COGS 118A, Winter 2020

Supervised Machine Learning Algorithms

Lecture 3: Decision stump and Estimation

The pace of the lectures so far:

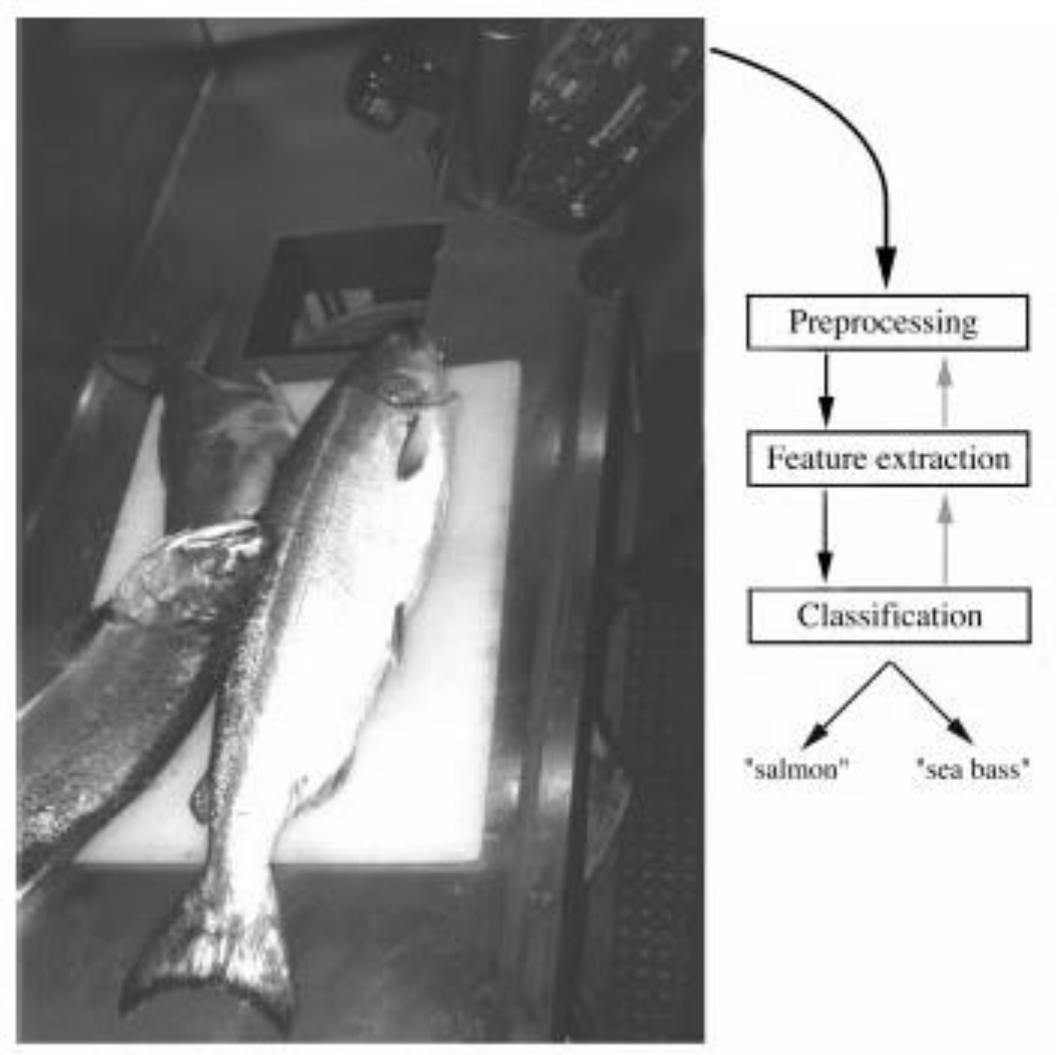
A. Too fast

B. Too slow

C. About right

D. Not sure

An example


Preprocessing
Feature extraction
Classification
"salmon"
"sea bass"

Summary of the problem

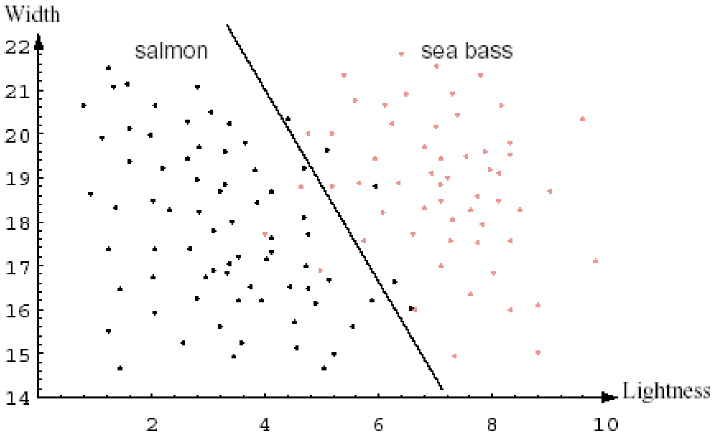
$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

$\mathbf{x} = (x_1, ..., x_m), x_i \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^m$

$y \in \{0, 1\}$ $y = 0$: negative
$y = 1$: positive

Classifier: $f(\mathbf{x}; W) \in \{0, 1\}$

Model parameter to be learned: W



Training error:

$$e_{training} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

Training errors

If we ignore the overfitting problem for now, nearly all supervised classifiers are learned by minimizing the training error (either implicitly or explicitly)

$$\text{Minimize } e_{training}$$

The definition of training error varies depending on the types of classifier.

Typically:

$$\text{Minimize } \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

Question?

Given a input set and a choice of classifier:

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

Classifier: $f(\mathbf{x}; W) \in \{0, 1\}$

If we try our best, what is the worst possible training error (assuming equal number of positives and negatives)?

$$e_{training} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

A. 0
B. 0.25
[✓ C. 0.5]
D. 0.75
E. 1.0

Question?

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

Classifier: $f(\mathbf{x}; W) \in \{0, 1\}$

$$e_{training} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

$e_{training} \leq 0.5$, if we try our best

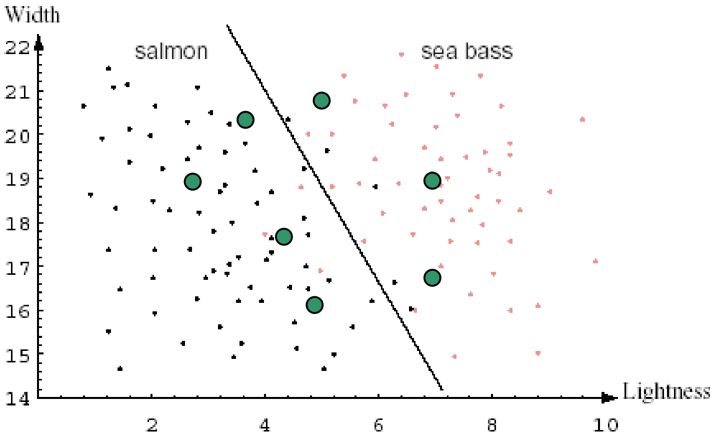
Why?

If $e_{training} > 0.5$

we simply define $f^{(reverse)}(\mathbf{x}_i; W) = 1 - f(\mathbf{x}_i; W)$

Testing

We use the training set to train a classifier $f(\mathbf{x}; W)$.



Given a set of testing data, $S_{testing}$, we make the prediction of each input and evaluate the algorithm.

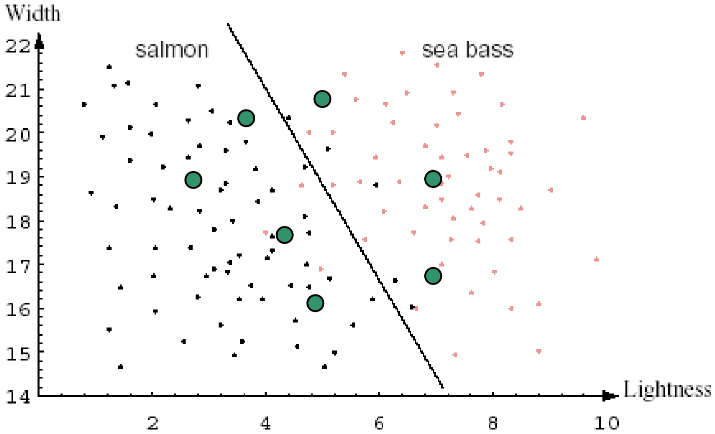
$$S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}.$$

For each $\mathbf{x}_i$ we want to predict its y_i.

y_i is given to evaluate the quality of a classifier and is not given in reality.

Testing

We use the training set to train a classifier $f(\mathbf{x}; W)$.



$$S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$e_{testing} = \frac{1}{q} \sum_{i=1}^{q} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$$

Training Error and Testing Error

$e_{training} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$ $\quad e_{training} \in [0, 1]$

$e_{testing} = \frac{1}{q} \sum_{i=1}^{q} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W))$ $\quad e_{testing} \in [0, 1]$

Both training and testing errors are between 0 and 1.

However in nearly all situations for any classifier, testing error is always no smaller than training error.

$$e_{testing} \geq e_{training}$$

The difference between them is called generalization error.

When $e_{testing} >> e_{training}$,
we call it "overfitting"

Training Error and Testing Error

$$e_{training} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W)) \qquad e_{training} \in [0,1]$$

$$e_{testing} = \frac{1}{q}\sum_{i=1}^{q} \mathbf{1}(y_i \neq f(\mathbf{x}_i; W)) \qquad e_{testing} \in [0,1]$$

$$e_{testing} \geq e_{training}$$

Why?

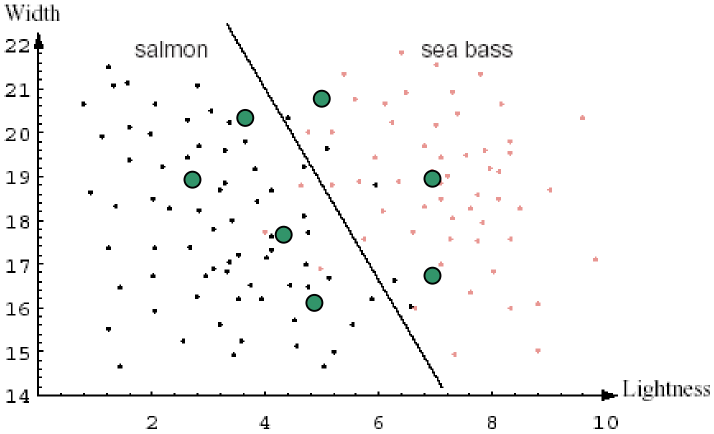
We always make better “prediction” to history than in the future.

Future is UNKNOWN whereas the training process have the full access to the history.

Testing error

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$e_{testing} = e_{training} + generalization(f)$$



$$e_{testing} = \frac{1}{q}\sum_{i=1}^{q} \mathbf{1}(y_i \neq f(\mathbf{x}_i))$$

$$e_{testing} = 0.5?$$

1. $e_{testing} = 0.5 + 0.0$
2. $e_{testing} = 0.0 + 0.5$

Question: What are the two extreme cases leading to the same testing error=0.5 (Assume we have the same number of positives and negatives.)

Testing error

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\} \qquad S_{testing} = \{(\mathbf{x}_i, y_i), i = 1..q\}$$

$$e_{testing} = e_{training} + generalization(f)$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

- We make a completely random guess even after training (didn’t learn anything and attained an error of 0.5).
- A random guess is however highly generalizable, which doesn’t incur any generalization error.

