant function (Discriminant function)
  - Perceptron
  - Support vector machine

- Probabilistic approach
  - Model the posterior distribution
  - Algorithms
    - Logistic regression

Regression

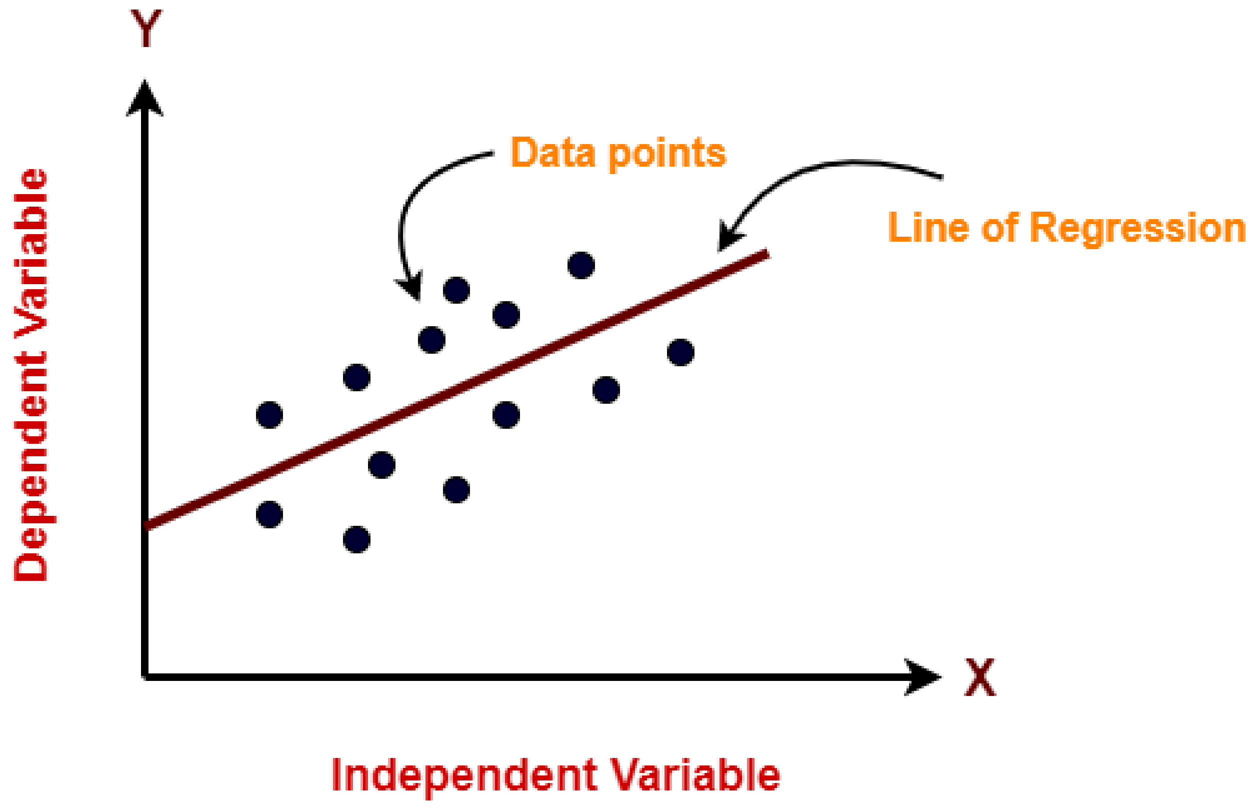
- Regression analysis is a predictive modelling technique
- It estimates the relationship between a dependent (target) and an independent (predictor) variables via a sloped straight line
- The sloped straight line representing the linear relationship that fits the given data best is called as a regression line.
- It is also called as best fit line.

Linear Regression

In Machine Learning,

- Linear Regression is a supervised machine learning algorithm
- It tries to find out the best linear relationship that describes the data you have
- It assumes that there exists a linear relationship between a dependent variable and independent variable(s)
- The value of the dependent variable of a linear regression model is a continuous value i.e. real numbers

Representing Linear Regression Model



Types of Linear Regression

- Simple Linear Regression
- Multiple Linear Regression
- Polynomial Linear Regression

Simple Linear Regression

- In simple linear regression, the dependent variable depends only on a single independent variable
- For simple linear regression, the form of the model is-

$$Y = \beta 0 + \beta 1X$$

Here,

- Y is a dependent variable
- X is an independent variable
- β0 and β1 are the regression coefficients
- β0 is the intercept or the bias that fixes the offset to a line
- β1 is the slope or weight that specifies the factor by which X has an impact on Y

Multiple Linear Regression

- In multiple linear regression, the dependent variable depends on more than one independent variables.

- For multiple linear regression, the form of the model is-

$$Y = \beta 0 + \beta 1X1 + \beta 2X2 + \beta 3X3 + \ldots\ldots + \beta nXn$$

Here,

- Y is a dependent variable
- X1, X2, …., Xn are independent variables
- β0, β1,…, βn are the regression coefficients
- βj (1<=j<=n) is the slope or weight that specifies the factor by which Xj has an impact on Y

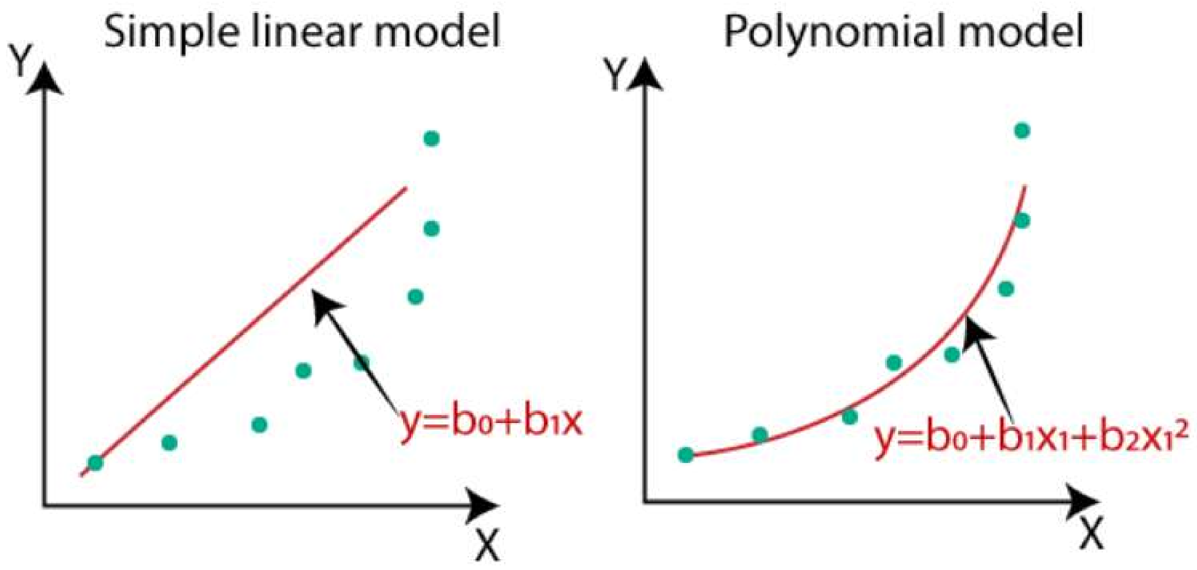
Polynomial Linear Regression

- Polynomial Regression is a regression algorithm that models the relationship between a dependent(y) and independent variable(x) as nth degree polynomial. The Polynomial Regression equation is given below:

  y= b0+b1x1+ b2x12+ b2x13+...... bnx1n

- It is also called the special case of Multiple Linear Regression in ML. Because we add some polynomial terms to the Multiple Linear regression equation to convert it into Polynomial Regression.
- It is a linear model with some modification in order to increase the accuracy.
- The dataset used in Polynomial regression for training is of non-linear nature.
- It makes use of a linear regression model to fit the complicated and non-linear functions and datasets.

Polynomial Linear Regression



Univariate Linear Regression

- Univariate linear regression focuses on determining relationship between one independent (explanatory variable) variable and one dependent variable. Regression comes handy mainly in situation where the relationship between two features is not obvious to the naked eye. For example, it could be used to study how the terrorist attacks frequency affects the economic growth of countries around the world or the role of unemployment in a country in the bankruptcy of the government
- Univariate data is the type of data in which the result depends only on one variable. For instance, dataset of points on a line can be considered as a univariate data where abscissa can be considered as input feature and ordinate can be considered as output/result
- For example:

  For line $Y = 2X + 3$;

  Input feature will be X and Y will be the result.

Univariate Linear Regression

X	Y
1	5
2	7
3	9
4	11
5	13

For univariate linear regression, there is only one input feature vector. The line of regression will be in the form of:

$$Y = b0 + b1 * X$$

Where,

b0 and b1 are the coefficients of regression.

Hence, it is being tried to predict regression coefficients b0 and b1 by training a model.

Regularized Linear Regression

In regularized linear regression, we choose θ to minimize

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

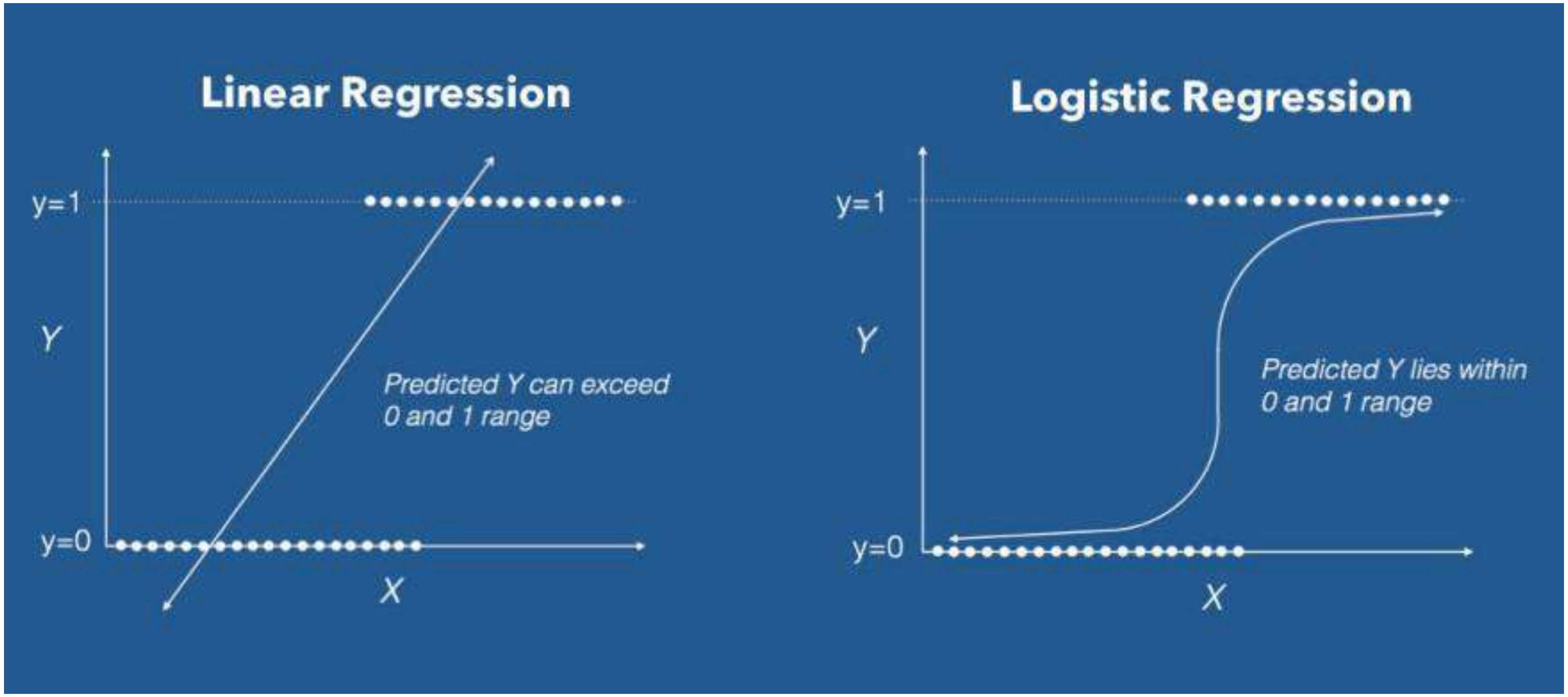
What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

- Algorithm works fine; setting λ to be very large can't hurt it
- Algortihm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

Logistic Linear Regression

- Logistic Regression is a Machine Learning algorithm which is used for the classification problems, it is a predictive analysis algorithm and based on the concept of probability
- Logistic regression is a classification algorithm used to assign observations to a discrete set of classes. Some of the examples of classification problems are Email spam or not spam, Online transactions Fraud or not Fraud, Tumor Malignant or Benign. Logistic regression transforms its output using the logistic sigmoid function to return a probability value.

