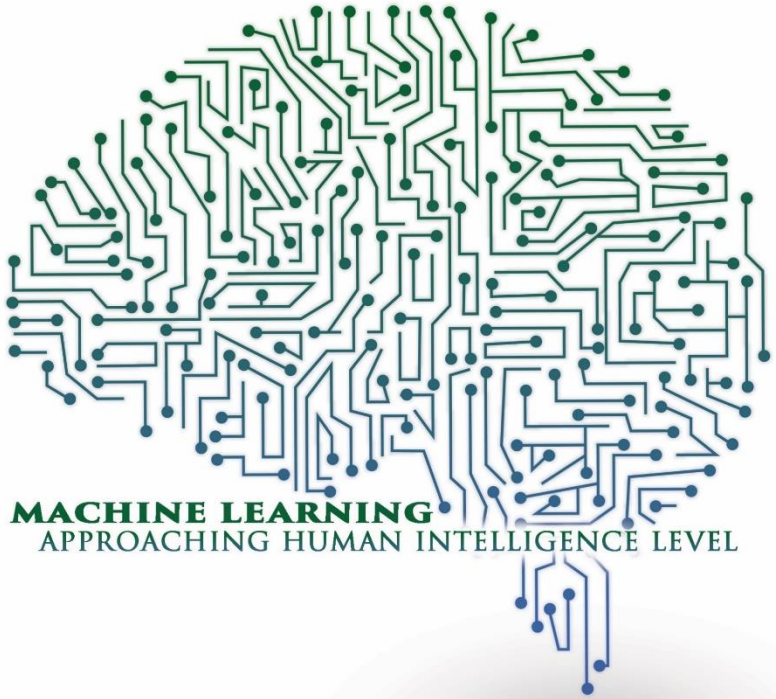
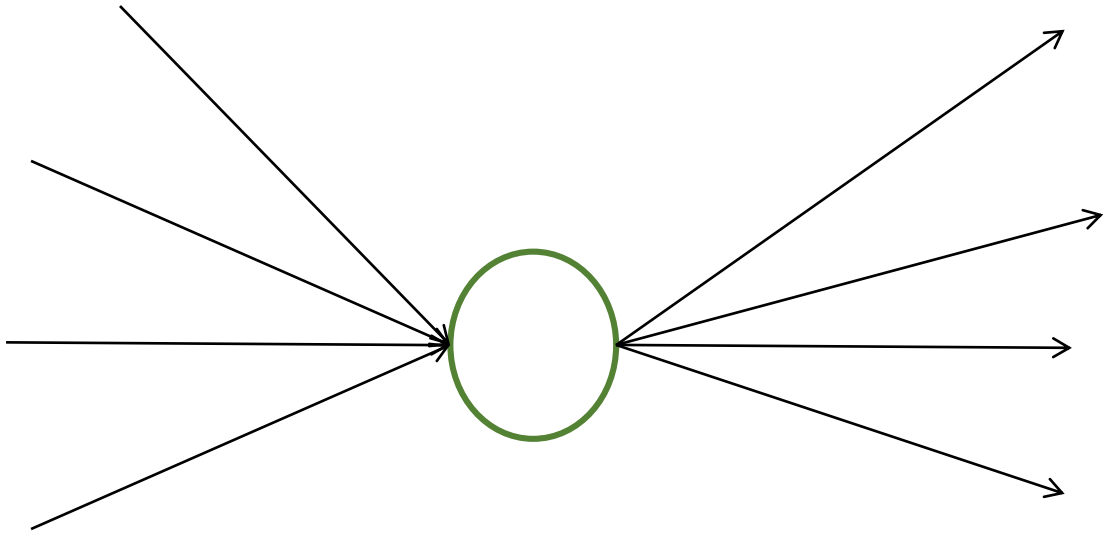


Artificial neural networks.

Backpropagation algorithm

For simplicity let us take an arbitrary neuron in a network.

$\delta_j^{(l)}$ = "error" of node j in layer l.



