CS 559 Machine Learning
Lecture 2: Linear Regression

Ping Wang
Department of Computer Science
Stevens Institute of Technology

Today's Lecture

- Linear regression
- Optimization
- Features
- Beyond linear regression models
- Bias and variance
- The curse of dimensionality
- Maximum likelihood estimator

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)**

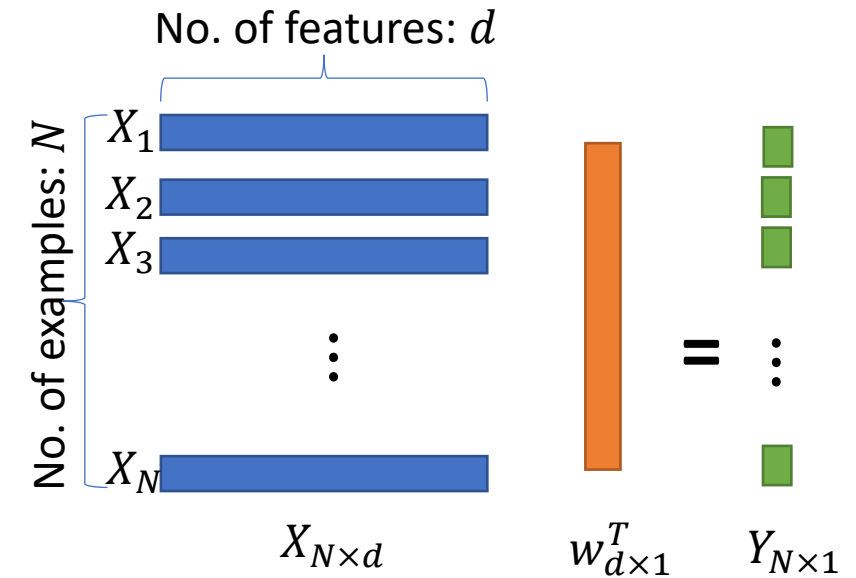**Features/Attributes (X)** $\longrightarrow$ Function f $\longrightarrow$ **Output (Y)**

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

Linear Regression

- Input: d-dimensional $x = (x_1, x_2, \ldots, x_d)$.

- Output: Real value output y.

- Regression Model:

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

No. of features: d

No. of examples: N

X_1
X_2
X_3

$\vdots$

X_N

$X_{N \times d}$ $w_{d \times 1}^T$ $Y_{N \times 1}$

where $w = (w_1, w_2, \ldots, w_d)$ 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 $f: X \rightarrow Y$ 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)

Empirical Risk Minimization

➢Assume each training and test example-label pair (x, y) is drawn independently at random from the same but unknown population of examples and labels.

➢We can represent this population as a joint probability distribution $P(x, y)$ so that each training/testing example is a sample from this distribution $(x_i, y_i) \sim P$.

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

$$Expected\ (testing)\ loss = E_{(x,y) \sim P} L(x, y)$$

➢The training loss based on a few sampled examples and labels serves as a proxy for the test performance measured over the whole population.

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.

Loss

Learning step

Minimum

Random
initial value

w

W
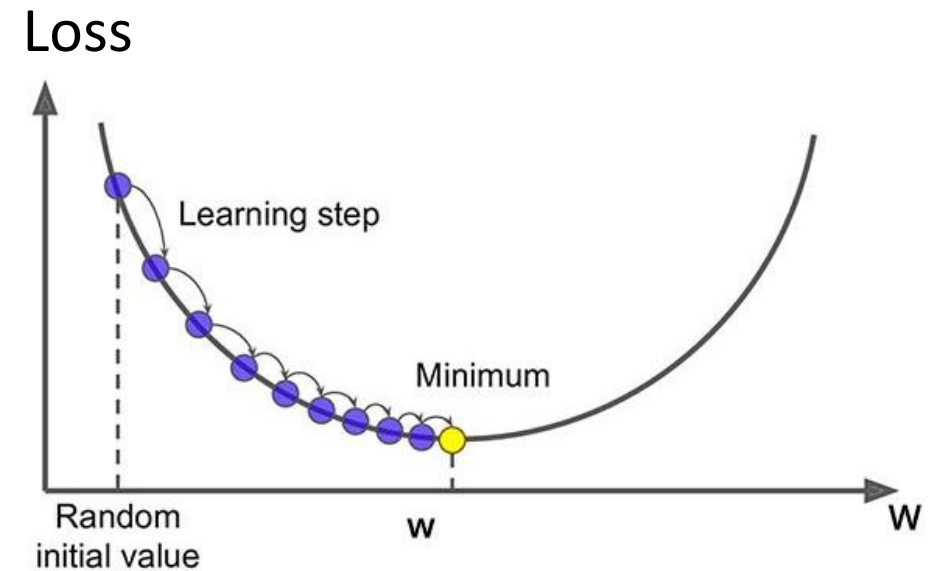
How to Optimize: Gradient Descent

- 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:

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

Step size   Gradient

Gradient Descent

```
1: procedure BATCH GRADIENT DESCENT
2:     for i in range(epochs) do
3:         g^(i)(w) = evaluate_gradient(TrainLoss, data, w)
4:         w = w − learning_rate * g^(i)(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:

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

Step size   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)

