# CMPUT 267 Basics of Machine Learning

## Linear Regression, Polynomial Regression

UNIVERSITY OF ALBERTA

March 5, 2024

## Announcements

- Keep track of assignments!

## Outline

## Recap: Linear Regression

A **linear predictor** has the form

$$f(\mathbf{x}) = w_0 + w_1 x_1 + \ldots + w_d x_d = \sum_{j=0}^{d} w_j x_j = \mathbf{w}^T \mathbf{x}$$

- Probabilistic approach:
  1. Assume **iid Gaussian noise**: $Y \sim \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$. $(y_i = \sum_{j=0}^{d} w_j x_{i,j} + \varepsilon_i)$
  2. Use MLE to estimate model from the resulting parametric family

$$\mathcal{F} = \{p(\cdot \mid \mathbf{x}) = \mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2) \mid \mathbf{w} \in \mathbb{R}^{d+1}\}$$

  3. Use the optimal predictor for the estimated model:

$$f^*(\mathbf{x}) = \mathbb{E}[Y \mid \mathbf{X} = \mathbf{x}] = \mathbf{w}^T\mathbf{x}$$

## Solving Linear Regression

- We derived the solution in analytical form

$$\underbrace{\left(\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i\mathbf{x}_i^\top\right)}_{A}\mathbf{w} = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i y_i}_{b}$$

$$\implies \mathbf{A}\mathbf{w} = \mathbf{b} \implies \mathbf{w} = \mathbf{A}^{-1}\mathbf{b} \quad \text{if } \mathbf{A} \text{ is invertible}$$

- But this can be costly, $O(nd^2 + d^3)$.
- Numerical solution. Stochastic Gradient Descent
  - $O(kbd)$ for $k$ iterations of SGD.

## Linear Regression for Nonlinear Predictors

- What if $f^*$ is not linear?

$$f(x) = w_o + w_1 x + w_2 x^2 + \ldots + w_p x^p.$$

- Case 1. $\underline{x \in \mathbb{R}}$ . Learn $f(x) = w_o + w_1 x + w_2 x^2 + \ldots + w_p x^p$ $p$-th degree polynomial
  Create a new function that is a transformation of the polynomial.

$$\underbrace{\phi_j(x) = x^j}$$

$$\phi(x) = [\overset{\phi_0(x)}{1}, \overset{\phi_1(x)}{x}, x^2, x^3, \ldots, \overset{\phi_p(x)}{x^p}] \quad \phi(\mathbf{x}) = \begin{bmatrix} \phi_0(\mathbf{x}) \\ \phi_1(\mathbf{x}) \\ \vdots \\ \phi_p(\mathbf{x}) \end{bmatrix}$$

$$f(\mathbf{x}) = <\underline{\phi(\mathbf{x}), \mathbf{w}}> = \boxed{\overset{\mathbb{R}}{\phi(\mathbf{x})^\top} \mathbf{w}} = \underbrace{\sum_{j=0}^{p} w_j x^j}$$

## Polynomial Regression

$$f(\mathbf{x}) = w_0 + w_1\mathbf{x} + w_2\mathbf{x}^2 + \cdots + w_p\mathbf{x}^p$$

(polynomial of degree $p$)

- Case 1: $x \in \mathbb{R}$
- Learn: $f(x) = w_0 + w_1x + w_2x^2 + \cdots + w_px^p$.
  1. Feature mapping: a new function that is a transformation of the polynomial
     $\phi(x) = (\phi_0(x), \phi_1(x), \phi_2(x), \ldots, \phi_p(x)), \quad \phi_j(x) = x^j$ (basis functions)
  2. Create a transformed dataset $\tilde{\mathcal{D}} = \{(\phi(x_i), y_i)\}_{i=1}^n$
  3. Apply linear regression on $\phi(x)$. Now, we want to learn
     $f(x) = \sum_{j=0}^p w_jx^j = \sum_{j=0}^p w_j\phi_j(x) = \mathbf{w}^\intercal\phi(x)$, $\mathbf{w} \in \mathbb{R}^{p+1}$.
  4. To predict on $x_{\text{new}}$, $\hat{y} = f(x_{\text{new}}) = \sum_{j=0}^p w_jx_{\text{new}}^j$
     $x_{\text{new}} \to \phi(x_{\text{new}}) = (1, x_{\text{new}}, x_{\text{new}}^2, \ldots, x_{\text{new}}^p) \quad \hat{y} = \phi(x_{\text{new}})^\intercal\mathbf{w}$

