Introduction to Machine Learning

Lecture 10

Instructor:
Dr. Tom Arodz

Recap

- Modern machine learning:

```
minimizeθ: error_metric(
   S, some_f(θ)(x in S).to_class_probs())
```

- Filling the gaps:
  - `some_f(θ)(x)` or `some_f(θ;x)`
    - Some function that takes two groups of inputs
      - Raw input features x
      - Trainable parameters θ
        - the parameters help us pick a specific function from a family of functions
  - `to_class_probs()`
    - `some_f(θ;x)` may not return probabilities, how to convert to: ≥0, sum=1
  - `error_metric`
    - Classification error won't work. MSE is not ideal. We need something better!
  - `minimizeθ`
    - **Gradient descent, over parameters θ of the model** `some_f(θ;x)`
    - $\boldsymbol{\theta_{n+1} = \theta_n - c\, \nabla_\theta\, \mathrm{error\_metric}(S, \mathrm{model}(\theta_n, S))}$

10.1. Functions (architectures) used in ML: Linear models

Linear models

```
minimizeθ: error_metric(
    S, some_f(θ)(x in S).to_class_probs())
```

- Functions (architectures) used in ML: Linear models
  - "predicted y" = y_{pred} $= w \cdot x + b \quad = \langle w,x \rangle + b$