Logistic Linear Regression


Linear Regression
y=1
Y
y=0
X
Predicted Y can exceed 0 and 1 range
Logistic Regression
y=1
Y
y=0
X
Predicted Y lies within 0 and 1 range

Logistic Linear Regression

- We can call a Logistic Regression a Linear Regression model but the Logistic Regression uses a more complex cost function, this cost function can be defined as the ‘Sigmoid function’ or also known as the ‘logistic function’ instead of a linear function
- The hypothesis of logistic regression tends it to limit the cost function between 0 and 1. Therefore linear functions fail to represent it as it can have a value greater than 1 or less than 0 which is not possible as per the hypothesis of logistic regression

$$0 \leq h_\theta(x) \leq 1$$

Logistic regression hypothesis expectation

Types of Logistic Regression

Binary or Binomial

- In such a kind of classification, a dependent variable will have only two possible types either 1 and 0. For example, these variables may represent success or failure, yes or no, win or loss etc

Multinomial

- In such a kind of classification, dependent variable can have 3 or more possible unordered types or the types having no quantitative significance. For example, these variables may represent "Type A" or "Type B" or "Type C"

Ordinal

- In such a kind of classification, dependent variable can have 3 or more possible ordered types or the types having a quantitative significance. For example, these variables may represent "poor" or "good", "very good", "Excellent" and each category can have the scores like 0,1,2,3

Logistic Regression Assumptions

- Before diving into the implementation of logistic regression, we must be aware of the following assumptions about the same −
- In case of binary logistic regression, the target variables must be binary always and the desired outcome is represented by the factor level 1
- There should not be any multi-collinearity in the model, which means the independent variables must be independent of each other
- We must include meaningful variables in our model
- We should choose a large sample size for logistic regression

Logistic Regression Models

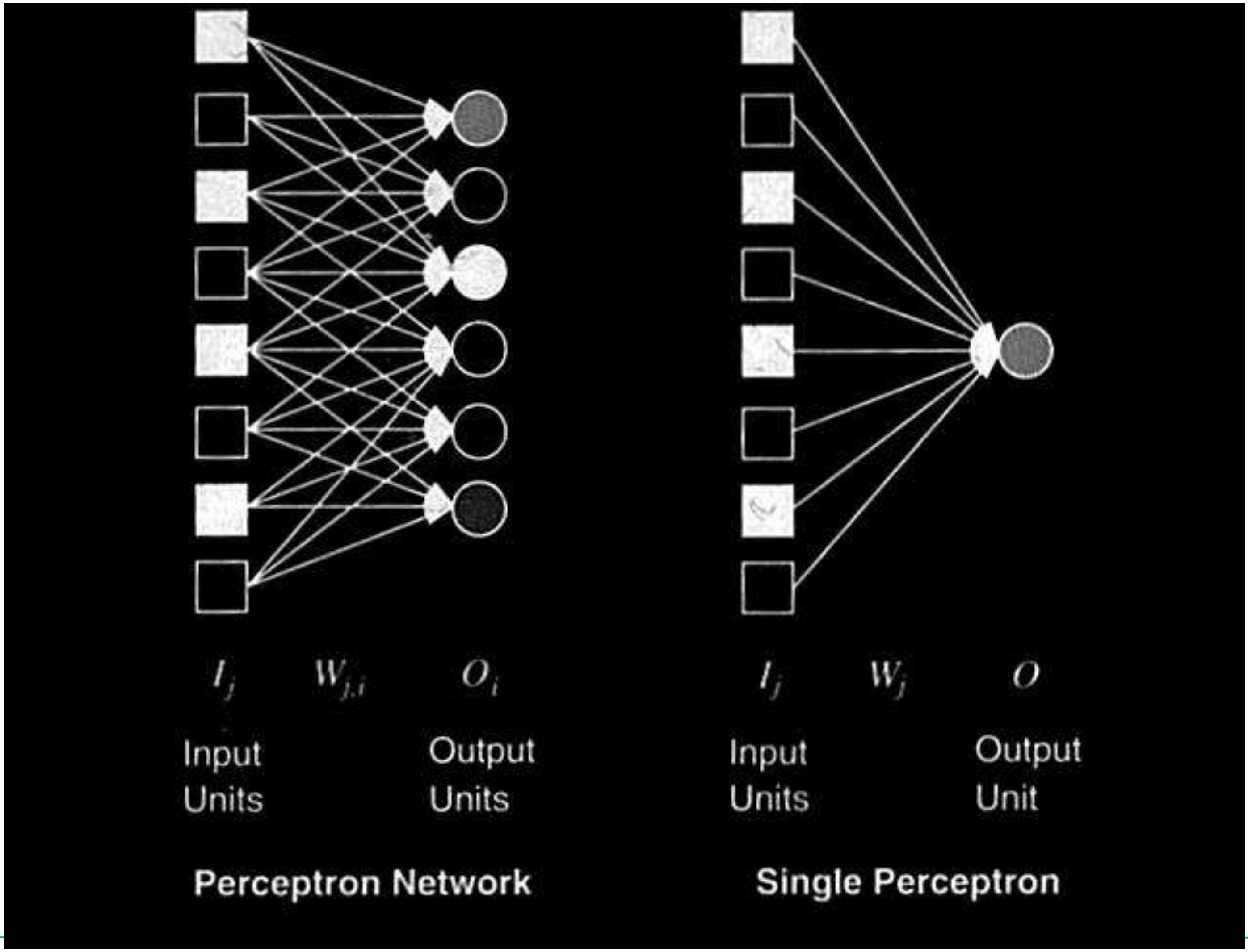
- Binary Logistic Regression Model − The simplest form of logistic regression is binary or binomial logistic regression in which the target or dependent variable can have only 2 possible types either 1 or 0

- Multinomial Logistic Regression Model − Another useful form of logistic regression is multinomial logistic regression in which the target or dependent variable can have 3 or more possible unordered types i.e. the types having no quantitative significance

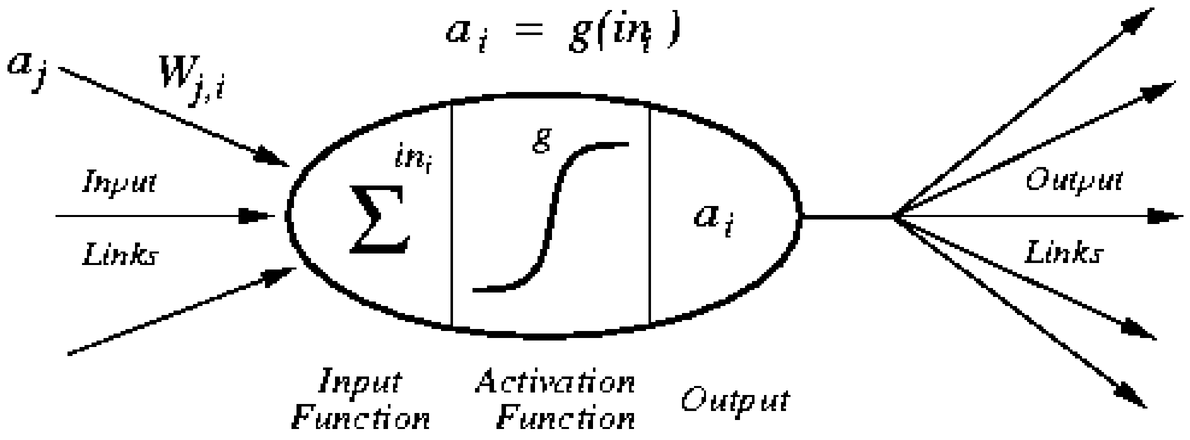
Perceptron

- Also known as Layered Feed-Forward Networks
- The only efficient learning element at that time was for single-layered networks
- Today, used as a synonym for a single-layer, feed-forward network

Perceptron


I_j
$W_{j,i}$
O_i
Input Units
Output Units
Perceptron Network
I_j
W_j
O
Input Units
Output Unit
Single Perceptron

Perceptron


$a_i = g(in_i)$
a_j
$W_{j,i}$
Input
Links
in_i
Σ
g
a_i
Output
Links
Input Function
Activation Function
Output

Perceptron Learning Rule

- Teacher specifies the desired output for a given input
- Network calculates what it thinks the output should be
- Network changes its weights in proportion to the error between the desired & calculated results

$$\Delta w_{i,j} = \alpha * [\text{teacher}_i - \text{output}_i] * \text{input}$$

where

α is the learning rate;

$\text{teacher}_i - \text{output}_i$ is the error term;

input_j is the input activation

$w_{i,j} = w_{i,j} + \Delta w_{i,j}$

Delta rule

Perceptron Learning Rule

If we define the learning signal r by $r = d - y$, we get the perceptron learning rule. That is,

$$\mathbf{w}^{new} = \mathbf{w}^{old} + c(d - y)\mathbf{x}$$

where d and y are, respectively, the desired output (or teacher signal) and the actual output. It is known that the above rule converges if a solution exists.

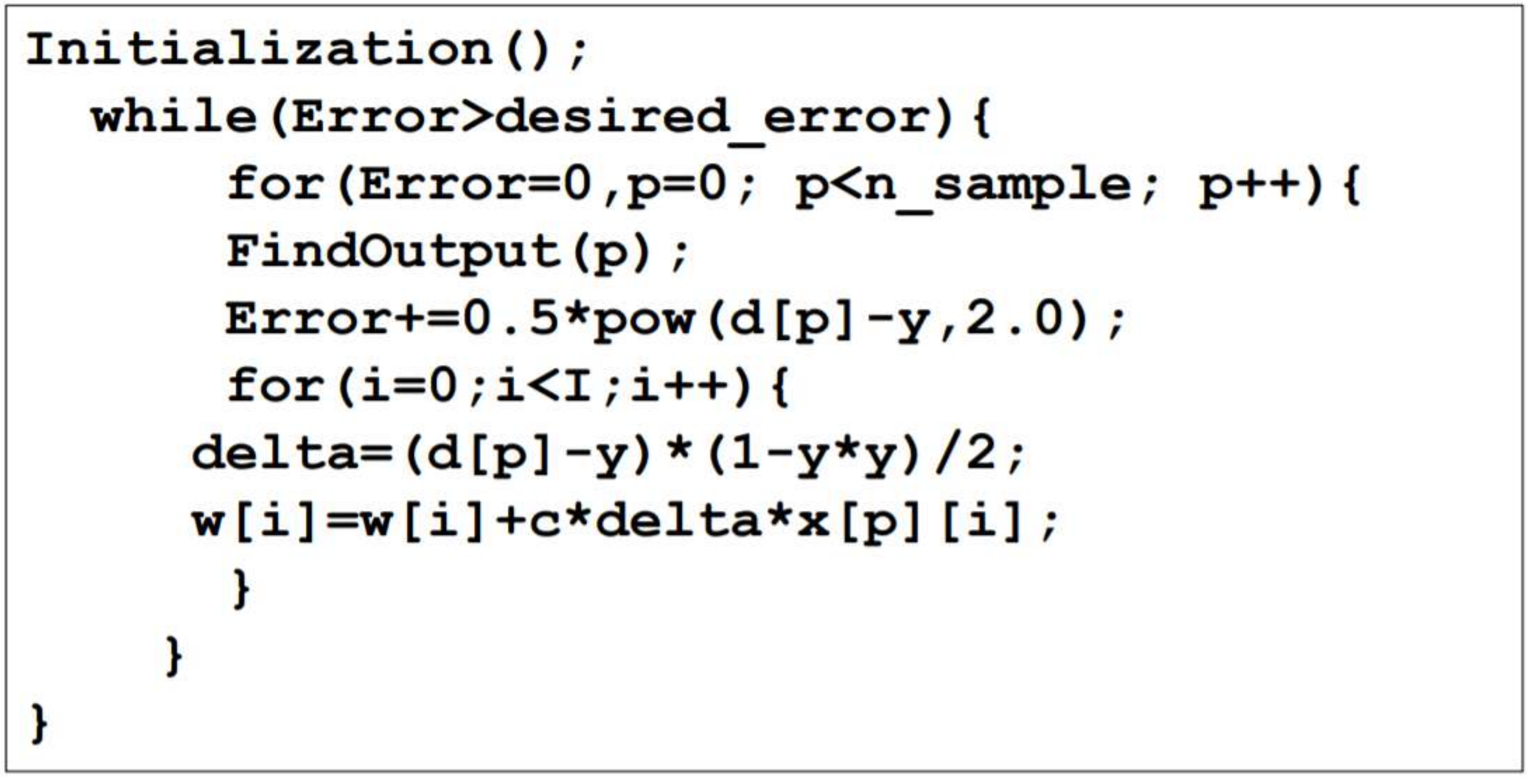
Delta Learning Rule

If the activation function is differentiable, we can use the following delta - learning rule :

$$\mathbf{w}^{new} = \mathbf{w}^{old} + c(d - y)g'(u)x$$

where $u = w^t x =< w, x >= \sum_{i=1}^{n} w_i x_i$ is the effective input, and $g'(u)$ is the first order direvative of the activiation function at u. Clearly, the learning signal r is defined by $(d - y)g'(u)$. It is known that the above rule convergies to the *mean squared error* solution.

