Introduction to Machine learning theory

Why learning

- Machine learning is programming computers to optimize a performance criterion using example data or past experience
  - Learning general models from a data of particular examples
  - Build a model that is *a good and useful approximation* to the data.
- Data is cheap and abundant (data warehouses, data marts); knowledge is expensive and scarce.

When learning

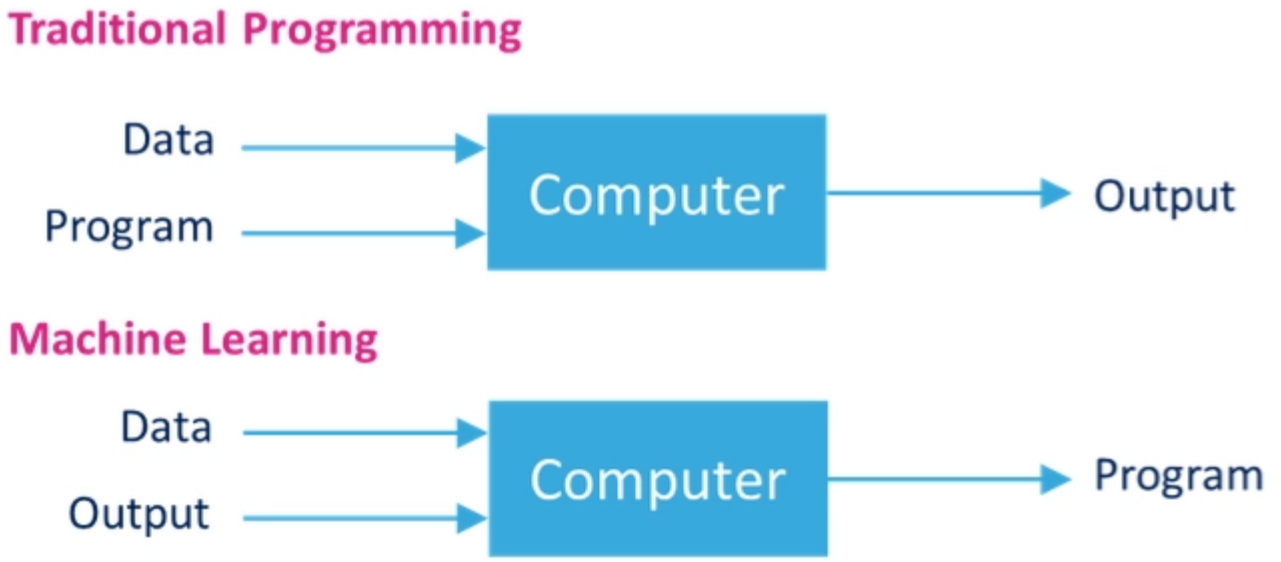
- Machine learning is suitable for problems where:
  - A pattern exists
  - We can not pin it down mathematically
  - We have data on it

- Learning is used when:
  - Human expertise does not exist (navigating on Mars)
  - Humans are unable to explain their expertise (speech recognition)
  - Solution changes in time (routing on a computer network)

Why learning

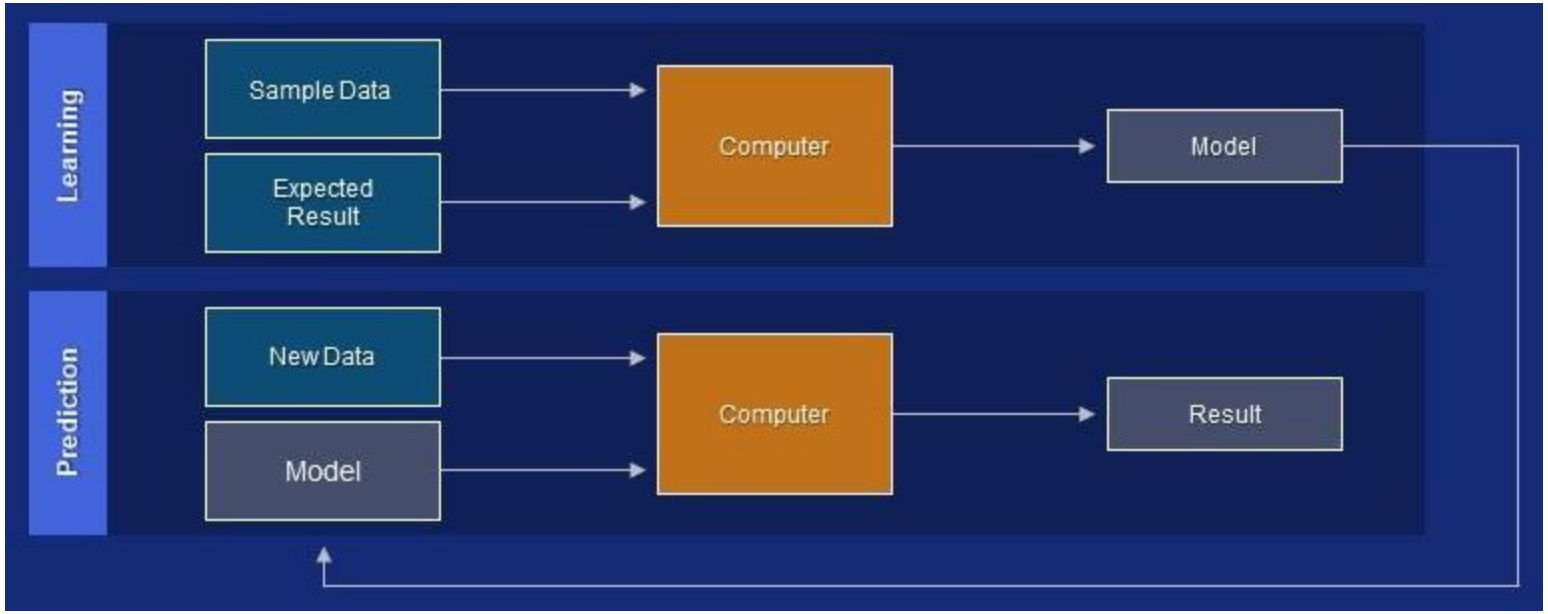
If you don't know how to solve a problem, write it out as an optimization problem

Learning vs Programming



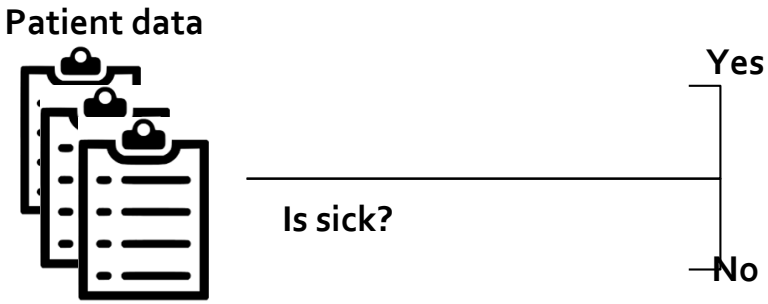
- Learning algorithm is implemented as a program

Learning vs Programming



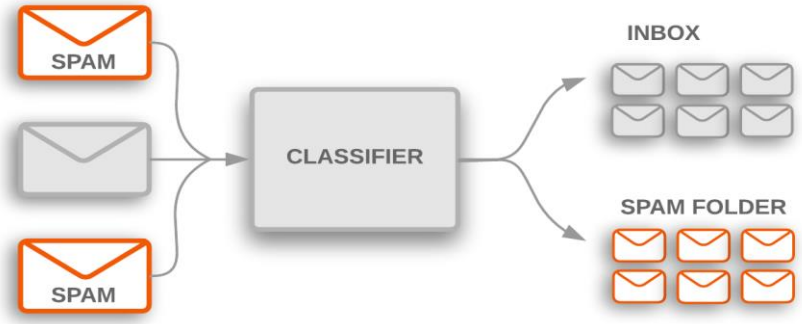
- The Learning algorithm and data are implemented as a program

Classification



Medical Diagnosis

Text Categorization



Spam detection

Classification assigns data items to target categories or classes

Classification

- The problem of classification is defined as:

- **Given**: A set of training data

  $(x_1, y_1), \ldots (x_n, y_n)$ where $x_i \; in \; \mathbb{R}^n$ and y_i is the class label

- **Find :** A classification function

  f: $\mathbb{R}^n \rightarrow \{c_1, \ldots, c_k\}$ which classifies well additional samples $\{x_k\}$

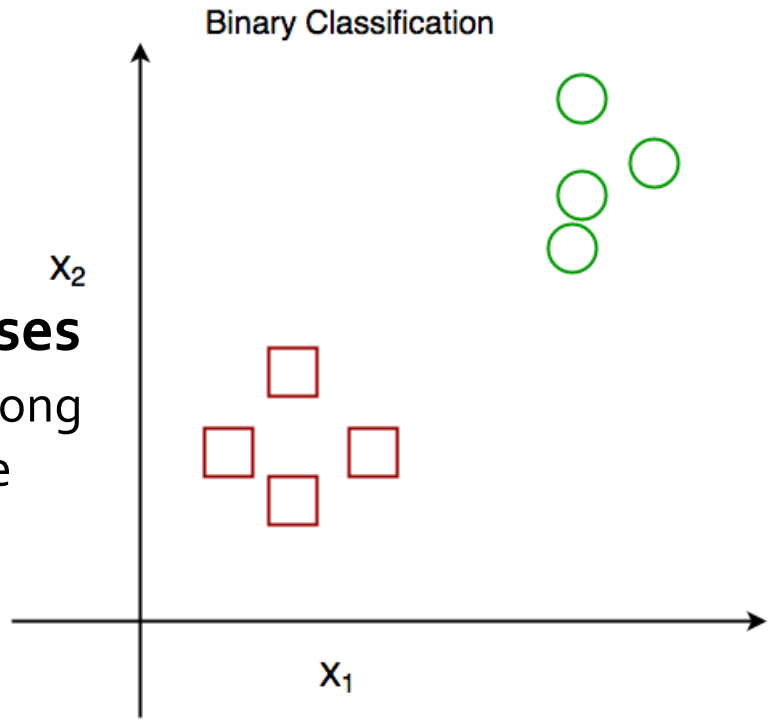
  k is the number of classes

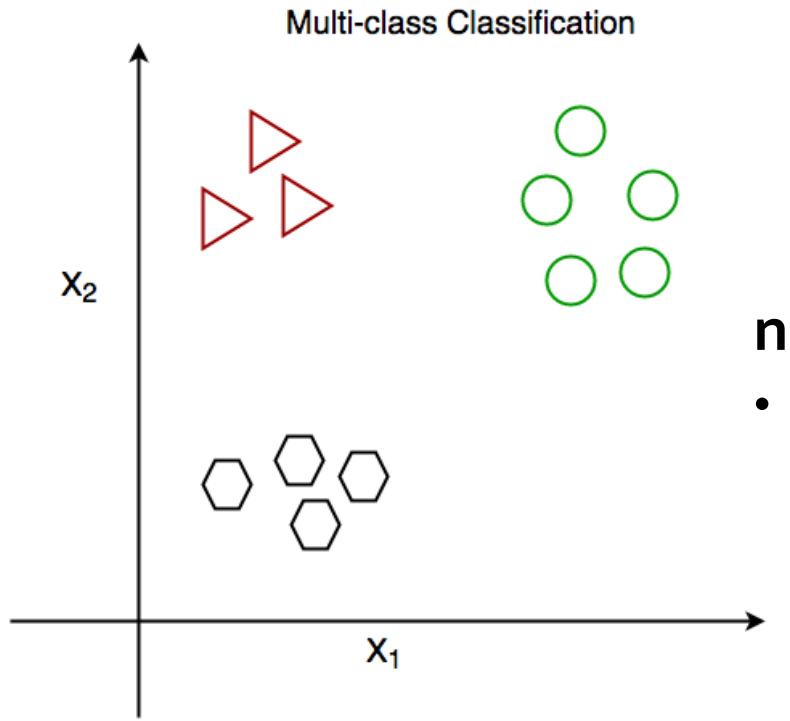
Classification

Binary vs multiclass classification

2 target classes

- Samples belong to one of the two classes

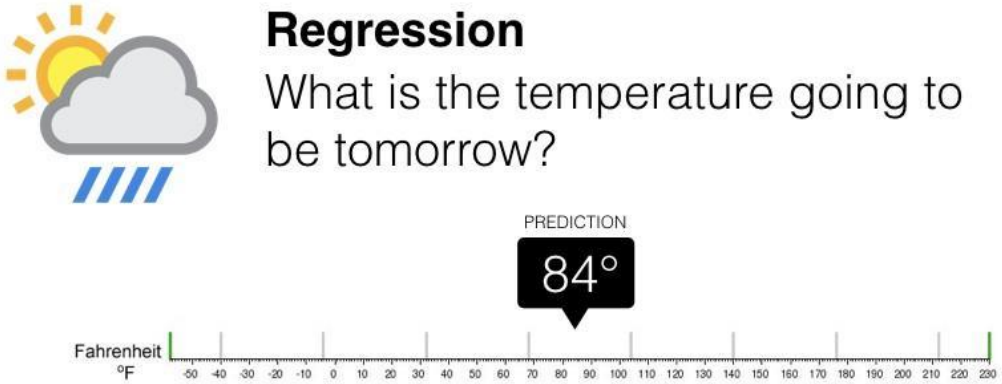




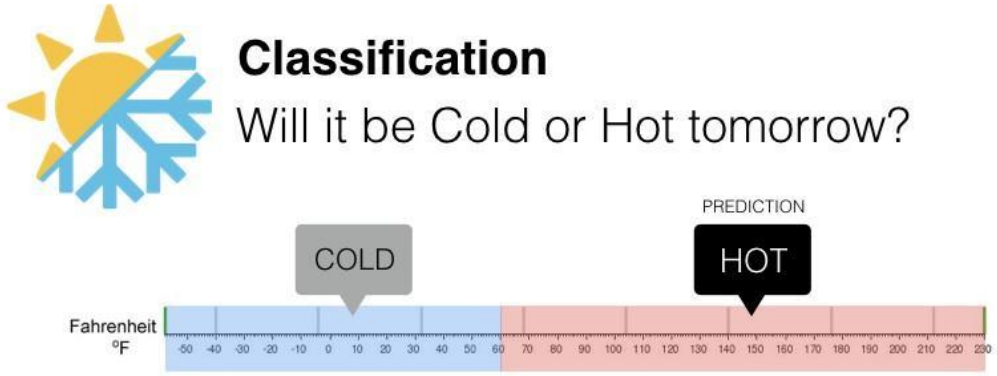
n target classes

- Samples belong to one of the n classes

Classification vs regression



Regression is the task of predicting a continuous quantity



Classification is the task of predicting a discrete class label

But, generally models return numbers (probability, ….)

