Linear Models for Regression

Key concepts:

- Sum of squared error for regression loss
- Gradient descent for loss minimization
- Closed form solution for minimizing SSE
- Maximum likelihood estimation and SSE's probabilistic interpretation
- Overfitting and regularization

CS534

Example Regression Problems

- Predict housing price based on
  - House size, lot size, Location, # of rooms …
- Predict stock price based on
  - Price history of the past month …
- Predict the abundance of a species based on
  - Environmental conditions, shrubs? trees? …

A Basic Set Up

Given: a training set, containing a total of N training examples $(\mathbf{x}_i, y_i)$, for $i = 1, \ldots, N$

Goal: learn a function $\hat{y}$ that maps from $\mathbf{x}$ to y

Choice of hypothesis space?

To begin, we consider a linear hypothesis space (thus the name linear regression):

Let $\mathbf{x} = [1, x_1, x_2, \ldots, x_d]^T$

$$\hat{y}(\mathbf{x}) = w_0 + w_1 x_1 + \cdots + w_d x_d = \mathbf{w}^T \mathbf{x}$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$

Setting Up Loss Function

- Given a set of training examples

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2) \ldots, (\mathbf{x}_N, y_N)$$

$$\mathbf{x}_i \in R^{d+1}, y_i \in R$$

- Hypothesis space (representation of the function):

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

- Loss function --- sum-squared error (more on why later):

$$L(\mathbf{w}) = \sum_{i=1}^{N} (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

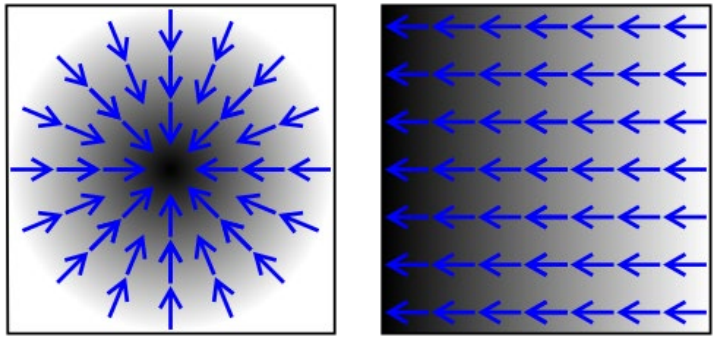
Or equivalently*, mean-squared error:

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N} (\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

Equivalent because the optimal solution w for one will also be optimal for the other.

Minimization of the loss function

- Many ways to optimize this objective function
- A simple approach is **gradient descent**
  - Start with some random guess of the parameter
  - Iteratively improve the parameter by following the **steepest descent direction**
- **Gradient**: multivariate generalization of derivative, points in the direction of greatest rate of increase of the function



From Wikipedia

Values of the function in black and white, black representing higher values
Gradient represented by blue arrows.

Gradient descent to minimize $f(\mathbf{w})$

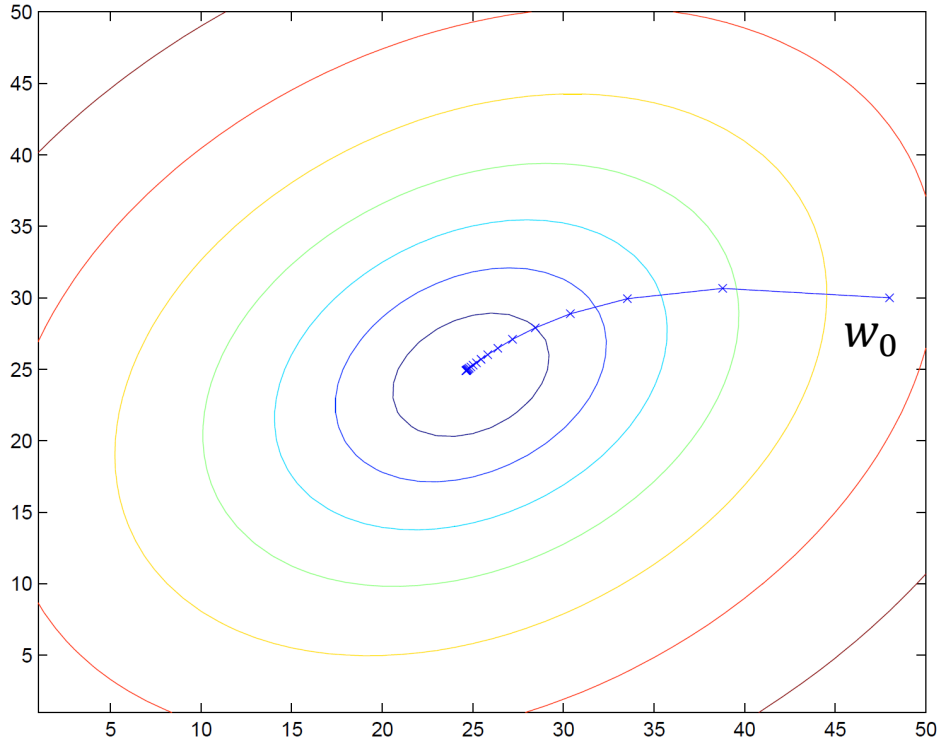


Figure shows the contours of a quadratic function. Gradient descent starts from w_0, and gradually moves toward the optimal solution.

