DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 3 | Part 1

Least Squares Regression ($d > 1$)

Last Time

- Introduced **linear models**:

$$\begin{aligned} H(\vec{x}) &= w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d \\ &= \text{Aug}(\vec{x}) \cdot \vec{w} \end{aligned}$$

Last Time

- Introduced **empirical risk minimization (ERM)**:
- Step 1: choose a **hypothesis class**
  - Let's assume we've chosen linear predictors
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Last Time

- Chose the **square loss**.
- Minimized the expected square loss for models with *one* feature (predictor variable):

$$H(x) = w_0 + w_1 x$$

- The **least squares solutions**:

$$w_1 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n} (x_i - \bar{x})^2} \qquad w_0 = \bar{y} - w_1 \bar{x}$$

Today

- Least squares solutions for > 1 feature.
- What about classification?

Setting

- A training set consisting of n pairs, $(\vec{x}^{(i)}, y_i)$:
  - $\vec{x}^{(i)} \in \mathbb{R}^d$ is the ith **feature vector**;
  - $y_i \in \mathbb{R}$ is the ith **target**

Our Goal

- Out of all **linear** functions $\mathbb{R}^d \to \mathbb{R}$, find the function H^* with the smallest **mean squared error** w.r.t. the training data.

- That is, find linear H minimizing:

$$R_{\text{sq}}(H) = \frac{1}{n} \sum_{i=1}^{n} \left(H(\vec{x}^{(i)}) - y_i \right)^2$$

- This problem is called **least squares regression**.

Recall

- A linear function $H(\vec{x}; \vec{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d$ is **completely determined** by its parameter vector, $\vec{w}$:

$$H(\vec{x}; \vec{w}) = \text{Aug}(\vec{x}) \cdot \vec{w}$$

- **Updated goal:** find $\vec{w} \in \mathbb{R}^{d+1}$ minimizing:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i \right)^2$$

Two Approaches

- **Approach #1:** Vector Calculus
- **Approach #2:** Orthogonal Projection (Geometry)
- Different ways of arriving at same solutions.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 3 | Part 2

Approach #1: Vector Calculus

Approach #1: Vector Calculus

- **Goal**: minimize $R_{\text{sq}}(\vec{w})$ with respect to $\vec{w}$.
- **Calculus Approach**: Find **gradient** of $R_{\text{sq}}(\vec{w})$; set to zero; solve for $\vec{w}$.
- Essentially what was done last time.
- Now: there are many more than 2 parameters.
- Rely on results from vector calculus.

First Step: Rewrite Risk

- Step one: rewrite R_{sq} in vector form.
- We will find:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}\left(\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w} - y_i\right)^2$$
$$= \frac{1}{n}\|X\vec{w} - \vec{y}\|^2$$

Recall

- If $\vec{u} = (u_1, u_2, \ldots, u_k)^T$, then:

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u} = \sum_{i=1}^{k} u_i^2$$

- So, if $\vec{a} = (a_1, \ldots, a_k)^T$ and $\vec{b} = (b_1, \ldots, b_k)^T$:

$$\begin{aligned} \|\vec{a} - \vec{b}\|^2 &= (\vec{a} - \vec{b}) \cdot (\vec{a} - \vec{b}) \\ &= \sum_{i=1}^{k} (a_i - b_i)^2 \end{aligned}$$

First Step: Rewrite Risk

- Define $p_i = \text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}$, and let $\vec{p} = (p_1, \ldots, p_n)^T$.
  - $\vec{p}$ is a vector of the predictions on training set.
  - Note: $\vec{p} \in \mathbb{R}^n$, not $\mathbb{R}^d$!

$w_0 + w_1 x_1^{(i)} + w_2 x_2^{(i)} + \ldots$

$\vec{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$

$\vec{p} = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_n \end{pmatrix}$

- Then:

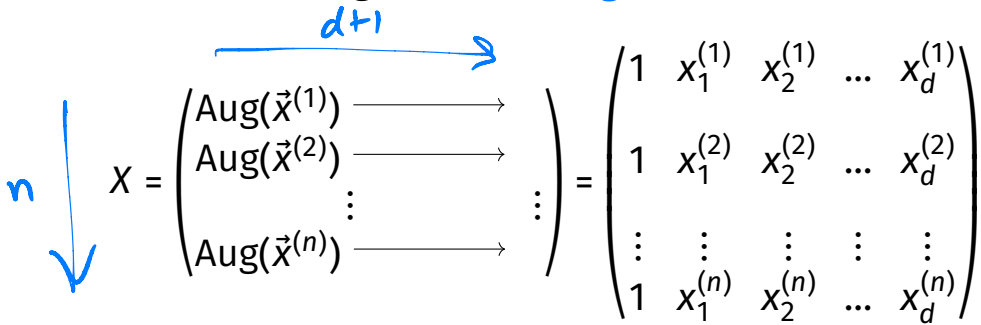
$$R_{\text{sq}}(\vec{w}) = \frac{1}{n} \sum_{i=1}^{n} \left(\underbrace{\text{Aug}(\vec{x}^{(i)}) \cdot \vec{w}}_{p_i} - y_i \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(p_i - y_i \right)^2$$

$$= \frac{1}{n} \| \vec{p} - \vec{y} \|^2$$

First Step: Rewrite Risk

- Define the (augmented) **design matrix**, X:



First Step: Rewrite Risk

- **Observe:** $\vec{p} = X\vec{w}$.

$$\underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) & \longrightarrow \\ \text{Aug}(\vec{x}^{(2)}) & \longrightarrow \\ \vdots & \vdots \\ \text{Aug}(\vec{x}^{(n)}) & \longrightarrow \end{pmatrix}}_{X} \underbrace{\begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix}}_{\vec{w}} = \underbrace{\begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \cdot \vec{w} \\ \text{Aug}(\vec{x}^{(2)}) \cdot \vec{w} \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \cdot \vec{w} \end{pmatrix}}_{\vec{p}}$$