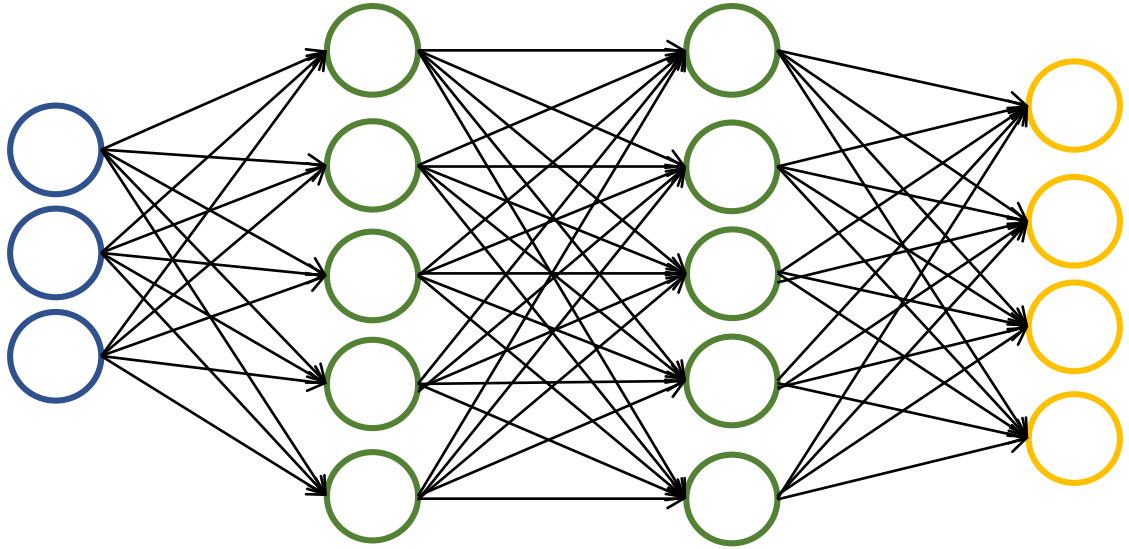
Backpropagation algorithm

For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$



Put it all together

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

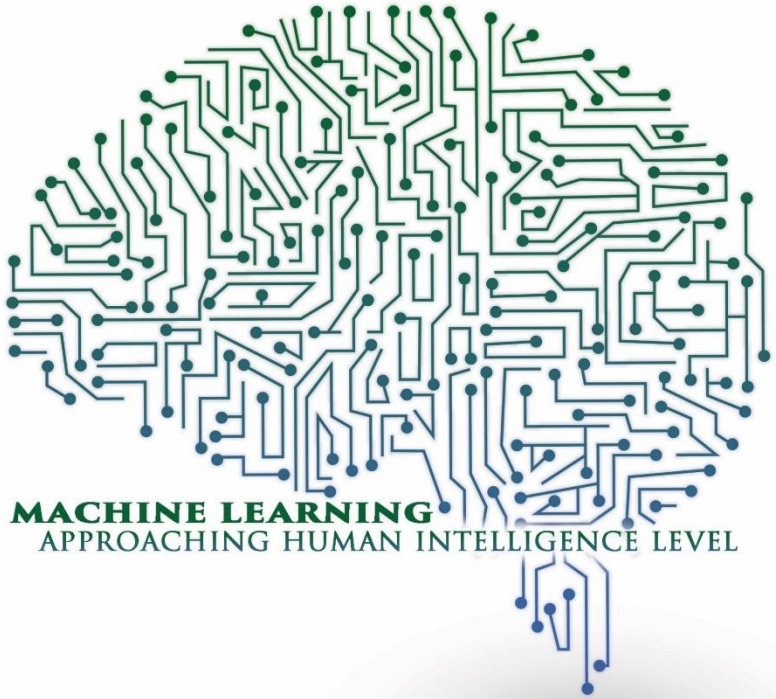
Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

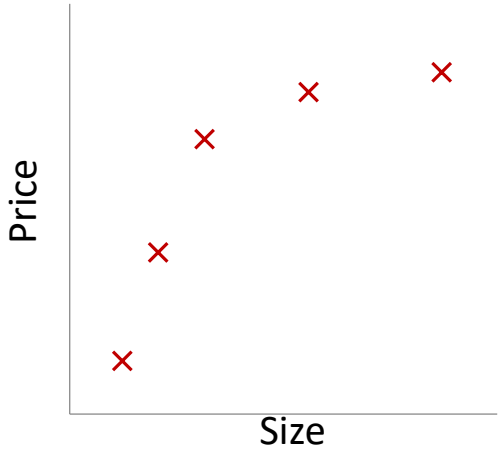
$$D_{ij}^{(l)} := \frac{1}{m} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

