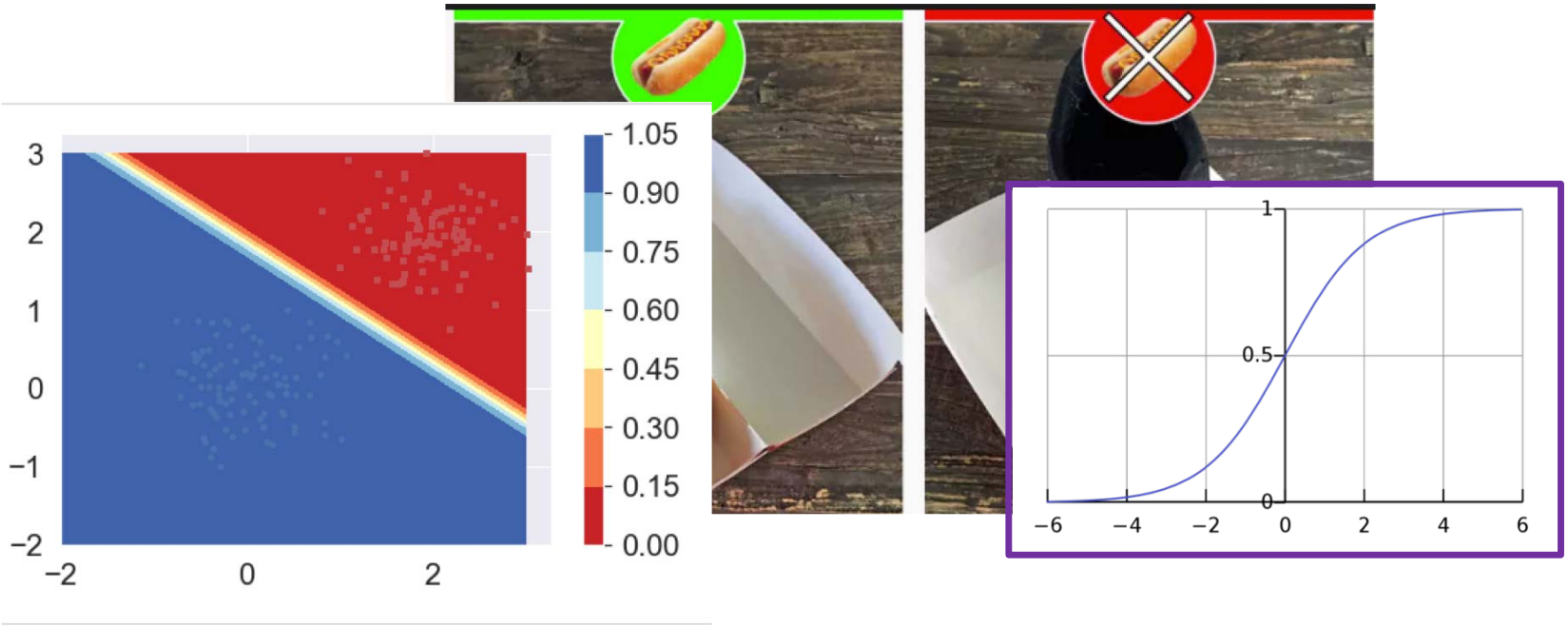
Tufts COMP 135: Introduction to Machine Learning

https://www.cs.tufts.edu/comp/135/2019s/

Logistic Regression



Many slides attributable to:
Erik Sudderth (UCI)
Finale Doshi-Velez (Harvard)
James, Witten, Hastie, Tibshirani (ISL/ESL books)

Prof. Mike Hughes

Logistics

- Waitlist: We have room, contact me ASAP

- HW3 due Wed
  - Please annotate pages in Gradescope!
  - Remember: Turn in on time!

