Machine Learning and Data Mining

Lecture 2.2: Basis Functions, Regularization, and Logistic Regression

CS 434

Question Break!

What best summarizes your current relationship with homework one?

A I've finished either the math part or the programming part (or both)

B I've read over it and think I know how to proceed with the work

C I haven't really looked at it yet

D I've read over it and have no idea where to even begin with some questions

RECAP

From Last Lecture

Your First Probabilistic Learning Algorithm: MLE

The intuitive answer is that 3/5 "fits the data" the best. We'll formalize that notion this lecture.

Maximum Likelihood Estimation – Find parameters that make the observed data most likely.

1. Assume a probabilistic model of how the data was generated $x \sim P(x; \theta)$ parameterized by some set of parameters θ

2. Find $\hat{\theta}_{MLE}$ that maximizes the probability (or likelihood) of generating the training data under the probabilistic model.

Why MLE?

- Often leads to "natural" or intuitive parameter estimates
- MLE is optimal if model class is correct (e.g. Normal model for normally distributed data)

Your Second Statistical Learning Algorithm: MAP

Maximum A Posteriori – Find parameters that make the observed data most likely but consider a prior over the parameters.

1. Assume a prior distribution over θ, $P(\theta)$

2. Assume a probabilistic model of how the data is generated:
   -- parameter $\theta \sim P(\theta)$ and then data $x \sim P(x|\theta)$

3. Find $\hat{\theta}_{MAP}$ that maximize the posterior $P(\theta|D) \propto P(D|\theta)P(\theta)$

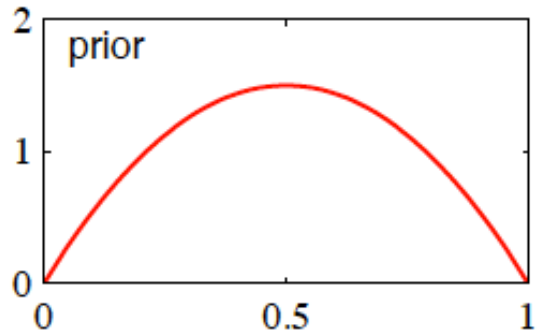
$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Why MAP?

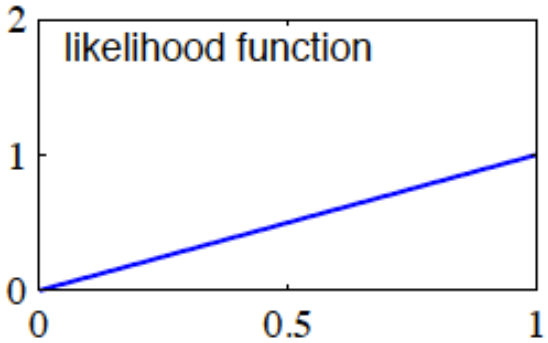
- Rigorous framework to combine observations (likelihood) with beliefs (prior)

Effect of Prior on Coin Flip

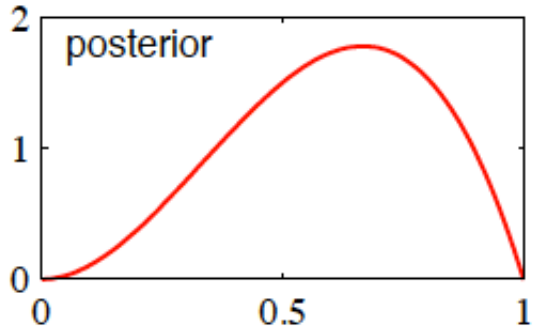
- **Prior = Beta(2,2)**
  - $\theta_{prior} = 0.5$



- **Dataset = {H}**
  - $L(\theta) = \theta$
  - $\theta_{MLE} = 1$



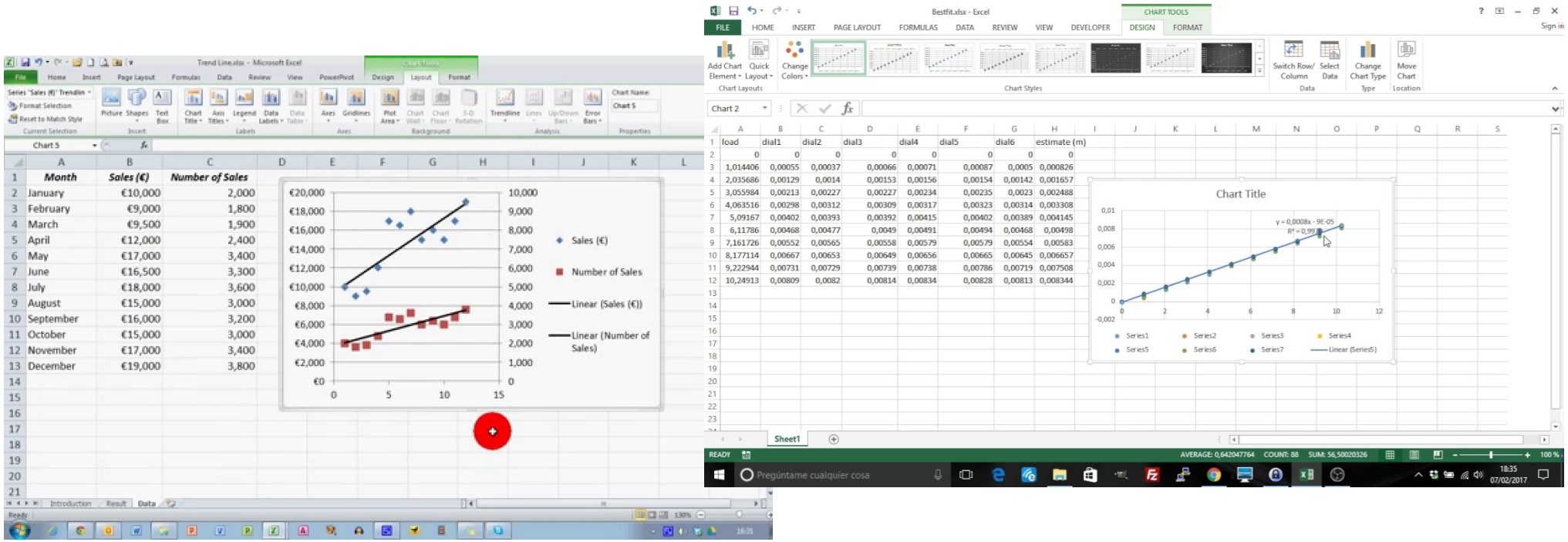
- **Posterior = Beta(3,2)**
  - $\theta_{MAP} = (3-1)/(3+2-2) = 2/3$



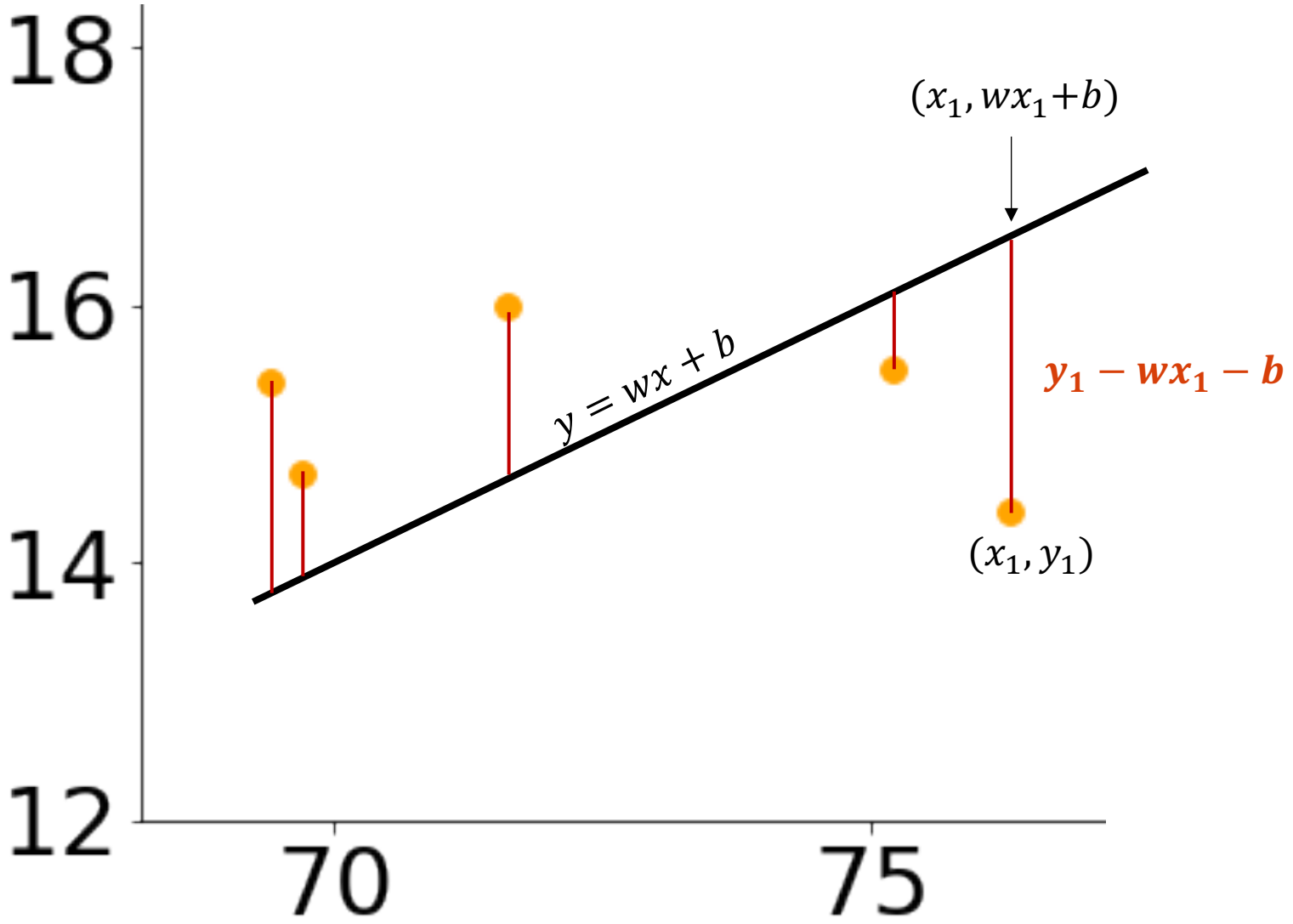
Linear Regression

Regression → We are predicting some continuous output y

Linear → We assume there is a linear relationship between input x and output y



Visualizing Sum of Squared Error



$$SSE(w,b) = \sum_{i=1}^{n} (y_i - wx_i - b)^2$$

Data from: The Song of Insects by Dr.G.W. Pierce, Harvard College Press

Multidimensional Linear Regression

1) Matrix/vector computation of sum of squared errors:

$$SSE(\boldsymbol{w}) = (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w})$$

2) Set equal to zero vector and solve:

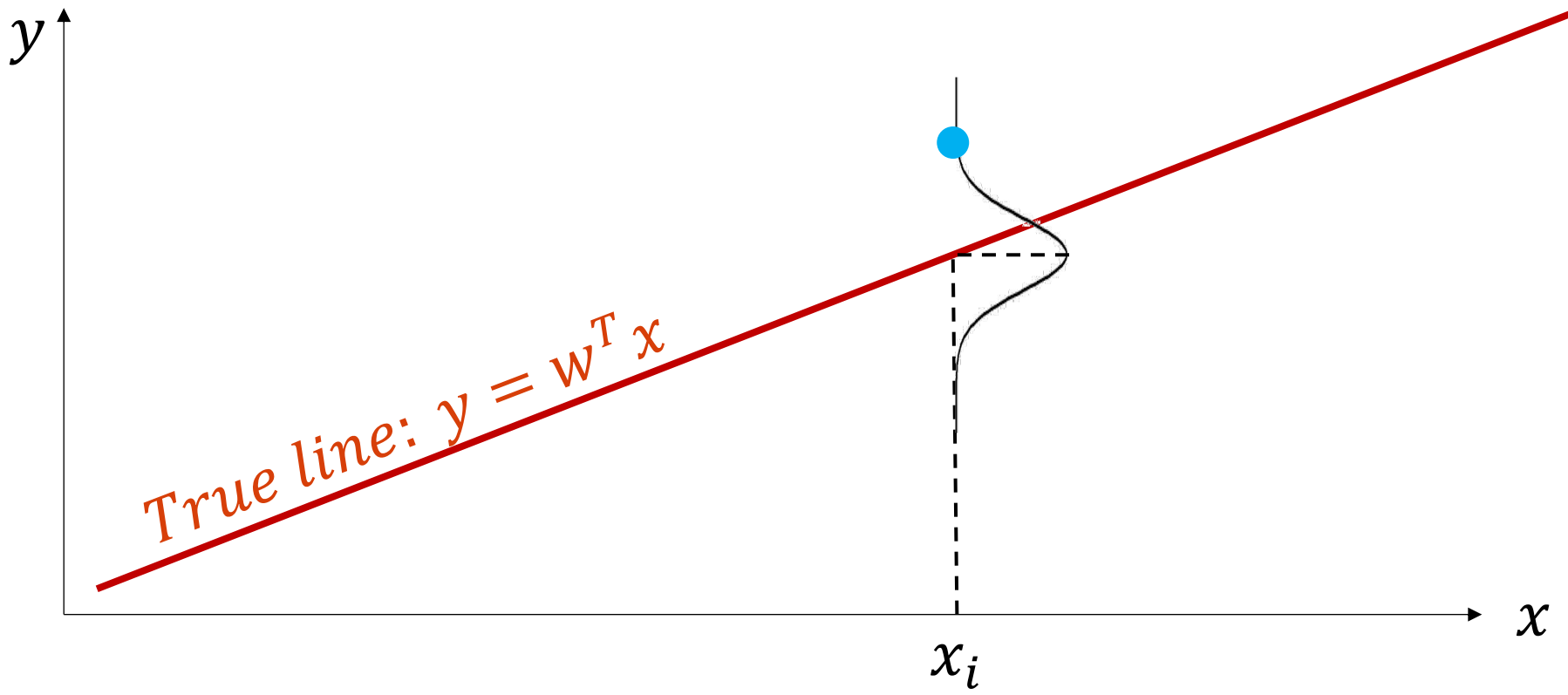
$$2X^T(\boldsymbol{y} - X\boldsymbol{w}) = \vec{\mathbf{0}}$$

$$\Longrightarrow X^T X\boldsymbol{w} = X^T\boldsymbol{y}$$

$$\Longrightarrow \boldsymbol{w}^* = (X^T X)^{-1}X^T\boldsymbol{y}$$

Our "generative story" for this data:

$$y_i = w^T x_i + \mathcal{N}(0, \sigma)$$



Okay. Still not seeing why we care? Let's start doing MLE.

Dataset: Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ assume the above conditional probability.

Model assumption: $$y_i = w^T x_i + \mathcal{N}(0, \sigma) \implies P(y_i | x_i, w) = \mathcal{N}(w^T x_i, \sigma)$$

Write out likelihood of the training data as a function of parameters θ:

$$\mathcal{L}(\theta) = P(D \mid \theta) = \prod_{i=1}^{N} P(y_i \mid w) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i - \boldsymbol{w^T x_i})^2}{2\sigma^2}}$$

Yikes. Let's apply a log and write the log-likelihood to clean this up:

$$\mathcal{LL}(\theta) = P(D \mid \theta) = N \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2} \sum_{i=1}^{N} (y_i - w^T x_i)^2$$

Let's view this from a different angle

Wait... this part is looking familiar...

$$\mathcal{LL}(\boldsymbol{\theta}) = N \log\left(\frac{1}{\sigma\sqrt{2\pi}}\right) - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2$$

How do we maximize this log-likelihood with respect to w?