First Step: Rewrite Risk

- Therefore, the MSE can be written:

$$\begin{aligned} R_{\text{sq}}(\vec{w}) &= \frac{1}{n}\sum_{i=1}^{n}\left(\text{Aug}(\vec{x}^{(i)})\cdot\vec{w} - y_i\right)^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}\left(p_i - y_i\right)^2 \\ &= \frac{1}{n}\|\vec{p} - \vec{y}\|^2 \\ &= \frac{1}{n}\|X\vec{w} - \vec{y}\|^2 \end{aligned}$$

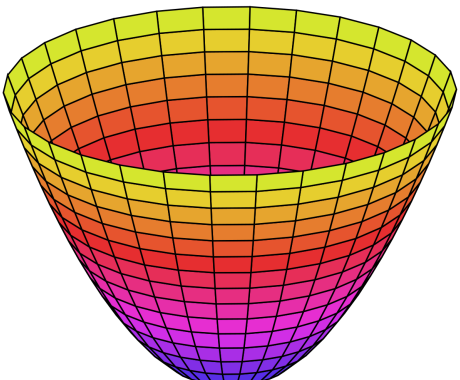
Goal

- Find $\vec{w} \in \mathbb{R}^{d+1}$ minimizing:

$$R_{\text{sq}}(\vec{w}) = \frac{1}{n}\|X\vec{w} - \vec{y}\|^2$$

- **Step Two:** find gradient, set to zero, solve.

Recall: the Risk Surface



Step Two: Find Gradient

- We want to compute:

$$\frac{d}{d\vec{w}}\left[R_{\text{sq}}(\vec{w})\right] = \frac{d}{d\vec{w}}\left[\frac{1}{n}\|X\vec{w} - \vec{y}\|^2\right]$$

- $\frac{dR_{\text{sq}}}{d\vec{w}}$ is the **gradient** of R_{sq}.

- It is the vector of partial derivatives:

$$\frac{dR_{\text{sq}}}{d\vec{w}} = \left(\frac{\partial R_{\text{sq}}}{\partial w_0}, \frac{\partial R_{\text{sq}}}{\partial w_1}, \ldots, \frac{\partial R_{\text{sq}}}{\partial w_d}\right)^T$$

$\frac{d}{dx} x^3 = 3x^2$

∇f

$\frac{df}{d\vec{x}}$

Idea

- While we could compute each of: $\frac{\partial R_{\text{sq}}}{\partial w_0}$, $\frac{\partial R_{\text{sq}}}{\partial w_1}$, ...
- Instead use rules from vector calculus.
- Example: $\frac{d}{d\vec{w}}\left[\vec{y}^T X \vec{w}\right] = X^T \vec{y}$

$n \times 1$ $n \times d$ $d \times 1$

scalar

$(1 \times n) \cdot (n \times d) \times (d \times 1)$

$= (1 \times d) \times (d \times 1) = 1 \times 1$

Good to know...

$$(A + B)^T = A^T + B^T$$

$$(AB)^T = B^T A^T$$

$$\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u} = \vec{u}^T \vec{v} = \vec{v}^T \vec{u}$$

$$(\vec{u} + \vec{v}) \cdot (\vec{w} + \vec{z}) = \vec{u} \cdot \vec{w} + \vec{u} \cdot \vec{z} + \vec{v} \cdot \vec{w} + \vec{v} \cdot \vec{z}$$

$$\|\vec{u}\|^2 = \vec{u} \cdot \vec{u}$$

Step Two: Find Gradient $\vec{u} \cdot \vec{v} = \vec{v} \cdot \vec{u}$

- Expand norm to make gradient easier.

$$\|X\vec{w} - \vec{y}\|^2 = (X\vec{w} - \vec{y}) \cdot (X\vec{w} - \vec{y})$$

$$= (X\vec{w}) \cdot (X\vec{w}) - (X\vec{w}) \cdot \vec{y} - \vec{y} \cdot (X\vec{w}) + \vec{y} \cdot \vec{y}$$

$$= (X\vec{w}) \cdot (X\vec{w}) - 2\vec{y} \cdot (X\vec{w}) + \vec{y} \cdot \vec{y}$$

$$= \vec{w}^T X^T X \vec{w} - 2\vec{y}^T X\vec{w} + \vec{y}^T \vec{y}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(H(\vec{x}^{(i)}) - y_i \right)^2$$

$$(X\vec{w}) \cdot (X\vec{w}) = (X\vec{w})^T (X\vec{w}) = \vec{w}^T X^T X \vec{w}$$

Step Two: Find Gradient

$$\frac{d}{d\vec{w}}\left[R_{\text{sq}}(\vec{w})\right] = \frac{1}{n}\frac{d}{d\vec{w}}\left[\vec{w}^T X^T X\vec{w} - 2\vec{y}^T X\vec{w} + \vec{y}^T\vec{y}\right]$$

$$= ?$$

$\frac{d}{dx} x^3 = 3x^2$

Claims

$\frac{d}{dz} az^2 = 2az$

- $\frac{d}{d\vec{w}}\left[\vec{w}^T X^T X \vec{w}\right] = 2X^T X\vec{w}$
- $\frac{d}{d\vec{w}}\left[\vec{y}^T X \vec{w}\right] = X^T\vec{y}$
- $\frac{d}{d\vec{w}}\left[\vec{y}^T\vec{y}\right] = \vec{0}$

$\frac{d}{dz}(az) = a$

How?

- General procedure: expand, differentiate, gather
  1. Expand $\vec{v}^T\vec{u}$ until coordinates $u_1, \ldots, u_k$ are visible.
  2. Compute $\partial d/\partial u_1$, $\partial d/\partial u_2$, ..., $\partial d/\partial u_k$.
  3. Gather result in vector form.

Example

$\vec{v} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix}$ $\vec{u} = \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}$

- Show $\frac{d}{d\vec{u}}\left[\vec{v}^T\vec{u}\right] = \vec{v}$.