1. Start from some initial guess $\mathbf{w}^0$

2. Find the direction of steepest descent – opposite of the gradient direction $\left(-\nabla f(\mathbf{w})\right)$

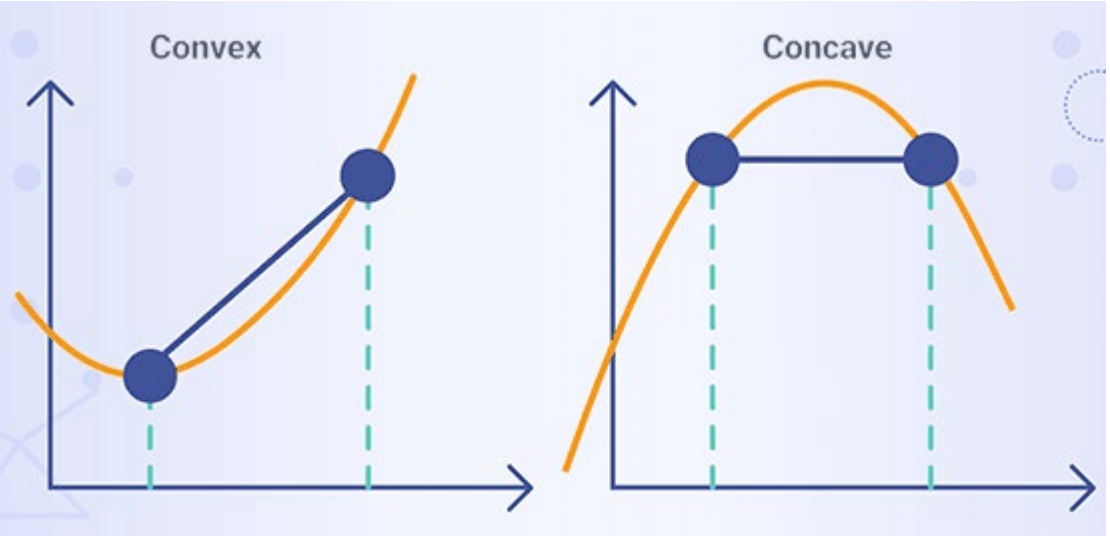
3. Take a update step, controlling step size by **learning rate** λ

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \lambda \nabla f(\mathbf{w}^t)$$

4. Repeat for fixed iteration or till no local improvement is possible, i.e., $|\nabla f(\mathbf{w}^t)| \leq \epsilon$

Convergence of Gradient Descent

- Given small learning rate, gradient descent is guaranteed to converge to the global minimum



Useful identities: derivatives

Product rule: $$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) \quad (5.29)$$

Quotient rule: $$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2} \quad (5.30)$$

Sum rule: $$(f(x) + g(x))' = f'(x) + g'(x) \quad (5.31)$$

Chain rule: $$\big(g(f(x))\big)' = (g \circ f)'(x) = g'(f(x))f'(x) \quad (5.32)$$

Derivative with examples:

$$f(x) = x^k; \quad f'(x) = kx^{k-1}$$
$$f(x) = e^x + 2x^2; \quad f'(x) = e^x + 4x$$

Example:

$$f(x) = e^x, g(t) = t^2$$
$$h(x) = g\big(f(x)\big) = (e^x)^2$$
$$h'(x) = 2e^x * e^x = 2e^{2x}$$

From derivative to gradient

Generalizing to multivariate function:

For $f(\mathbf{x})$ where $\mathbf{x} = [x_1, \dots, x_d]^T$

Gradient: $\nabla f(\mathbf{x}) = \left[\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_d}\right]$

*Gradient is often represented as a row vector, but sometimes also as column vector for convenience of representation. Watch for context to ensure proper interpretation.

Read Chapter 5 Section 1-3 of Mathematics for Machine learning for the necessary background

Gradient Descent for MSE

$$L(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)^2$$

We need to compute the gradient of $L(\mathbf{w})$:

$$\frac{\partial L}{\partial w_k} = \frac{2}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)x_{ik}$$

$$\nabla L(\mathbf{w}) = \frac{2}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)\mathbf{x}_\mathrm{i} = \frac{2}{N}\sum_{i=1}^{N}(\widehat{y_i} - y_i)\mathbf{x}_\mathrm{i}$$

Batch Gradient Descent for MSE

$\mathbf{w} = \mathbf{w}^0$

Repeat{

$$\nabla L(\mathbf{w}) = \frac{2}{N}\sum_{i=1}^{N}(\mathbf{w}^T\mathbf{x}_i - y_i)\mathbf{x}_i$$