Case 2:

2. $e_{testing} = 0.0 + 0.5$

- We make perfect classifications on the training data (memorizing the labels for every training sample).
- Merely memorizing the training samples with their corresponding classification labels is however highly non-generalizable, incurring the largest generalization error (no identical training sample will appear in testing).

Both are extreme cases that lead to trivial ML models.

Testing error

$$e_{testing} = e_{training} + generalization(f)$$

Case 1:

1. $e_{testing} = 0.5 + 0.0$

Case 2:

2. $e_{testing} = 0.0 + 0.5$

Both are extreme cases that lead to trivial models that we should avoid.

Ideally: $e_{testing} = 0.0 + 0.0$

A classification model that is perfect after training and makes no error in testing.

In practice: $e_{testing} = 0.05 + 0.1$

A classification model that does well after training and generalizes reasonably well in testing.

Nearly all practical ML algorithms seek a compromise!

Summary of the classification problem

$\mathbf{x} = (x_1, x_2, \ldots)$

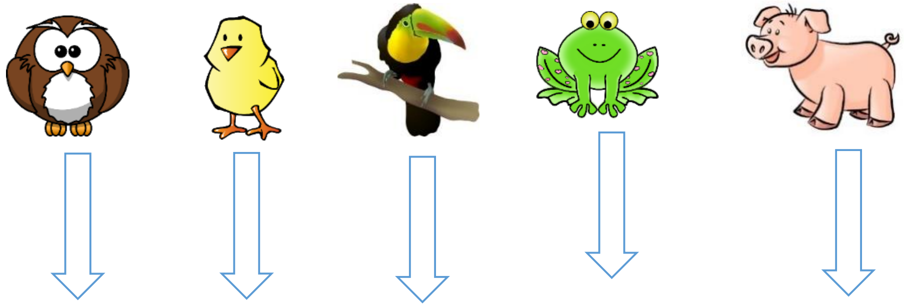
$y = 1(bird)$

x_1: color

x_2 weight

$\ldots$

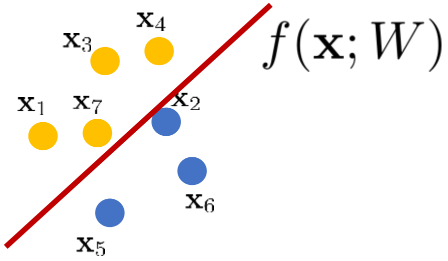
$$S_{training} =$$



$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4), (\mathbf{x}_5, y_5)\}$$

Train classifier $f(\mathbf{x}; W)$

W: model parameter

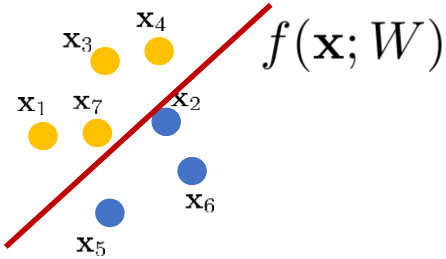


Three key variables we are dealing with

Input: $\mathbf{x} = (x_1, x_2, ...)$

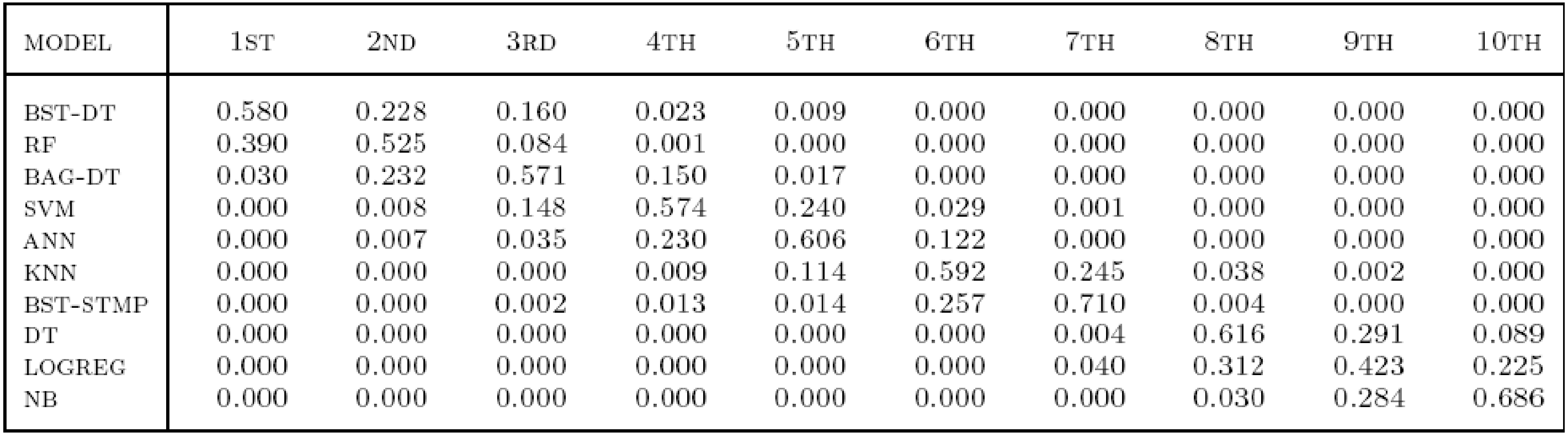
Label: $y \in 0, 1$

Model parameter: W



Empirical Comparisons of Different Algorithms

Caruana and Niculesu-Mizil, ICML 2006



MODEL	1ST	2ND	3RD	4TH	5TH	6TH	7TH	8TH	9TH	10TH
BST-DT	0.580	0.228	0.160	0.023	0.009	0.000	0.000	0.000	0.000	0.000
RF	0.390	0.525	0.084	0.001	0.000	0.000	0.000	0.000	0.000	0.000
BAG-DT	0.030	0.232	0.571	0.150	0.017	0.000	0.000	0.000	0.000	0.000
SVM	0.000	0.008	0.148	0.574	0.240	0.029	0.001	0.000	0.000	0.000
ANN	0.000	0.007	0.035	0.230	0.606	0.122	0.000	0.000	0.000	0.000
KNN	0.000	0.000	0.000	0.009	0.114	0.592	0.245	0.038	0.002	0.000
BST-STMP	0.000	0.000	0.002	0.013	0.014	0.257	0.710	0.004	0.000	0.000
DT	0.000	0.000	0.000	0.000	0.000	0.000	0.004	0.616	0.291	0.089
LOGREG	0.000	0.000	0.000	0.000	0.000	0.000	0.040	0.312	0.423	0.225
NB	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.030	0.284	0.686

Overall rank by mean performance across problems and metrics (based on bootstrap analysis).

BST-DT: boosting with decision tree weak classifier

RF: random forest

BAG-DT: bagging with decision tree weak classifier

SVM: support vector machine

ANN: neural nets

KNN: k nearest neighboorhood

BST-STMP: boosting with decision stump weak classifier

DT: decision tree

LOGREG: logistic regression

NB: naïve Bayesian

It is informative, but by no means final.

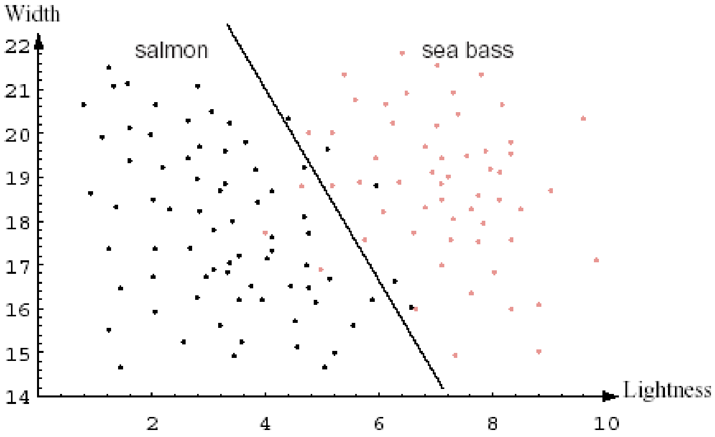
Decision boundary

Let **x** be the input vector (observation) and y be its label:

Often, we are given a set of training data

$$S = \{(\mathbf{x}_i, y_i), i = 1..n\} \quad \mathbf{x} = (x_1, ..., x_m), x_i \in \mathcal{R}, \quad \mathbf{x} \in \mathcal{R}^m$$

A classifier f(**x**):



Decision boundary:
$\{\mathbf{x}_i, f(\mathbf{x}_i) = 0\}$

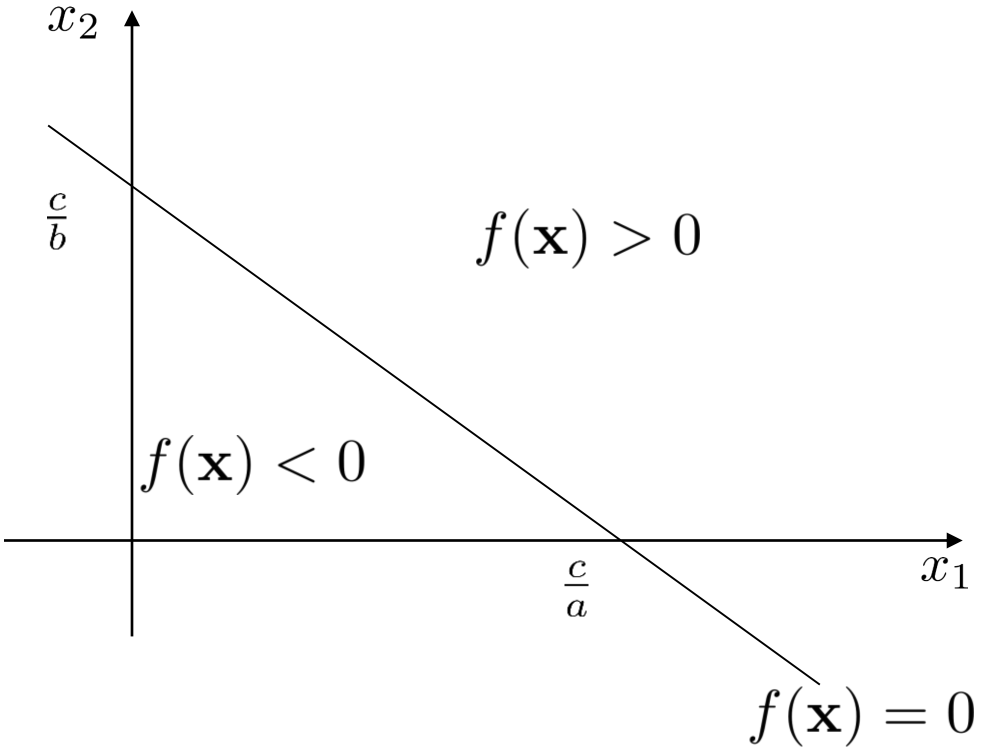
Decision boundary

Let **x** be the input vector (observation) and y be its label:

Decision boundary:

$\{\mathbf{x}_i, f(\mathbf{x}_i) = 0\}$

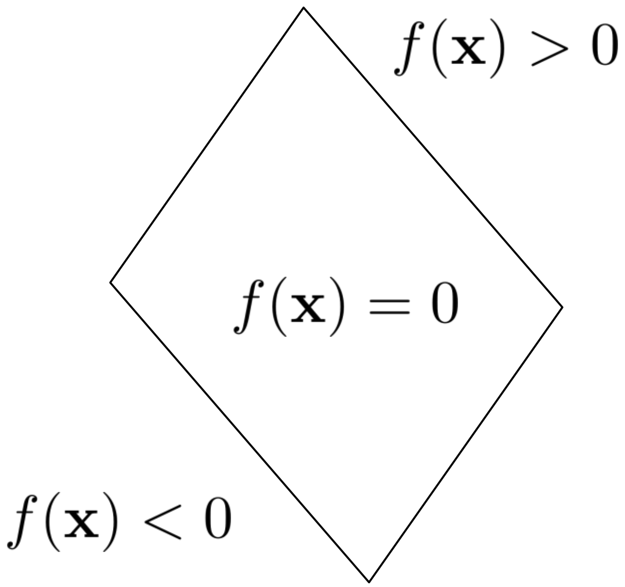
$$f(\mathbf{x}) = a \times x_1 + b \times x_2 - c$$



Decision boundary

Let **x** be the input vector (observation) and y be its label:

Decision boundary:
$\{\mathbf{x}_i, f(\mathbf{x}_i) = 0\}$



Decision boundary

“Decision boundary” is one of the most important concepts in machine learning.