Program for Delta Learning Rule



```
Initialization();
  while(Error>desired_error){
      for(Error=0,p=0; p<n_sample; p++){
      FindOutput(p);
      Error+=0.5*pow(d[p]-y,2.0);
      for(i=0;i<I;i++){
     delta=(d[p]-y)*(1-y*y)/2;
     w[i]=w[i]+c*delta*x[p][i];
      }
    }
}
```

Adjusting Perceptron Weights

- $\Delta w_{i,j} = \alpha * [\text{teacher}_i - \text{output}_i] * \text{input}_j$
- miss_i is $(\text{teacher}_i - \text{output}_i)$

	miss<0	miss=0	miss>0
input < 0	alpha	0	-alpha
input = 0	0	0	0
input > 0	-alpha	0	alpha

- Adjust each $w_{i,j}$ based on input_j and miss_i
- The above table shows adaptation.
- Incremental learning.

Perceptron Convergence Theorem

- If a set of <input, output> pairs are learnable (representable), the delta rule will find the necessary weights

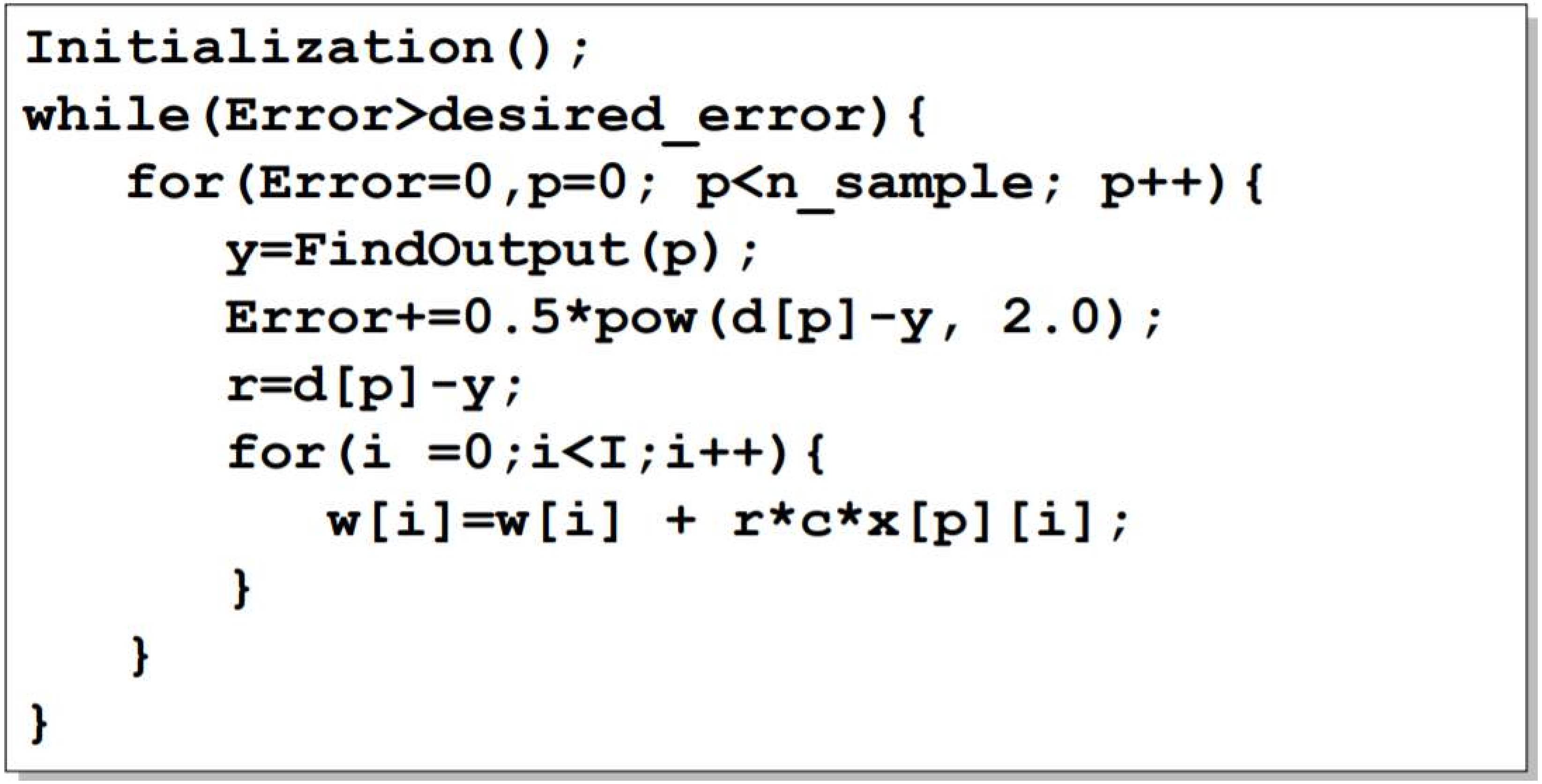
  -in a finite number of steps

  -independent of initial weights

- However, a single layer perceptron can only learn linearly separable concepts

  -it works iff gradient descent works

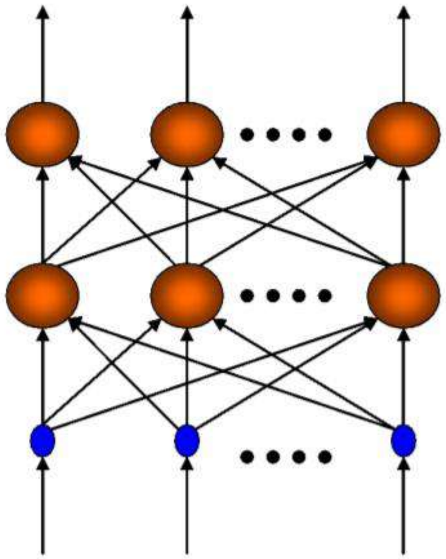
Program for Perceptron Learning



```
Initialization();
while(Error>desired_error){
    for(Error=0,p=0; p<n_sample; p++){
        y=FindOutput(p);
        Error+=0.5*pow(d[p]-y, 2.0);
        r=d[p]-y;
        for(i =0;i<I;i++){
            w[i]=w[i] + r*c*x[p][i];
        }
    }
}
```

Multilayer Perceptron

- The right figure is a multilayer neural network or multilayer perceptron (MLP).
- There are three layers: input layer, hidden layer, and output layer.
- There can be more than two hidden layers.
- MLP is known as a general approximator that be used to approximate any given function.



Multilayer Neural Networks

- Identifying the appropriate nonlinear functions to be used can be difficult and incredibly expensive
- We need a way to learn the non-linearity at the same time as the linear discriminant
- Multilayer Neural Networks, in principle, do exactly this in order to provide the optimal solution to arbitrary classification problems
- Multilayer Neural Networks implement linear discriminants in a space where the inputs have been mapped non-linearly
- The form of the non-linearity can be learned from simple algorithms on training data
- Note that neural networks require continuous functions to allow for gradient descent

Multilayer Neural Networks

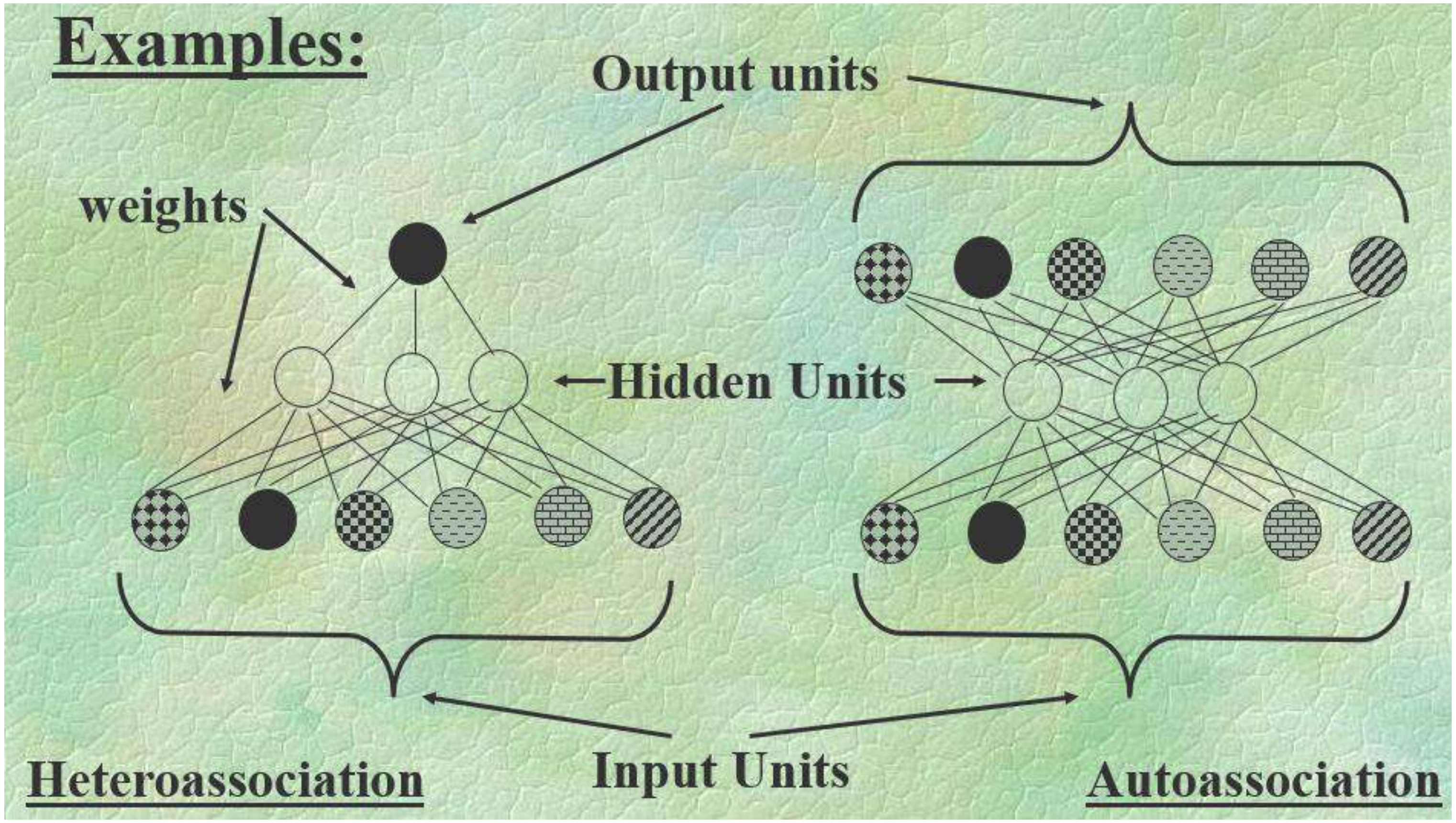
- Training multilayer neural networks can involve a number of different algorithms, but the most popular is the back propagation algorithm or generalized delta rule
- Back propagation is a natural extension of the LMS algorithm
- The back propagation method is simple for models of arbitrary complexity
- This makes the method very flexible
- One of the largest difficulties with developing neural networks is regularization, or adjusting the complexity of the network
- Too many parameters = poor generalization
- Too few parameters = poor learning

Learning in Multilayer Neural Networks

- Learning consists of searching through the space of all possible matrices of weight values for a combination of weights that satisfies a database of positive and negative examples (multi-class as well as regression problems are possible)

- Note that a Neural Network model with a set of adjustable weights defines a restricted hypothesis space corresponding to a family of functions. The size of this hypothesis space can be increased or decreased by increasing or decreasing the number of hidden units present in the network

Multilayer Neural Network Representation



How a Multi-Layer Neural Network Works?