- Recitation tonight (730-830pm, Room 111B)
  - Practical binary classifiers in Python with sklearn
  - Numerical issues and how to address them

Objectives: Logistic Regression Unit

Refresher: "taste" of 3 Methods

- Logistic Regression, k-NN, Decision Trees

Logistic Regression: A Probabilistic Classifier

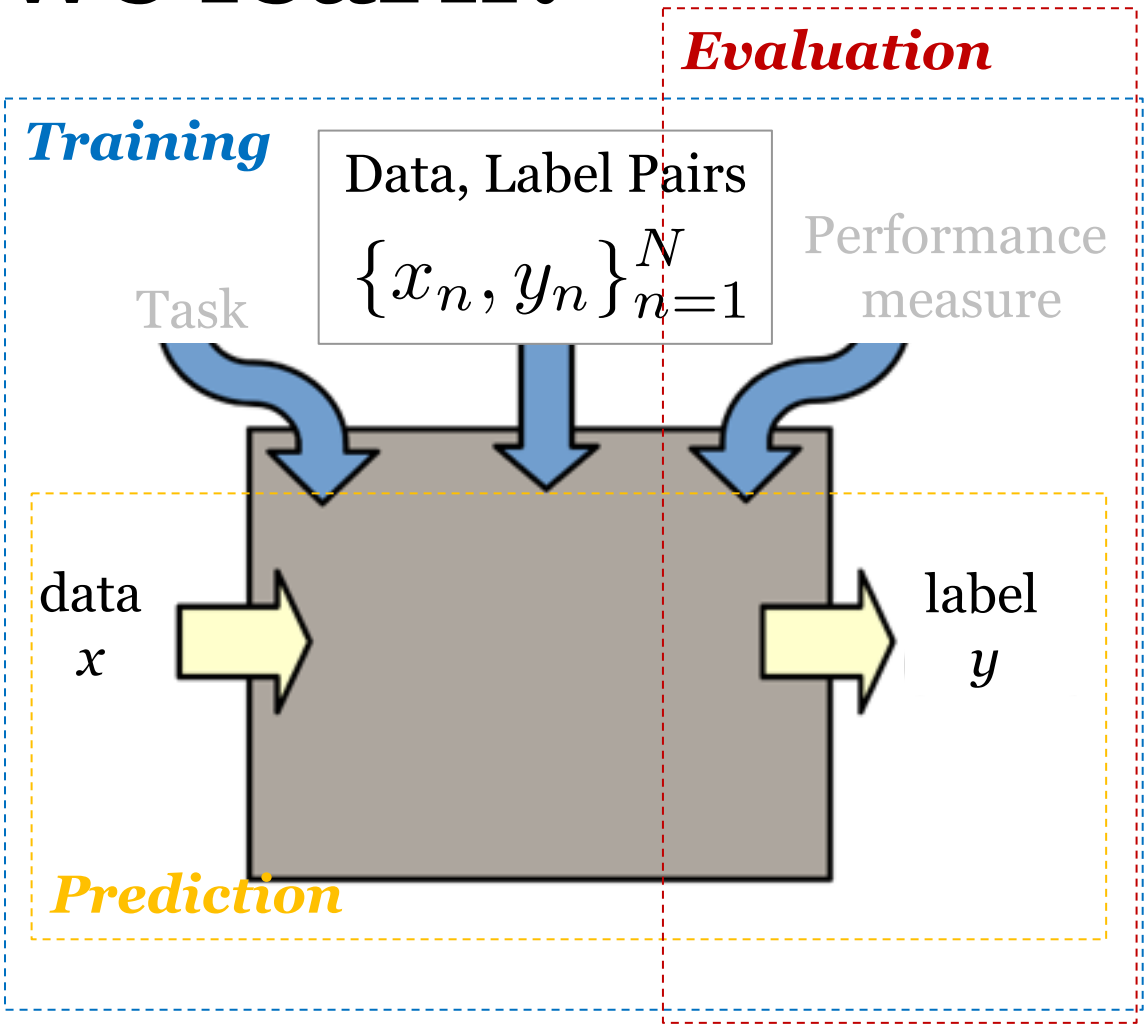
- 3 views on why we optimize log loss
  - Upper bound error rate
  - Minimize (cross) entropy
  - Maximize (log) likelihood
- Computing gradients
- Training via gradient descent

What will we learn?

