CS 559 Machine Learning

Linear Regression

Yue Ning
Department of Computer Science
Stevens Institute of Technology

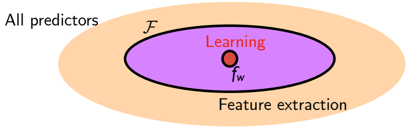
Plan for today

Regression Task

$$\begin{tikzcd} x \arrow[r] & \boxed{\text{function } f} \arrow[r] & y \end{tikzcd}$$

The Learning Problem

- **Hypothesis class**: we consider some restricted set $\mathcal{F}$ of mappings $f : \mathcal{X} \to \mathcal{Y}$ from input data to output.



- **Estimation**: on the basis of a set of training examples with their labels, $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, we find an estimate $\hat{f} \in \mathcal{F}$
- **Evaluation**: we measure how well $\hat{f}$ generalizes to yet unseen examples, i.e., whether $\hat{f}(\mathbf{x}_{\text{new}})$ agrees with y_{new}

Estimation Criterion

Loss

A loss function $\text{Loss}(\mathbf{x}, y, \mathbf{w})$ quantifies how wrong you would be if you used $\mathbf{w}$ to make a prediction on $\mathbf{x}$ when the correct output is y. It is the object we want to minimize.

- Loss is a function of the parameters $\mathbf{w}$ and we can try to minimize it directly.
- We reduce the estimation problem to a minimization problem.
- 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.

Linear Regression

Prediction

$$f_w(\mathbf{x}) = \hat{y} = \mathbf{w}^T\mathbf{x}$$

Residual

The residual is the amount by which prediction overshoots the target.

Squared loss

$$L(\mathbf{x}, y, \mathbf{w}) = (\underbrace{f_w(\mathbf{x}) - y}_{\text{residual}})^2$$

Example:

$$\mathbf{w} = [2, -1], \mathbf{x} = [2, 0], y = -1, L(\mathbf{x}, y, \mathbf{w}) = 25$$

Linear Regression

Prediction: $f_w(\mathbf{x}) = \hat{y} = \mathbf{w}^T\mathbf{x} = w_0 + w_1 x$

Loss: $L(\mathbf{x}, y, \mathbf{w}) = \underbrace{(f_w(\mathbf{x}) - y)}_{\text{residual}}{}^2$

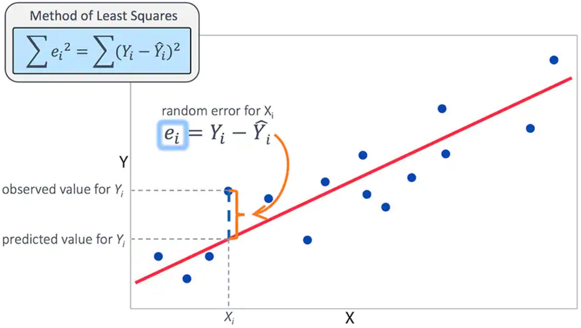


Figure: Fitting one-dimensional linear regression

Figure from JMP

Loss Minimization Framework

Loss is easy to minimize if there is only one example.

Minimize Total Training Loss

$$\min_{\mathbf{w}\in\mathbb{R}^d} \text{TrainLoss}(\mathbf{w}) = \underbrace{\frac{1}{N}\sum_{(x,y)\in D_{\text{train}}} L(\mathbf{x}, y, \mathbf{w})}_{\text{empirical loss}}$$

Key: learn a **w** to make global tradeoffs (N is the number of examples in D_{train}).

Training and Test performance: sampling

- Assume each training and test example-label pair $(\mathbf{x}, y)$ 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 $P(\mathbf{x}, y)$ so that each training/test example is a sample from this distribution $(\mathbf{x}_i, y_i) \sim P$.

$$\text{Empirical (training) loss} = \frac{1}{N} \sum_i \text{Loss}(y_i, f_w(\mathbf{x}))$$

$$\text{Expected (test) loss} = E_{(\mathbf{x},y)\sim P}\{\text{Loss}(y_i, f_w(\mathbf{x}))\}$$

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

Regression Loss Functions

Squared loss

$$L(\mathbf{x}, y, \mathbf{w})_2 = (f_w(\mathbf{x}) - y)^2$$

Absolute loss

$$L(\mathbf{x}, y, \mathbf{w})_1 = |f_w(\mathbf{x}) - y|$$

Which loss to use?

Example:

$$D_{\text{train}} = \{(1,0),(1,2),(1,1000)\}$$

- For least square regression (L2): **w** that minimizes training loss is mean y

$$\frac{\partial L_2}{\partial f_w} = -\frac{2}{N}\sum_{i=1}^{N}(y_i - f_w) = 0 \rightarrow f_w = \frac{1}{N}\sum_{i=1}^{N} y_i$$

- Mean: tries to accommodate every example, popular

Which loss to use?