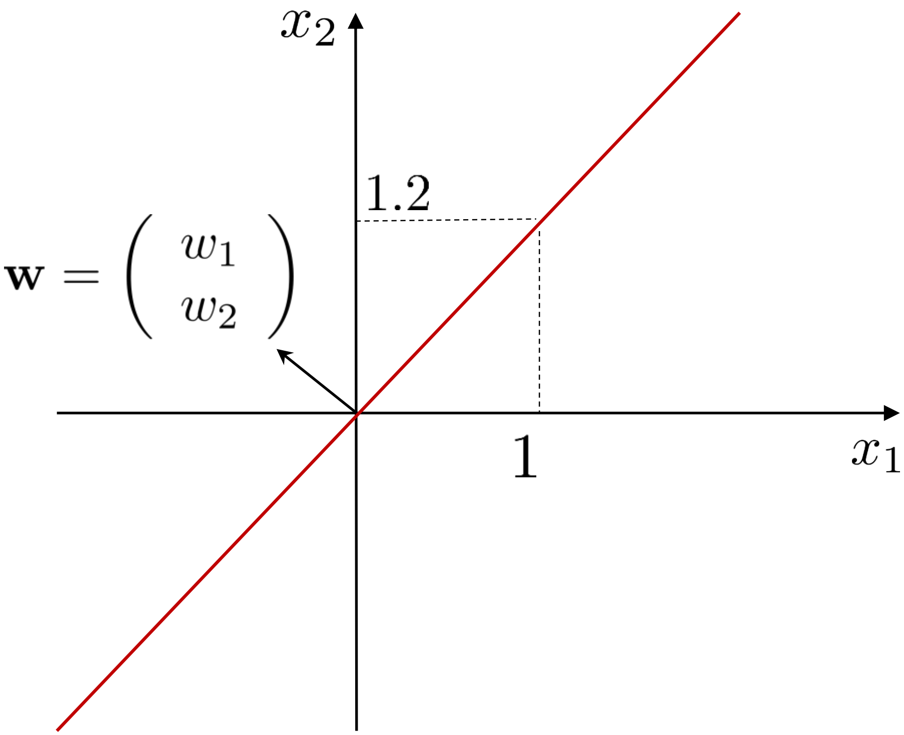
The decision boundary of a binary classifier refers to the set of data samples that are “on the fence” between making the decision being positive or negative: that is being 50%-50% for classification.

Decision boundary is a characteristic of your learned model so it varies due to the choice of your classification model and how it has been trained.

Typically, decision boundary: $\{\mathbf{x}_i, f(\mathbf{x}_i) = 0\}$

Line and vector

an example:



$\mathbf{w}$ is the normal direction of the line

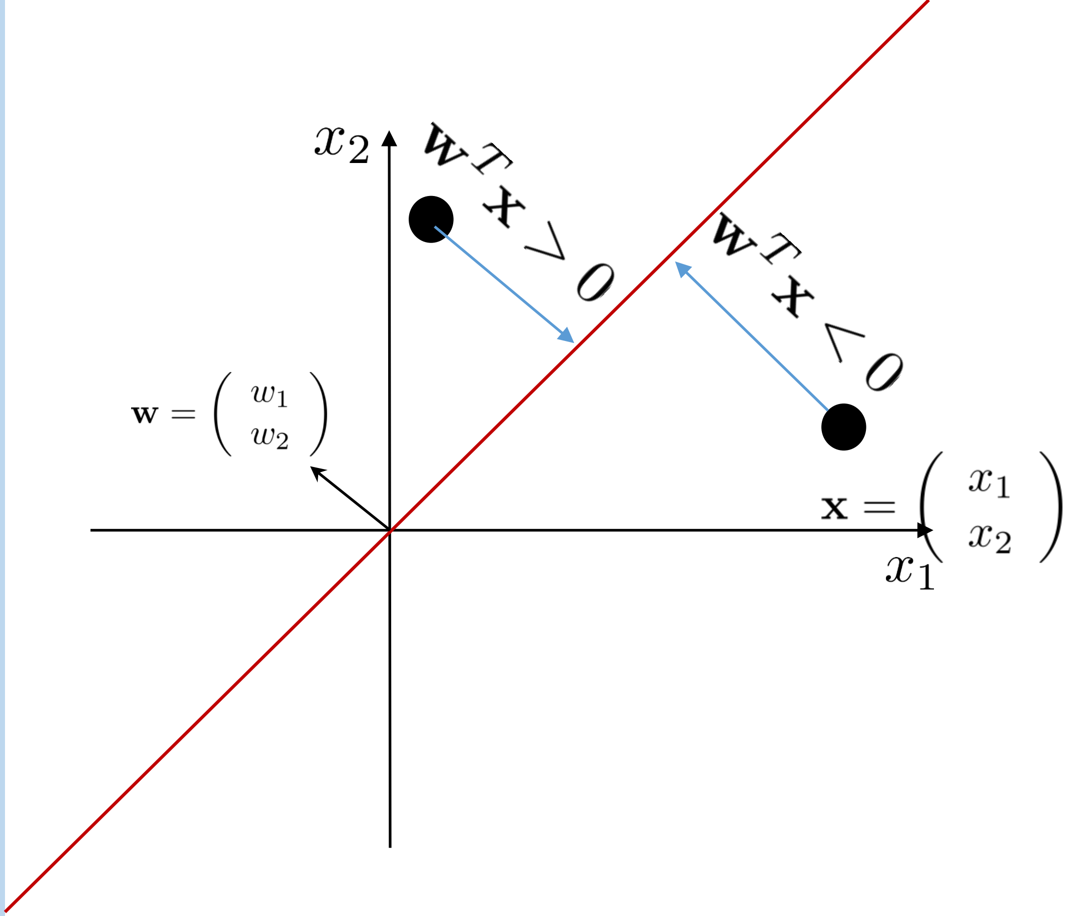
Often: $||\mathbf{w}||_2 = 1$: a unit vector

$$x_2 = 1.2 \times x_1$$

$$\mathbf{w} = \begin{pmatrix} -\frac{1.2}{\sqrt{2.44}} \\ \frac{1}{\sqrt{2.44}} \end{pmatrix}$$

Distance to the decision boundary

(arguably the most important concept in machine learning)



x_2
$\mathbf{w}^T\mathbf{x} > 0$
$\mathbf{w}^T\mathbf{x} < 0$
$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
x_1

The distance (signed) of any point $\mathbf{x}$ to the line is:

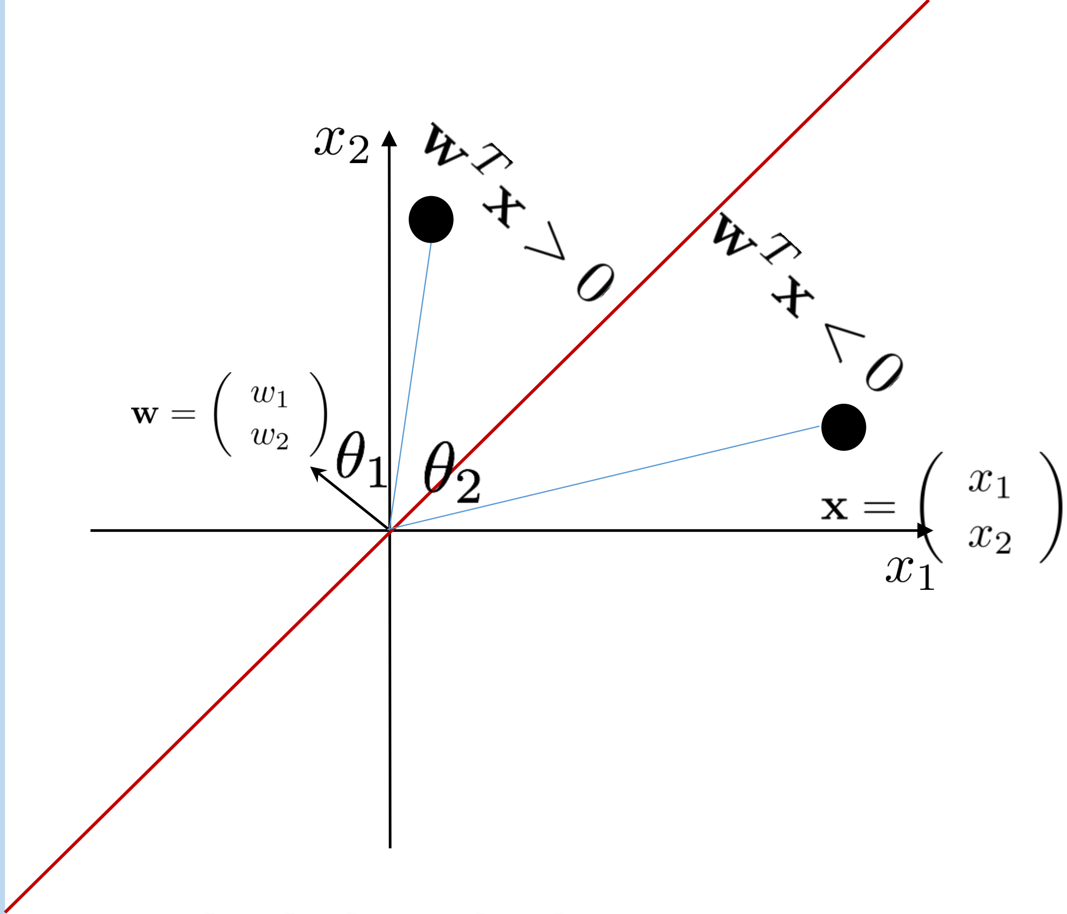
$$\mathbf{w}^T\mathbf{x} \equiv < \mathbf{w}, \mathbf{x} >$$

$\mathbf{w}^T\mathbf{x} > 0$: above the line

$\mathbf{w}^T\mathbf{x} < 0$: below the line

Distance to the decision boundary

(arguably the most important concept in machine learning)


x_2
$\mathbf{w}^T\mathbf{x} > 0$
$\mathbf{w}^T\mathbf{x} < 0$
$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$
θ_1
θ_2
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
x_1