```
1:  procedure STOCHASTIC GRADIENT DESCENT
2:      for i in range(epochs) do
3:              np.random.shuffle(data)
4:          for example ∈ data do
5:                  g^(i)(w) = evaluate_gradient(loss, example, w)
6:                  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.
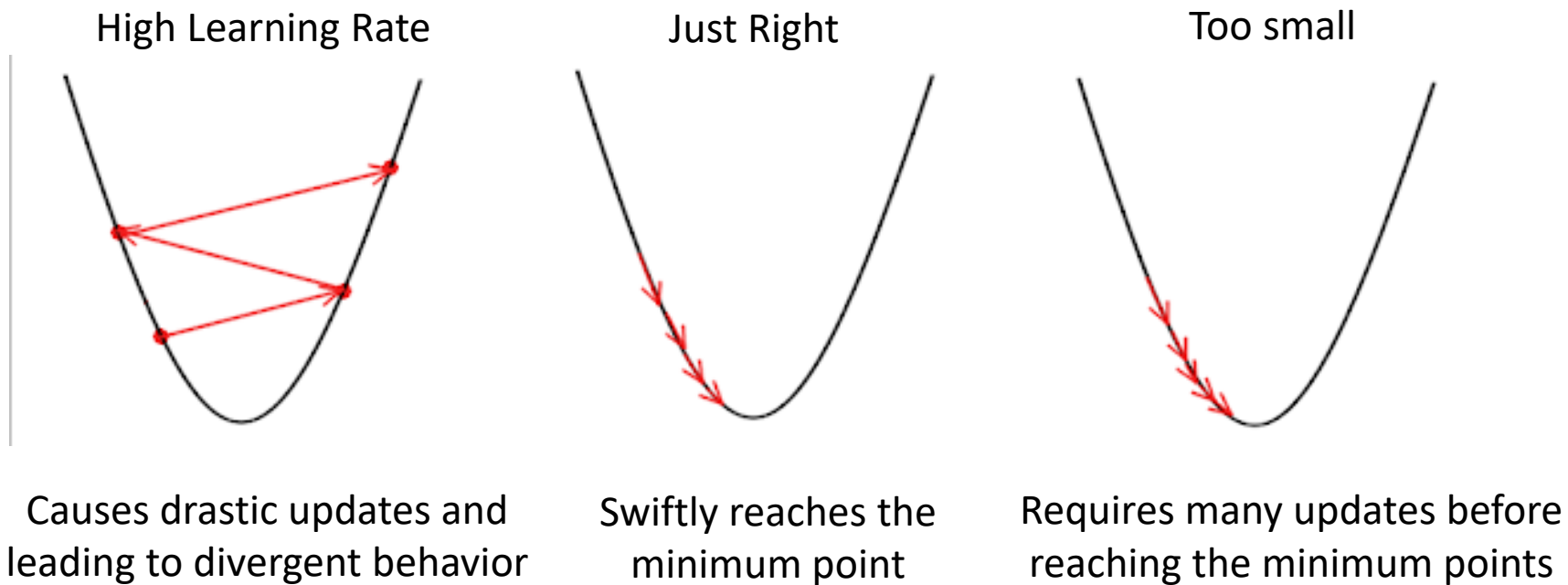
Mini-batch Gradient Descent

```
1:  procedure MINI-BATCH GRADIENT DESCENT
2:      for i in range(epochs) do
3:          np.random.shuffle(data)
4:          for batch ∈ get_batches(data, batch_size=50) do
5:              g^(i)(w) = evaluate_gradient(loss, batch, w)
6:              w = w − learning_rate * g^(i)(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.
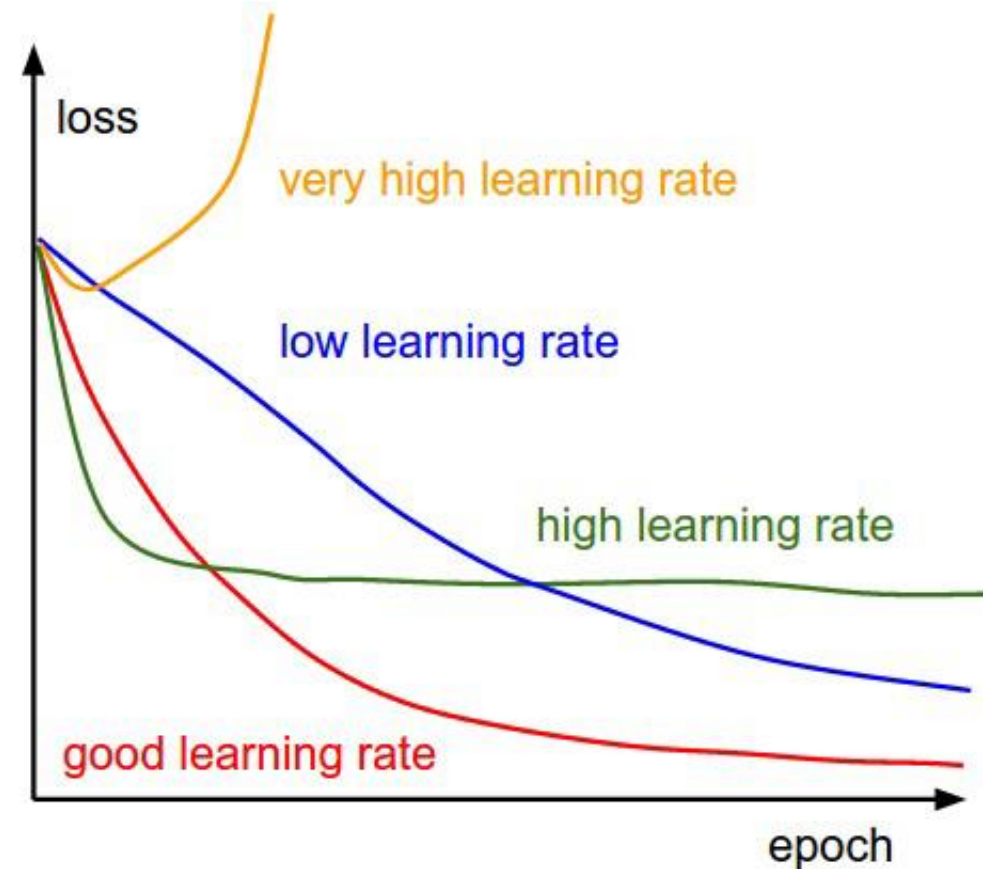
Learning Rate

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   Just Right   Too small

Causes drastic updates and leading to divergent behavior Swiftly reaches the minimum point Requires many updates before reaching the minimum points

Learning Rate

Set the learning rate carefully:

- If there are more than 3 parameters, it is hard to visualize the loss w.r.t the parameters.

- But you can visualize the loss w.r.t the # of parameter updates (epoch).
  - If the loss increases, we need to decrease the value of the learning rate.
  - If the loss is decreasing at a very slow rate, we need to increase the value of the learning rate.
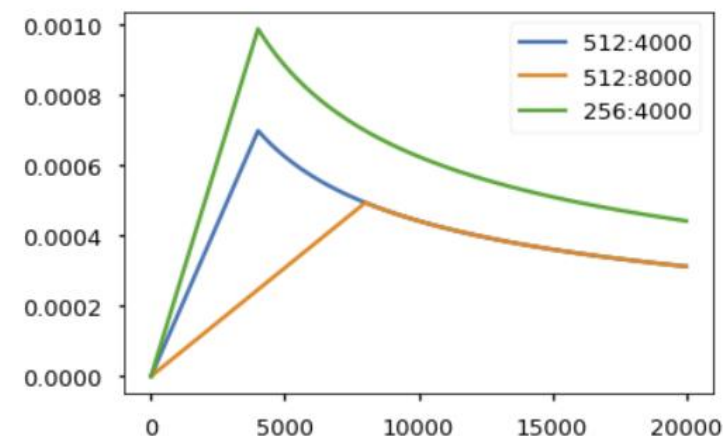
Learning Rate

- Gradient update:

$$w \leftarrow w - \boxed{\eta} * \boxed{\nabla_w TrainLoss(w)}$$

Step size   Gradient

- Question: How to choose the value of step size (learning rate)?

- Strategies: For each $(x, y) \in D_{train}$,
  - Constant: $\eta = 0.1$
  - Adaptive learning rates
    - Decreasing
    - Warm-up

Adaptive Learning Rates

➢Popular & Simple Idea: Reduce the learning rate by some factor every few epochs.