    $= w^T x + b \quad = \Sigma_i w_i x_i + b$
    $= w_1 x_1 + w_2 x_2 + \ldots + w_F x_F + b$
    Many ways to write the same formula
    sometimes: w_0 instead of b (x_0=1)
    –b instead of +b
    - returns a real value from -∞ to +∞, sign indicates class
  - `some_f(θ;x) = linear(w,b;x) = wTx+b`
    - two groups of inputs:
      - Raw input features x
      - **Trainable parameters `θ = (w,b)`**
        - Features weights **w** and bias **b**

Linear models

- Modern machine learning:

```
minimizeθ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

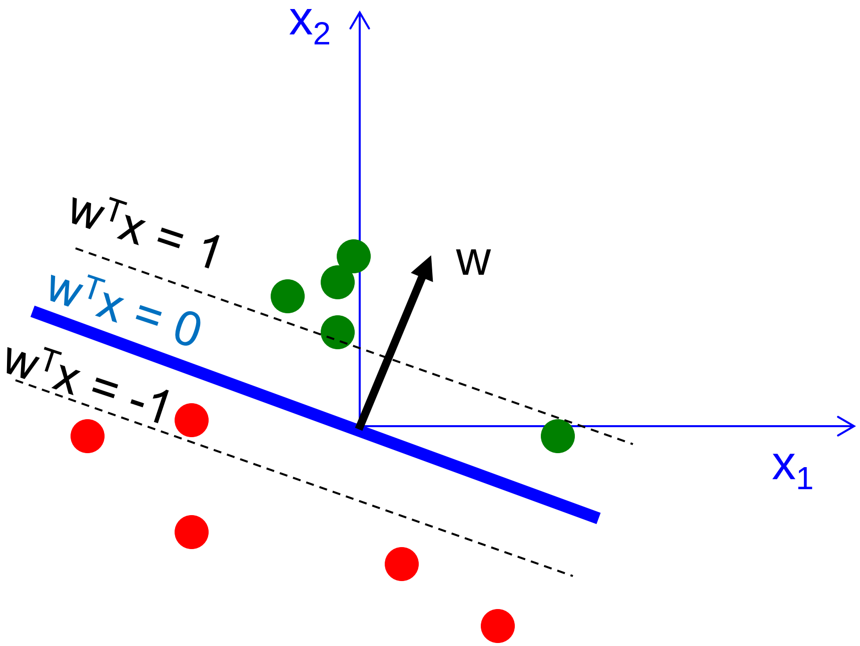
- Functions (architectures) used in ML: Linear models
  - $y_{pred} = w^T x + b$ on input space of F features can be interpreted/viewed in several ways:
    - A linear decision boundary (line/hyperplane) separating 2 classes in F-dim space
    - A projection on a 1-dim space
    - A linear function in F-dim space

Linear Models – decision boundary

- $y = w^T x = w_1 x_1 + w_2 x_2$

  or

- $y = w^T x = w_1 x_1 + w_2 x_2 + b$



w is a vector with the same dimensionality *d* as the samples (e.g. here, 2 features => 2D) We can plot it in the same *d*-dim space (here: 2D) together with samples

