GHENT UNIVERSITY

FACULTY OF ENGINEERING AND ARCHITECTURE

E016350 - Artificial Intelligence

Lecture 2

Machine learning

Supervised Learning: Linear Regression and Classification

Aleksandra Pizurica

Ghent University
Spring 2025

Outline

[R&N], Chapter 19

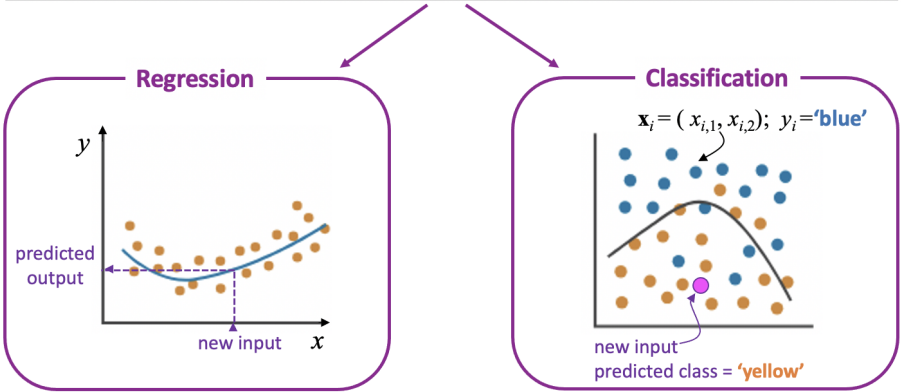
These slides are based on: S. Russel and P. Norvig: *Artificial Intelligence: A Modern Approach*, (Fourth Ed.), http://aima.cs.berkeley.edu/ and the Stanford course of M. Charikar and Koyejo (CS221): Artificial Intelligence: Principles and Techniques.

Supervised learning

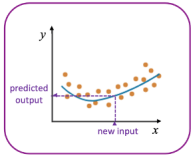
Given a training set of N example input-output pairs

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots (\mathbf{x}^{(N)}, y^{(N)})$$

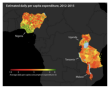
where each $y^{(j)}$ was generated by an unknown function $y = f(\mathbf{x})$, discover a function h that approximates the true function f.



Regression: Examples



$x \rightarrow \boxed{f} \rightarrow y \in \mathbb{R}$ **response**



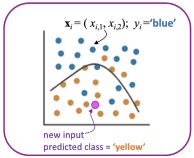
Poverty mapping: satellite image $\rightarrow$ asset wealth index

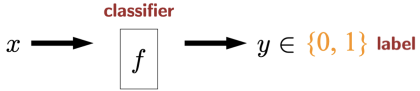
Housing: information about house $\rightarrow$ price

Arrival times: destination, weather, time $\rightarrow$ time of arrival

Slightly adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Classification: Examples





Fraud detection: credit card transaction $\rightarrow$ fraud or no fraud

Toxic comments: online comment $\rightarrow$ toxic or not toxic

Higgs boson: measurements of event $\rightarrow$ decay event or background

Extension: multiclass classification: $y \in \{1, \ldots, K\}$

Slightly adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Outline

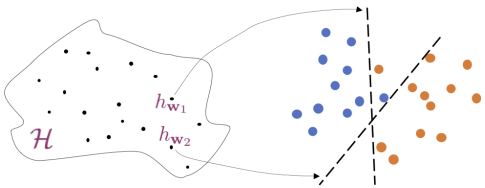
Supervised learning task: How do we approach it?

Given a training set of N example input-output pairs

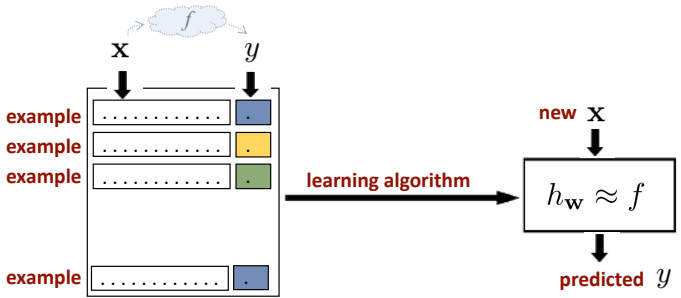
$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots (\mathbf{x}^{(N)}, y^{(N)})$$

where each y_j was generated by an unknown function $y = f(\mathbf{x})$, discover a function h that approximates the true function f.