- At the beginning, we are far from the destination, so we use larger learning rate
- After several epochs, we are close to the destination, so we reduce the learning rate
- For example: $\frac{1}{t}$ decay, $\eta^{(t)} = \frac{\eta}{\sqrt{t+1}}$

➢Learning rate cannot be one-size-fits-all
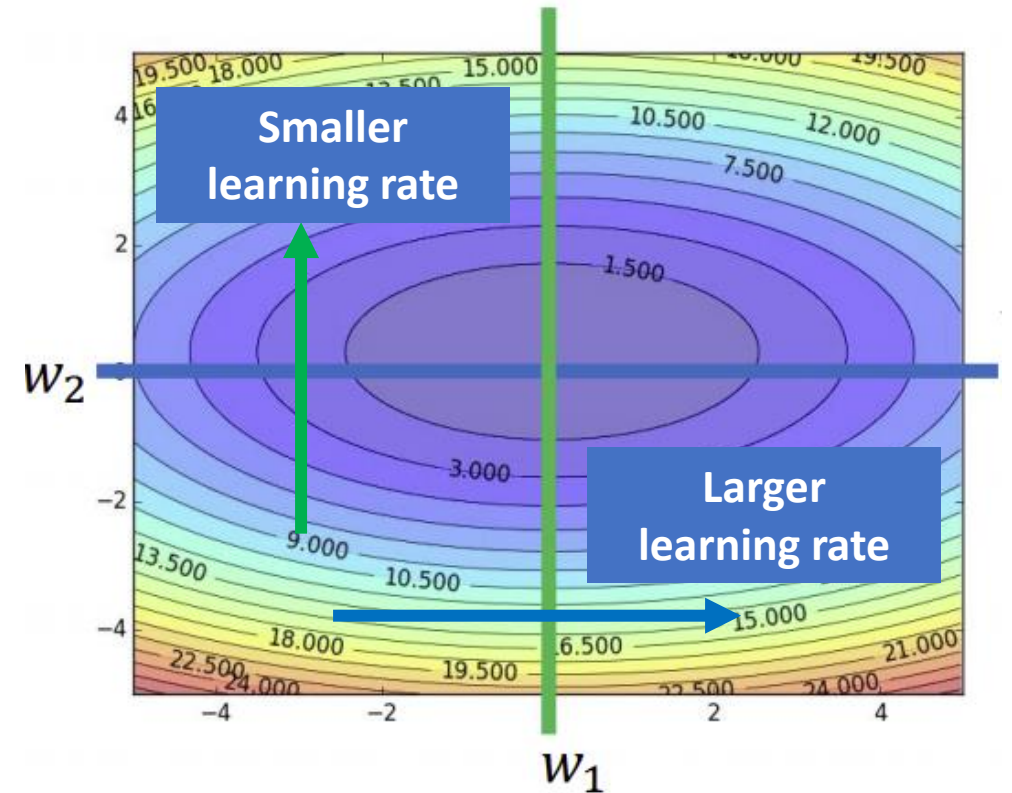
- Giving different parameters different learning rates

Adagrad

- Divide the learning rate of each parameter by the root mean square of its previous derivatives $\sigma^{(t)}$.

- Moves uniformly in all the directions by adding an element-wise scaling of the gradient.

$$\eta^{(t)} = \frac{\eta}{\sqrt{t+1}}$$

$$g^{(t)} = \frac{\partial L(x, y, w^{(t)})}{\partial w^{(t)}}$$

$$w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta^{(t)}}{\sigma^{(t)}} g^{(t)}$$

$$\text{with } \sigma^{(t)} = \sqrt{\frac{1}{t+1} \sum_{i=1}^{t} (g^{(i)})^2 + \varepsilon}$$

Small gradient, larger learning rate
Large gradient, smaller learning rate

Adagrad

$$w^{(1)} \leftarrow w^{(0)} - \frac{\eta^{(0)}}{\sigma^{(0)}} g^{(0)}$$

$$w^{(2)} \leftarrow w^{(1)} - \frac{\eta^{(1)}}{\sigma^{(1)}} g^{(1)}$$

$$w^{(3)} \leftarrow w^{(2)} - \frac{\eta^{(2)}}{\sigma^{(2)}} g^{(2)}$$

...

$$w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta^{(t)}}{\sigma^{(t)}} g^{(t)}$$

$$\sigma^{(0)} = \sqrt{(g^{(0)})^2 + \varepsilon}$$

$$\sigma^{(1)} = \sqrt{\frac{1}{2}(g^{(0)} + g^{(1)})^2 + \varepsilon}$$

$$\sigma^{(2)} = \sqrt{\frac{1}{3}(g^{(0)} + g^{(1)} + g^{(2)})^2 + \varepsilon}$$

...

$$\sigma^{(t)} = \sqrt{\frac{1}{t+1}\sum_{i=1}^{t}(g^{(i)})^2 + \varepsilon}$$

Where ε is a smoothing term that avoids division by zero (usually on the order of $1e - 8$).

Warm-up Learning Rate

- **Motivation**: At the beginning of training, the weights of the model are randomly initialized. If you choose a larger learning rate, it may lead to instability (oscillation) of the model.

