



CS 540 Introduction to Artificial Intelligence
Linear Models & Linear Regression
University of Wisconsin-Madison
Fall 2026 Sections 1 & 2

Announcements

- HW3 due on **Friday Oct 2nd at 11:59 PM**

- Class roadmap:

ML Linear Regression
Machine Learning: K - Nearest Neighbors & Naive Bayes
Machine Learning: Neural Networks I (Perceptron)

} Supervised Learning

Outline

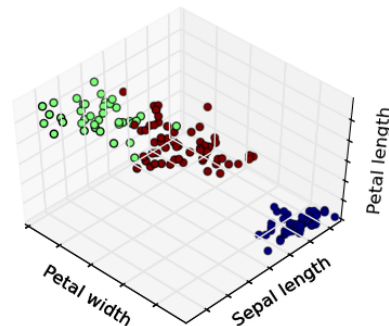
- Unsupervised Learning Recap
- Supervised Learning with Linear Models
 - Parameterized model, model classes, linear models, train vs. test
- Linear Regression
 - Least squares, normal equations, residuals, logistic regression

Unsupervised Learning Recap

Dataset: $x_1, x_2, \dots, x_n$ **No labels**

Goal: find patterns & structures that help better understand data.

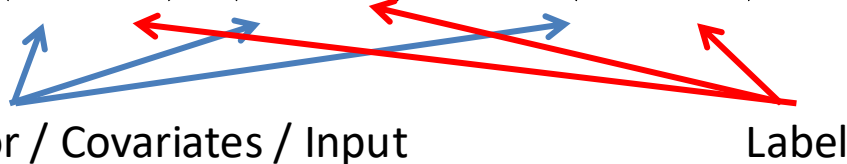
- Clustering:
 - Hierarchical Clustering
 - Center-based Clustering (*k-means*)
 - Spectral clustering
- Dimensionality reduction
 - PCA
 - Visualization via T-SNE
- Density Estimation



Supervised Learning

Supervised learning:

- Make predictions, classify data, perform regression
- Dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$



- Goal: find function $f : X \rightarrow Y$ to predict label on **new** data

Select a **model** f
from a class of
possible models



indoor



outdoor

Regression

- Continuous label $y \in \mathbb{R}$
- Squared loss function $\ell(f(x), y) = (f(x) - y)^2$
- Finding f that minimizes the empirical risk

$$\frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$$

Common Recipe for Supervised Learning

1. Select model class
2. Select loss
3. Optimize parameters of the model class
4. Evaluate output

Functions/Models

The function f is usually called a model

- Which possible functions should we consider?
- One option: **all functions**

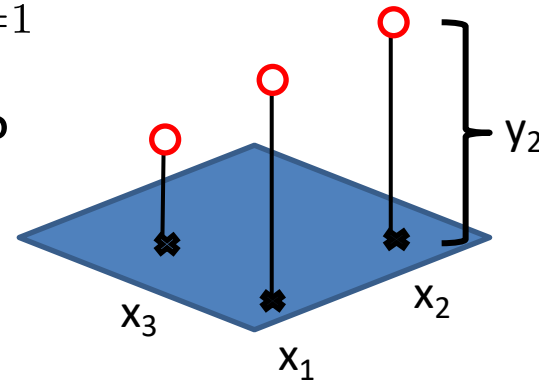
- Not a good choice. Consider

$$f(x) = \sum_{i=1}^n 1\{x = x_i\} y_i$$

- Training loss: **zero**. Can't do better!

- How will it do on x not in the training set?

(cannot generalize)



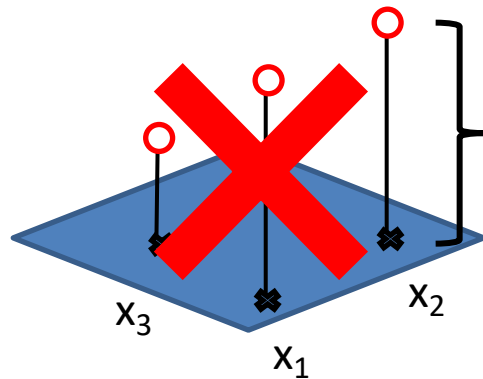
Functions/Models

Don't want all functions

- Instead, pick a specific class
- Parametrize it by weights/parameters
- **Example:** linear models