Find w^* such that $\sum_{i=1}^{N}(y_i - \boldsymbol{w}^T\boldsymbol{x_i})^2$ is as small as possible

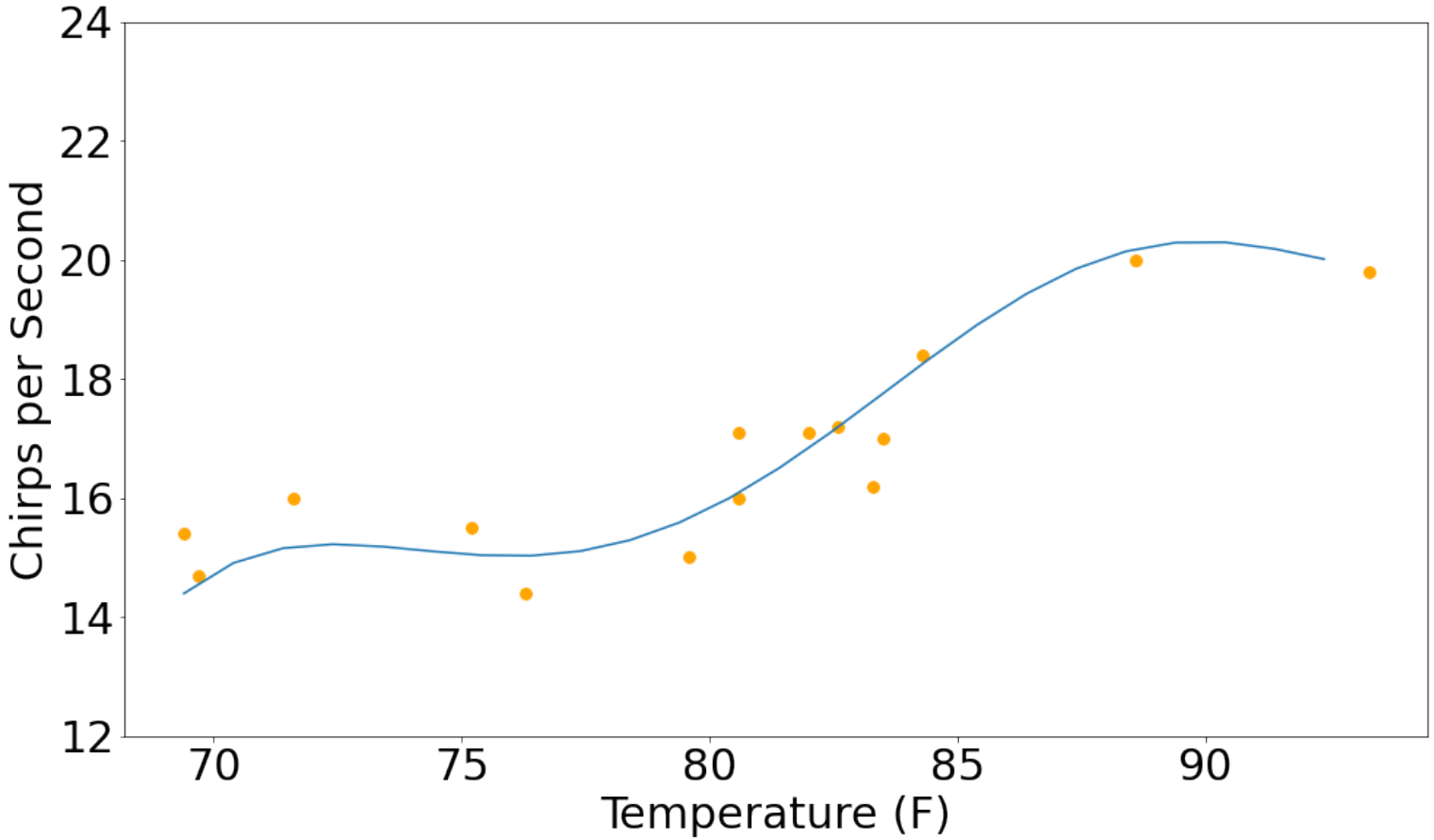
AKA find weights that minimize the sum of squared error!

Linear regression is just MLE of a linear model with Gaussian noise!

Teaser

You: It's just fitting straight lines. Boring.

Me: I fit this with least-squares linear regression too. Tune in next time.



Today's Learning Objectives

Be able to answer:

- Wrapping up a few details about OLS.
- How can we use least-squares regression to fit more complex functions?
  - What is polynomial regression?
  - More generally, how can we learn linear functions over basis functions?
- What is regularization?
- What is logistic regression?

A Note on Nomenclature

Last time we were talking about linear regression and proposed the objective:

$$SSE(\boldsymbol{w}) = (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w})$$

This is often referred to as "ordinary least squares" or OLS regression. Usually when someone says "linear regression" this is what they mean.

Invertibility of $X^T X$

Solution to Ordinary Least Squares (OLS) where X is a n-by-d data matrix:

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

Problem: Is $X^T X$ always invertible? Nope. To be invertible, matrix must be full rank (i.e., have no linearly dependent columns).

$$\boldsymbol{Rank}(\boldsymbol{X^T X}) = \boldsymbol{Rank}(\boldsymbol{X})$$

If X is full rank, then $X^T X$ is too.

If n < d then Rank($X^T X$) < d.
If I have fewer data points than dimensions, no inverse.

$$\boldsymbol{Rank}(\boldsymbol{X^T X}) = \boldsymbol{Rank}(\boldsymbol{X}) \leq \boldsymbol{min}(\boldsymbol{n}, \boldsymbol{d})$$

Invertibility of $X^T X$

Solution to Ordinary Least Squares (OLS) where X is a n-by-d data matrix:

$$\boldsymbol{w}^* = \underbrace{(X^T X)^{-1} X^T}\boldsymbol{y}$$

Moore-Penrose Psuedo-Inverse X^+

Moore-Penrose Pseudo-Inverse is **always defined** and exactly equal to the above when X has full rank. Still the correct solution otherwise (proof)

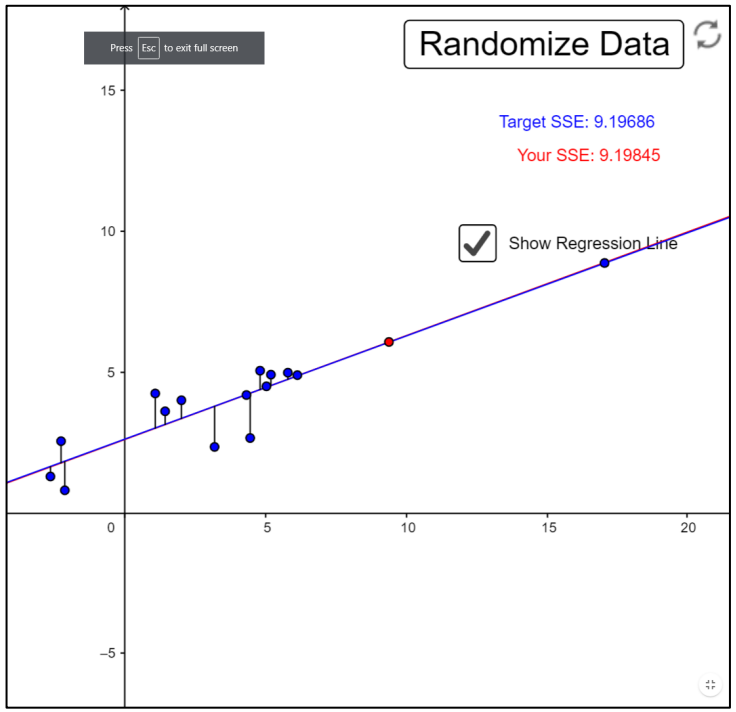
$$\boldsymbol{w}^* = X^+ \boldsymbol{y}$$

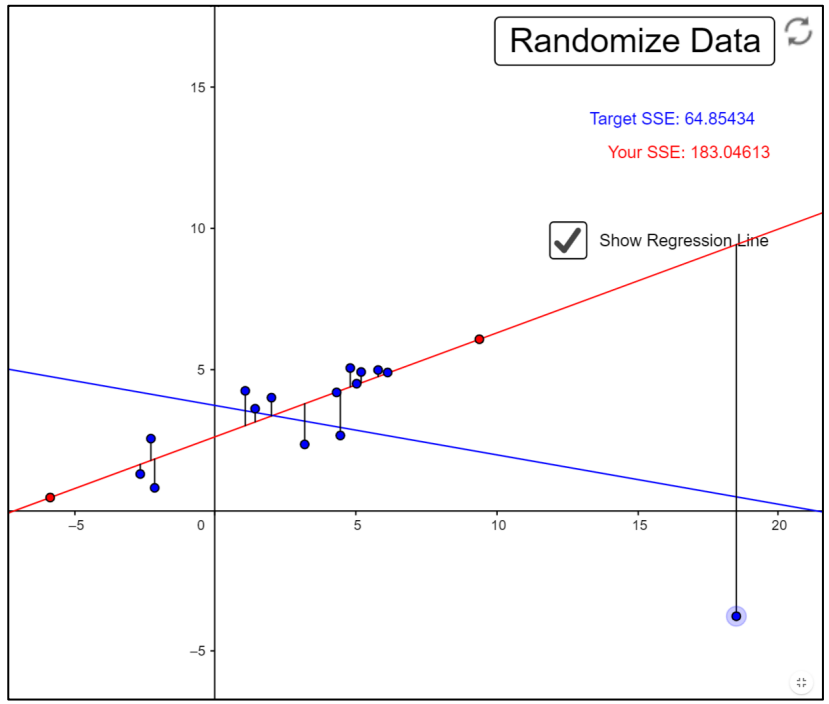
The Psuedo-Inverse solution tends to be more stable in general to compute.

In numpy: numpy.linalg.pinv(X)

Robustness of Ordinary Least Squares (OLS)

Minimizing sum of squared error is *very* sensitive to outliers.





Probabilistic view: very little density in the tails of Gaussians.

https://www.geogebra.org/m/xC6zq7Zv

Question Break!

In general, how many serious outliers are necessary to alter the solution to least-squares linear regression?

A 20% of the dataset

B 5% of the data

C A single example if its bad enough

D >50% of the dataset

Today's Learning Objectives

Be able to answer:

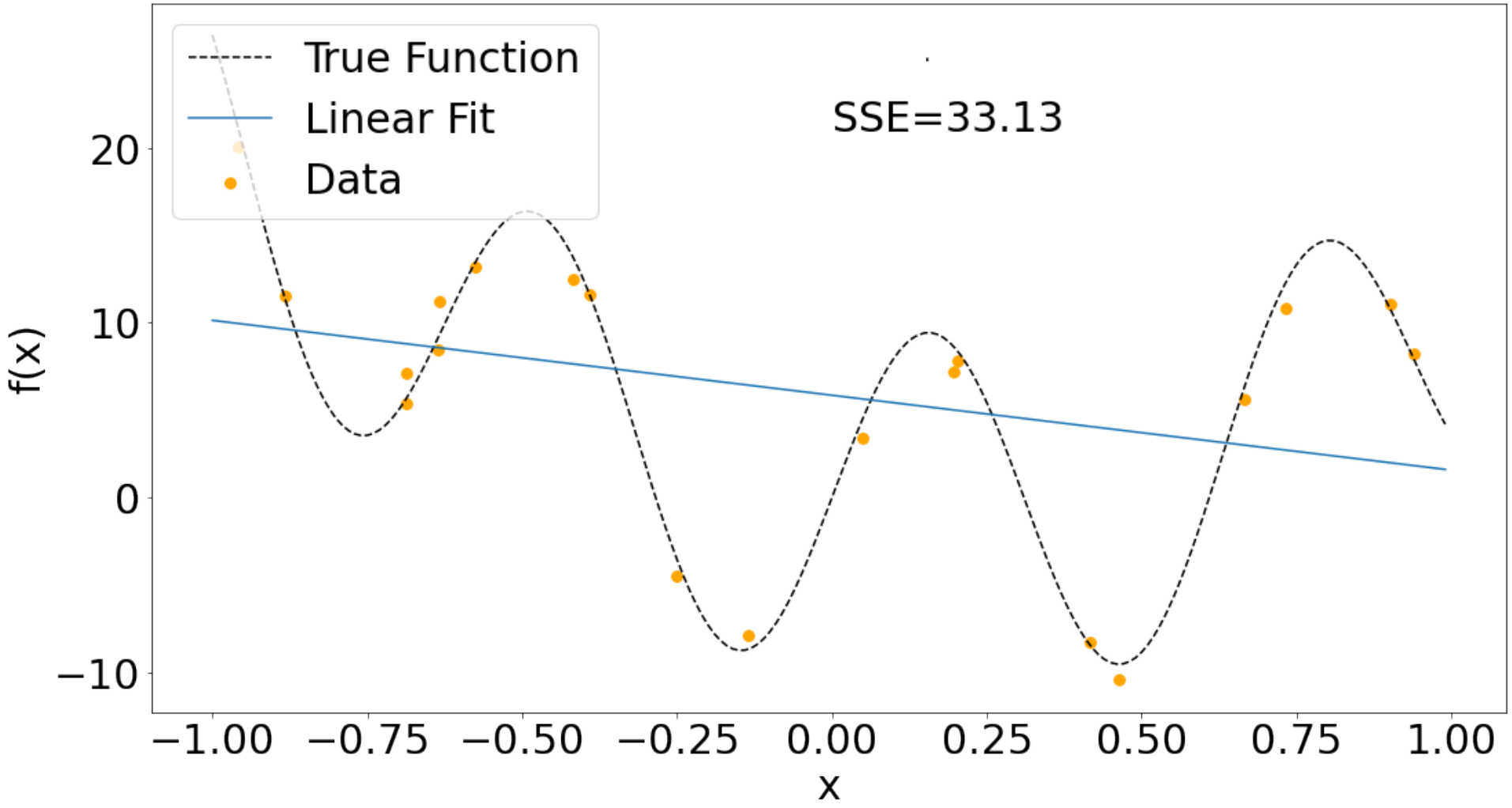
- ~~Wrapping up a few details about OLS.~~

- How can we use least-squares regression to fit more complex functions?
  - What is polynomial regression?
  - More generally, how can we learn linear functions over basis functions?

- What is regularization?

Beyond Lines but Still Linear?

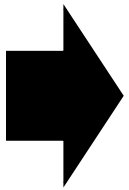
Fitting lines seems cool but many, many functions of interest aren't lines.



Beyond Lines but Still Linear?

Idea: Solve a linear regression problem in a feature space that is non-linear in the original input! For example, could add a x^2 term.