it's a vector, we plot it starting from (0,0,…)

Decision boundary is always orthogonal (at 90 degree angle) to vector w.
It is a subspace of dimensionality (d-1) consisting of all x for which $w^T x = 0$
(d=2 → a 1D line; d=3 → a 2D plane, d=4 → a 3D hyperplane…)

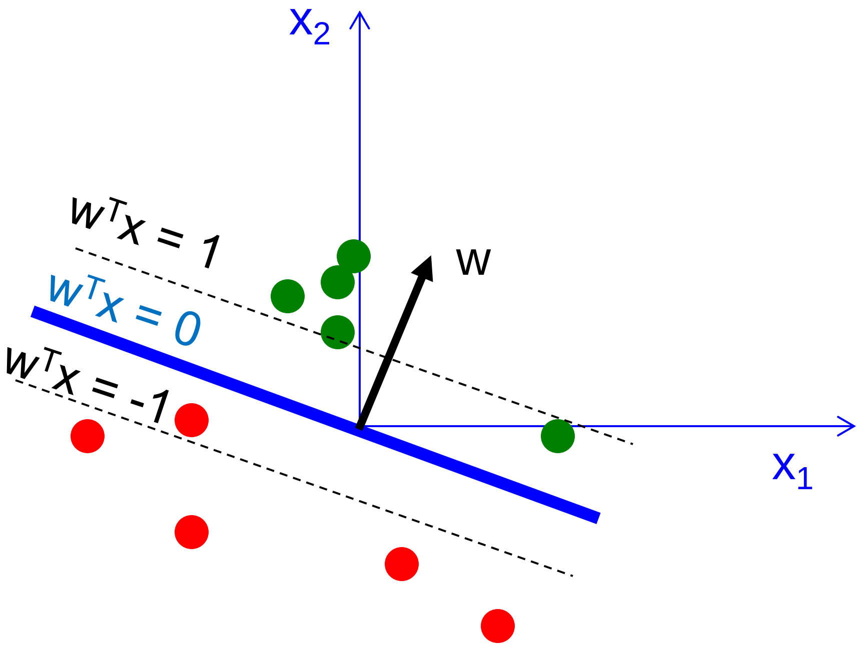
if there is no bias (b=0) then decision boundary has to pass through (0,0,…)

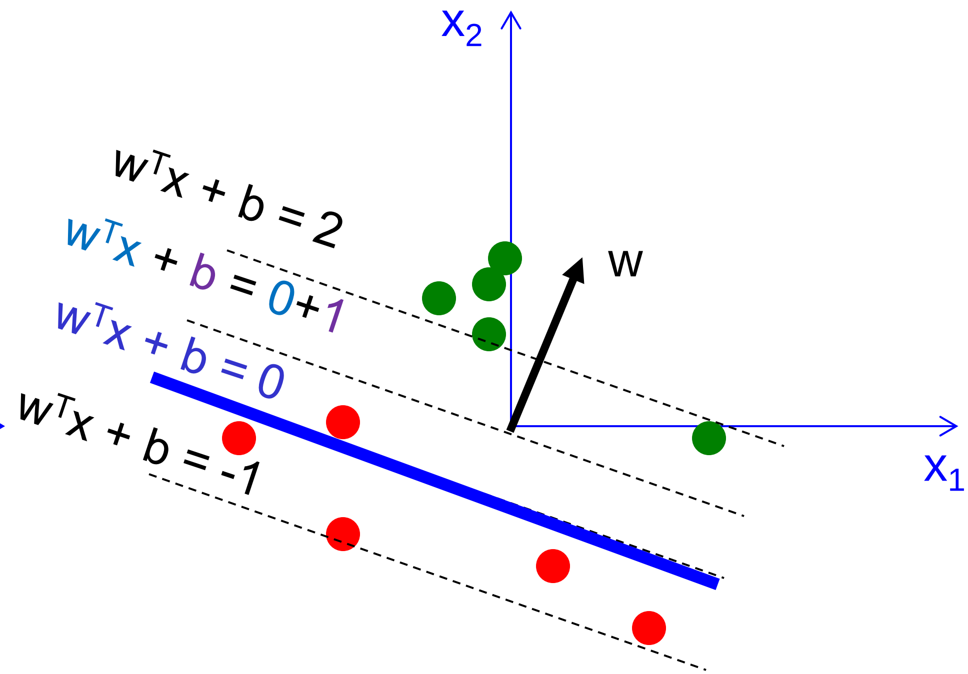
Vector "w" points in the direction in which predictions increase most steeply towards +1 class

Linear Models – decision boundary

- $y=w^Tx=w_1x_1+w_2x_2$
- $y=w^Tx=w_1x_1+w_2x_2+b$

let $b = +1$



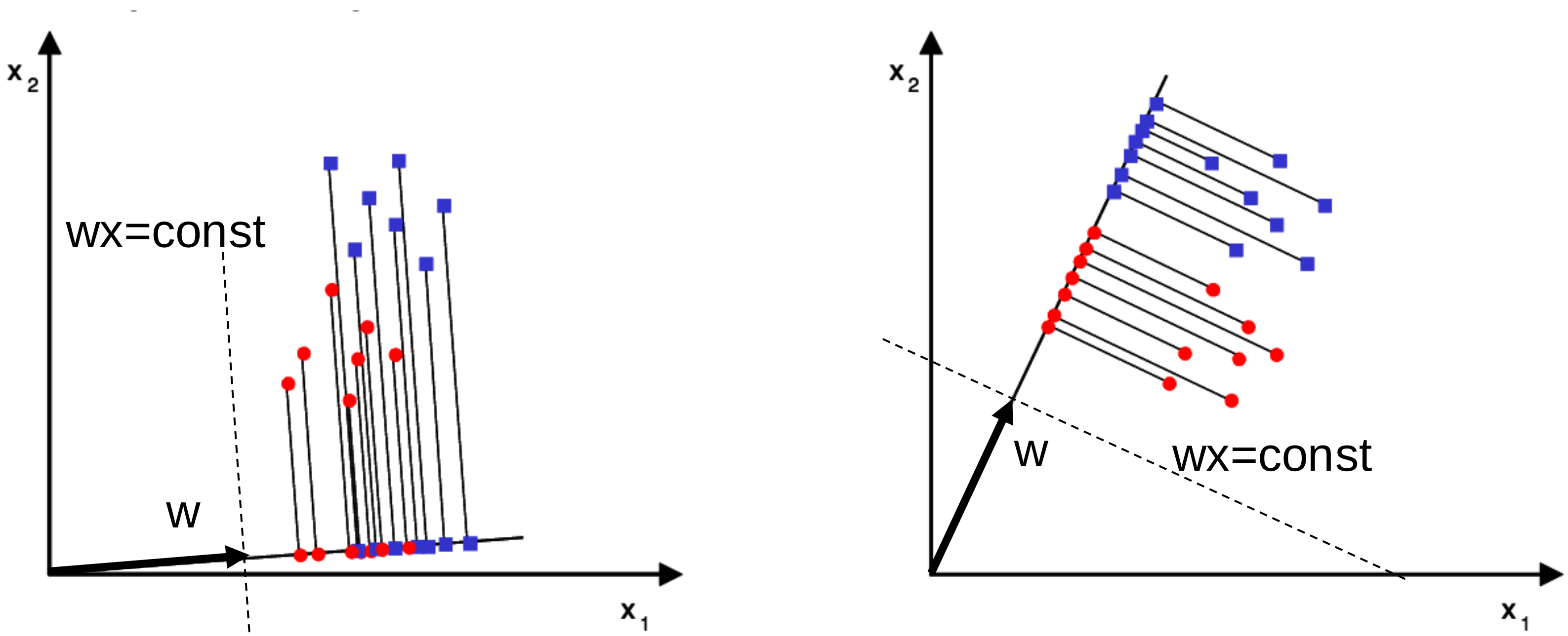


Non-zero bias b shifts the decision boundary along the w vector

positive +b shifts the decision in direction of -w, negative b in direction of +w

With non-zero bias, the decision boundary no longer has to pass through (0,0,…)

Linear models - projection

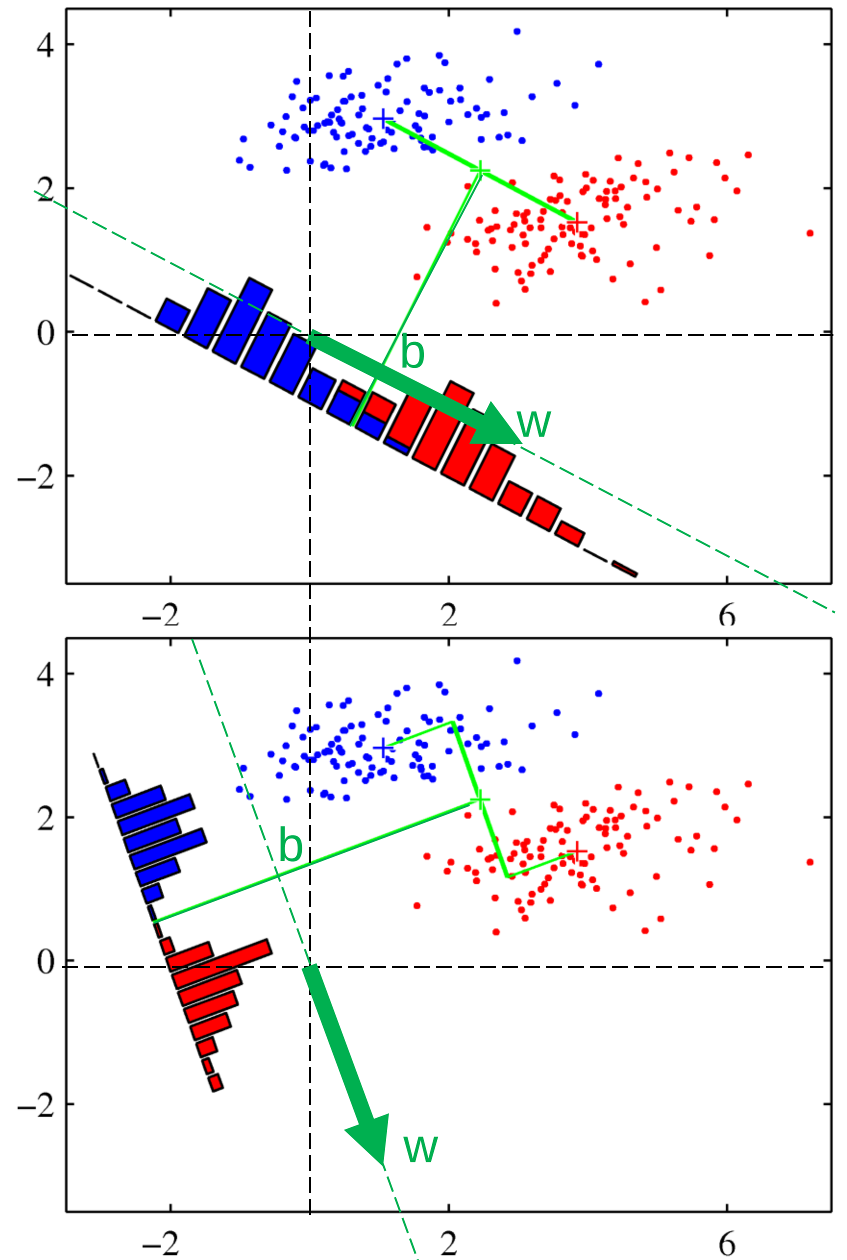


- $y_{pred} = w^T x + b$
  - $w^T x$ - linear projection of samples on a line parallel to w
  - b - decision threshold on that line