Example:

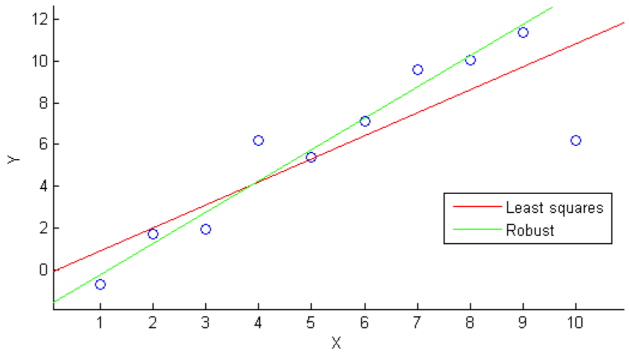
$$D_{\text{train}} = \{(1,0),(1,2),(1,1000)\}$$

- For least absolute deviation regression (L1): **w** that minimizes training loss is median y

  $$\frac{\partial L_1}{\partial f_w} = -\frac{1}{N}\sum_{i=1}^{N} \text{sgn}(y_i - f_w)$$

  (The derivative equals to 0 when there is the same number of positive and negative terms among $y_i - f_w$, which means f_w should be the median of y_i)
- Median: more robust with outliers.

Comparison



```
Least squares:          Y = -0.188327 + 1.10351*X          RMS error = 2.21375
Robust:                 Y = -1.77278 + 1.50415*X           RMS error = 1.43663
```

Figure: Least squares vs. robust regression; Demo from Mathworks

Squared loss: Closed Form Solution (Normal Equation)

- L2 loss:

$$\text{Squared loss} = \frac{1}{N} \sum_{(x,y) \in D_{\text{train}}} (\mathbf{w}^\top \mathbf{x} - y)^2$$

(Set the gradient w.r.t **w** to be zero?)

- Solution:

$$\mathbf{w}^* = (X^\top X)^{-1} X^\top y$$

Reading: Lineare regression lecture at Cornell CS

Linear regression on Wikipedia

Another solution for optimization: Gradient Descent

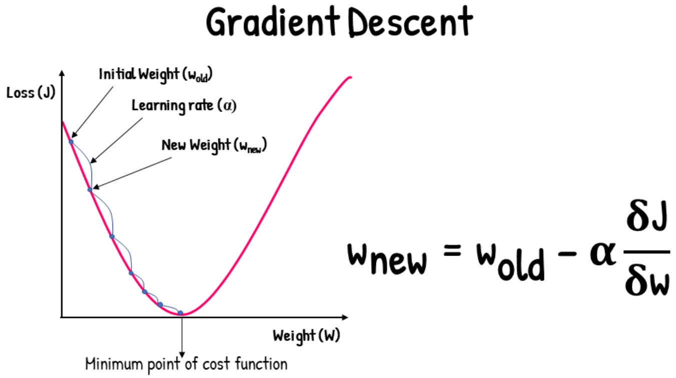

Gradient Descent
Loss (J)
Initial Weight (w_{old})
Learning rate (α)
New Weight (w_{new})
Weight (W)
Minimum point of cost function
$w_{new} = w_{old} - \alpha \frac{\delta J}{\delta w}$

Figure: From Understanding Gradient Descent Algorithm and Its Role in Linear Regression

Linear regression using gradient descent

Objective function

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

Gradient with respect to w (use chain rule)

$$\nabla_w \text{TrainLoss}(\mathbf{w}) = \nabla_w L(\mathbf{w}) = \frac{1}{N} \sum_{(\mathbf{x},y)\in D_{\text{train}}} 2(\underbrace{\mathbf{w}^T\mathbf{x}}_{\text{prediction}} - \underbrace{y}_{\text{target}})\mathbf{x}$$

Gradient descent is slow

Gradient

$$\nabla_w L(\mathbf{w}) = \frac{1}{N} \sum_{(x,y)\in D_{\text{train}}} 2(\underbrace{\mathbf{w}^T\mathbf{x}}_{\text{prediction}} - \underbrace{y}_{\text{target}})\mathbf{x}$$

Gradient descent update

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_w L(\mathbf{w})}_{\text{gradient}}$$

Each iteration requires going over all training examples – expensive when have lots of data

Stochastic Gradient Descent

Gradient Descent (GD)

$$\mathbf{w} \leftarrow \mathbf{w} - \underbrace{\eta}_{\text{step size}} \underbrace{\nabla_w L(\mathbf{w})}_{\text{gradient}}$$

Stochastic Gradient Descent (SGD)

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

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w \mathrm{L}_{(\mathbf{x},y)}(\mathbf{w})$$

Key: It's not about quality, it's about quantity.