- The inputs to the network correspond to the attributes measured for each training tuple
- Inputs are fed simultaneously into the units making up the input layer
- They are then weighted and fed simultaneously to a hidden layer
- The number of hidden layers is arbitrary, although usually only one
- The weighted outputs of the last hidden layer are input to units making up the output layer, which emits the network's prediction
- The network is feed-forward in that none of the weights cycles back to an input unit or to an output unit of a previous layer
- From a statistical point of view, networks perform nonlinear regression: Given enough hidden units and enough training samples, they can closely approximate any function

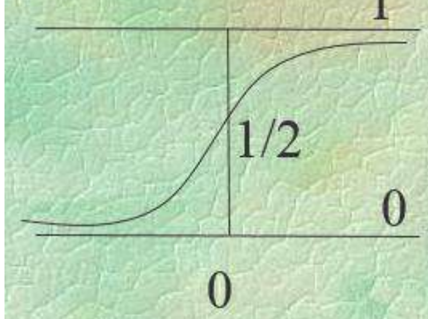
How is a Function Computed by a Multilayer Neural Network?

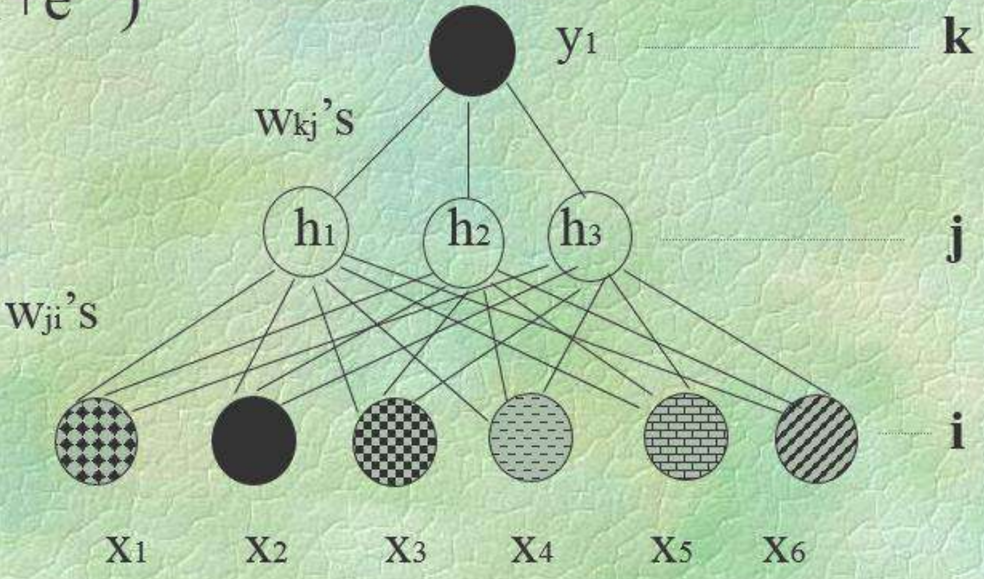
- $h_j = g(\sum_i w_{ji} . x_i)$
- $y_1 = g(\sum_j w_{kj} . h_j)$

where $g(x) = 1/(1+e^{-x})$

Typically, $y_1=1$ for positive example and $y_1=0$ for negative example

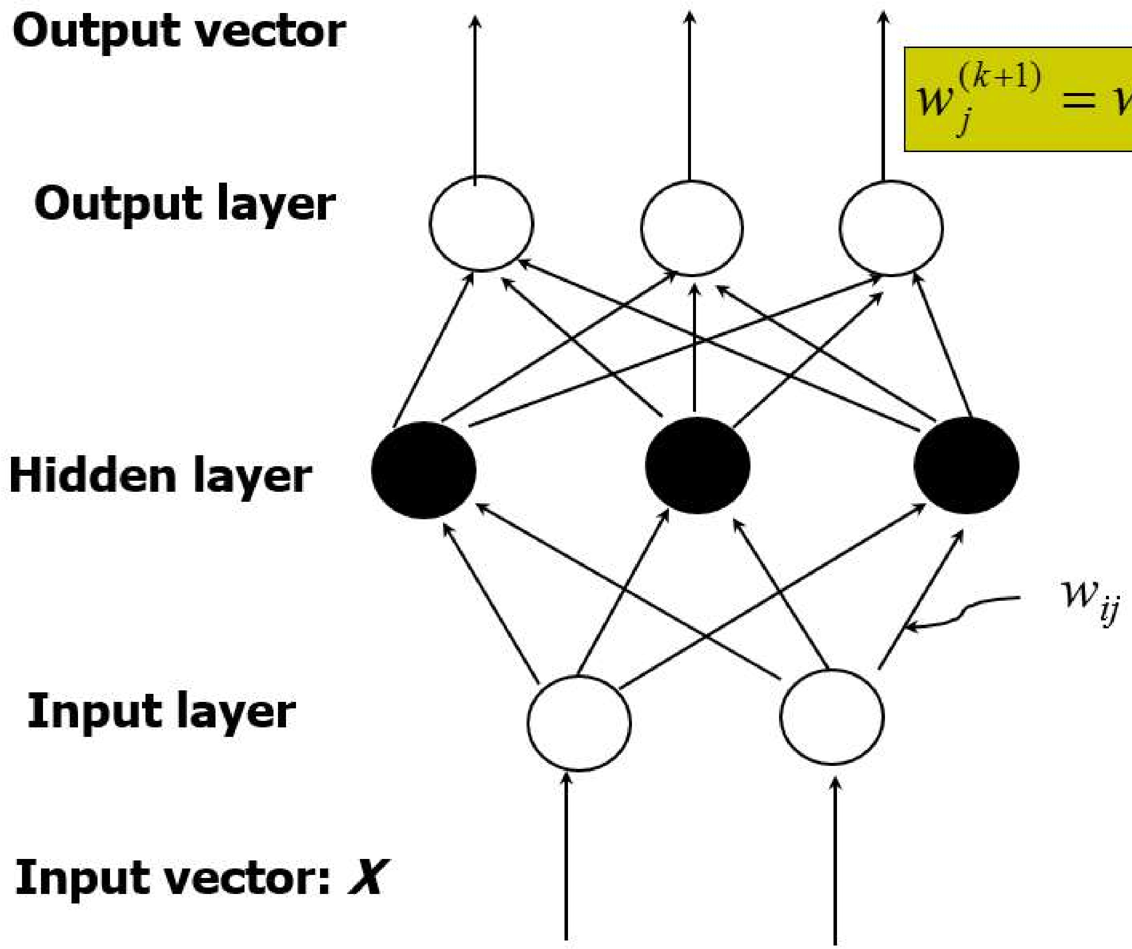
g (sigmoid):





A Multi-Layer Feed-Forward Neural Network

$$w_j^{(k+1)} = w_j^{(k)} + \lambda(y_i - \hat{y}_i^{(k)})x_{ij}$$



Support Vector Machines

- Support Vector Machine (SVM) is related to statistical learning theory
- SVM was first introduced in 1992
- SVM becomes popular because of its success in handwritten digit recognition
- SVM is now regarded as an important example of “kernel methods”, one of the key area in machine learning

Support Vector Machines

- Three main ideas:

1. Define what an optimal hyperplane is (taking into account that it needs to be computed efficiently): maximize margin
2. Generalize to non-linearly separable problems: have a penalty term for misclassifications
3. Map data to high dimensional space where it is easier to classify with linear decision surfaces: reformulate problem so that data are mapped implicitly to this space

Motivation for Support Vector Machines

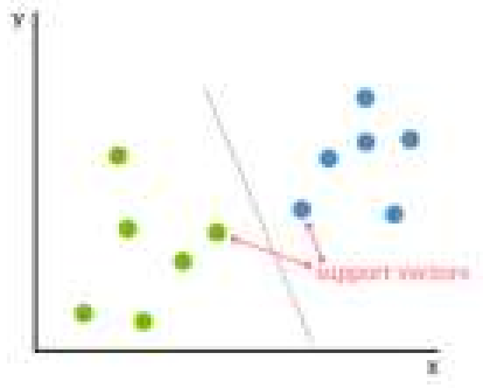
- The problem to be solved is one of the **supervised binary classification**. That is, we wish to categorize new unseen objects into two separate groups based on their properties and a set of known examples, which are already categorized
- A good example of such a system is classifying a set of new *documents* into positive or negative sentiment groups, based on other documents which have already been classified as positive or negative
- Similarly, we could classify new emails into spam or non-spam, based on a large corpus of documents that have already been marked as spam or non-spam by humans. SVMs are highly applicable to such situations

Motivation for Support Vector Machines

- A Support Vector Machine models the situation by creating a feature space, which is a finite-dimensional vector space, each dimension of which represents a "feature" of a particular object. In the context of spam or document classification, each "feature" is the prevalence or importance of a particular word
- The goal of the SVM is to train a model that assigns new unseen objects into a particular category.
- It achieves this by creating a linear partition of the feature space into two categories
- Based on the features in the new unseen objects (e.g. documents/emails), it places an object "above" or "below" the separation plane, leading to a categorization (e.g. spam or non-spam). This makes it an example of a non-probabilistic linear classifier. It is non-probabilistic, because the features in the new objects fully determine its location in feature space and there is no stochastic element involved

Objectives of Support Vector Machines

- Support vector machines (SVM) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis
- It is a machine learning approach
- They analyze the large amount of data to identify patterns from them.
- SVMs are based on the idea of finding a hyperplane that best divides a dataset into two classes, as shown in the image below



Support Vectors

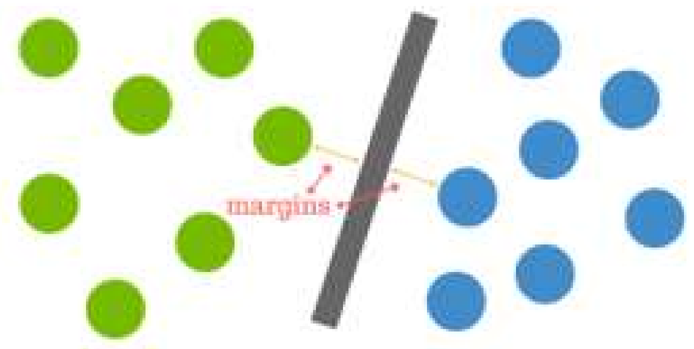
- Support Vectors are simply the co-ordinates of individual observation. Support Vector Machine is a frontier which best segregates the two classes (hyper-plane/ line)
- Support vectors are the data points that lie closest to the decision surface (or hyperplane)
- They are the data points most difficult to classify
- They have direct bearing on the optimum location of the decision surface
- We can show that the optimal hyperplane stems from the function class with the lowest “capacity” (VC dimension)
- Support vectors are the data points nearest to the hyperplane, the points of a data set that, if removed, would alter the position of the dividing hyperplane. Because of this, they can be considered the critical elements of a data set

What is a hyperplane?

- As a simple example, for a classification task with only two features, you can think of a hyperplane as a line that linearly separates and classifies a set of data
- Intuitively, the further from the hyperplane our data points lie, the more confident we are that they have been correctly classified. We therefore want our data points to be as far away from the hyperplane as possible, while still being on the correct side of it
- So when new testing data are added, whatever side of the hyperplane it lands will decide the class that we assign to it

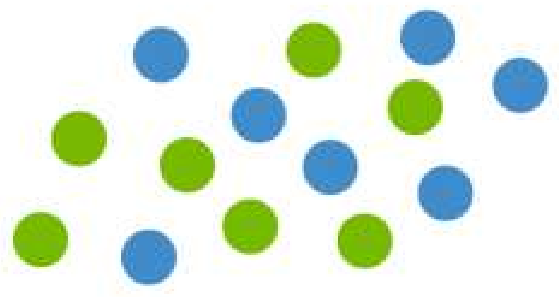
How do we find the right hyperplane?