1) Expand:

$$\vec{v}^T\vec{u} = \begin{pmatrix} v_1 & v_2 & \cdots & v_d \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ \vdots \\ u_d \end{pmatrix}$$

$$= v_1u_1 + v_2u_2 + \cdots + v_du_d$$

Example

- Show $\frac{d}{d\vec{u}}\left[\vec{v}^T\vec{u}\right] = \vec{v}$.

$$\frac{d}{d\vec{u}}\vec{v}^T\vec{u} = \begin{pmatrix} \frac{\partial}{\partial u_1}\vec{v}^T\vec{u} \\ \frac{\partial}{\partial u_2}\vec{v}^T\vec{u} \\ \vdots \\ \frac{\partial}{\partial u_d}\vec{v}^T\vec{u} \end{pmatrix} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix}$$

2) Gradient:

$$\frac{\partial}{\partial u_1}(\vec{v}^T\vec{u}) = \frac{\partial}{\partial u_1}\left[u_1v_1 + u_2v_2 + \dots + u_dv_d\right]$$

$$= v_1 + 0 + 0 + \dots + 0$$

$$\frac{\partial}{\partial u_2}(\vec{v}^T\vec{u}) = \frac{\partial}{\partial u_2}\left[u_1\overset{0}{v_1} + u_2\overset{v_2}{v_2} + \dots + u_d\overset{0}{v_d}\right] =$$

Example

- Show $\frac{d}{d\vec{u}}\left[\vec{v}^T\vec{u}\right] = \vec{v}$.

3) Gather:

$$\frac{d}{d\vec{u}}\vec{v}^T\vec{u} = \begin{pmatrix} v_1 \\ v_2 \\ \vdots \\ v_d \end{pmatrix} = \vec{v}$$

Step Two: Find Gradient

- We claimed

$$\frac{d}{d\vec{w}}\left[\vec{w}^T X^T X \vec{w}\right] = 2X^T X\vec{w} \qquad \frac{d}{d\vec{w}}\left[\vec{y}^T X\vec{w}\right] = X^T\vec{y} \qquad \frac{d}{d\vec{w}}\left[\vec{y}^T\vec{y}\right] = 0$$

- So:

$$\frac{d}{d\vec{w}}\left[R_{sq}(\vec{w})\right] = \frac{1}{n}\frac{d}{d\vec{w}}\left[\vec{w}^T X^T X\vec{w} - 2\vec{y}^T X\vec{w} + \vec{y}^T\vec{y}\right]$$

$$= \frac{1}{n}\left[2X^T X\vec{w} - 2X^T\vec{y} + \vec{0}\right]$$

Solution

- We have found:

$$\frac{d}{d\vec{w}}\left[R_{\text{sq}}(\vec{w})\right] = \frac{1}{n}\left(2X^TX\vec{w} - 2X^T\vec{y}\right)$$

- To minimize $R_{\text{sq}}(\vec{w})$, set gradient to zero, solve:

$$2X^TX\vec{w} - 2X^T\vec{y} = 0 \implies X^TX\vec{w} = X^T\vec{y}$$

- This is a system of equations in matrix form, called the **normal equations**.

The Normal Equations

- The least squares solutions for $\vec{w}$ are found by solving the **normal equations**:

$$X^T X \vec{w} = X^T \vec{y}$$

- Mathematically, solved by:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

- In practice: **don't invert**! Use the **Singular Value Decomposition** (SVD).

Aside

- $X^+ = (X^T X)^{-1} X^T$ is known as the **Moore-Penrose pseudoinverse** of X.
- A generalization of the matrix inverse for non-square matrices.
- With this, the least squares solution is: $\vec{w}^* = X^+ \vec{y}$

Two Approaches

- **Approach #1:** Vector Calculus
- **Approach #2:** Orthogonal Projection (Geometry)
- Different ways of arriving at same solutions.

DSC 140A

Probabilistic Modeling & Machine Learning

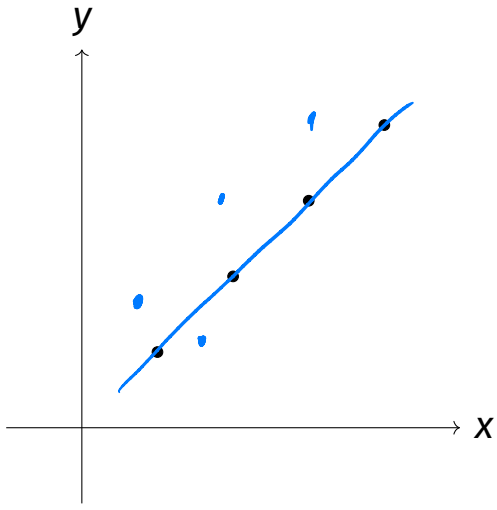
Lecture 3 | Part 3

Approach #2: Geometry

Recall

- Recall the **column space** of a matrix A: the set of all vectors that can be made from a linear combination of the columns of A.

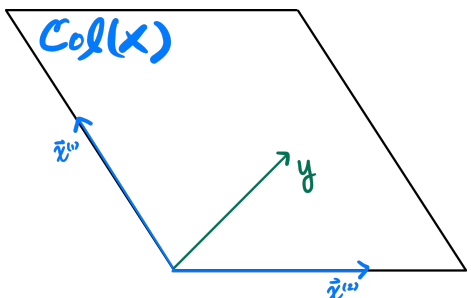
- $\vec{v}$ is in the column space of A if and only if there is an $\vec{u}$ such that $A\vec{u} = \vec{v}$.


y
x

Approach #2: Geometry

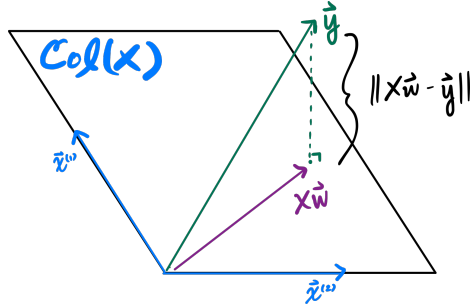
- In the collinear case, $\vec{y}$ can be predicted **exactly**.
- That is, there is a $\vec{w}$ such that $X\vec{w} = \vec{y}$.
  - $\vec{y}$ is in the column space of X.