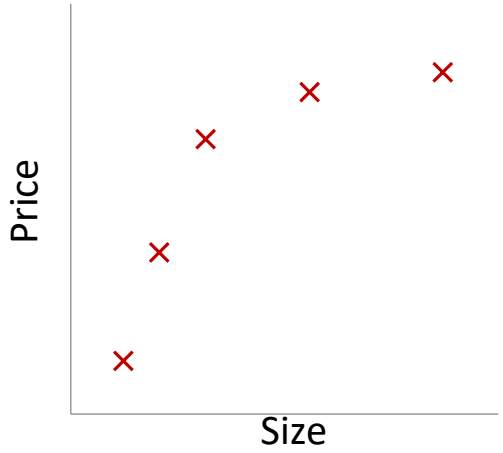


overfitting & underfitting

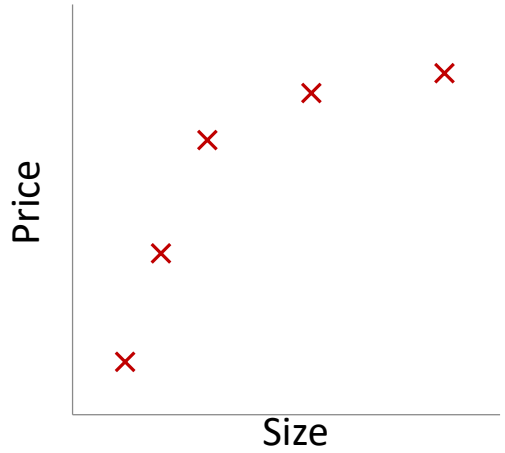
The problem of overfitting & underfitting



$\theta_0 + \theta_1 x$



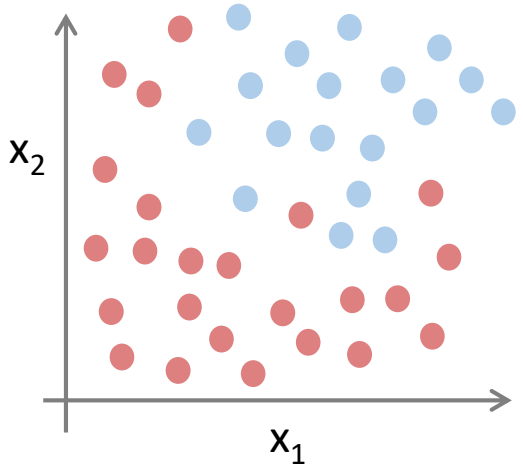
$\theta_0 + \theta_1 x + \theta_2 x^2$

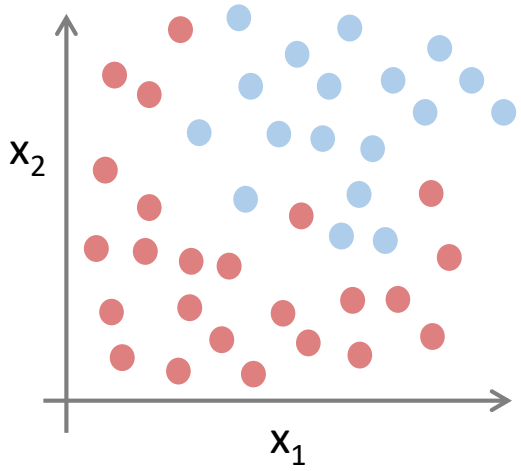


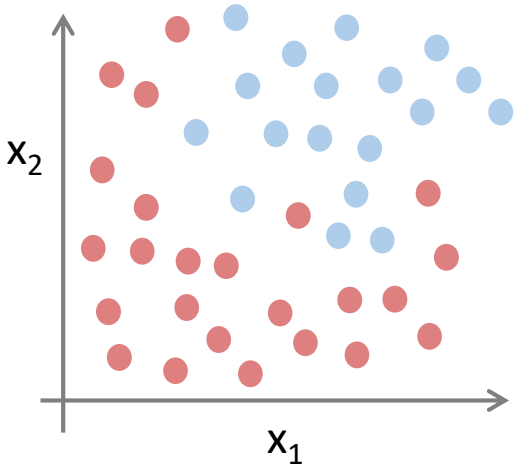
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Logistic regression