Supervised Learning

Unsupervised Learning

Reinforcement Learning



Task: Binary Classification

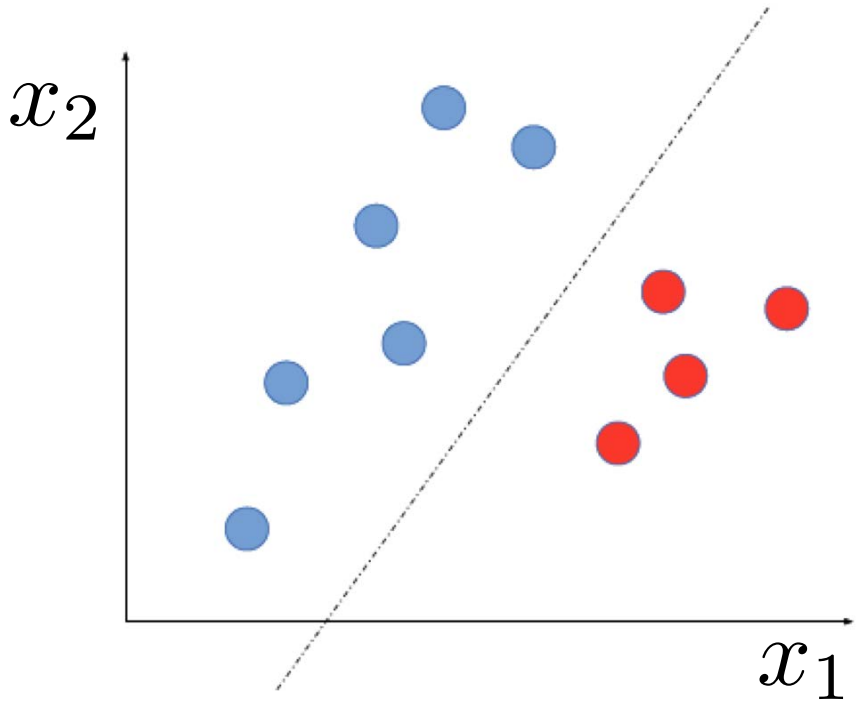
Supervised Learning

binary classification

Unsupervised Learning

Reinforcement Learning

y is a binary variable (red or blue)



Example: Hotdog or Not

https://www.theverge.com/tldr/2017/5/14/15639784/hbo-silicon-valley-not-hotdog-app-download

Binary Prediction

Goal: Predict label (0 or 1) given features x

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $y_i \in \{0, 1\}$

  "responses" or "labels"

  Binary label (0 or 1)

```
>>> yhat_N = model.predict(x_NF)
>>> yhat_N[:5]
[0, 0, 1, 0, 1]
```

Probability Prediction

Goal: Predict probability of label given features

- Input: $x_i \triangleq [x_{i1}, x_{i2}, \ldots x_{if} \ldots x_{iF}]$

  "features"
  "covariates"
  "attributes"

  Entries can be real-valued, or other numeric types (e.g. integer, binary)

- Output: $\hat{p}_i \triangleq p(Y_i = 1 | x_i)$

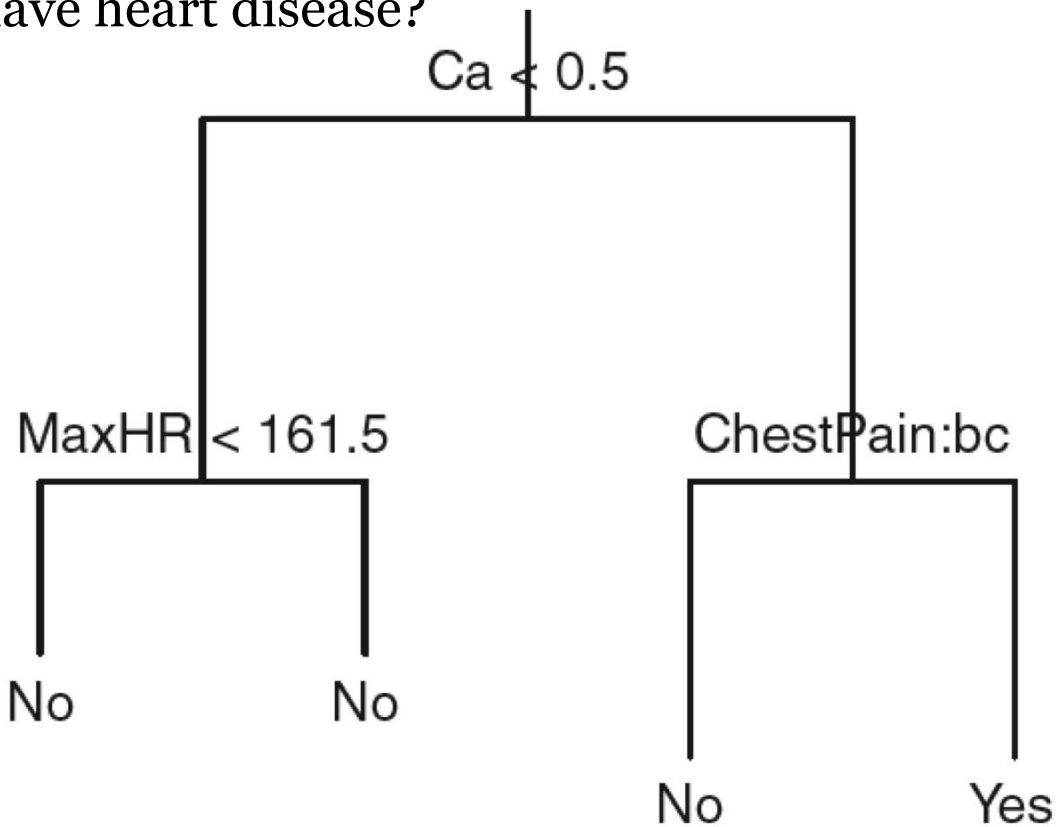
  "probability"

