CS 556 Mathematical Foundations of Machine Learning

Linear Regression

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Today’s Lecture

- Linear regression
- Optimization
- Features
- Beyond linear regression models
- Bias and variance

In Praise of Linear Models!

- Despite its simplicity, the linear model has distinct advantages in terms of its interpretability and often shows good predictive performance.
- In this lecture, we discuss (1) optimization, (2) some ways in which the simple linear model can be improved, by replacing ordinary least squares fitting with some alternative fitting procedures.

Regression Task

- Example: given the gender and height information, predict the weight.

Data sample	Gender	Height (inches)	Y: Weight (pounds)
1	Male	73	190
2	Male	67	165
3	Female	65	130

Features/Attributes (X) **Output (Y)**

$$\begin{tikzcd}\textbf{Features/Attributes (X)} \arrow[r] & \boxed{\text{Function } f} \arrow[r] & \textbf{Output (Y)}\end{tikzcd}$$

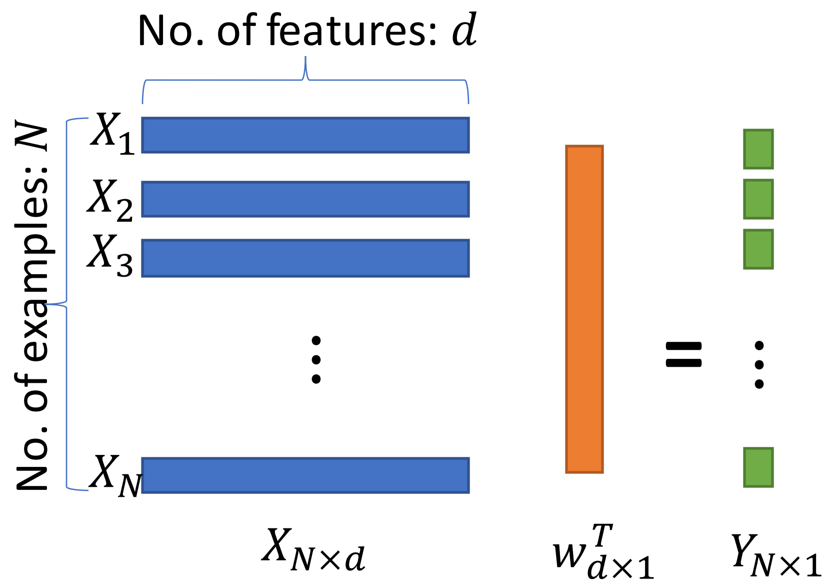
$$y \approx f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$

Linear Regression

- Input: d-dimensional $x = (x_1, x_2, \dots, x_d)$.
- Output: Real value output y.
- Regression Model:

$$y \approx f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$

where $w = (w_1, w_2, \dots, w_d)$ are parameters (weights/coefficients).



- Training Data: $(X_1, Y_1), (X_2, Y_2), \cdots, (X_N, Y_N)$ used for the estimation of the parameters w.

The Learning Problem

- Hypothesis class: Consider some restricted set F of mappings $f: X \to Y$ from input data to output.
- Estimation: Find the best estimate of mapping $\hat{f} \in F$ based on the training data.
- Evaluation: Measure how well $\hat{f}$ generalizes to the testing data (unseen examples), i.e., whether $\hat{f}(x_{new})$ agrees with y_{new}.

Estimation Criterion

- Loss function $Loss(x, y, w)$: a function of w that quantifies how well (more accurately, how badly) you would be if you use w to make a prediction on x when the correct output is y.
- Objective: the estimation problem can be solved by minimizing the loss function.
- Valid for any parameterized class of mapping from examples to predictions.
- Valid when the predictions are discrete labels or real valued as long as the loss is defined properly.

Objective of Linear Regression

- Prediction: for one data example (x, y), the prediction is:
$$f_w(x) = xw^T$$
- Residual: difference between the prediction and the target.
- Squared loss: $Loss(x, y, w) = (y - f_w(x))^2$; smaller values of loss indicates that $f_w(x)$ is a good approximation of y.
- Example:
  - $w = [2, -1], x = [2,0], y = -1, L(x, y, w) = 25$

Objective of Linear Regression

- Loss is easy to minimize if there is only one example.
- Overall training loss: on the training data D_{train} with N examples:

$$\min_{w \in R^d} TrainLoss(w) = \min_{w \in R^d} \frac{1}{N} \sum_{(x,y) \in D_{train}} L(x,y,w)$$

- Key: learn the parameters to make the global tradeoffs.
- If $Loss(x,y,w) = (y - f_w(x))^2$, the residual sum of squares (RSS) is defined as $RSS = \sum_{(x,y) \in D_{train}} L(x,y,w)$.

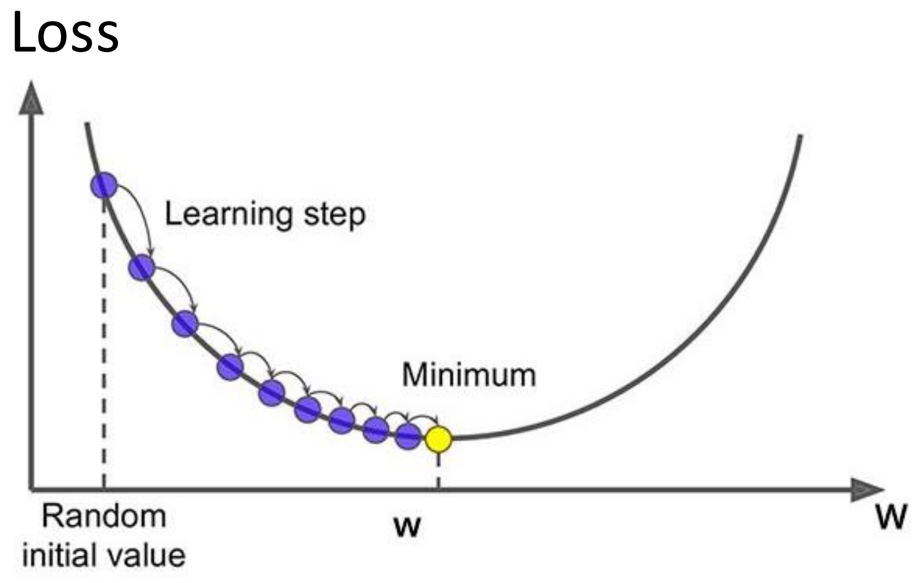
Regression Loss Functions

- Squared loss: $L_2(x, y, w) = (y - f_w(x))^2$
  - The overall loss will be mean squared error (MSE)
  - Root mean squared error (RMSE): $RMSE = \sqrt{MSE}$