- Choose a type of models, i.e., the hypothesis class $\mathcal{H}$.
- Find $h \in \mathcal{H}$ that fits best with the examples
  - For parametric models (different $h \in \mathcal{H}$ have different parameters $\mathbf{w}$)
    - ⋆ Find $\mathbf{w}$ that best fits the data. This gives us the model $h_{\mathbf{w}}$, our best hypothesis

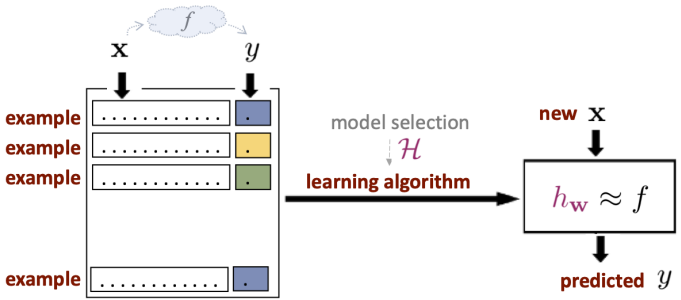


Supervised learning a.k.a. Learning from examples



Note that this is a reflex-based model (feedforward input $\rightarrow$ output operation)

Core concepts of learning from examples in a nutshell



Design choices:

- Model selection – hypothesis class $\mathcal{H}$ (Which predictors are possible?)
- Model optimization – the best hypothesis within $\mathcal{H}$ (find the best predictor h)
  - How to compute the best predictor? **learning algorithm**
    - ⋆ How good is a predictor? **loss function**

Hyperparameters

Definition (Hyperparameters)

Design decisions (hypothesis class, training objective, optimization algorithm) that need to be made before running the learning algorithm.

How do we choose hyperparameters?

- Choose hyperparameters to minimize the error on the training set?
  - Bad idea! – Why?
- Choose hyperparameters to minimize the error on the test set?
  - Bad idea! – Why?
- Hence, we need a separate validation set!

Definition (Validation set)

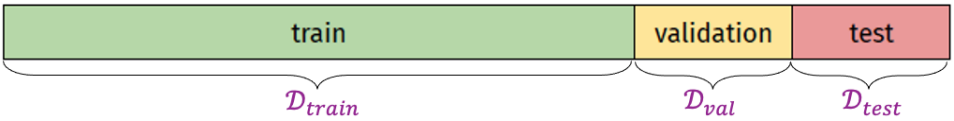
A validation set is taken out of the training set and used to optimize hyperparameters.

Training, validation and test sets

Optimization of the hyperparameters means selection of a particular **candidate model** (E.g., depth of a decision tree, number of layers in a neural network, batch size etc.)

Let $\mathcal{D}$ denote the dataset of all available labelled examples
Basic ML rule: split $\mathcal{D}$ into three disjoint sets:

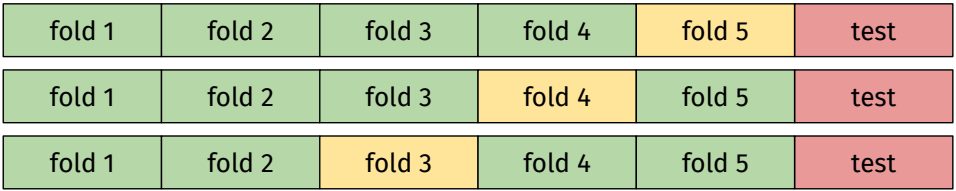
- Training set ($\mathcal{D}_{train}$) – to train candidate models
- Validation set ($\mathcal{D}_{val}$) – to evaluate the candidate models and choose the best one
- Test set ($\mathcal{D}_{test}$) – to do a final unbiased evaluation of the best model



Cross-validation

Splitting $\mathcal{D} \setminus \mathcal{D}_{test}$ into $\mathcal{D}_{train}$ and $\mathcal{D}_{val}$:

- **Hold-out cross validation**
  - split randomly
  - fails to use all available data
- **k-fold cross validation**
  - k subsets of data; k rounds of learning
    – each time $1/k$ of the data is held as a validation set



Outline

Formal description of supervised learning

- Let $\mathcal{H}$ denote the hypothesis space
- The learning problem is realizable if the hypothesis space contains the true function
- A possible formal description is a probabilistic interpretation: "Choose the hypothesis $h^\star$ that is most probable given the data":

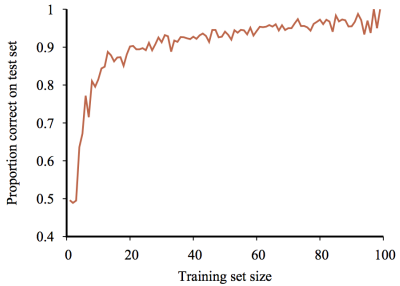
$$h^\star = \arg\max_{h \in \mathcal{H}} P(h|data) = \arg\max_{h \in \mathcal{H}} P(data|h)P(h)$$

Performance measurement

How do we know that $h \approx f$? (Hume's **Problem of Induction**)

1. Use theorems of computational/statistical learning theory
   - Any seriously wrong hypothesis will be "find out" with high probability after a small number of examples because it will make an incorrect prediction
2. Try h on a new test set of examples