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda\nabla L(\mathbf{w})$$

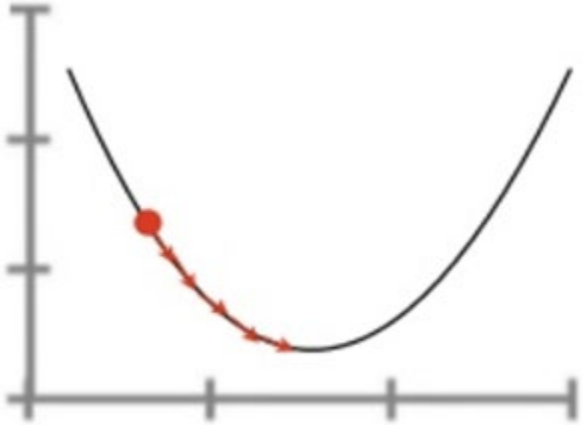
} until $|\nabla L(\mathbf{w})| \leq \epsilon$

Using SSE is equivalent to using MSE but with $N\lambda$ learning rate

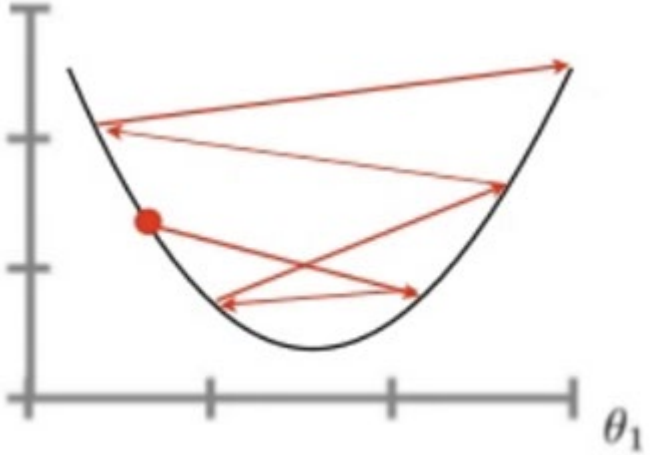
- The update rule goes through all training examples before taking each update step – thus called batch gradient descent
  - Examples with larger error ($\mathbf{w}^T\mathbf{x}_i - y_i$) contribute more to the update
- λ: learning rate or step size
- ϵ: convergence criterion
- The MSE Objective is convex – converge to global optimal

Impact of learning rate

- Large λ makes faster progress, but could lead to oscillation or divergence
- Small λ avoids overstepping, but overly small λ leads to slow convergence



Small λ: smooth convergence



large λ: overshooting and diverging

How to choose λ?

- Fixed value
  - Too large: can lead to divergence
  - Too small: can lead to overly slow convergence
- Reducing λ on a schedule
  - Decrease λ based on a schedule e.g., $\lambda = \frac{1}{t}$, where t is the iteration counter
  - Larger steps for early iterations, smaller step when approaching the minimum
- Adaptive control of λ
  - If diverging (e.g., loss increases), shrink the step size
  - If converging (e.g, loss decreases), increase the step size

Stochastic gradient descent

$\mathbf{w} = \mathbf{w}^0$

Repeat for fixed # of epochs {

  Randomly shuffle training examples

  for each training example i

    $\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{2}{N} (\mathbf{w}^T \mathbf{x}_i - y_i) \mathbf{x}_i$ //update after each example

}

- It repeatedly runs through the training examples, each round of going through the whole training set is called one epoch
- It updates $\mathbf{w}$ immediately after seeing each example, based on the gradient for that training instance
- Depending on the order of the training examples, the training trajectory can vary drastically
- Randomly shuffling the order avoids detrimental effect of a bad ordering and improves convergence speed

Batch, Stochastic and Mini-batch gradient descent

- Batch: go through all training examples for computing the gradient
  - Can be inefficient
  - Sometimes infeasible for LARGE data (for example, memory issue)
- Stochastic gradient descent: compute gradient contribution for each training example and update immediately
  - Training can be jumpy, and convergence can be slow to achieve

- Stochastic gradient descent with Minibatch: go through a subset of the training example before updating
  - More robust
  - Can be a good tradeoff