- Absolute loss: $L_1(x, y, w) = |y - f_w(x)|$
  - The overall loss will be mean absolute error (MAE)

How to Optimize: Gradient Descent

- A first-order iterative optimization algorithm for finding the local minimum of a differentiable function.
- To get the parameter that minimizes the objective function, we iteratively move in the opposite direction of the gradient of the function.
- The size of each step is determined by a parameter which is called Learning Rate.



How to Optimize: Gradient Descent

- Objective: $TrainLoss(w) = \min_{w \in R^d} \frac{1}{N} \sum_{(x,y) \in D_{train}} (y - xw^T)^2$
- Gradient with respect to w: the direction that increases the loss the most.

$$\nabla_w TrainLoss(w) = \frac{1}{N} \sum_{(x,y) \in D_{train}} 2(y - xw^T)\, x$$

- Gradient descent update:

$$w \leftarrow w - \underbrace{\boxed{\eta}}_{\text{Step size}} * \underbrace{\boxed{\nabla_w TrainLoss(w)}}_{\text{Gradient}}$$

Gradient Descent

```
1: procedure BATCH GRADIENT DESCENT
2:     for $i$ in range(epochs) do
3:         $g^{(i)}(\mathrm{w}) = \text{evaluate\_gradient}(\text{TrainLoss, data, w})$
4:         $\mathrm{w} = \mathrm{w} - \text{learning\_rate} * g^{(i)}(\mathrm{w})$
```

Some Limitations:

- Each iteration requires going over all training examples (Batch gradient descent) – expensive and slow when have lots of data.
- Intractable for datasets that don't fit in memory.
- Guaranteed to converge to the global minimum for convex functions, but may end up at a local minimum for non-convex functions.

Stochastic Gradient Descent (SGD)

- Considers only one point at a time to update weights.
- Not calculate the total error for whole data in one step, instead we calculate the error of each point and use it to update weights.
- Gradient descent update:

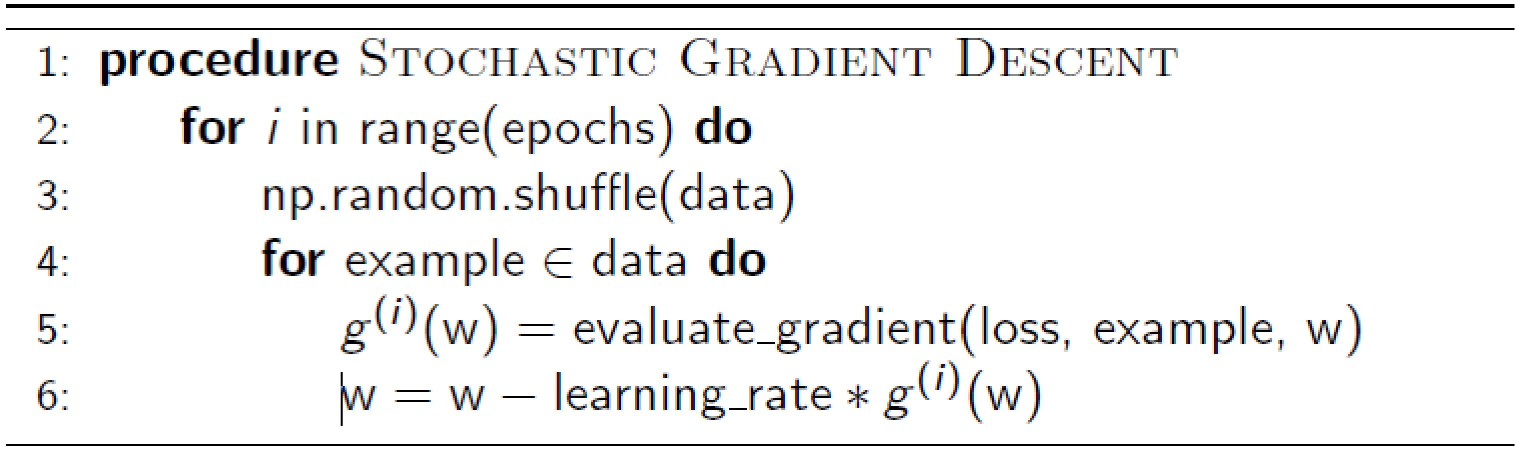
$$w \leftarrow w - \underbrace{\boxed{\eta}}_{\text{Step size}} * \underbrace{\boxed{\nabla_w TrainLoss(w)}}_{\text{Gradient}}$$

- Stochastic Gradient Descent:

  For each $(x, y) \in D_{train}$,

$$w \leftarrow w - \eta \nabla_w Loss(x, y, w)$$

Stochastic Gradient Descent (SGD)



```
procedure Stochastic Gradient Descent
    for $i$ in range(epochs) do
        np.random.shuffle(data)
        for example $\in$ data do
            $g^{(i)}(w)$ = evaluate_gradient(loss, example, w)
            $w = w -$ learning_rate $* g^{(i)}(w)$
```

- Shuffling: to ensure that each data point creates an "independent" change on the model, without being biased by the same points before them.
- Easy to fit in memory as only one data point needs to be processed at a time, thus, computationally less expensive.
- With a high variance that cause the objective function to fluctuate heavily.
- May never reach local minima and oscillate around it due to the fluctuations in each step.