- How do we best segregate the two classes within the data?
- The distance between the hyperplane and the nearest data point from either set is known as the margin. The goal is to choose a hyperplane with the greatest possible margin between the hyperplane and any point within the training set, giving a greater chance of new data being classified correctly. There will never be any data point inside the margin

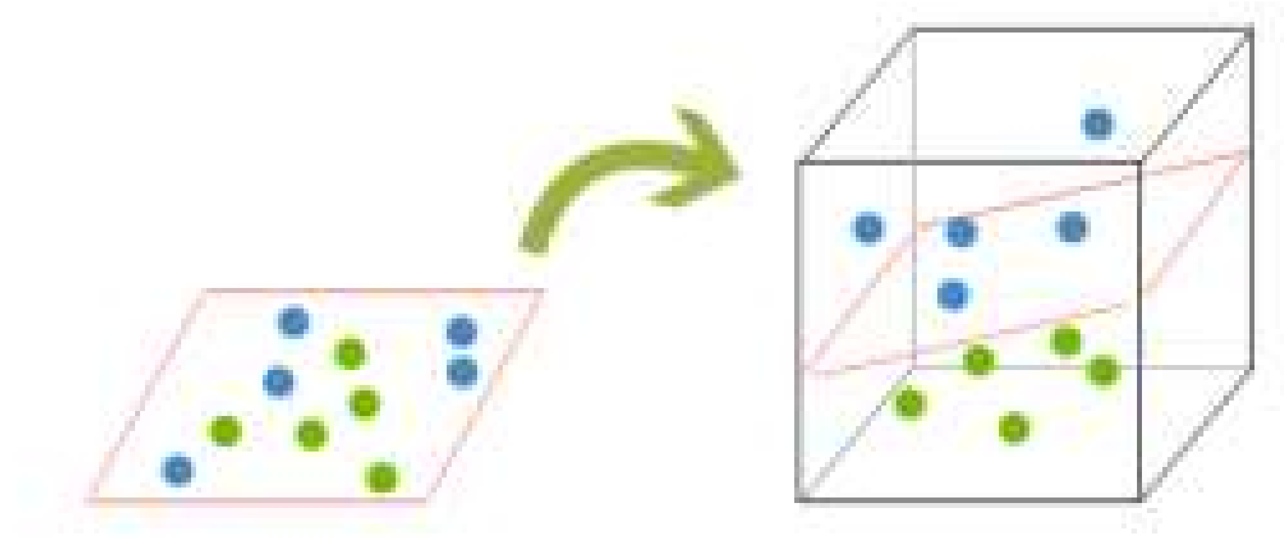


What happens when there is no clear hyperplane?

- Data are rarely ever as clean as our simple example above. A dataset will often look more like the jumbled balls below which represent a linearly non separable dataset
- In order to classify a dataset like the one above it's necessary to move away from a 2d view of the data to a 3d view. Explaining this is easiest with another simplified example. Imagine that our two sets of colored balls above are sitting on a sheet and this sheet is lifted suddenly, launching the balls into the air. While the balls are up in the air, you use the sheet to separate them. This 'lifting' of the balls represents the mapping of data into a higher dimension. This is known as kernelling



What happens when there is no clear hyperplane?



- Because we are now in three dimensions, our hyperplane can no longer be a line. It must now be a plane as shown in the example above. The idea is that the data will continue to be mapped into higher and higher dimensions until a hyperplane can be formed to segregate it.

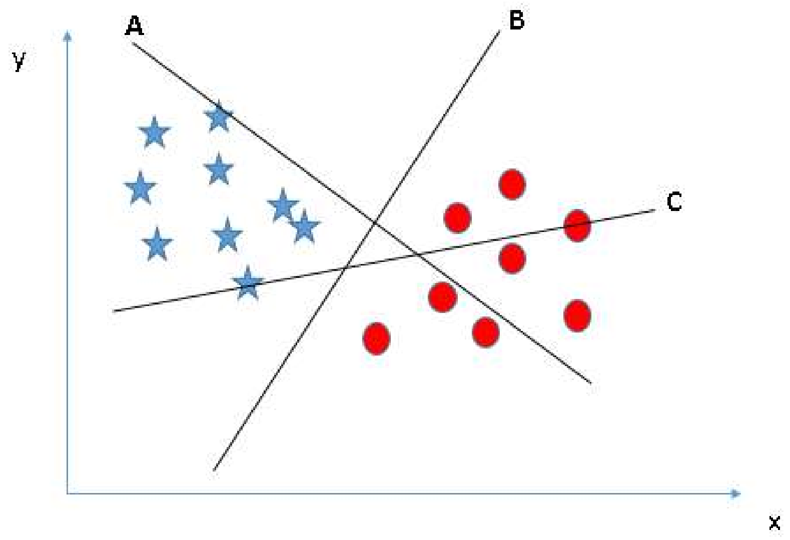
How does it work? How can we identify the right hyper-plane?

- You need to remember a thumb rule to identify the right hyper-plane:

"Select the hyper-plane which segregates the two classes better".

Identify the right hyperplane (Scenario-1)

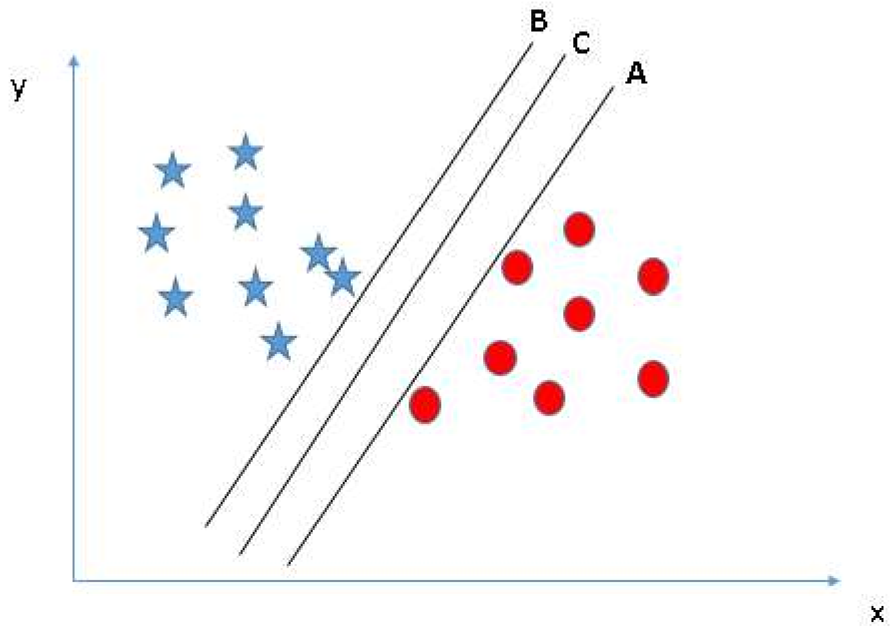
- Here, we have three hyperplanes (A, B and C). Now, identify the right hyperplane to classify star and circle.



- Hyperplane "B" has excellently performed this job.

Identify the right hyperplane (Scenario-2)

- Here, we have three hyperplanes (A, B and C) and all are segregating the classes well. Now, how can we identify the right hyperplane?

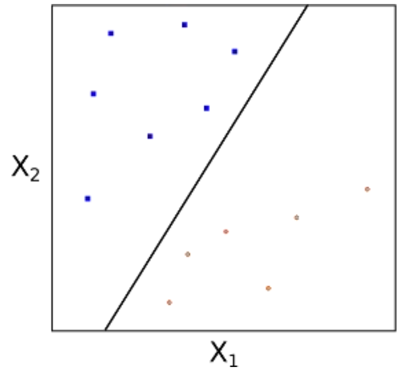


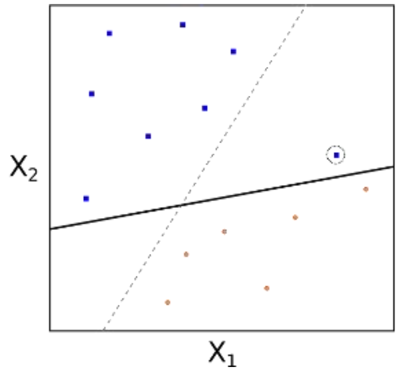
- Here, maximizing the distances between nearest data point (either class) and hyperplane will help us to decide the right hyperplane

Support Vector Classifiers

- Essentially, we have to relax the requirement that a separating hyperplane will perfectly separate every training observation on the correct side of the line (i.e. guarantee that it is associated with its true class label), using what is called a soft margin. This motivates the concept of a support vector classifier(SVC)
- MMCs can be extremely sensitive to the addition of new training observations

MMH perfectly separating the two classes

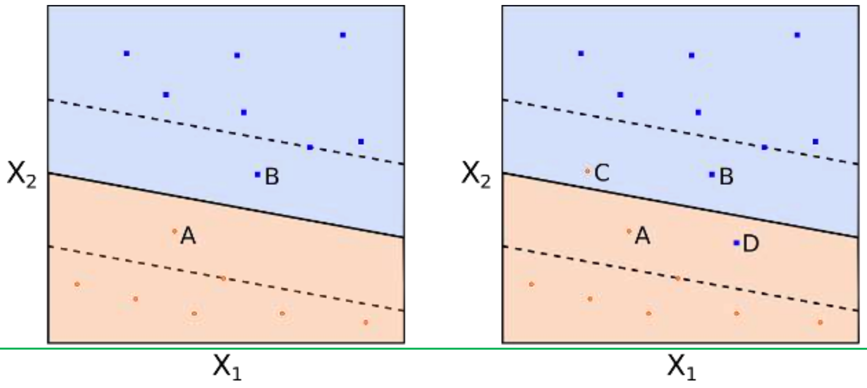




If we add one point to the +1 class, we see that the location of the MMH changes substantially. Hence in this situation the MMH has clearly been over-fit.

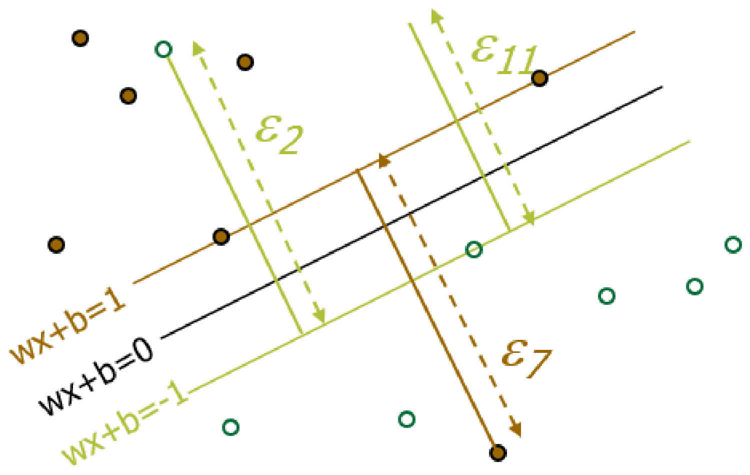
Support Vector Classifiers

- We could consider a classifier based on a separating hyperplane that doesn't perfectly separate the two classes, but does have a greater robustness to the addition of new individual observations and has a better classification on most of the training observations. This comes at the expense of some misclassification of a few training observations
- This is how a support vector classifier or soft margin classifier works. A SVC allows some observations to be on the incorrect side of the margin (or hyperplane), hence it provides a "soft" separation. The following figures demonstrate observations being on the wrong side of the margin and the wrong side of the hyperplane respectively:



Soft Margin Classification

Slack variables ξi **can be added to allow misclassification of difficult or noisy examples.**



What should our quadratic optimization criterion be?