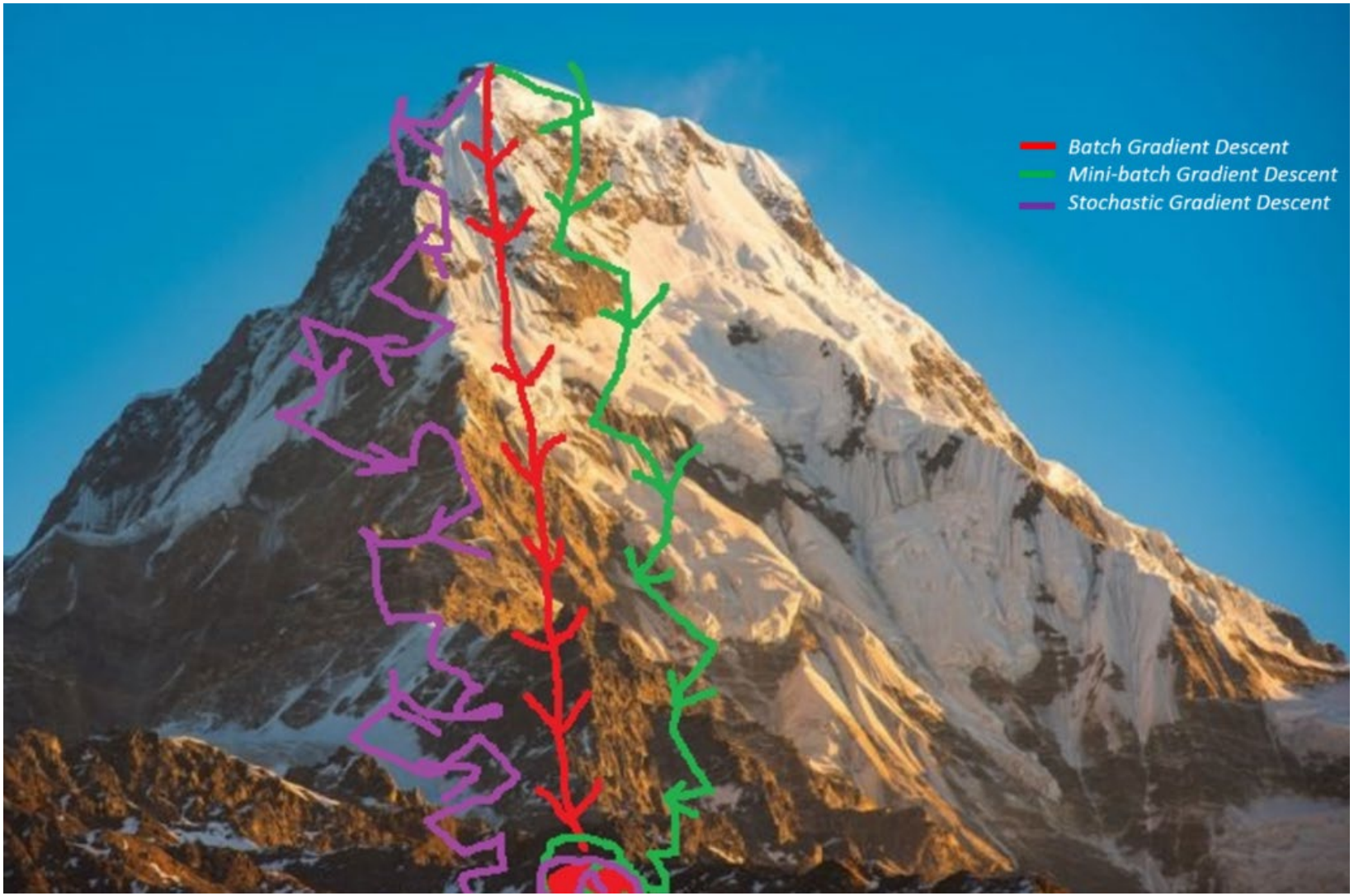


Image source:
https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Alternative way to optimizing $L(\mathbf{w})$

- By setting the gradient to zero, we can directly find the extreme (minimum) point of $L(\mathbf{w})$

- Define:

$$\boldsymbol{X} = \begin{bmatrix} x_{10} & x_{11} & \dots & x_{1d} \\ x_{20} & x_{21} & \dots & x_{2d} \\ \dots & \dots & \dots & \dots \\ x_{N0} & x_{N1} & \dots & x_{Nd} \end{bmatrix}, \qquad Y = \begin{bmatrix} y_1 \\ \dots \\ y_N \end{bmatrix}$$

We can show that MSE is:

$$L(w) = \frac{1}{N}\|X\mathbf{w} - Y\|^2 = \frac{1}{N}(X\mathbf{w} - Y)^T(X\mathbf{w} - Y)$$

$$= \frac{1}{N}(\boldsymbol{w}^T X^T X\boldsymbol{w} - 2\boldsymbol{w}^T X^T Y + Y^T Y)$$

IMP: Note here X = (N*d), w = (

$$\nabla L(\mathbf{w}) = \frac{1}{N}(2X^T X\mathbf{w} - 2X^T Y)$$

Setting gradient to zero

$$\nabla L(\mathbf{w}) = \frac{1}{N}(2X^T X\mathbf{w} - 2X^T Y) = 0$$

$$\boldsymbol{X^T X\mathbf{w} - X^T Y = 0}$$

$$\boldsymbol{X^T X\mathbf{w} = X^T Y}$$

The Normal Equation

$$\mathbf{w} = \boldsymbol{\left(X^T X\right)^{-1} X^T Y}$$

Direct solution vs. (stochastic) gradient descent

- Why do we not always use the direct solution?
  - Computationally expensive for large data and high dimensionality

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

  - Matrix inverse takes $O(d^3)$ time, where d is the dimension of the matrix to be inverted
- Gradient descent and especially stochastic gradient descent are more useful in large scale machine learning
  - In practice, we do not necessarily care about optimizing to high accuracy. It often does not pay off in terms of statistical performance

Our roadmap

- ✓ Sum of squared error for regression loss
- ✓ Gradient descent for loss minimization
- ✓ Closed form solution for minimizing SSE
- ❑ Maximum likelihood estimation and SSE's probabilistic interpretation
- ❑ Overfitting and regularization

Why MSE: a Probabilistic Interpretation

- One might ask, why is MSE a good objective?
- There is a natural probabilistic explanation for MSE with some simple probabilistic assumptions
- Before that, we will provide a brief of maximum likelihood estimation

Background: Maximum Likelihood Estimation

Parameter Estimation

- Given observation of some random variable(s), assuming the variable(s) follow some fixed distribution, how to estimate the parameters?