Mini-batch Gradient Descent

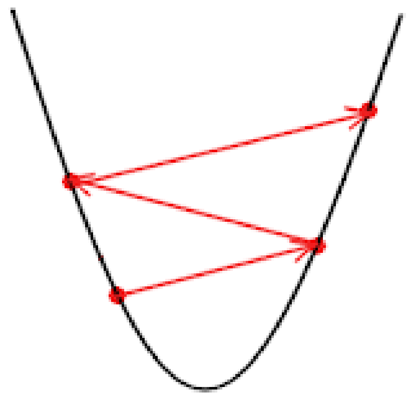
```
procedure MINI-BATCH GRADIENT DESCENT
    for $i$ in range(epochs) do
        np.random.shuffle(data)
        for batch $\in$ get_batches(data, batch_size=50) do
            $g^{(i)}(\mathrm{w})$ = evaluate_gradient(loss, batch, w)
            $\mathrm{w} = \mathrm{w} -$ learning_rate $* \, g^{(i)}(\mathrm{w})$
```

- Instead of using the whole data for calculating gradient, we use only a mini-batch of it (batch size is a hyperparameter).
- Reduces the variance of the parameter updates, which can lead to more stable convergence.
- Can make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient.

Learning Rate

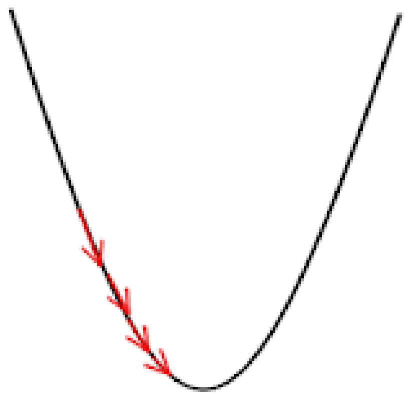
Choosing an appropriate value of learning rate is important because it helps in determining how much we have to descent in each iteration.

High Learning Rate



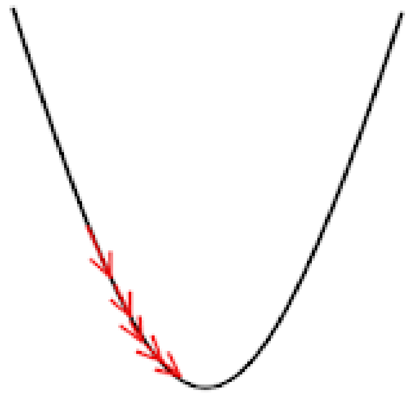
Causes drastic updates and leading to divergent behavior

Just Right



Swiftly reaches the minimum point

Too small



Requires many updates before reaching the minimum points

https://medium.com/analytics-vidhya/gradient-descent-why-and-how-e369950ae7d3

Summary so far

- Linear predictors:

$$f(x) = \sum_{i=1}^{d} w_i x_i = x w^T$$

- Loss minimization: learning as optimization

$$\min_{w \in R^d} TrainLoss(w)$$

- Optimization algorithm: gradient decent or other variants

$$w \leftarrow w - \eta * \nabla_w TrainLoss(w)$$

Today’s Lecture

- Linear regression
- Optimization
- **Features**
- Beyond linear regression models
- Bias and variance

Input Features

- The input features x_i can come from different sources:
  - Quantitative inputs
  - Transformations of quantitative inputs, such as log and square-root.
  - Basis expansions, such as $x_2 = x_1^2$ leading to a polynomial representation.
  - Numeric or ‘dummy’ coding of the levels of qualitative inputs.
    - If G is a five-level factor input, we can create $x_i, i = 1, \dots, 5$ such that $x_i = I(G = i)$.
    - This group of features represent the effect of G by a set of level-dependent constraints, since only one x_i is 1, and the others are 0.
  - Interactions between features, for example: $x_3 = x_1 x_2$.

Feature Extraction

- Example task: predict the weight of a person
- Question: what properties of a person might be relevant for predicting the weight y?
- Feature extractor: given input x, output a set of (feature name, feature value) pairs.

$$\begin{bmatrix} Gender\text{:} & Height\text{:} & Age\text{:} \\ Female & 165\ (cm) & 25 \end{bmatrix}$$

Feature Vector Representation

- Mathematically, feature vector doesn't need feature names.

$$\begin{bmatrix} Gender: & Height: & Age: \\ Female & 165\ (cm) & 25 \end{bmatrix} \rightarrow [0 \quad 165 \quad 25]$$

- Feature vector representation: For input x, x is a point in the high-dimensional space :
    - Array representation: good for dense features

$$x = [x_1, x_2, \dots, x_d]$$

    - Map representation: good for sparse features

$$x = \{\text{Gender}: 0, Height: 165, Age: 25\}$$

Linear Predictors

Weight vector $w \in R^d$

$$\begin{bmatrix} Gender & height & age \\ 1.5 & 0.8 & 1.1 \end{bmatrix}$$

Feature vector $x \in R^d$

$$\begin{bmatrix} Gender & height & age \\ 0 & 165 & 25 \end{bmatrix}$$

Predicting score:

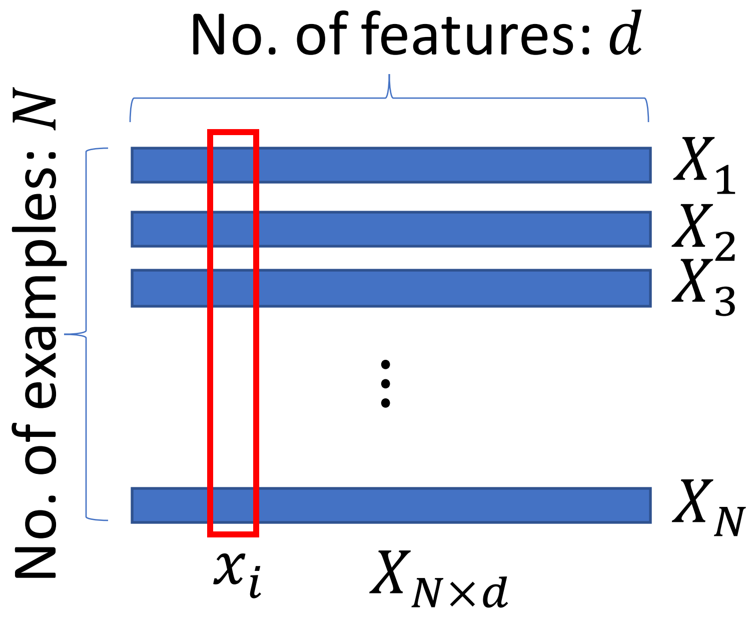
$$f(x) = \sum_{i=1}^{d} w_i x_i = xw^T$$

Feature Scaling

- Data may include features with varying magnitudes.

$$\begin{bmatrix} Gender: & Height: & Age: \\ Female & 165\ (cm) & 25 \end{bmatrix}$$

- ML algorithms may assign higher weight to greater values regardless of the unit of the values (e.g. 165 cm > 1.95 m), and hence provide wrong prediction.
- Normalization: each feature contributes approximately proportionally.



Feature Scaling

- Gradient descent converges much faster with feature scaling than without it.

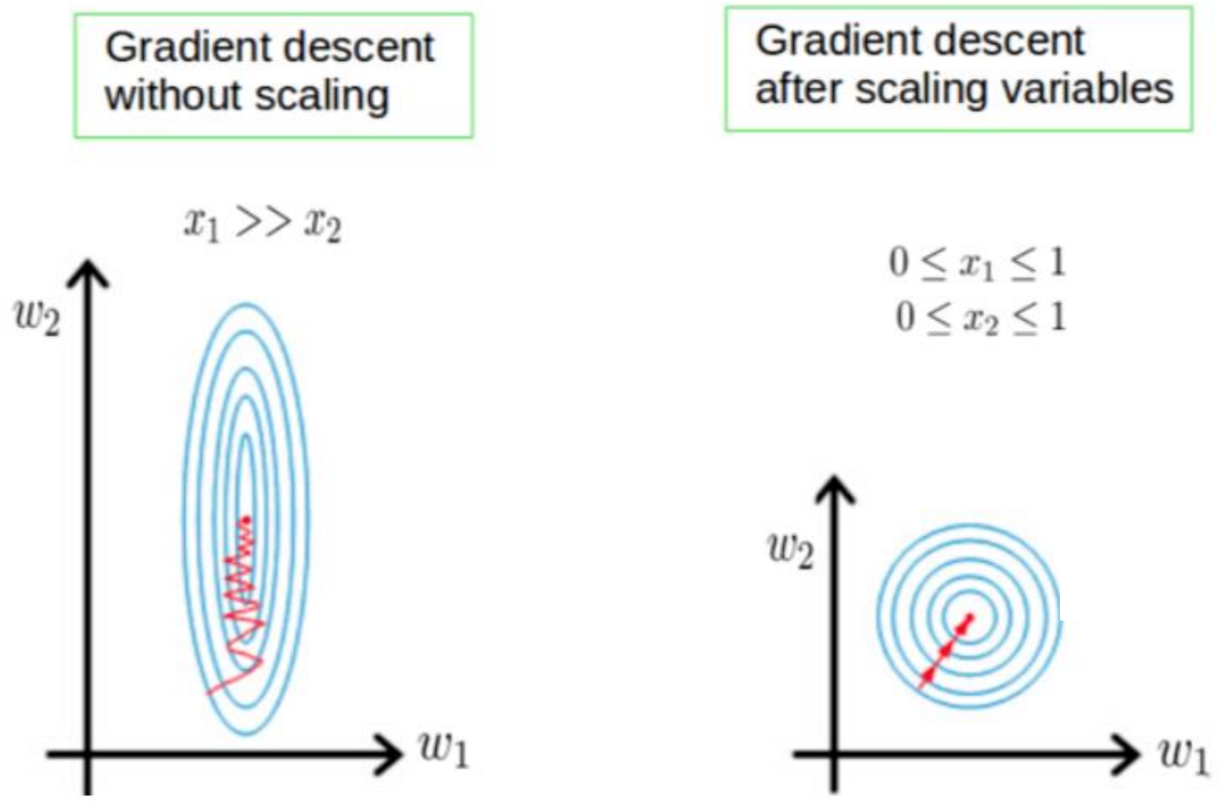
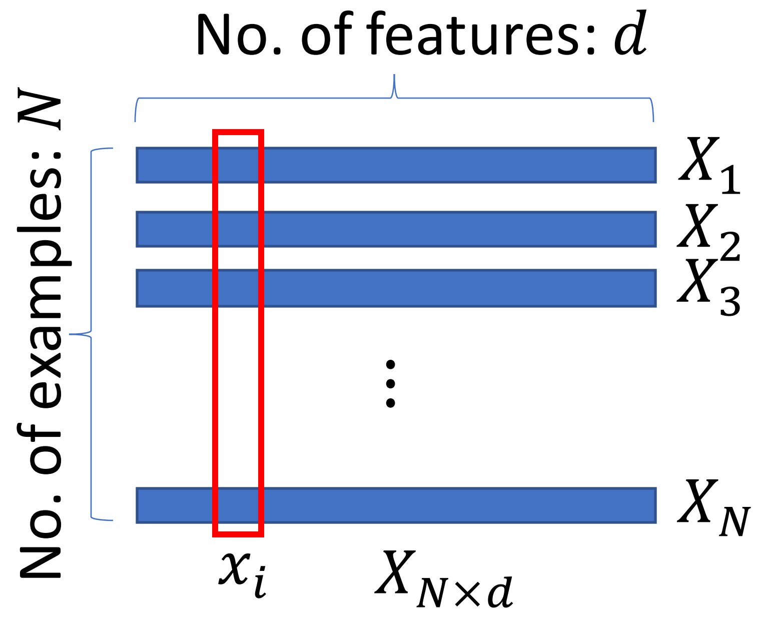