## Polynomial Regression, $d > 1$

$[x_1, x_2]$

- Case 2: $x \in \mathbb{R}^2, p \underline{= 2}$
- Learn:
  $f(\mathbf{x}) = w_0 + w_1 \underline{x_{[1]}} + w_2 \underline{x_{[2]}} + w_3 x_{[1]} x_{[2]} + w_4 x_{[1]}^2 + w_5 x_{[2]}^2.$
  1. Feature mapping
  2. Create a transformed dataset $\tilde{\mathcal{D}} = \{(\phi(\mathbf{x}_i), y_i)\}_{i=1}^n$
  3. Apply linear regression on $\phi(x)$. Now, we want to learn

     $f(\mathbf{x}) = \sum_{j=0}^5 w_j \phi_j(\mathbf{x}) \ = \mathbf{w}^\top \phi(\mathbf{x})$ ,

     $\mathbf{w} \in \mathbb{R}^5$

$$\phi(\mathbf{x}) = \begin{bmatrix} \phi_0(\mathbf{x}) = 1.0 \\ \phi_1(\mathbf{x}) = x_1 \\ \phi_2(\mathbf{x}) = x_2 \\ \phi_3(\mathbf{x}) = x_1 x_2 \\ \phi_4(\mathbf{x}) = x_1^2 \\ \phi_5(\mathbf{x}) = x_2^2 \end{bmatrix}$$

## Polynomial Regression, $d > 1$

- Case: $x \in \mathbb{R}^d, p > 1$
  1. Feature mapping
  2. Create a transformed dataset $\tilde{\mathcal{D}} = \{(\phi(\mathbf{x}_i), y_i)\}_{i=1}^n$
  3. Apply linear regression on $\phi(x)$. Now, we want to learn $f(\mathbf{x}) = \sum_{j=0}^m w_j \phi_j(\mathbf{x}) = \mathbf{w}^\intercal \phi(\mathbf{x}), \mathbf{w} \in \mathbb{R}^m,\ m = \binom{d+p}{p}$.

$$\phi(\mathbf{x}) = \begin{bmatrix} \phi_0(\mathbf{x}) = 1.0 \\ \phi_1(\mathbf{x}) = x_1 \\ \phi_2(\mathbf{x}) = x_2 \\ \vdots \\ \phi_d(\mathbf{x}) = x_d \\ \phi_{d+1}(\mathbf{x}) = x_1 x_2 \\ \phi_{d+2}(\mathbf{x}) = x_1 x_3 \\ \vdots \\ \phi_m(\mathbf{x}) = x_d^p \end{bmatrix}$$

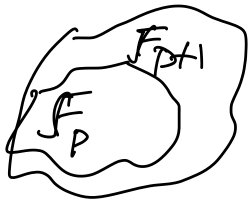
(x+1)

# coefficients , $\underline{m}$ : nb. of combinations of choosing p elements from (d+1) elements with repetitions $\binom{d+p}{p}$

## How to pick the model, ($p$)?

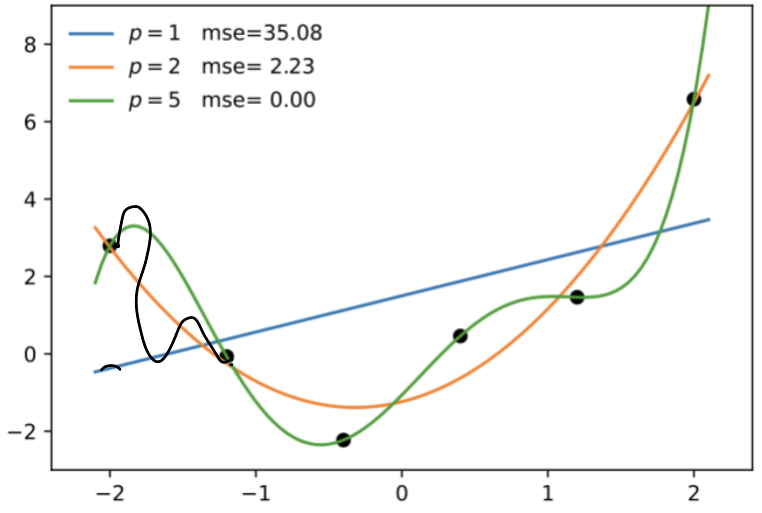
- A larger $p$ means that we have a more general function class
    - (Assume $x \in \mathbb{R}(d = 1)$.)