(Batch) Gradient Descent

```
1: procedure BATCH GRADIENT DESCENT
2:     Initialize $\mathbf{w}$ with random values
3:     for $i$ in range(epochs) do
4:         $g^{(i)}(\mathbf{w}) = \text{evaluate\_gradient(TrainLoss, data, } \mathbf{w})$
5:         $\mathbf{w} = \mathbf{w} - \text{learning\_rate} * g^{(i)}(\mathbf{w})$
6:     Return $\mathbf{w}$ from the break point
```

- Can be very slow.
- Intractable for datasets that don't fit in memory.
- Doesn't allow us to update our model online, i.e. with new examples on-the-fly.
- Guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.

Stochastic Gradient Descent

```
1: procedure Stochastic Gradient Descent
2:     Initialize $\mathbf{w}$ with random values
3:     for $i$ in range(epochs) do
4:         np.random.shuffle(data)
5:         for example $\in$ data do
6:             $g^{(i)}(\mathbf{w})$ = evaluate_gradient(loss, example, $\mathbf{w}$)
7:             $\mathbf{w} = \mathbf{w} -$ learning_rate $* g^{(i)}(\mathbf{w})$
8:         Check stopping criteria
9:     return $\mathbf{w}$ from the break point
```

- Allow for online update with new examples.
- With a high variance that cause the objective function to fluctuate heavily.

Mini-batch Gradient Descent

```
1: procedure MINI-BATCH GRADIENT DESCENT
2:     Initialize $\mathbf{w}$ with random values
3:     for $i$ in range(epochs) do
4:         np.random.shuffle(data)
5:         for batch $\in$ get_batches(data, batch_size=50) do
6:             $g^{(i)}(\mathbf{w})$ = evaluate_gradient(loss, batch, $\mathbf{w}$)
7:             $\mathbf{w} = \mathbf{w} -$ learning_rate $* g^{(i)}(\mathbf{w})$
8:         Check stopping criteria
9:     return $\mathbf{w}$ from the break point
```

- 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

Gradient Update:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w \text{Loss}(x, y, \mathbf{w})$$

Question: what should the step size (learning rate) be?

Strategies:

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

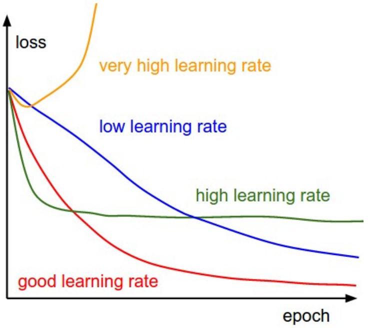
- Constant: $\eta = 0.1$
- Descreasing: $\eta = \frac{1}{\sqrt{\#\text{ updates made so far}}}$

Learning Rate

Set the learning rate η carefully:

$$\mathbf{w}^{(i)} \leftarrow \mathbf{w}^{(i-1)} - \eta \nabla L(\mathbf{w}^{(i-1)})$$

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).



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
  - e.g. $\frac{1}{t}$ decay: $\eta^{(t)} = \frac{\eta}{\sqrt{t+1}}$
- Learning rate cannot be one-size-fits-all
  - Giving different parameters different learning rates
  - AdaGrad! [paper in Journal of Machine Learning Research]

Reading: An overview of gradient descent optimization algorithms

Adagrad

- We have

$$\eta^{(t)} = \frac{\eta}{\sqrt{t+1}}, \qquad g_i^{(t)} = \frac{\partial L(x, y, \mathbf{w}_i^{(t)})}{\partial w_i^{(t)}}$$

- Vanilla Gradient descent

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

- Adagrad: Divide the learning rate of each parameter by the root mean square of its previous derivatives

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

where $\sigma^{(t)}$ is the root mean square of the previous derivatives of parameter w

Adagrad

$$
\begin{array}{ll}
w^{(1)} \leftarrow w^{(0)} - \frac{\eta^{(0)}}{\sigma^{(0)}} g^{(0)} & \sigma^{(0)} = \sqrt{(g^{(0)})^2 + \epsilon} \\
w^{(2)} \leftarrow w^{(1)} - \frac{\eta^{(1)}}{\sigma^{(1)}} g^{(1)} & \sigma^{(1)} = \sqrt{\frac{1}{2}[(g^{(0)})^2 + (g^{(1)})^2] + \epsilon} \\
w^{(3)} \leftarrow w^{(2)} - \frac{\eta^{(2)}}{\sigma^{(2)}} g^{(2)} & \sigma^{(2)} = \sqrt{\frac{1}{3}[(g^{(0)})^2 + (g^{(1)})^2 + (g^{(2)})^2] + \epsilon} \\
\ldots & \ldots \\
w^{(t+1)} \leftarrow w^{(t)} - \frac{\eta^{(t)}}{\sigma^{(t)}} g^{(t)} & \sigma^{(t)} = \sqrt{\frac{1}{t+1} \sum_{i=1}^{t} (g^{(i)})^2 + \epsilon}
\end{array}
$$