X		Y
	x	Y
1	-0.25	-4.47
1	0.9	11.08
1	0.46	-10.44
1	0.2	7.19
1	-0.69	7.08
1	-0.69	5.38
1	-0.88	11.55
1	0.73	10.85
1	0.2	7.85
1	0.42	-8.31



X'			Y
	x	x^2	Y
1	-0.25	0.06	-4.47
1	0.9	0.81	11.08
1	0.46	0.21	-10.44
1	0.2	0.04	7.19
1	-0.69	0.48	7.08
1	-0.69	0.48	5.38
1	-0.88	0.77	11.55
1	0.73	0.53	10.85
1	0.2	0.04	7.85
1	0.42	0.18	-8.31

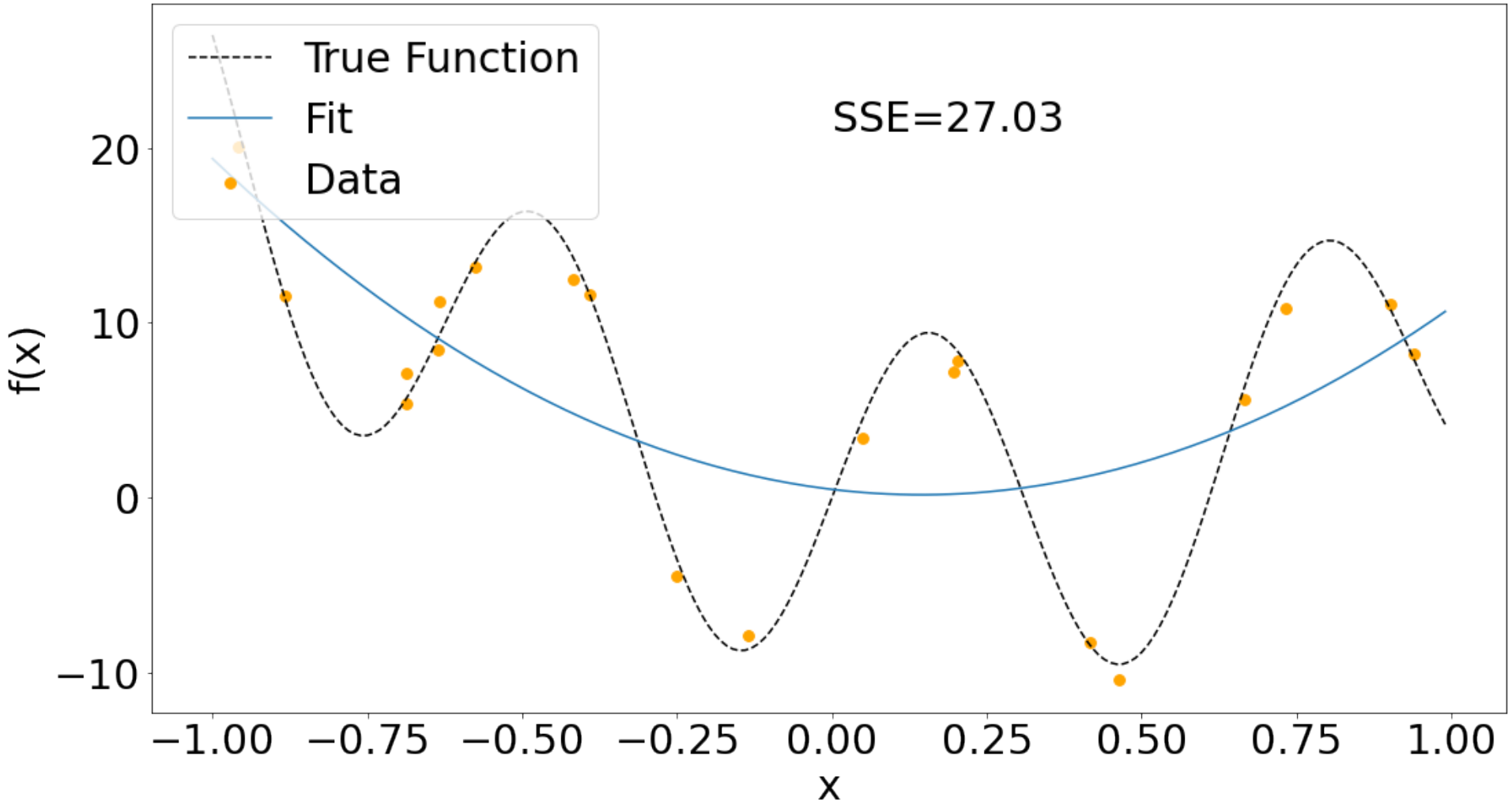
Solve for w such that:

$$y_i = w_0 1 + w_1 x_i + w_2 x_i^2$$

Linear function of non-linear transformations of x

$$y_i = \begin{bmatrix} 1, x_i, x_i^2 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Beyond Lines but Still Linear?



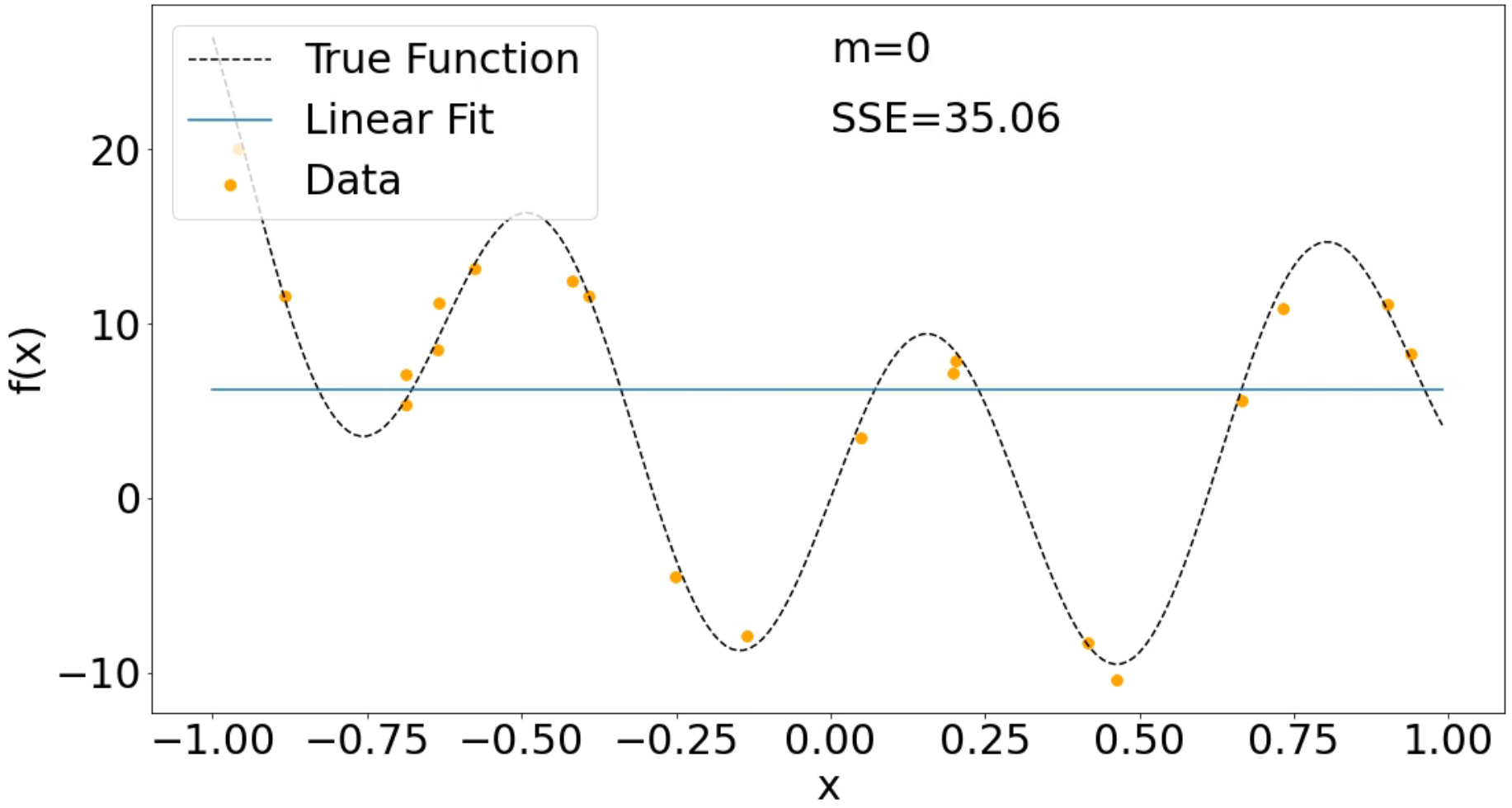
More general example: m-order polynomial regression in the 1d case.

$$\Phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^m \end{bmatrix}^T \qquad \boldsymbol{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}_{n \times 1} \Rightarrow \boldsymbol{X'} = \begin{bmatrix} \boldsymbol{\Phi}(x_1) \\ \boldsymbol{\Phi}(x_2) \\ \vdots \\ \boldsymbol{\Phi}(x_n) \end{bmatrix}_{n \times m}$$

Want to find a weight vector **w** such that the sum of squared error is minimized:

$$SSE(\boldsymbol{w}) = \sum_{i=1}^{n} (y_i - \Phi(x_i)\boldsymbol{w})^2 \qquad \Longrightarrow \boldsymbol{w}^* = X'^{+}\boldsymbol{y}$$

Polynomial Regression



Number of data points n=20

What just happened? (Ignore the overfitting at the end for now)

We only had one **real** feature **x**, but we transformed it into a vector of non-linear "derived" features $(\mathbf{1}, \boldsymbol{x}, \boldsymbol{x^2}, \boldsymbol{x^3}, \ldots)$ or "basis functions". Then we solved the linear regression problem in this non-linear feature space.

To predict y for a new point x, just do $\phi(x)w$!

What about when the input features are already multidimensional?

Let's generalize this further.

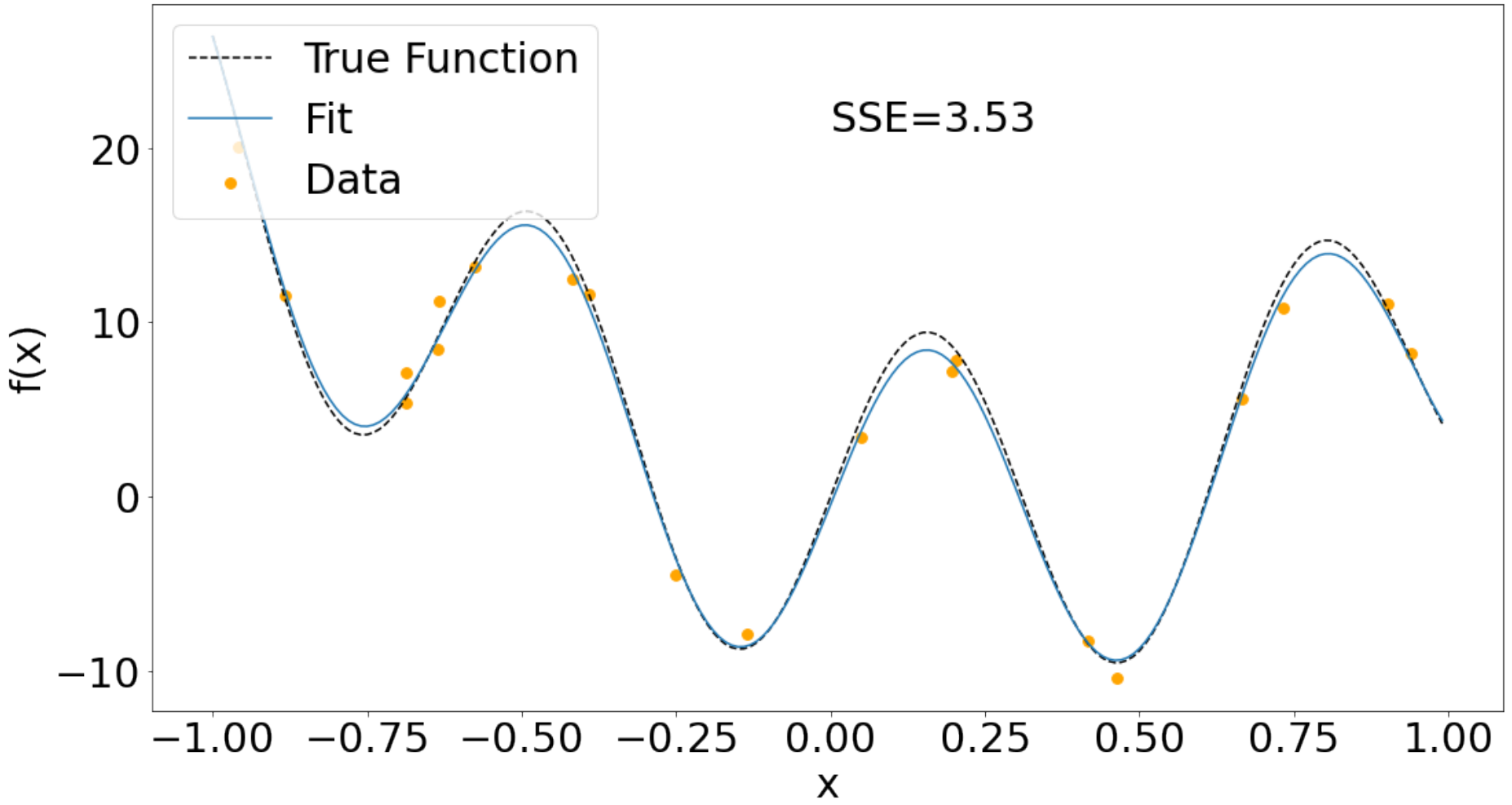
Each data point has d features. Let $\boldsymbol{\Phi}$: $\mathbb{R}^d \rightarrow \mathbb{R}^{d'}$ be our basis function. Can make any arbitrary choice we want here based on how the data looks.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^{\boldsymbol{T}}_{1\times d} \qquad \boldsymbol{\Phi}_{\mathrm{A}}(x) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1 x_2 \\ \sin(x_1) \\ \cos(x_2) \end{bmatrix}^{\boldsymbol{T}} \qquad \boldsymbol{\Phi}_{\mathrm{B}}(x) = \begin{bmatrix} 1 \\ x_1 x_2 \\ x_1 x_3 \\ x_1 x_4 \\ \vdots \\ x_n x_n \end{bmatrix}^{\boldsymbol{T}} \qquad \boldsymbol{\Phi}_{\mathrm{C}}(x) = \begin{bmatrix} 1 \\ \prod x_i \\ \sqrt{x_1} \end{bmatrix}^{\boldsymbol{T}}$$

No matter what we choose, the procedure is the same. Replace each $\boldsymbol{x}$ (row) in our data matrix with $\boldsymbol{\Phi}(\mathbf{x})$, then solve the linear regression problem.

Linear Regression over Basis Functions

$$\boldsymbol{\Phi}(x) = [1 \quad x \quad x^2 \quad sin(10x)]$$



Summary of Linear Regression over Basis Functions

Recap:

Linear regression over non-linear basis functions of **x** results in fitting non-linear functions of **x**!

Pros:

- Super simple to implement.
- Can easily add more basis functions.
- Can be solved exactly.

Cons:

- We need some intuitions from the data about what basis function to use.
- Each basis function we add increases the number of columns in our data matrix – we may not have enough data to fit such a complex function!
  - Easy to overfit if our basis gets too big.