ML: Classification methods

Approaches to learn classifiers/predictors: learning algorithms

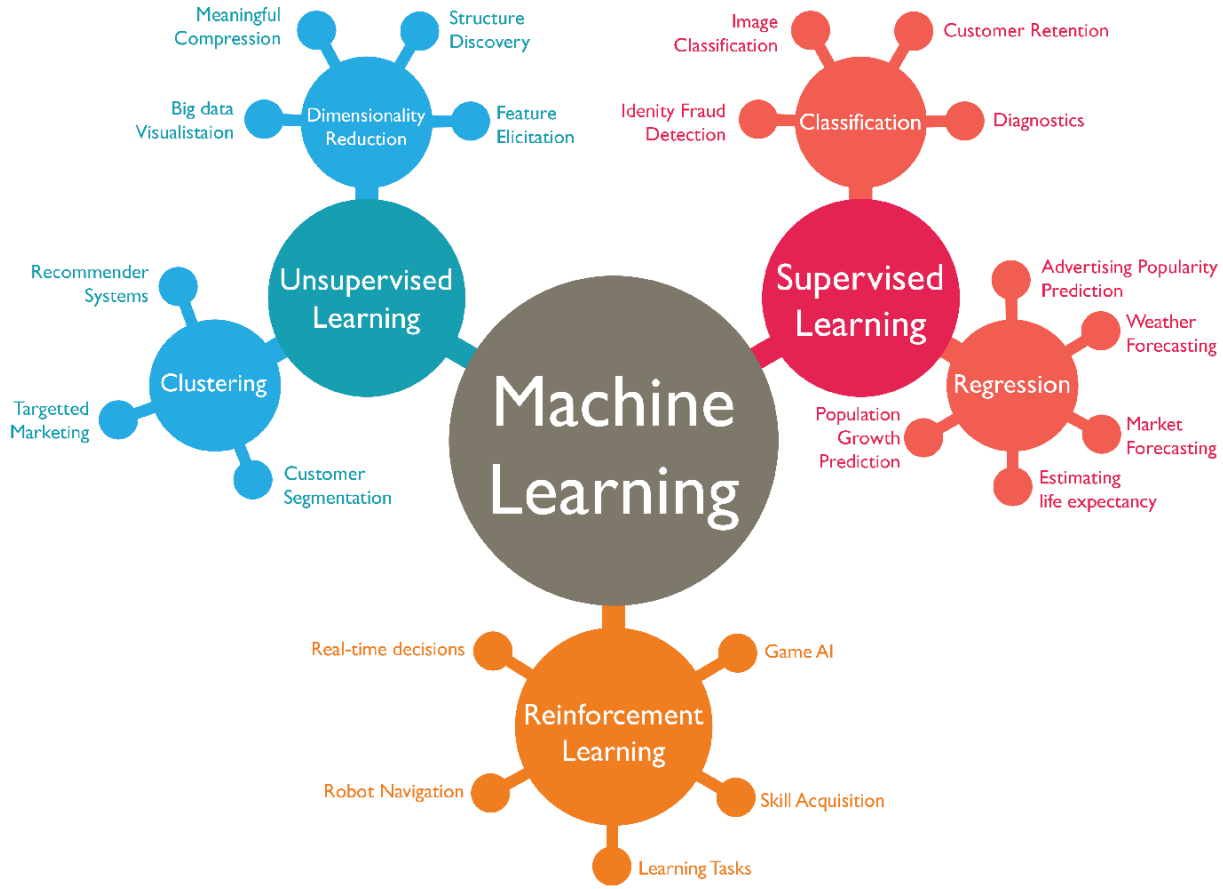
- Linear classifiers: linear Regression, Bayesian classifier
- Support Vector Machines (SVM)
- Decision trees
- Random Forest
- K-Nearest Neighbor
- Neural Networks
- …

What the matter between the different machine learning algorithms?

ML: Classification methods

Approaches to learn classifiers/predictors:

- Linear classifiers: linear Regression, Bayesian classifier
- Support Vector Machines (SVM)
- Decision trees
- Random Forest
- K-Nearest Neighbor
- Neural Networks
- …



What the matter between the different machine learning algorithms?

ML: Main components

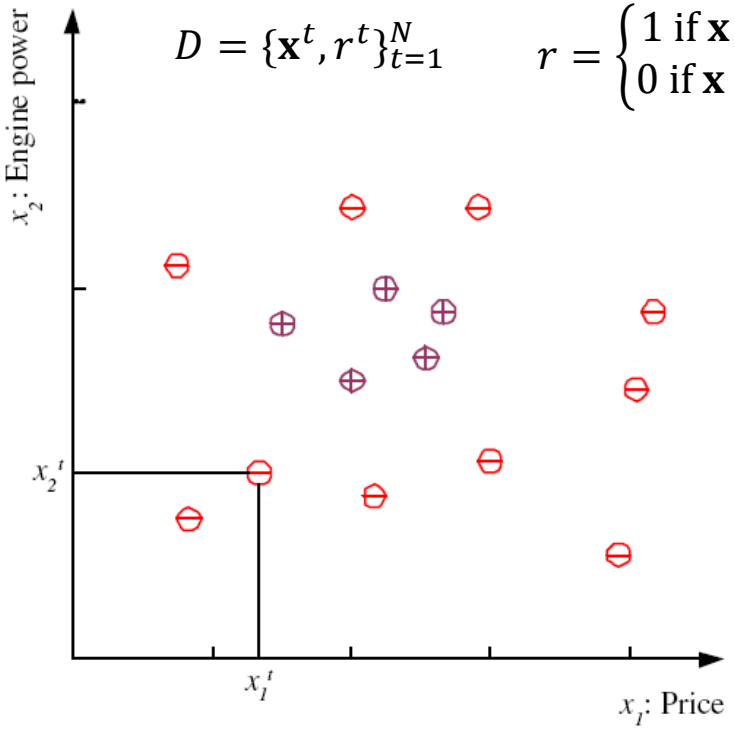
- Model representation (hypothesis space):
  - Structure of the functional form of the knowledge to be extracted (Trees, partition, graph,…)
- Search method (learning algorithm):
  - Strategy used to explore the search space and find the optimal or "good" model (backpropagation, local search, divide-and-conquer, greedy search, …)
- Objectif function (cost function):
  - Measure the quality of the model (Gini, Entropy, RMSE, logloss, …)

Reviewing the learning models

Hypothesis space (class) $\mathcal{H}$

- Let start with a sample problem of binary classification

$$D = \{\mathbf{x}^t, r^t\}_{t=1}^{N} \qquad r = \begin{cases} 1 \text{ if } \mathbf{x} \text{ is positive} \\ 0 \text{ if } \mathbf{x} \text{ is negative} \end{cases}$$

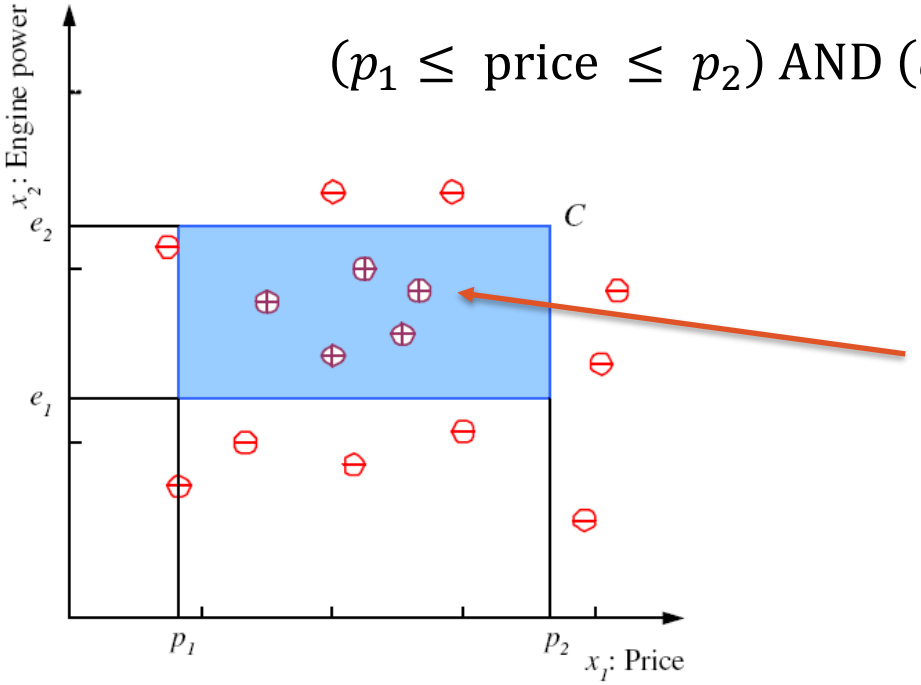


$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

- Prediction of "family car"
  - Prediction: Is car x a family car?
- Knowledge extraction:
  - What do people expect from a family car?
- Output:
  - Positive (+) and negative (–) examples
- Input representation:
  - x1: price, x2 : engine power

Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - Hypothesis is any axis aligned Rectangle. Inside rectangle is positive

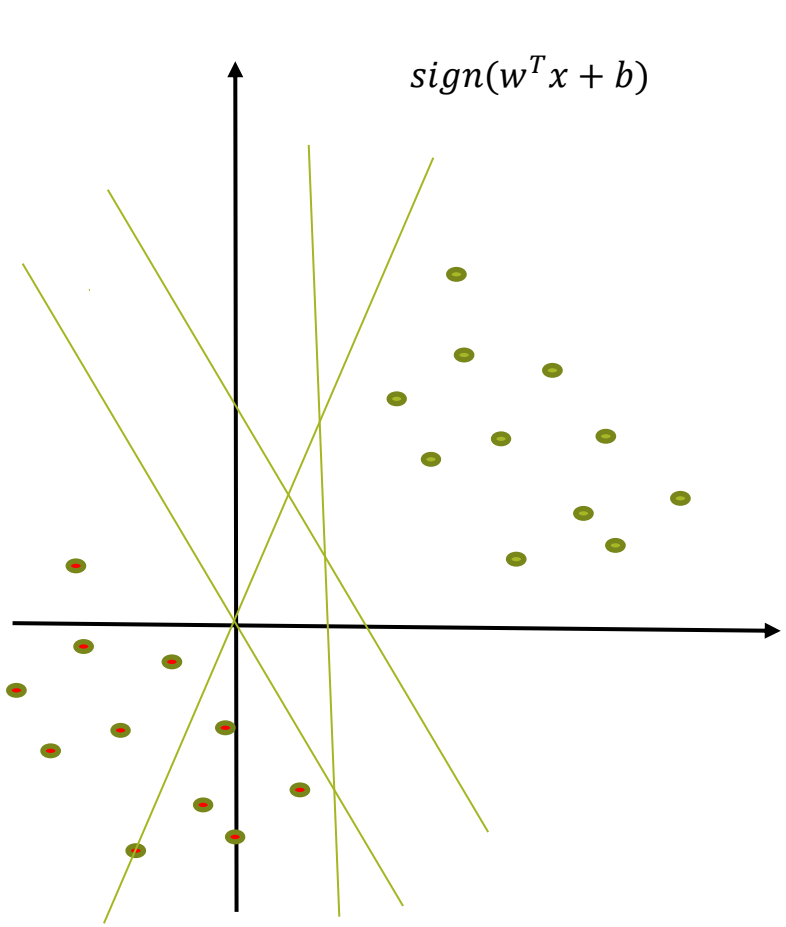


$$(p_1 \leq \text{price} \leq p_2) \text{ AND } (e_1 \leq \text{engine power} \leq e_2)$$

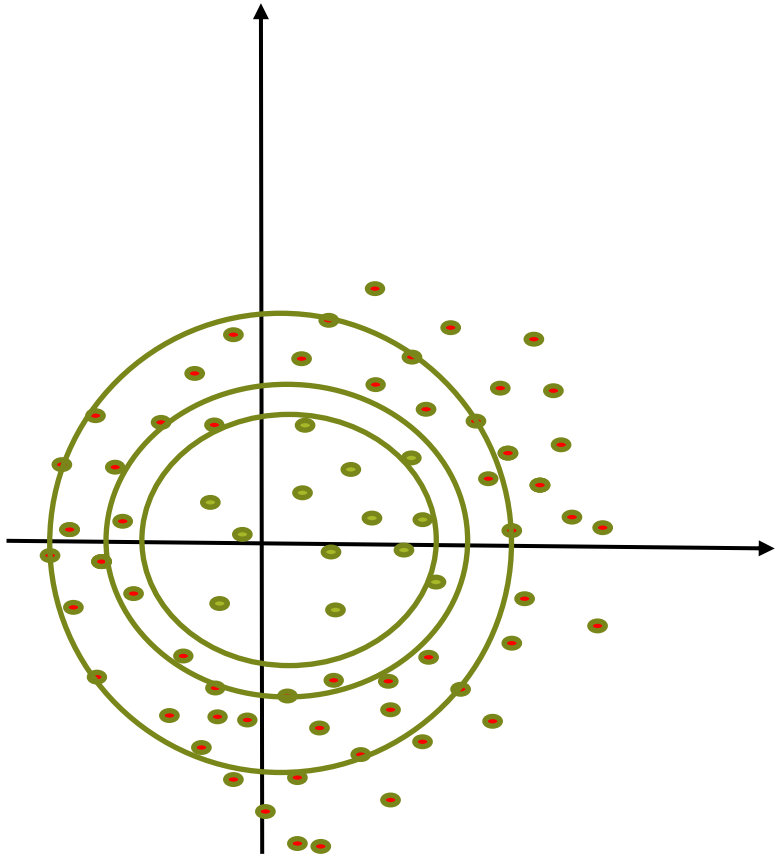
$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } h \text{ says } \mathbf{x} \text{ is positive} \\ 0 \text{ if } h \text{ says } \mathbf{x} \text{ is negative} \end{cases}$$

  - $\mathcal{H}$ is the set of all possible rectangle called the hypothesis space

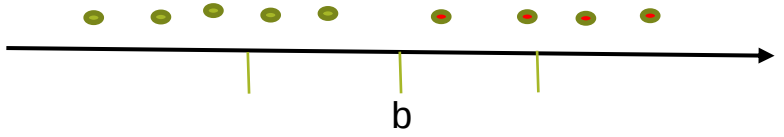
Hypothesis space (class) $\mathcal{H}$



$sign(w^T x + b)$



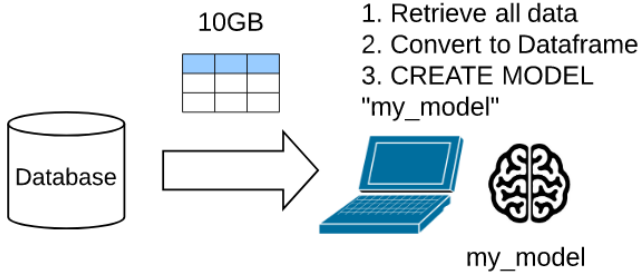
$sign(x^2 - b)$



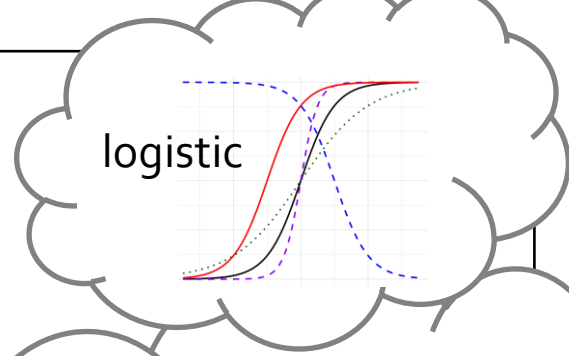
$sign(x - b)$

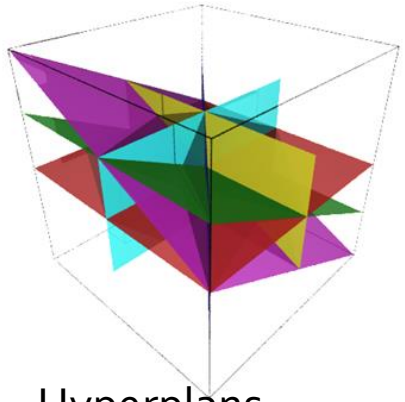
Hypothesis space $\mathcal{H}$

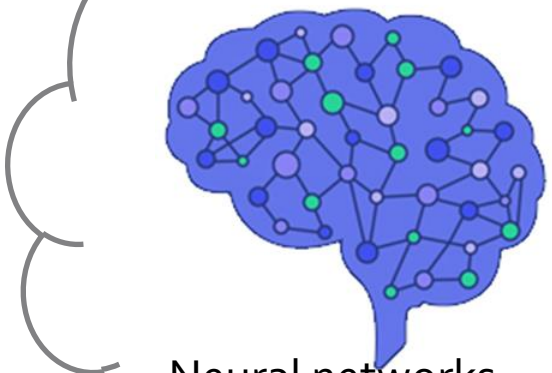
Which hypothesis space ?