- **Solution**: At the beginning of training, it uses a small learning rate to train some epochs or steps, and then modifies it to the preset learning for training.

- The model can gradually become stable, which makes the convergence speed of the model become faster and the model effect is better.
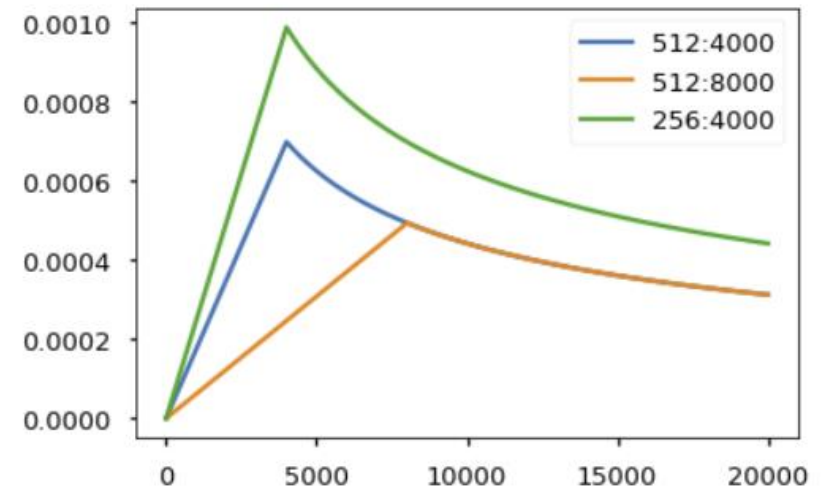
Summary so far

- 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
- The curse of dimensionality
- Maximum likelihood estimator

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.

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

Feature Vector Representation

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

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

- Feature vector representation: For input x, x is a point in the high-dimensional space :

  - Array representation: good for dense features

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

  - 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:

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

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 weigh 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.

No. of features: d

No. of examples: N

X_1
X_2
X_3

$\vdots$

X_N

x_i　　$X_{N \times d}$
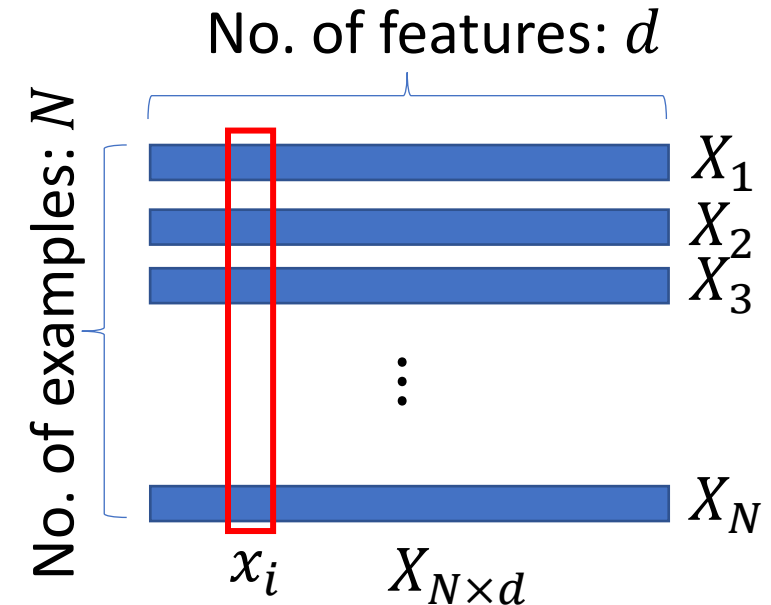
Feature Scaling

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

Gradient descent without scaling

Gradient descent after scaling variables

$x_1 \gg x_2$

$0 \leq x_1 \leq 1$
$0 \leq x_2 \leq 1$
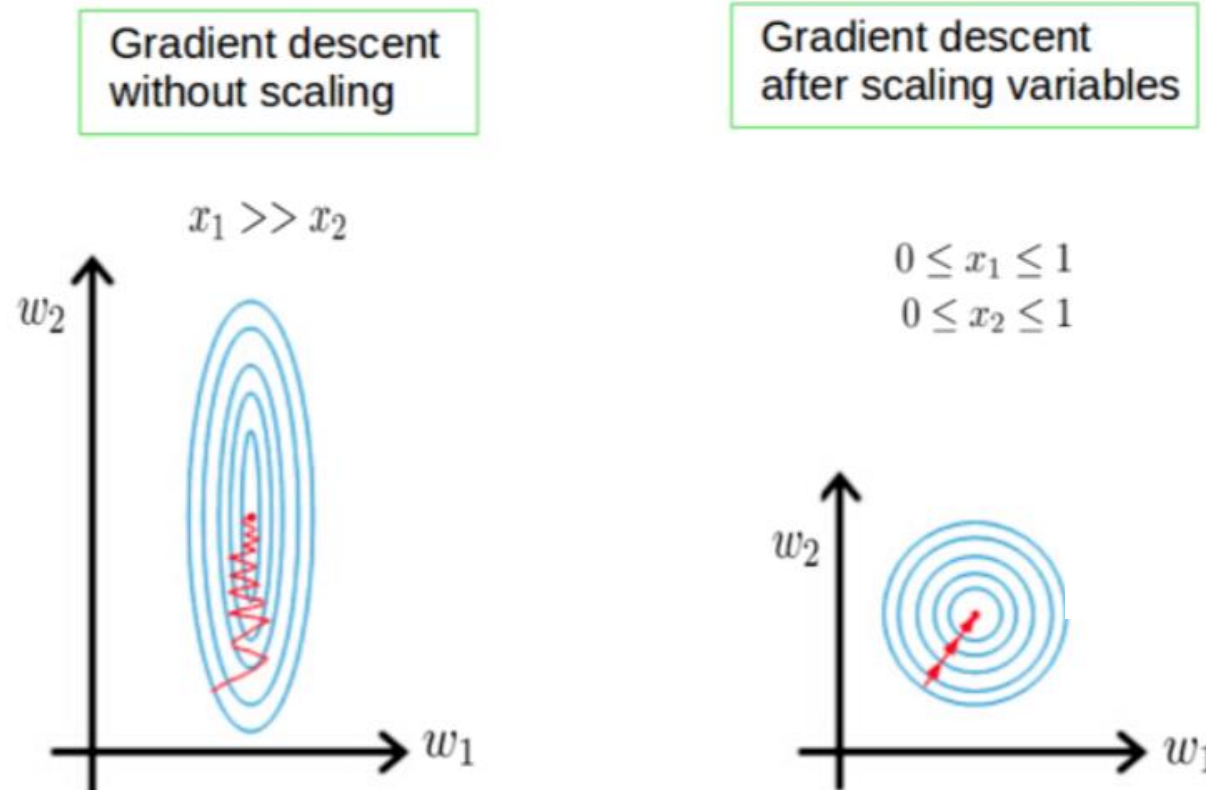
Feature Scaling: Standardization

- 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:

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

No. of features: d

No. of examples: N

X_1
X_2
X_3

$\vdots$

X_N

x_i $X_{N \times d}$
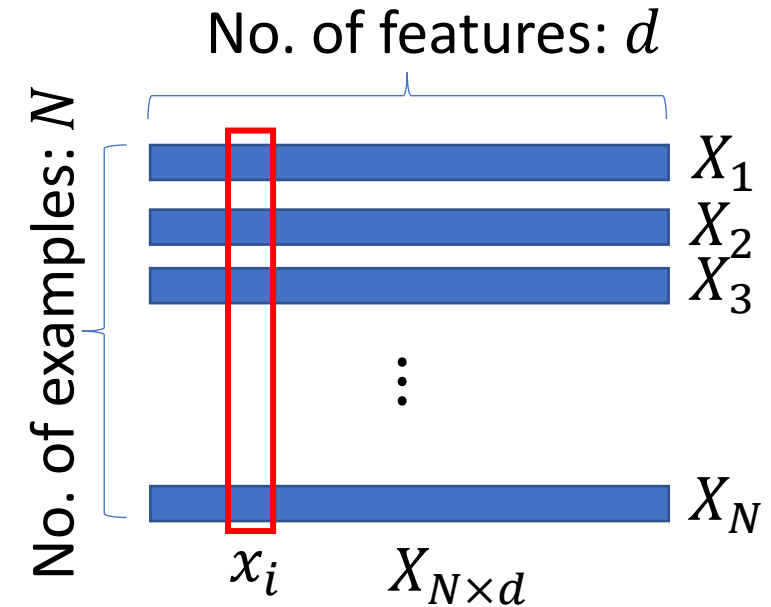
Feature Scaling: Min-Max Scaling

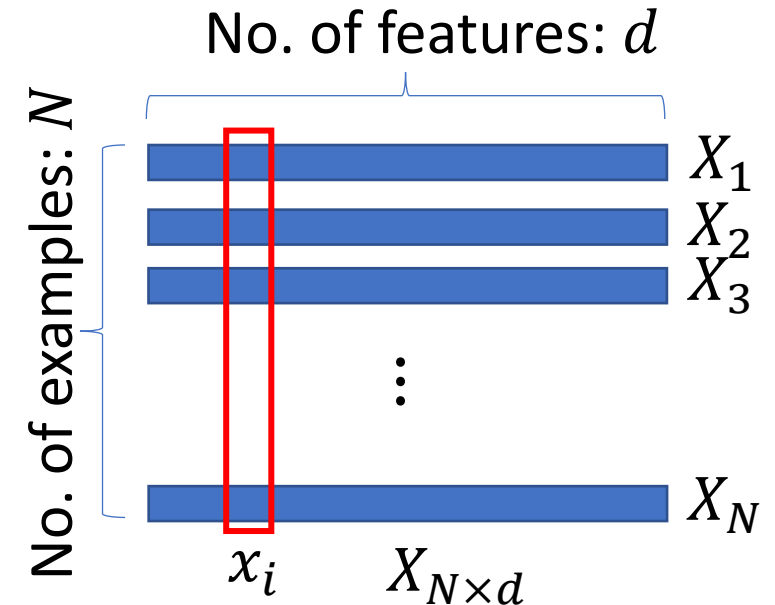- Transform the data to $[x_{min}^{new}, x_{max}^{new}]$.