Gradient descent without scaling
$x_1 >> x_2$
w_2
w_1
Gradient descent after scaling variables
$0 \leq x_1 \leq 1$
$0 \leq x_2 \leq 1$
w_2
w_1

Figure: https://medium.com/analytics-vidhya/feature-scaling-normalization-standardization-and-scaling-c920ed3637e7

Feature Scaling: Standardization

- Z-score normalization: transform the data such that the resulting data has a mean of 0 and a standard deviation of 1.
- For each feature x_i, calculate the mean m_i, and the standard deviation σ_i.
- Scaling features:

$$x_i \leftarrow \frac{x_i - m_i}{\sigma_i}$$



Feature Scaling: Min-Max Scaling

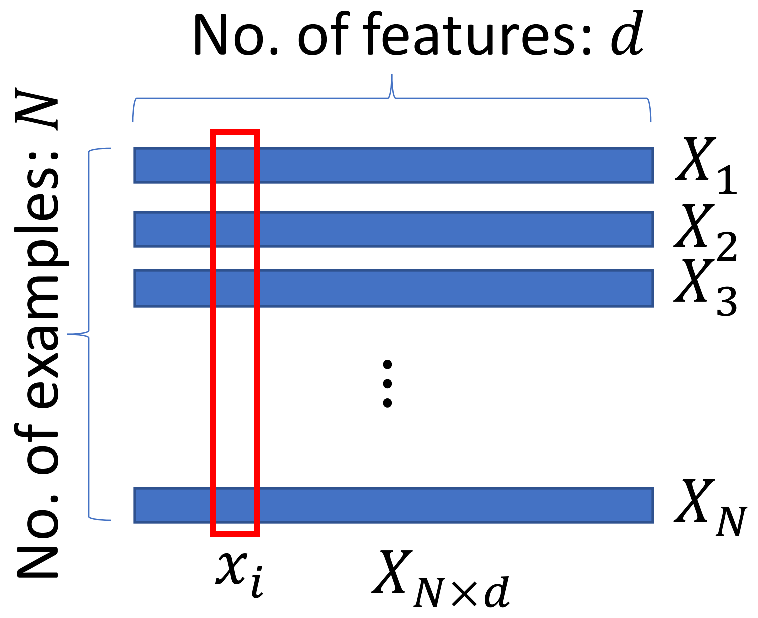
- Transform the data to $[x_{min}^{new}, x_{max}^{new}]$.
- Scaling features:

$$x_i \leftarrow \frac{x_i - x_{min}}{x_{max} - x_{min}}(x_{max}^{new} - x_{min}^{new}) + x_{min}^{new}$$

- If the new range is $[0,1]$,

$$x_i \leftarrow \frac{x_i - x_{min}}{x_{max} - x_{min}}$$

- Example: If the weight of the students is in the range of [100, 150], after normalization to [0,1], the weight 120 is mapped to $\frac{120-100}{150-100} = 0.4$



Closed-form Solution

- Learning objective with $(X_{N\times d}, Y_{N\times 1}, w^T_{d\times 1})$

$$\min_{w\in R^d} \frac{1}{N} \sum_{i=1}^{N} (Y_i - X_i w^T)^2 = \min\ (Y - Xw^T)^T (Y - Xw^T)$$

- Differentiating w.r.t w:

$$X^T (Y - Xw^T) = 0$$

- Setting the gradient to zero, we get closed form of estimates:

$$w = (X^T X)^{-1} X^T Y$$

- The resulting prediction errors $\varepsilon_i = Y_i - X_i w^T$ are uncorrelated with any linear function of the inputs X.
- Easily extend to non-linear functions of the inputs.

Beyond Linear Regression

- Prediction driven by score: $f(x) = xw^T$
- Score is linear function of w, which permits efficient learning.
- Predictors can be expressive non-linear functions and decision boundaries of x.
- Example: m^{th} order polynomial regression on one-dimensional input where $f: R \rightarrow R$ is given by:

$$f(w) = w_0 + w_1 x + \cdots + w_{m-1} x^{m-1} + w_m x^m$$

Polynomial Regression

- Polynomial regression is a special case of linear regression.
- Problem:

$$f(w) = w_0 + w_1 x + \cdots + w_{m-1} x^{m-1} + w_m x^m$$

- Solution:

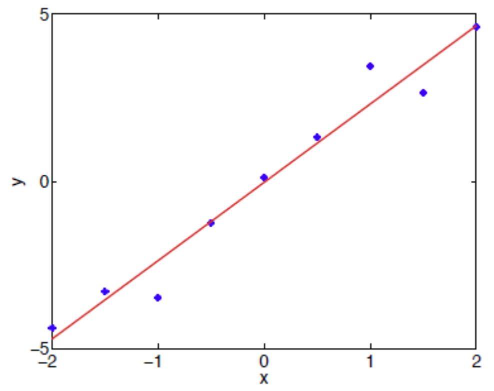
$$w = (X^T X)^{-1} X^T Y$$

- Where

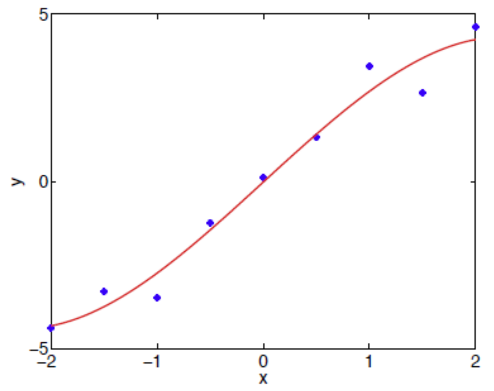
$$w^T = \begin{pmatrix} w_0 \\ w_1 \\ \dots \\ w_m \end{pmatrix} \quad X = \begin{pmatrix} 1 & x_1 & x_1^2 & \dots & x_1^m \\ 1 & x_2 & x_2^2 & \dots & x_2^m \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_N & x_N^2 & \dots & x_N^m \end{pmatrix}$$