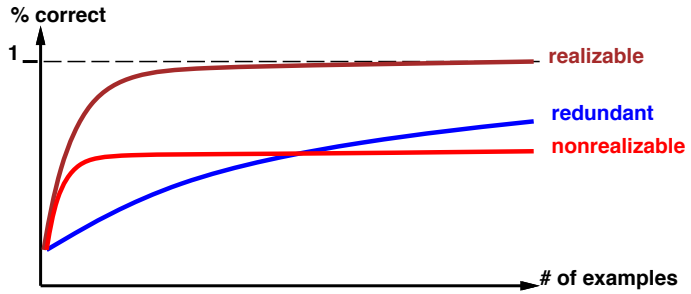
Learning curve = % correct on test set as a function of training set size



Performance measurement cont'd

Learning curve can indicate

- A realizable or non-realizable learning task
  - realizable meaning it can express target function
  - non-realizability can be due to missing attributes or too restricted hypothesis class
- redundant expression, e.g., due to loads of repetitive/irrelevant data



Error rate and loss

- Error rate: proportion of mistakes that a hypothesis makes
- **Loss function**: the amount of utility lost by replacing the correct answer $f(x) = y$ by a hypothesis $h(x) = \hat{y}$

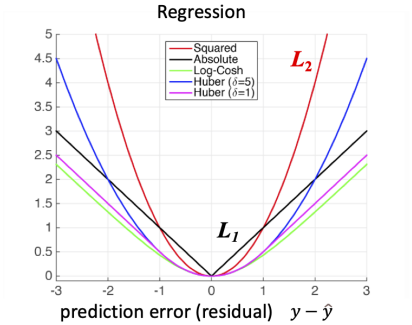
Definition (Loss function)

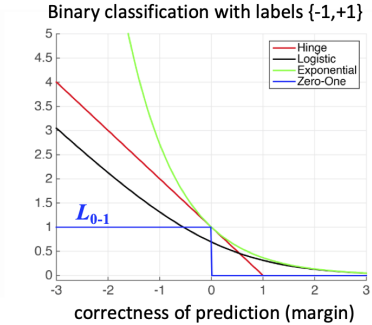
$$\begin{aligned} L(x, y, \hat{y}) &= Utility(\text{result of using } y \text{ given an input } x) \\ &- Utility(\text{result of using } \hat{y} \text{ given an input } x) \end{aligned}$$

- Sometimes it is more convenient to write $L(y, h(\mathbf{x}))$ or simply $L(y, \hat{y})$
- In the case of parameteric learning $\hat{y} = h_{\mathbf{w}}(x)$
  $\rightarrow$ We will use interchangeably $L(y, h_{\mathbf{w}}(x))$ and $L(x, y, \mathbf{w})$
  (since the loss depends on x, y and $\mathbf{w}$ in both)

Some common loss functions

- Absolute-value loss: $L_1(y, \hat{y}) = |y - \hat{y}|$
- Squared-error loss: $L_2(y, \hat{y}) = C(y - \hat{y})^2$ where C is a constant[1]
- Zero-one loss: $L_{0-1}(y, \hat{y}) = 0$ if $\hat{y} = y$, otherwise 1





[1] typically set to 1 but sometimes to $1/2$ for normalization of the expressions with derivatives.

Generalization loss and the best hypothesis

The learning agent maximizes its expected utility by choosing the best hypothesis according to some well-defined criterion

Definition (Generalization loss)

The expected generalization loss for a hypothesis h, with respect to loss function L is

$$GenLoss_L(h) = \sum_{(x,y)\in\mathcal{E}} L(y, h(x))P(x, y)$$

$\mathcal{E}$ – the set of all possible input-output pairs; $P(x, y)$ – joint probability of x and y.

The best hypothesis is the one with the minimum expected generalization loss:

$$h^* = \arg\min_{h\in\mathcal{H}} GenLoss_L(h)$$

Empirical loss and Training loss

In reality, $P(x, y)$ is not known, so the learning agent can only estimate the generalization loss with an empirical loss on a set of examples, E:

$$EmpLoss_{L,E}(h) = \frac{1}{|E|} \sum_{(x,y)\in E} L(y, h(x))$$

The estimated best hypothesis is the one with the minimum empirical loss

$$\hat{h}^* = \arg\min_{h\in\mathcal{H}} EmpLoss_{L,E}(h)$$

Definition (Training loss)

Training loss is the empirical loss over the set of training examples:

$$TrainLoss_{L,\mathcal{D}_{train}}(h) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(y, h(x))$$

Training loss: Notation and examples

For compactness, we will omit from the expression for the training loss the indices that denote the loss function and the training set, i.e., we write it as

$$TrainLoss(h) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(y, h(x))$$

When learning parametrized models (the task is to find $\mathbf{w}$), we will write:

$$TrainLoss(\mathbf{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} L(y, h_{\mathbf{w}}(x))$$