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

Summary so far

- Linear predictors:
  $f_w(x)$ based on score $\mathbf{w}^T\mathbf{x}$
- Training loss minimization: learning as optimization:

$$\min_w \mathrm{L}(w)$$

- Stochastic gradient descent: optimization algorithm:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_w \mathrm{L}_{(\mathbf{x},y)}(\mathbf{w})$$

Two components

Score:

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

- Previous: learning **w** via optimization
- Next: feature extraction and vector representation based on prior domain knowledge

Example: Data

Example: specifies that y is the ground-truth output for **x**

$$(\mathbf{x}, y)$$

Training data: a set of examples

$$\begin{aligned} D_{\text{train}} = [&(\text{"...located in Hoboken, NJ..."}, 1.4M),\\ &(\text{"...Fairfax, VA..."}, 0.7M),\\ &(), ...] \end{aligned}$$

Feature Extraction

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

$$\begin{bmatrix} \text{average neighborhood house price:} & \$1.1M \\ \text{average school score:} & 5.6 \\ \text{distance to train stations:} & 3.4\text{(m)} \\ \text{parking garage:} & 1 \\ \text{size:} & 1200\text{(sqrt)} \end{bmatrix}$$

Feature Vector

Mathematically, feature vector doesn't need feature names:

$$\begin{bmatrix} \text{average neighborhood house price:} & \$1.1M \\ \text{average school score:} & 5.6 \\ \text{distance to train stations:} & 3.4(\text{m}) \\ \text{parking garage:} & 1 \\ \text{size:} & 1200(\text{sqrt}) \end{bmatrix} \rightarrow \begin{bmatrix} 1.1 \\ 5.6 \\ 3.4 \\ 1 \\ 1200 \end{bmatrix}$$

Feature vector

For in input X, its feature vector is:

$$\mathbf{x} = x_1, ..., x_d$$

$\mathbf{x}$ as a point in a high-dimensional space.

Linear Predictors

Weight vector $\mathbf{w} \in \mathbb{R}^d$

$$\begin{bmatrix} \text{avg\_n\_price} & 1.8 \\ \text{avg\_school} & 1.4 \\ \text{dist\_train:} & 0.8 \\ \text{parking:} & 1.5 \\ \text{size:} & 1.6 \end{bmatrix}$$

Feature vector $\mathbf{x} \in \mathbb{R}^d$

$$\begin{bmatrix} \text{avg\_n\_price} & \$1.1 \\ \text{avg\_school} & 5.6 \\ \text{dist\_train:} & 3.4 \\ \text{parking:} & 1 \\ \text{size:} & 1200 \end{bmatrix}$$

Score

weighted combination of features

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

Organization of Features

Task: predict the price of a house

$$\text{A house} \xrightarrow{\text{feature extraction}} \begin{bmatrix} \text{avg\_n\_price} & \$1.1 \\ \text{avg\_school} & 5.6 \\ \text{dist\_train:} & 3.4 \\ \text{parking:} & 1 \\ \text{size:} & 1200 \end{bmatrix}$$

Which features to construct? need prior knowledge

Feature Templates

A feature template is a group of features all computed in a similar way. Input:

a house located in......

Some feature templates:

- Average school score is __
- Distance to train stations less than __
- Located in __

Sparsity in feature vectors

Feature template: located in __

$$\text{A house located in ...} \xrightarrow{\text{feature extraction}} \begin{bmatrix} \text{located\_in\_nyc} & 1 \\ \text{located\_in\_DC} & 0 \\ \text{located\_in\_seattle} & 0 \\ \text{located\_in\_pittsburg} & 0 \\ \dots & \dots \\ \text{located\_in\_sf} & 0 \end{bmatrix}$$

Inefficient to represent all the zeros...