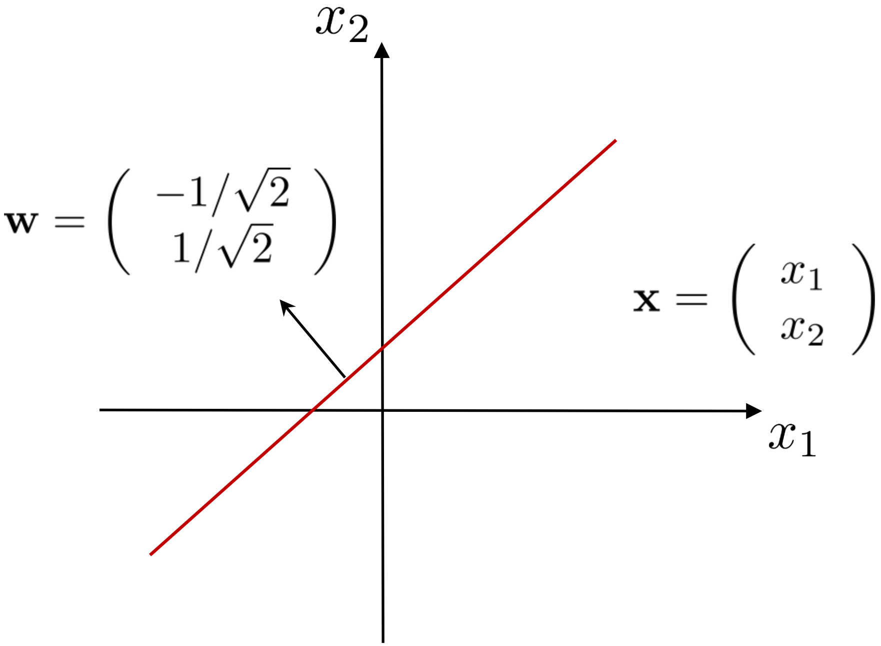
Why below the line is < 0 while above the line is > 0?

$$\mathbf{w}^T\mathbf{x} \equiv < \mathbf{w}, \mathbf{x} >$$

$$\mathbf{w}^T\mathbf{x} = \|\mathbf{w}\| \cdot \|\mathbf{x}\| \cdot \cos\theta$$

It depends on the angle formed by w and x

Decision boundary?


x_2
$\mathbf{w} = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix}$
$\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$
x_1

Can the line (in red) be the decision boundary of the classifier below

$$\mathbf{w} = \begin{pmatrix} -1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \qquad y = \begin{cases} +1 & if\ \mathbf{w}^T\mathbf{x} > 0 \\ -1 & if\ \mathbf{w}^T\mathbf{x} < 0 \end{cases}?$$

A. Yes

B. No

C. It depends

Decision boundary

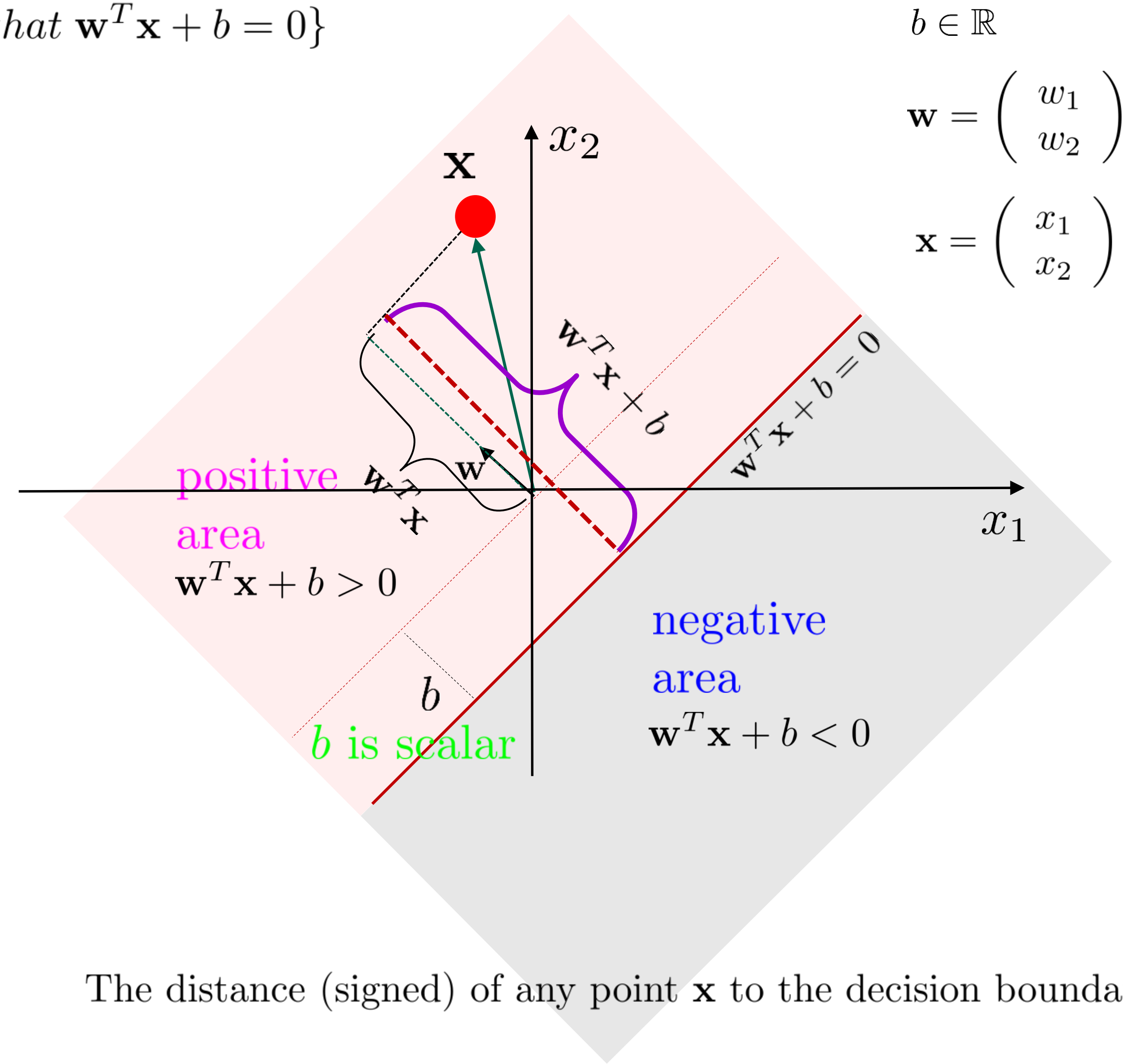
Decision boudary:

$\{\mathbf{x}; \forall \mathbf{x} \textit{ such that } \mathbf{w}^T\mathbf{x} + b = 0\}$

$b \in \mathbb{R}$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

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



$\mathbf{x}$
x_2
x_1
$\mathbf{w}$
$\mathbf{w}^T\mathbf{x}$
$\mathbf{w}^T\mathbf{x} + b$
$\mathbf{w}^T\mathbf{x} + b = 0$
positive area
$\mathbf{w}^T\mathbf{x} + b > 0$
negative area
$\mathbf{w}^T\mathbf{x} + b < 0$
b
b is scalar

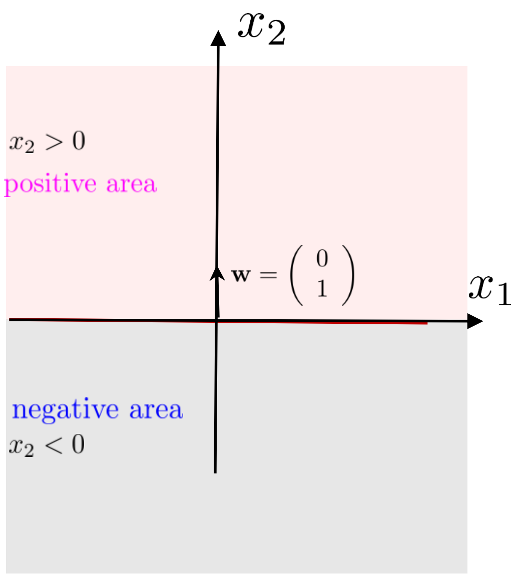
The distance (signed) of any point $\mathbf{x}$ to the decision boundary is: $\mathbf{w}^T\mathbf{x} + b$

Some typical examples

Assuming $\mathbf{w}$ being normalized: $||\mathbf{w}||_2 = \sqrt{w_1^2 + w_2^2} = 1.$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \qquad b \in \mathbb{R} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$0 \times x_1 + 1 \times x_2 = 0$$
$$\Downarrow$$
$$x_2 = 0$$



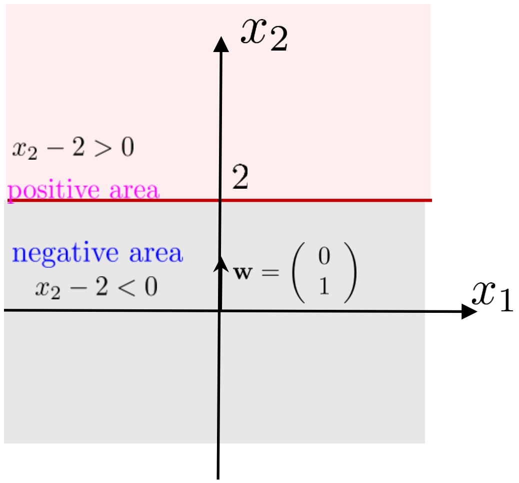
$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$b = 0$$

Decision boundary:

$$0 \times x_1 + 1 \times x_2 = 0$$
$$\Downarrow$$
$$x_2 = 0$$

$$0 \times x_1 + 1 \times x_2 - 2 = 0$$
$$\Downarrow$$
$$x_2 - 2 = 0$$



$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$b = -2$$

Decision boundary:

$$0 \times x_1 + 1 \times x_2 - 2 = 0$$
$$\Downarrow$$
$$x_2 - 2 = 0$$

Decision boundary

The decision boundary of a binary classifier refers to the set of data samples that are “on the fence” between making the decision being positive or negative: that is being 50%-50% for classification.

For a linear model, the decision boundary is a line/hyper-plane, depending upon the dimension of the data.

The model parameter **w** is along the normal direction that is orthogonal to the decision boundary.

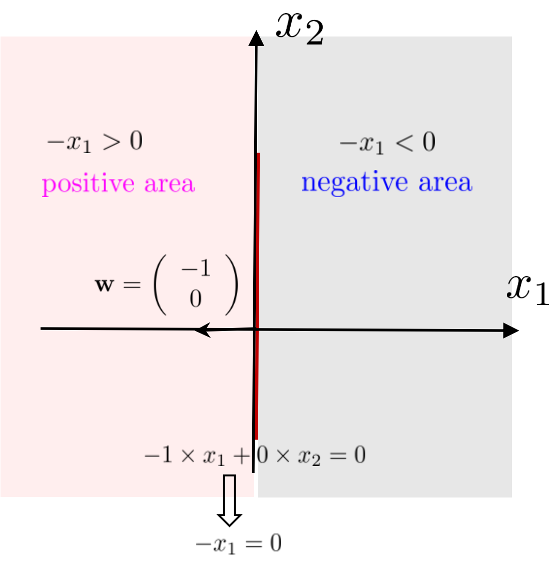
There is often a bias terms, b (scalar), refers to as the translation (shift) of the decision boundary.

Note that **w** itself is NOT the decision boundary.