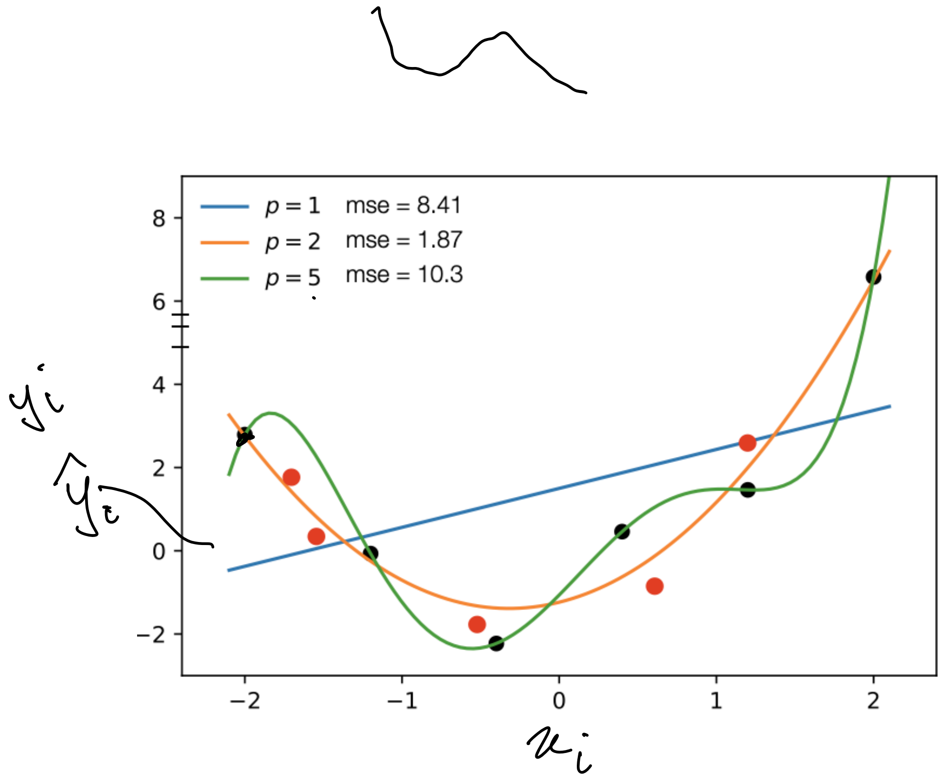
$$\mathcal{F}_p = \{f : \mathbb{R} \to \mathbb{R} \mid f(x) = \sum_{j=0}^{p} x^j w_j,\ \mathbf{w} \in \mathbb{R}^{p+1}\}$$



- $\mathcal{F}_p \subseteq \mathcal{F}_{p+1}$
- Larger $p$ is not necessarily better
    - Computationally more expensive (more parameters to estimate)
    - Overfitting

# How to pick the model, (*p*)?





- ▷ Here, polynomial regression with $p = 5$ has the best error.
- ▷ But with more data (new data, red points) $p = 2$ has lower error.

## How to compare models?

- Parametric learning
- There is an underlying distribution that generates the data points ; “true parameter" $\omega$ : this is the quantity we want to estimate.
- MLE estimator $\widehat{\mathbf{W}}_{\text{MLE}}$: this is a random variable, a function of the observed dataset.
  - for a specific instance, or realization of the dataset, ($\mathcal{D}$, an iid sample), we estimate with $\mathbf{w}_{\text{MLE}}$.
- So how do we compare between $\widehat{\mathbf{w}}_{\text{MLE}}$ for two different models?