Linear models - projection

- $y_{pred} = w^T x + b$
  - $w^T x$ - linear projection of samples on a line parallel to w
  - b - decision threshold on that line

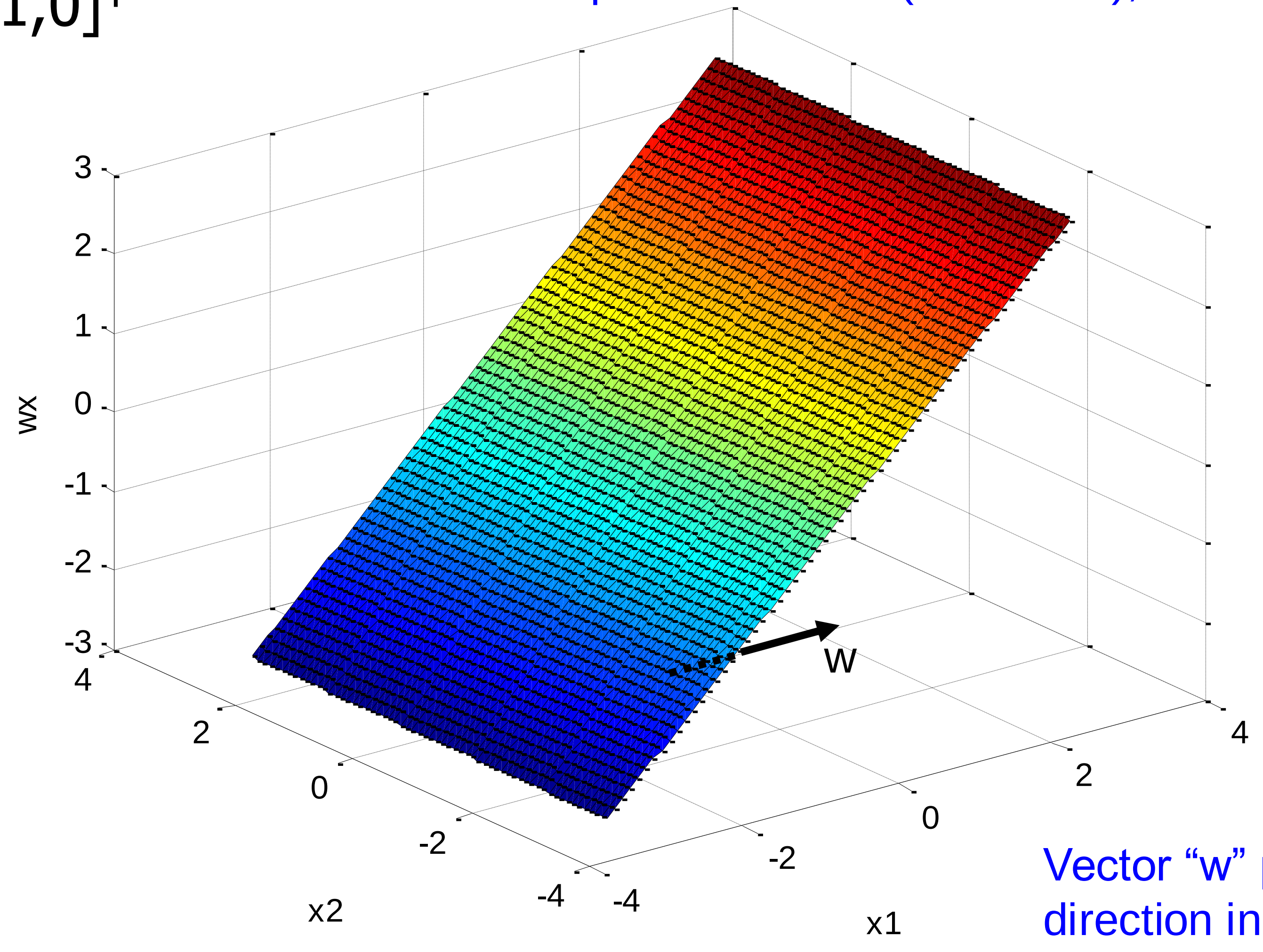
- w is a vector, we always "anchor" it a (0,0,...)



Linear Models – linear function

- $y = w^T x = w_1 x_1 + w_2 x_2$
- $w = [1,0]^T$

We can plot $w^T x$ as a real-function of x
surface of f(x) will be a line if x is 1D (y=ax+b)
or a plane if x is 2D (like below), or a hyperplane

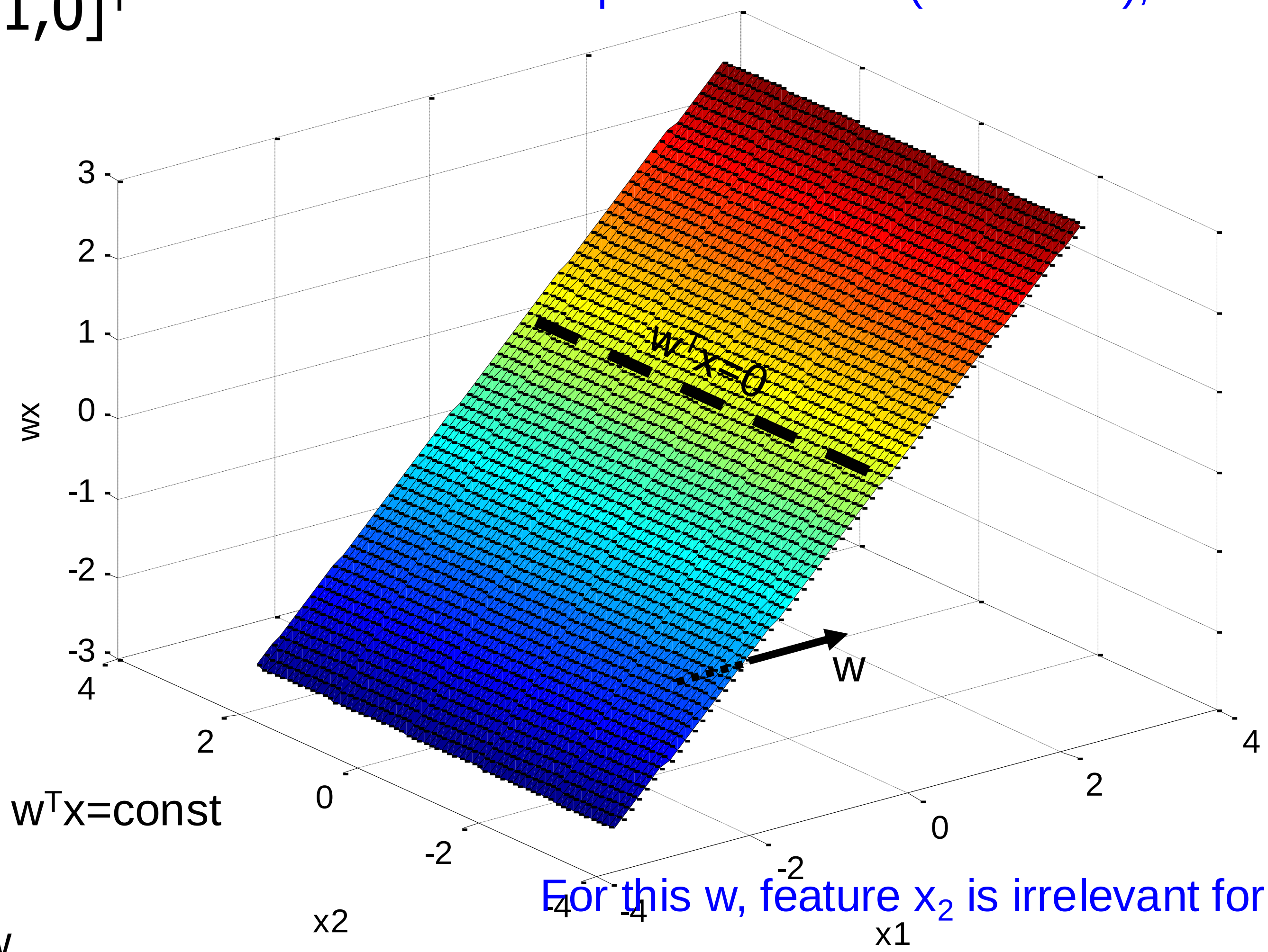


Vector "w" points in the direction in which f(x) increases most steeply

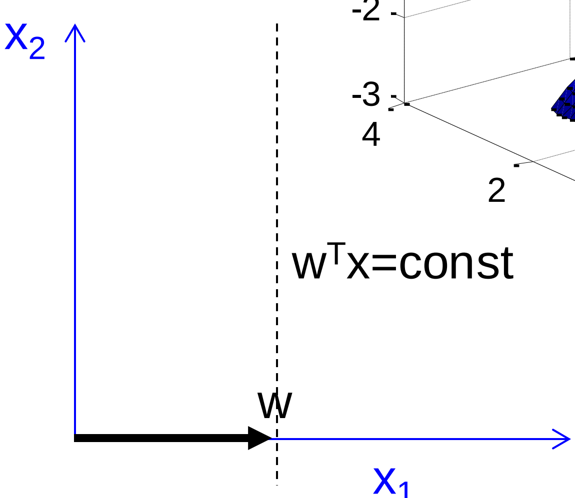
Linear Models – linear function

- $y = w^T x = w_1 x_1 + w_2 x_2$
- $w = [1,0]^T$

We can plot $w^T x$ as a real-function of x
surface of f(x) will be a line if x is 1D (y=ax+b)
or a plane if x is 2D (like below), or a hyperplane



For this w, feature x_2 is irrelevant for predictions

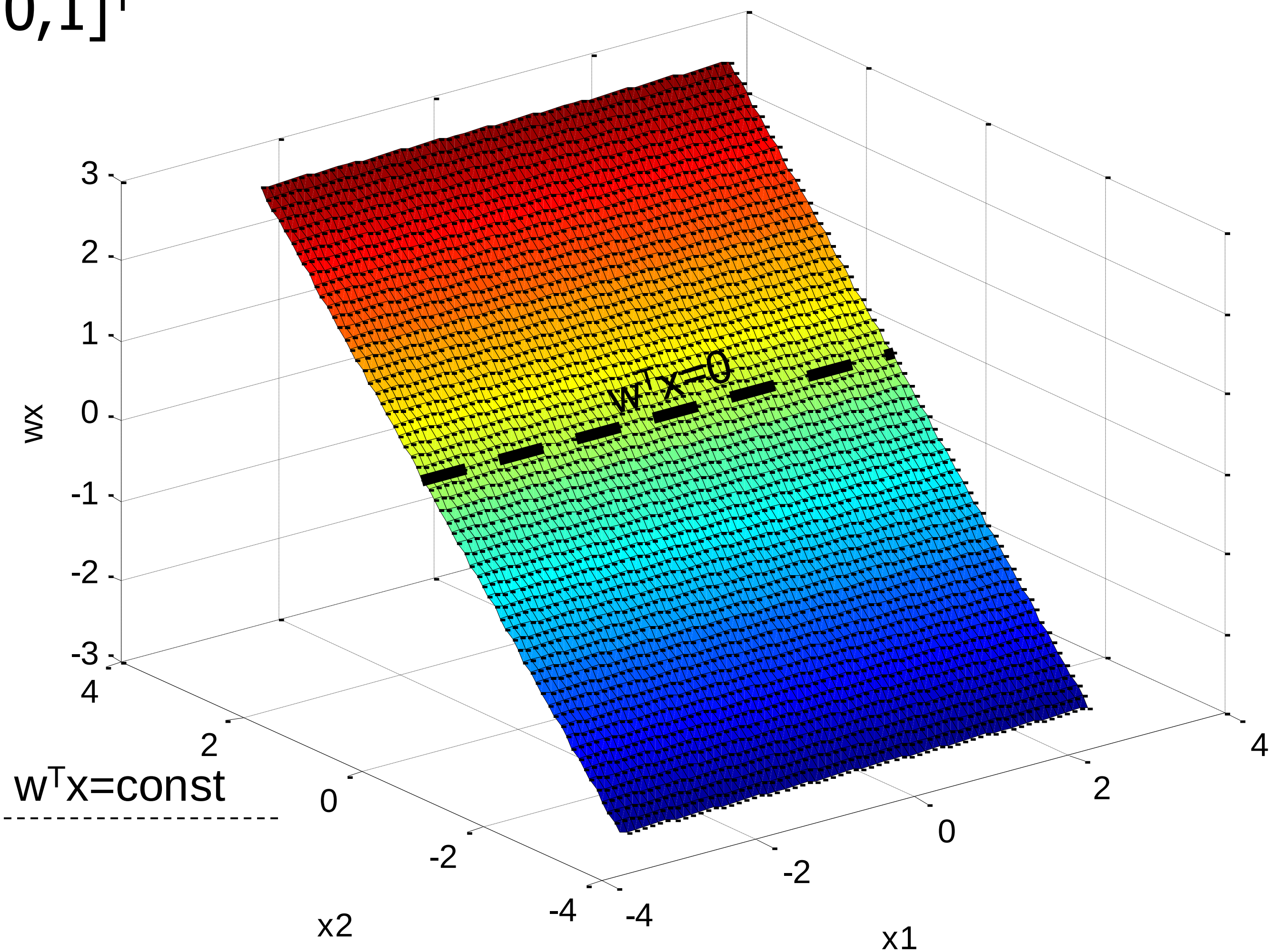


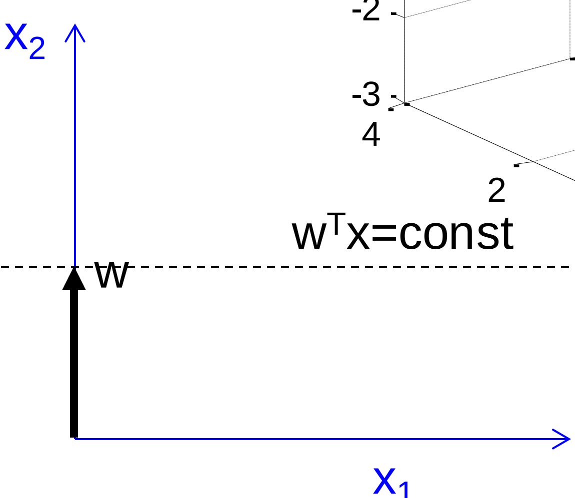
Decision boundary is orthogonal to w

Linear Models – linear function

- $y=w^Tx=w_1x_1+w_2x_2$
- $w=[0,1]^T$

Which features are important here?