Some typical examples

Assuming $\mathbf{w}$ being normalized: $||\mathbf{w}||_2 = \sqrt{w_1^2 + w_2^2} = 1.$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \qquad b \in \mathbb{R} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

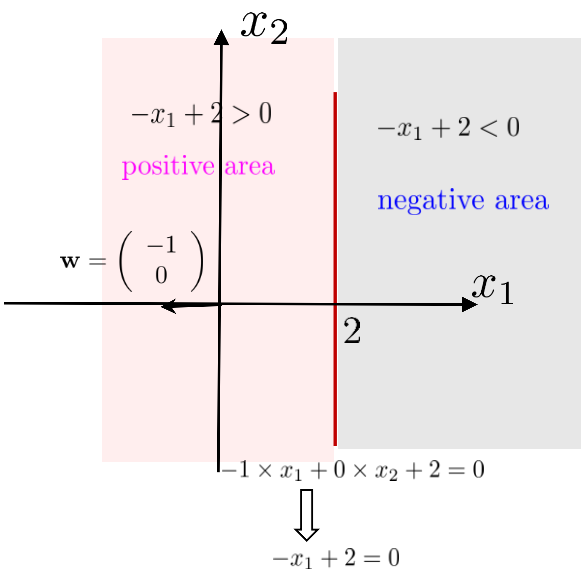

x_2
$-x_1 > 0$
positive area
$-x_1 < 0$
negative area
$\mathbf{w} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$
x_1
$-1 \times x_1 + 0 \times x_2 = 0$
$\Downarrow$
$-x_1 = 0$

$$\mathbf{w} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

$b = 0$

Decision boundary:

$$-1 \times x_1 + 0 \times x_2 = 0$$
$$\Downarrow$$
$$-x_1 = 0$$



$$\mathbf{w} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

$b = 2$

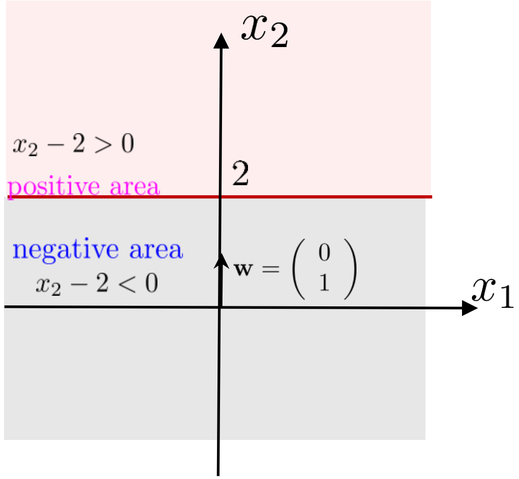
Decision boundary:

$$-1 \times x_1 + 0 \times x_2 + 2 = 0$$
$$\Downarrow$$
$$-x_1 + 2 = 0$$

Pay attention to the bias term b

The sign of b depends on w too!

$0 \times x_1 + 1 \times x_2 - 2 = 0$

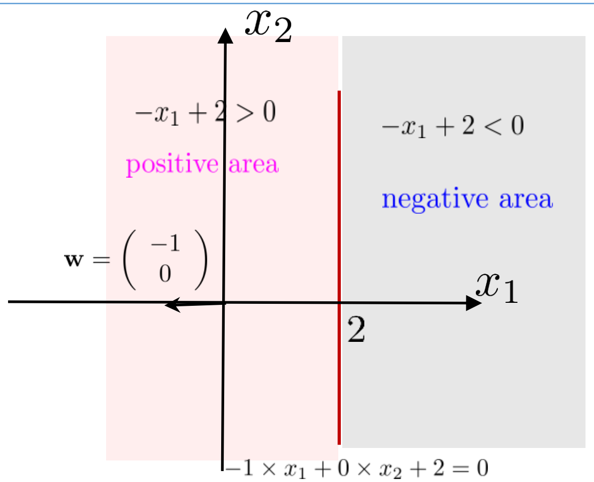


$$\mathbf{w} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

Decision boundary:

$$0 \times x_1 + 1 \times x_2 - 2 = 0$$

$$b = -2$$


x_2
$-x_1 + 2 > 0$
positive area
$-x_1 + 2 < 0$
negative area
$\mathbf{w} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$
x_1
2
$-1 \times x_1 + 0 \times x_2 + 2 = 0$

$$\mathbf{w} = \begin{pmatrix} -1 \\ 0 \end{pmatrix}$$

Decision boundary:

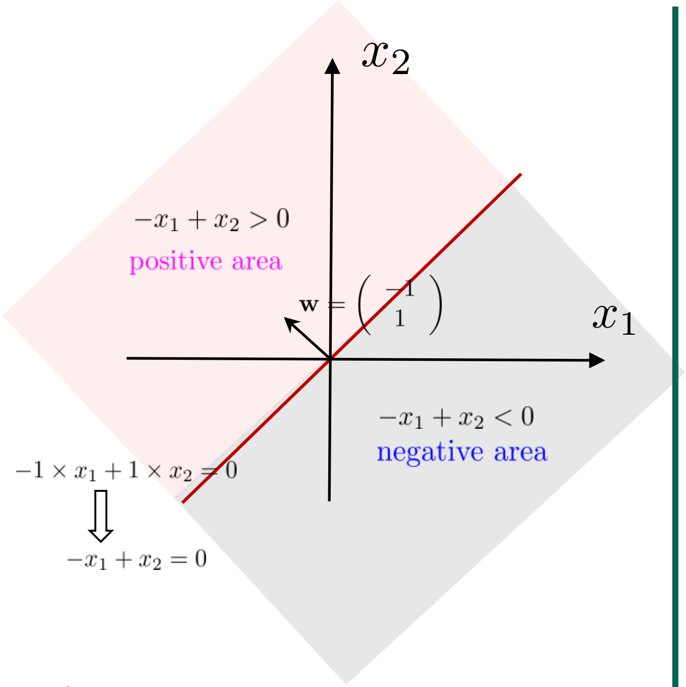
$$-1 \times x_1 + 0 \times x_2 + 2 = 0$$

$$b = 2$$

Some typical examples

Assuming $\mathbf{w}$ being normalized: $||\mathbf{w}||_2 = \sqrt{w_1^2 + w_2^2} = 1.$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix} \qquad b \in \mathbb{R} \qquad \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

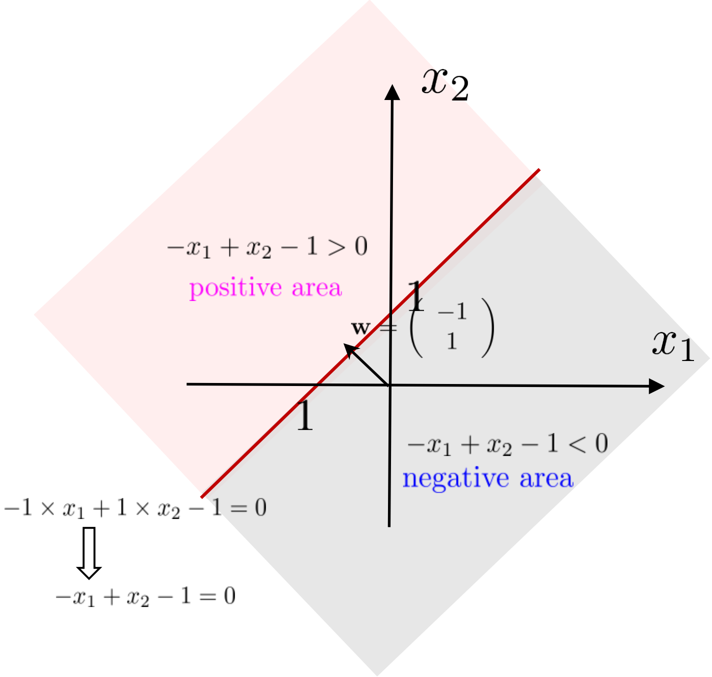


$$\mathbf{w} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$b = 0$

Decision boundary:

$$-1 \times x_1 + 1 \times x_2 = 0$$
$$\Downarrow$$
$$-x_1 + x_2 = 0$$



$$\mathbf{w} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

$b = -1$

Decision boundary:

$$-1 \times x_1 + 1 \times x_2 - 1 = 0$$
$$\Downarrow$$
$$-x_1 + x_2 - 1 = 0$$

Take home message

Any data sample (point) lying on the decision boundary receives a classification decision that is equally positive and negative.

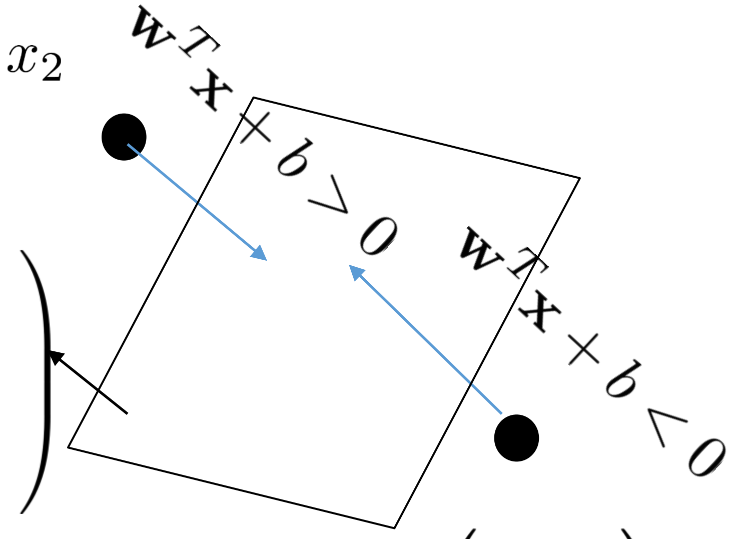
The decision boundary of a linear classifier is a hyper-plane.

The model parameter **w** is along the normal direction of the decision boundary, pointing to the positive samples.

The bias terms, b (scalar), refers to as the translation (shift) of the decision boundary.

Distance to the decision boundary

(arguably the most important concept in machine learning)


x_2
$\mathbf{w}^T\mathbf{x} + b > 0$
$\mathbf{w}^T\mathbf{x} + b < 0$

$$\mathbf{w} = \begin{pmatrix} w_1 \\ . \\ . \\ w_n \end{pmatrix}$$

$$\mathbf{w} \in \mathbb{R}^n$$

$$b \in \mathbb{R}$$

$$\mathbf{x} = \begin{pmatrix} x_1 \\ . \\ . \\ x_n \end{pmatrix}$$

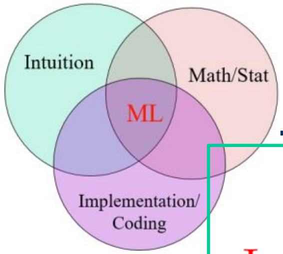
$$\mathbf{x} \in \mathbb{R}^n$$

The distance (signed) of any point $\mathbf{x}$ to the hyper-plane is:

$$\mathbf{w}^T\mathbf{x} + b \equiv < \mathbf{w}, \mathbf{x} > + b \equiv \mathbf{w} \cdot \mathbf{x} + b$$

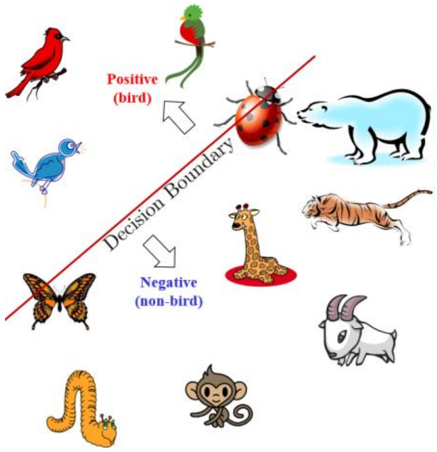
$\mathbf{w}^T\mathbf{x} + b > 0$: above the hyper-plane