$X\vec{w} = \vec{y}$



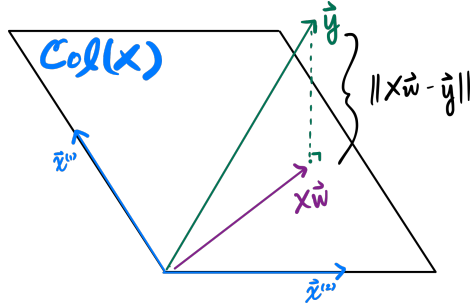
Approach #2: Geometry

- In practice, $\vec{y}$ is **not** in the **column space** of X.
- **Goal:** choose $\vec{w}$ so that $X\vec{w} \approx \vec{y}$.



Approach #2: Geometry

- **Idea:** $\|X\vec{w} - \vec{y}\|$ is smallest when $X\vec{w}$ is the **orthogonal projection** of $\vec{y}$ onto the column space of X.



Recall: Projections

- Let A be a matrix and $\vec{u}$ be a vector.
- The **orthogonal projection** of $\vec{u}$ onto the column space of A is given by:

$$A(A^TA)^{-1}A^T\vec{u}$$

Approach #2: Geometry

- So, the orthogonal projection of $\vec{y}$ onto the column space of X is given by:

$$X \underbrace{(X^T X)^{-1} X^T \vec{y}}_{\vec{w}^*}$$

Least Squares Regression

- We can now fit functions of the form:

$$H(\vec{x}) = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d$$

  by minimizing expected square loss.

- Example:
  - x_1: years of experience
  - x_2: # of interview questions missed
  - x_3: favorite number

Procedure

1. Form (augmented) $n \times (d+1)$ **design matrix**:

$$X = \begin{pmatrix} \text{Aug}(\vec{x}^{(1)}) \longrightarrow \\ \text{Aug}(\vec{x}^{(2)}) \longrightarrow \\ \vdots \\ \text{Aug}(\vec{x}^{(n)}) \longrightarrow \end{pmatrix} = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_d^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_d^{(2)} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_1^{(n)} & x_2^{(n)} & \dots & x_d^{(n)} \end{pmatrix}$$

2. Solve the **normal equations**:

$$\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$$

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 3 | Part 4

Example: Gaussian Basis Functions

Recall

- The “linear” in “linear regression” refers to the fact that we are fitting a linear function of the parameters.

Recall: Basis Functions

- We can fit any function of the form:

$$H(\vec{x}; \vec{w}) = w_0 + w_1\phi_1(\vec{x}) + w_2\phi_2(\vec{x}) + \ldots + w_k\phi_k(\vec{x})$$

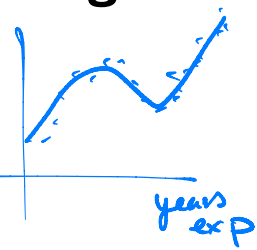
- $\phi_i(\vec{x}) : \mathbb{R}^d \to \mathbb{R}$ is called a **basis function**.

Example: Polynomial Curve Fitting

- Fit a function of the form:

$$H(x; \vec{w}) = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

z_1 z_2 z_3



- Use basis functions:

$$\phi_0(x) = 1 \qquad \phi_1(x) = x \qquad \phi_2(x) = x^2 \qquad \phi_3(x) = x^3$$

Procedure

1. Define $\vec{\phi}^{(i)} = (\phi_1(\vec{x}^{(i)}), \phi_2(\vec{x}^{(i)}), \ldots, \phi_k(\vec{x}^{(i)}))^T$

2. Form $n \times k$ **design matrix**:

$$\Phi = \begin{pmatrix} \text{Aug}(\vec{\phi}^{(1)}) \longrightarrow \\ \text{Aug}(\vec{\phi}^{(2)}) \longrightarrow \\ \vdots \\ \text{Aug}(\vec{\phi}^{(n)}) \longrightarrow \end{pmatrix} = \begin{pmatrix} \phi_1(\vec{x}^{(1)}) & \phi_2(\vec{x}^{(1)}) & \ldots & \phi_k(\vec{x}^{(1)}) \\ \phi_1(\vec{x}^{(2)}) & \phi_2(\vec{x}^{(2)}) & \ldots & \phi_k(\vec{x}^{(2)}) \\ \vdots & \vdots & \vdots & \vdots \\ \phi_1(\vec{x}^{(n)}) & \phi_2(\vec{x}^{(n)}) & \ldots & \phi_k(\vec{x}^{(n)}) \end{pmatrix}$$

3. Solve the **normal equations**:

$$\vec{w}^* = (\Phi^T\Phi)^{-1}\Phi^T\vec{y}$$

Example: Polynomial Curve Fitting

- Design matrix becomes:

$$\Phi = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \dots & \dots & \ddots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix}$$

Gaussian Basis Functions

- **Gaussians** make for useful basis functions.

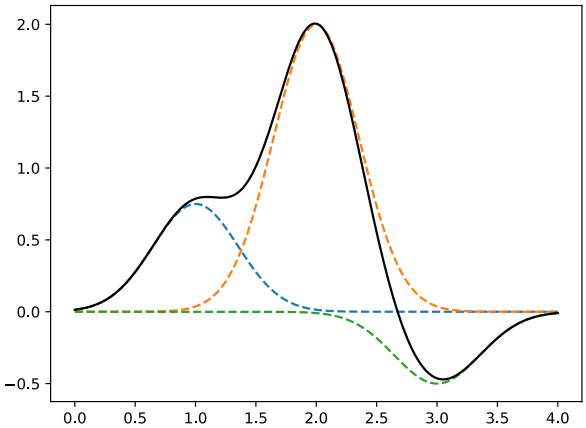
$$\phi_i(x) = \exp\left(-\frac{(x - \mu_i)^2}{\sigma_i^2}\right)$$

- Must specify[1] **center** μ_i and **width** σ_i for each Gaussian basis function.

[1] You pick these; they are not learned!