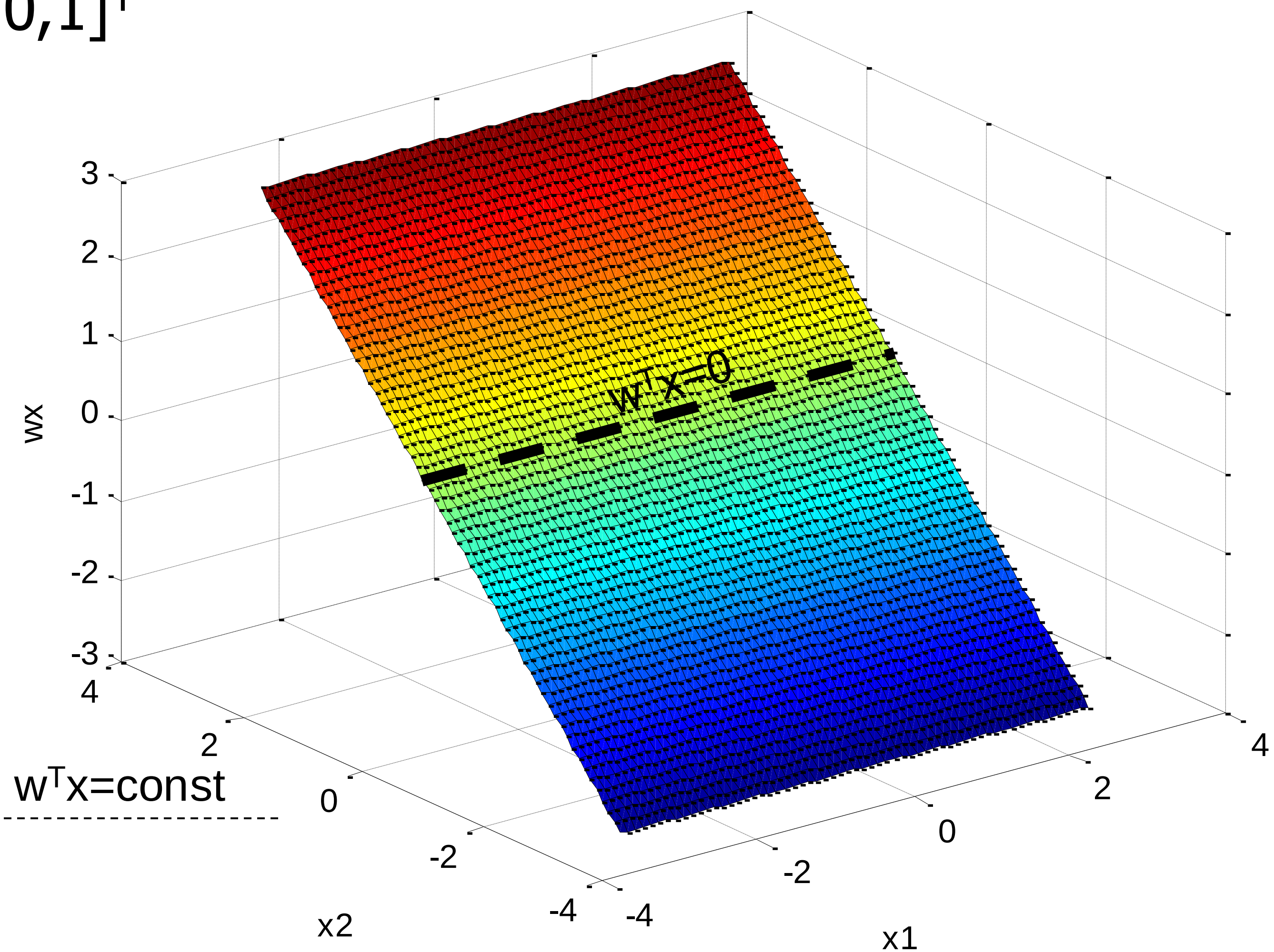


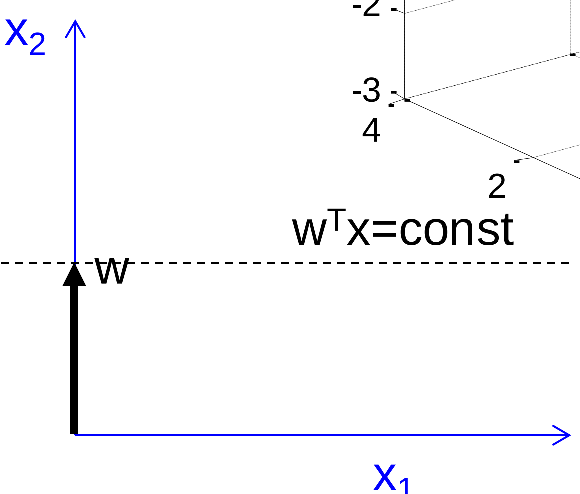


Linear Models – linear function

- $y = w^T x = w_1 x_1 + w_2 x_2$
- $w = [0,1]^T$

For this w, feature x_1 is irrelevant for predictions

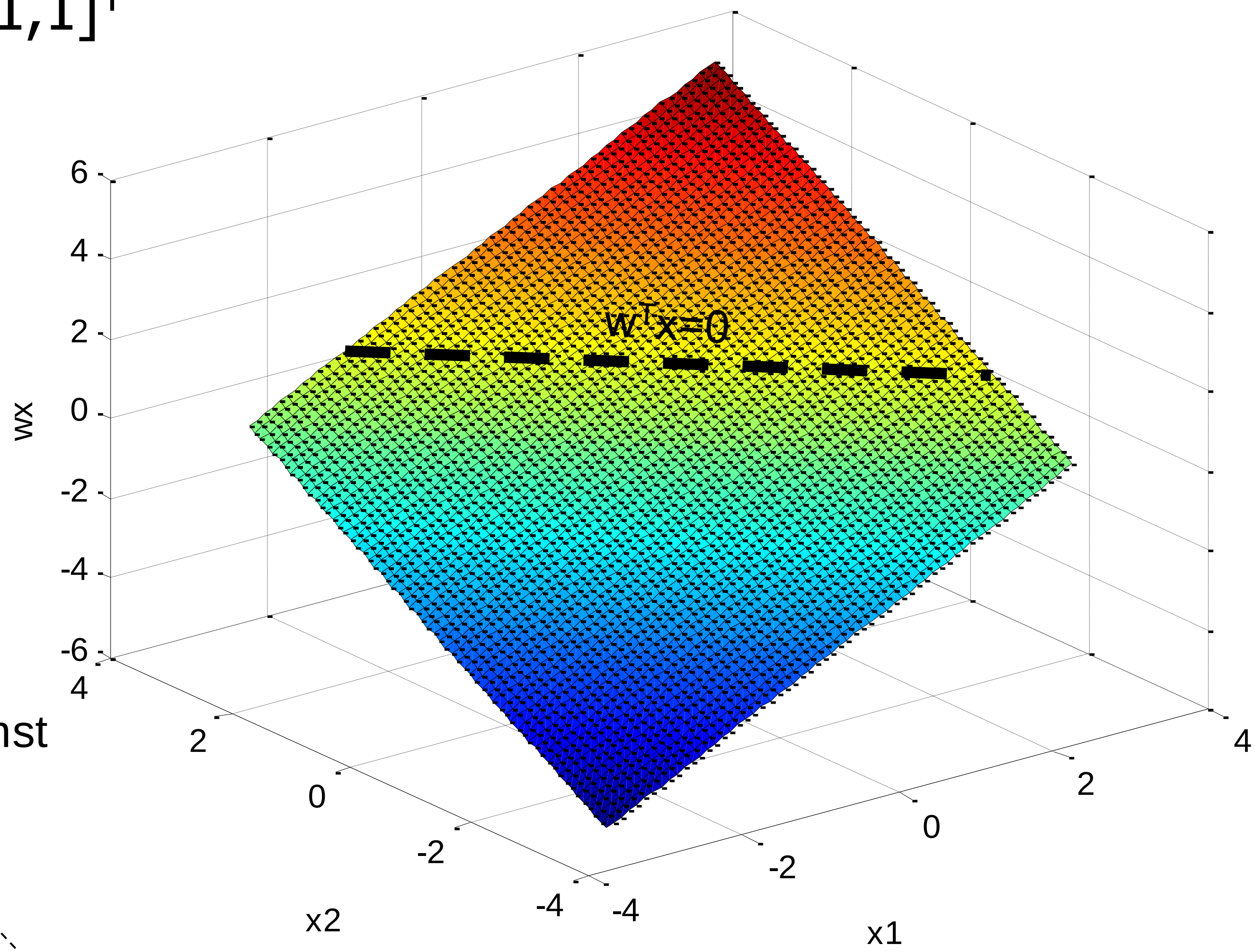


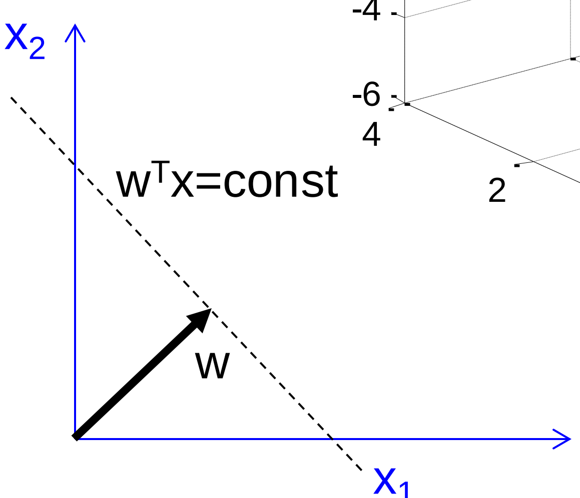


Linear Models – linear function

- $y = w^T x = w_1 x_1 + w_2 x_2$
- $w = [1,1]^T$

For this w, both features are equally relevant for predictions

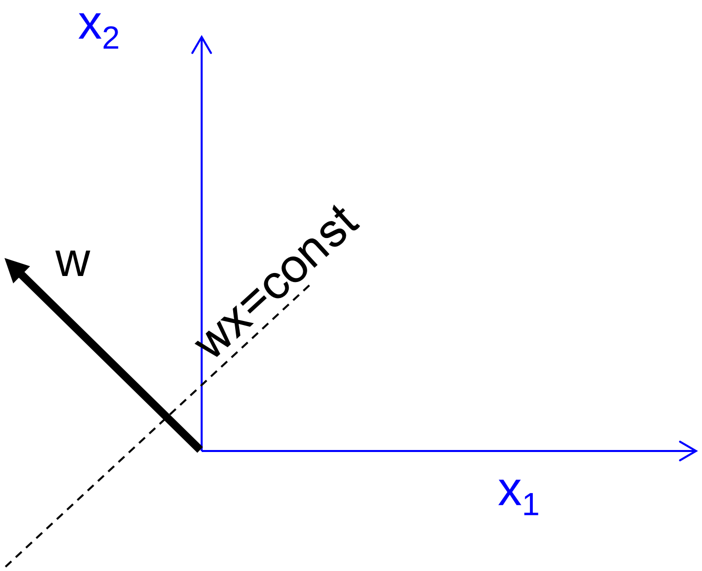




Linear Models – interpreting "w"

- $y=w^Tx=w_1x_1+w_2x_2$
- $w=[-1,1]^T$

For this w, both features are equally relevant for predictions
but x_1 contributes negatively
(low values of x_1 => class +1)
while x_2 contributes positively
(high values of x_2 => class +1



We sometimes interpret values w_i as "importance" of features x_i

But:

- May be misleading if features are correlated
  e.g. extreme case:
  if x2=x1, then for $w=[-1,1]^T$, $w^Tx=0$
  i.e., together, they don't contribute anything to predictions, even though
  both features "look" important!

Space of classification models

- We have observations in a fixed F-dimensional feature space $\mathcal{X} \subset \mathbb{R}^F$
  - Every sample x is a vector (point) in that feature space

    $$\mathbf{x} \in \mathcal{X} \qquad \mathbf{x} = \left[x^{\langle 1 \rangle}, x^{\langle 2 \rangle}, ..., x^{\langle F \rangle}\right]^T$$

- We want to construct a classifier, some function h(x) e.g. a linear model:

  $$h(x) = w^T x + b$$

- Overall, a classifier is a function that returns real values:
  - If h(x)>0, we predict 1,
  - If h(x)<0, we predict -1
  - All possible classifiers

    $$h : \mathcal{X} \mapsto \mathbb{R}.$$

    of a certain type (e.g. all linear classifiers) form a space $\mathcal{H}$
    - For linear models, it's an F+1 dimensional space of all possible weights **w** (F-dim) and all possible biases b (1-dim)

Recap

- Modern machine learning:

```
minimizeθ: error_metric(
  S, some_f(θ)(x in S).to_class_probs())
```

  - Filling the gaps:
    - `some_f(θ)(x)` or `some_f(θ;x)`
      - What family of functions? Simple choice that often works:
        `some_f(θ;x) = linear(w,b;x) = wᵀx+b`
        Linear models - simple and often very effective
    - `to_class_probs()`
      - `some_f(θ;x)` may not return probabilities, how to convert to: ≥0, sum=1
    - `error_metric`
      - Classification error won't work. MSE is not ideal. We need something better
    - `minimizeθ`
      - **How?**
        Gradient descent, over parameters θ **of** `some_f(θ;x)`

10.2. Gradient descent for linear models

Building ML Approach #3

- Modern machine learning:

```
minimize_θ: error_metric(
   S, some_f(θ)(x in S).to_class_probs())
```

  - Filling the gaps:
    - `some_f(θ)(x)`   or `some_f(θ;x)`
      - **`some_f(θ;x) = linear(w,b;x) =`** $w^T x + b$