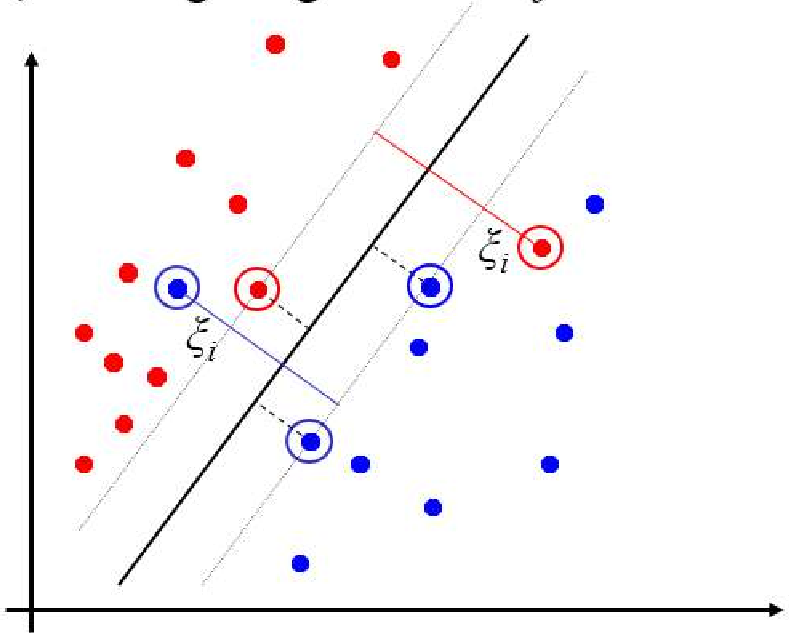
Minimize

$$\frac{1}{2}\mathbf{w}.\mathbf{w} + C\sum_{k=1}^{R}\varepsilon_k$$

- Overfitting can be controlled by soft margin approach

Soft Margin Classification

- What if the training set is not linearly separable?
- *Slack variables* ξ_i can be added to allow misclassification of difficult or noisy examples, resulting margin called *soft*.



Soft Margin Classification Mathematically

- The old formulation:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w}$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1$

- Modified formulation incorporates slack variables:

Find **w** and b such that
$\mathbf{\Phi}(\mathbf{w}) = \mathbf{w}^T\mathbf{w} + C\Sigma\xi_i$ is minimized
and for all $(\mathbf{x}_i, y_i)$, $i=1..n$: $y_i (\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_{i,}$, $\xi_i \geq 0$

- Parameter C can be viewed as a way to control overfitting: it “trades off” the relative importance of maximizing the margin and fitting the training data.

Soft Margin Classification – Solution

- Dual problem is identical to separable case (would *not* be identical if the 2-norm penalty for slack variables $C\Sigma\xi_i^2$ was used in primal objective, we would need additional Lagrange multipliers for slack variables):

Find $\alpha_1 \dots \alpha_N$ such that
$\mathbf{Q}(\boldsymbol{\alpha}) = \Sigma\alpha_i - \frac{1}{2}\Sigma\Sigma\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_j$ is maximized and
(1) $\Sigma\alpha_i y_i = 0$
(2) $0 \leq \alpha_i \leq C$ for all α_i

- Again, $\mathbf{x}_i$ with non-zero α_i will be support vectors.
- Solution to the dual problem is:

$\mathbf{w} = \Sigma\alpha_i y_i \mathbf{x}_i$
$b = y_k(1 - \xi_k) - \Sigma\alpha_i y_i \mathbf{x}_i^{\mathbf{T}}\mathbf{x}_k$ for any k s.t. $\alpha_k > 0$

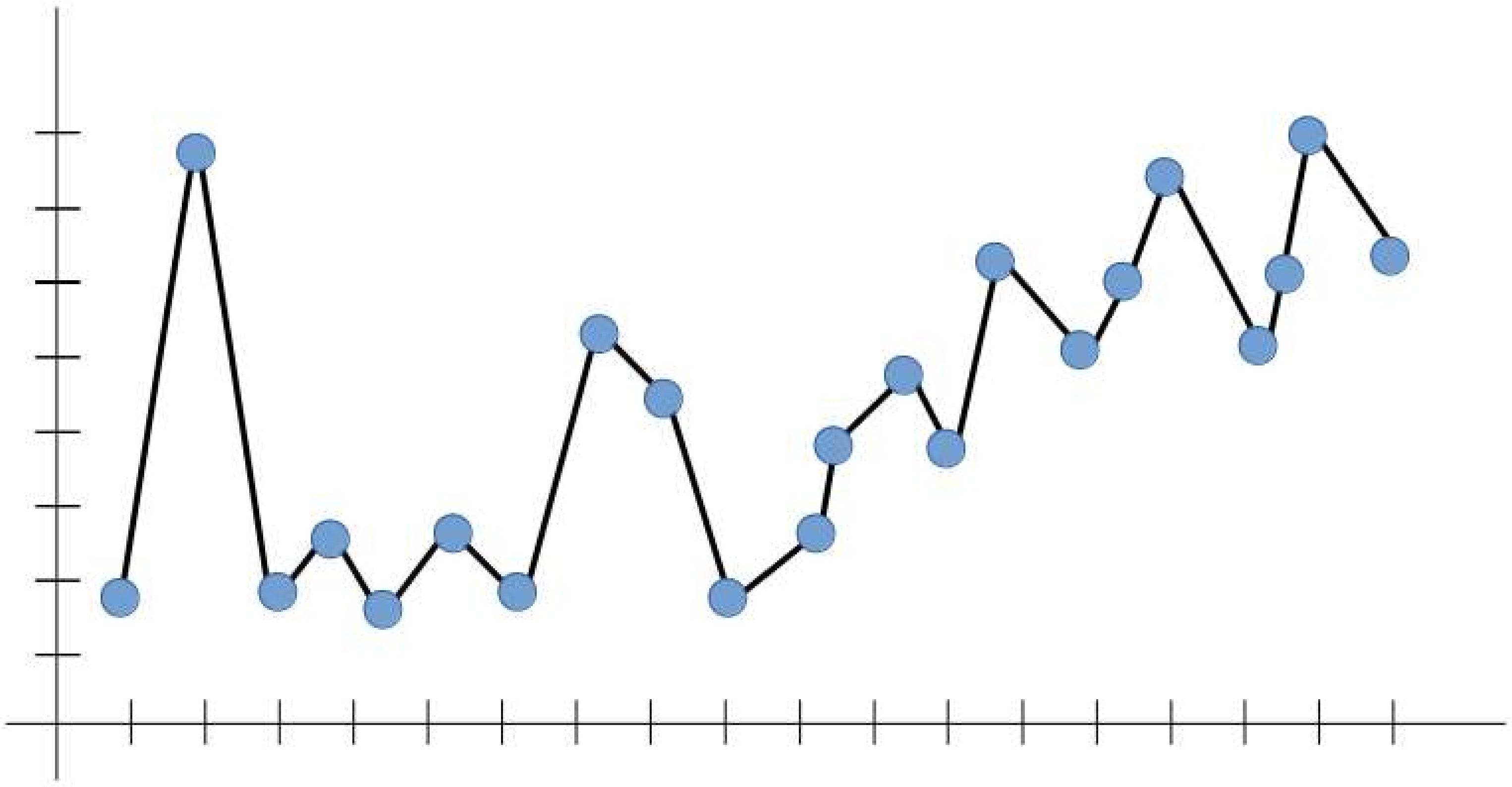
Again, we don't need to compute **w** explicitly for classification:

$f(\mathbf{x}) = \Sigma\alpha_i y_i \mathbf{x}_i^{\mathbf{T}}\mathbf{x} + b$

Generalization and Over Fitting

- Generalization is the model’s ability to give sensible outputs to sets of input that it has never seen before
- Arguably, Machine Learning models have one sole purpose; to generalize well
- A model that generalizes well is a model that is neither underfit nor overfit
- When we run our training algorithm on the data set, we allow the overall cost (i.e. distance from each point to the line) to become smaller with more iterations. Leaving this training algorithm run for long leads to minimal overall cost. However, this means that the line will be fit into all the points (including noise), catching secondary patterns that may not be needed for the generalizability of the model
- If we leave the learning algorithm running for long, it cold end up fitting the line in the following manner:

Generalization and Over Fitting



- This looks good, right? Yes, but is it reliable? Well, not really

Generalization and Over Fitting

- In the figure above, the algorithm captured all trends — but not the dominant one. If we want to test the model on inputs that are beyond the line limits we have (i.e. generalize), what would that line look like? There is really no way to tell. Therefore, the outputs aren't reliable

- If the model does not capture the dominant trend that we can all see (positively increasing, in our case), it can't predict a likely output for an input that it has never seen before — defying the purpose of Machine Learning to begin with

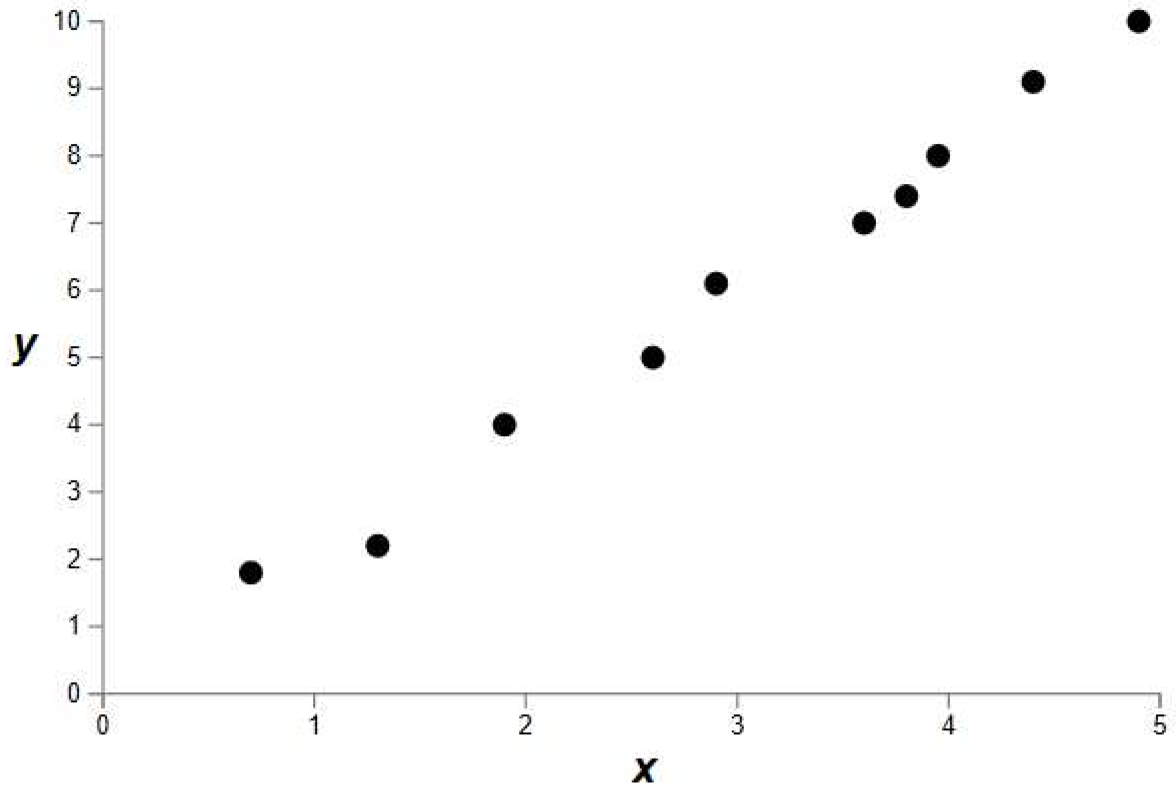
Regularization

- Too many parameters = overfitting
- Not enough parameters = underfitting
- More data = less chance to overfit
- How do we know what is required?

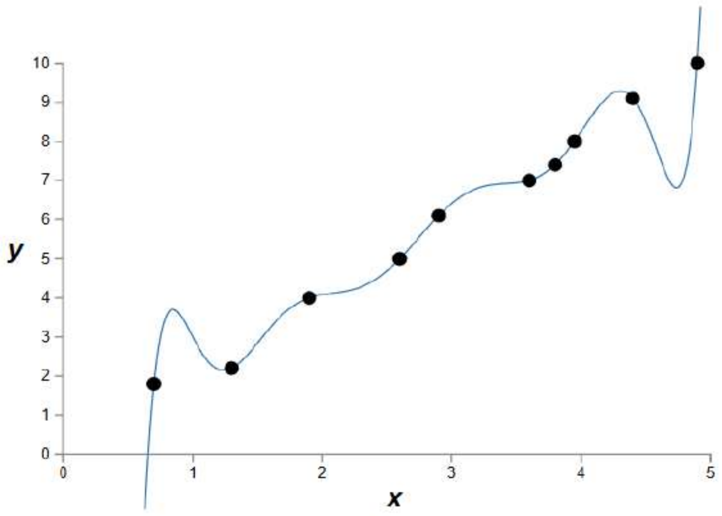
Regularization

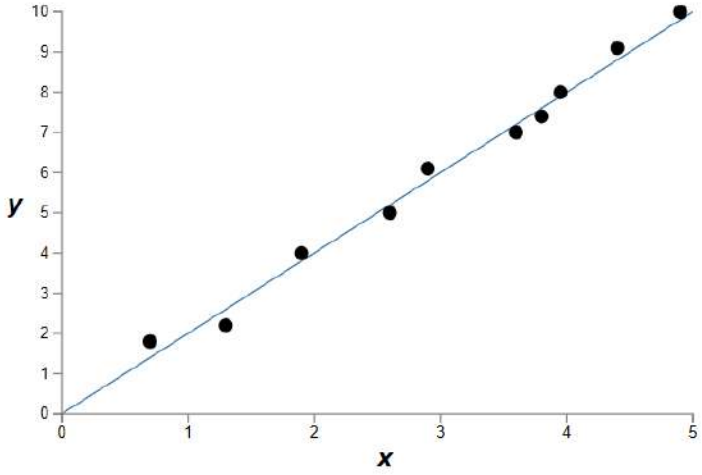
- Attempt to guide solution to not overfit
- But still give freedom with many parameters

Data fitting problem



Which is better a priori?