- Example:
  - coin tosses (Bernoulli distribution, parameter: *p – probability of head*)
  - Dice or grades (Discrete distribution, parameters: p_i for $i = 1, \dots, k-1$ with k categories)
  - Height and weight of a person (continuous variable, parameters depends on the distribution, for example for Gaussian Distribution, the parametes are μ, the mean and Σ, the covariance

Maximum Likelihood Principle

- We will use a general term θ to denote the collection of parameters we try to estimate
- We will use a general term $\boldsymbol{D}$ to denote the data that we observe
  - $\boldsymbol{D}$ is a set of examples, let D_i denote the i-th example in $\boldsymbol{D}$
- Assuming a fixed parameter θ, what is the probability of observing $\boldsymbol{D}$?
  - This can be written as $P(\boldsymbol{D}; \theta)$
- The maximum likelihood principle seeks to find M that maximizes this probability
- This is called the likelihood function
  - $L(\theta) = P(\boldsymbol{D}; \theta) = \prod_{i=1}^{N} P(D_i; \theta)$ --- here we are making the IID assumption, that is examples in ***D*** are **independently and identically distributed**, thus we can use the product
- It is often more convenient to work with the log of $P(\boldsymbol{D}; \theta)$
  - $l(\theta) = \log P(\boldsymbol{D}; \theta) = \sum_{i=1}^{N} \log P(D_i; \theta)$
- Goal: find θ that maximizes $l(\theta)$

Example: Coin Toss

- You are given a coin, and want to decide p- the probability of head of this coin
- You toss it for 500 times, and observe heads 242 times
- What is your estimate of p?

$$p = \frac{242}{500} = 0.484$$

- But why? – this is doing maximum likelihood estimation
- In other words, $p = 0.484$ leads to the highest probability of observing 242 heads out of 500 tosses among all possible values of p
- Now let's go over the math to convince ourselves

Example Cont.

- We have $D = \{x_1, \dots, x_N\}$, where x_i is the outcome of the m-th coin toss,
  - 1 means head, and 0 means tail
- Lets write down the likelihood function: $L(p) = \prod_{i=1}^{N} P(x_i; p)$
- For binary variable, we have a nice compact form to write down its probability mass function $P(x_i; p) = p^{x_i}(1-p)^{1-x_i}$
- So we have

$$L(p) = \prod_{i=1}^{N} p^{x_i}(1-p)^{1-x_i}$$

- Take the log:

$$l(p) = \log L(p) = \sum_{i=1}^{N} \log p^{x_i}(1-p)^{1-x_i}$$

$$= \boxed{\sum_{i=1}^{N} x_i} \log p + \boxed{\sum_{i=1}^{N} (1-x_i)} \log(1-p) = n_1 \log p + n_0 \log(1-p)$$

n_1: # of 1s $\qquad$ n_0: # of 0s

Maximizing the likelihood

- Take derivative of $l(p) = n_1 \log p + n_0 \log(1-p)$

$$\frac{dl(p)}{dp} = \frac{n_1}{p} - \frac{n_0}{1-p}$$

- Setting it to zero:

$$\frac{n_1}{p} = \frac{n_0}{1-p}$$

- Solving this leads to:

$$p = \frac{n_1}{n_0 + n_1} = \frac{n_1}{N}$$

End of quick intro on MLE

Read Section 8.3 of Mathematics for Machine Learning for an introduction to MLE

Basic steps of MLE

- Identify θ, the parameters to be estimated
- Write out the likelihood function

$$L(\theta) = \prod_i P(D_i; \theta)$$

- Take the log

$$l(\theta) = \log L(\theta) = \sum_i \log P(D_i; \theta)$$

- Solve the optimization problem:

$$\theta_{MLE} = \text{argmax}_\theta \, l(\theta)$$

*This is equivalent to minimizing negative log-likelihood loss

$$\text{argmin}_\theta - l(\theta)$$

Back to Linear Regression

- **Assumption**: target variable y and the input $\mathbf{x}$ are related as follows:

$$y = \mathbf{w}^T\mathbf{x} + \epsilon$$

where $\epsilon \sim N(0, \sigma^2)$, i.e., it is Gaussian noise

- Equivalently we have: $y|\mathbf{x} \sim N(\mathbf{w}^T\mathbf{x}, \sigma^2)$
- Given a set of observed $\mathbf{x}$ and y pairs: $(\mathbf{x}_1, y_1)$, $(\mathbf{x}_2, y_2)$ …, $(\mathbf{x}_N, y_N)$, **independently and identically distributed (IID),** use maximum likelihood estimation to estimate $\mathbf{w}$
- **Conditional Likelihood function:** given the inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, what is the probability of observing target outputs $y_1, y_2, \ldots, y_N$?

$$L(\mathbf{w}) = P(y_1, \ldots, y_N|\mathbf{x}_1, \ldots, \mathbf{x}_N; \mathbf{w}) = \prod_{i=1}^{N} P(y_i|\mathbf{x}_i; \mathbf{w})$$