$\mathbf{w}^T\mathbf{x} + b < 0$: below the hyper-plane



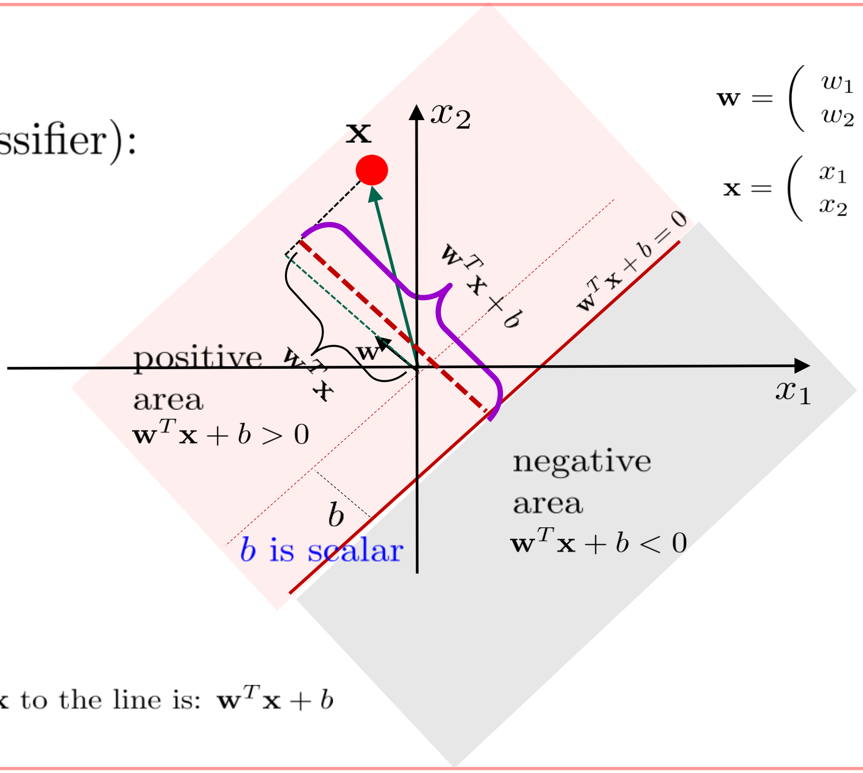
Recap: Decision Boundary

Intuition: Decision boundary of a discriminative classifier is a set that consists of possible samples that are on the border (typically 50% – 50%) of the separation between the positive and negative areas (for two-class classification).



Math:

Decision boudary (for a linear classifier):
$\{\mathbf{x}; \forall \mathbf{x}\ \textit{such that}\ \mathbf{w}^T\mathbf{x} + b = 0\}$


x_2
$\mathbf{x}$
$\mathbf{w}^T\mathbf{x} + b$
$\mathbf{w}^T\mathbf{x} + b = 0$
x_1
positive
area
$\mathbf{w}^T\mathbf{x} + b > 0$
$\mathbf{w}^T\mathbf{x}$
$\mathbf{w}$
negative
area
$\mathbf{w}^T\mathbf{x} + b < 0$
b
b is scalar

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$$

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

The distance (signed) of any point $\mathbf{x}$ to the line is: $\mathbf{w}^T\mathbf{x} + b$

We look at another simpler classifier that is learned using exhaustive search.

Decision Stump Classifier

Given a training set:

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

For each sample $\mathbf{x}$ with its corresponding label y:

$$\mathbf{x} = (x_1, ..., x_m), x_j \in \mathbb{R}, \qquad \mathbf{x} \in \mathbb{R}^m$$

$$y \in \{0, 1\}$$ sometimes also use $y \in \{-1, +1\}$

We look for a decision stump classifier $f(\mathbf{x}, j, Th)$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

Decision Stump Classifier

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

The optimal decision stump classifier $f(\mathbf{x}, j^*, Th^*)$ minimizes the training error:

$$e_{training} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i))$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th)$$

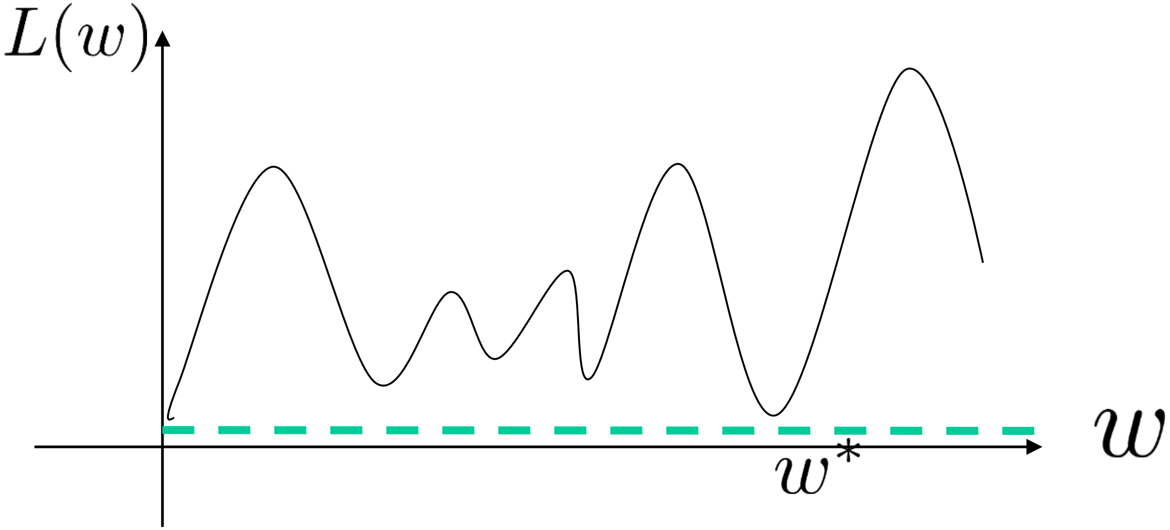
$$= \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Optimization: argmin

$$w^* = \arg\min_w L(w)$$

The operator $\arg\min$ defines the optimal value (in the argument of function $L()$) w^* that minimizes $L(w)$

$\arg\min L(w)$ doesn't return the value of $L(w)$

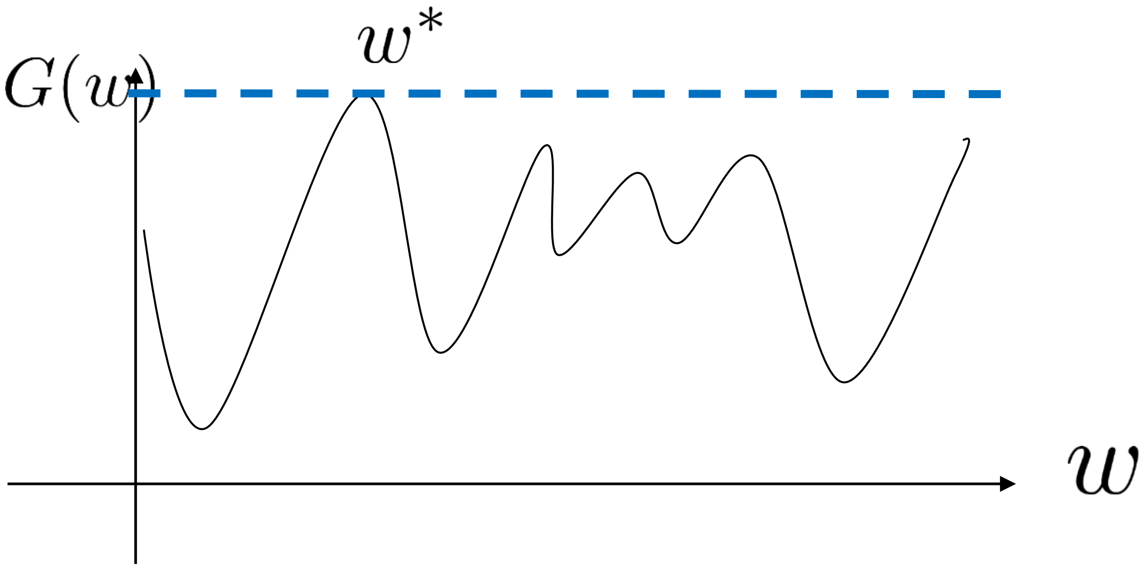


Optimization: argmax

$$w^* = \arg\max_w G(w)$$

The operator $\arg\max$ defines the optimal value (in the argument of function $G()$) w^* that maximizes $G(w)$

$\arg\max G(w)$ doesn't return the value of $G(w)$



argmin and argmax

argmin and argmax are two commonly used terms in machine learning and optimization.

The reason being we want to perform learning: a process in which the “best” model parameters are to be learned.

For example, who is the richest person in the world in 2019 (the answer concerns with the person not the amount of money this person has).

$$\text{person}^{richest} = \arg\max_{\text{person}} NetWorth(\text{person})$$

The answer is Jeff Bezos, not the net worth ($113 billion).

$$\text{person}^{richest} = \text{Jeff Bezos}$$

argmin and argmax

In addition:

$$w^* = \arg\min_w L(w)$$

$$= \arg\max_w -L(w)$$

$$\text{person}^{richest} = \arg\max_{\text{person}} NetWorth(\text{person})$$

$$= \arg\min_{\text{person}} -NetWorth(\text{person})$$

$$\text{person}^{richest} = \text{person}^{-poorest}$$

Decision Stump

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

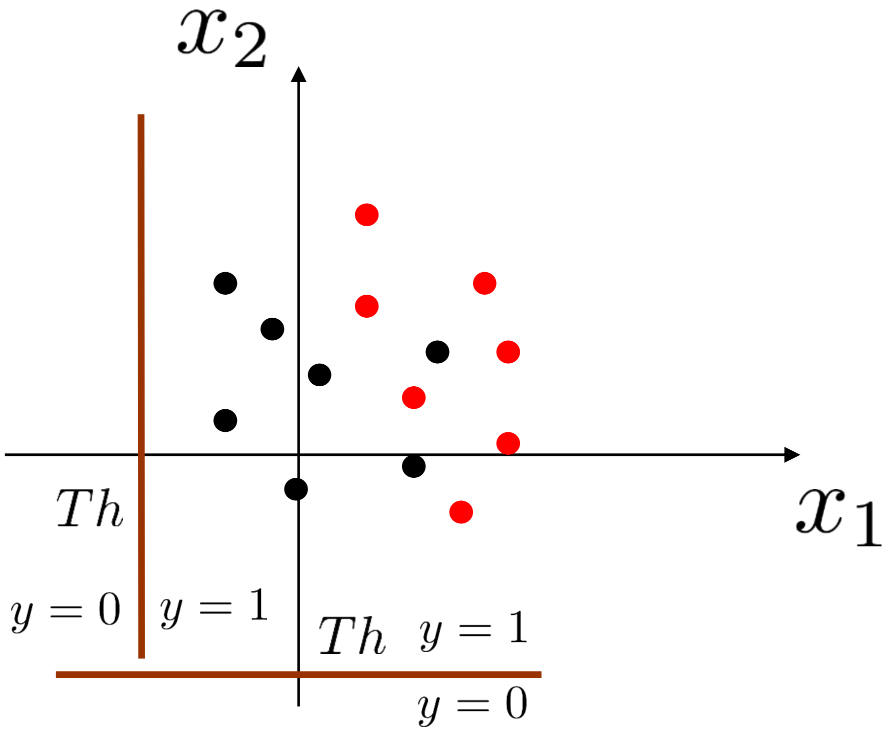
$\mathbf{x} = (x_1, x_2), x_1, x_2 \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^2$

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

● $y = 1$

● $y = 0$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$



Decision Stump

- Decision stump is one of the simplest classifiers.