Feature vector representation

$$\begin{bmatrix} \text{avg\_n\_price} & \$1.1 \\ \text{avg\_school} & 5.6 \\ \text{dist\_train:} & 3.4 \\ \text{parking:} & 1 \\ \text{size:} & 1200 \end{bmatrix}$$

Array representation: (good for dense features):

$$[1.1, 5.6, 3.4, 1, 1200]$$

Map representation: (good for sparse features):

$$\{\text{“avg\_n\_price”} : 1.1, \text{“avg\_school”} : 5.6\}$$

Example: Vector Representation

Linear functions:

$$\mathbf{x} = [x]$$

$$\mathcal{F}_1 = \{x \to w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 = 0\}$$

Quadratic functions:

$$\mathbf{x} = [x, x^2]$$

$$\mathcal{F}_2 = \{x \to w_1 x + w_2 x^2 : w_1 \in \mathbb{R}, w_2 \in \mathbb{R}\}$$

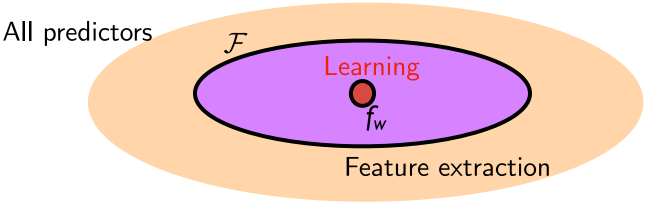
Quadratic functions are supersets of linear functions

Expressivity

Hypothesis class

A hypothesis class is the set of possible predictors with a fixed **x** and a varying **w**

$$\mathcal{F} = \{f_w : \mathbf{w} \in \mathbb{R}^d\}$$



Features in linear predictors

Example: medical diagnosis

- Input features: height, weight, body temperature, blood pressure, etc.
- Output: real value.

Three issues: (non-linear in original measurements)

- Non-monotonicity
- Saturation
- Interactions between features

Non-monotonicity (1)

- Features, Try(1):

$$\mathbf{x} = [1, \text{temperature}(x)]$$

- Output:

$$\text{health } y \in \mathbb{R}$$

- Problem: favor extremes; true relationship is non-monotonic

Non-monotonicity (2)

- Features, Try(2): transform inputs

$$\mathbf{x} = [1, (\text{temperature}(x) - 37)^2]$$

- Output:

$$\text{health } y \in \mathbb{R}$$

- Disadvantage: requires manually-specified domain knowledge

Non-monotonicity (3)

- Features, Try(3): transform inputs

$$\mathbf{x} = [1, \text{temperature}(x), \text{temperature}(x)^2]$$

- Output:

$$\text{health } y \in \mathbb{R}$$

- General rule: features should be simple building blocks to be pieced together

Saturation (1)

Example: product recommendation

- Input: product information x
- Output: relevance y

The number of people who bought x

$$\text{Identity: } \mathbf{x} = N(x)$$

Problem: is 1000 people really 10 times more relevant than 100 people? Not quite...

Saturation (2)

- The number of people who bought x

$$\text{Identity: } \mathbf{x} = N(x)$$

- Log

$$\mathbf{x} = \log N(x)$$

- Discretization

$$\mathbf{x} = [\mathbb{I}(1 < N(x) \leq 10), \mathbb{I}(10 < N(x) \leq 100), ...]$$

Interaction between features (1)

Example: predicting health

- Input: patient information x
- Output: health y

Try (1):

$$\mathbf{x} = [\text{height}(x), \text{weight}(x)]$$

Problem: cannot capture relationship between height and weight.

Interaction between features (2)

Try (2):

$$\mathbf{x} = (52 + 1.9(\text{height}(x) - 60) - \text{weight}(x))^2$$

Solution: define features that combine inputs.
Disadvantage: requires manually-specified domain knowledge.

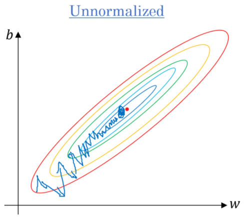
Interaction between features (3)

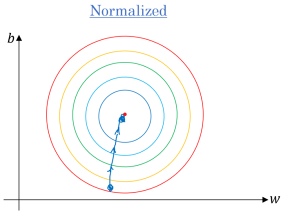
Try (3):

$$\mathbf{x} = [1, \text{height}(x), \text{weight}(x), \text{height}(x)^2, \text{weight}(x)^2, \text{height}(x)\text{weight}(x)]$$

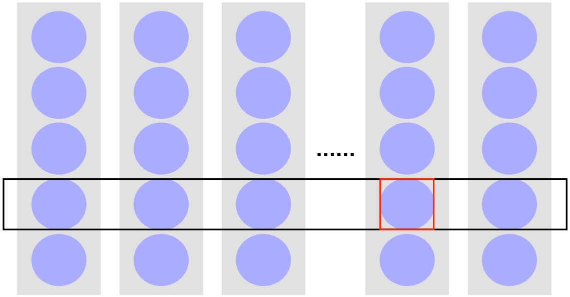
Solution: add features involving multiple measurements.