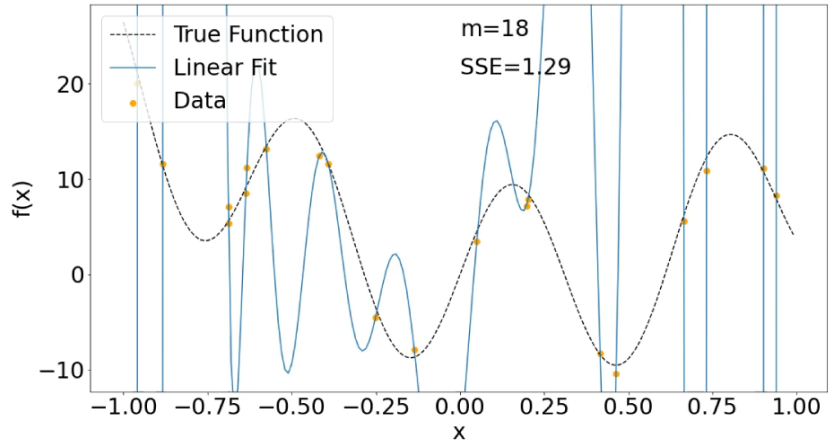
Today's Learning Objectives

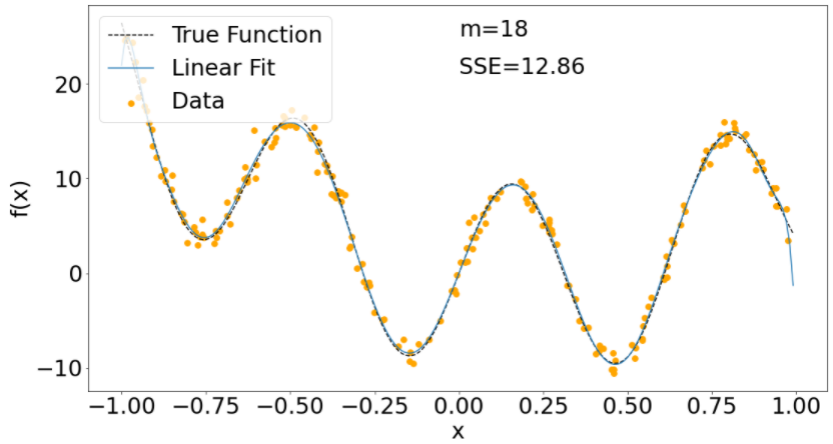
Be able to answer:

- ~~Wrapping up a few details about OLS.~~
- ~~How can we use least squares regression to fit more complex functions?~~
  - ~~What is polynomial regression?~~
  - ~~More generally, how can we learn linear functions over basis functions?~~
- What is regularization?
- What is logistic regression?

What can we do to reduce overfitting?

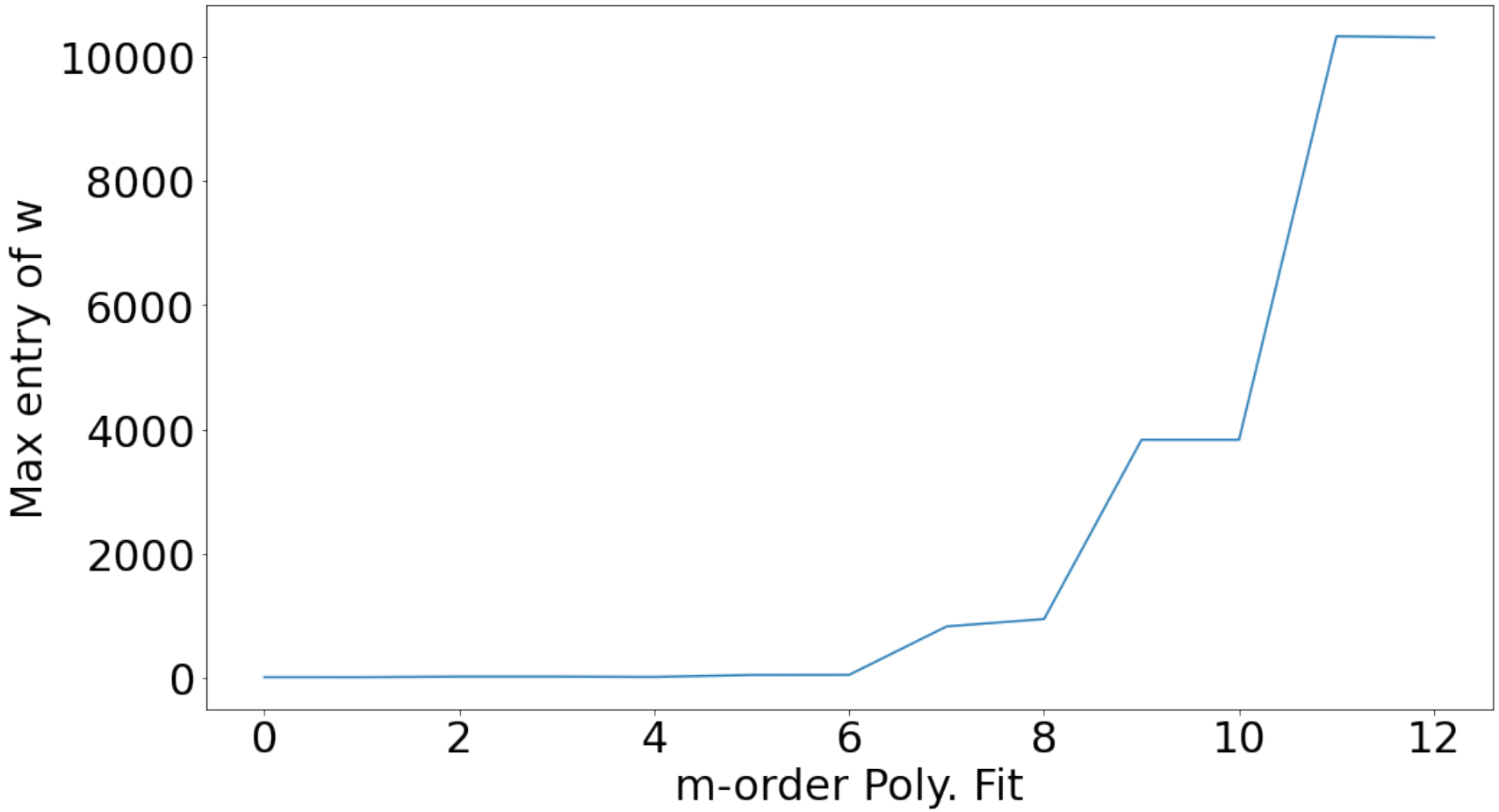
- View it as an estimation error
  - *Solution:* Get more data
- Do model selection to find a simpler model
  - Less complicated model, less data needed.
- **Regularization**
  - Add a *prior* belief that the model should be simple!
  - Okay. But how to encode this?

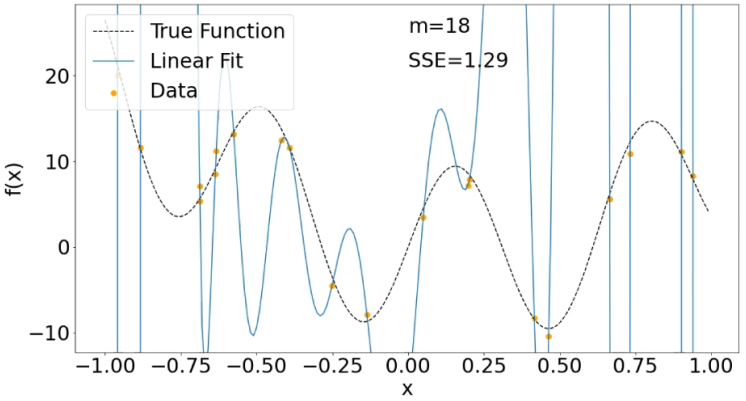


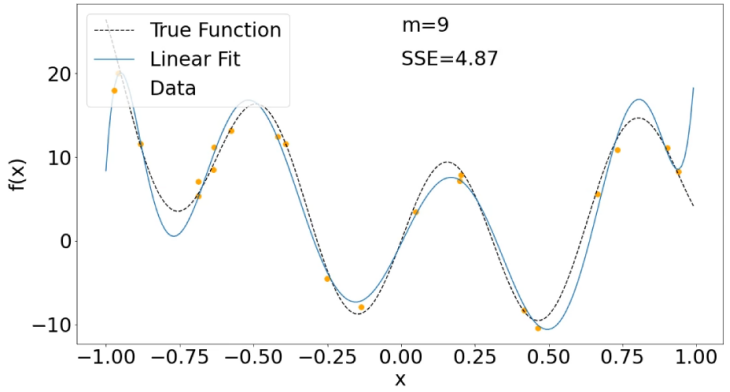


Characterizing Overfitting in Linear Models

In linear models, overfitting can often by characterized by large weights.







Characterizing Overfitting in Linear Models

Intuition 1:

With large weights**, small changes in input** *may* result in **large output changes.**

Intuition 2:

If I set weights to (nearly) zero, I'm dropping dimensionality.

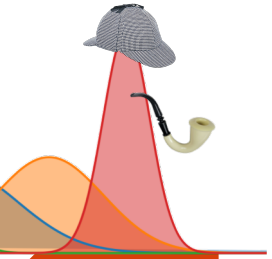
$$y_i = w_0 1 + w_1 x_i + 0 x_i^2$$

Zero-valued weights reduce the complexity of our model.

Regularization by Pushing Weights Towards Zero

Seems like a useful prior belief would be that weights should be relatively small while still fitting the data well -- with some tradeoff between the two.

If only we had learned about a rigorous framework for reasoning about prior beliefs….



L2 Regularization

Regularizer View

Add a regularization penalty
$\lambda \boldsymbol{w^T w}$ to the SSE

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w^T w}$$

Bayesian View

Assume a Gaussian prior
$w \sim \mathcal{N}(\vec{0}, \beta I)$

$$w^* = \underset{w}{argmax} \quad logP(D|w) + logP(w)$$

Arrive at the same solution.

L2 Regularization – The Bayesian View

Dataset: Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$ assume the above conditional probability.

Model assumptions: $$y_i = w^T x_i + \mathcal{N}(0, \sigma) \implies P(y_i | x_i, w) = \mathcal{N}(w^T x_i, \sigma)$$

$$\boldsymbol{w} \sim \mathcal{N}(\vec{\mathbf{0}}, \beta \boldsymbol{I}) \quad \Rightarrow \quad P(w_i) = \frac{1}{\beta\sqrt{2\pi}} e^{-\frac{w_i^2}{2\beta^2}}$$

Write out posterior of the training data as a function of parameters:

$$\boldsymbol{P(w|D)} \propto P(D \mid w)P(w) = P(w) \prod_{i=1}^{N} P(y_i \mid w) = \left(\prod_{j=1}^{d} \frac{1}{\beta\sqrt{2\pi}} e^{-\frac{w_j^2}{2\beta^2}} \right) \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i - \boldsymbol{w^T x_i})^2}{2\sigma^2}}$$

L2 Regularization – The Bayesian View

Write out posterior of the training data as a function of parameters:

$$\boldsymbol{P(w|D)} \propto P(D \mid w)P(w) = P(w)\prod_{i=1}^{N} P(y_i \mid w) = \left(\prod_{j=1}^{d} \frac{1}{\beta\sqrt{2\pi}} e^{-\frac{w_j^2}{2\beta^2}}\right)\prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(y_i - \boldsymbol{w^T x_i})^2}{2\sigma^2}}$$

Take log of the posterior to clean things up:

$$logP(w|D) = -\frac{1}{2\beta^2}\sum_{j=1}^{d} w_j^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{N}(y_i - w^T x_i)^2 + \cdots$$

Vectorizing stuff:

$$= -\frac{1}{2\beta^2}\boldsymbol{w^T w} - \frac{1}{2\sigma^2}(\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \cdots$$

L2 Regularization

Regularizer View

Add a regularization penalty
$\lambda \boldsymbol{w^T w}$ to the SSE

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w^T w}$$

Bayesian View

Assume a Gaussian prior
$w \sim \mathcal{N}(\vec{0}, \beta I)$

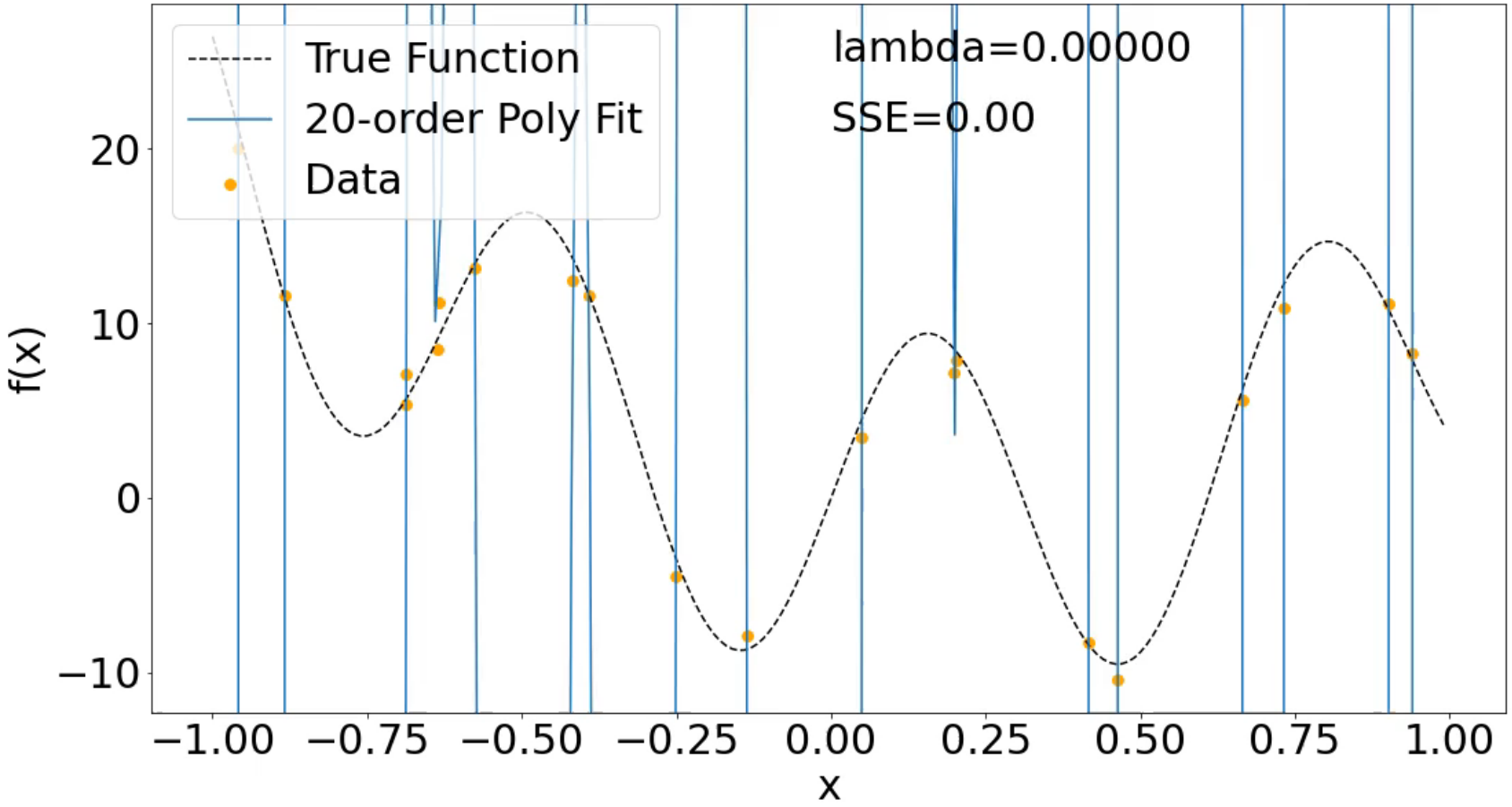
$$w^* = \underset{w}{argmax} \quad logP(D|w) + logP(w)$$

Arrive at the same solution for L2 regularized least squares.

$$\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$

Effect of Regularization – Example 1

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w^T w} \quad \Rightarrow \quad \boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X^T \boldsymbol{y}$$



Question Break!