        Linear models - simple and often very effective:
    - `minimize_θ`
      - how? **Gradient descent!**

- **Let's try gradient descent for linear models**

Gradient of a linear model

- $y_{pred} = f(w,b;x) = w \cdot x + b = w_1x_1+w_2x_2 + \ldots +w_Fx_F + b$

- **<u>Let's start with simple scenario:</u>**
  - one sample: x
  - let's assume the class is supposed to be $y_{true} = -1$
  - that means, we want to make $y_{pred}=f(w;x)$ negative, i.e., minimize it
- The task in this scenario is:
  - $\texttt{minimize}_w\texttt{: f(w,b;x)}$
- Minimization via gradient descent:
  - Move w in the direction of negated gradient of f:
    - $w_{new} = w_{old} - c\, \nabla f(w,b;x)$

Gradient of a linear model

- $y_{pred} = f(w,b;x) = w \cdot x + b = w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b$

- $\nabla f = \nabla_{(w,b)} f(w,b;x)$
  $= [\partial f(w;x) / \partial w_1, \partial f(w;x) / \partial w_2, \ldots, \partial f(w;x) / \partial w_F, \partial f(w;x) / \partial b\ ]$

- $\partial f(w,b;x) / \partial w_1 = \partial(w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b) / \partial w_1$
  $= \partial w_1x_1 / \partial w_1 + \ldots + \partial w_Fx_F / \partial w_1 + \partial b / \partial w_1$
  $= x_1\, \partial w_1 / \partial w_1 + \ldots + 0 + 0 = x_1$

- Rule of differentiation recap:
  - Sum:
    $\partial(g(a)+h(a)) / \partial a = \partial g(a)/\partial a + \partial h(a)/\partial a$

Gradient of a linear model

- $y_{pred} = f(w,b;x) = w \cdot x + b = w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b$

- $\nabla f(w,b;x)$
  $= [\partial f(w;x)/\partial w_1, \partial f(w;x)/\partial w_2, \ldots, \partial f(w;x)/\partial w_F, \partial f(w;x)/\partial b]$
- $\partial f(w,b;x)/\partial w_1 = x_1$

- $\partial f(w,b;x)/\partial b$
  $= \partial(w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b)/\partial b$
  $= \partial w_1x_1/\partial b + \ldots + \partial w_Fx_F/\partial b + \partial b/\partial b$
  $= 0 + \ldots + 0 + 1 = 1$

- $\nabla_{w,b} f(w,b;x)$
  $= [\partial f(w;x)/\partial w_1, \partial f(w;x)/\partial w_2, \ldots, \partial f(w;x)/\partial w_F, \partial f(w;x)/\partial b]$
  $= [x_1, x_2, \ldots, x_F, 1]$

Gradient of a linear model

- $y_{pred} = f(w;x) = w \cdot x + b = w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b$

- **Another simple scenario:**
  - one sample: x, let's assume the class is supposed to be $y_{true} = +1$
  - that means, we want to make $y_{pred}=f(w;x)$ positive, i.e., maximize it
    - i.e., minimize: *negate*(f(w;x))= -1 * f(w;x)
- The task in this scenario is:
  - $\texttt{minimize}_w\texttt{: negate(f(w;x))}$
- Minimization via gradient descent:
  - move w in the direction of negated gradient of f:
    - $w_{new} = w_{old} - c\, \nabla negate(f(w;x))$

Gradient of a linear model