$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(= sigmoid function)

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_2^2 + \theta_5 x_1 x_2)$

$g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \theta_6 x_1^3 x_2 + \dots)$

Regularization

Small values for parameters, $\theta_1, \theta_2, \theta_3, \ldots$

- "Simpler" hypothesis
- Less prone to overfitting

By forcing the optimization to minimize over the parameters

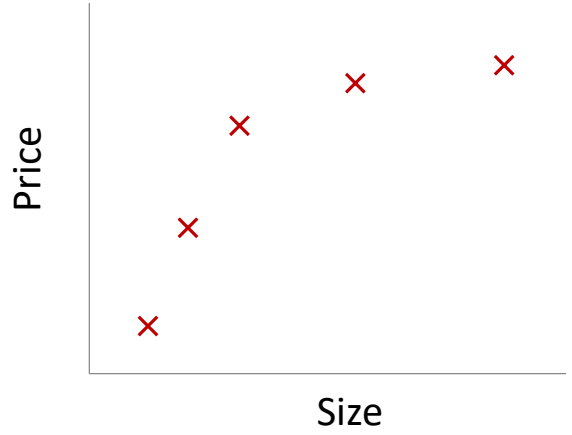
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2$$

Regularization (Ridge regularizer)

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

What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{100}$)?

- Algorithm works fine; setting 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.



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Regularized linear regression

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

Gradient descent

Repeat

$$\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_j := \theta_j - \alpha$$

}

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

Regularized logistic regression.

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

Gradient descent

Repeat

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

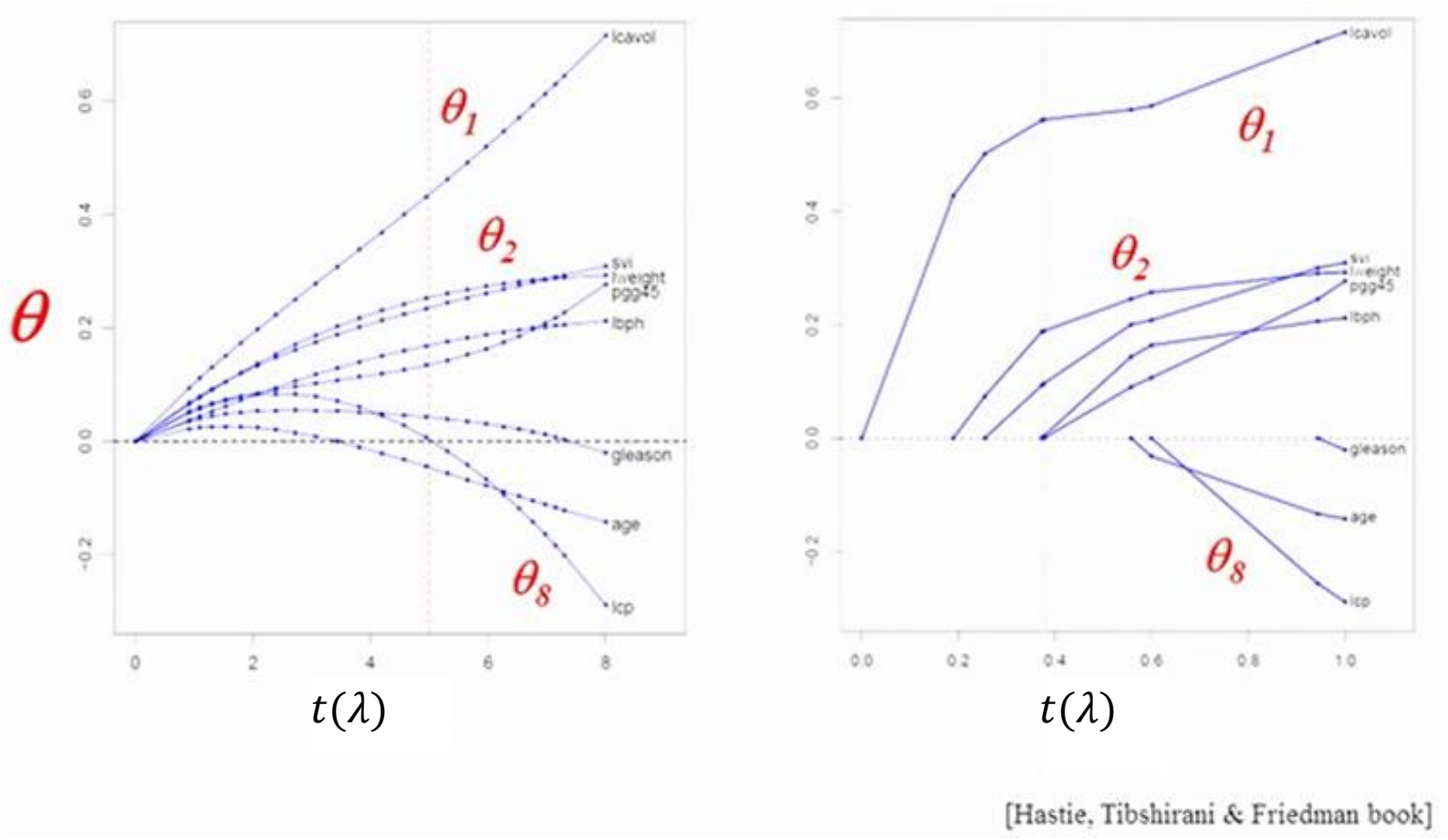
$$\theta_j := \theta_j - \alpha \quad \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$(j = 0, 1, 2, 3, \ldots, n)$$