Polynomial Regression

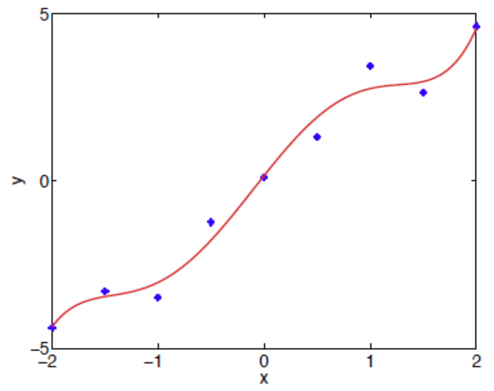
- Can be generally used when linear regression fails.
- Does not require the linear relationship between x and y.
- Predicts the best fit line that follows the pattern (curve) of the data.
- When the degree increases, the performance tend to increase, however, it also increases the risk of overfitting and underfitting.



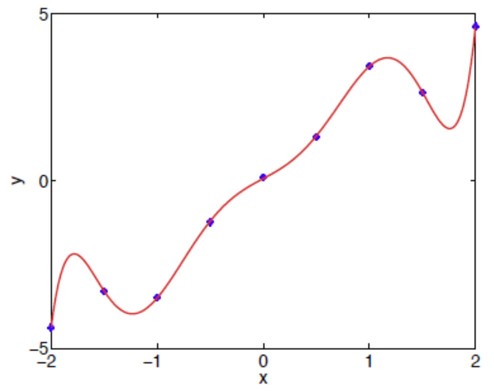
degree $= 1$



degree $= 3$



degree $= 5$

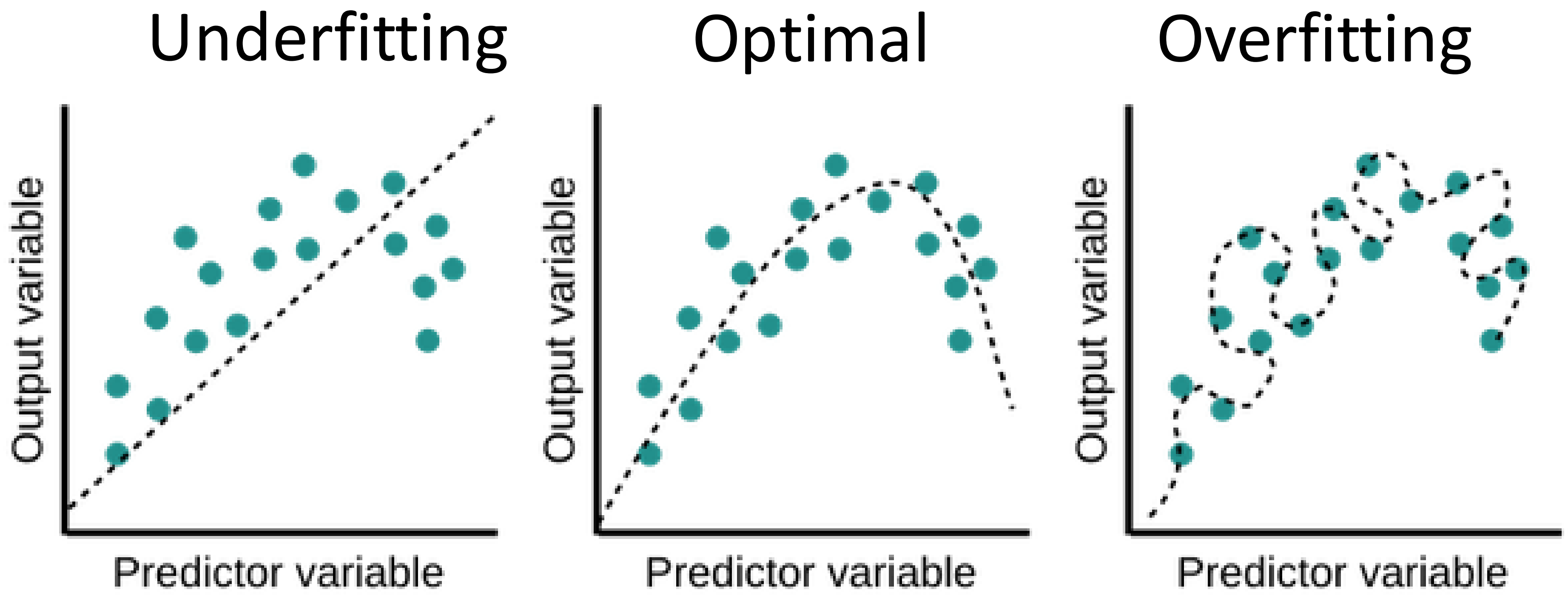


degree $= 7$

Underfitting and Overfitting

	Training Accuracy	Testing Accuracy	Possible Reason
Underfitting	0.24	0.25	Model is too simple: • Low dimensional • Heavily regularized • Bad assumption
Overfitting	0.95	0.27	Model is too complex: • High dimensional or non-parametric • Weakly regularized • Not enough data