- It learns to look at ONE feature only and finds the “best” threshold to make a corresponding decision.

- The decision stump classifier is a simple classifier that can be efficiently learned and implemented.

- Once trained, a decision stump classifier has an extremely low model complexity (feature ID + the threshold).

- Decision stump can be adopted in other classifiers as a basic component (e.g. decision tree and boosting).

Decision Stump - Training

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

$\mathbf{x} = (x_1, ..., x_m), x_j \in \mathbb{R}, \quad \mathbf{x} \in \mathbb{R}^m \qquad y \in \{0, 1\}$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th) = \arg\min_{j,Th} \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Pseudo-code:

```
for j=1 to m
    for $Th = min\_val\_j$ to $max\_val\_j$
        Compute $e_{training}(j, Th) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$
Report the $(j^*, Th^*)$ with the lowest $e_{training}(j, Th)$
```

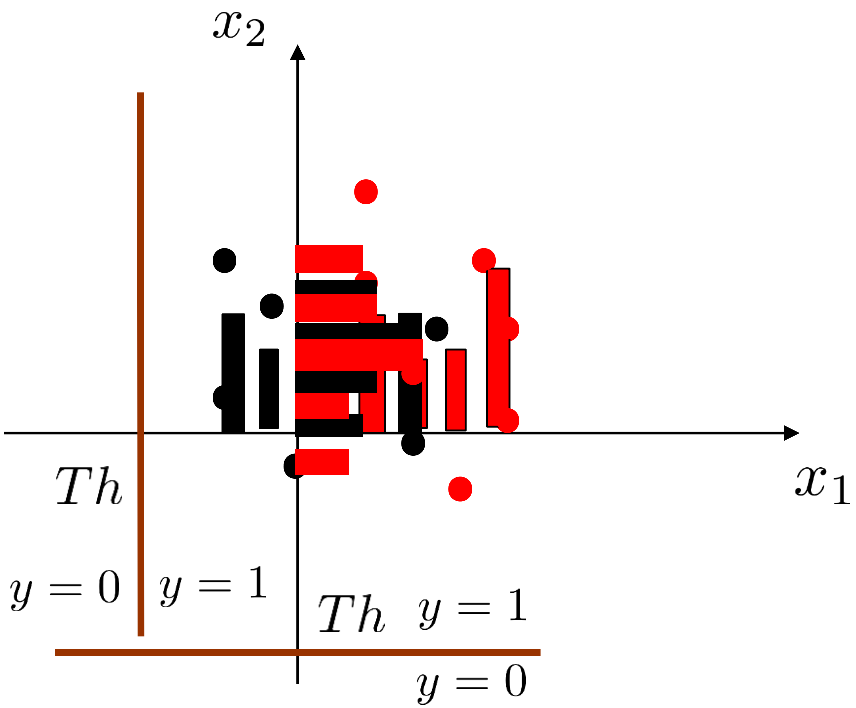
Decision Stump- Training

$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

Pseudo-code:

```
for j=1 to m
    for $Th = min\_val\_j$ to $max\_val\_j$
        Compute $e_{training}(j, Th) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$
Report the $(j^*, Th^*)$ with the lowest $e_{training}(j, Th)$
```



Decision Stump- Training

$$S_{training} = \{(\mathbf{x}_i, y_i), i = 1..n\}$$

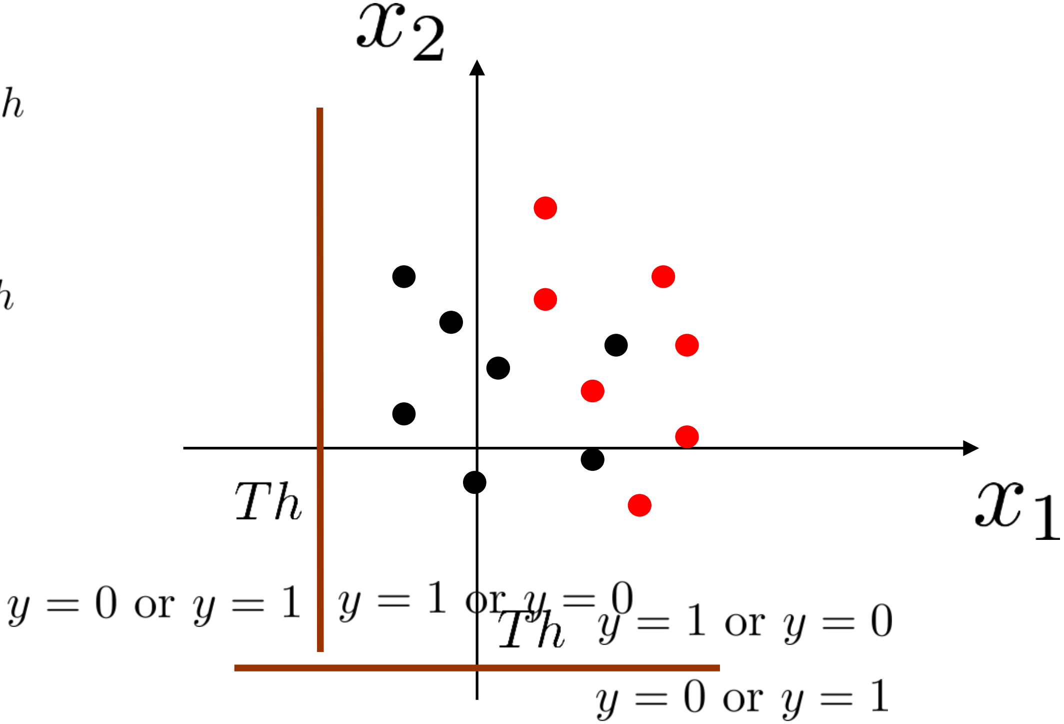
$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

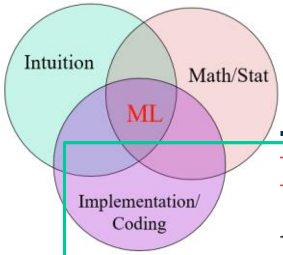
Pseudo-code:

```
for j=1 to m
    for $Th = min\_val\_j$ to $max\_val\_j$
        Compute $e_{training}(j, Th) = min(\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_{>}(\mathbf{x}_i, j, Th)), \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_{<}(\mathbf{x}_i, j, Th)))$
Report the $(j^*, Th^*)$ with the lowest $e_{training}(j, Th)$
```

$$f_{>}(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$$f_{<}(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \leq Th \\ 0 & otherwise \end{cases}$$





Recap: Decision Stump Classifier

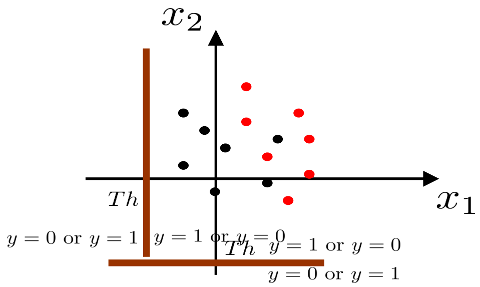
Intuition: Find the best feature at the best threshold to separate the two classes. After training, only one feature from the input is needed for testing/prediction. The model complexity (only the feature index and one threshold are stored) is extremely low; its computational complexity is also very low) since only one operation (comparison) is needed. So even the decision stump classifier might be restricted in its classification power, it is still practically very useful in many situations, e.g. to be adopted in the AdaBoost classifier as weak classifiers and KD-tree for each tree node.

Math:

$$f(\mathbf{x}, j, Th) = \begin{cases} 1 & if\ \mathbf{x}(j) \geq Th \\ 0 & otherwise \end{cases}$$

$$(j^*, Th^*) = \arg\min_{j,Th} e_{training}(j, Th) \qquad = \arg\min_{j,Th} \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f(\mathbf{x}_i, j, Th))$$

Implementation:



```
for j=1 to m
    for $Th = min\_val\_j$ to $max\_val\_j$
        Compute $e_{training}(j, Th) = min(\frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_>(\mathbf{x}_i, j, Th)), \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}(y_i \neq f_<(\mathbf{x}_i, j, Th)))$
Report the $(j^*, Th^*)$ with the lowest $e_{training}(j, Th)$
```

What is estimation and model learning?

Rule of thumb to learn a good classifier

1. Reduce the training error

2. Simply the classifier to avoid overfitting

Both trying too hard (memorize everything but cannot generalize)

and not trying (learn nothing)

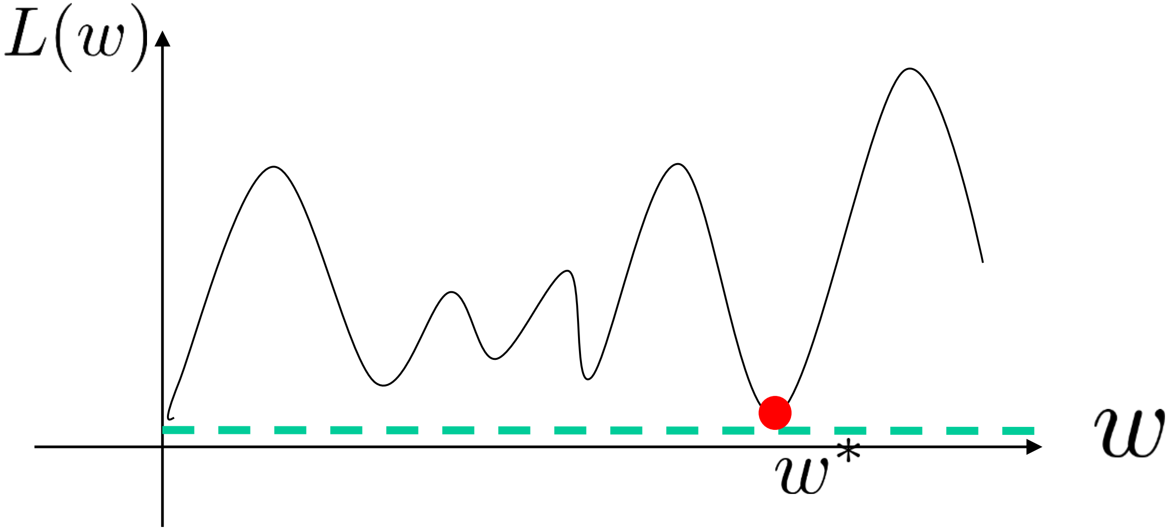
should be avoided.