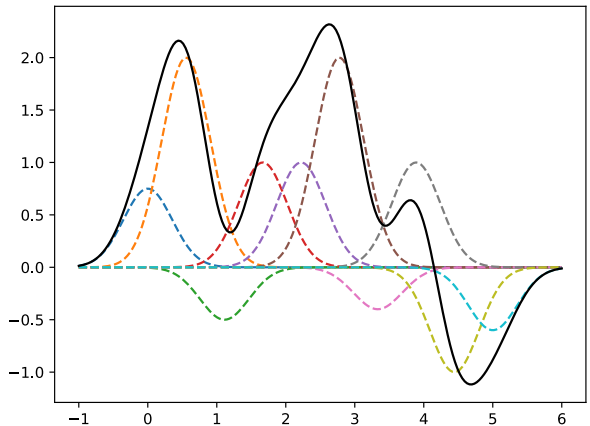
Example: $k = 3$

- A function of the form: $H(x) = w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x)$, using 3 Gaussian basis functions.

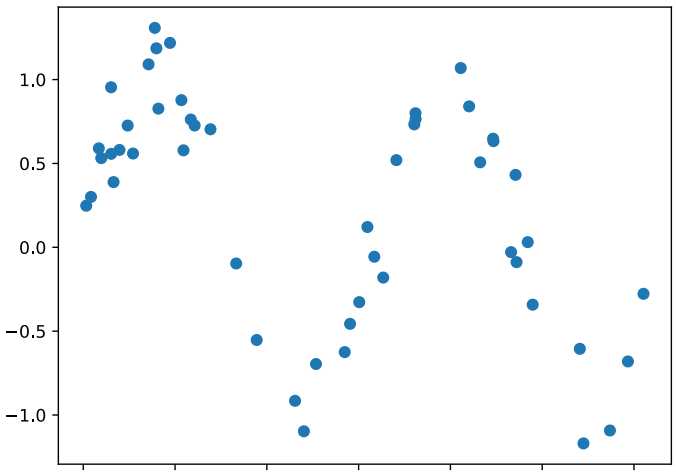


Example: $k = 10$

- The more basis functions, the more complex H can be.