Maximum likelihood estimation of $\mathbf{w}$

We will now work with log of the likelihood:

$$l(\mathbf{w}) = \log \prod_{i=1}^{N} P(y_i|\mathbf{x}_i; \mathbf{w})$$

$$= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} \exp \frac{-(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}$$

$$= -N \log \sqrt{2\pi}\sigma + \sum_{i=1}^{N} \frac{-(y_i - \mathbf{w}^T\mathbf{x}_i)^2}{2\sigma^2}$$

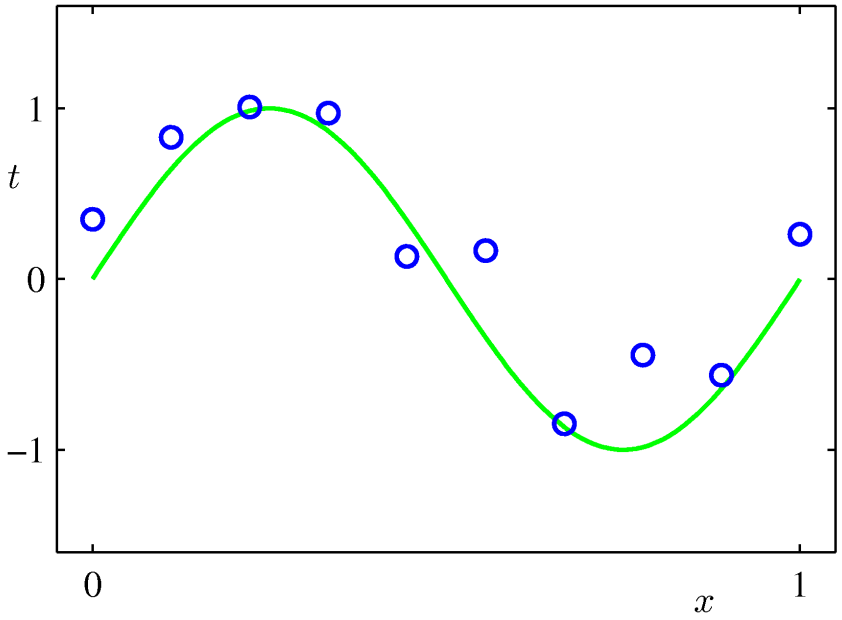
$$\arg\max_{\mathbf{w}} l(\mathbf{w}) = \boxed{\arg\min_{\mathbf{w}} \frac{1}{2} \sum_{i=1}^{N} (y_i - \mathbf{w}^T\mathbf{x}_i)^2}$$

Maximum Likelihood Estimation under Gaussian noise leads to Sum of Squared Error objective

What we have seen so far

- We introduced linear regression
  - Goal: learn a linear function $\hat{y}(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$
  - Approach:
    - Choose a loss function: SSE
    - Optimize the loss function (Gradient descent, or analytically solving the optimization problem)
- What if the relationship between $\mathbf{x}$ and y is not linear? For example, quadratic or cubic?
  - Linear regression can still be used
  - All we need is to introduce nonlinear features

- Revisit example: Polynomial Curve Fitting

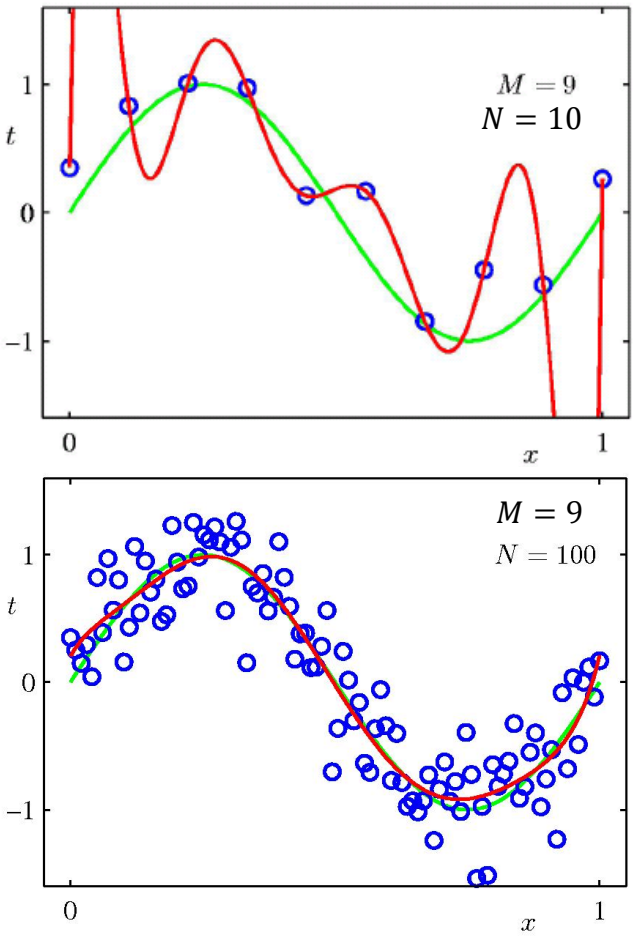


$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

- In this example, there is only one feature x. We learn a function of M-order polynomial
- Alternatively, we could view this as learning a linear model considering $(1, x, x^2, \ldots, x^M)$ as the features.
- Note that this new feature space is derived from the original input x
- We refer to such derived features as **the basis functions**

- In practice, we don’t know the true underlying function
- One strategy is to include all possible inputs, and a large set of basis functions (interaction terms, higher order terms etc.) and let the data determine what features and terms are important
- With increased complexity, this inevitably lead to potential overfitting

Over-fitting issue



- What can we do to curb over-fitting
  - Use less complex model (but what about underfitting?)
  - Use more training examples (who is going to pay for that?)
  - **Regularization**
    - Introducing a penalty term that increases with complexity
    - There has to be sufficient gain in the loss to justify the increased complexity
    - What penalty term?