Feature Scaling





Feature Scaling



Assuming we have n $(r = 1, ..., n)$ examples. For each dimension i, the mean is m_i, and the standard deviation is σ_i. Scaling features:

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

Linear in what?

Prediction driven by score:

$$\mathbf{w}^T \mathbf{x}$$

- Linear in $\mathbf{w}$? Yes
- Linear in $\mathbf{x}$? Yes
- Linear in each input feature? No!

Non-linearity

- Predictors can be expressive non-linear functions and decision boundaries of x
- Score is linear function of w, which permits efficient learning

Summary so far

- Goal: define features so that the hypothesis class contains good predictors
- Pay attention to non-linearity in x: non-monotonicity, saturation, interaction between features
- Suggested approach: define features $\mathbf{x}$ to be building blocks (e.g., monomials)
- Linear prediction: actually very powerful!

Closed-form solution

- Objective:
$$\min L = \min \sum_{n=1}^{N} \{y_n - \mathbf{w}^T \mathbf{x}_n\}^2$$
- Transform into matrix operation ($\mathbf{y}_{N\times 1}, X_{N\times d}, \mathbf{w}_{d\times 1}$):
$$\min (\mathbf{y} - X\mathbf{w})^T (\mathbf{y} - X\mathbf{w})$$
- Differentiating w.r.t **w**:
$$X^T(\mathbf{y} - X\mathbf{w}) = 0$$
- Setting the gradient to zero, we get a closed form of estimates:
$$\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$$
where X is an $N \times d$ matrix. The solution is a linear function of the outputs **y**
- The resulting prediction errors $\epsilon_i = y_i - f(\mathbf{x}_i)$ are uncorrelated with any linear function of the inputs **x**.
- Easily extend to non-linear functions of the inputs.

Beyond Linear Regression

- Example extension: m^{th} order polynomial regression on one-dimensional input where $f : \mathcal{R} \to \mathcal{R}$ is given by:

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

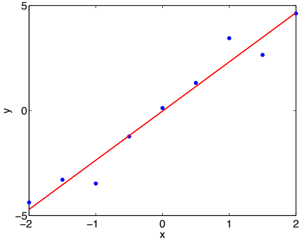
- Solution as before:

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

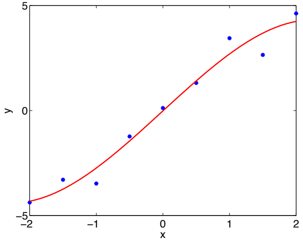
where:

$$\mathbf{w} = \begin{pmatrix} w_0 \\ w_1 \\ \ldots \\ w_m \end{pmatrix}, \ \mathbf{X} = \begin{pmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^m \\ 1 & x_2 & x_2^2 & \ldots & x_2^m \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_N & x_N^2 & \ldots & x_N^m \end{pmatrix}$$

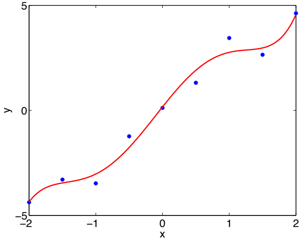
Polynomial Regression



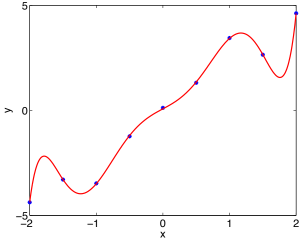
degree = 1



degree = 3



degree = 5



degree = 7

Underfitting and Overfitting

- Underfitting:
  - Training an ML algorithm on training data $\rightarrow$ 0.24 accuracy
  - Applying the ML algorithm on test data$\rightarrow$ 0.25 accuracy
- Overfitting:
  - Training an ML algorithm on training data $\rightarrow$ 0.95 accuracy
  - Applying the ML algorithm on test data$\rightarrow$ 0.27 accuracy (huge generalization error!)