Underfitting and Overfitting



https://www.educative.io/edpresso/overfitting-and-underfitting

Bias and Variance

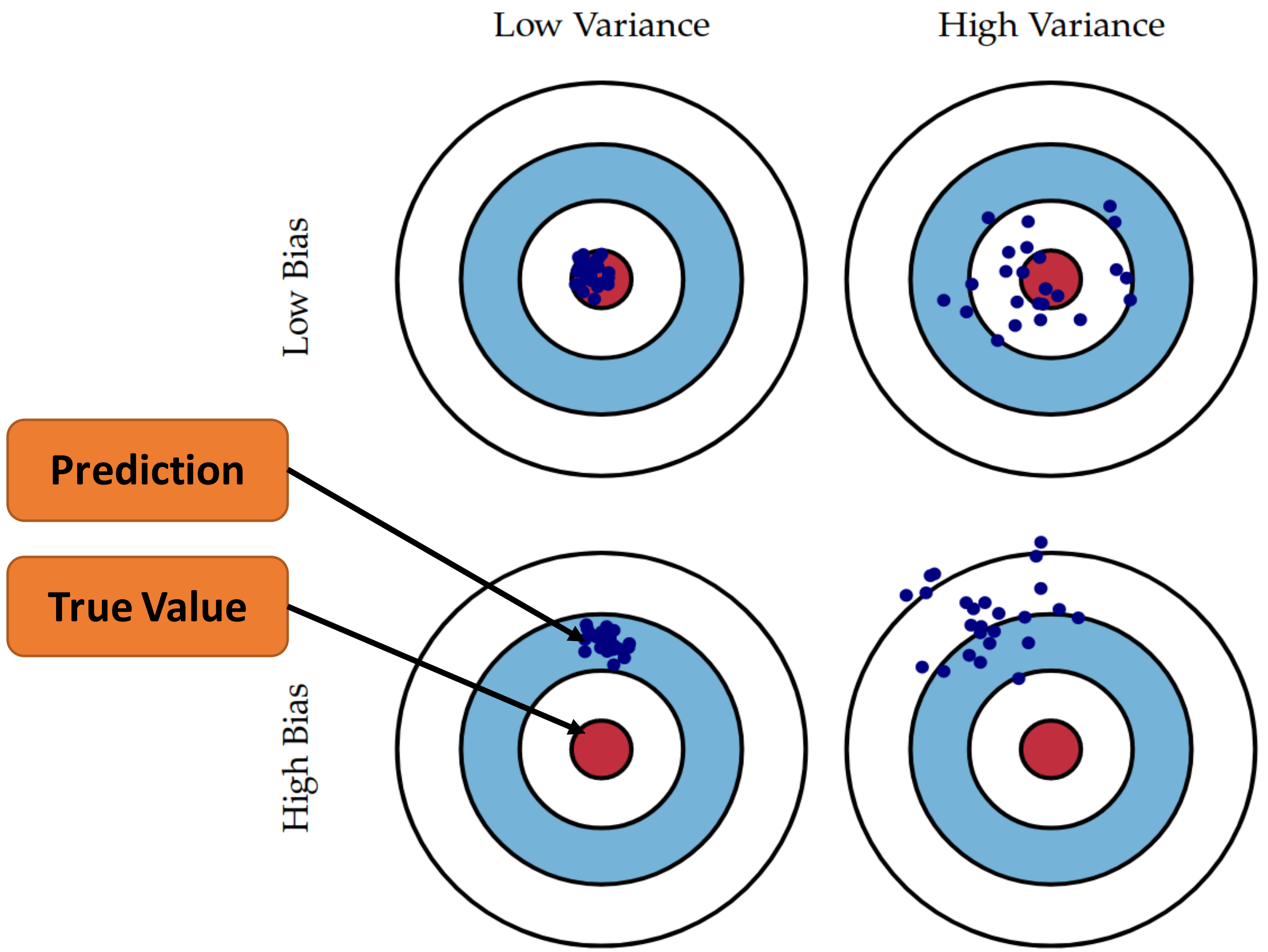
- Assuming we are predicting Y based on the features X. There is a function $Y = f(X) + \varepsilon$ where the error term is normally distributed with mean 0 and variance σ^2.
- The goal for us is to find a function $\hat{f}(X)$ that approximates the true function $f(X)$.
- Expected squared prediction error at a point X will be:

$$E\left[\left(Y - \hat{f}(X)\right)^2\right] = \left(\boxed{E\left[\hat{f}(X)\right] - f(X)}\right)^2 + \boxed{\left(E\left[\hat{f}(X)^2\right] - E^2\left[\hat{f}(X)\right]\right)} + \sigma^2$$

$$= \quad Bias^2 \quad + \quad Variance + \sigma^2$$

- Bias: Error from incorrect modeling assumption.
- Variance: Error from random noise.

Bias and Variance Tradeoff



https://towardsdatascience.com/bias-and-variance-in-linear-models-e772546e0c30

Solution for Large Bias

- If your model cannot even fit the training examples, then you have large bias (underfitting)
- If you can fit the training data, but large error on testing data, then you probably have large variance (overfitting)
- Solution: redesign your model
  - Add more features as input
  - A more complex model

Solution for Large Variance

- The least squares estimates often have low bias but large variance.
- It is possible to trade in a little bias for a larger reduction in variance leading to a smaller MSE than least square.
- Solution:
  - More data (very effective, but not always practical)
  - Feature subset selection
  - Regularized regression

With a large number of features, we often would like to determine a smaller subset that exhibit the strongest effects.

Stepwise Selection

- Forward-stepwise Search
  - Start with no features
  - Greedily include the most relevant feature
  - Stop when selected the desired number of features
- Backward-stepwise Search
  - Start with all features
  - Greedily remove the least relevant feature
  - Stop when selected the desired number of features
- Inclusion/Removal criteria uses cross-validation

Regularized Regression

- Subset selection, in which variables are either retained or discarded, is a discrete process. It often exhibits high variance, and so doesn't reduce the prediction error of the full model.
- Instead of directly minimize the MSE, the regularized regression usually take the form:

$$\min_{w \in R^d} \frac{1}{N} \sum_{(x,y) \in D_{train}} \left(y - xw^T\right)^2 + \lambda P(w)$$

Where $P(w)$ is the penalized term on the smoothness or complexity of the function w.