  Value between 0 and 1
  e.g. 0.001, 0.513, 0.987

```
>>> yproba_N2 = model.predict_proba(x_NF)
>>> yproba1_N = model.predict_proba(x_NF)[:,1]
>>> yproba1_N[:5]
[0.143, 0.432, 0.523, 0.003, 0.994]
```

Decision Tree Classifier

Goal: Does patient have heart disease?



Leaves make binary predictions! *(but can be made probabilistic)*

Decision Tree Classifier

Parameters:

- *at each internal node: x* variable id and threshold
- *at each leaf:* probability of positive *y* label

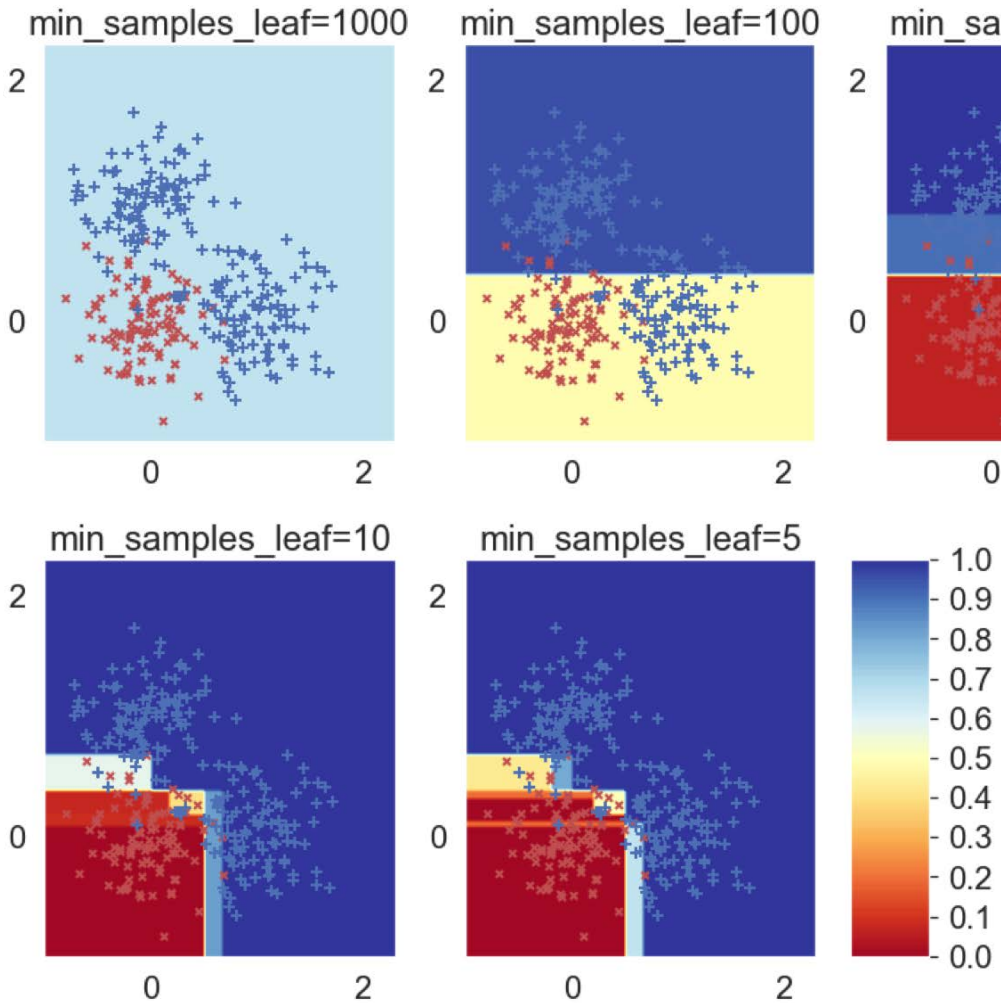
Prediction:

- identify rectangular region for input x
- predict: most common y value in region
- predict_proba: report fraction of each label in regtion

Training:

- minimize error on training set
- often, use greedy heuristics

Decision Tree: Predicted Probas



Pretty flexible!

Function is piecewise constant and axis-aligned

K nearest neighbor classifier

Parameters:

K : number of neighbors

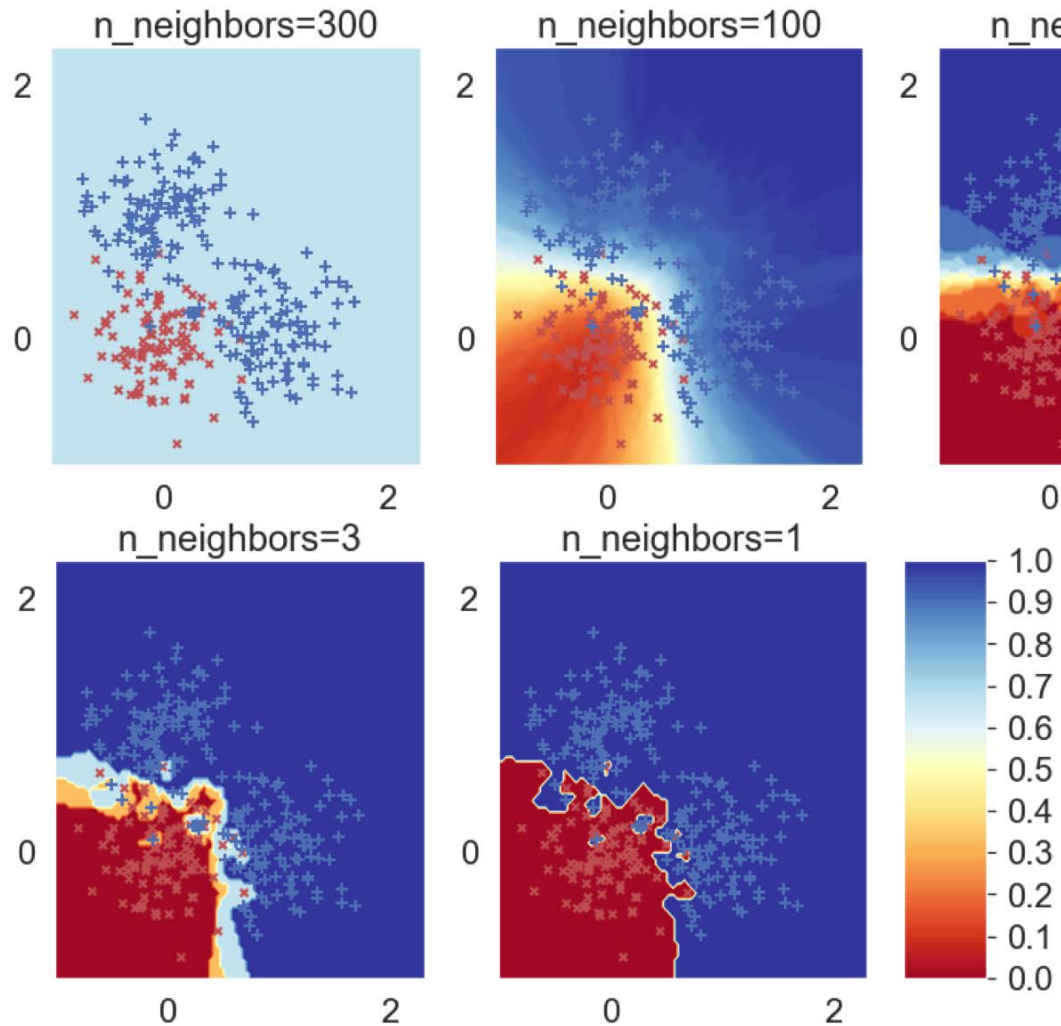
Prediction:

- find K "nearest" training vectors to input x
- predict: vote most common y in neighborhood
- predict_proba: report fraction of labels

Training:

none needed (use training data as lookup table)

KNN: Predicted Probas



Very flexible!

Function is piecewise constant

Logistic Regression classifier

Parameters:

weight vector $w = [w_1, w_2, \ldots w_f \ldots w_F]$

bias scalar b

Prediction:

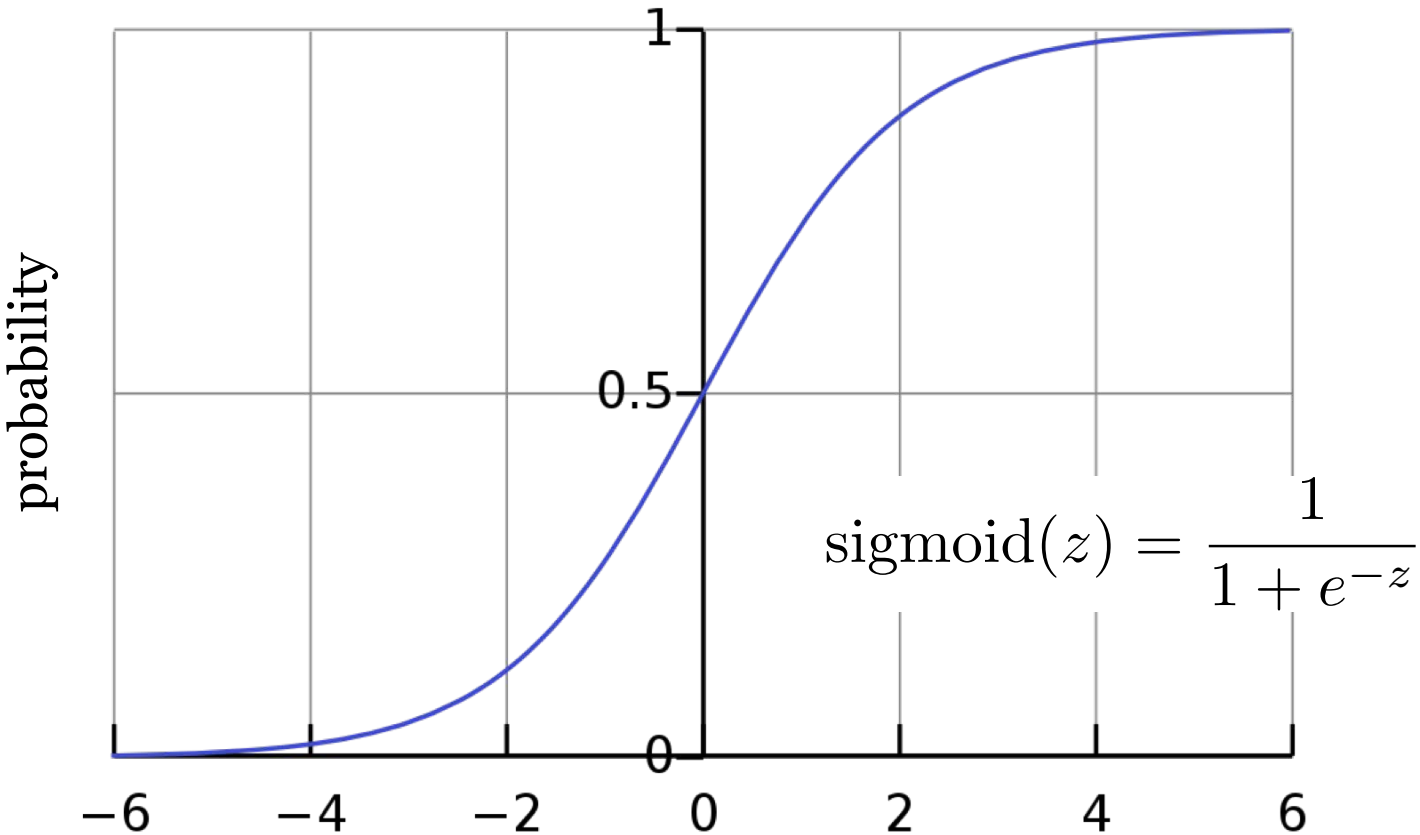
$$\hat{p}(x_i, w, b) = p(y_i = 1 | x_i) \triangleq \text{sigmoid}\left(\sum_{f=1}^{F} w_f x_{if} + b\right)$$