Our task is then $\min_{\mathbf{w}} TrainLoss(\mathbf{w})$. For example, with the squared-error loss:

$$TrainLoss(\mathbf{w}) = \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y)\in\mathcal{D}_{train}} (h_{\mathbf{w}}(x) - y)^2$$

Regularization

Search for a hypothesis that directly minimizes the weighted sum of empirical loss and the complexity of the hypothesis, which is also called total cost

$$Cost(h) = EmpLoss(h) + \lambda\, Complexity(h)$$

$$\hat{h}^* = \arg\min_{h \in \mathcal{H}} Cost(h)$$

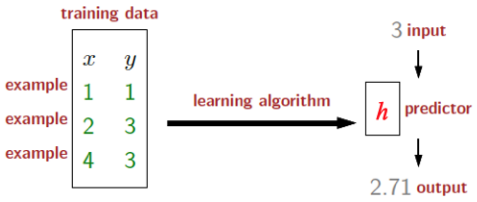
λ is a parameter, often determined by cross-validation. This process explicitly penalizes complex hypotheses and is therefore called regularization (preference is given to more regular functions). In practice, our training objective will become:

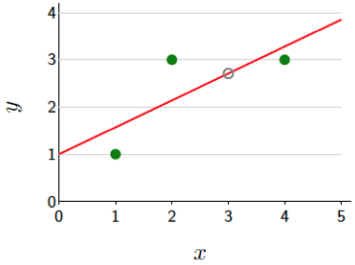
$$\min_{\mathbf{w}} \frac{1}{|\mathcal{D}_{train}|} \sum_{(x,y) \in \mathcal{D}_{train}} L(y, h_{\mathbf{w}}(x)) + Reg(\mathbf{w})$$

where $Reg(\mathbf{w})$ is some regularization function imposed on the weights, e.g., ℓ_1-regularization: $Reg(\mathbf{w}) = |\mathbf{w}|$ or ℓ_2-regularization: $Reg(\mathbf{w}) = \mathbf{w}^2$.

Outline

Linear regression framework





Design choices:

- Model selection – hypothesis class $\mathcal{H}$ (Which predictors are possible?)
- Model optimization – the best hypothesis within $\mathcal{H}$ (find the best predictor)
  - How to compute the best predictor? **learning algorithm**
    - ★ How good is a predictor? **loss function**

Slide adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Univariate linear regression

Goal: learn the real-valued coefficients w_0 and w_1 of a univariate linear function

$$y = w_1 x + w_0$$

w_0 and w_1 can be seen as **weights**: the value of y is changed by changing the relative weight of one term or another

Denote the vector of weights by $\mathbf{w} = [w_0 \; w_1]$ and define

$$h_{\mathbf{w}}(x) = w_1 x + w_0$$

Linear regression is the task of finding $h_{\mathbf{w}}(x)$ that best fits the data
(i.e. finding $\mathbf{w}$ such that $h_{\mathbf{w}}(x)$ best fits the data)

Univariate linear regression cont'd

Find the values of weights $\mathbf{w} = [w_0 \; w_1]$ that minimize the empirical loss

Traditionally, the squared-error loss L_2 is used*

$$TrainLoss(\mathbf{w}) = \sum_{(x,y)\in\mathcal{D}_{train}} (y - (w_1 x + w_0))^2$$

The training loss is minimized by setting

$$\frac{\partial}{\partial w_0} TrainLoss(\mathbf{w}) = 0; \quad \frac{\partial}{\partial w_1} TrainLoss(\mathbf{w}) = 0$$

Carl Friedrich Gauss
1777-1855
The least squares linear regression
Prediction of the Ceres orbit (1801)

*Gauss showed that if the noise in y's is normally distributed, the most likely values of the weights are obtained by minimizing the sum of the squared errors.

Univariate linear regression cont'd

For compactness of the expressions that follow, denote by $(x^{(i)}, y^{(i)})$ the i-th example (x, y) from the training set and by $N = |\mathcal{D}_{train}|$ the number of training examples.

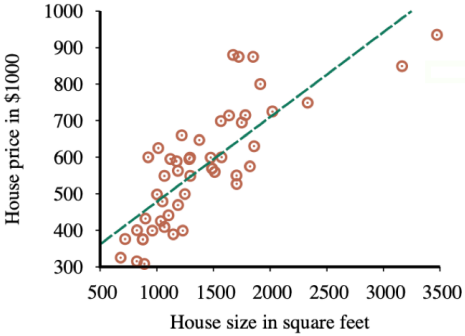
Then, from

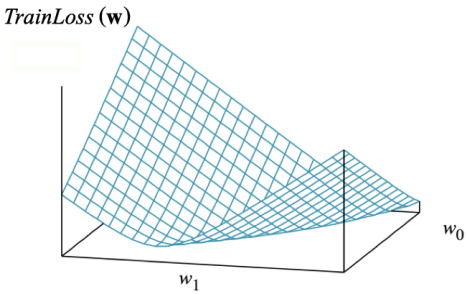
$$\frac{\partial}{\partial w_0} \sum_{i=1}^{N} (y^{(i)} - (w_1 x^{(i)} + w_0))^2 = 0; \qquad \frac{\partial}{\partial w_1} \sum_{i=1}^{N} (y^{(i)} - (w_1 x^{(i)} + w_0))^2 = 0$$