In linear regression, overfitting can often be characterized by large weights (volatile functions)

	M = 0	M = 1	M = 3	M = 9
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Regularized Linear Regression

- Consider the following loss function:

$$L_D(\mathbf{w}) + \lambda L_R(\mathbf{w})$$

Data term + Regularization term (penalize complex models)

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\sum_{j=1}^{M}|w_j|^q$$

Encourage small weight values

	M = 0	M = 1	M = 3	M = 9
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18

L2 Regularized Linear Regression

- With the MSE loss and a **quadratic regularizer**, we get

$$\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\mathbf{w}^T\mathbf{w}$$

Remeber X= (N*d), w + (d*

which is minimized by $\mathbf{w} = (N\lambda\boldsymbol{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T Y$

Compared to un-regularized solution:

$$(\mathbf{X}^T\mathbf{X})^{-1} \rightarrow (N\lambda\boldsymbol{I} + \mathbf{X}^T\mathbf{X})^{-1}$$

With high dimensional data and small training set, $\mathbf{X}^T\mathbf{X}$ may not be full rank. Regularization adds a diagonal matrix to $\mathbf{X}^T\mathbf{X}$, introducing numerical stability

- λ: regularization parameter, controls the trade-off between model complexity and the fit to the data
  - Larger λ encourages simple model (driving more elements of $\mathbf{w}$ to 0)
  - Small λ encourages better fit of the data (driving MSE to zero)

Regularizations is equivalent to constraining the solution

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda \sum_{j=1}^{M}|w_j|^q$$

Is equivalent to

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

$$\text{subject to} \quad \sum_{i=1}^{M}|w_i|^q \leq \epsilon$$

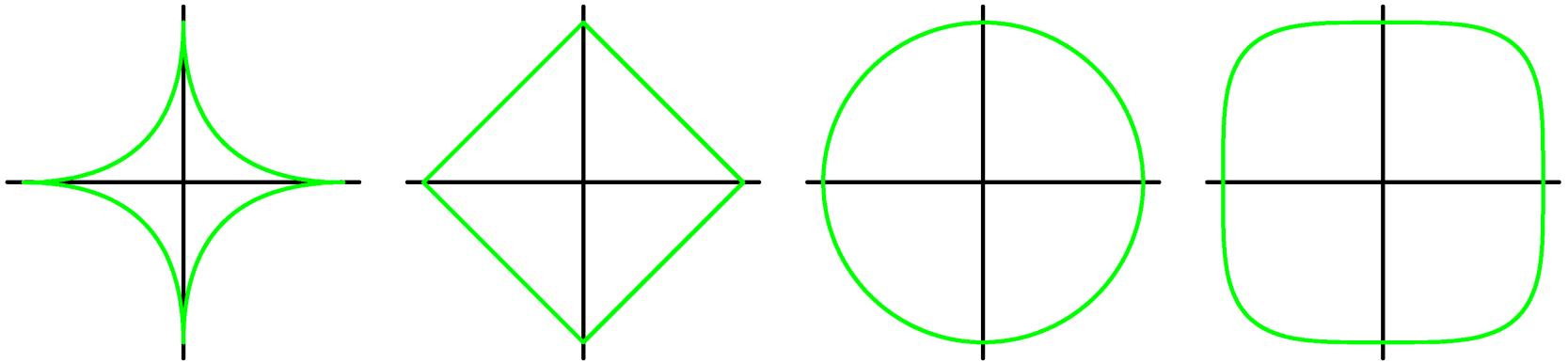
for some ϵ, where for each unique value of λ (↑), there is a corresponding ϵ (↓)

A good explanation of this equivalence is provided here:
http://math.stackexchange.com/questions/335306/why-are-additional-constraint-and-penalty-term-equivalent-in-ridge-regression

Visualizing the constraints for different regularization terms

The constraint $\sum_{i=1}^{M} |w_i|^q \leq \epsilon$ leads to different shapes based on q