Underfitting and Overfitting

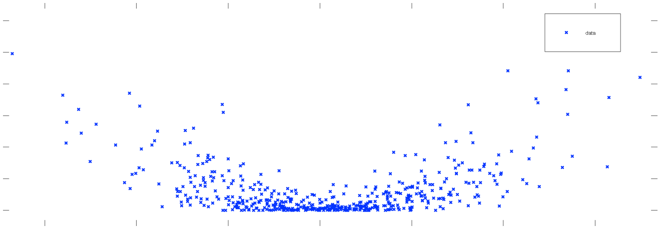


Figure: Regression data

Underfitting and Overfitting

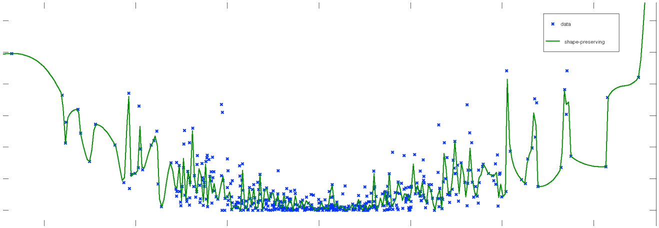


Figure: Overfitting

Underfitting and Overfitting

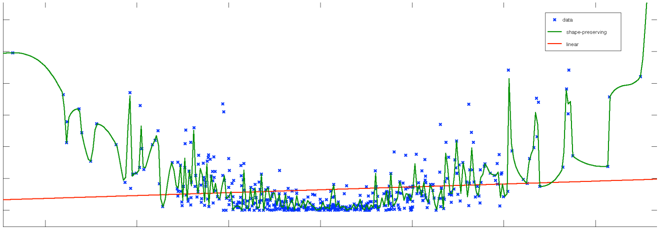


Figure: underfitting

Underfitting and Overfitting

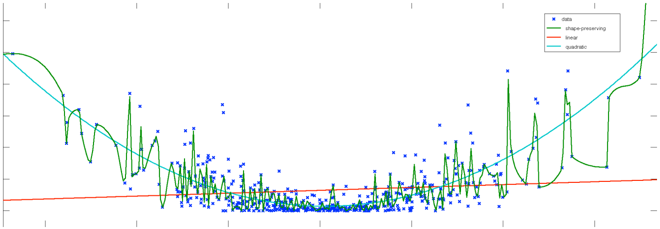


Figure: Overfitting, underfitting, and generalization

Underfitting and Overfitting

- Underfitting:
  - Low dimensional
  - Heavily regularized
  - Bad modeling assumption
- Overfitting:
  - High dimensional or non-parametric
  - Weakly regularized
  - Not enough data

Summary

- Training models too complex can cause overfitting
- Training models too simple (or wrong) can cause underfitting

Bias and Variance

- A training set $(x_1, ..., x_n)$ and y values
- True function $y = f(x) + \epsilon$ (ϵ has mean 0 and variance σ^2).
- $\hat{f}(x)$ that approximates the true function $f(x)$
- Expected squared prediction error at a point x is:

$$E[(y - \hat{f}(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{bias}^2} + \underbrace{(E[\hat{f}(x)^2] - E^2[\hat{f}(x)])}_{\text{Variance}} + \sigma^2$$

where

$$\text{Bias}[\hat{f}(x)] = E[\hat{f}(x)] - f(x), \text{Var}[\hat{f}(x)] = E[\hat{f}(x)^2] - E^2[\hat{f}(x)]$$

- Bias: error from incorrect modeling assumption
- Variance: error from random noise

Bias and Variance

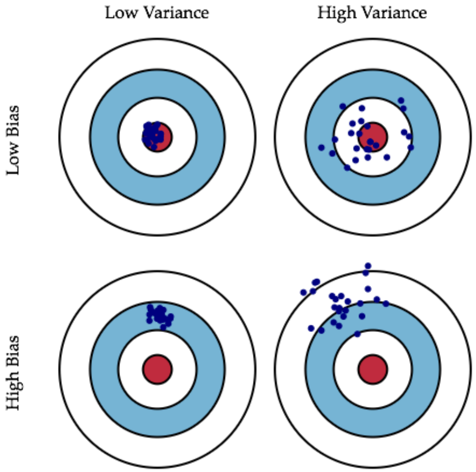


Figure: Graphical illustration of bias vs. variance

What to do with large bias?

Diagnosis:

- 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 test data, then you probably have large variance (overfitting)

For bias: redesign your model

- Add more features as input
- A more complex model

What to do with large variance?

- More data (very effective, but not always practical)
- Regularization

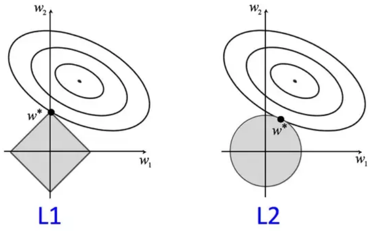
Regularization

- L-2 regularization (Ridge regression)

$$L_2(X, y, \mathbf{w}) = \sum_{n=1}^{N} (f_w(x_n, y_n, \mathbf{w}) - y_n)^2 + \underbrace{\alpha \mathbf{w}^T \mathbf{w}}_{\text{L2 regularization}}$$

- L-1 regularization (LASSO regression)

$$L_1(X, y, \mathbf{w}) = \sum_{n=1}^{N} (f_w(x_n, y_n, \mathbf{w}) - y_n)^2 + \underbrace{\alpha |\mathbf{w}|}_{\text{L1 regularization}}$$



Model Selection

- Usually a trade-off between bias and variance
- Select a model that balances two kinds of error to minimize total error
- Cross-validation: it allows us to estimate the generalization error based on training examples alone.