we obtain

$$w_1 = \frac{N \sum_i x^{(i)} y^{(i)} - \sum_i x^{(i)} \sum_i y^{(i)}}{N \sum_i (x^{(i)})^2 - \left(\sum_i x^{(i)}\right)^2}; \qquad w_0 = \left(\sum_i y^{(i)} - w_1 \sum_i x^{(i)}\right)/N$$

Univariate linear regression cont'd





Observe that the training loss function is **convex**.
This is true for every linear regression problem with an L_2 loss function.

Gradient descent

We need a general method for minimizing loss that can be applied to any loss function (such that we don't need to depend on finding the zeroes of its derivatives)

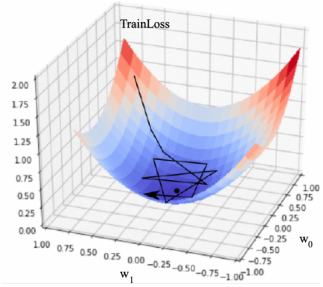
Idea: Search through a continuous weight space by incrementally modifying the parameters.

One such general local search algorithm is gradient descent algorithm:

w ← any point in the parameter space
while not converged **do**
    For each w_j **in w do**

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} TrainLoss(\mathbf{w})$$

step size (learning rate)



Univariate linear regression: Deriving the learning rule

Let us now derive the learning rule for the linear regression with the common L_2 loss.

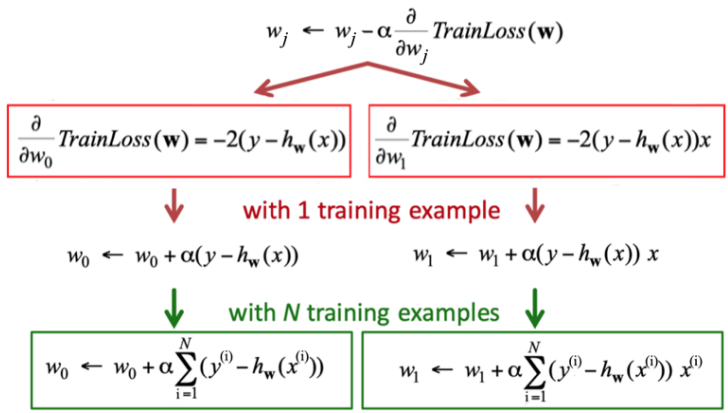
We start from a simplified case with one training example (x, y):

$$\begin{aligned}\frac{\partial}{\partial w_j} TrainLoss(\mathbf{w}) &= \frac{\partial}{\partial w_j}(y - h_{\mathbf{w}}(x))^2 \\ &= 2(y - h_{\mathbf{w}}(x))\frac{\partial}{\partial w_j}(y - h_{\mathbf{w}}(x)) \\ &= 2(y - h_{\mathbf{w}}(x))\frac{\partial}{\partial w_j}(y - (w_1 x + w_0))\end{aligned}$$

$$\frac{\partial}{\partial w_0} TrainLoss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x)) \qquad \frac{\partial}{\partial w_1} TrainLoss(\mathbf{w}) = -2(y - h_{\mathbf{w}}(x))x$$

We will easily generalize this result to the case with multiple samples knowing that the derivative of the sum is the sum of the derivatives.

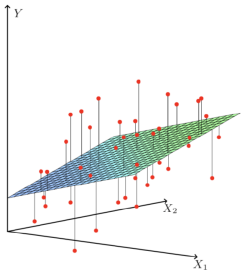
Univariate linear regression: Deriving the learning rule



Note: the factor 2 was folded into the unspecified learning rate α.

– Why is it interesting to consider this procedure at all when we can have the exact solutions for the weights as analytical expressions?

Multivariable linear regression



A generalization of univariate lin. regression: input examples are vectors $[x_1, ..., x_d]^\top$:

$$h_{\mathbf{w}}(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_n x_d = w_0 + \sum_j w_j x_j$$

To treat w_0 in the same way as all other w_j, introduce a dummy input $x_0 = 1$ and denote $\mathbf{x} = [x_0, ..., x_d]^\top$. Then

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \mathbf{w}^\top \mathbf{x} = \sum_j w_j x_j$$

We say that the output of the linear regression is the **score** $\mathbf{w} \cdot \mathbf{x}$.