Given that L2 regularized least squares solves the following:

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \boldsymbol{w}^T\boldsymbol{w}$$

As λ gets very large, the error on the training set approaches $\boldsymbol{y}^T\boldsymbol{y}$

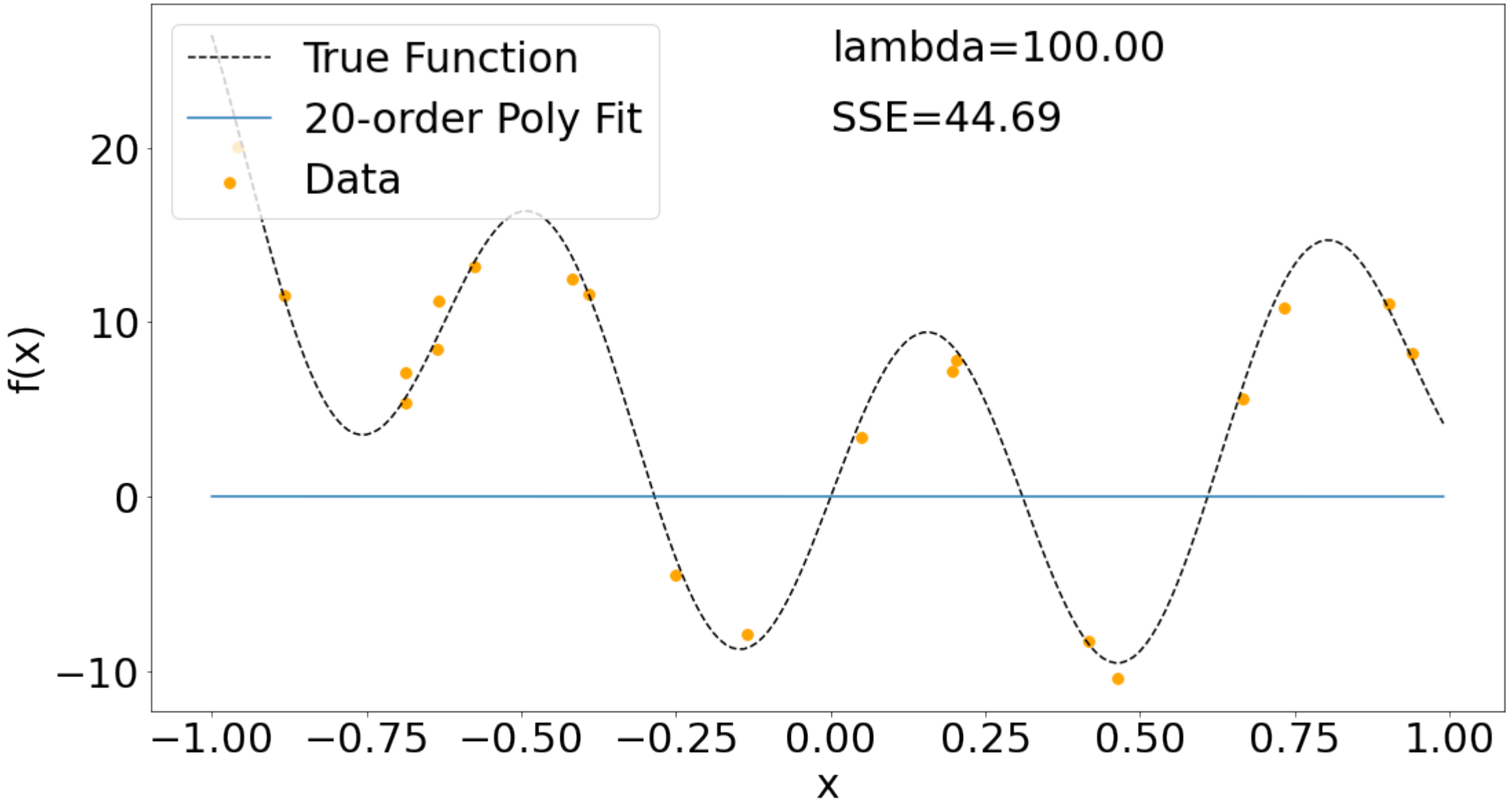
A True and I understand why

B False and I understand why

C True but I'm not sure why

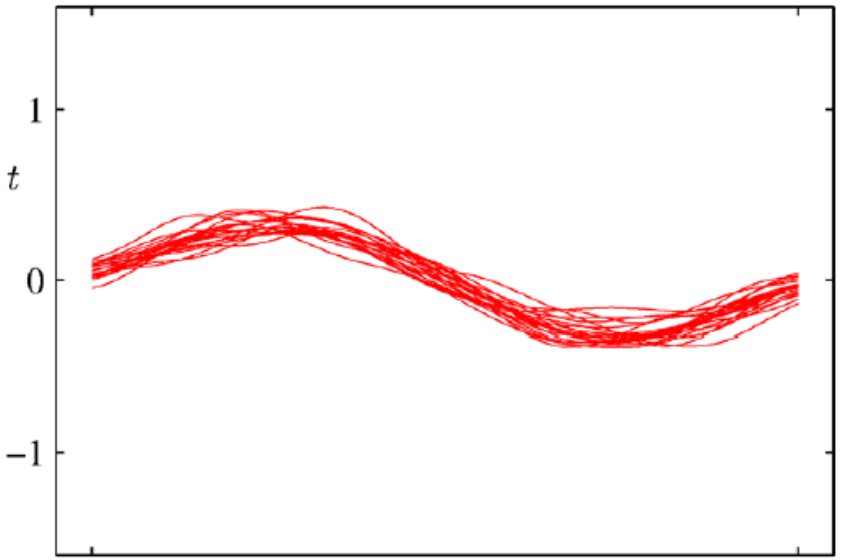
D False but I'm not sure why

Overregularized models may underfit

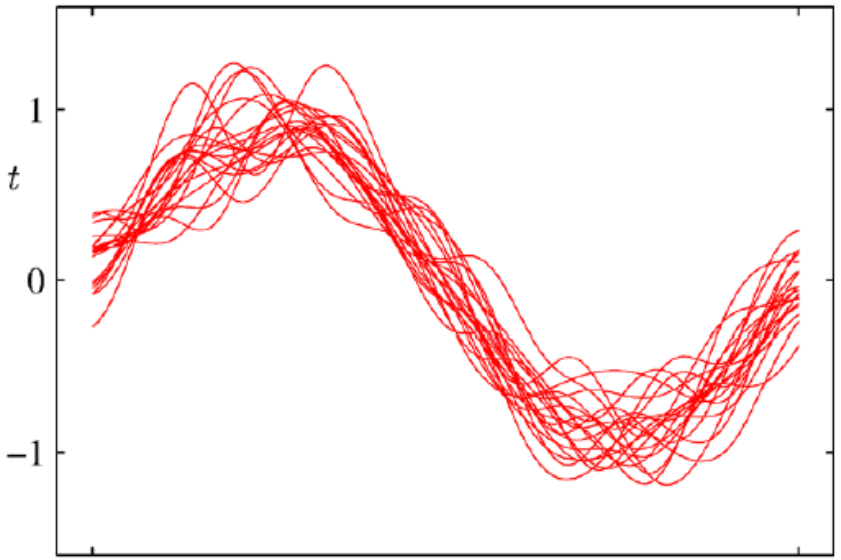


Effect of Regularization – Example 2

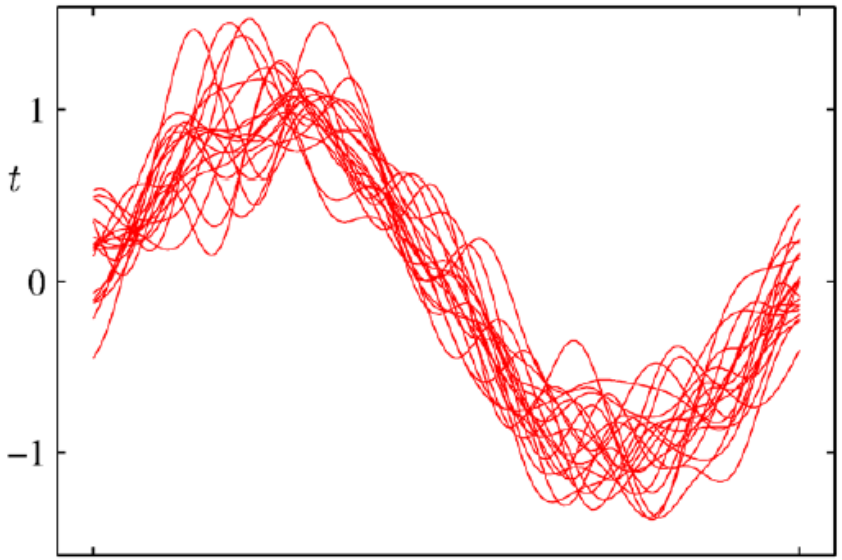
Each red line is the result of a 9^{th} degree polynomial fit on 10 random points noisily sampled from the true curve.



$\lambda \approx 13.46$



$\lambda \approx 0.73$



$\lambda \approx 0.09$

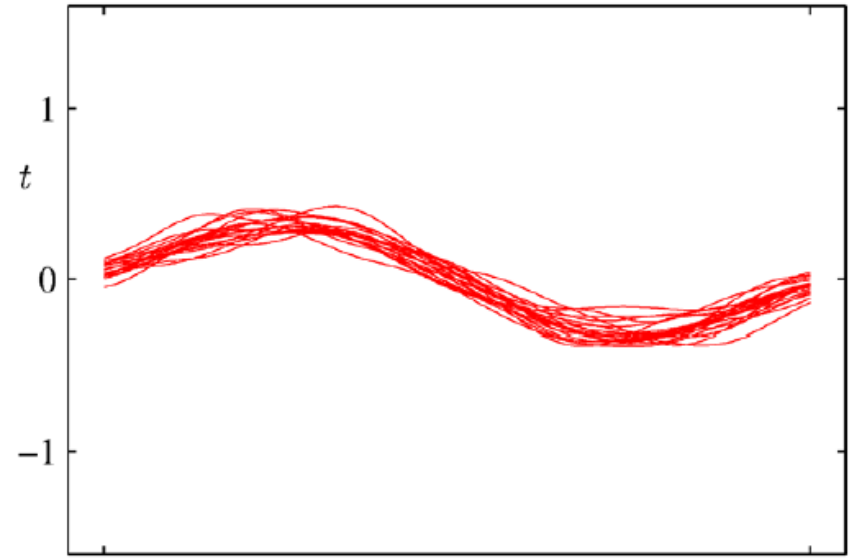
Smaller λ resulted in more complex curves that better fit the data but has a high variance.

Effect of Regularization – Example 2

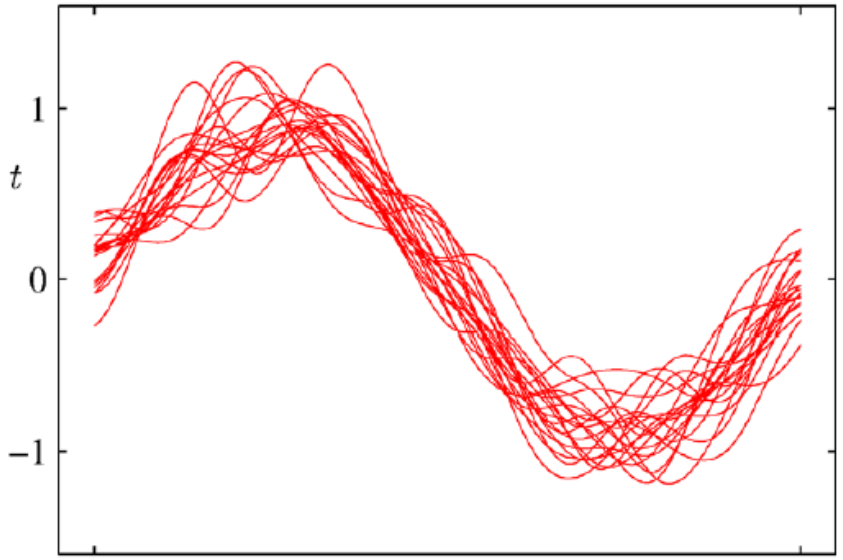
Consider regularization (or priors in general) as trading off between bias and variance.

- No or low regularization has low bias, but high variance.
- Highly regularized models have high bias, but low variance.

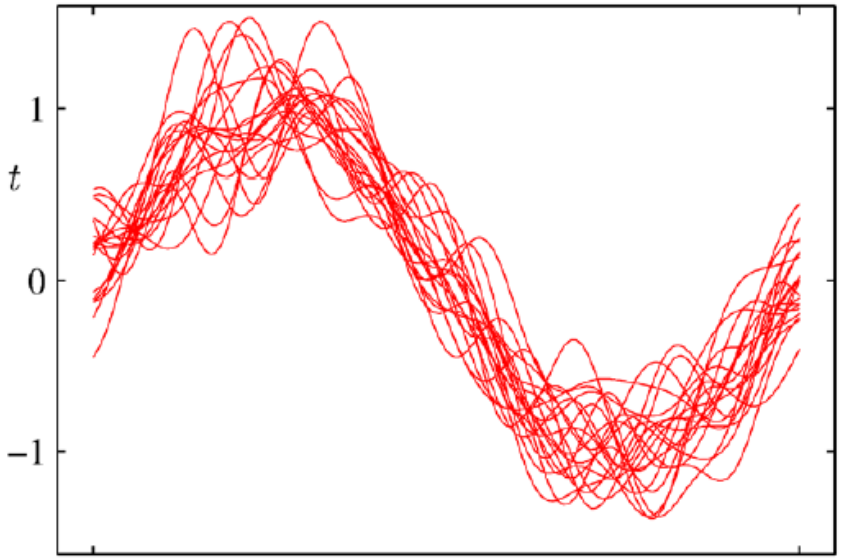
- Often referred to as a "**bias-variance tradeoff**".



$\lambda \approx 13.46$



$\lambda \approx 0.73$



$\lambda \approx 0.09$

Other forms of Regularization

Could consider a whole family of regularization techniques based on different norms.

$$w^* = \underset{w}{argmin} \quad (\boldsymbol{y} - X\boldsymbol{w})^T(\boldsymbol{y} - X\boldsymbol{w}) + \lambda \|w\|_p^p$$

where $\|w\|_p^p = \sum_{i=1}^{d} |x_i|^p$

Just Minkowski norm raised to power p

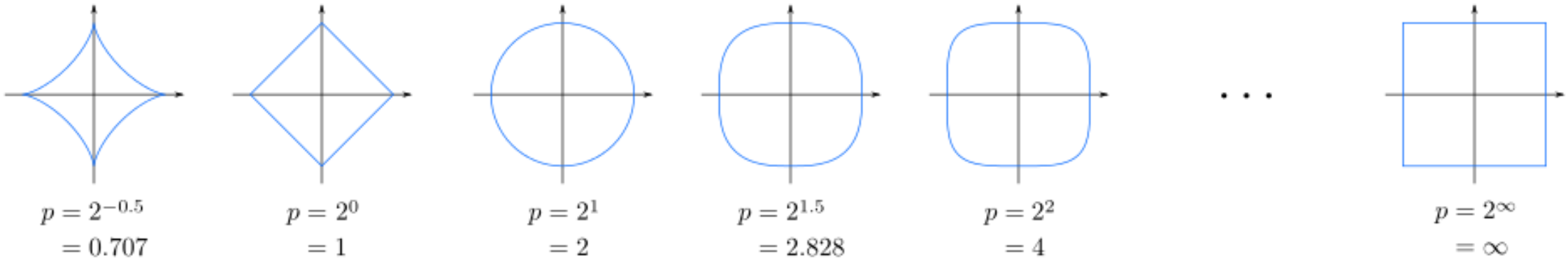
Equivalent to minimizing the sum of squared error (SSE) subject to constraint (proof):

$$\|w\|_p^p \leq \epsilon \text{ for some constant } \epsilon$$

Other forms of Regularization

Suggests regularization constrains solutions to be within some distance of the origin:

$$\|w\|_p^p \leq \epsilon \text{ for some constant } \epsilon$$



p defines the shape of the region of viable solutions, λ determines its size

Common forms of regularization in linear regression

Ridge Regression – The L2 penalized least-squares objective we've been dealing with so far.