Example in 2-d



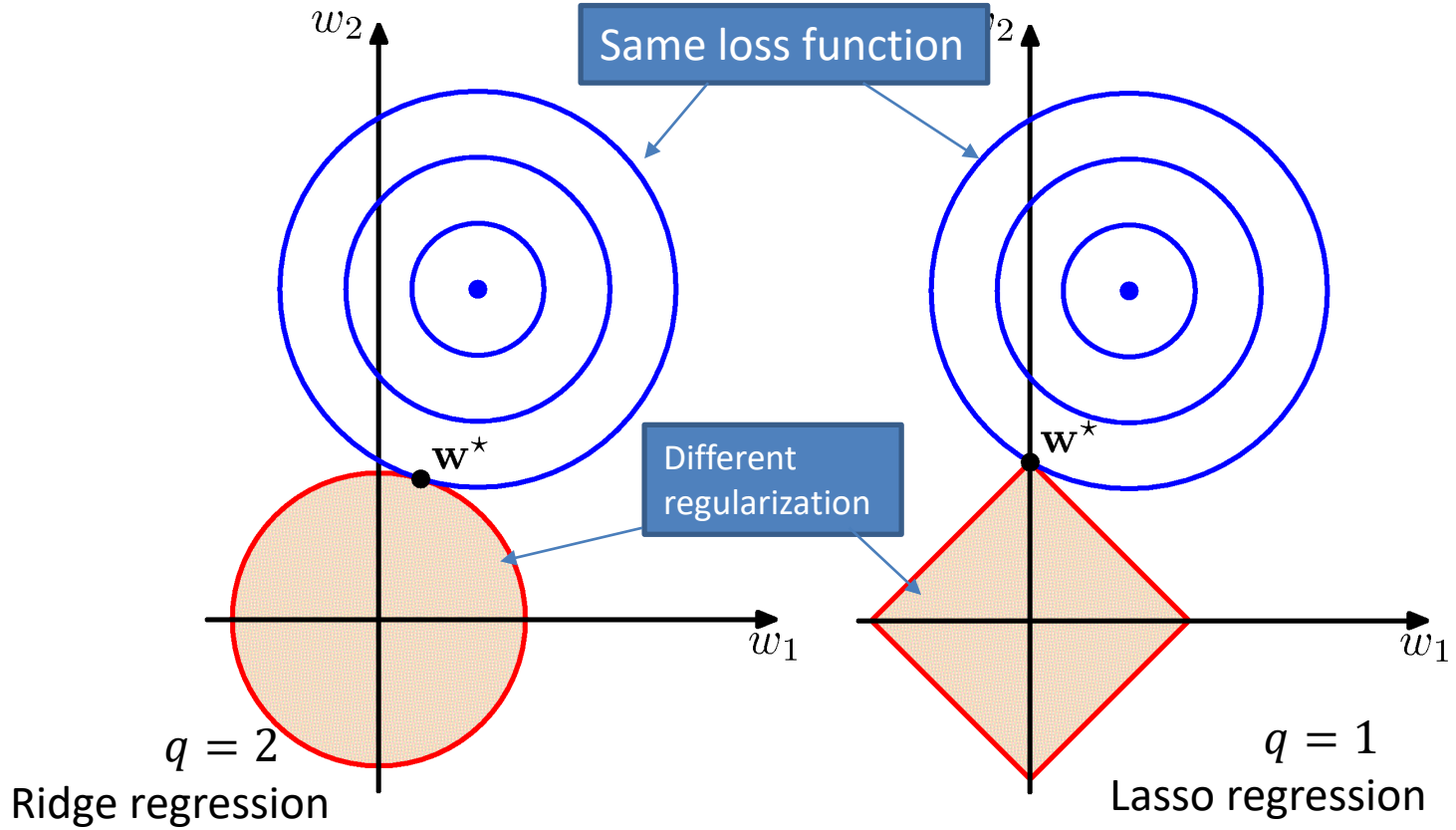
$q = 0.5$ $q = 1$ $q = 2$ $q = 4$

Lasso, L_1 Ridge, L_2

Shape is determined by q, size determined by λ

Constrain shape impacts solution

- Lasso ($q = 1$) tends to generate sparser solutions (many weights shrink to zero) than a quadratic regularizer ($q = 2$, often called ridge regression).



Commonly used regularizors

- L-2 regularization $\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \lambda\sum_{j=1}^{M} {w_j}^2$

  Poly-time close-form solution
  Curbs overfitting but does not produce sparse solution

- L-1 regularization $\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \sum_{j=1}^{M}|w_j|$

  Poly-time approximation algorithm
  Sparse solution – potentially many zeros in $\mathbf{w}$

- L-0 regularization $\frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2 + \sum_{j=1}^{M} I(w_j \neq 0)$

  Seek to identify optimal feature subset
  NP-complete problem!

Summary

- Linear regression with SSE objective
  - Gradient descent algorithm for optimization
  - Closed-form solution by solving the normal equation
  - Maximum likelihood estimation assuming IID Gaussian noise with zero mean
- Maximum likelihood estimation: a basic probabilistic principle for setting up learning objectives
- Using basis functions allows us to learn nonlinear functions
- Controlling over-fitting by Regularization
  - The regularization term is equivalent to adding a constraint to the parameters
  - In effect, Quadratic regularization for linear regression adds a diagonal matrix to $\mathrm{X}^T\mathrm{X} \rightarrow (\lambda I + \mathrm{X}^T\mathrm{X})$ – it also serve the purpose of conditioning $X^T X$ when it is not full rank (not invertible)
  - LASSO regularization encourages sparse solution