Analytical solution for the weights

Let $\mathbf{X}$ be the data matrix and $\mathbf{y}$ a vector of all target values from the training set:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)\top} \\ \vdots \\ \mathbf{x}^{(N)\top} \end{bmatrix} = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & \cdots & x_d^{(1)} \\ \vdots & & & \\ x_0^{(N)} & x_1^{(N)} & \cdots & x_d^{(N)} \end{bmatrix} ; \qquad \mathbf{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix}$$

The training loss can now be expressed as

$$TrainLoss(\mathbf{w}) = \sum_{(\mathbf{x},y)\in\mathcal{D}_{train}} L_2(y, \mathbf{w}\cdot\mathbf{x}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 = (\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$$

Setting its gradient to zero yields the normal equation: $\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y}$. Thus,

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{(\mathbf{x},y)\in\mathcal{D}_{train}} L_2(y, \mathbf{w}\cdot\mathbf{x}) = \arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 = \underbrace{(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top}_{pseudoinverse}\mathbf{y}$$

In practice, it may be difficult to calculate the pseudoinverse of a large data matrix.

Least Means Squares (LMS) algorithm

Recall that our model is

$$h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} = \sum_j w_j x_j$$

Now optimize $\mathbf{w}$ using the gradient descent algorithm:

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} TrainLoss(\mathbf{w})$$

Under the L_2 loss, it boils down to the update (simultaneous for all $j = 0, ..., d$):

$$w_j \leftarrow w_j + \alpha \sum_{(\mathbf{x},y)\in\mathcal{D}_{train}} (y - h_{\mathbf{w}}(\mathbf{x}))x_j$$

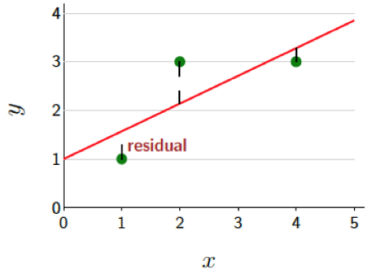
which reaches for this linear regression model the unique minimum $\mathbf{w}^*$.

Learning linear regression: Example

$$h_{\mathbf{w}}(x) = \mathbf{w} \cdot \mathbf{x}$$
$$\mathbf{w} = [1, 0.57]$$
$$\mathbf{x} = [1, x]$$

training data $\mathcal{D}_{\text{train}}$

x	y
1	1
2	3
4	3



$$L(\mathbf{x}, y, \mathbf{w}) = (h_{\mathbf{w}}(\mathbf{x}) - y)^2 \text{ squared loss}$$

$$L(1, 1, [1, 0.57]) = ([1, 0.57] \cdot [1, 1] - 1)^2 = 0.32$$
$$L(2, 3, [1, 0.57]) = ([1, 0.57] \cdot [1, 2] - 3)^2 = 0.74$$
$$L(4, 3, [1, 0.57]) = ([1, 0.57] \cdot [1, 4] - 3)^2 = 0.08$$

$$\text{TrainLoss}(\mathbf{w}) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(\mathbf{x}, y) \in \mathcal{D}_{\text{train}}} L(\mathbf{x}, y, \mathbf{w})$$

$$\text{TrainLoss}([1, 0.57]) = 0.38$$

Slide from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

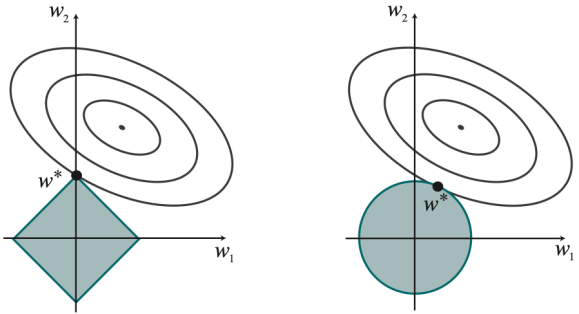
Multivariate linear regression: the influence of regularization

The use of regularization is common in multivariate regress. (to avoid overfitting)

$$Cost(h) = EmpLoss(h) + \lambda Complexity(h)$$

A common class of regularization functions is

$$Complexity(h_{\mathbf{w}}) = \ell_p(\mathbf{w}) = \sum_j |w_j|^p$$



Multivariate linear regression: Ridge and LASSO regression

Consider squared error loss and ℓ_p-regularization.

$$Cost(h_{\mathbf{w}}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \ell_p(\mathbf{w}); \qquad \mathbf{w}^* = \arg\min_{\mathbf{w}} Cost(h_{\mathbf{w}});$$

For $p = 2$, this is known as the Ridge regression or Tikhonov regularization:

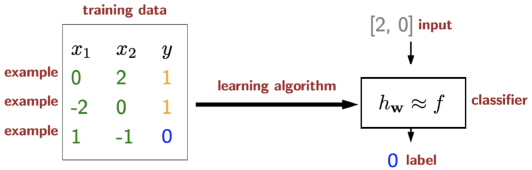
$$Cost(h_{\mathbf{w}}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 = \sum_{(x,y)\in\mathcal{D}_{train}} \Big(y - h_{\mathbf{w}}(x)\Big)^2 + \lambda \sum_j w_j^2$$

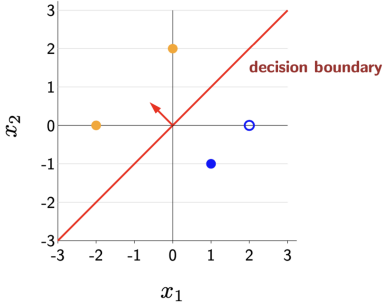
For $p = 1$, we have LASSO (Least Absolute Shrinkage and Selection Operator) regression:

$$Cost(h_{\mathbf{w}}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2 + \lambda \|\mathbf{w}\|_1 = \sum_{(x,y)\in\mathcal{D}_{train}} \Big(y - h_{\mathbf{w}}(x)\Big)^2 + \lambda \sum_j |w_j|$$

Outline

Linear classification framework



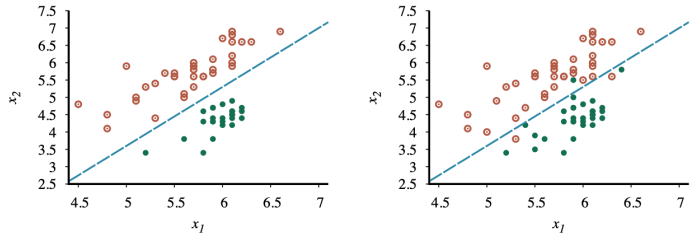


Design choices:

- Model selection – hypothesis class $\mathcal{H}$ (Which predictors are possible?)
- Model optimization – the best hypothesis within $\mathcal{H}$ (find the best predictor)
  - How to compute the best predictor? **learning algorithm**
    - ★ How good is a predictor? **loss function**

Slide adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Linear classifier with a hard threshold



A decision boundary is a line (or a surface, in higher dimensions) that separates the two classes

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear classifier with a hard threshold

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

We can think of the classifier h as the result of passing the linear operation $\mathbf{w} \cdot \mathbf{x}$ through a threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\mathbf{w} \cdot \mathbf{x}), \quad \text{where} \quad Threshold(z) = \begin{cases} 1 & z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

A simple update rule, identical to the update rule for linear regression:

$$w_j \leftarrow w_j + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_j$$

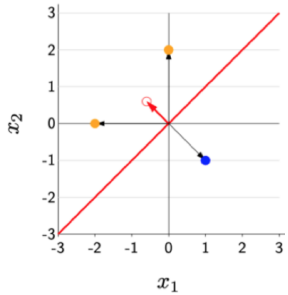
called **perceptron learning rule** converges to the perfect linear separator (provided that data are linearly separable) Reason about the meaning and effects of this rule!

Binary classification: Example

$$h_{\mathbf{w}}(\mathbf{x}) = Threshold(\overbrace{[-0.6, 0.6]}^{\mathbf{w}} \cdot [x_1, x_2])$$

$$Threshold(z) = \begin{cases} 1 & \text{if } z \geqslant 0 \\ 0 & \text{if } z < 0 \end{cases}$$

x_1	x_2	$f(x)$
0	2	1
-2	0	1
1	-1	0



$$h_{\mathbf{w}}([0,2]) = Thr([-0.6, 0.6] \cdot [0,2]) = Thr(1.2) = 1$$
$$h_{\mathbf{w}}([-2,0]) = Thr([-0.6, 0.6] \cdot [-2,0]) = Thr(1.2) = 1$$
$$h_{\mathbf{w}}([1,-1]) = Thr([-0.6, 0.6] \cdot [1,-1]) = Thr(-1.2) = 0$$

Decision boundary: $\mathbf{x}$ such that $\mathbf{w} \cdot \mathbf{x} = 0$

Adapted from M. Charikar and Koyejo: Artificial Intelligence: Principles and Techniques (Stanford)

Problems with hard threshold

The hard nature of the threshold causes some problems:

- The hypothesis $h_{\mathbf{w}}(\mathbf{x})$ is not differentiable and is in fact a discontinuous function of its inputs and its weights
- This makes learning with the perception rule very unpredictable
- Furthermore, the linear classifier always announces a "completely confident" prediction 0 or 1. We often need more graduated predictions

These problems are alleviated by softening the threshold function: approximating a hard threshold with a continuous, differentiable function