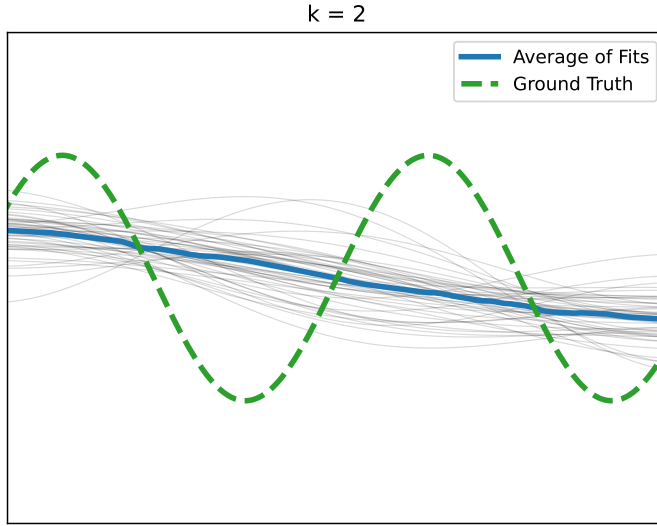


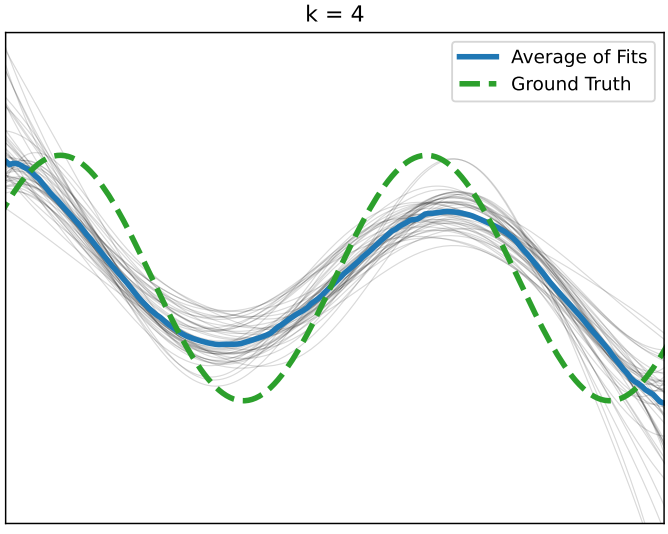
Example

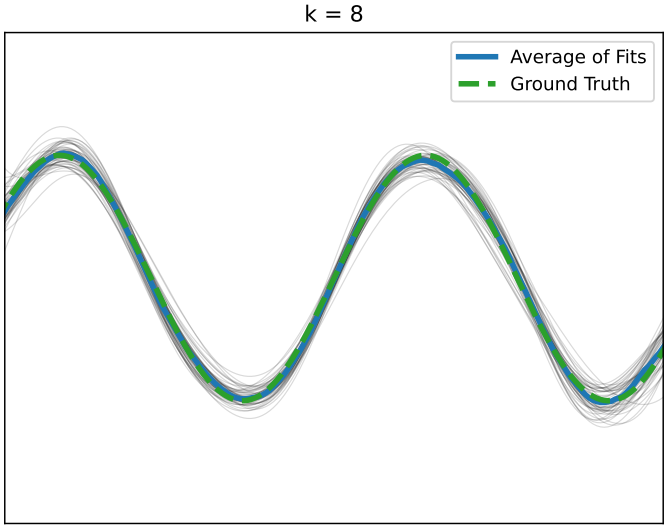


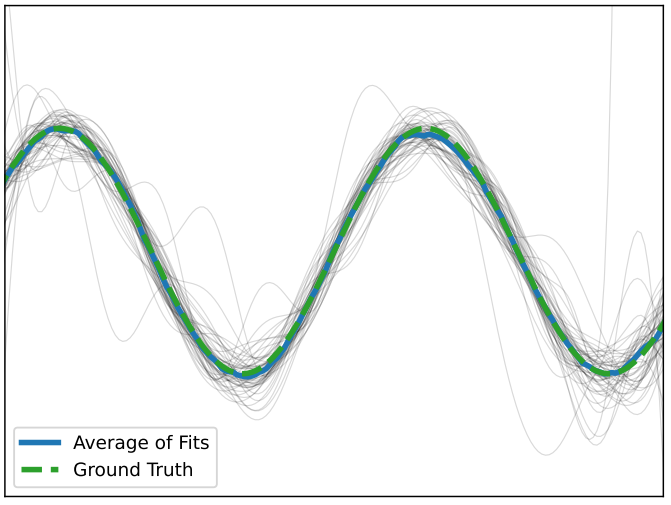
Example

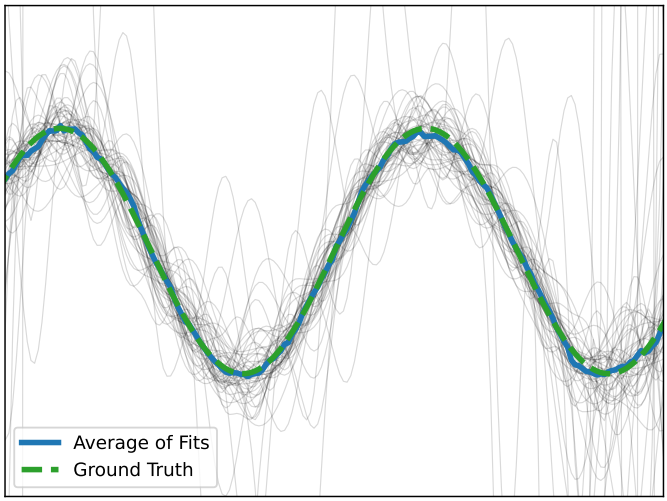
- Will use k Gaussian basis functions, equally spaced in interval $[0, 2\pi]$.
- Same width for each: $\sigma_i = \sigma = 2\pi/k$.
- Will fit on 50 random sinusoidal datasets.


k = 2
Average of Fits
Ground Truth


k = 4
Average of Fits
Ground Truth

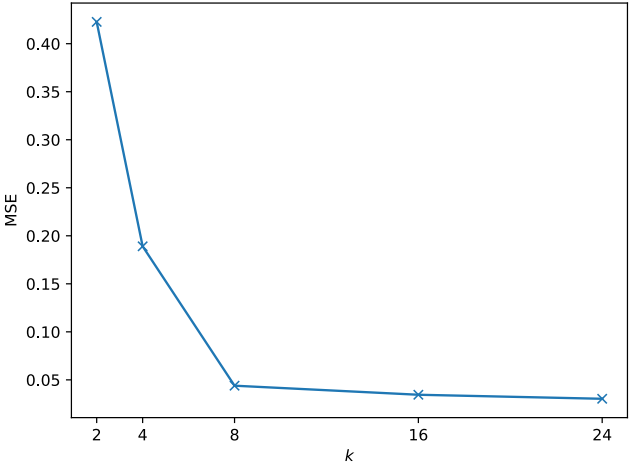

k = 8
Average of Fits
Ground Truth


k = 16
Average of Fits
Ground Truth

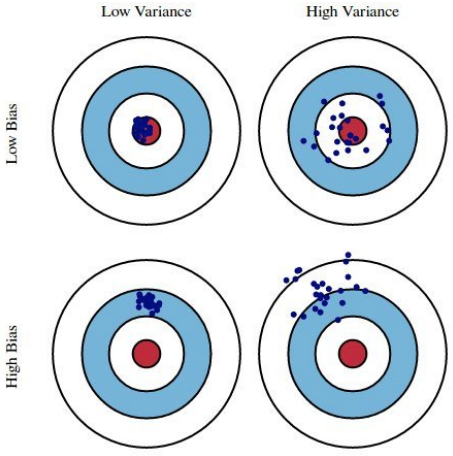

k = 24
Average of Fits
Ground Truth

Observation #1

- **Mean Squared Error** decreases as k increases.



Observation #2



Observation #2

- When k is small: **high bias**, **low variance**
- When k is large **low bias**, **high variance**
- There appears to be a tradeoff.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 3 | Part 5

Linear Classification

Classification

- We've been considering **regression**. What about **classification**?

- In classification, output is a discrete **label**.

Example: Mango Ripeness

- Predict whether a mango is ripe given greenness and hardness.

- Convention: encode “yes“ as 1 and “no” as -1.

Greenness	Hardness	Ripe
0.7	0.9	1
0.2	0.5	-1
0.3	0.1	-1
⋮	⋮	⋮

Binary vs. Multiclass Classification

- **Binary** classification: predict “yes” / “no”.
  - E.g., is this picture of a dog? (“yes” / “no”)
- **Multiclass** classification: predict one of k possible labels.
  - E.g., what animal is in this picture?
  - cat, dog, giraffe, mongoose, etc.
- We’ll focus on **binary** for simplicity.

Linear Classification

- Linear prediction functions output real numbers:

$$H(\vec{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

- Can turn output into a “yes” / “no” answer by thresholding:

$$\text{sign}(z) = \begin{cases} 1 & z > 0 \\ -1 & z < 0 \\ 0 & \text{otherwise} \end{cases}$$