## Generalization error

- For learning the predictor, we want to minimize the true cost:

$$\mathbb{E}[C] = \mathbb{E}[\mathrm{cost}(f(\mathbf{X}), Y)] = \int_{\mathcal{X}\times\mathcal{Y}} p(\mathbf{x}, y)\, \mathrm{cost}(f(\mathbf{x}), y)\, d\mathbf{x}\, dy$$

- This is the generalization error.
- But since we don't have access to $p(\mathbf{x}, y)$, we instead do an empirical minimization, on the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$, we do have:

$$c(w) = \frac{1}{n}\sum_{i=1}^{n} (f(\mathbf{x}_i) - y_i)^2 \,.$$

- This is the empirical error, which is a proxy for the true cost.
- Since this is computed on the dataset we learn on, this is also called *training error*

## How to compare models

- Can we use the empirical error to compare models?
- We might want to use empirical error to compare models, but this error only tells us how good a model performs on that given dataset. For a new set of data we observe, they may not have similar performance.
  - This is seen in the previous example on slide **??**. We see that while the polynomial function with $p = 5$ has the best error on the given dataset, the polynomial with $p = 2$ has better error when new data is added.
- So we can't compare using empirical error because a low empirical error may mean overfitting, if the generalization error is also not low.
- Overfitting occurs when we select a model that has good empirical error but poor generalization error.

## How to avoid overfitting?

- How to detect overfitting?
  - generalization error is high
- How do we estimate generalization error?
- We can use empirical error to estimate GE, but on a different set of iid samples (not the dataset we used to train the model).
  - If we have a set of iid samples, this is an unbiased estimator.
- Create a new set of iid samples by splitting the original dataset : keep a hold out dataset
- The observed dataset is split into
  - the non-held-out set - this is the training set, and
  - the held-out set - this is the testing set.

## Estimating Generalization error

training data : $\{(x_i, y_i)_{i=1}^{n}\}$

Heldout or test data

$\{(x_i, y_i)_{i=1}^{m}\}$

learned predictor $\hat{f}(x)$

test data to estimate GE

$$\widetilde{GE}_m = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{f}(x_{n+i}) - y_{n+i}\right)^2$$

overfitting : assumed too complex a model, optimized for training error only

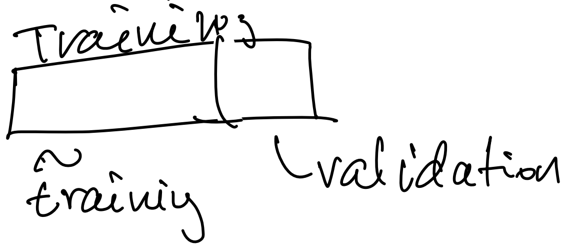
underfitting : overly simplified model

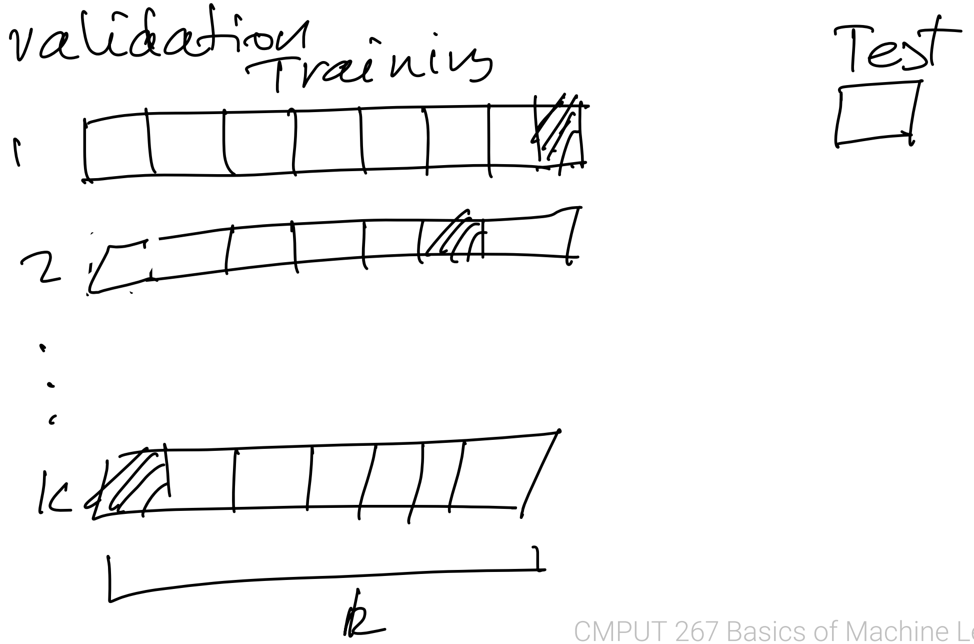
Solution : Split data, test on heldout data