Logistic function

The **logistic** (a.k.a. **sigmoid**) function

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

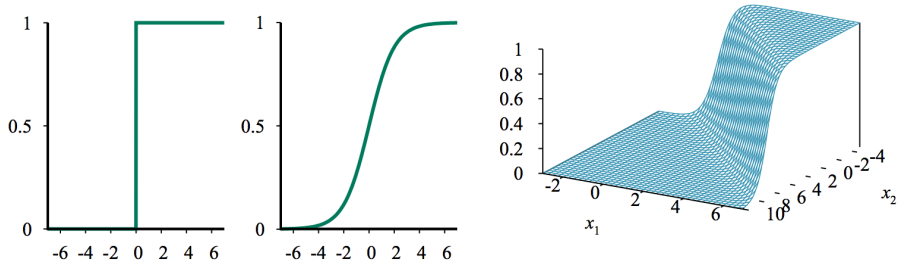
was first published by a Belgian mathematician P.F. Verhulst in the 19th century.

For the logistic regression:

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Pierre François Verhulst (1804-1849) born in Brussels, a mathematician and a doctor in number theory from Ghent University

Logistic regression



- The output of the logistic function can be interpreted as a probability of belonging to the class labeled 1
- The process of fitting the weights with this model is called logistic regression

Least-square error logistic regression

We first derive the update rule for the logistic regression under L_2 loss.

Let g denote the logistic function and g' its derivative. As we did for linear regression, we will use the chain rule for the derivatives: $\partial g(f(x))/\partial x = g'(f(x))(\partial f(x)/\partial x)$

We start again from a simplified case with one training example $(\mathbf{x}, y)$. The derivation is similar as for the linear regression but now $h_w(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})$, so we have:

$$\begin{aligned}
\frac{\partial}{\partial w_j} TrainLoss(\mathbf{w}) &= \frac{\partial}{\partial w_j}(y - h_{\mathbf{w}}(\mathbf{x}))^2 \\
&= 2(y - h_{\mathbf{w}}(\mathbf{x}))\frac{\partial}{\partial w_j}(y - h_{\mathbf{w}}(\mathbf{x})) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x}))g'(\mathbf{w} \cdot \mathbf{x})\frac{\partial}{\partial w_j}(\mathbf{w} \cdot \mathbf{x}) \\
&= -2(y - h_{\mathbf{w}}(\mathbf{x}))g'(\mathbf{w} \cdot \mathbf{x})x_j
\end{aligned}$$

Least-square error logistic regression

We derived:

$$\frac{\partial}{\partial w_j} TrainLoss(\mathbf{w}) = \frac{\partial}{\partial w_j}(y - h_{\mathbf{w}}(\mathbf{x}))^2 = -2(y - h_{\mathbf{w}}(\mathbf{x}))g'(\mathbf{w} \cdot \mathbf{x})x_j$$

The derivative of the logistic function satisfies $g'(z) = g(z)(1 - g(z))$, so we have

$$g'(\mathbf{w} \cdot \mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x})(1 - g(\mathbf{w} \cdot \mathbf{x})) = h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))$$

and the weight update for minimizing the loss is

$$w_j \leftarrow w_j + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x}))x_j$$

Note: this rule was derived for one training example (stochastic gradient descent)

– How does it generalize to the update rule based on N examples?

Alternative formal representations for linear classification

We used the convention $y \in \{0, 1\}$ for which the classification hypothesis is

$$h_{\mathbf{w}}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

If we use instead the label set $y \in \{-1, +1\}$, we can have a more compact formulation in some expressions. With the label set $\{-1, +1\}$, we have:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x})$$

Remark (Linear classification with $\{-1, +1\}$ labels)

In this formulation, the **prediction is the sign of the score**. *The* **score** $\mathbf{w} \cdot \mathbf{x}$ *of an example* (x, y) *tells us how* **confident** *we are in predicting* $+1$ *and the* **margin** $(\mathbf{w} \cdot \mathbf{x})y$ *tells us how* **correct** *we are.*

Alternative formal representations for linear classification

Convention $y \in \{0, 1\}$

score: $z = \mathbf{w} \cdot \mathbf{x}$

prediction: $\hat{y} = h_{\mathbf{w}}(\mathbf{x}) = Threshold(z)$

perceptron update rule:

$$w_j \leftarrow w_j + \alpha(y - h_{\mathbf{w}}(\mathbf{x}))x_j$$

Convention $y \in \{-1, 1\}$

score: $z = \mathbf{w} \cdot \mathbf{x}$

prediction: $\hat{y} = h_{\mathbf{w}}(\mathbf{x}) = \text{sign}(z)$

perceptron update rule:

if $h_{\mathbf{w}}(\mathbf{x}) = y$

do nothing

else

$$w_j \leftarrow w_j + \alpha y x_j$$

- Regardless of the convention, the weights are effectively updated in the same way!
- Holds always (the form of the update rule gets adapted accordingly)

Next lecture

- Optimization in ML
- Stochastic gradient descent
- Logistic regression