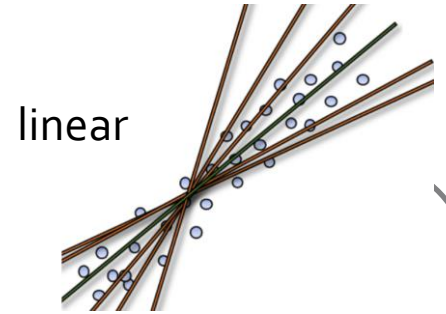


?



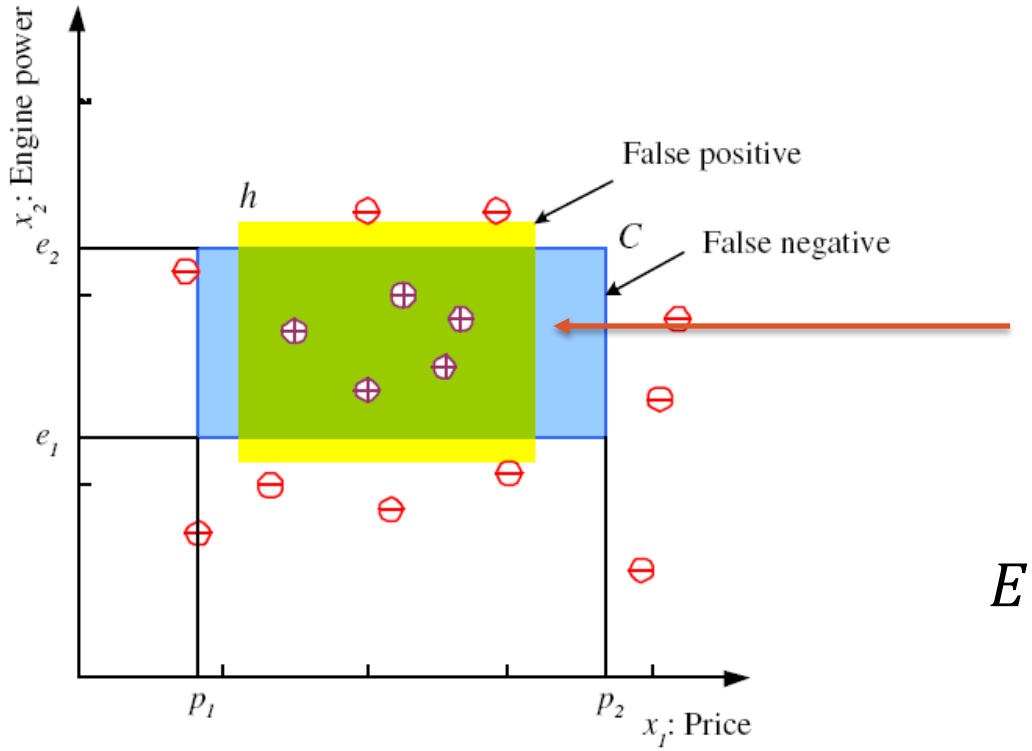






Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - Assume that *h* is given: what is the Error of *h* on $\mathcal{H}$?



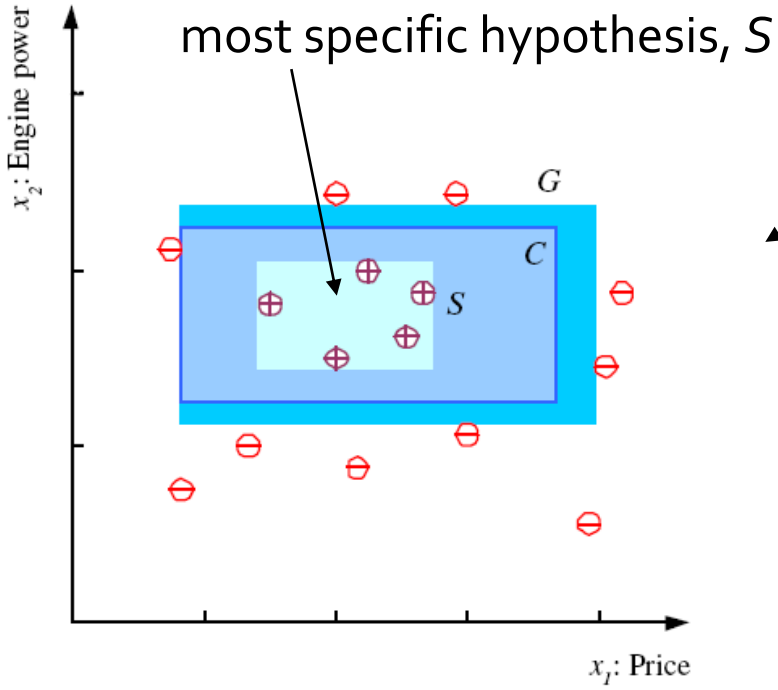
$$D = \{\mathbf{x}^t, r^t\}_{t=1}^N \subset X$$

$$h(\mathbf{x}) = \begin{cases} 1 \text{ if } h \text{ says } \mathbf{x} \text{ is positive} \\ 0 \text{ if } h \text{ says } \mathbf{x} \text{ is negative} \end{cases}$$

$$E(h|D) = \sum_{t=1}^{N} 1(h(\mathbf{x}^t) \neq r^t)$$

Hypothesis space $\mathcal{H}$

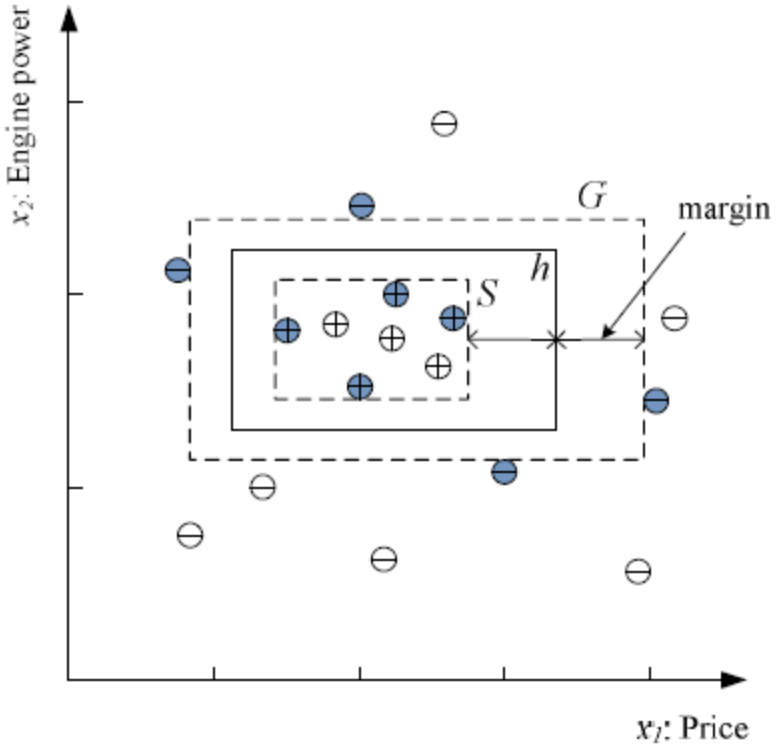
- Rectangle learning:
  - The version space (VS) with the respect to a hypothesis space $\mathcal{H}$ and the training set D is the subset of hypothesis from $\mathcal{H}$ consistant with the training set D



$h \in \mathcal{H}$, between S and G is **Consistent ($E(h \mid D) = 0$) and** make up the version space (Mitchell, 1997)

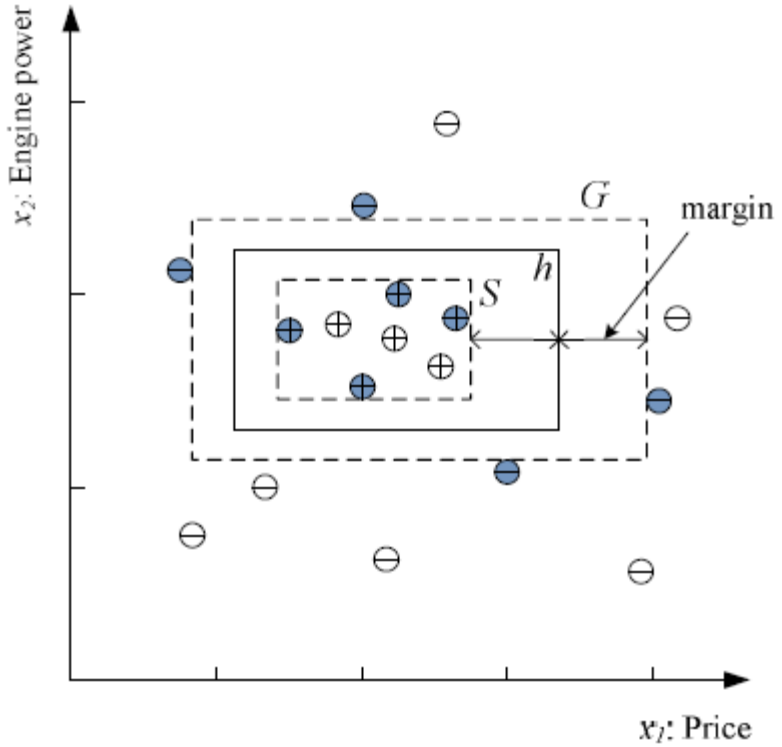
Hypothesis space $\mathcal{H}$

- Rectangle learning:
  - To generalize better, choose *h* with largest margin



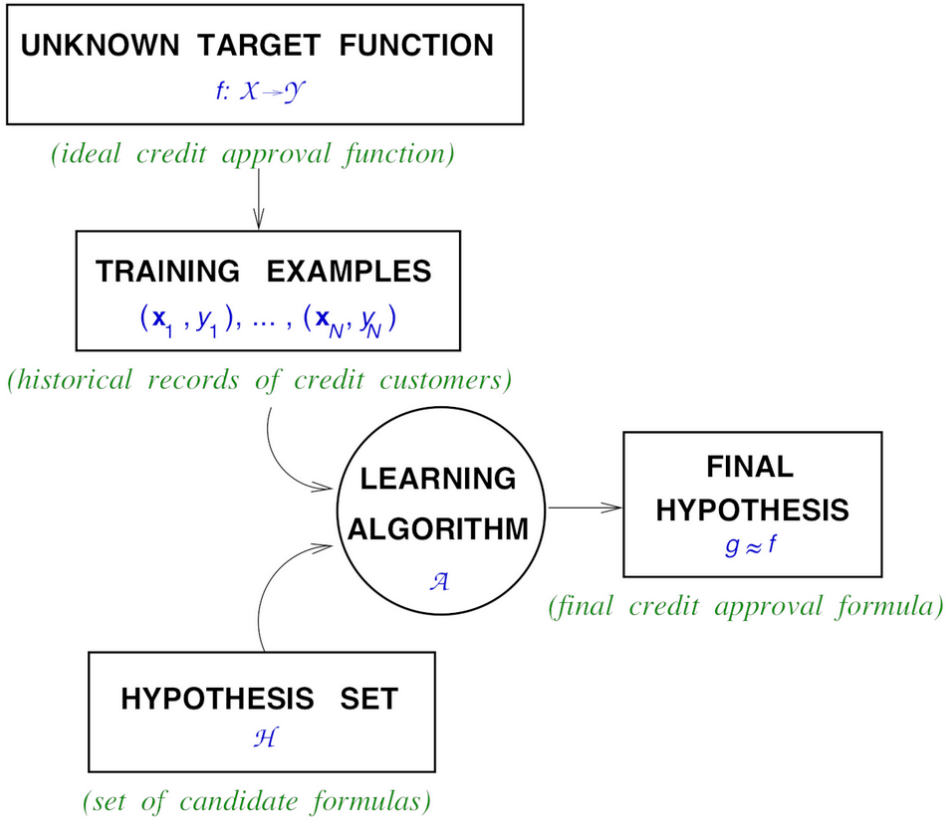
Hypothesis space $\mathcal{H}$

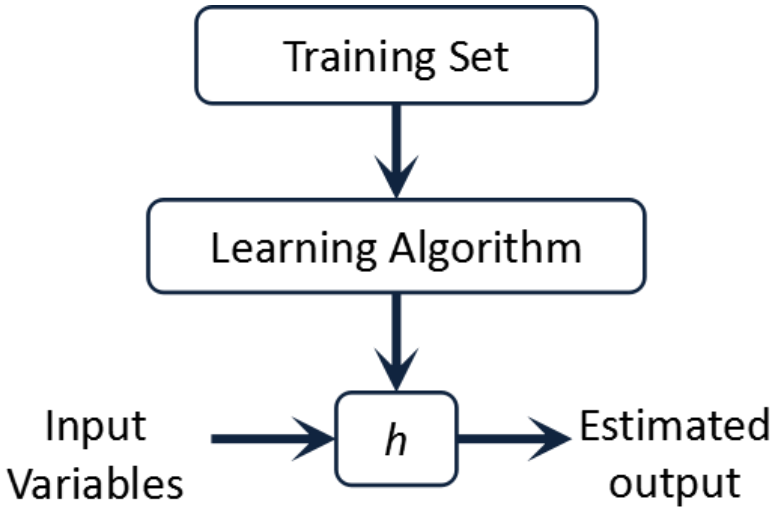
- Rectangle learning:
  - But for a given dataset X, how to compute h? **Learning algorithm (or learner)**



Hypothesis space $\mathcal{H}$

- Each hypothesis space has it learning algorithm
  - The hypothesis set and learning algorithm together, is called a **Learning Model**





Hypothesis space $\mathcal{H}$

- The hypothesis set and learning algorithm together, is called a **Learning Model**
  - **Question 1: what is the best learning model?**
  - **Question2: how good is a learning model, given dataset ?**
  - **Question 3: what is the capacity (complexity, expressive power, richness, or flexibility) of a learning model ?**

Q1: No Free Launch (NFL) theorem

- In his 1996 paper, "**The Lack of A Priori Distinctions Between Learning Algorithms"**, Wolpert introduced the NFL theorem for supervised machine learning.

" The theorem states that given a noise-free dataset, for any two machine learning algorithms A and B (learning model), the average performance of A and B will be the same across all possible problem instances drawn from a uniform probability distribution"

Q1: No Free Launch (NFL) theorem

- Does it mean that all algorithms are equal? No, of course not.
  - **Some algorithms may generally perform better than others on certain types of problems, but every algorithm has disadvantages and advantages due to the prior assumptions that come with that algorithm.**
  - XGBoost may win hundreds of Kaggle competitions yet fail miserably at forecasting tasks because of the limiting assumptions involved in tree-based models.
  - Neural networks may perform really well when it comes to complex tasks like image classification and speech detection, yet suffer from overfitting due to their complexity if not trained properly.

Q1: No Free Launch (NFL) theorem

- In practice, this is what "no free lunch" means for you:
  - No single algorithm will solve all your machine learning problems better than every other algorithm.
  - Make sure you completely understand a machine learning problem and the data involved before selecting an algorithm to use.
  - All models are only as good as the assumptions that they were created with and the data that was used to train them.
  - Simpler models like logistic regression have more bias and tend to underfit, while more complex models like neural networks have more variance and tend to overfit.
  - The best models for a given problem are somewhere in the middle of the two bias-variance extremes.
  - To find a good model for a problem, you may have to try different models and compare them using a robust cross-validation strategy.

Q1: Universal Approximation Theorem.

- A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.