- 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 $\frac{120-100}{150-100} = 0.4$

No. of features: d

No. of examples: N

X_1
X_2
X_3
$\vdots$
X_N

$x_i \qquad X_{N \times d}$

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 $f: R \to R$ 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

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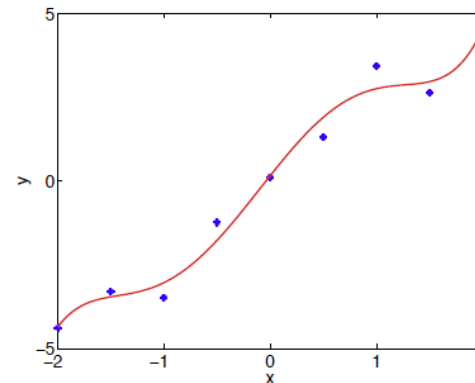
Polynomial Regression

- 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.
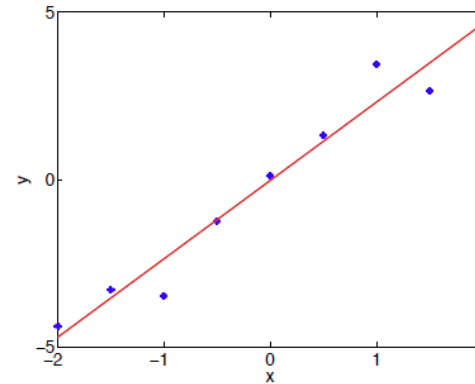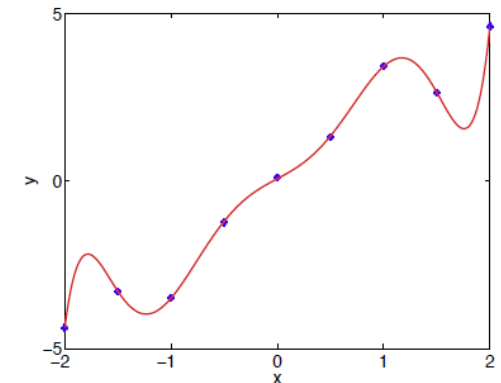

degree = 1
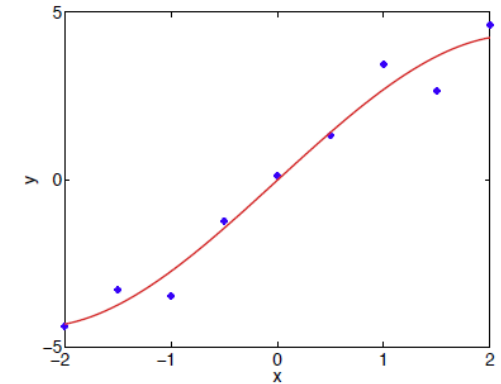
degree = 3

degree = 5

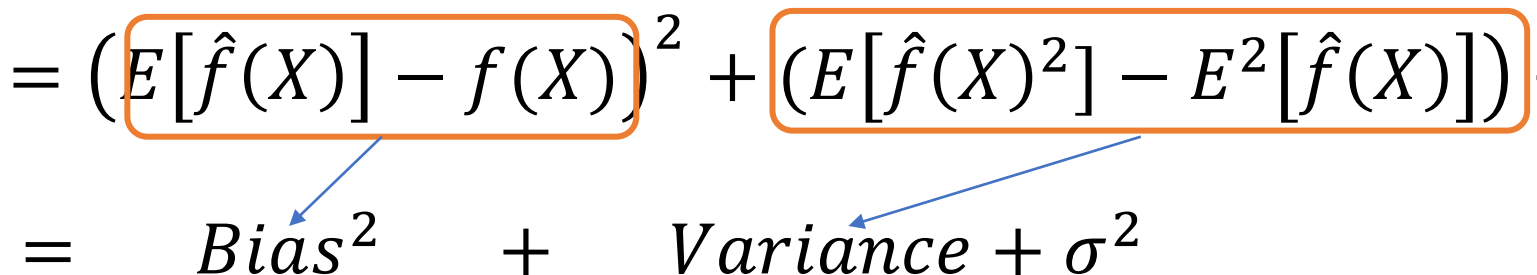degree = 7

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

Underfitting   Optimal   Overfitting

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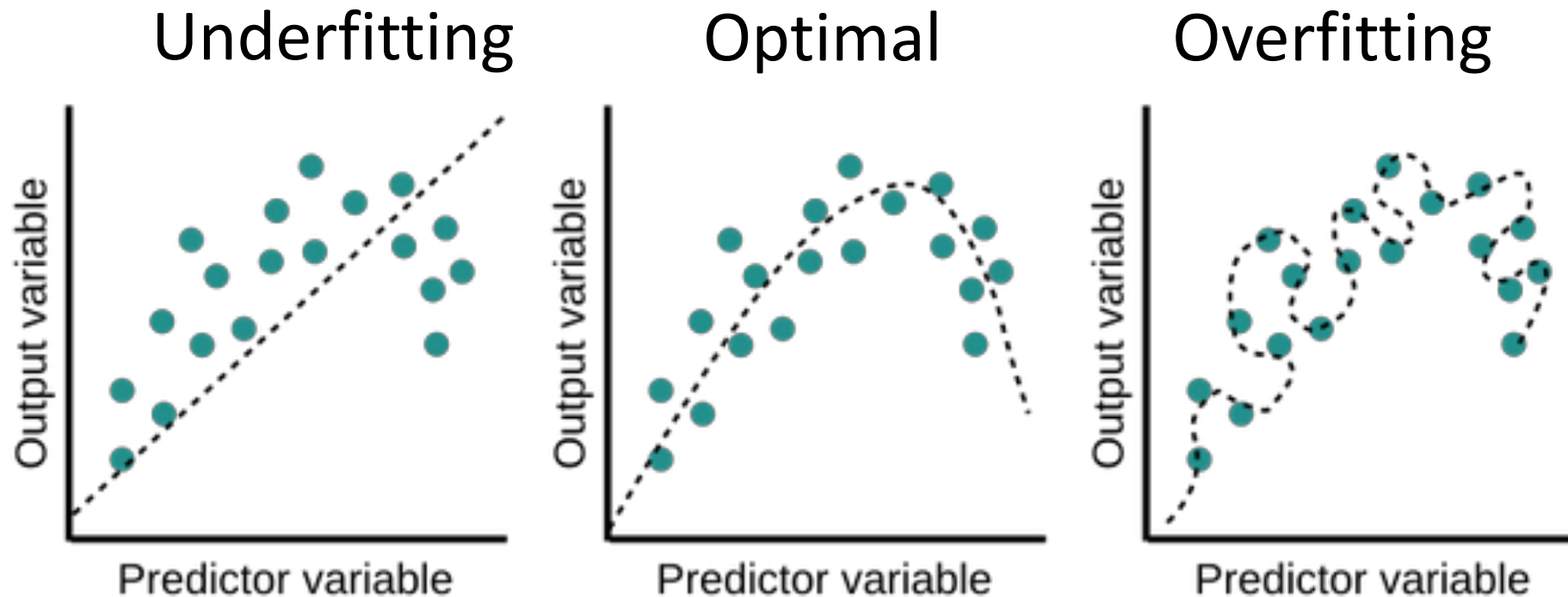
Bias and Variance

- 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:

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

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

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

Bias and Variance Tradeoff
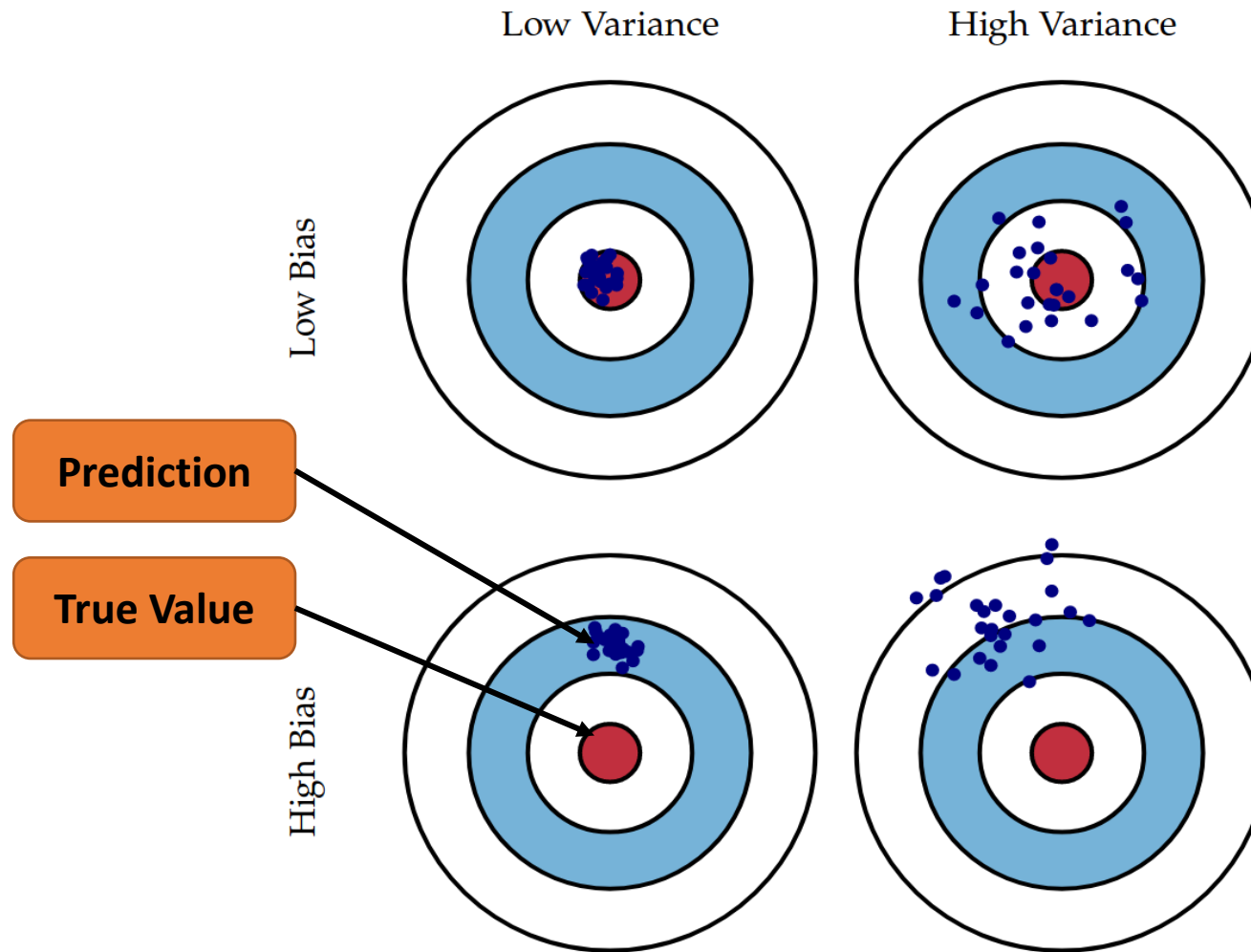
Solution for Large Bias

- 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:

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

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

Regularization

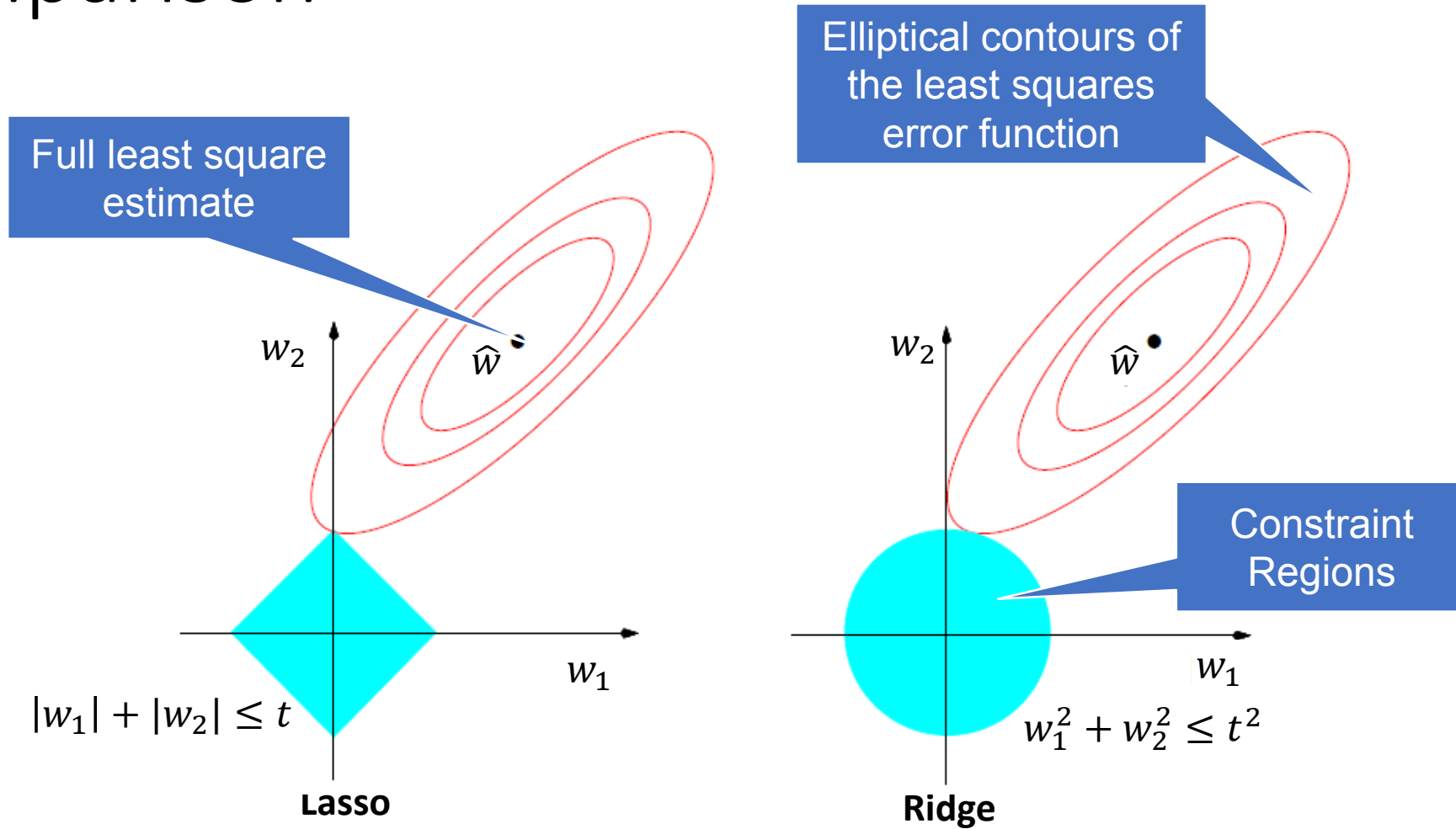- L_1 regularization: Lasso regression

$$L_1(X, Y, w) = \sum_{i=1}^{N} (f_w(X_i, w) - Y_i)^2 + \lambda|w|$$

- 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.

Comparison

Full least square estimate

Elliptical contours of the least squares error function

Constraint Regions

w_2

$\widehat{w}$

w_1

$|w_1| + |w_2| \le t$

Lasso

w_2

$\widehat{w}$
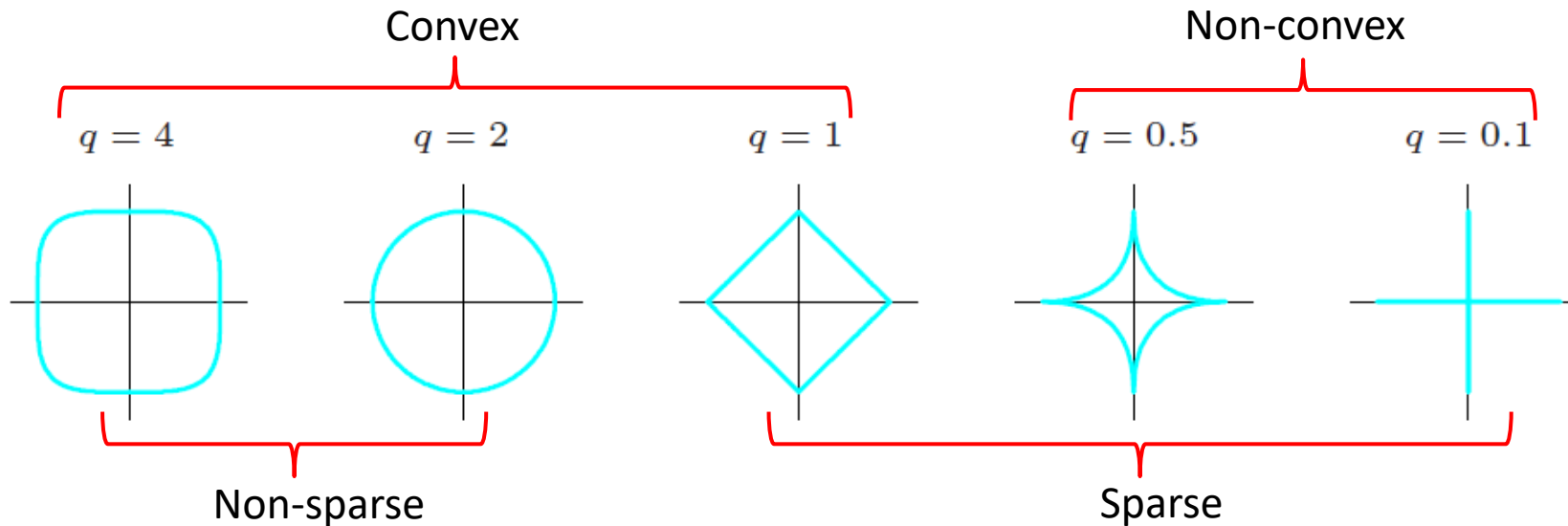
w_1

$w_1^2 + w_2^2 \le t^2$