9th order polynomial

1st order polynomial

Regularization

- Attempt to guide solution to not overfit
- But still give freedom with many parameters
- Idea:
  Penalize the use of parameters to prefer small weights.

- Idea: add a cost to having high weights
- λ = regularization parameter

$$C = C_0 + \frac{\lambda}{2n} \sum_w w^2,$$

Forms of Regularization

- Adding more data is a kind of regularization
- Pooling is a kind of regularization
- Data augmentation is a kind of regularization

Validation

- Cross-validation generates an approximate estimate of how well the learned model will do on “unseen” data

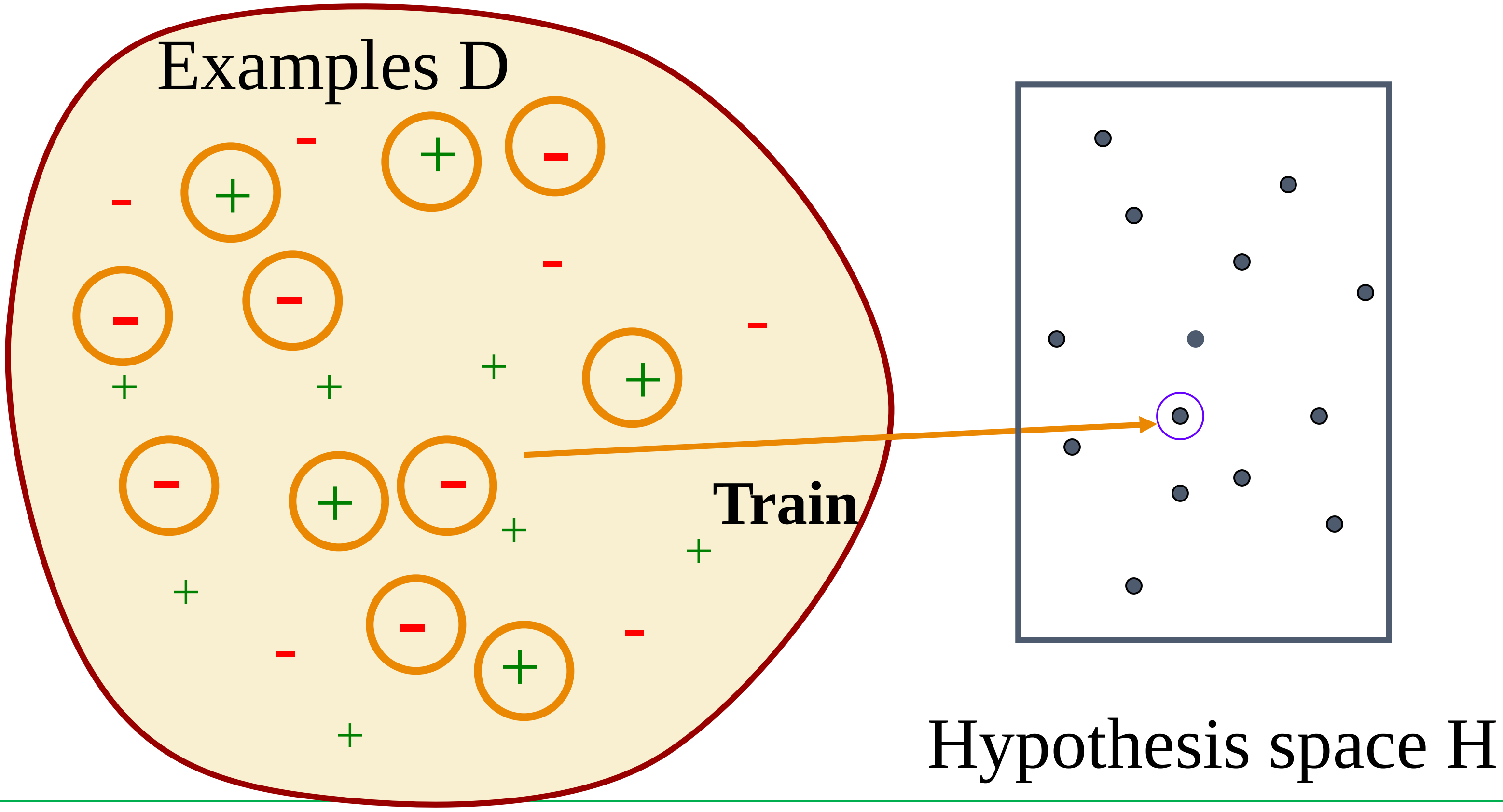
- By averaging over different partitions it is more robust than just a single train/validate partition of the data

- “k-fold” cross-validation is a generalization
    - partition data into disjoint validation subsets of size n/k
    - train, validate, and average over the v partitions
    - e.g., k=10 is commonly used

- k-fold cross-validation is approximately k times computationally more expensive than just fitting a model to all of the data