Regularization

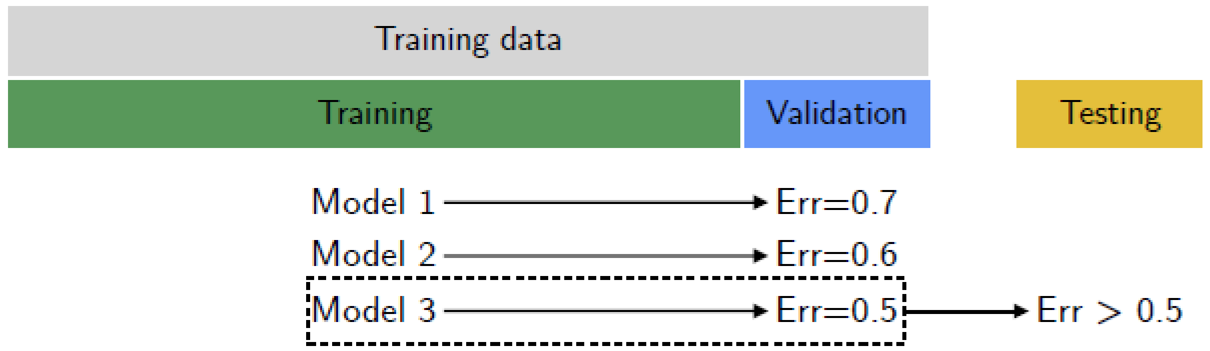
- L_1 regularization: Lasso regression
$$L_1(X,Y,w) = \sum_{i=1}^{N} (f_w(X_i,w) - Y_i)^2 + \lambda |w|$$
- L_2 regularization: Ridge regression:
$$L_2(X,Y,w) = \sum_{i=1}^{N} (f_w(X_i,w) - Y_i)^2 + \lambda w^T w$$

- The tuning parameter λ serves to control the relative impact of these two terms on the regression coefficient estimates.

- Selecting a good value for λ is critical; cross-validation is used for this.

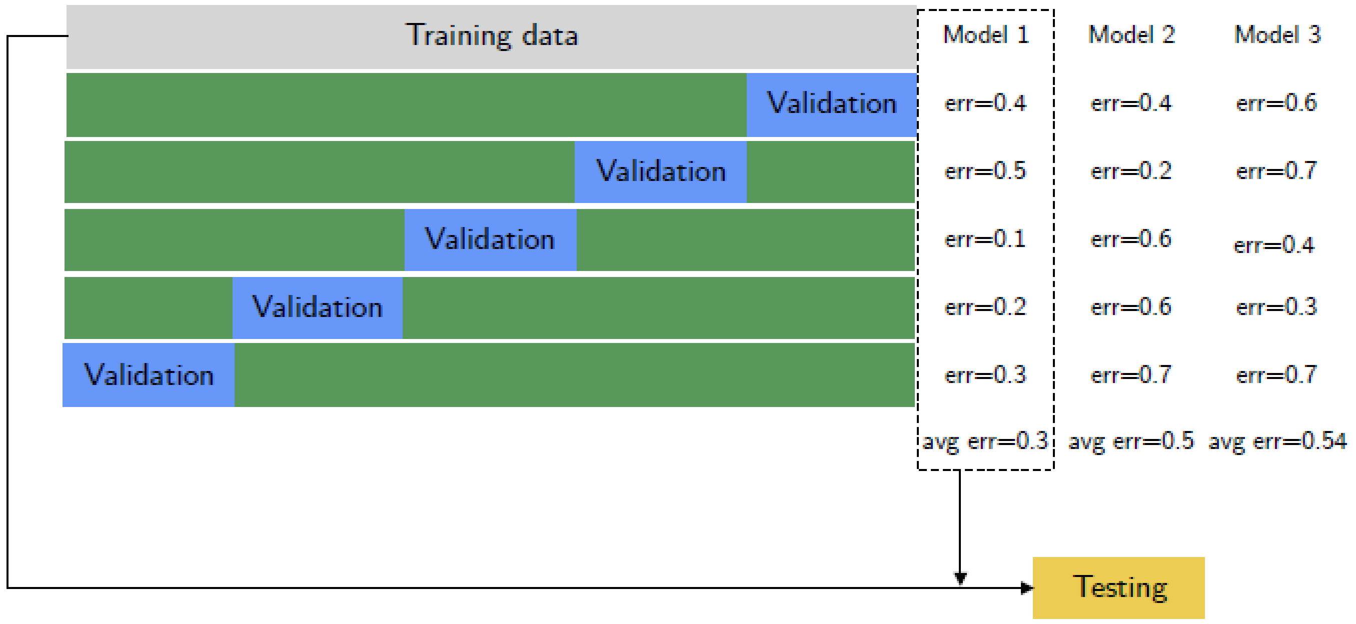
Model Selection

- Usually a trade-off between bias and variance.
- We require a method to determine which of the models under consideration is best and can balance two kinds of error to minimize total error.
- Cross-validation: choose a grid of values for the parameters and compute the cross-validation error rate for each value.
- We then select the tuning parameter value for which the cross-validation error is smallest.



Training vs. Validation vs. Testing

k-fold Cross Validation



Model 1	Model 2	Model 3
err=0.4	err=0.4	err=0.6
err=0.5	err=0.2	err=0.7
err=0.1	err=0.6	err=0.4
err=0.2	err=0.6	err=0.3
err=0.3	err=0.7	err=0.7
avg err=0.3	avg err=0.5	avg err=0.54

- In general: we perform k runs. Each run, it allows to use $\frac{k-1}{k}$ of the available data for training.
- If the number of data is very limited, we can set $k = N$ (total number of data points). This gives the leave-one-out cross-validation technique.

Summary of Today’s Lecture

- Linear regression
- Optimization
- Features
- Beyond linear regression models
- Bias and variance

Slides

- Courtesy of Ping Wang.