Universal Approximation Theorem: Fix a continuous function $\sigma : \mathbb{R} \to \mathbb{R}$ (activation function) and positive integers d, D. The function σ is not a polynomial if and only if, for every continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$ (target function), every compact subset K of $\mathbb{R}^d$, and every $\epsilon > 0$ there exists a continuous function $f_\epsilon : \mathbb{R}^d \to \mathbb{R}^D$ (the layer output) with representation

$$f_\epsilon = W_2 \circ \sigma \circ W_1,$$

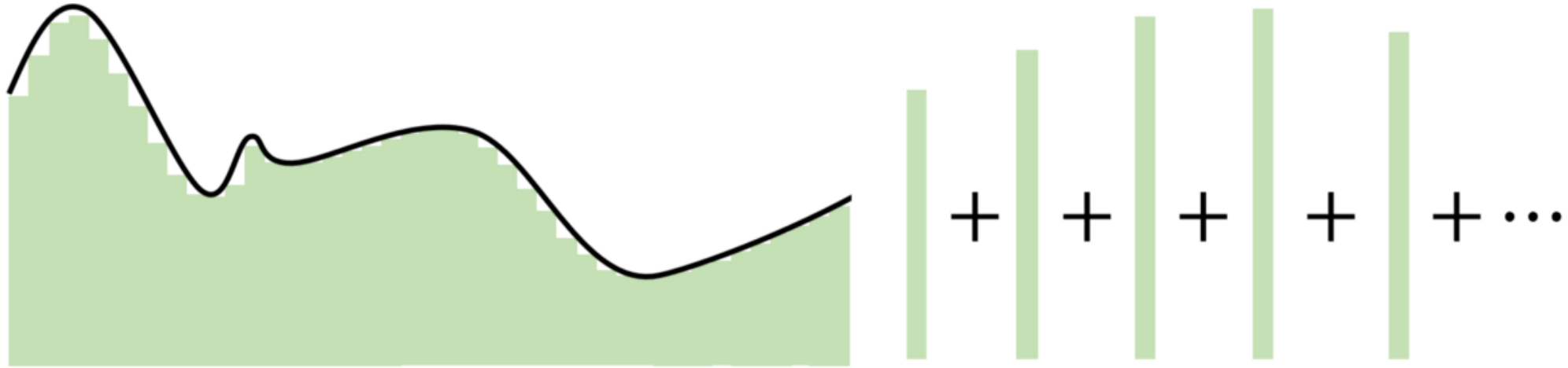
where W_2, W_1 are composable affine maps and $\circ$ denotes component-wise composition, such that the approximation bound

$$\sup_{x \in K} \|f(x) - f_\epsilon(x)\| < \varepsilon$$

holds for any ϵ arbitrarily small (distance from f to f_ϵ can be infinitely small).

Q1: Universal Approximation Theorem.

- The key point in the Universal Approximation Theorem is that instead of creating complex mathematical relationships between the input and output, it uses simple linear manipulations to divvy up the complicated function into many small, less complicated pieces, each of which are taken by one neuron.



Q1: Universal Approximation Theorem.

- The number of hidden neurons should be between the size of the input layer and the output layer.
  - The most appropriate number of hidden neurons is sqrt(input layer nodes * output layer nodes)

Q1: Universal Approximation Theorem.

- A model with too little capacity cannot learn the problem, whereas a model with too much capacity can learn it too well and overfit the training dataset. Both cases result in a model that does not generalize well.
  - The capacity of a neural network model is defined by both it's structure in terms of nodes and layers and the parameters in terms of its weights. Therefore, we can reduce the complexity of a neural network to reduce overfitting in one of two ways:
    - Change network complexity by changing the network structure (number of weights): grid search until a suitable number of nodes and/or layers
    - Change network complexity by changing the network parameters (values of weights): keeping network weights small

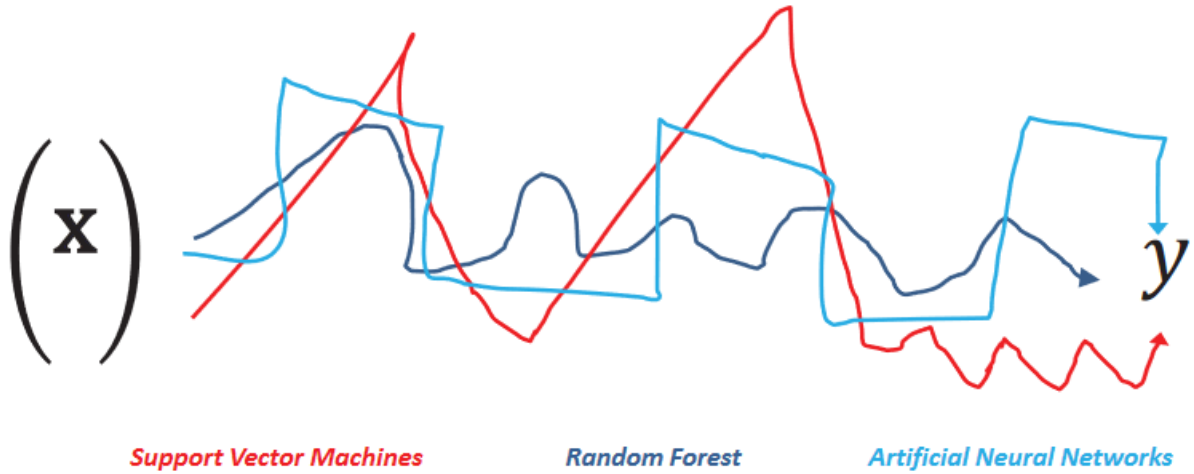
Hypothesis space $\mathcal{H}$

- The hypothesis set and learning algorithm together, is called a **Learning Model**
  - Question 1: what is the best learning model?
  - **Question 2: how good is a learning model, given a dataset ?**
  - Question 3: what is the capacity (complexity, expressive power, richness, or flexibility) of a learning model ?

ML: Learners

Which learning algorithms? Approaches to learn classifiers/predictors

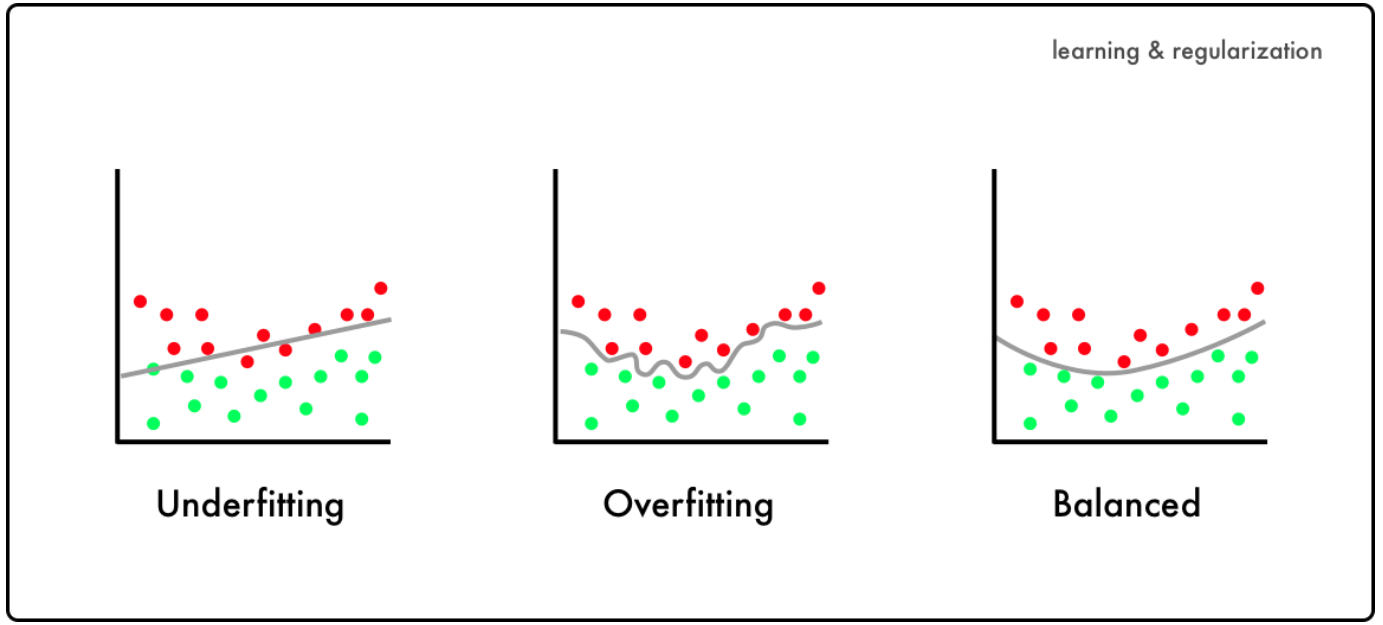
$$\left(\mathbf{x}\right) \xrightarrow{f(\mathbf{X},\alpha)\ ?} y$$



ML: Objective function

- At the end of the day, the learning algorithm in a given hypothesis space will give me a candidate model
- If you look too hard at a dataset, you will find something, but it might not generalize beyond the data you're looking at (unseen data) = Overfitting

But
HOW GOOD IS YOUR MODEL



ML: Objective function

- Cost function quantifies the error between predicted values and expected values and **presents it in the form of a single real number**.
- We have various measures of model error (any norm) depending on the hypothesis space
- Mean Absolute Error (MAE) $$MAE = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}|$$
- Mean Square Error (MSE) $$MSE = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2$$
- Root Mean Square Error (RMSE) $$RMSE = \sqrt{MSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y})^2}$$

ML: Objective function

- Building a model is an optimization problem
- Machine learning goal
  - learn $f(x)$ such that the cost function $J(y_i, f(x_i))$ is minimized
- For parametric models:
  - find the optimal configuration of model parameters w_i that minimizes the cost function J
- Example of linear regression ($f_w(x) = w^T x + w_0$)

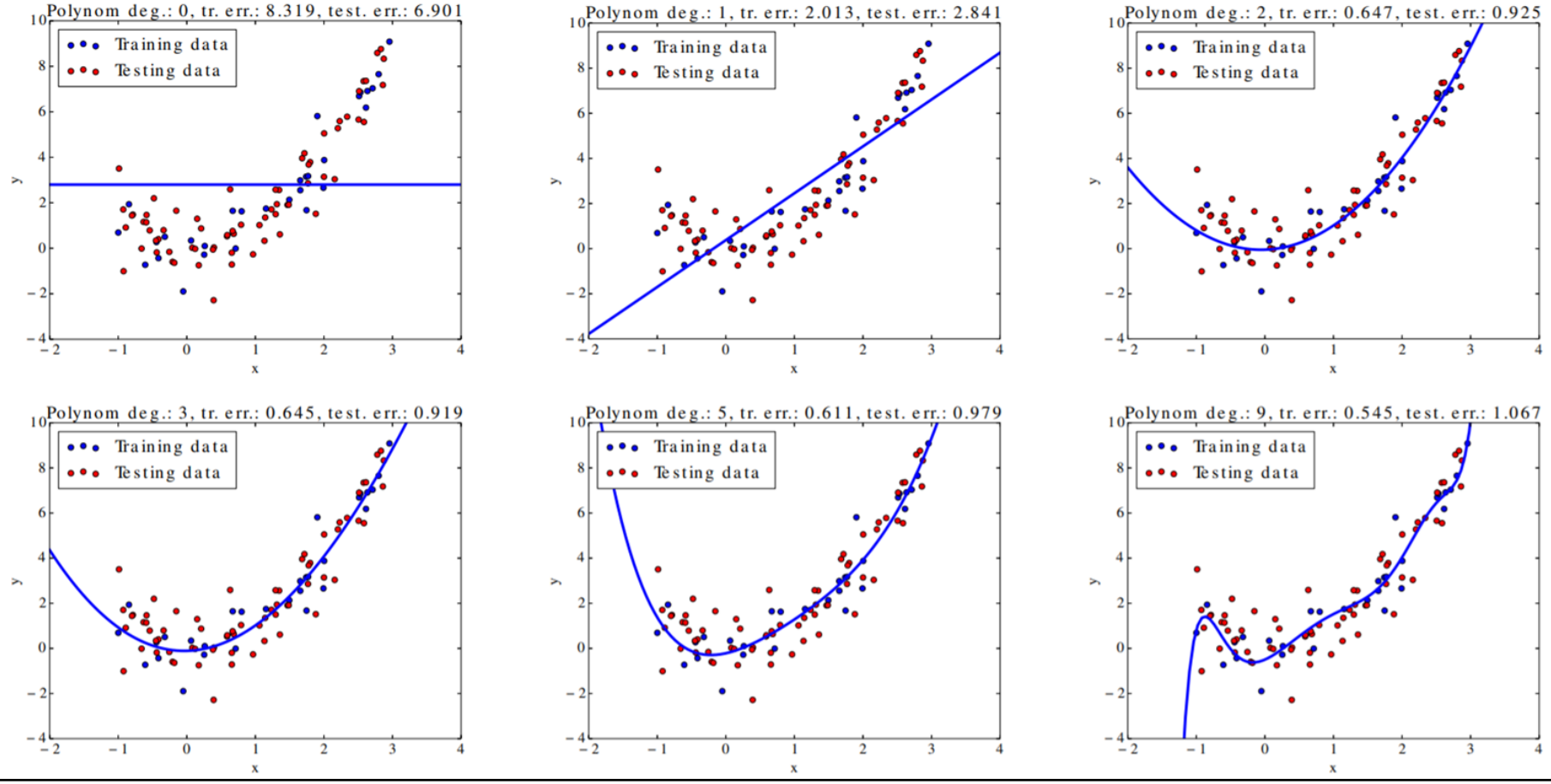
$$\min_{w,w_0} J(w, w_0) = \frac{1}{N} \sum_{i=1}^{N} (f_w(x_i) - yi)^2$$