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d = \theta_0 + x^T \theta$$

Weights/ Parameters



Training The Model

- Parametrize it by weights/parameters
- Minimize the loss

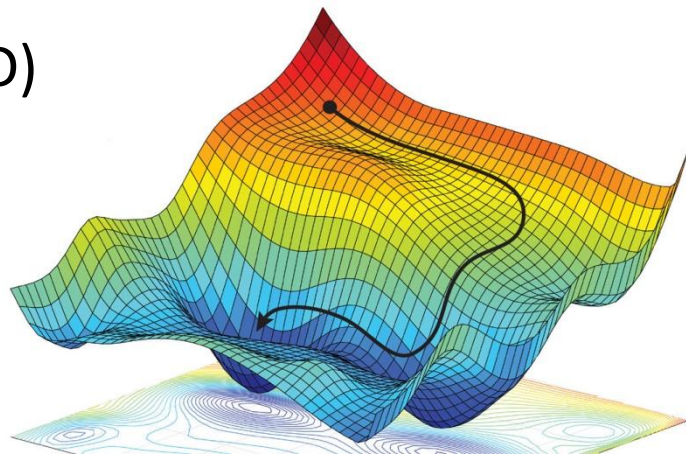
Best parameters = best function f  $\min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(f(x_i), y_i)$

$= \frac{1}{n} \sum_{i=1}^n \ell(\theta_0 + x_i^T \theta, y_i)$  Linear model class f

$= \frac{1}{n} \sum_{i=1}^n (\theta_0 + x_i^T \theta - y_i)^2$  Square loss

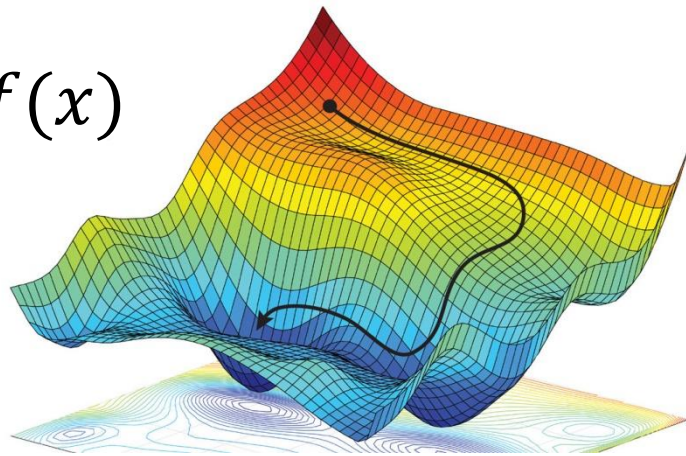
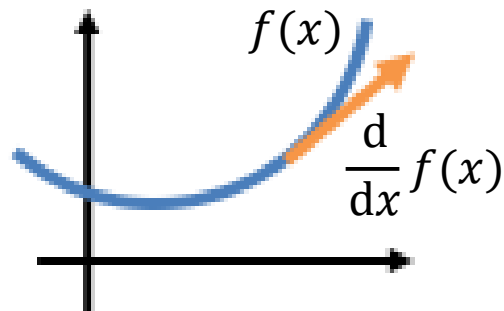
How Do We Optimize Parameters?

- Need to solve something that looks like $\min_{\theta} L(\theta)$
- Generic optimization problem; many algorithms
- Most popular: variants of **gradient descent**
 - Stochastic Gradient Descent (SGD)
 - Momentum
 - Adam



Gradients & Gradient Descent

- One dimension: derivative $\frac{d}{dx} f(x)$
 - How to shift x to make $f(x)$ larger
- Higher dimensions: gradient $\nabla f(x)$
 - Direction where f grows fastest



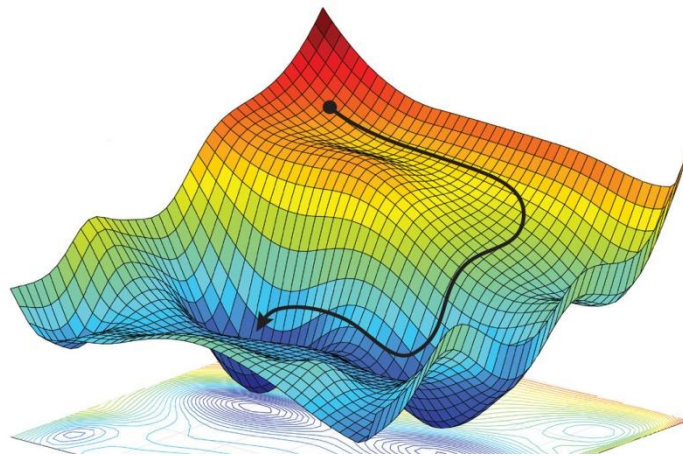
Gradients & Gradient Descent

- Gradient descent takes iterative steps to make loss function smaller

Gradient Descent

Input: dataset (X, y) , loss function L ,
number of steps T , step size η

1. Initialize θ_0
2. For $t = 1, 2, \dots, T$
3. Calculate $g_t = \nabla L(\theta_{t-1}; X, y)$
4. Update $\theta_t \leftarrow \theta_{t-1} - \eta g_t$
5. Return θ_T