- Final prediction: $\text{sign}(H(\vec{x}))$

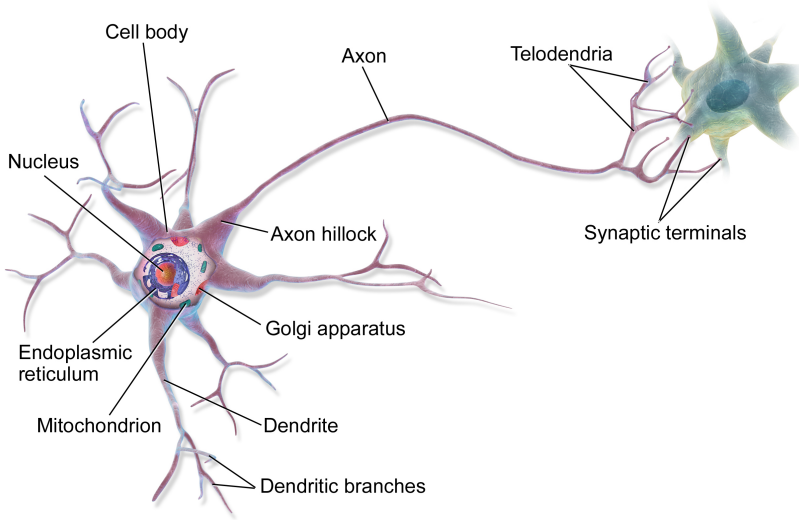
Interpretation: Weighted Vote

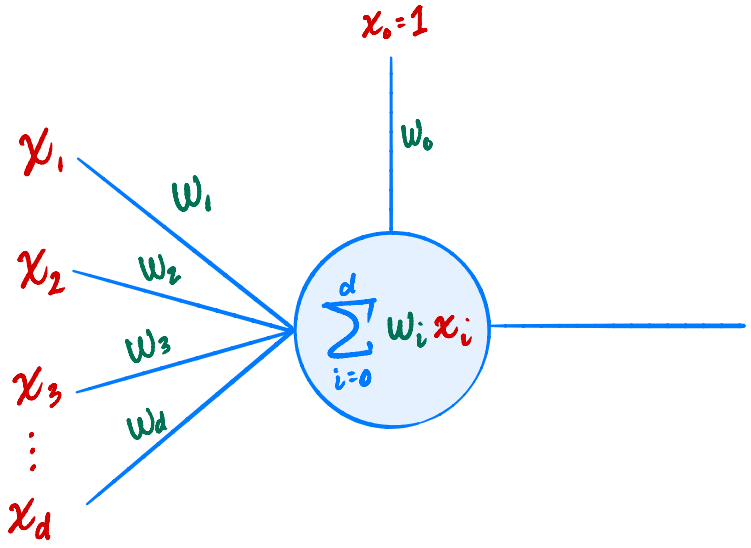
- A linear decision function can be thought of as a **weighted vote**:

$$H(\vec{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d$$

Interpretation: A Simple Neuron

- A linear decision function is a simple model for a neuron.
- Are the basis of modern neural networks.


Cell body
Axon
Telodendria
Nucleus
Axon hillock
Synaptic terminals
Golgi apparatus
Endoplasmic reticulum
Mitochondrion
Dendrite
Dendritic branches


$x_0 = 1$
w_0
x_1
w_1
x_2
w_2
w_3
x_3
w_d
x_d
$\sum_{i=0}^{d} w_i x_i$

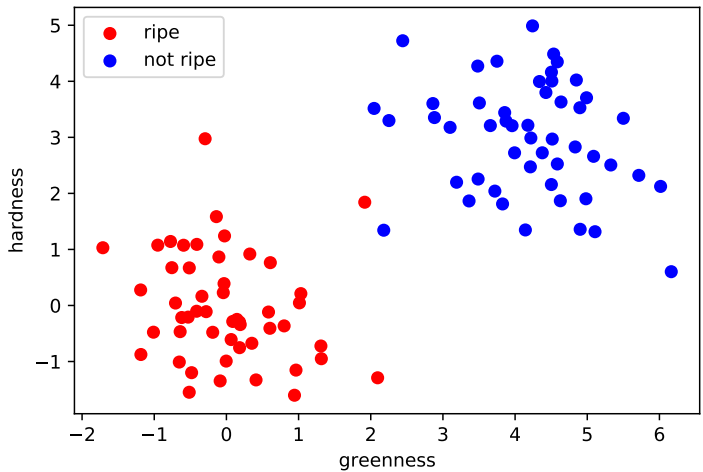
Decision Boundary

- The **decision boundary** is the place where the output of $H(x)$ switches from "yes" to "no".
  - If $H > 0 \mapsto$ "yes" and $H < 0 \mapsto$ "no", the decision boundary is where $H = 0$.

- If H is a linear predictor and[2]
  - $\vec{x} \in R^1$, then the decision boundary is just a number.
  - $\vec{x} \in \mathbb{R}^2$, the boundary is a straight line.
  - $\vec{x} \in \mathbb{R}^d$, the boundary is a $d - 1$ dimensional (hyper) plane.

[2] when plotted in the original feature coordinate space!

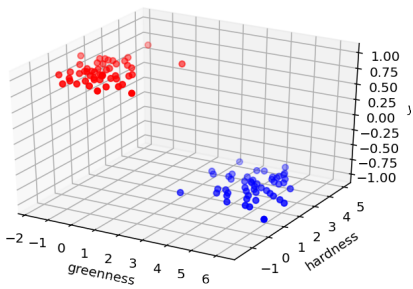
Example

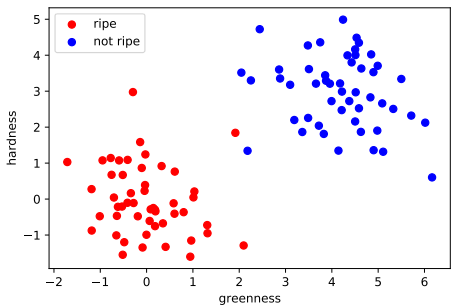


Prediction Surface

- Although our final prediction uses only the sign, the magnitude of H is useful.
- Recall: the plot of a linear prediction function H is a (hyper)plane.
- The problem of binary classification can be though of as regression where the targets are 1 and -1.

Example





Useful Fact

- $|H(\vec{x})|$ is proportional to the distance from the decision boundary.
- If $H(\vec{x}^{(1)}) = H(\vec{x}^{(2)})$, $\vec{x}^{(1)}$ and $\vec{x}^{(2)}$ are equally far away from the boundary.
- If $H(\vec{x}) = 0$, $\vec{x}$ is on the boundary.
- $|H(\vec{x})|$ can be used as a measure of “confidence”.

DSC 140A

Probabilistic Modeling & Machine Learning

Lecture 3 | Part 6

ERM for Linear Classifiers

Learning a Classifier

- We can learn a linear classifier using the same ERM paradigm as before.