}

Regularization and feature selection (lasso)

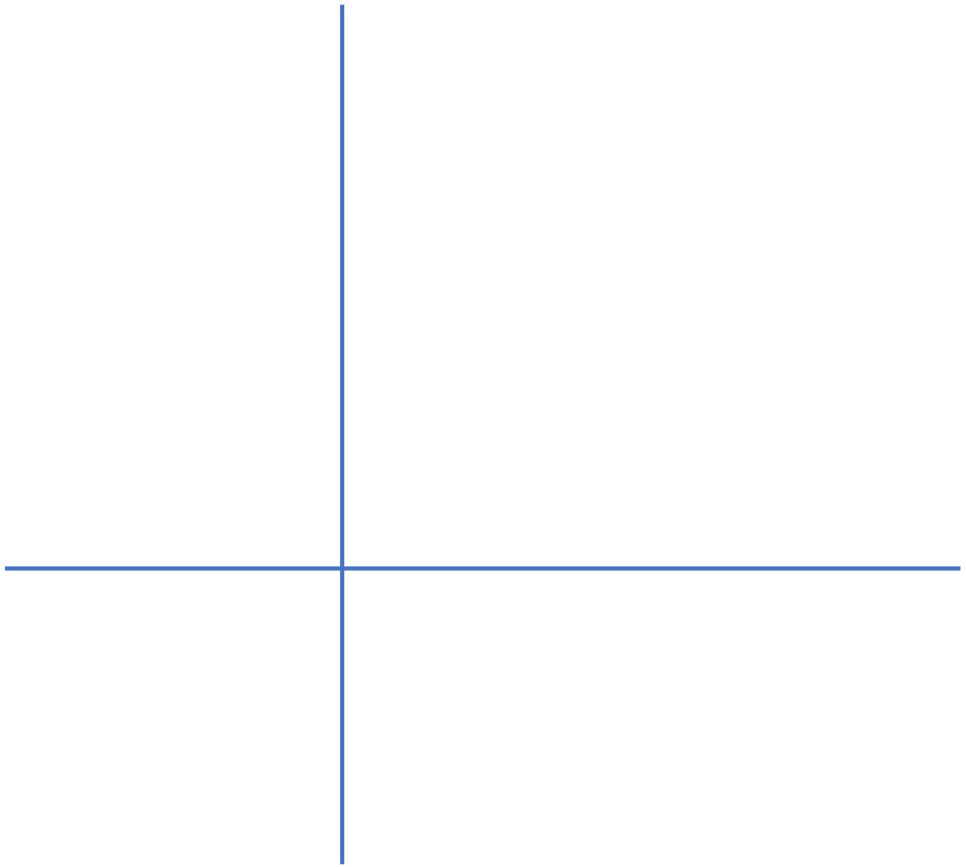
As λ increases, $t(\lambda)$ decreases and each θ_i goes to zero, but too slowly for ridge.
Lasso will ensure that irrelevant features x_i have weight θ_i = 0.