Ridge

Note: Lasso has more opportunities for the estimated parameters to be zero.

Generalization

- By generalizing the Ridge and Lasso, the criterion will become:

$$L_q(X, Y, w) = \sum_{i=1}^{N} (f_w(X_i, w) - Y_i)^2 + \lambda \sum_{j=1}^{d} |w_j|^q$$

- We can try using different values for q.

Convex   Non-convex

$q = 4$ $q = 2$ $q = 1$ $q = 0.5$ $q = 0.1$

Non-sparse   Sparse
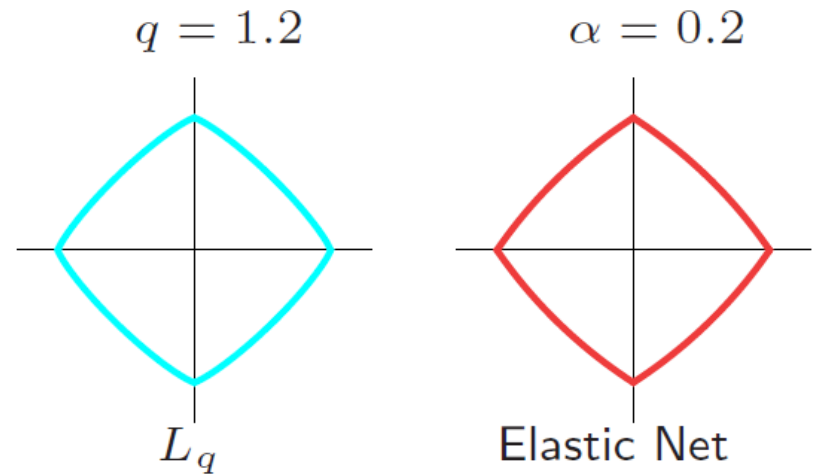
Elastic Net Penalty

- Elastic Net Penalty:

$$\sum_{j=1}^{d} (\alpha w_j^2 + (1 - \alpha)|w_j|)$$

- Proposed reasons:
  - For $q > 1$, the penalty cannot set the coefficients exactly to 0 as the way in Lasso.
  - Computational tractability.

- A compromise between Ridge and Lasso.

Elastic Net Penalty

- It is hard to detect the difference by eye.

- The elastic-net selects variables like Lasso, and shrinks together the coefficients of correlated features like Ridge.

Although visually very similar, the elastic-net has sharp (non-differentiable) corners, while the q = 1.2 penalty does not.

Selection Criteria

- Mean Squared Error (MSE): $MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - f(X_i))^2$

- Mean Absolute Error (MAE): $MAE = \frac{1}{N}\sum_{i=1}^{N}|Y_i - f(X_i)|$

- Root Mean Squared Error (RMSE): square toot of the MSE.

- R^2: represents the variance of the output variable explained by the model, and how much of the total error remains in the model.

$$RSS = \sum_{i=1}^{N}(Y_i - f(X_i))^2 \qquad TSS = \sum_{i=1}^{N}(Y_i - \bar{Y})^2$$

$$R^2 = 1 - \frac{RSS}{TSS}$$

Selection Criteria

- **MSE, MAE, RMSE**: the smaller the better.
- R^2: the larger the better.

- The model containing all of the features will always have the smallest MSE and the largest R^2, since these quantities are related to the training error.

- We wish to choose a model with low test error, not a model with low training error.