Training and testing error

- Example: Polynomial regression with varying degree

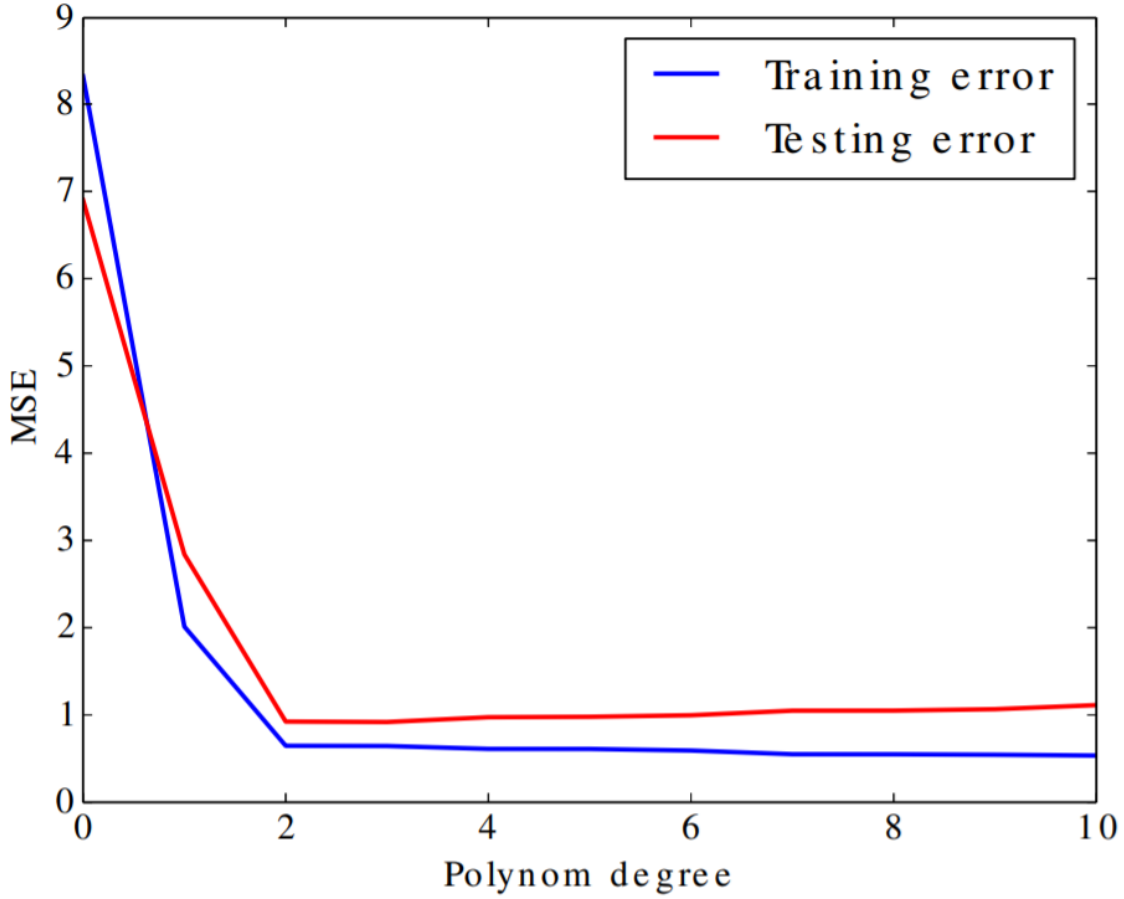
$$X \sim U(-1,3)$$

$$Y \sim X^2 + N(0,1)$$



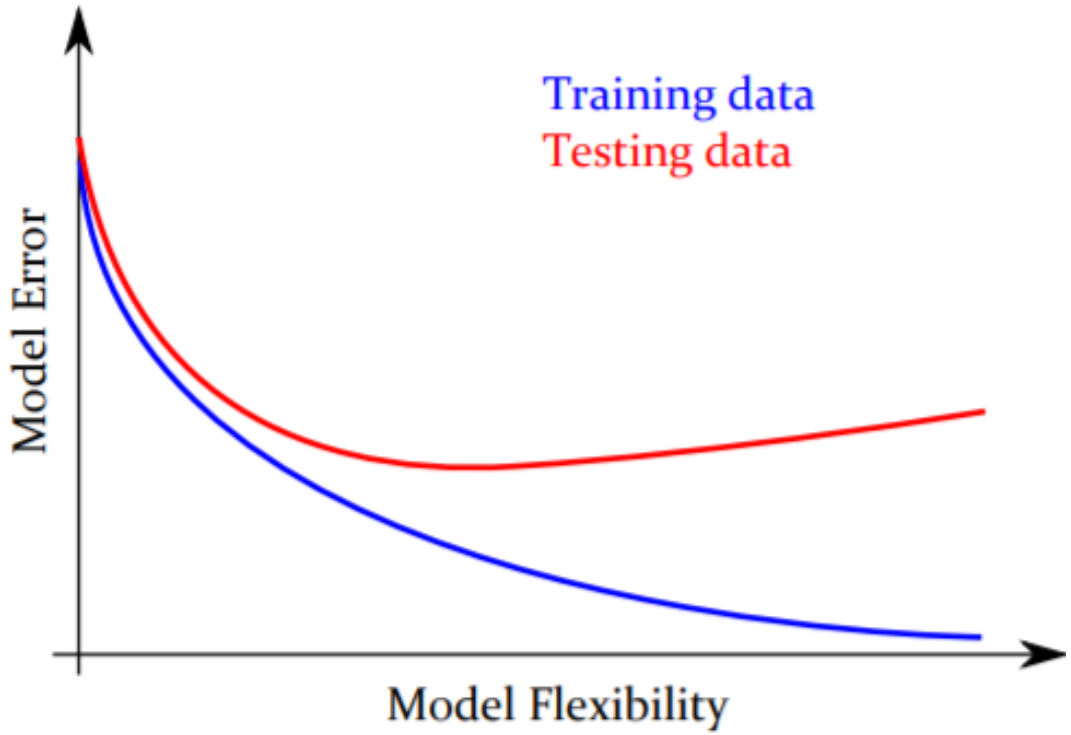
Training and testing error

- The training error decreases with increasing model flexibility (the curviness).
- The testing error is minimal for certain degree of model flexibility.



Overfitting

- Overfitting is a general phenomenon affecting all kinds of inductive learning.
- When overfitted, the model works well for the training data, but fails for new (testing) data.



Bias vs Variance

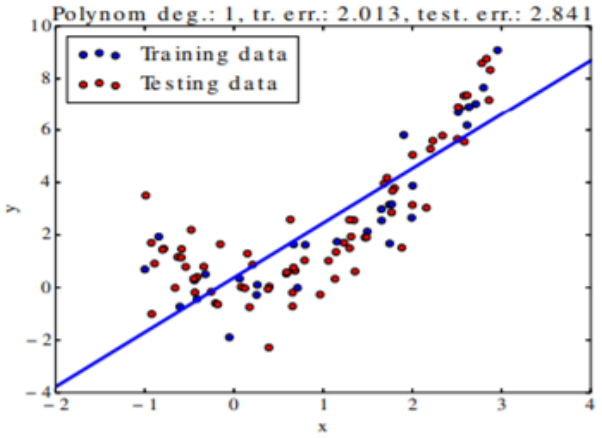
- The bias error is an error from erroneous assumptions in the learning algorithm. High bias can cause an algorithm to miss the relevant relations between features and target outputs (underfitting).

$$\text{Bias} = E[\hat{\theta}] - \theta$$

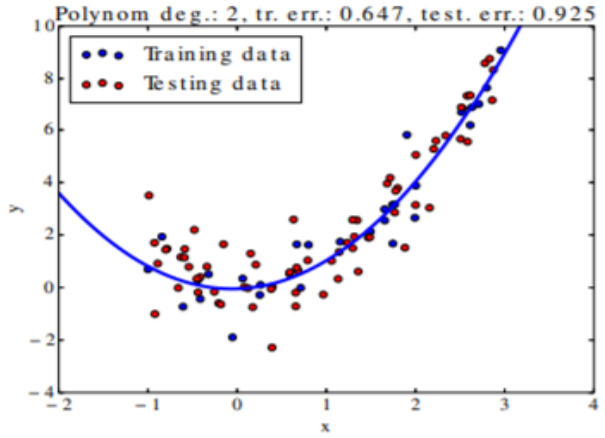
- The variance is an error from sensitivity to small fluctuations in the training set. High variance may result from an algorithm modeling the random noise in the training data (overfitting).

$$\text{Var}(\hat{\theta}) = E[\hat{\theta}^2] - \left(E[\hat{\theta}]\right)^2 \qquad \text{Var}(\hat{\theta}) = E[(E[\hat{\theta}] - \hat{\theta})^2]$$

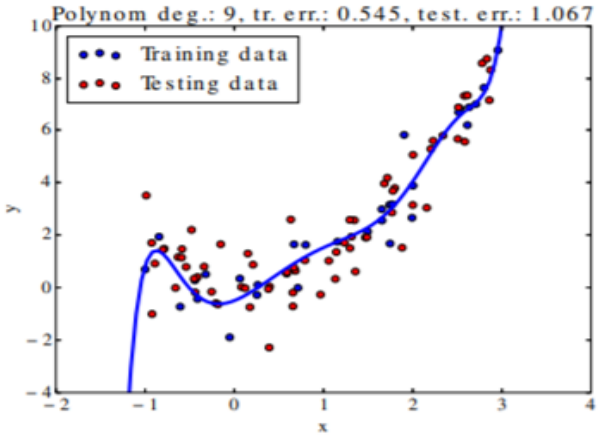
Bias vs Variance



High bias:
model not flexible enough
(Underfit)



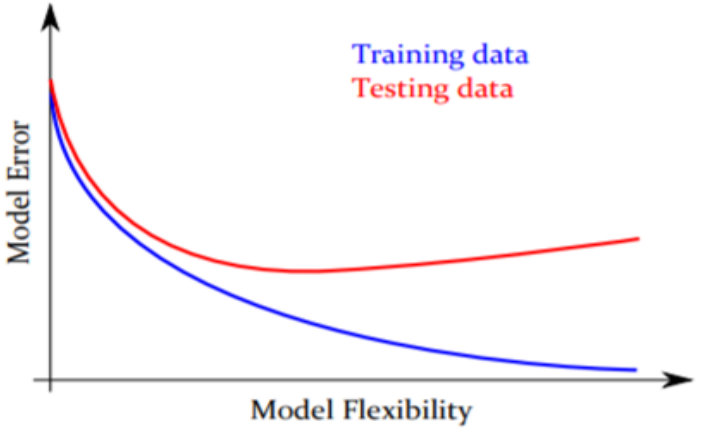
"Just right"
(Good fit)



High variance:
model flexibility too high
(Overfit)

High bias problem:

- Err_{Tr} is high
- $Err_{Tst} \approx Err_{Tr}$



High variance problem:

- Err_{Tr} is low
- $Err_{Tst} >> Err_{Tr}$

Bias – Variance Tradeoff

- Simple example

$$
\begin{aligned}
MSE(\hat{\beta}) &= \mathbb{E}((\hat{\beta}-\beta)^T(\hat{\beta}-\beta)) \\
&= \mathbb{E}(\hat{\beta}-\beta)\mathbb{E}((\hat{\beta}-\beta)^T) + \mathrm{Var}(\hat{\beta}) \\
&= ||\mathrm{Bias}(\hat{\beta})||^2 + \mathrm{Var}(\hat{\beta})
\end{aligned}
$$

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y})^2$$

Suppose that

$$Y \sim N(\mu, \sigma^2)$$

and the estimator $\hat{\mu} = \alpha Y$ where $0 \leq \alpha \leq 1$.

The bias is $\mathbb{E}(\hat{\mu}) - \mu = (\alpha - 1)\mu$. The variance is $\sigma^2\alpha^2$.

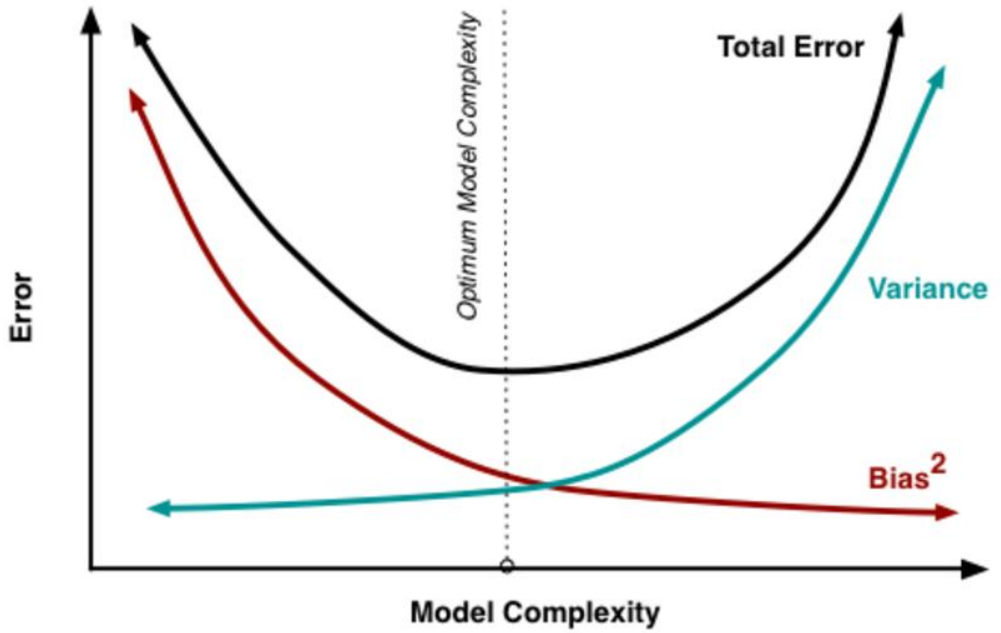
bias^2 + *variance* $= (\alpha - 1)^2\mu^2 + \sigma^2\alpha^2$

For $\alpha \to 0$, the bias increases and the variance decreases.

The optimal is $\alpha^* = \mu^2/(\sigma^2 + \mu^2)$

Regularization

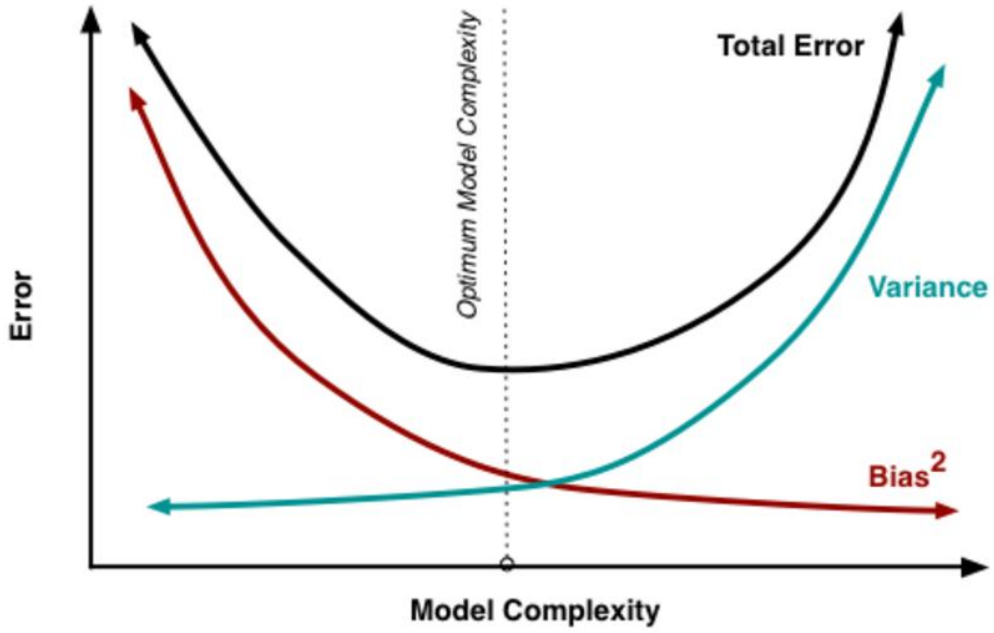
- Who to deal with overfitting ?



Regularization

- Reduce the model complexity
  - Reduce the width of the tree, size, …
  - Reduce the number of parameters (number of neurons, layers, …)
  - Reduce the values of parameters (small values, zeros, …)
- In parameterized model, a regularization technique is a penalty mechanism which applies shrinkage (driving them closer to zero) of coefficient to build a more robust and parsimonious model.

- Reduce model variance at the cost of introducing some bias.