[Hastie, Tibshirani & Friedman book]

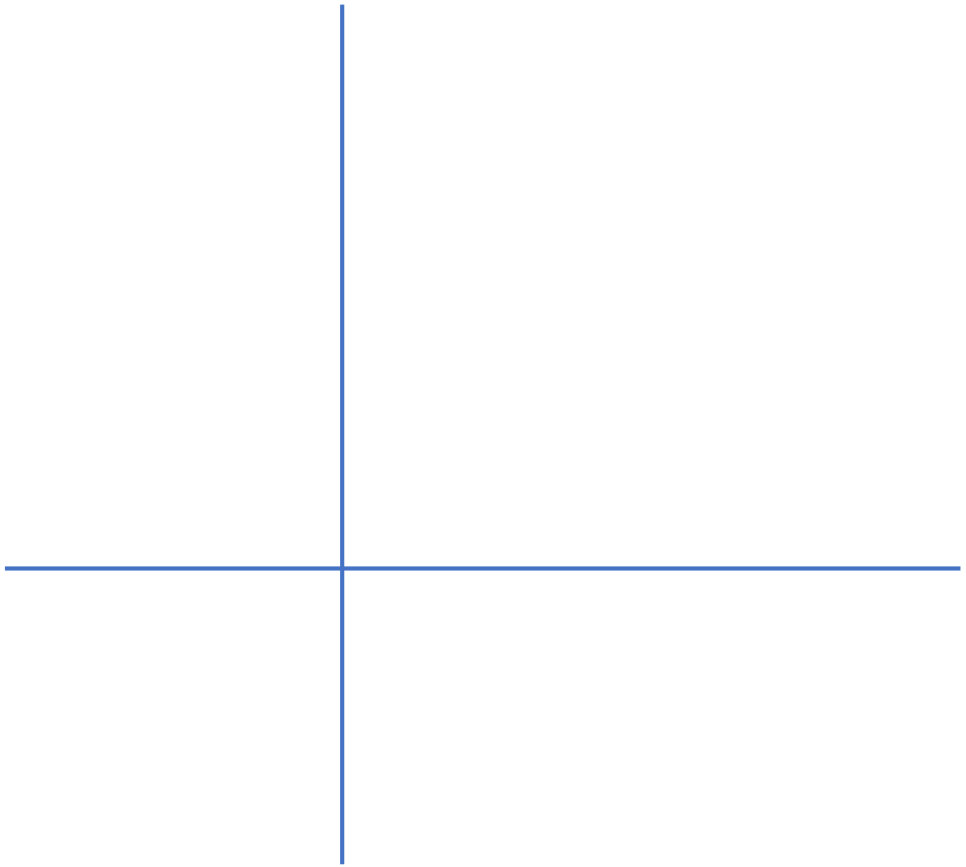
Regularized linear regression

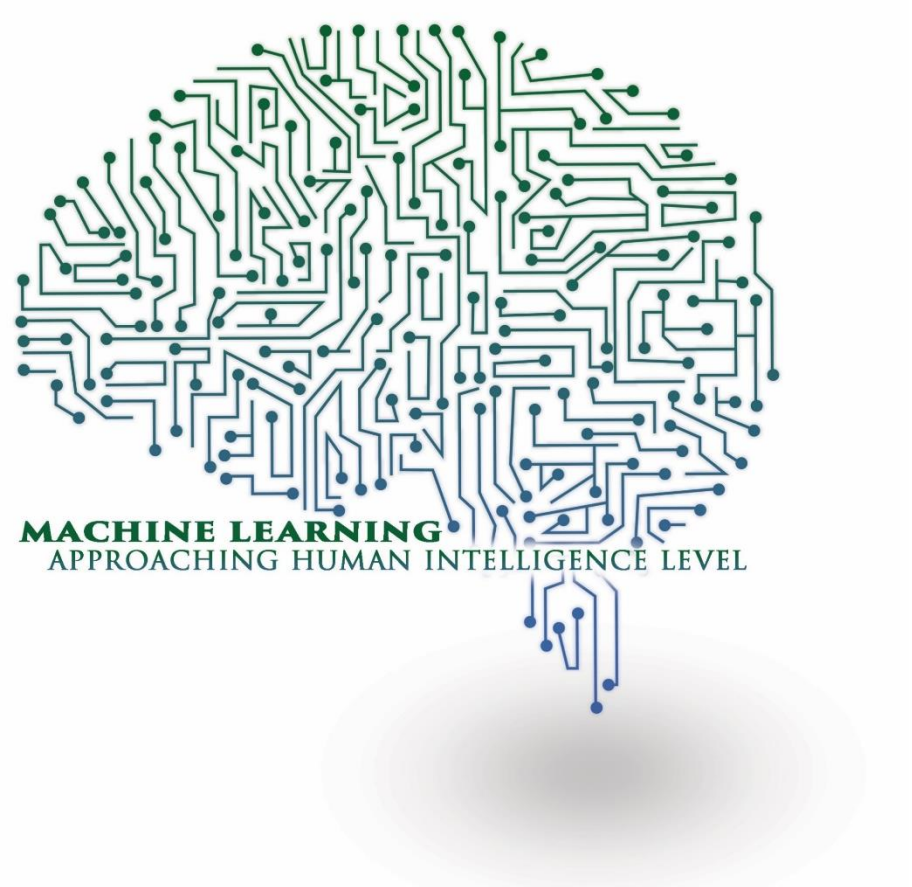
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$