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.
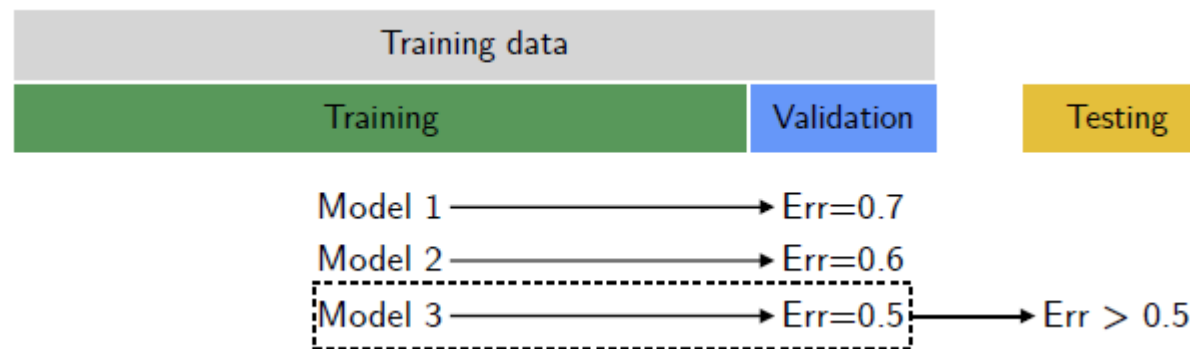
Training vs. Validation vs. Testing

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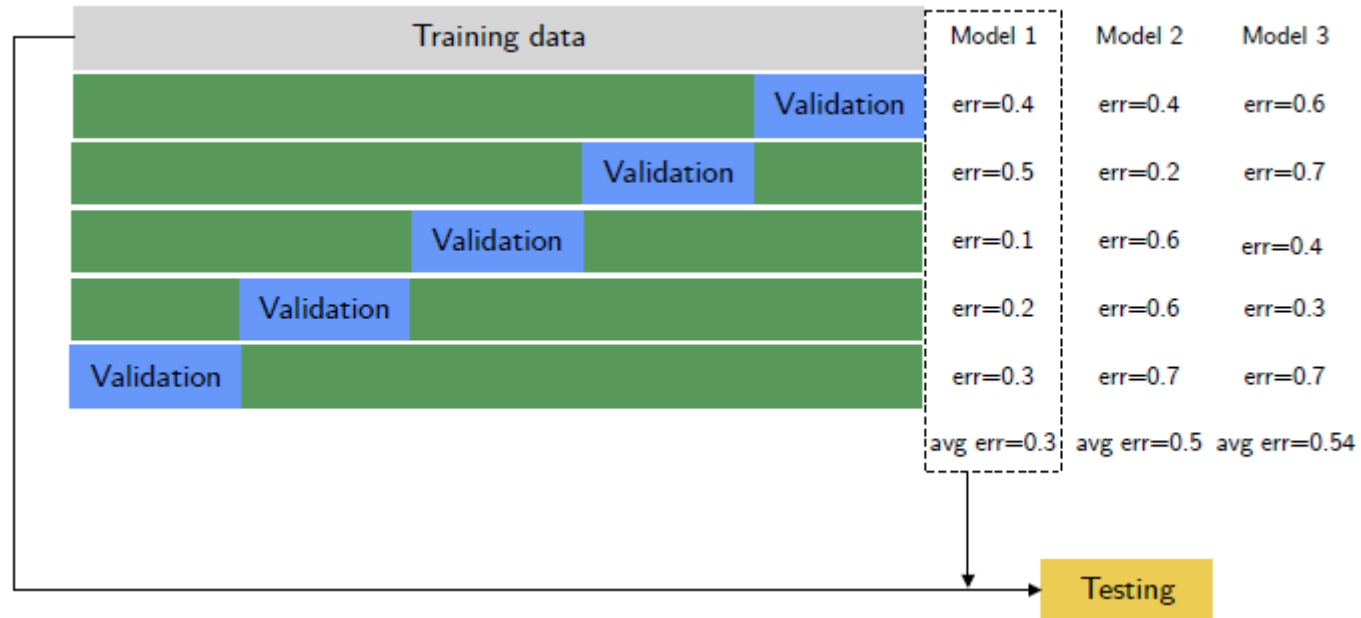
- In general: we perform k runs. Each run, it allows to use $\frac{k-1}{k}$ of the available data for training.
- If the number of data is very limited, we can set $k = N$ (total number of data points). This gives the leave-one-out cross-validation technique.

The Curse of Dimensionality - when d is large

- As dimensionality grows: fewer observations per region.

- When a measure such as a Euclidean distance is defined using many coordinates, there is little difference in the distances between different pairs of samples.

- The proportion of an inscribed hypersphere with radius r and dimension d, to that of a hypercube with edges of length $2r$:

$$\frac{V_{hypersphere}}{V_{hypercube}} = \frac{\frac{2r^d \pi^{d/2}}{d\Gamma(d/2)}}{(2r)^d} = \frac{\pi^{d/2}}{d 2^{d-1} \Gamma(d/2)} \to 0 \; as \; d \to \infty$$

- The hypersphere becomes an insignificant volume relative to that of the hypercube

The Curse of Dimensionality - another angle

- Consider a hypersphere of radius $r = 1$ in a space of d dimensions, and ask what is the fraction of the volume of the hypersphere that lies between radius $r = 1 - \varepsilon$ and $r = 1$. We can evaluate this fraction by noting that the volume of a hypersphere of radius r in d dimensions must scale as r^d, and so we write:

$$V_d(r) = K_d r^d$$

where K_d depends on d. Thus the fraction is

$$\frac{V_d(1) - V_d(1 - \varepsilon)}{V_d(1)} = 1 - (1 - \varepsilon)^d$$

- Most of the volume of a hypersphere is concentrated in a thin shell near the surface!

The Curse of Dimensionality

- Real data will often be confined to a region of the space having lower effective dimensionality, and in particular the directions over which important variations in the target variables occur may be so confined.