- $y_{pred} = f(w,b;x) = w \cdot x + b = w_1x_1 + w_2x_2 + \ldots + w_Fx_F + b$

- $\nabla negate(f(w,b;x))$
  $= [\partial negate(f(w;x)) / \partial w_1, \ldots, \partial negate(f(w;x)) / \partial b]$

- $\partial negate(f(w,b;x)) / \partial w_1 = \partial negate(y_{pred}) / \partial y_{pred} * \partial y_{pred} / \partial w_1$
  $= \partial(-1 * y_{pred}) / \partial y_{pred} * \partial f(w,b;x) / \partial w_1$
  $= \partial(-1 * f(w,b;x)) / \partial f(w,b;x) * \partial f(w,b;x) / \partial w_1$
  $= -1 * \partial f(w,b;x) / \partial w_1 = -1 * x_1 = -x_1$
- $\nabla_{w,b} negate(f(w,b;x)) = [-x_1, -x_2, \ldots, -x_F, -1]$

- Rule of differentiation recap:
  - Chain rule:
    $\partial f(g(a)) / \partial a = \partial f / \partial g * \partial g / \partial a$
    $= \partial f(z) / \partial z * \partial g(a) / \partial a$ where $z = g(a)$

Gradient of a linear model

- **Two-sample (A & B) simple scenario:**
  - sample x_A, true $y_A = +1$, and sample x_B, true $y_B = -1$
  - we want: $y_A = f(w;x_A)$ more positive and $y_B = f(w;x_B)$ more negative
    - `minimize: negate(f(w,b;x_A)) + f(w,b;x_B)`
  - we calculated already:
    - $\nabla_{w,b}\text{negate}(f(w,b;x_A)) = \nabla_{w,b} -f(w,b;x_A) = [-x_{A,1}, -x_{A,2}, \ldots, -x_{A,F}, -1]$
    - $\nabla_{w,b} f(w,b;x_B) = \nabla_{w,b} f(w,b;x_B) = [+x_{B,1}, x_{B,2}, \ldots, x_{B,F}, +1]$
  - We can write it jointly as:
    - `minimize:` $-f(w,b;x_A) \quad +f(w,b;x_B)$
    - `minimize:` $-y_A f(w,b;x_A) \quad -y_B f(w,b;x_B)$
    - `minimize:` $\Sigma_{k=\{A,B\}} -y_k f(w,b;x_k)$
  - To move w,b in the proper direction, it makes sense to minimize: $\nabla_{w,b} \Sigma_{k=\{A,B\}} -y_k f(w;x_k)$

$$= \Sigma_{k=\{A,B\}} [-y_k x_{k,1}, -y_k x_{k,2}, \ldots, , -y_k x_{k,F}, -y_k]$$

the two-part term $y_k f(w;x_k) = y_{true} * y_{pred}$ is very common

10.3. From *error metric* to loss function

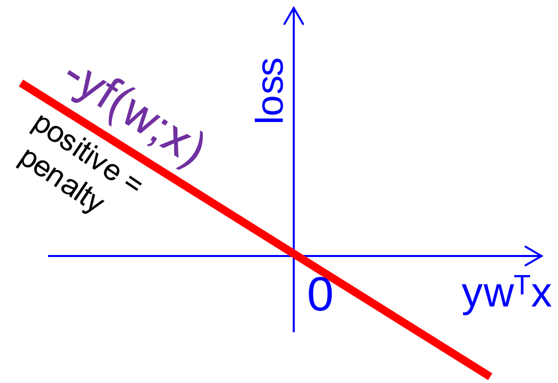
Better error metric?

- Problems with error metrics seen so far:
  - Classification error / is-incorrect:
    natural and intuitive, but flat (∇=0), not continuous
  - MSE is not ideal for classification problems

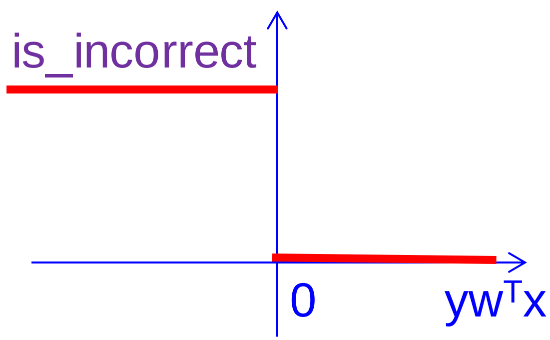
- Recall this from "gradient descent":
  - Two-sample (A & B) simple scenario:
    - sample x_A, true y_A = +1, and sample x_B, true y_B = -1
    - we want: y_A =f(w;x_A) more positive and y_B=f(w;x_B) more negative
      - `minimize: negate(f(w,b;`x_A`)) + f(w,b;`x_B`)`
    - We can write it jointly as:
      - `minimize:` $-\mathbf{y_A}$`f(w,b;`x_A`)` $-\mathbf{y_B}$`f(w,b;`x_B`)`
      - `minimize:` $\boldsymbol{\Sigma}_{k=\{A,B\}}$ $-\mathbf{y_k}$`f(w,b;`x_k`)`
- Can we use $-\mathbf{y_k}$`f(w,b;`x_k`)`
  as a per-sample error metric to be minimized?

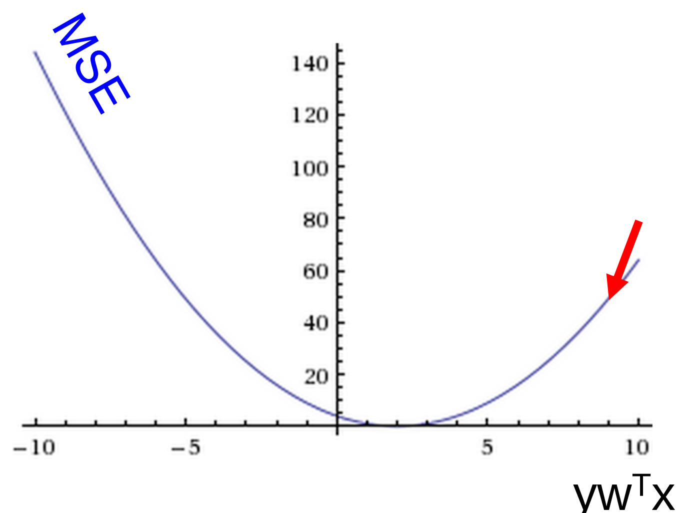
Better error metric?

- Can we use $-y_k f(w, b; x_k)$
  as a per-sample error metric to be minimized?