Lasso Regression - Penalizes the L1-norm of the weights (i.e. minimizing sum of absolute value of weights).

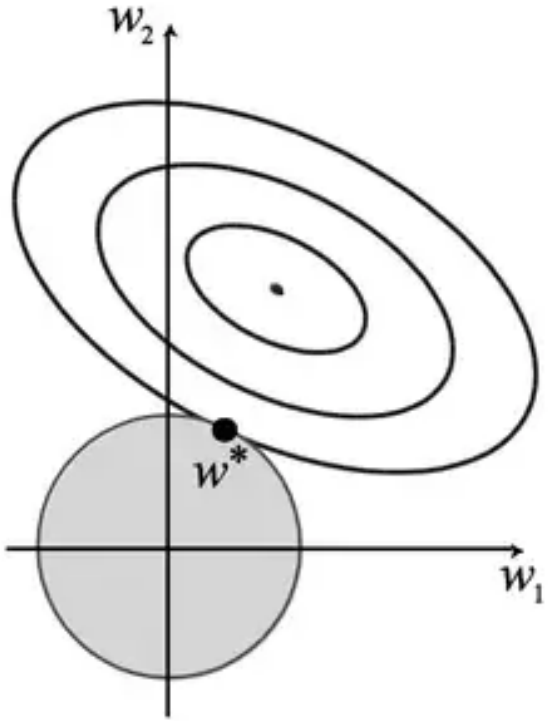
L0 or Sparse Regression- Penalizes the number of non-zero weights. This requires combinatoric optimization to solve exactly – NP-complete problem!

L2 vs L1 Regularization

Ridge Regression (L2)

Tends to yield denser weights

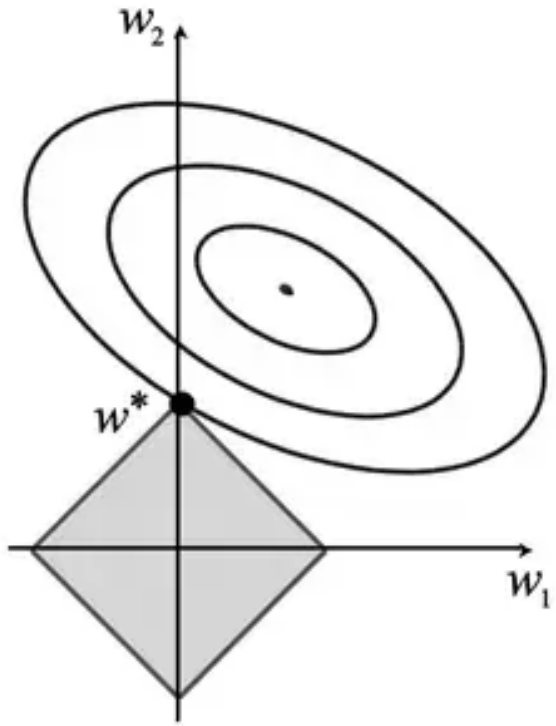
Computationally easy



Lasso Regression (L1)

Tends to yield sparser weights

More computationally involved



Summary of Regularization

Regularization is a much more general concept in machine learning. Consider any learning task to minimize a loss $\mathcal{L}(w)$ on the training data – could add a regularizer term.

$$w^* = \underset{w}{argmin}\, \mathcal{L}(w) + \lambda * regularizer(w)$$

Generally speaking, larger $\lambda's$ lead to simpler models and reduce overfitting, smaller $\lambda's$ lead to more complex models but improve fit on training data.

Most commonly used regularizers are based on the norm of the weight vector.

Today's Learning Objectives

Be able to answer:

- ~~Wrapping up a few details about OLS.~~
- ~~How can we use least squares regression to fit more complex functions?~~
  - ~~What is polynomial regression?~~
  - ~~More generally, how can we learn linear functions over basis functions?~~
- ~~What is regularization?~~
- What is logistic regression?

Moving onto a new topic. Binary classification with logistic regression.

- Binary (or two class) classification means our output y should be 0 or 1.

Logistic Regression Idea: Assume the probability of a data point x belonging to class 1 can be predicted by a warping the output of a linear model to be between 0 and 1.

- Logistic regression is a "linear classifier" because it will only be able to represent linear decision boundaries.

Why not regress probabilities directly?

Consider trying to regress a probability directly:

$$e.g., \qquad P(y_i|\boldsymbol{x_i};\boldsymbol{w}) = \boldsymbol{w^T x}$$

Some issues:

- Nothing constrains the output to be $0 \leq P(y_i|\boldsymbol{x_i};\boldsymbol{w}) \leq 1$
- Nothing constraints $P(y_i = 1|\boldsymbol{x_i};\boldsymbol{w}) + P(y_i = 0|\boldsymbol{x_i};\boldsymbol{w}) = 1$

Want something that "forces" it to the range 0-1…

Why not regress probabilities directly?

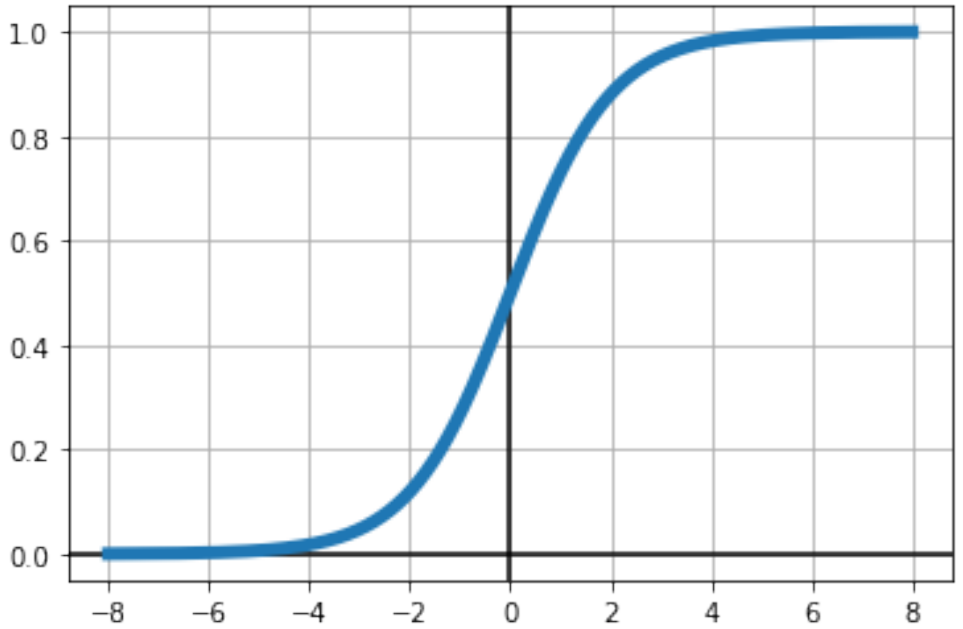
Introducing the logistic function:

- May also see it referred to as a sigmoid function or "logit" function

Logistic Function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Maps $(-\infty, \infty)$ to $(0,1)$



Logistic Regression Model Assumption:

$$P(y_i = 1|\boldsymbol{x_i}; \boldsymbol{w}) = \sigma(\boldsymbol{w^T x_i}) = \frac{1}{1 + e^{-\boldsymbol{w^T x_i}}}$$

$$P(y_i = 0|\boldsymbol{x_i}; \boldsymbol{w}) = 1 - \sigma(\boldsymbol{w^T x_i}) = \frac{e^{-\boldsymbol{w^T x_i}}}{1 + e^{-\boldsymbol{w^T x_i}}}$$

Logistic Regression Model's Decision Boundary:

Predict 1 if

$$P(y_i = 1|\boldsymbol{x_i}; \boldsymbol{w}) > P(y_i = 0|\boldsymbol{x_i}; \boldsymbol{w})$$

Divide both sides by $P(y_i = 0|\boldsymbol{x_i}; \boldsymbol{w})$

$$\Rightarrow \quad \frac{P(y_i = 1|\boldsymbol{x_i}; \boldsymbol{w})}{P(y_i = 0|\boldsymbol{x_i}; \boldsymbol{w})} > 1$$

Just some algebra

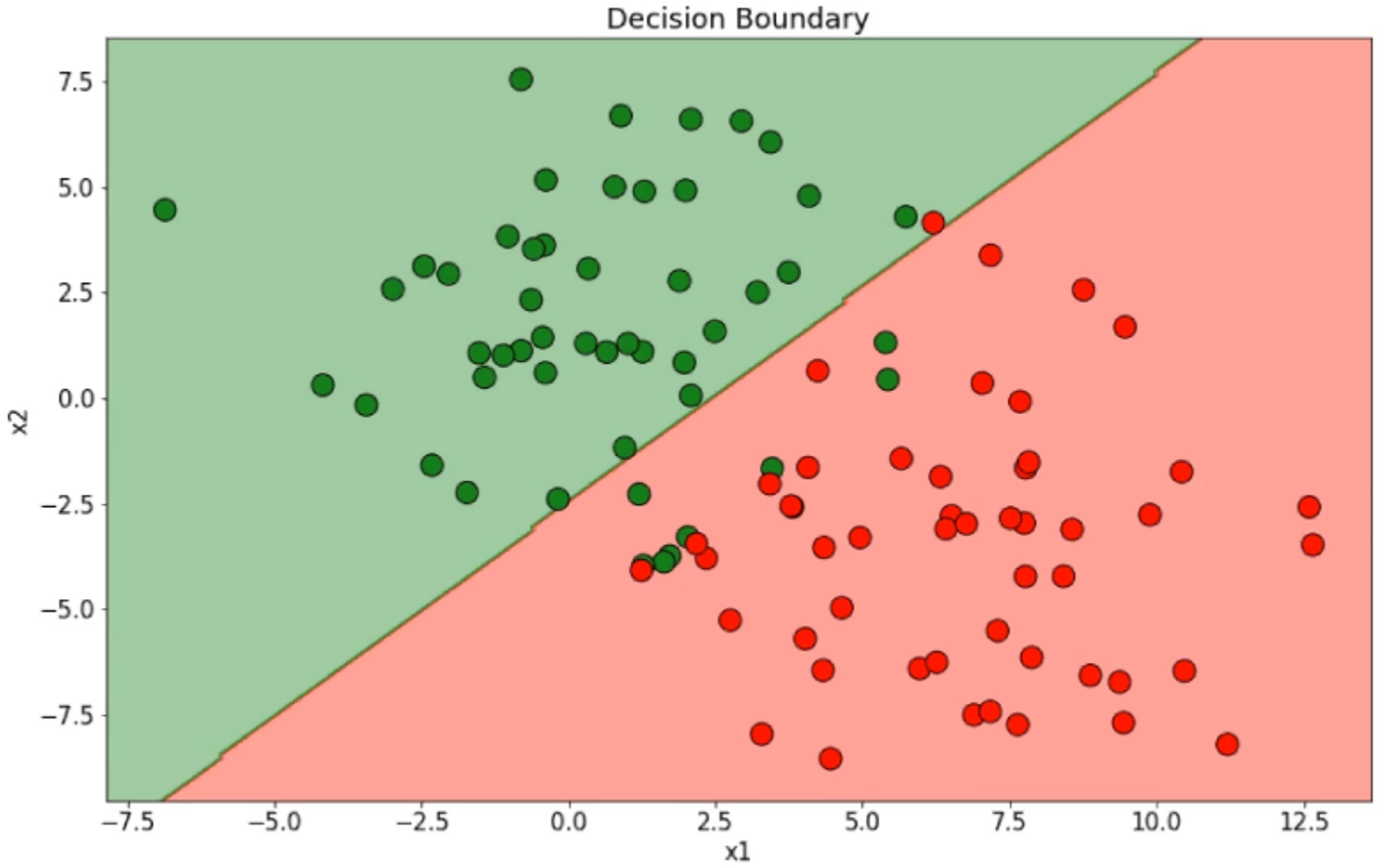
$$\Rightarrow \quad e^{\boldsymbol{w^T x}} > 1$$

Log of both sides

$$\Rightarrow \quad \boldsymbol{w^T x} > 0$$

This tells us the decision boundary is the line $w^T x = 0$

Example logistic regression decision boundary – linear in the features.

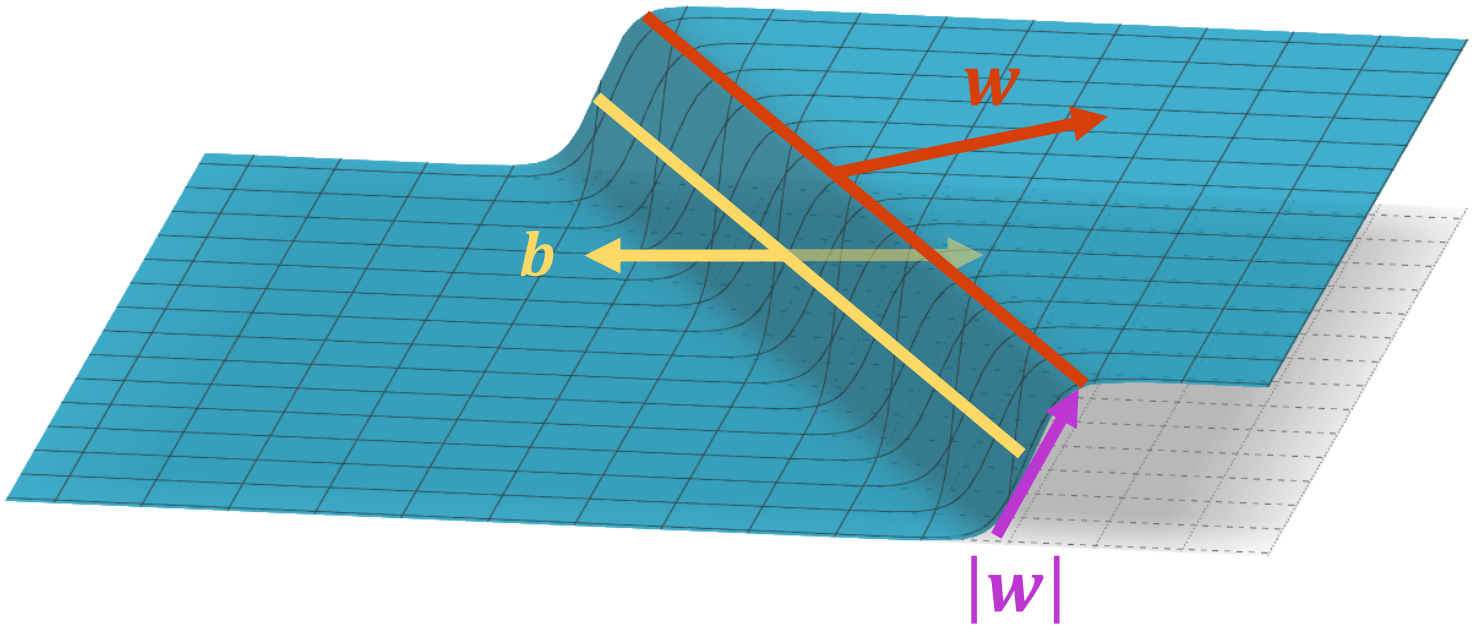


Intuition for Logistic Regression in 2D

$$\sigma(b + \boldsymbol{w}^T \boldsymbol{x}) = \frac{1}{1 + e^{-(\boldsymbol{w}^T \boldsymbol{x} + b)}}$$