Regularized linear regression

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x_i) - y_i)^2 + \lambda\sum_{j=1}^{n}|\theta_j|$$





Cross validation

Cross validation

Model selection

- Almost invariably, all pattern recognition techniques have one or more free parameters (hyperparameters)
  - The number of neighbours in a kNN classification rule
  - The network size, learning parameters and weights in MLPs

How do we select the “optimal” parameter(s) or model for a given classification problem?

Performance estimation

- One approach is to use the entire training data to select our classifier and estimate the error rate.
  - The final model will normally overfit the training data (large parameter model).
  - The error rate estimate will be overly optimistic (lower than the true error rate) .

Solution

Split dataset into two groups:

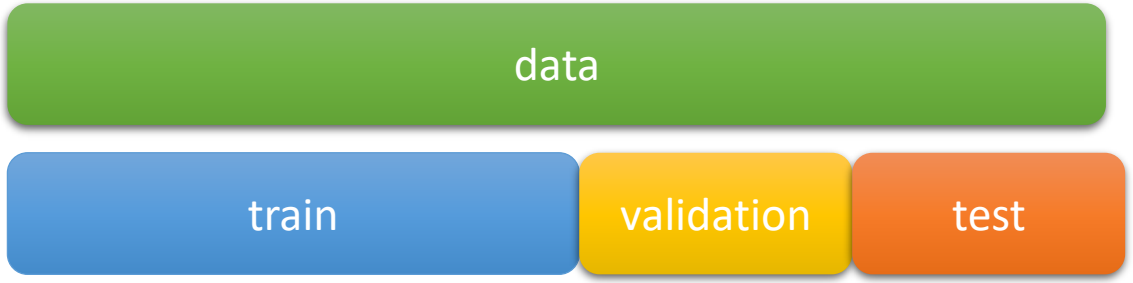
- Training set: used to train the classifier
- Test set: used to evaluate the trained classifier

But what to do for the hyperparameters ?

If we modify (tune) the hyperparameter on the result from the test set, we actually have roing the error form the test set.

Three way cross validation
Split dataset into two groups:

- Training set: used to train the classifier
- Validation set: used to modify the hyperparameter
- Test set: used to evaluate the trained classifier