  - It's not flat
  - It's continuous and differentiable
  - It's monotonic (decreasing)



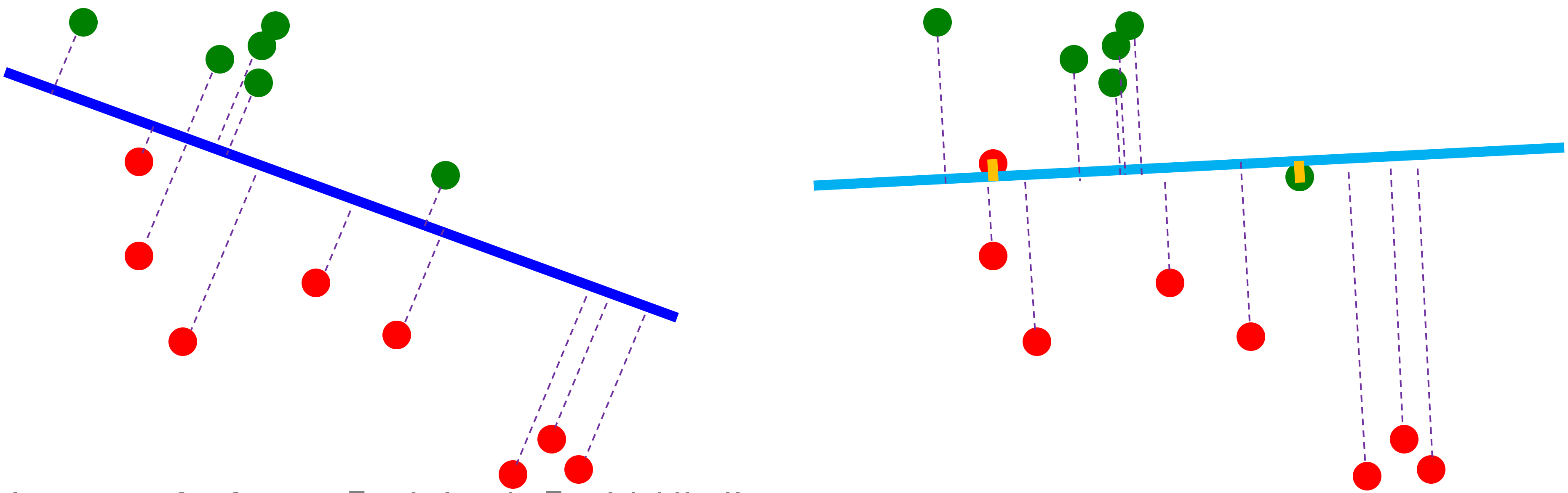
- It avoids problems we had with error_rate and MSE





Better error metric?

- Is $-y_k f(w,b;x_k)$ a good error metric?
  - For linear model $f=w^Tx+b$, value $-yf(x)=-y(w^Tx+b)$ is proportional to distance of x from the decision boundary
  - Positive for incorrect (a penalty)
  - Negative for correct (a reward)

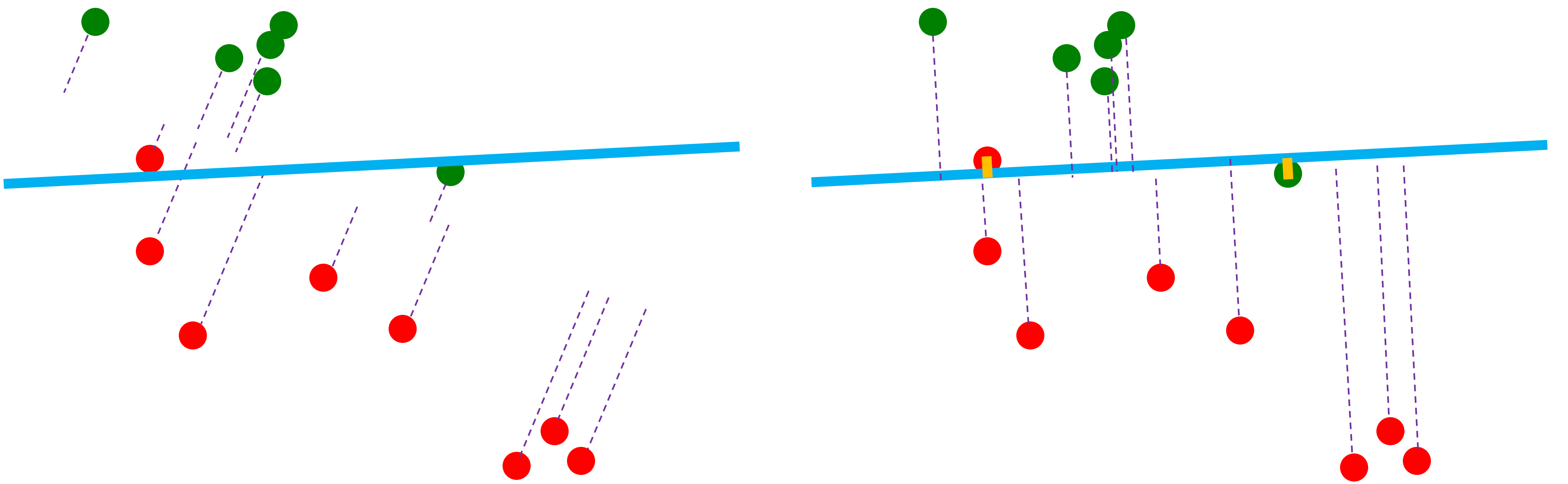


Distance of x from w^Tx+b is: $|w^Tx+b| / ||w||$
where $||w|| = w^Tw$ is norm (length) of w

Which blue decision boundary has **smaller** (penalty – reward)? i.e., **larger** (reward – penalty)?

Better error metric?

- Is $-\mathbf{y_k} f(w,b;x_k)$ a good error metric? No!
  - Positive for incorrect (a penalty)
  - Negative for correct (a reward)



Which blue decision boundary has **smaller** (penalty – reward)?
i.e., **larger** (reward – penalty)?

Better error metric?

- Can we use $-\mathbf{y}_k \mathtt{f(w,b;x}_k\mathtt{)}$
  as an per-sample error metric to be minimized?

- -yf(w;x) is not a good error metric:
  - can lead to errors even if 0-error decision boundary exists!
    - left: loss=2*(-1)+(-1)=-3 vs right: loss=2*(-4) +2=-6

- Giving "reward" (negative value under minimization) is not a good idea!
  - Conclusion: error metric should not have any negative values

Error metric => Loss function

- A classifier is a function h(x) that returns real values:
    - If h(x)>0, we predict 1,
    - If h(x)<0, we predict -1
  - All possible classifiers of a certain type form a space $\mathcal{H}$
- We design a measure of quality of various *h* in $\mathcal{H}$ on a single sample z=(x,y)
  - Non-negative (no rewards), ideally without flat regions esp. in the "wrong prediction" region
  - We call it a penalty or, most commonly, **a loss**
  - **Loss function** $\ell : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}_+$:
    - Input: a classifier *h*, and a labeled sample *z=(x,y)*
    - Output: a **non-negative** real number
      - 0: no loss – we're happy with the prediction
      - >0: some loss – quantifies how unhappy we are

Empirical risk minimization

- Loss function $\ell : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}_+$ takes a single sample
- Our classifier should be good (low loss) for all samples
  - In general, our samples come from distribution D over space of samples (features , class): $\mathcal{Z} = \mathcal{X} \times \{-1, +1\}$
  - The expected (avg. over infinite # of samples) loss for distribution D is called *risk* $R(h, D) = \mathbb{E}_{\mathbf{z} \sim D}[\ell(h, \mathbf{z})]$
  - We want low risk, but: distrib. D unknown => risk unknown
  - We have the training set!
  - We can calculate the average loss on the training set
    - Empirical risk: $\widehat{R}_{S_m}(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, \mathbf{z}_i)$
- How should we choose classifier h from $\mathcal{H}$?
- Empirical risk minimization: $\arg\min_{h \in \mathcal{H}} \widehat{R}_{S_m}(h)$

Losses so far

All have some problems:

- 0/1 loss (is_incorrect): flat
- -yf(x): rewards can lead to errors
- MSE for classification: penalty for correct predictions