$$\boldsymbol{w} = [10, 5]^T$$

$$b = -1$$

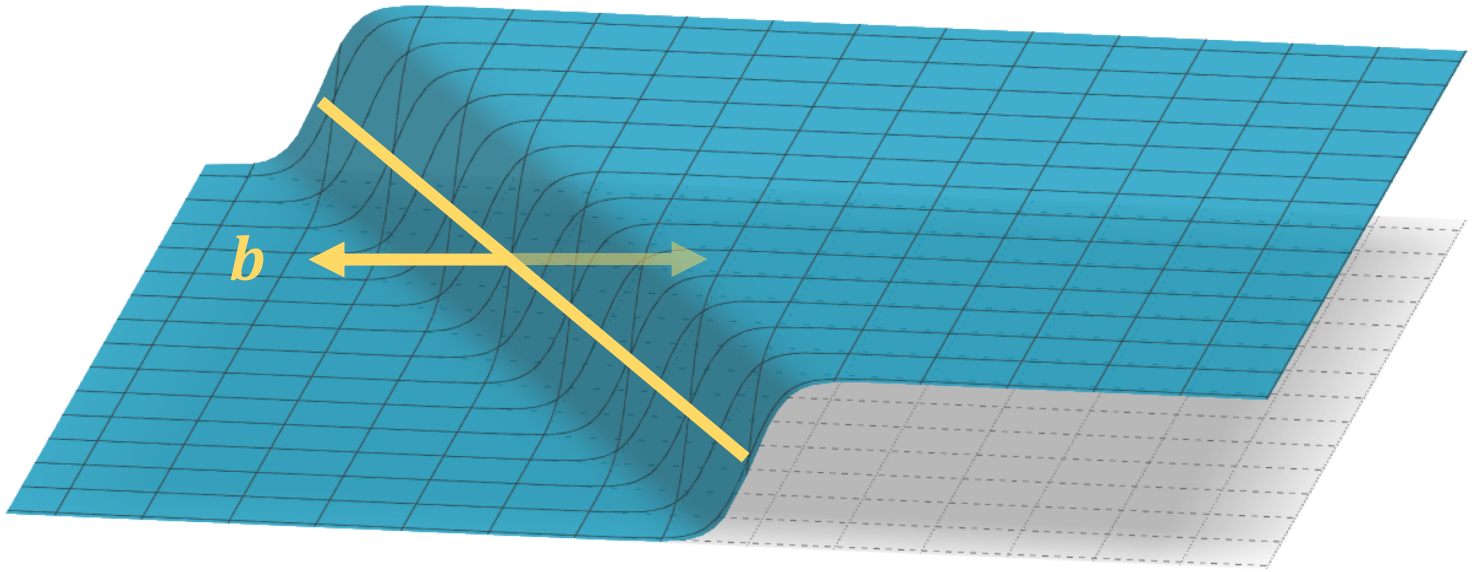


Intuition for Logistic Regression in 2D

$$\sigma(b + \boldsymbol{w}^T \boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})}}$$

$$\boldsymbol{w} = [\mathbf{10}, \mathbf{5}]^{\boldsymbol{T}}$$

$$\boldsymbol{b} = \mathbf{15}$$

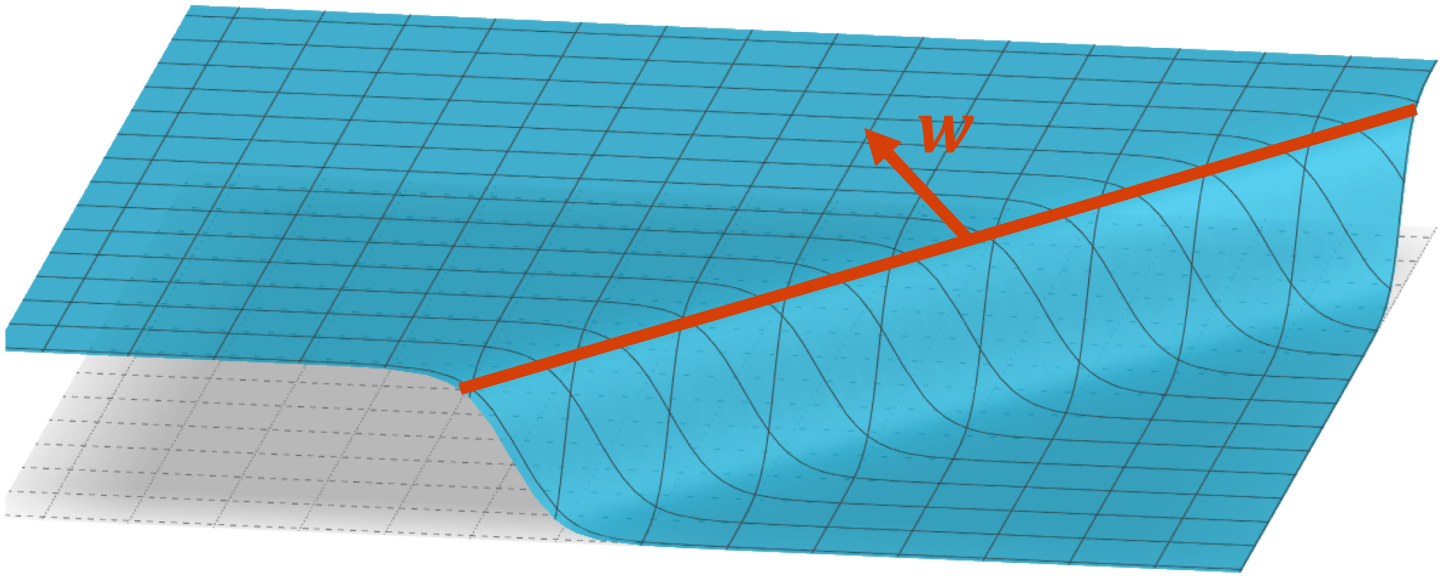


Intuition for Logistic Regression in 2D

$$\sigma(b + \boldsymbol{w}^T \boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})}}$$

$$\boldsymbol{w} = [-7, 5]^T$$

$$\boldsymbol{b} = 15$$

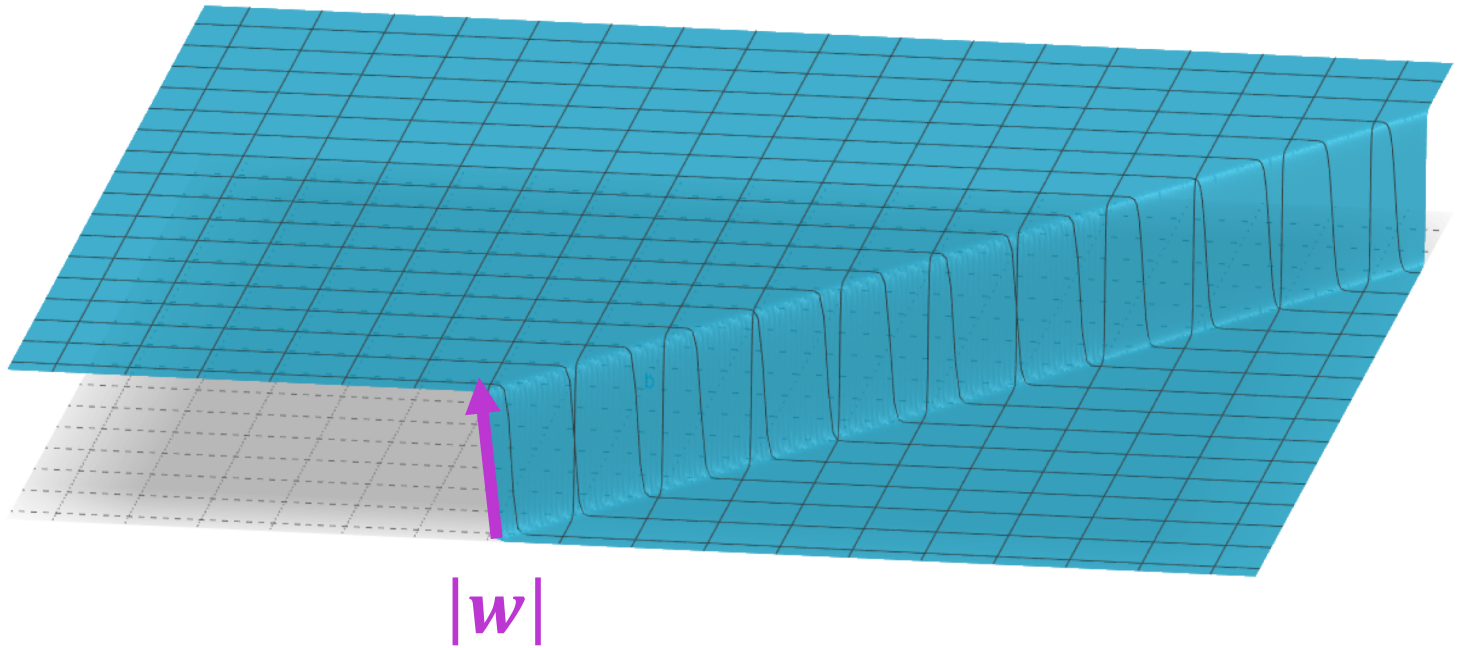


Intuition for Logistic Regression in 2D

$$\sigma(b + \boldsymbol{w}^T \boldsymbol{x}) = \frac{\mathbf{1}}{\mathbf{1} + \boldsymbol{e}^{-(\boldsymbol{w}^T \boldsymbol{x} + \boldsymbol{b})}}$$

$$\boldsymbol{w} = [-\mathbf{70}, \mathbf{50}]^T$$

$$\boldsymbol{b} = \mathbf{150}$$



MLE for Logistic Regression

Dataset: Given a dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$

Model assumptions: $y_i \sim Bernoulli\left(\theta = \sigma(\boldsymbol{w^T x_i})\right)$

$$\Rightarrow P(y_i|x_i, w) = \sigma(\boldsymbol{w^T x_i})^{y_i}\left(1 - \sigma(\boldsymbol{w^T x_i})\right)^{(1-y_i)}$$

Write out log-likelihood of the training data as a function of parameters:

$$logP(D \mid w) = \sum_{i=1}^{N} logP(y_i|\boldsymbol{x_i}, \boldsymbol{w}) = \sum_{i=1}^{N} y_i \log \sigma(\boldsymbol{w^T x_i}) + (1 - y_i) \log\left(1 - \sigma(\boldsymbol{w^T x_i})\right)$$

Finding the best weights comes down to maximizing this (or equivalently minimizing the negative log-likelihood).

MLE for Logistic Regression

Let's consider the negative log likelihood instead (convenient for later)

$$-logP(D \mid w) = \sum_{i=1}^{N} \left(-y_i \log \sigma\left(\boldsymbol{w^T x_i}\right) - (1 - y_i) \log\left(1 - \sigma\left(\boldsymbol{w^T x_i}\right)\right)\right)$$

$$= \sum_{i=1}^{N} l_i \qquad \text{Where } l_i = \left(-y_i \log \sigma\left(\boldsymbol{w^T x_i}\right) - (1 - y_i) \log\left(1 - \sigma\left(\boldsymbol{w^T x_i}\right)\right)\right)$$

We usually take derivatives and set to zero to find the minimum:

$$-\nabla logP(D \mid w) = \sum_{i=1}^{N} \nabla l_i$$

No closed form solution in this case. Let's see why.

MLE for Logistic Regression

Let's focus in on just the loss of a particular example:

$$\nabla l_i = \nabla\left(-y_i \log \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right) - (1 - y_i) \log\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right)\right)$$

$$= -y_i \nabla \log \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right) - (1 - y_i) \nabla \log\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right)$$

$$= \left(-\frac{y_i}{\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)} - \frac{1 - y_i}{1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)}\right) \nabla \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)$$

Gradient of the logistic function $\nabla \sigma(z) = \sigma(z)\left(1 - \sigma(z)\right)$, so $\nabla \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right) = \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right) \mathbf{x_i}$

$$\nabla l_i = \left(-\frac{y_i}{\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)} - \frac{1 - y_i}{1 - \sigma(\boldsymbol{w}^T \boldsymbol{x}_i)}\right) \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\left(1 - \sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right)\right) \mathbf{x_i}$$

$$= \left(\sigma\left(\boldsymbol{w}^T \boldsymbol{x}_i\right) - y_i\right) \mathbf{x_i}$$

Gradient is non-linear in parameters

Derivation of Derivative of Logistic Function

$$
\begin{aligned}
\frac{d}{dx}\sigma(x) &= \frac{d}{dx}\left[\frac{1}{1+e^{-x}}\right] \\
&= \frac{d}{dx}\left(1+\mathrm{e}^{-x}\right)^{-1} \\
&= -(1+e^{-x})^{-2}(-e^{-x}) \\
&= \frac{e^{-x}}{(1+e^{-x})^2} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{e^{-x}}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \frac{(1+e^{-x})-1}{1+e^{-x}} \\
&= \frac{1}{1+e^{-x}} \cdot \left(\frac{1+e^{-x}}{1+e^{-x}} - \frac{1}{1+e^{-x}}\right) \\
&= \frac{1}{1+e^{-x}} \cdot \left(1 - \frac{1}{1+e^{-x}}\right) \\
&= \sigma(x) \cdot (1-\sigma(x))
\end{aligned}
$$

So we have an expression for the gradient but can't find the optimal analytically…

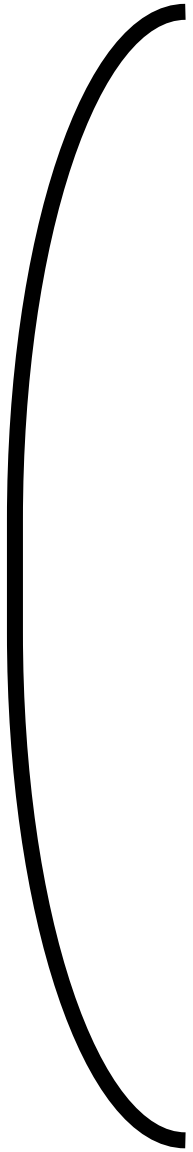
$$\nabla - logP(D \mid w) = \sum_{i=1}^{N} \nabla l_i = \sum_{i=1}^{N} \left(\sigma\left(\boldsymbol{w^T x_i}\right) - y_i\right)\mathbf{x_i}$$

Idea: The gradient points in the direction of steepest ascent of the function from a point.

- Can we use that to find the minimum of a function?
- Start with a random set of weights and iteratively move the parameters in the opposite direction of the gradient

This is referred to as gradient descent and is a first-order optimization method!

Gradient Descent



An Aside to Understand Gradient Descent

Given a function $\mathcal{L}: w \rightarrow \mathbb{R}$, find the minimum value. You can think of it as "loss surface" – for each setting of inputs, there is some function value.

Goal: Find a setting of w that has as low of loss as possible.

Recall: Gradient (vector of partial derivatives) point in the direction of steepest ascent.