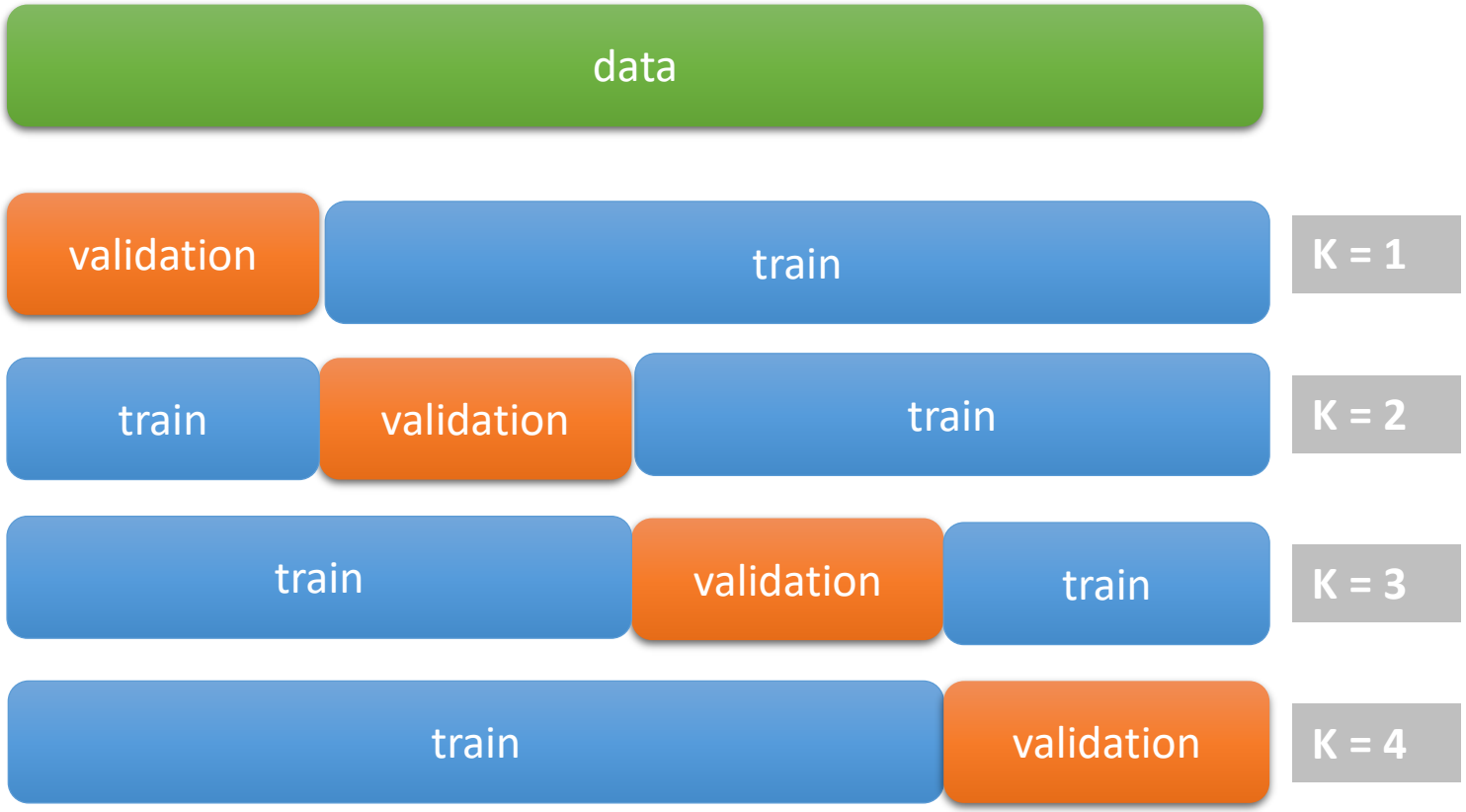


After assessing the final model with the test set, YOU MUST NOT further *tune* the model

K-Fold Cross-validation

Split dataset into k groups:

- For each of K experiments, use K-1 folds for training and the remaining one for testing.



The advantage of K-Fold Cross validation is that all the examples in the dataset are eventually used for both training and testing.

How many folds are needed?

With a large number of folds

+ The bias of the true error rate estimator will be small (the estimator will be very accurate).
- The variance of the true error rate estimator will be large.
- The computational time will be very large as well (many experiments).

With a small number of folds

+ The number of experiments and, therefore, computation time are reduced.
+ The variance of the estimator will be small.
- The bias of the estimator will be large (conservative or higher than the true error rate).

In practice, the choice of the number of folds depends on the size of the dataset

- For large datasets, even 3-Fold Cross Validation will be quite accurate
- For very sparse datasets, we may have to use leave-one-out in order to train on as many examples as possible.

A common choice for K-Fold Cross Validation is K=10

How to choose the hyperparameter

Taking the learning rate λ for simplicity

λ	Train error	Validation error	Max	Min-Max	average
0.1	100	2			
1	10	11			
10	1	9			
50	20	0			
100	100	1000			