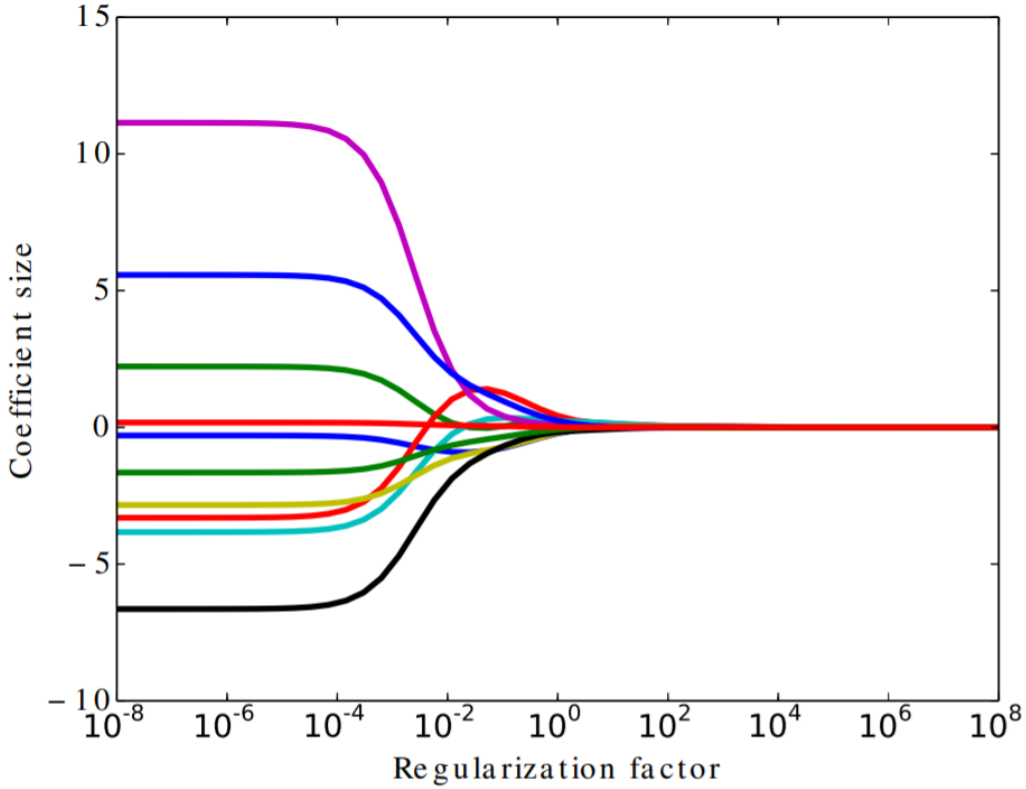


Ridge regularization (L2 regularization)

- Ridge regularization imposes a penalty on the size of the model coefficients.
- The penalty is equal to the sum of the squared value of the coefficients w_i

$$Error_{Ridge} = Error + \alpha \sum_{i=1}^{d} {w_i}^2$$

- α is the regularization parameter
- Ridge forces **the parameters to be relatively small**
- The bigger the penalization, the smaller (and the more robust) the coefficients are.



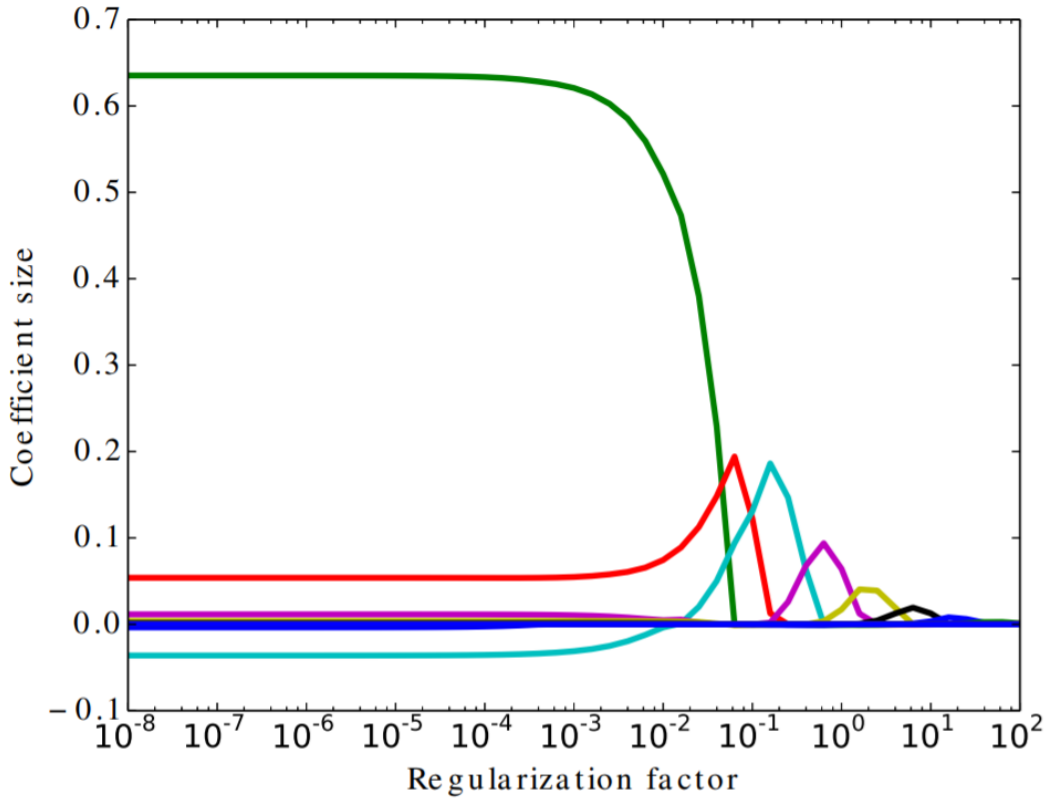
- $\alpha \to 0, \quad w^{ridge} \to w^{Error}$
- $\alpha \to \infty, \quad w^{ridge} \to 0$

Lasso regularization (L1 regularization)

- Like ridge, Lasso regularization penalizes the size of the model coefficients.
- The penalty is equal to the sum of the absolute value of the coefficients w_i

$$Error_{Lasso} = Error + \alpha \sum_{i=1}^{d} |w_i|$$

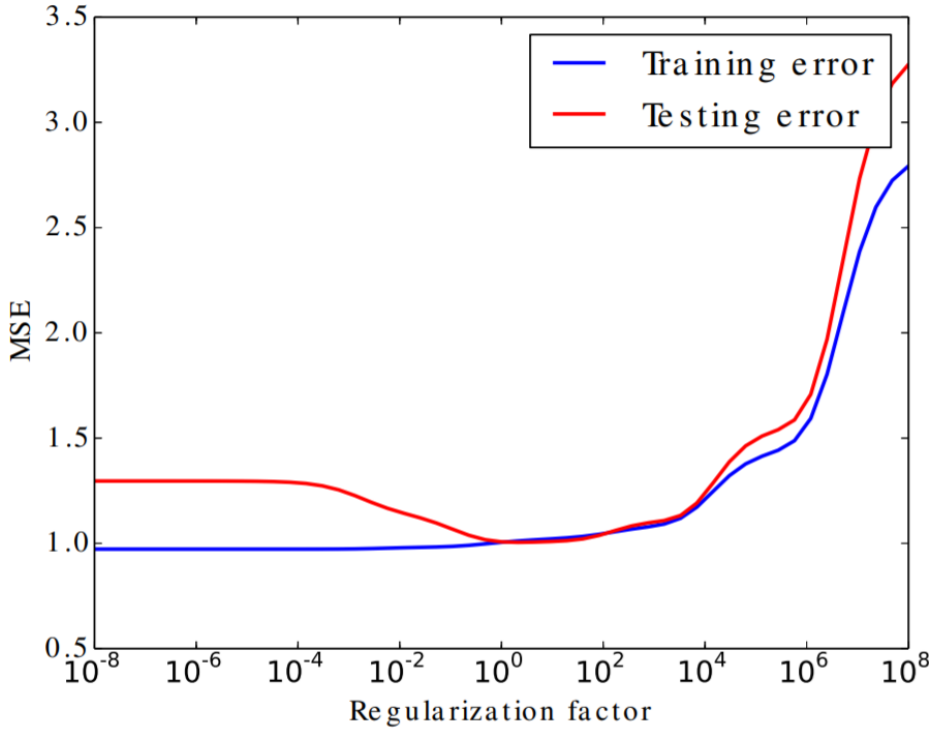
- Lasso regularization will **shrink some parameters to zero**.
- It can be seen as a way to select features in a model.



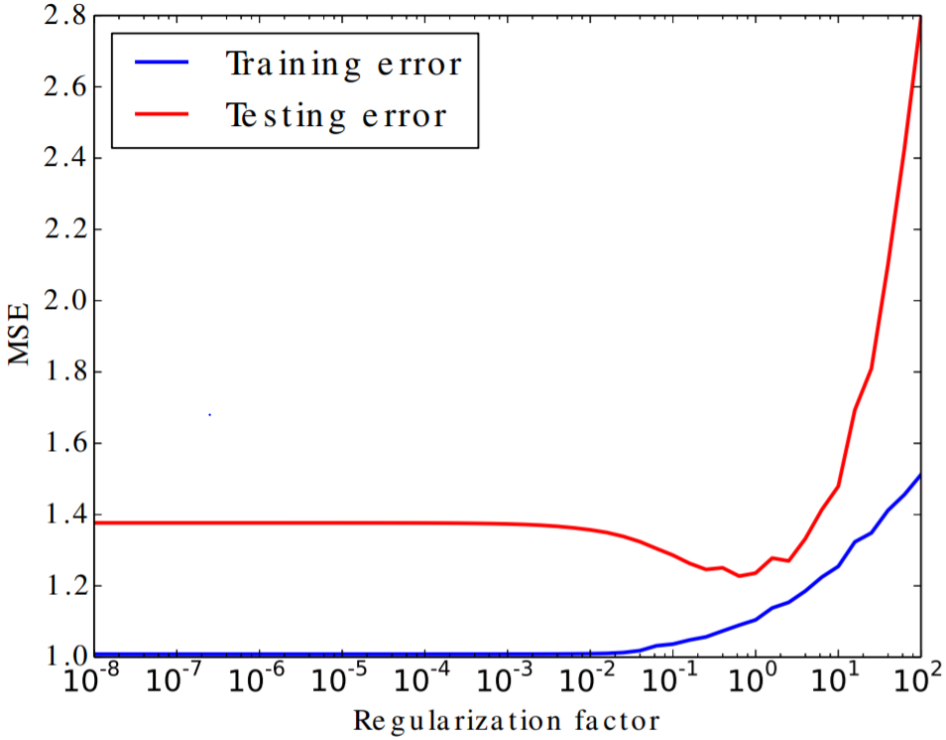
- $\alpha \to 0, \quad w^{Lasso} \to w^{Error}$
- $\alpha \to \infty, \quad w^{Lasso} = 0$

Ridge vs Lasso

- Evolution of error vs regularization factor



Ridge (L2 regularization)



Lasso (L1 regularization)

When the LASSO fails?

- In the d > n case, the lasso can select at most n variables before it saturates (d: number of paramters and n: number of samples)
- If there is a group of variables with very high pairwise correlations, the lasso tends to select only one variable from the group, not caring which one
- For usual n > d situations, if there are high correlations between variables, the prediction performance of lasso is poor with respect to ridge.

Elastic-net

- The lasso sometimes does not perform well with highly correlated variables, and often performs worse than ridge in prediction
- Elastic-net is a mix of **both Ridge and Lasso regularizations**.

$$Error_{Elastic} = Error + \alpha\,(\rho \sum_{i=1}^{d} |w_i| + \left(\frac{1-\rho}{2}\right) \sum_{i=1}^{d} {w_i}^2)$$

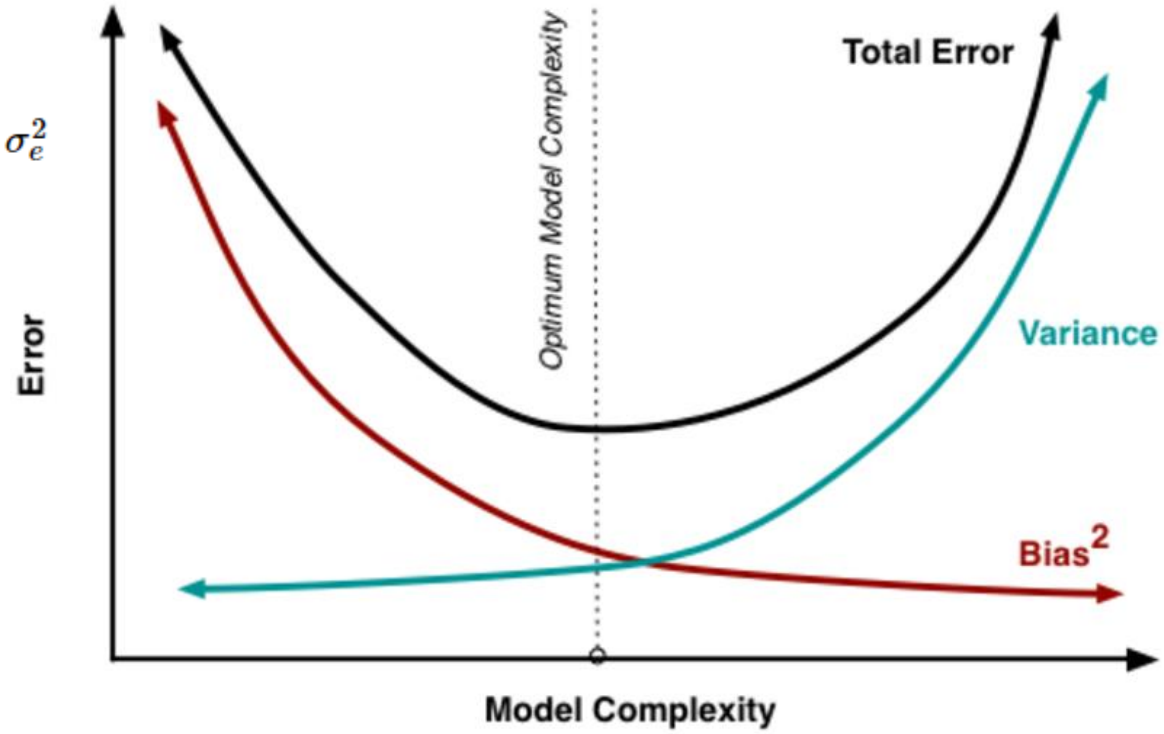
- α is a shared penalization parameter
- ρ is a mixing parameter, it sets the ration between ridge (L2) and lasso (L1) regularization
  - $\rho = 0$: Ridge regularization
  - $\rho = 1$: Lasso regularization

Bias-Variance Tradeoff

- To get good predictions, we need to find a balance of Bias and Variance that minimizes total error

$$Err(x) = \left(E[\hat{f}\,(x)] - f(x)\right)^2 + E\left[\left(\hat{f}\,(x) - E[\hat{f}\,(x)]\right)^2\right] + \sigma_e^2$$

$$Err(x) = \text{Bias}^2 + \text{Variance} + \text{Irreducible Error}$$



Reviewing some ML models

ML: Tree-based models

- Hypothesis space: Tree-based models

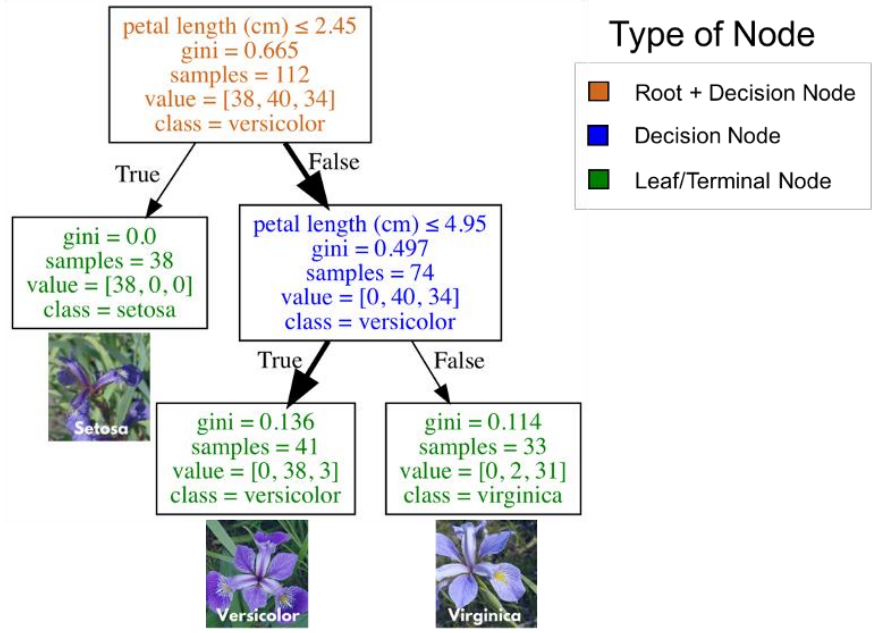
- Representation = Tree or Forest
- Learning algorithm = Greedy algorithms
  - Split the data based on an attribute that optimizes certain criterion
  - Examples:
    - CHAID, CART
    - ID3, C4.5, SLIQ, SPRINT
  - The learning algorithms different on the:
    - The considered attributes
    - Theirs types
    - How they are used (one or several time)
    - The form of the tree/forest
    - The criteria to optimize