Optimization: argmin

$$w^* = \arg\min_w L(w)$$

The operator $\arg\min$ defines the optimal value (in the argument of function $L()$) w^* that minimizes $L(w)$

$\arg\min L(w)$ doesn't return the value of $L(w)$

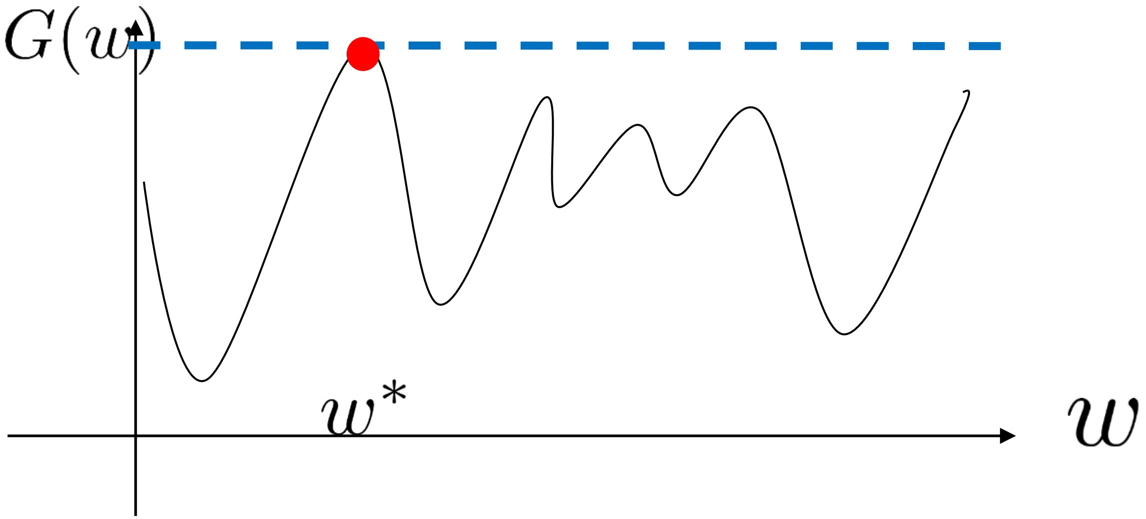


Optimization: argmax

$$w^* = \arg\max_w G(w)$$

The operator $\arg\max$ defines the optimal value (in the argument of function $G()$) w^* that maximizes $G(w)$

$\arg\max G(w)$ doesn't return the value of $G(w)$



Optimization: argmin

$$w^* = \arg\min_w L(w)$$

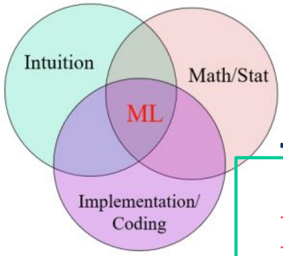
If a function $g(v)$ is monotonic, e.g. $\forall v_1 > v_2$ it is always true that $g(v_1) > g(v_2)$, then:

$$w^* = \arg\min_w L(w) = \arg\min_w g(L(w))$$

For example,

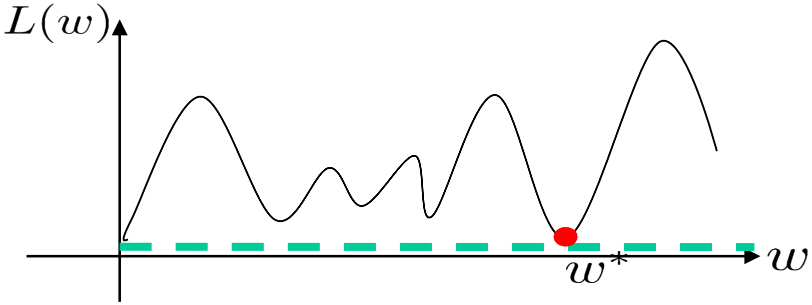
if $g(v) = 2 \times v + 10$

$$w^* = \arg\min_w L(w) = \arg\min_w 2 \times L(w) + 10$$



Recap: Estimation

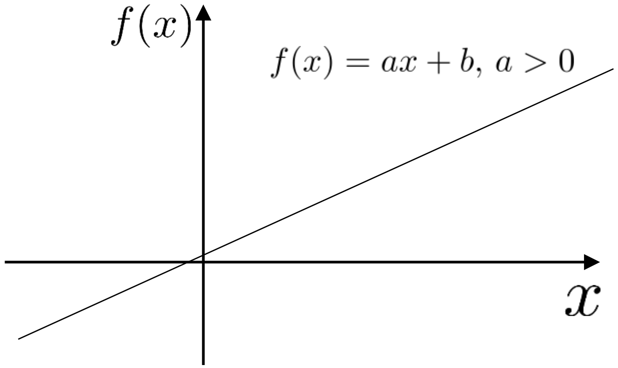
Intuition: Typically, optimization in machine learning refers the process in which an objective function is minimized/maximized. A major task in machine learning is to perform training to attain the optimal parameter that minimizes/maximizes the corresponding objective function. Note that the optimal model parameter is not the minimal/maximal value of the function itself.

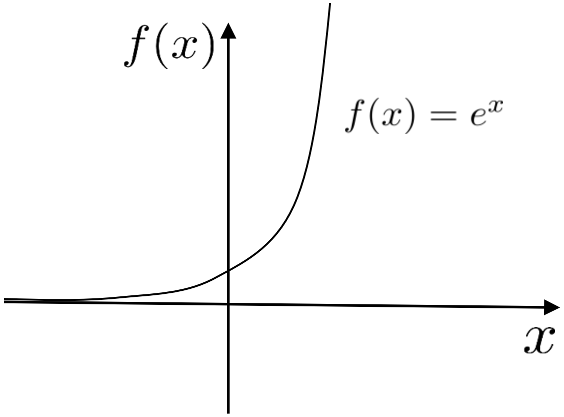


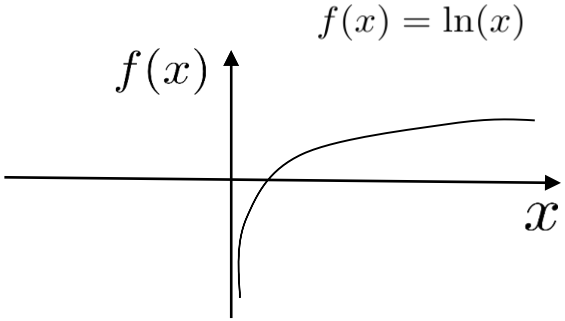
Math:

$$w^* = \arg\min_w L(w)$$

Monotonic functions







For a monotonically increasing function:

$f(x)$

Is $-f(x)$ monotonically decreasing?

A. Yes

B. No

C. It depends

For a monotonically increasing function:

$$f(x) \in R^+$$

Is $\ln f(x)$ monotonically increasing?

A. Yes

B. No

C. It depends

For two monotonically increasing functions:

$f(x)$ and $g(x)$

Is $f(x) + g(x)$ monotonically increasing?

A. Yes

B. No

C. It depends

For two monotonically increasing functions:

$f(x)$ and $g(x)$

Is $f(x) - g(x)$ monotonically increasing?

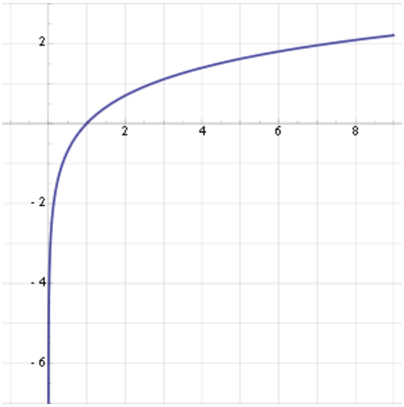
A. Yes

B. No

C. It depends

argmin and argmax

The function $\ln(v)$ is monotonic, e.g. $\forall v_1 > v_2$ it is always true that $\ln(v_1) > \ln(v_2)$, then:

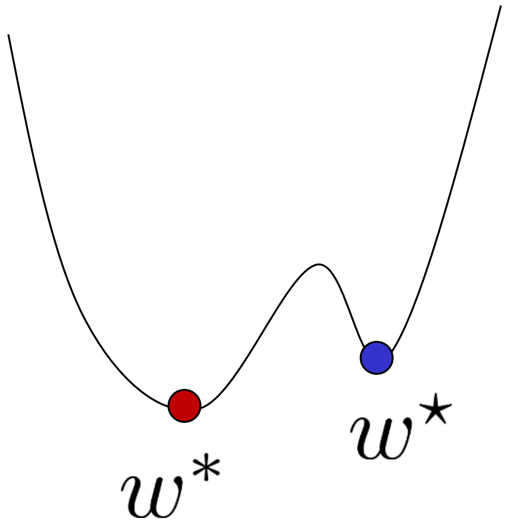


$$w^* = \arg\max_w G(w)$$

$$= \arg\max_w \ln(G(w))$$

$$= \arg\min_w -\ln(G(w))$$

Optimization



Things we ofen need to be able to do to solve optimization problem:

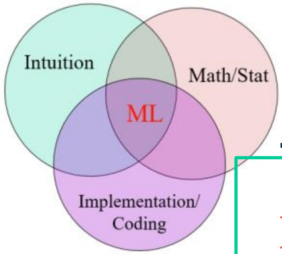
1. $\forall w$, check if $w \in \Omega$?

2. For $\forall w$, computing $L(w)$, $\nabla L(w)$, $\nabla^2 L(w)$.

Definition:

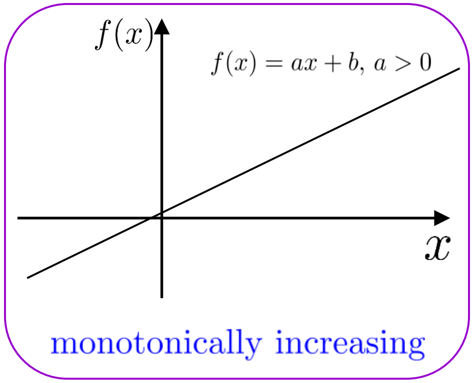
1. w^* is a globally optimal solution for $\theta^* \in \Omega$ and $L(w^*) \leq L(w) \forall w \in \Omega$

2. $w^\star$ is a locally optimal solution if there is a neighborhood $\mathcal{N}$ around w such that $w^\star \in \Omega$, $L(w^\star) \leq L(w)$, $\forall w \in \mathcal{N} \cap \Omega$.

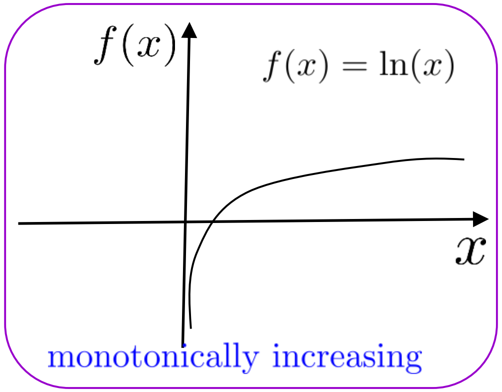


Recap: Monotonicity

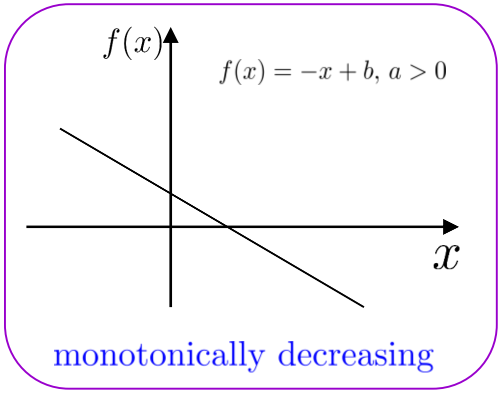
Intuition: In machine learning, we use the monotonicity of functions to help significantly reduce the difficulty/complexity of an estimation/learning problem.



monotonically increasing



monotonically increasing



monotonically decreasing

Math:

$$w^* = \arg\max_w \prod_{i=1}^n p(y_i|x_i; w)$$

$$= \arg\max_w \ln[\prod_{i=1}^n p(y_i|x_i; w)]$$

$$= \arg\min_w - \sum_{i=1}^n \ln[p(y_i|x_i; w)]$$