- Real data will typically exhibit some smoothness properties (at least locally) so that for the most part small changes in the input variables will produce small changes in the target variables, and so we can exploit local interpolation-like techniques to allow us to make predictions of the target variables for new values of the input variables.

Maximum Likelihood vs. Least Square Method

- In a linear model, if the errors belong to a normal distribution, the least squares estimators are the same as the maximum likelihood estimators.

Maximum Likelihood

- We assume there is a true function $f(x)$ and the true value is given by $y = f(x) + \varepsilon$ where $\varepsilon \sim N(0, \sigma^2)$. Thus we can write:

$$p(y|x, w, \beta) = N(y|f(x), \beta^{-1})$$

- Where $\beta^{-1} = \sigma^2$. Assuming the data points are drawn independently from the distribution, we obtain the likelihood function as:

$$p(Y|X, w, \beta) = \prod_{i=1}^{N} N(Y_i|f(X_i), \beta^{-1})$$

Maximum Likelihood

- The likelihood function:

$$p(Y|X,w,\beta) = \prod_{i=1}^{N} N(Y_i|f(X_i),\beta^{-1})$$

- The log-likelihood function:

$$\log p(Y|X,w,\beta) = \sum_{i=1}^{N} N(Y_i|f(X_i),\beta^{-1})$$

$$= \boxed{\frac{N}{2}log\beta - \frac{N}{2}\log 2\pi} - \boxed{\beta E_D(w)}$$

Not related to w   **sum-of-squares loss function**

$$E_D(w) = \frac{1}{2}\sum_{i=1}^{N}(Y_i - f(X_i))^2$$

Summary

- Thus, maximum likelihood is equivalent to minimizing the sum-of-squares loss function.

- Solving for w in linear regression when $f(X_i) = X_i w^T$:
$$w_{ML} = (X^T X)^{-1} X^T Y$$

Where X is an $N \times d$ matrix.

Summary of Today's Lecture

- Linear regression

- Optimization

- Features

- Beyond linear regression models

- Bias and variance

- The curse of dimensionality

- Maximum likelihood estimator