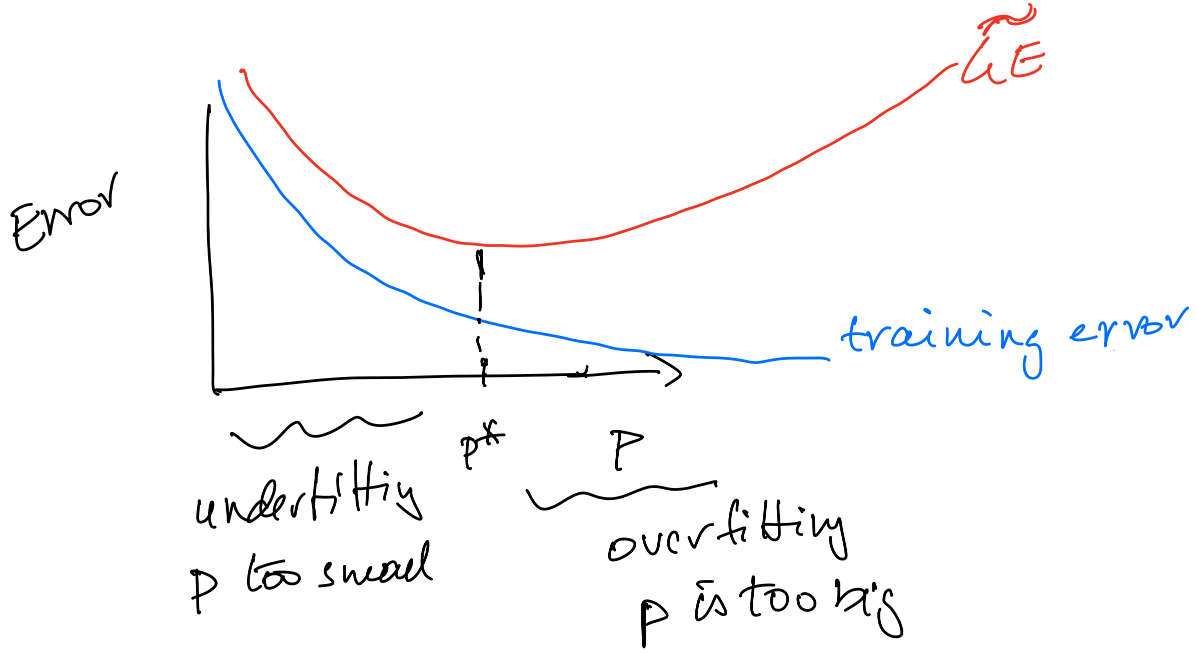
Drawbacks of splitting data

1. smaller training data
2. we can only use the test set once, but we need to optimize over values of p

→ k-fold cross validation






Error
$\widetilde{GE}$
training error
$p^*$
P
underfitting
p too small
overfitting
p is too big

Goal: avoid overfitting

Find the simplest solution that gives the best fit (low p)

How?

1. Pick lowest p, such that GE is higher for larger p

2. Prefer weights ~~be~~ zero ($w_j = 0$) (regularization)

Estimate GE with test error

$$TE_m = \frac{1}{m} \sum_{i=1}^{m} \left( \hat{f}(x_i) - y_i \right)^2$$

$\hat{f}$: learned on training set

How close $TE_m$ ~~to~~ GE

$TE_m$ is an unbiased estimate of GE

Confidence intervals $[TE_m - \varepsilon, TE_m + \varepsilon]$

How to derive CI for GE?

Split dataset into training & testing set

90/10

80/20

75/25

1. Learn $\vec{w}$ on training set $D = \{(x_i, y_i)\}_{i=1}^{n}$

2. Compute test error on the testing set

$$TE_m = \frac{1}{m} \sum_{i=1}^{m} \left( f_{\vec{w}}(x_i) - y_i \right)^2$$

3. CI $\varepsilon$, $\delta = 0.05$  $GE \in [TE_m - \varepsilon, TE_m + \varepsilon]$ w/ prob. $\delta$