Train vs Test

Now we've trained, have some f parametrized by θ

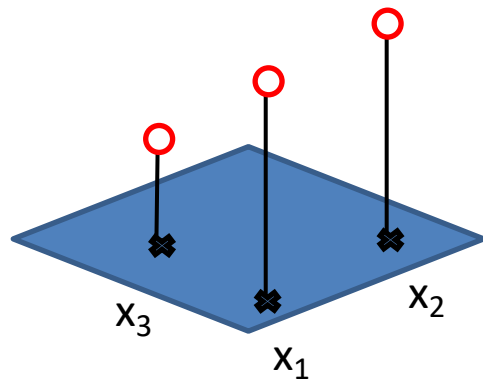
- Train loss is small $\rightarrow f$ predicts most x_i correctly
- How does f do on points not in training set? **“Generalizes!”**
- To evaluate this, reserve a **test** set. Do **not** train on it!

$$\underbrace{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)}_{\text{Training Data}} \quad \underbrace{(\mathbf{x}_{n+1}, y_{n+1}), \dots, (\mathbf{x}_{n+p}, y_{n+p})}_{\text{Test Data}}$$

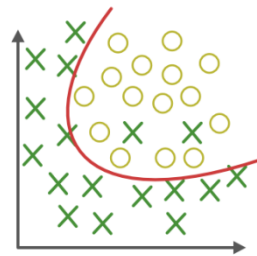
Train vs Test

Use the test set to evaluate f

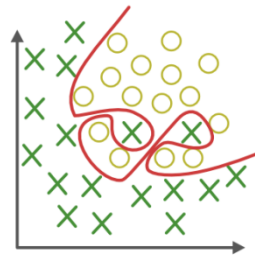
- Why? Back to our “perfect” train function
- Training loss: 0. Every point matched perfectly
- How does it do on test set? **Fails completely!**



- Test set helps detect **overfitting**
 - Overfitting: too focused on train points
 - “Bigger” class: more prone to overfit
 - Need to consider **model capacity**



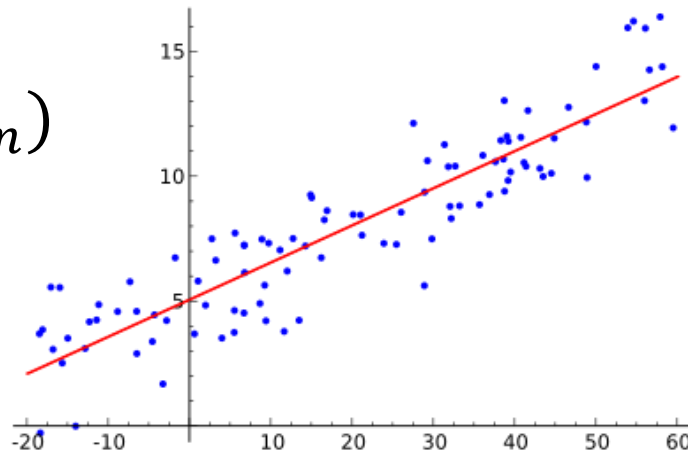
Appropriate-fitting



Over-fitting

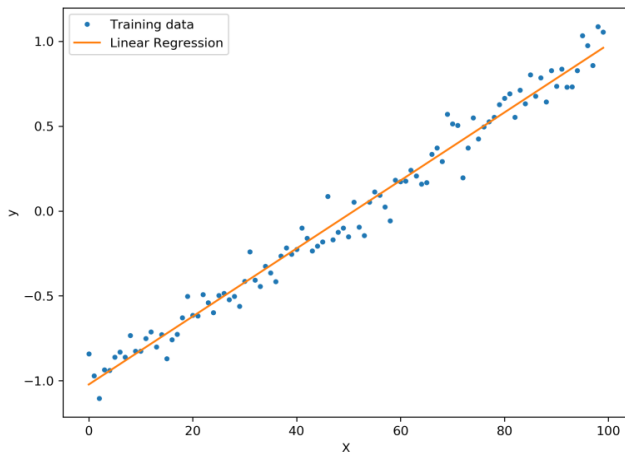
Linear Regression

- Simplest form of regression
- Find a line that fits the data
- Example:
 - Training data $(x_1, y_1), \dots, (x_n, y_n)$
 - Find $a, b \in \mathbb{R}$ such that
$$\forall i, y_i \approx ax_i + b$$