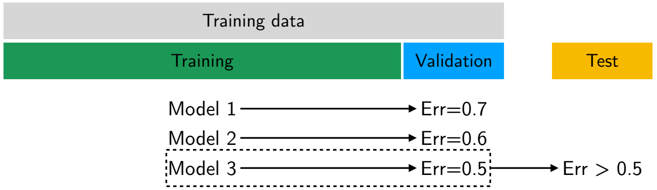


Figure: Training vs. Validation vs. Test

K-fold Cross Validation

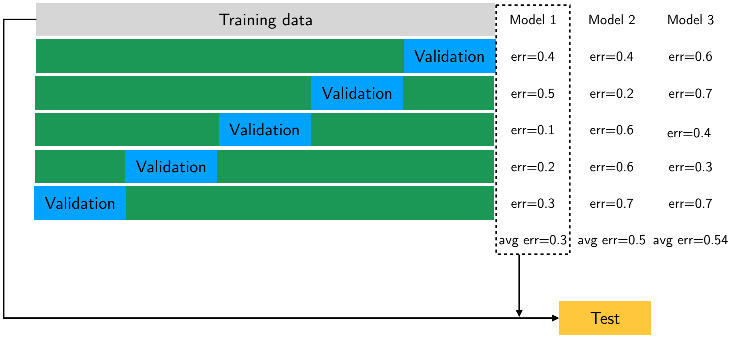


Figure: K-fold Cross Validation

- In general: we perform K runs, Each run, it allows to use (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 $2r$:

$$\frac{V_{\text{hypersphere}}}{V_{\text{hypercube}}} = \frac{\frac{2r^d \pi^{d/2}}{d\Gamma(d/2)}}{(2r)^d} = \frac{\pi^{d/2}}{d2^{d-1}\Gamma(d/2)} \to 0 \text{ as } d \to \infty$$

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 $r = 1 - \epsilon$ 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:

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

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.

Warning of Math

Maximum Likelihood

We assume there is a true function $f(x)$ and the true value is given by $y = f(x) + \epsilon$ where ϵ is a Gaussian distribution with mean 0 and variance σ^2. Thus we can write:

$$p(y|x, \mathbf{w}, \beta) = \mathcal{N}(y|f(x), \sigma^2)$$

Assuming the data points are drawn independently from the distribution

- The likelihood function: $p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(y_n|f(x_n), \sigma^2)$

Maximum Likelihood

The likelihood function:

$$p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(y_n|f(x_n), \sigma^2)$$

The log-likelihood function:

$$\begin{aligned}
\log p(\mathbf{y}|\mathbf{X}, \mathbf{w}, \beta) &= \sum_{n=1}^{N} \log \mathcal{N}(y_n|f(x_n), \sigma^2) && (1)\\
&= \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_n - f(x_n))^2}{2\sigma^2}} && (2)\\
&= \underbrace{\frac{N}{2} \log \frac{1}{\sigma^2} - \frac{N}{2} \log(2\pi)}_{\text{not related to } \mathbf{w}} - \frac{1}{\sigma^2} E_D(\mathbf{w}) && (3)
\end{aligned}$$

where:

$$E_D(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y_n - f(x_n)\}^2$$

Maximum A Posterior (MAP) Estimation

- Assuming **w** has a prior distribution
- We estimate $p(\mathbf{w}|\mathbf{x}, y)$ which is equivalent to $\underbrace{p(y|\mathbf{x}, \mathbf{w})}_{\text{likelihood}} \times \underbrace{p(\mathbf{w})}_{\textit{prior}}$
- MAP is equivalent to minimizing the **regularized** sum-of-squares loss under certain assumptions.

Summary

- Maximum likelihood (ML) $\approx$ minimizing the sum-of-squares
- MAP $\approx$ minimizing the regularized sum-of-squares
- Solving for $\mathbf{w}$ in linear regression when $f(\mathbf{x}_n) = \mathbf{w}^\top \mathbf{x}_n$:

$$\mathbf{w}_{ML} = (X^T X)^{-1} X^T \mathbf{y}$$

  where X is an $N \times d$ matrix. The solution is a linear function of the outputs $\mathbf{y}$

End of Warning

Acknowledgements

Slides adapted from Dr. Percy Liang and Dr. Stefano Ermon's AI at Stanford University and Dr. Bert Huang's Machine Learning at Virginia Tech.