ML: Tree-based models

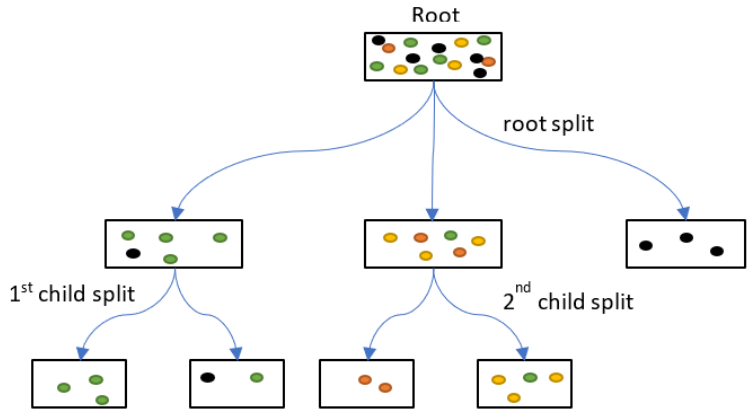
- The most important criteria in Trees is the notion of impurity
  - Used to judge the splits and to mesure the order/desorder

What class (species) is a flower with the following feature?

petal length (cm): 4.5



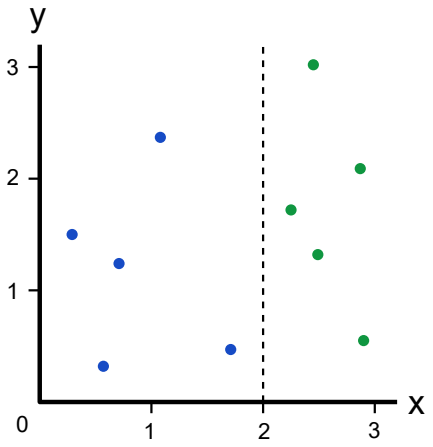
Species counts are: setosa=0, versicolor=38, virginica=3
Prediction is **versicolor** as it is the majority class



Based on attributes and their modalities

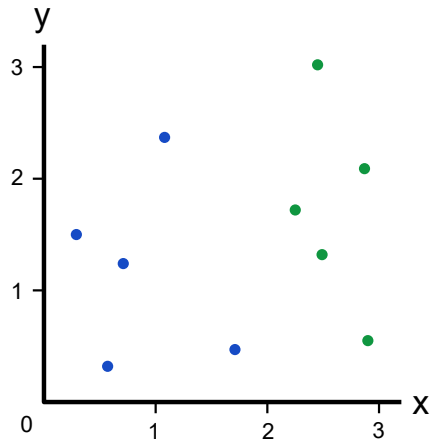
ML: Tree-based models

- Gini impurity
  - If we have *C* total classes and *p(i)* is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as: $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$



$$G_{left} = 1 * (1 - 1) + 0 * (1 - 0) = \boxed{0}$$

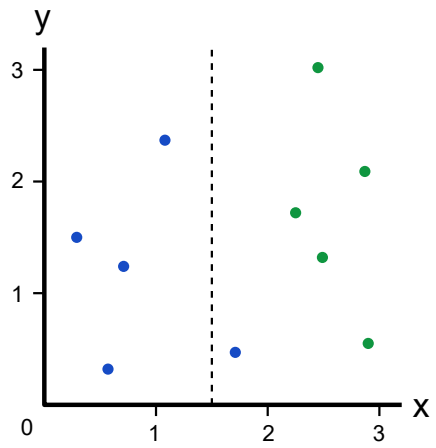
$$G_{right} = 0 * (1 - 0) + 1 * (1 - 1) = \boxed{0}$$



$$G = p(1) * (1 - p(1)) + p(2) * (1 - p(2))$$
$$= 0.5 * (1 - 0.5) + 0.5 * (1 - 0.5)$$
$$= \boxed{0.5}$$

ML: Tree-based models

- Gini impurity
  - If we have *C* total classes and *p(i)* is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as: $G = \sum_{i=1}^{C} p(i) * (1 - p(i))$



$$G_{right} = \frac{1}{6} * (1 - \frac{1}{6}) + \frac{5}{6} * (1 - \frac{5}{6})$$
$$= \frac{5}{18}$$
$$= \boxed{0.278}$$

$$G_{left} = \boxed{0}$$

The information gain: how much Gini we removed

$$0.5 - 0.167 = \boxed{0.333}$$

Since Left Branch has 4 elements and Right Branch has 6, we get (Mean):

$$(0.4 * 0) + (0.6 * 0.278) = 0.167$$

ML: Tree-based models

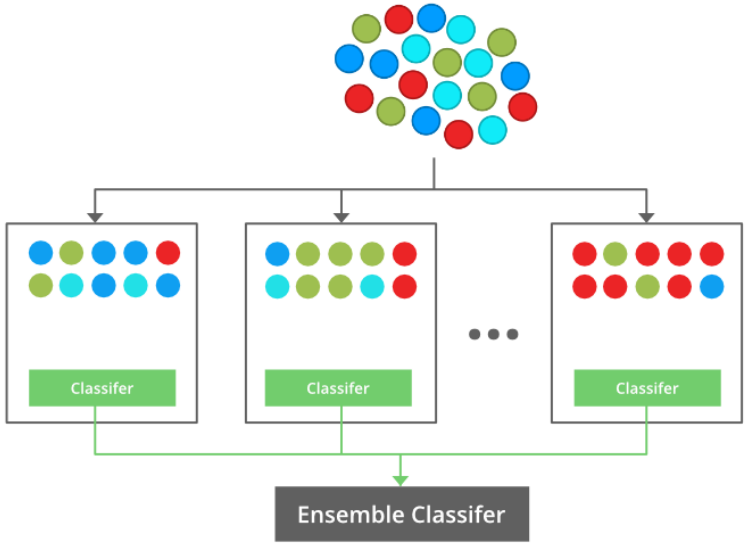
- Entropy impurity
  - If we have *C* total classes and *p(i)* is the probability of picking a datapoint with class i, then the Gini Impurity is calculated as: $\mathrm{E} = -\sum_{i=1}^{C} p(i) * log2(p(i))$
  - Maximum Gini or Entropy gain is used as splitting criterion

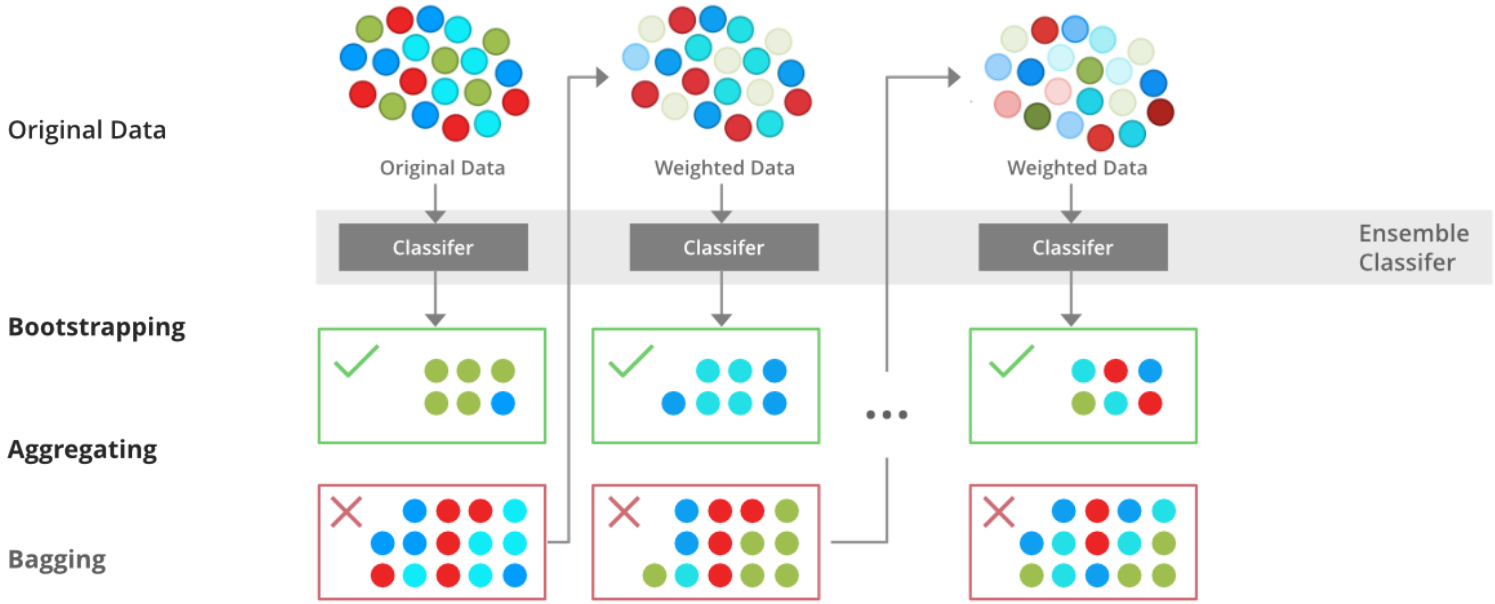
ML: Tree-based models

- Learning algorithms
  - Node impurity
    - Gini, Entropy, Gain, Khi2, …
  - Bagging = Bootstrap + aggregating
    - **Subsamples (random forest) vs all samples (DT, ET, …)**
  - Attribute selection: (random) greedy heuristics
    - **optimum split (random forest)**
    - **randomly split (extra trees)**
  - Nodes with homogenous class distribution are preferred
  - Boosting (more general)
  - Bias-variance Tradeoff
    - Multiple objectives: Misclassification error; Tree size (Prunning), depth, ..

ML: Tree-based models

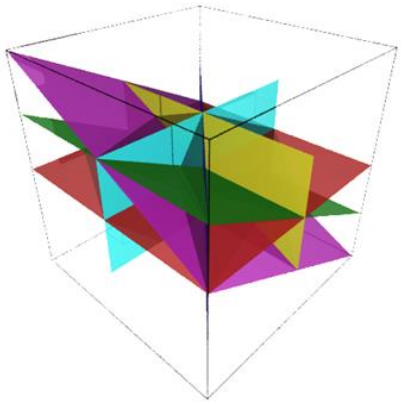
- Bagging vs Boosting





ML: Hyperplane models

- Hypothesis space: SVM models



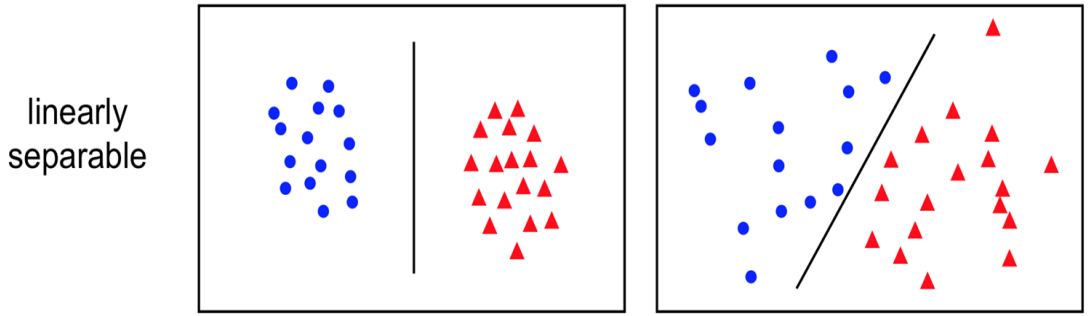
- Representation = Hyperplane
- Learning algorithm = quadratic optimization algorithm with linear constraints
  - A method used for regression and classification (mostly used in classification problems)
  - SVM is fundamentally a binary classifier, but can be extended for multiclass problems
  - Classification performed by learning a linear separator of the data

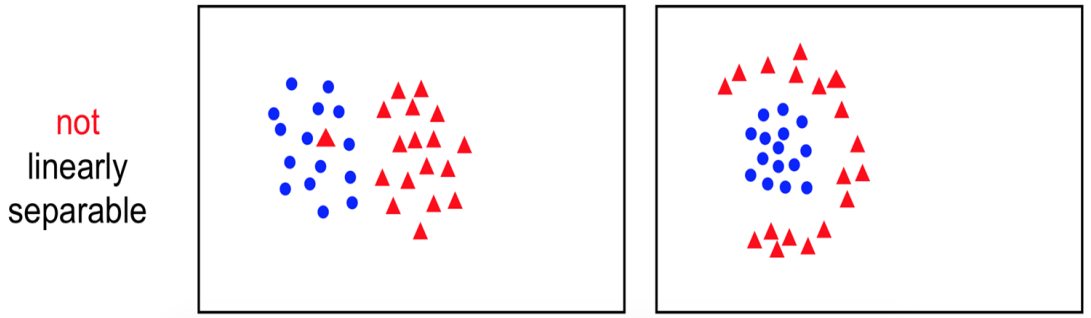
ML: SVM models

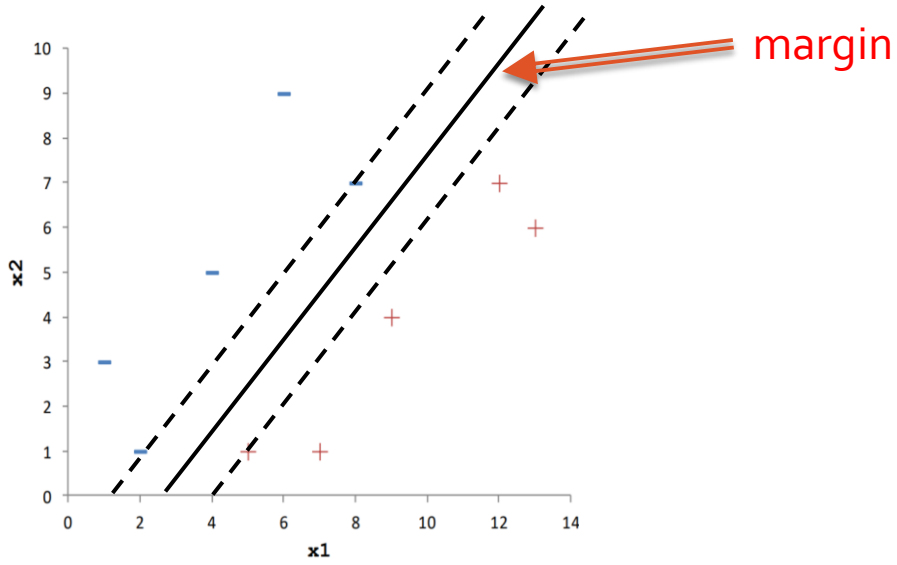
- Linear SVM

  There exists an infinite number of linear separators which one is optimal?