Linear Regression

- **Inputs:** $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$
 - $\mathbf{x}$'s are vectors, y 's are scalars.
 - “**Linear**”: predict a linear combination of $\mathbf{x}$ components + intercept



C. Hansen

$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_d x_d = \theta_0 + x^T \theta$$

- We should estimate the parameters θ

Linear Regression Setup

Problem Setup

- Goal: figure out how to minimize square loss
- Let's organize it. Train set

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$$

Notational Trick

- When x is a scalar: $f_{(a,b)}(x) = ax + b$
- When x is a vector:
$$f(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_d x_d = \theta_0 + x^T \theta$$
- Give x a “dummy dimension” to simplify notation

$$\begin{array}{ll} \text{Old} & \text{New} \\ x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} & x = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \end{array} \quad f(x) = [1 \quad x_1 \quad x_2 \quad \cdots \quad x_d] \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ x_d \end{bmatrix} = \langle x, \theta \rangle = x^T \theta$$

Linear Regression Setup

Problem Setup

- Train set $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)$
- Take train features and make it a $n \times (d+1)$ matrix, and y a vector:

$$X = \begin{bmatrix} x_1^T \\ \dots \\ x_n^T \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ \dots \\ y_n \end{bmatrix}$$

- Then, the empirical risk is $\frac{1}{n} \|X\theta - y\|^2$

Finding The Estimated Parameters

Have our loss: $\frac{1}{n} \|X\theta - y\|^2$

- Could optimize it with SGD, etc...
- But the minimum also has a closed-form solution

(vector calculus, proof in the Suggested Readings-optional):


$$\hat{\theta} = (X^T X)^{-1} X^T y$$

Hat: indicates an
estimate



Not always
invertible...

**“Normal
Equations”**

How Good are the Estimated Parameters?

Now we have parameters $\hat{\theta} = (X^T X)^{-1} X^T y$

- How good are they?
- Predictions are $f(x_i) = \hat{\theta}^T x_i = ((X^T X)^{-1} X^T y)^T x_i$
- Errors (“residuals”)

$$|y_i - f(x_i)| = |y_i - \hat{\theta}^T x_i| = |y_i - ((X^T X)^{-1} X^T y)^T x_i|$$

- If data is linear, residuals are 0. Almost never the case!
- **Mean squared error** on a test set $\frac{1}{m} \sum_{i=n+1}^{n+m} (\hat{\theta}^T x_i - y_i)^2$

Linear Regression → Classification?

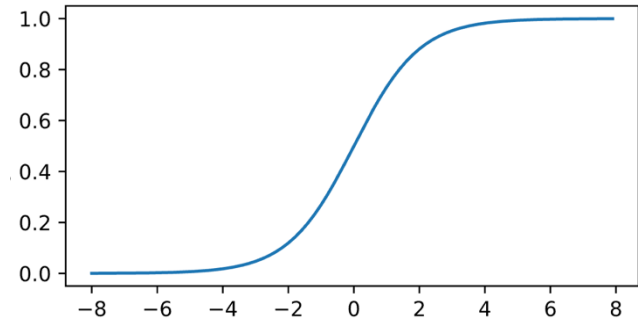
What if we want the same idea, but y is 0 or 1?



- Need to convert the $\theta^T x$ to a probability in $[0,1]$

Logistic function

$$p(y = 1|x) = \frac{1}{1 + \exp(-\theta^T x)}$$



Why does this work?

- If $\theta^T x$ is really big, $\exp(-\theta^T x)$ is really small $\rightarrow p$ close to 1
- If really negative exp is huge $\rightarrow p$ close to 0

“Logistic Regression”

Suggested Reading

- Linear regression, logistic regression, stochastic gradient descent by Prof. Zhu
<https://pages.cs.wisc.edu/~jerryzhu/cs540/handouts/regression.pdf>