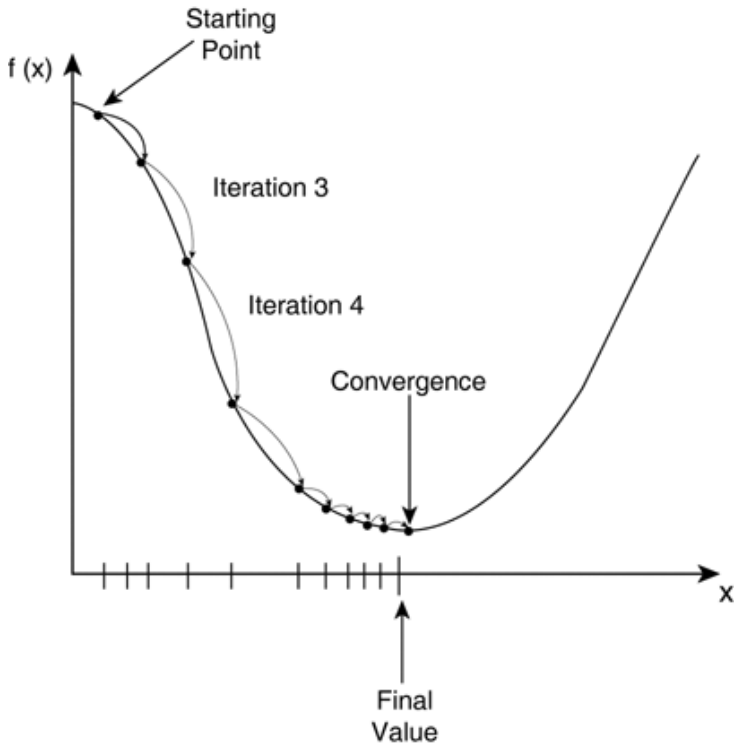
Gradient Descent Algorithm

$w \leftarrow random(\quad)$

while $|\nabla_w \mathcal{L}|_2 > \epsilon$ *or iters remain*

$w = w - \alpha \nabla_w \mathcal{L}(w)$

Learning Rate / Step Size



An Aside to Understand Gradient Descent

Different starting points may lead to different local minima if the function being optimized is non-convex.

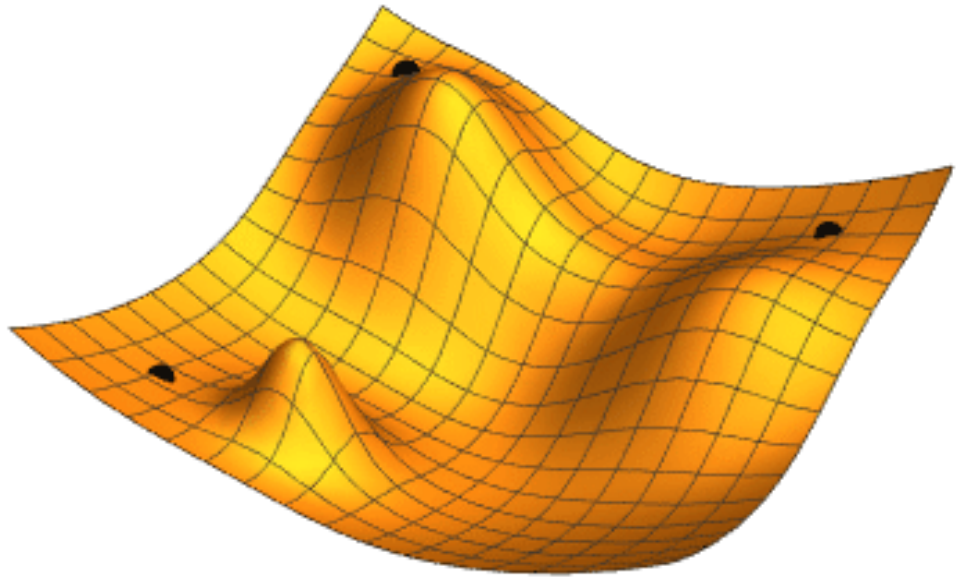
Gradient Descent Algorithm

$w \leftarrow random(\quad)$

while $|\nabla_w \mathcal{L}|_2 > \epsilon$ *or iters remain*

$w = w - \alpha \nabla_w \mathcal{L}(w)$

Learning Rate / Step Size



An Aside to Understand Gradient Descent

Choice of learning rate matters a lot. Usually good to decay it over time.

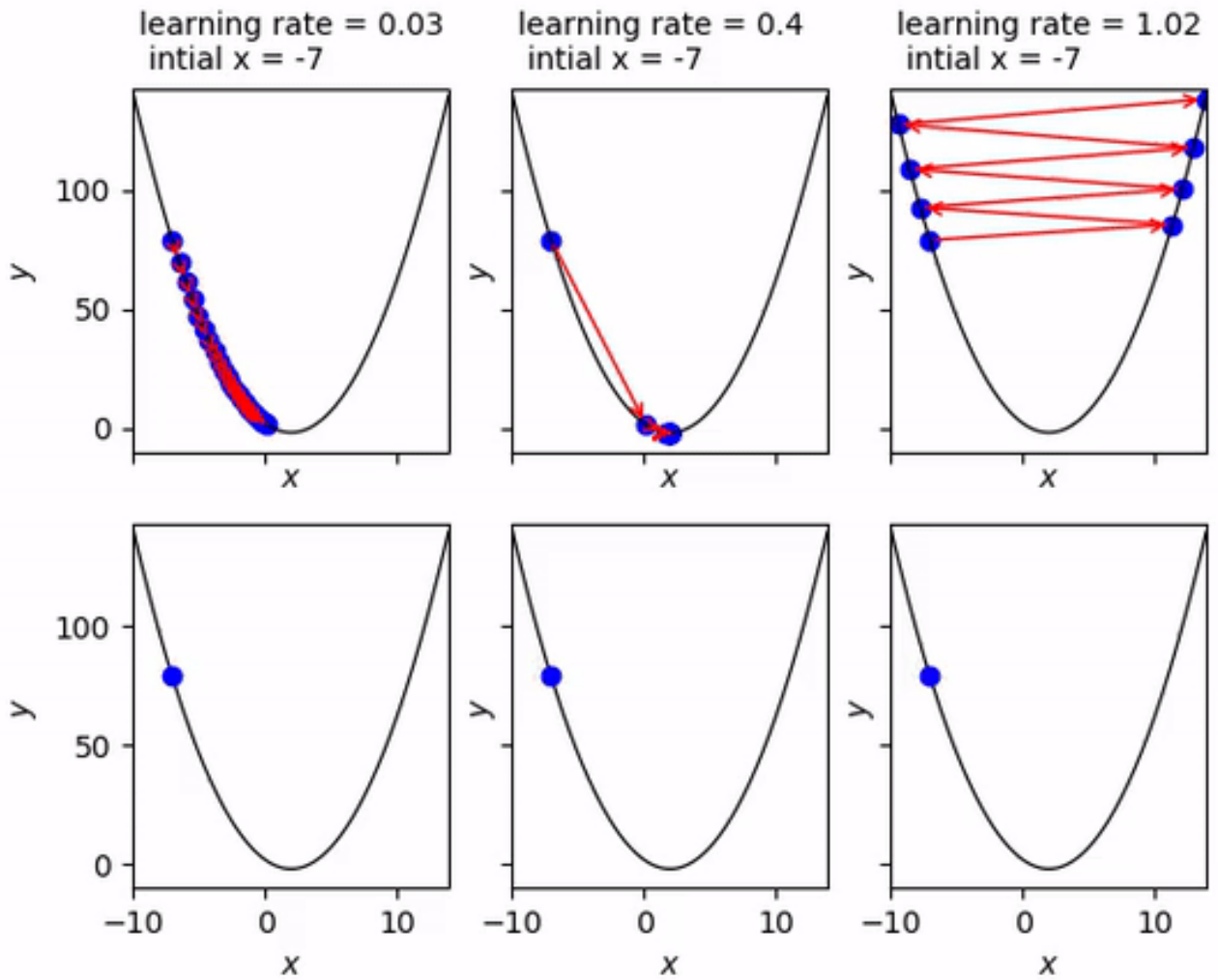
Gradient Descent Algorithm

$w \leftarrow random(\quad)$

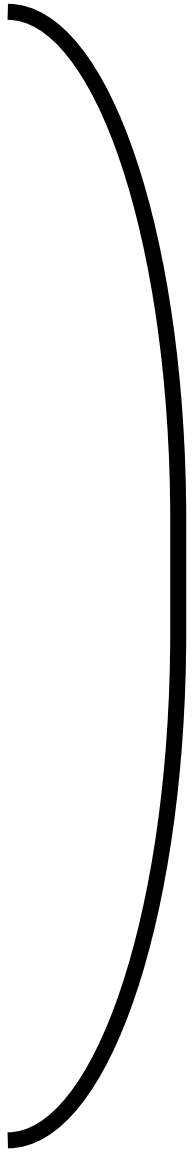
while$|\nabla_w \mathcal{L}| > \epsilon$ *or iters remain*

$w = w - \alpha \nabla_w \mathcal{L}(w)$

Learning Rate / Step Size



Gradient Descent



So we have an expression for the gradient but can't find the optimal analytically…

$$\nabla - logP(D \mid w) = \sum_{i=1}^{N} \nabla l_i = \sum_{i=1}^{N} \left(\sigma\left(\boldsymbol{w^T x_i}\right) - y_i\right)\mathbf{x_i}$$

Gradient Descent Algorithm for Logistic Regression

w = random(d) // randomly initialize weight vector

Repeat:

for each example $x_i, y_i \in D$: // compute gradient

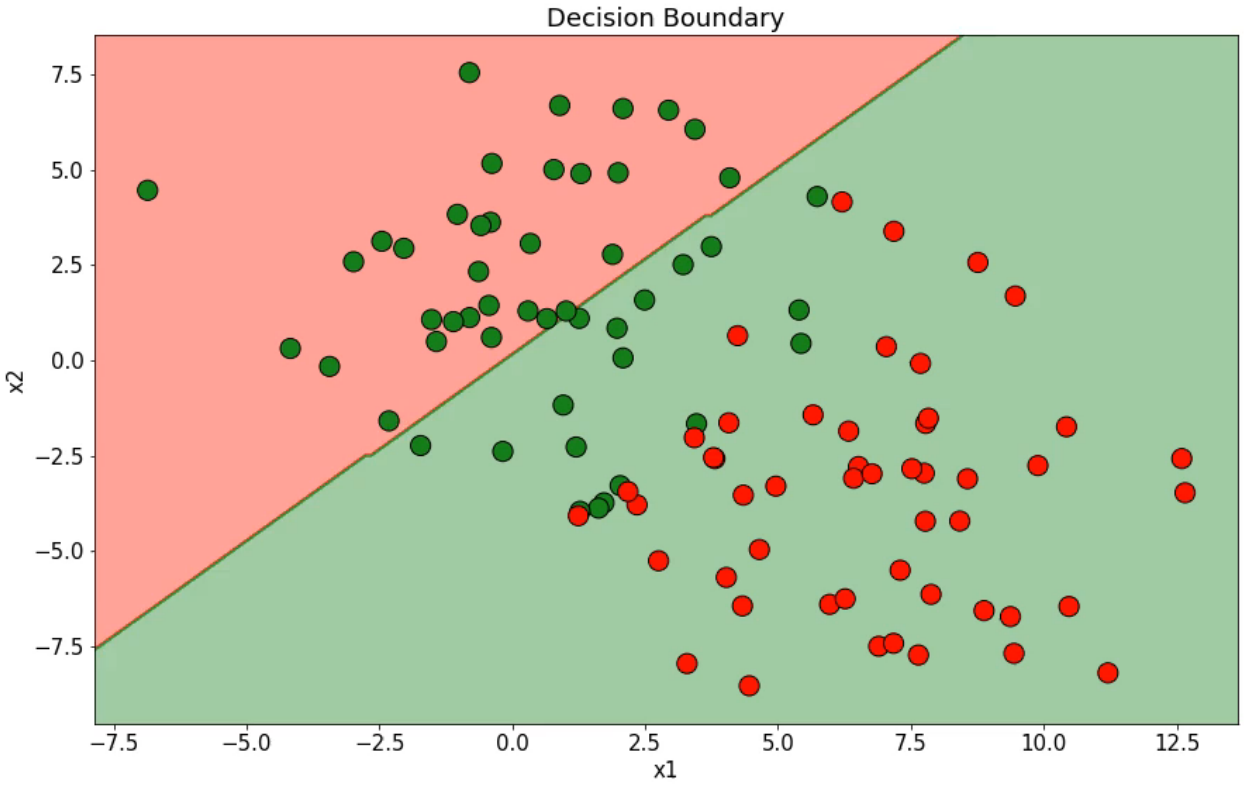
$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = \frac{1}{1+e^{-\boldsymbol{w^T x_i}}}$ // compute the prediction for this point

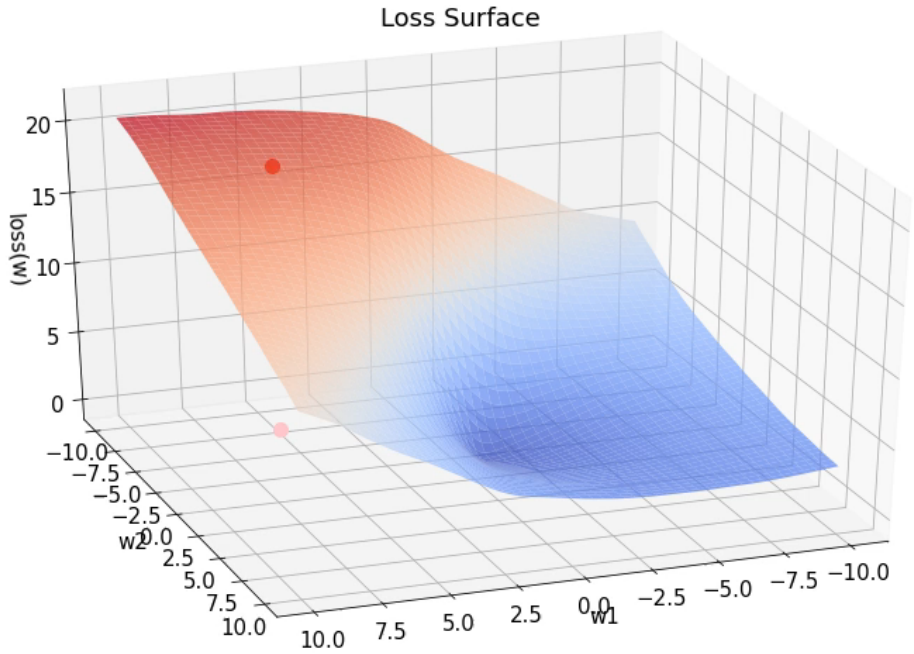
$\boldsymbol{\nabla} \mathrel{+}= (\hat{y}_i - y_i)\boldsymbol{x_i}$ // compute gradient of loss of point and sum

$\mathbf{w} = \mathbf{w} - \alpha\boldsymbol{\nabla}$ // take a step in the opposite direction of gradient

Until $|\nabla| \leq \epsilon$ // keep taking steps until convergence

An example of Logistic Regression with Gradient Descent





Summary of Logistic Regression

- Logistic regression uses $\sigma(\cdot)$ to warp the output of a linear function to (0,1), which is interpreted as $P(y = 1|x; w)$

- Learns a vector $\boldsymbol{w}$ s.t. for input $\boldsymbol{x_i}$ the desired correct output y_i has high probability, and other incorrect outputs have low probabilities.
  - Maximize the log likelihood or minimize the negative log likelihood

- Learn $\boldsymbol{w}$ iteratively using gradient descent
  - Compute gradient and take a step in the opposite direction
  - Give minima when function is convex, otherwise local minima

- Logistic regression learns a linear decision boundaries
  - Can apply the same basis function trick as in linear regression to learn nonlinear boundaries

Drawing a Parallel Between Linear Regression and Logistic Regression

Recall the model assumptions for linear regression and logistic regression:

$$y_i \sim \mathcal{N}\left(\mu = \boldsymbol{w}^T \boldsymbol{x_i}, \sigma\right)$$

$$y_i \sim Bernoulli\left(\theta = \sigma\left(\boldsymbol{w}^T \boldsymbol{x_i}\right)\right)$$

Both model y coming from some probability distribution with parameters derived from some function of x. This is a very general notion.

$$y_i \sim Categorical\left(\theta_c = \sigma\left(\boldsymbol{w_c}^T \boldsymbol{x_i}\right)\right)$$

Next time

Next Time: We'll talk about the perceptron algorithm (a very close neighbor to logistic regression) and then move on to Naïve Bayes.