Empirical Risk Minimization

- Step 1: choose a **hypothesis class**
  - Let's assume we've chosen linear predictors
- Step 2: choose a **loss function**
- Step 3: minimize **expected loss (empirical risk)**

Exercise

Can we use the square loss for classification?

$$(H(\vec{x}^{(i)}) - y_i)^2$$

Square Loss for Classification

- **Yes!** We can use the square loss, but it may not be the best choice.

- E.g., suppose $H(\vec{x}^{(i)}) = 11$ and $y_i = 1$.
  - Square loss: 100. **Large!**

- E.g., suppose $H(\vec{x}^{(i)}) = -9$ and $y_i = 1$.
  - Square loss:

Main Idea

While the square loss can be used for classification, it may not be the best choice because it penalizes predictions that are “very correct”.

It is designed for regression.

Least Squares Classifier

- **Least squares classification** is performed *exactly* the same as least squares regression.
  - I.e., solve the normal equations: $\vec{w}^* = (X^T X)^{-1} X^T \vec{y}$

- Except the prediction is thresholded:

$$H(\vec{x}) = \text{sign}\left(\text{Aug}\left(\vec{x}\right) \cdot \vec{w}^*\right)$$

Least Squares and Outliers

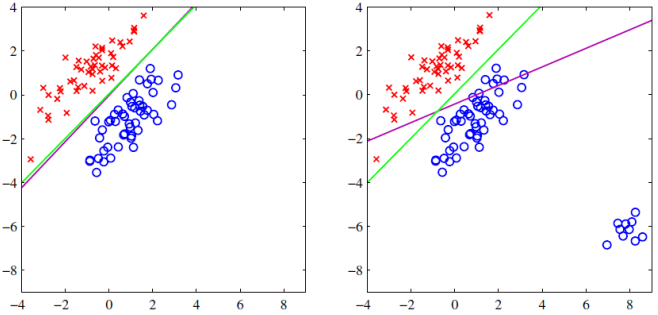


Figure 4.4 The left plot shows data from two classes, denoted by red crosses and blue circles, together with the decision boundary found by least squares (magenta curve) and also by the logistic regression model (green curve), which is discussed later in Section 4.3.2. The right-hand plot shows the corresponding results obtained when extra data points are added at the bottom left of the diagram, showing that least squares is highly sensitive to outliers, unlike logistic regression.

[3]Bishop, *Pattern Recognition and Machine Learning*

Another Loss Function

- What about the **0-1 loss**?
  - Loss = 0 if prediction is **correct**.
  - Loss = 1 if prediction is **incorrect**.

- More formally:

$$L_{0\text{-}1}(H(\vec{x}^{(i)}), y_i) = \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

Expected 0-1 Loss

- The expected 0-1 loss (empirical risk) has a nice interpretation:

$$R_{0\text{-}1}(H) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} 0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i \end{cases}$$

Exercise

What is it?

Answer

- The empirical risk with respect to the 0-1 loss is (1 - the **accuracy**) of the classifier.

$$R_{0\text{-}1}(H) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases}0 & \text{if } \text{sign}(H(\vec{x}^{(i)})) = y_i \\ 1 & \text{if } \text{sign}(H(\vec{x}^{(i)})) \neq y_i\end{cases}$$

$$= \frac{\#\text{ of } \textbf{incorrect} \text{ predictions}}{n}$$

ERM for the 0-1 Loss

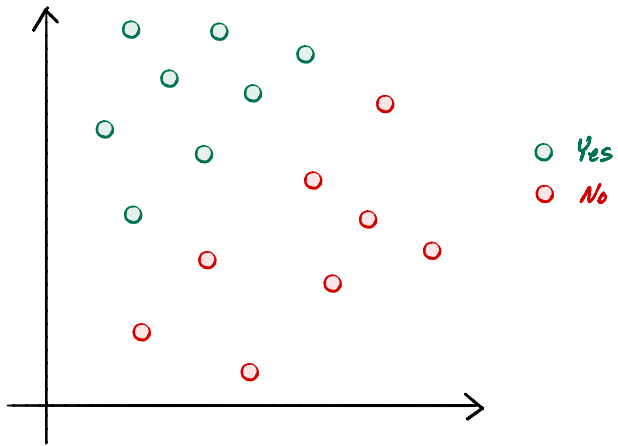
- Assume the 0-1 loss, linear prediction functions.
- Next step: find linear H minimizing the empirical risk.
- That is: find H which maximizes the **accuracy** on the training set.

Problem

- The 0-1 loss is not differentiable.
- Can’t even use gradient descent...

Why?

- The 0-1 risk surface is flat almost everywhere.



Computationally Difficult

- It is **not feasible** to minimize 0-1 risk in general.
- More formally: NP-Hard to optimize expected 0-1 loss in general.[4]

[4] It is efficiently doable if the classes are linearly separable by finding convex hulls of each class. If non-separable, it is difficult.

Main Idea

It is computationally difficult (NP-Hard) to maximize the accuracy of a linear classifier.

Surrogate Loss

- Instead of the 0-1 loss, we use a **surrogate loss**.
- That is, a loss that is similar in spirit, but has better mathematical properties.

The Perceptron Loss

- The **perceptron loss** is designed for binary classification.
  - Loss = 0 if prediction is correct.
  - Loss > 0 if prediction incorrect.
  - Loss increases with distance from boundary.

- More formally:

$$L_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

ERM for the Perceptron

- **Goal:** minimize the empirical expected perceptron loss (risk):

$$H_{\text{tron}}(H(\vec{x}), y) = \begin{cases} 0, & \text{sign}(H(\vec{x})) = \text{sign}(y) \\ |H(\vec{x})|, & \text{sign}(H(\vec{x})) \neq \text{sign}(y) \end{cases}$$

Problem

- The percepton loss is **not differentiable**.
- However:
  - it is **not flat**!
  - its derivative exists almost everywhere.
- We can train iteratively using **gradient descent** (next time).

Some History

- Perceptrons were one of the first “machine learning” models.
- The basis of modern neural networks.

Rosenblatt's Perceptron