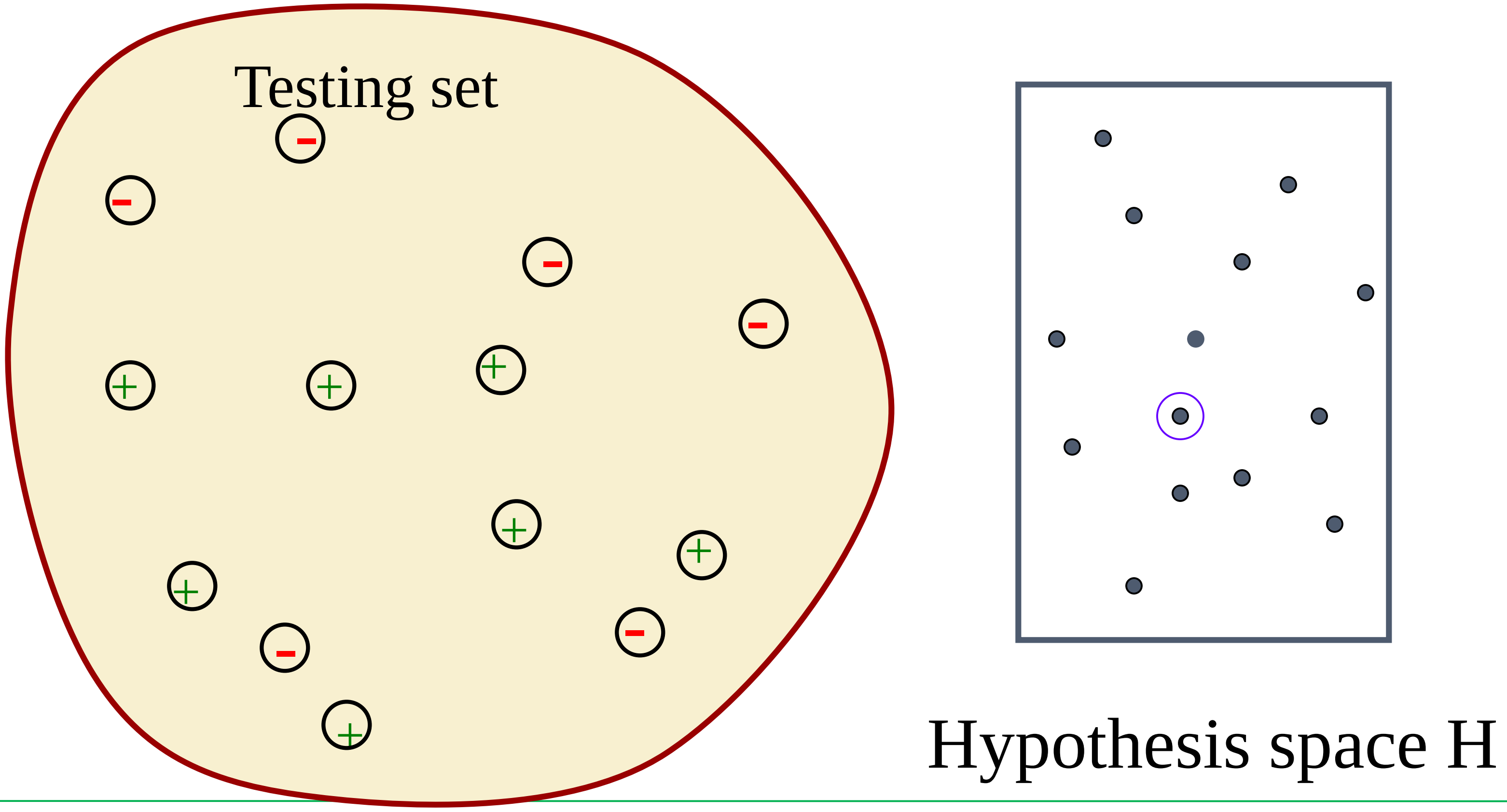
Cross-Validation

- Split original set of examples, train



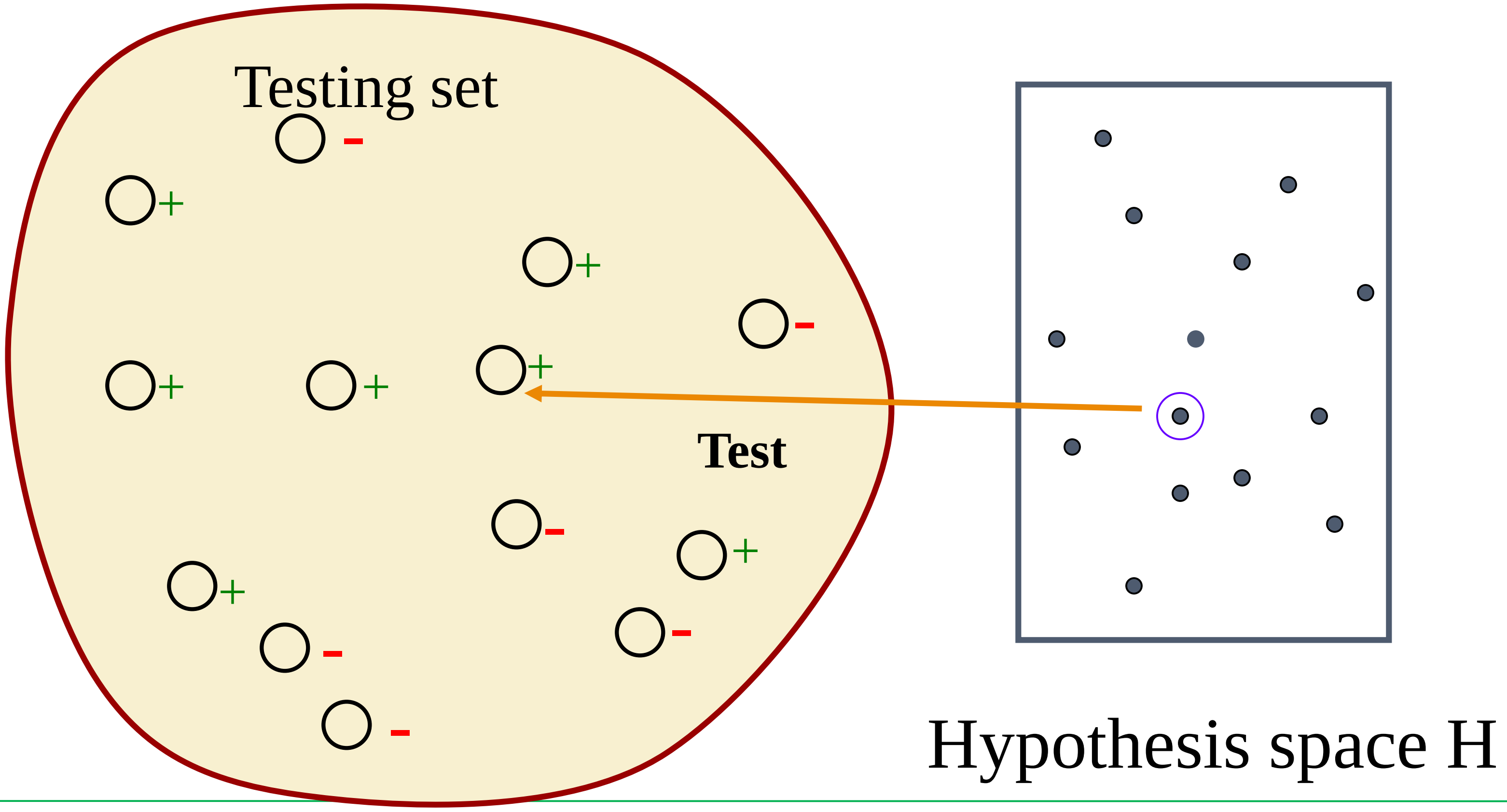
Cross-Validation

- Evaluate hypothesis on testing set



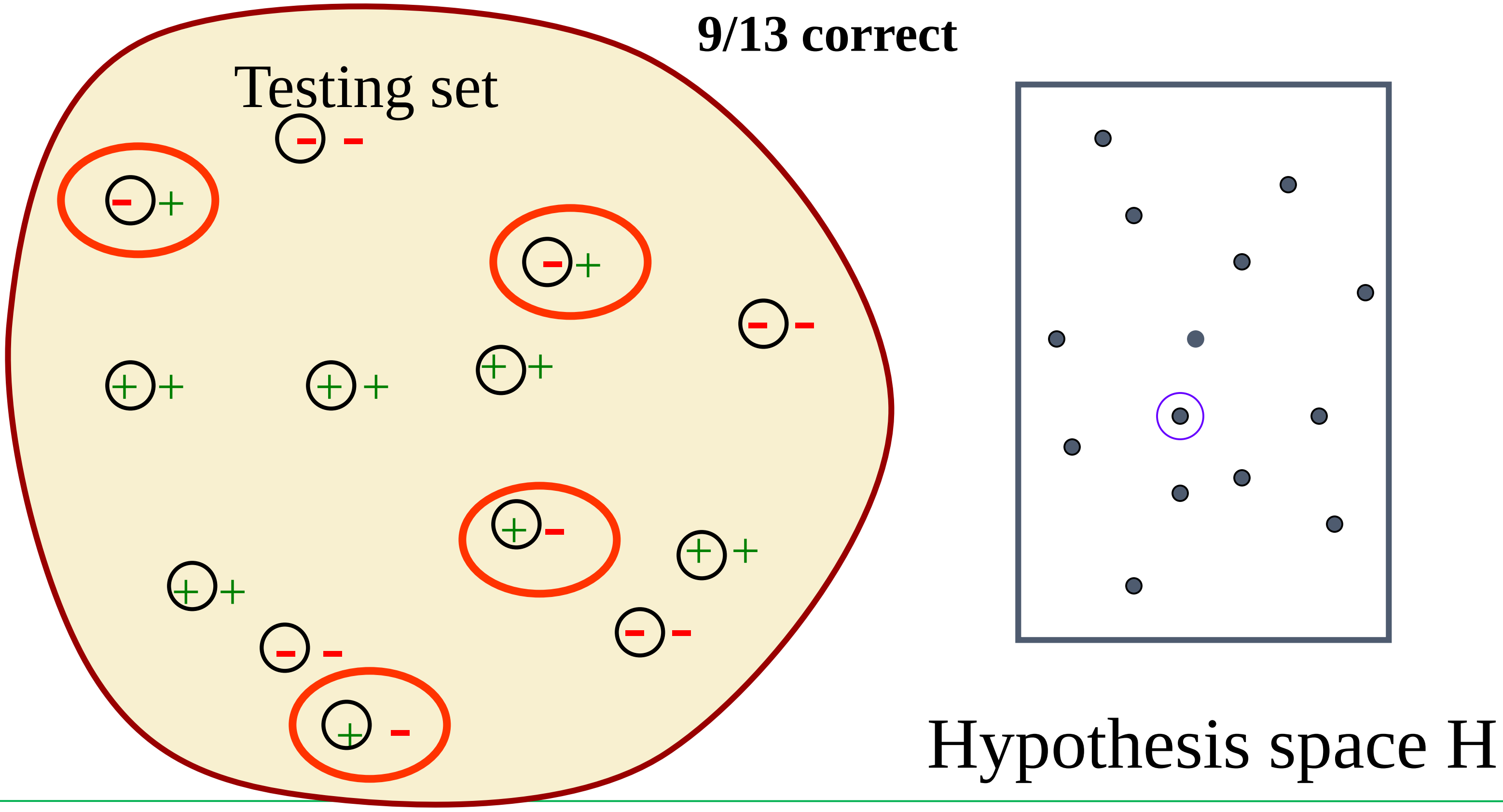
Cross-Validation

- Evaluate hypothesis on testing set



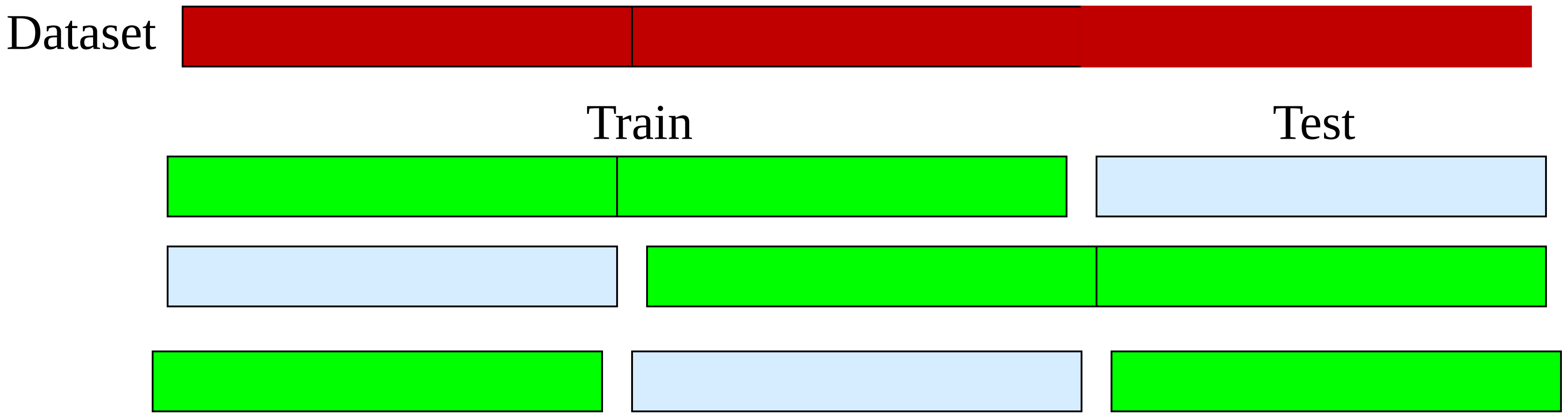
Cross-Validation

- Compare true concept against prediction



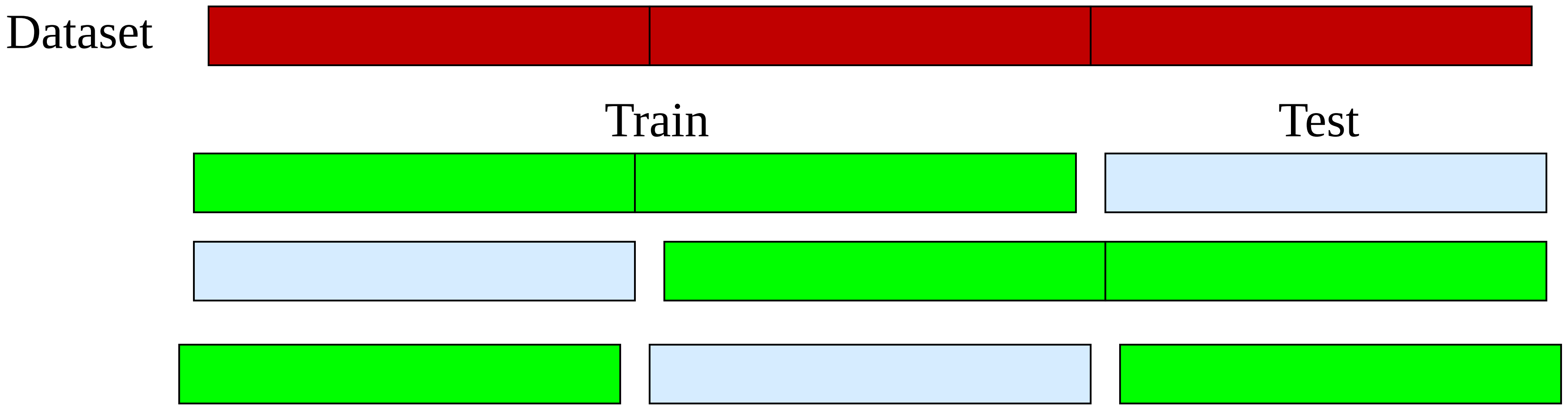
Common Splitting Strategies

- k-fold cross-validation

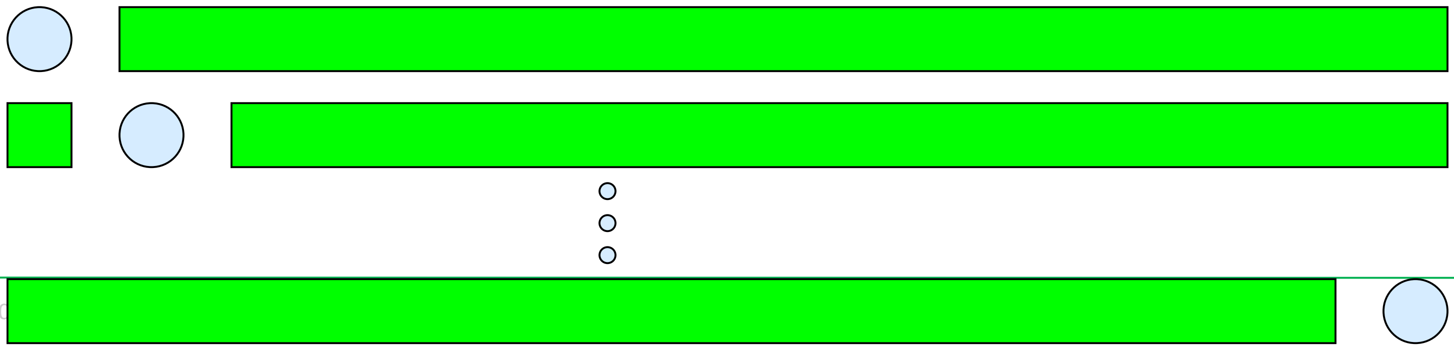


Common Splitting Strategies

- k-fold cross-validation



- Leave-one-out (n-fold cross validation)



References

Duda, R., Hart, P., Stork, D. Pattern Classification, 2nd ed. John Wiley & Sons,2001.

Y. LeCun, L. Bottou, G.B. Orr, and K.-R. Muller. Efficient Backprop. NeuralNetworks: Tricks of the Trade, Springer Lecture Notes in Computer Sciences,No. 1524, pp. 5-50, 1998.

P. Simard, D. Steinkraus, and J. Platt. Best Practice for Convolutional Neural Networks Applied to Visual Document Analysis. International Conference on Document Analysis and Recognition, ICDAR 2003, IEEE Computer Society, Vol. 2, pp. 958-962, 2003.

L. Tsu-Chang, Structure Level Adaptation for Artificial Neural Networks, KluwerAcademic Publishers, Boston, 1991.

Ramirez, J.M.; Gomez-Gil, P.; Baez-Lopez, D., "On structural adaptability of neural networks in character recognition," Signal Processing, 1996., 3rd International Conference on , vol.2, no., pp.1481-1483 vol.2, 14-18 Oct 1996.

THANK YOU