  Larger margin is preferred for generalization





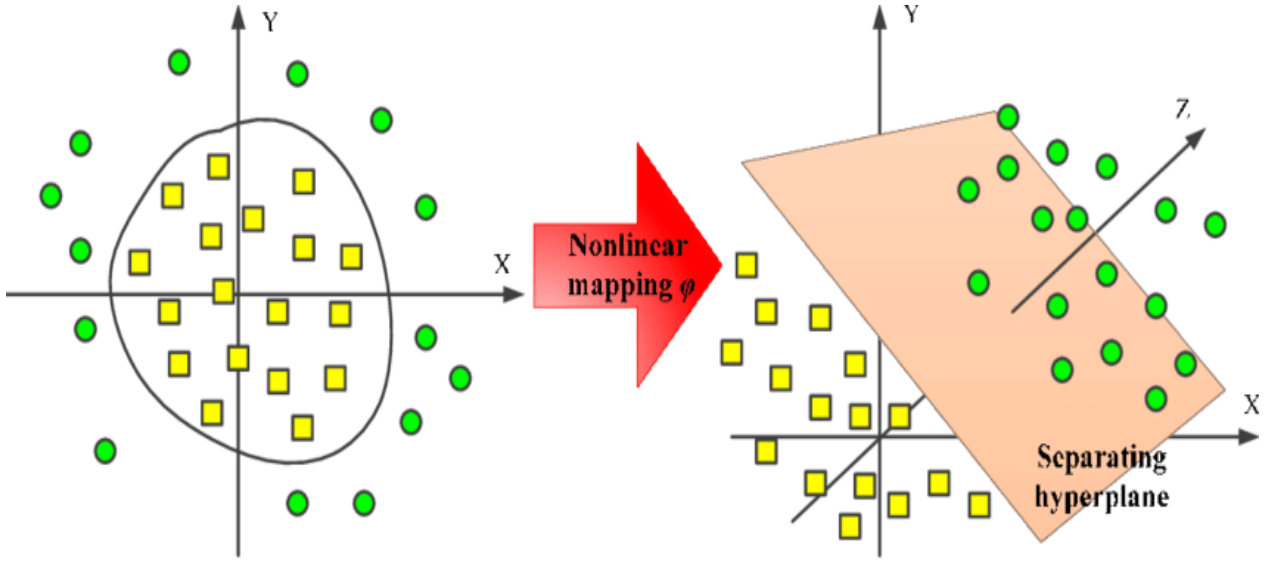


ML: SVM models

- Non-Linear SVM

  Kernel methods: sigmoid, polynomial, …

  More difficult problem, we use metaheuristics



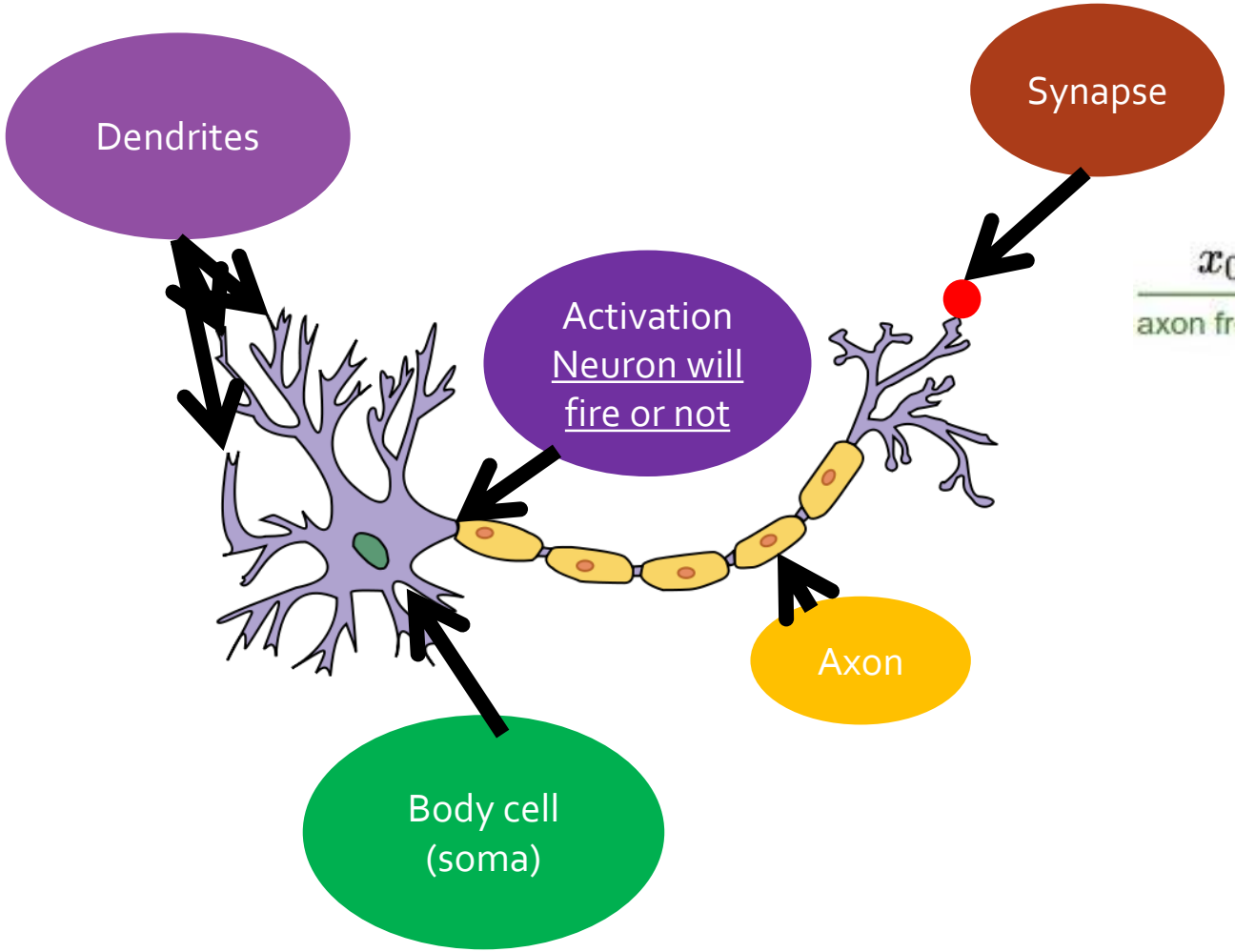
- Transform the data to a higher dimensional space where it can be separated by a linear hyperplane
- Learn a linear classifier for the new space : $f(x) = w^T \varphi(x) + b$

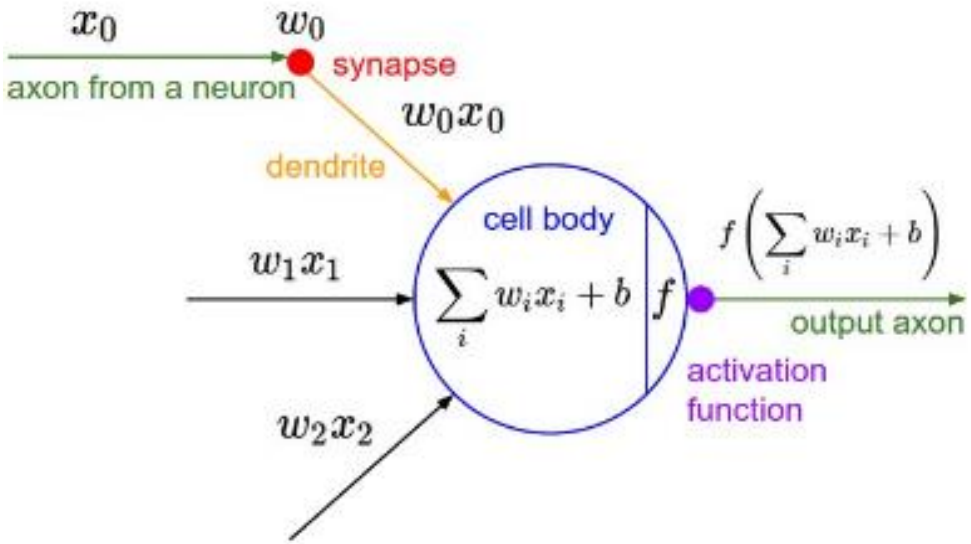
ML: Neural Network models

- Hypothesis space: NN models

- NN = Direct graph
- Objective function: classification error
- Learning algorithm = optimization algorithm (backpropagation using gradient, …)
  - NP-Hard and complex problem
  - Metaheuristics have been widely used

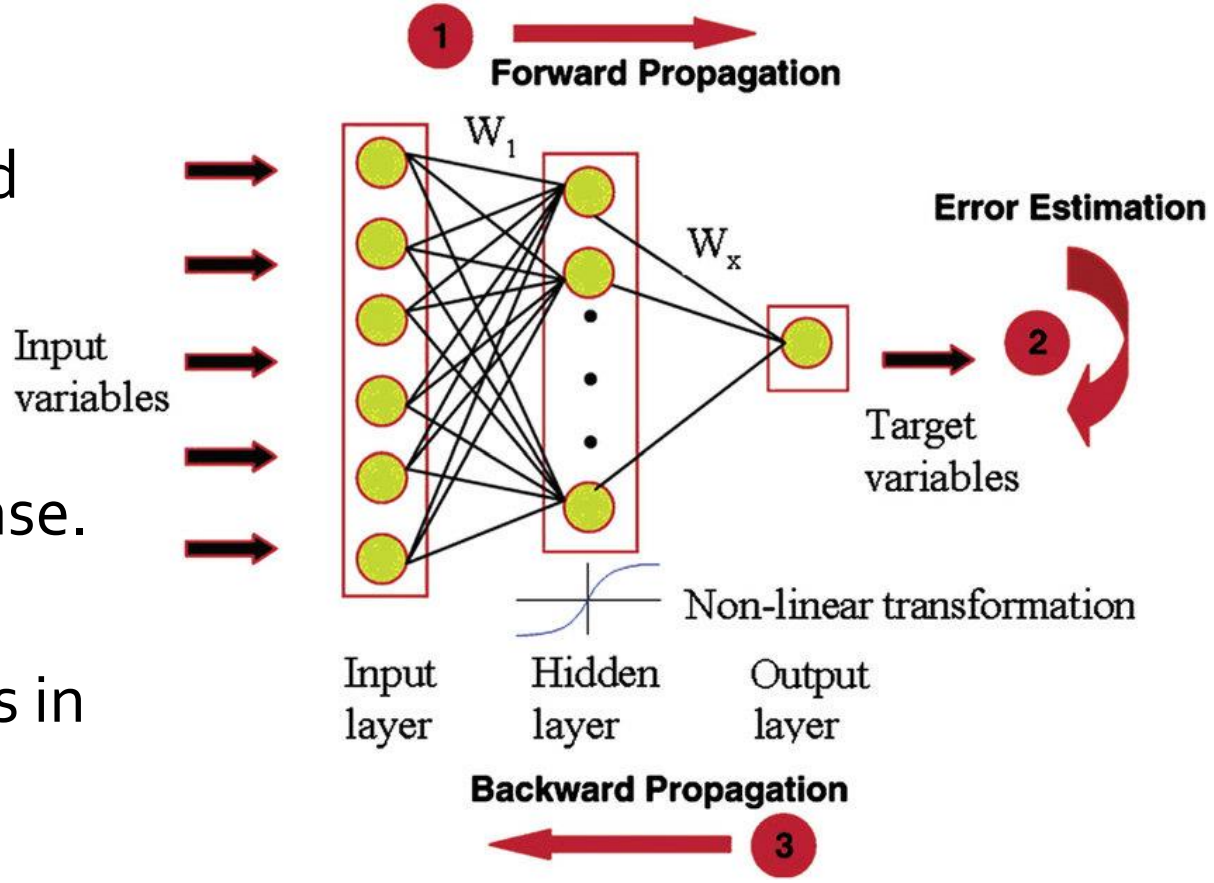
ML: Neural Network models



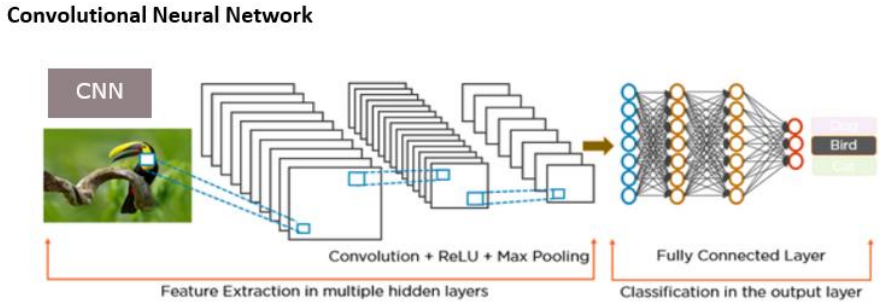


ML: Neural Network models

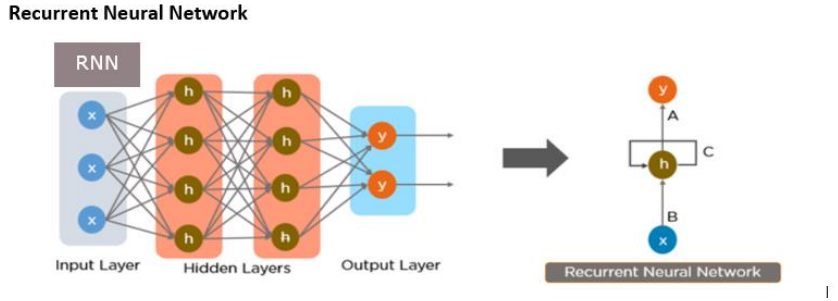
- Forward propagation: the weights are fixed and the input signal is propagated through the network layers, until it reaches the output.
- Error is estimated by comparing the network output and the desired response.
- Backward propagation: the error is propagated through the network layers in the backward direction



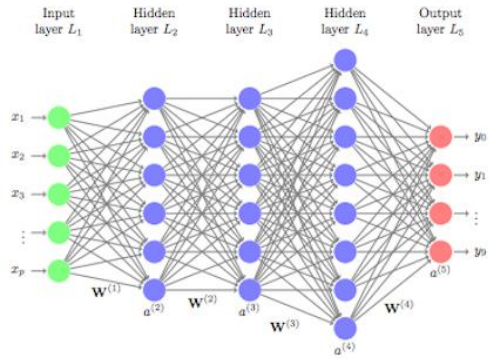
ML: Neural Network models



- Forward
  - Several architectures: FFNN, CNN, RNN



- Backpropagation
  - Several optimizer to deal with vanishing and exploding gradient:
    - Gradient Descent (GD), Stochastic GD, mini-batch GD, Momentum, Adam, AdaGrad, RMSPROP,



ML: Neural Network models

- Overfitting

You should always consider using regularization, unless you have a very large dataset,

- Reduce overfitting by training the network on more examples.

- Reduce overfitting by changing the complexity of the network.
  - Change network complexity by changing the network structure (number of weights).
  - Change network complexity by changing the network parameters (values of weights)

ML: Neural Network models

- Regularization Methods for Neural Networks
  - Weight Regularization (weight decay): Penalize the model during training based on the magnitude of the weights.
  - Activity Regularization: Penalize the model during training base on the magnitude of the activations.
  - Weight Constraint: Constrain the magnitude of weights to be within a range or below a limit.
  - Dropout: Probabilistically remove inputs during training.
  - Noise: Add statistical noise to inputs during training.
  - Early Stopping: Monitor model performance on a validation set and stop training when performance degrades.

ML: Neural Network models

- Some more specific recommendations for Multilayer Perceptrons and Convolutional Neural Networks.
  - Classical: use early stopping and weight decay (L2 weight regularization).
  - Alternate: use early stopping and added noise with a weight constraint.
  - Modern: use early stopping and dropout, in addition to a weight constraint.

- Some more specific recommendations for recurrent neural nets
  - Classical: use early stopping with added weight noise and a weight constraint such as maximum norm.
  - Modern: use early stopping with a backpropagation-through-time-aware version of dropout and a weight constraint.