Training:

find weights and bias that minimize error

Logistic Sigmoid Function

Goal: Transform real values into probabilities



Logistic Regression: Training

Optimization: Minimize total log loss on train set

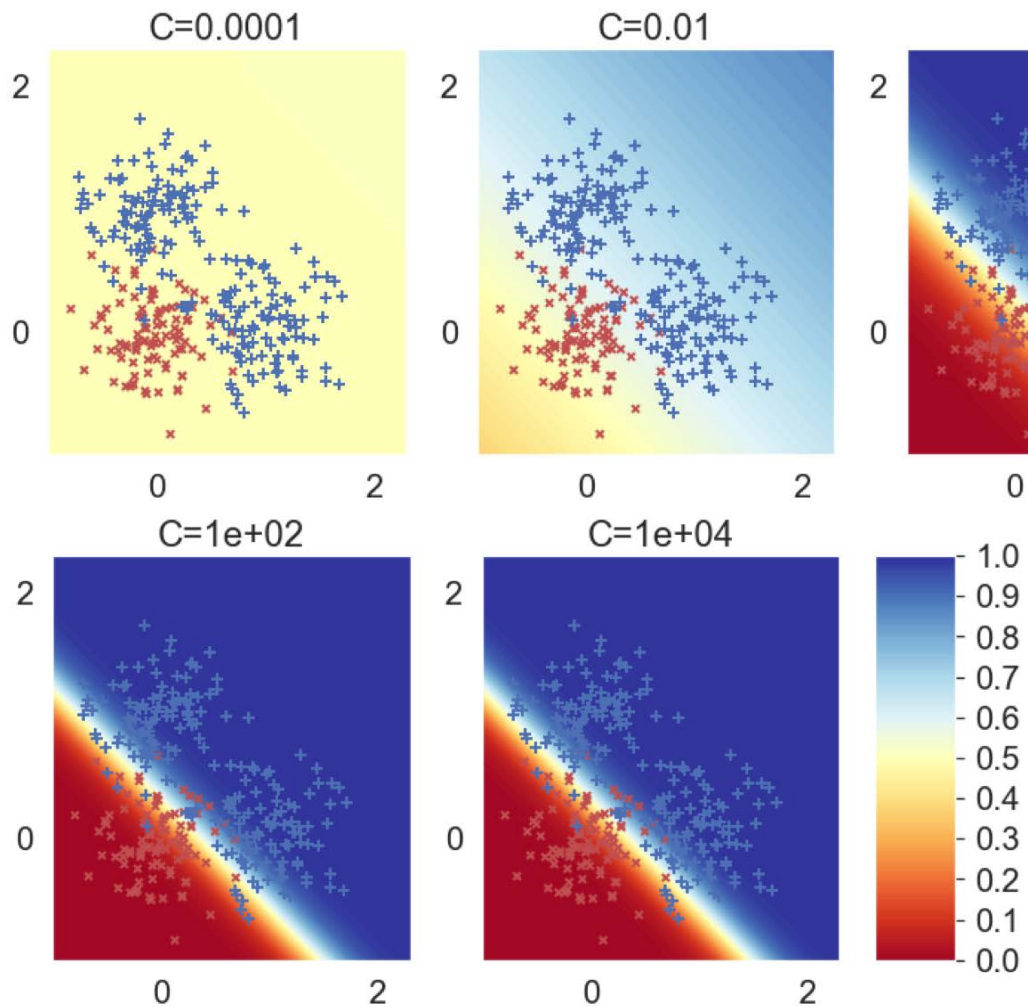
$$\min_{w,b} \sum_{n=1}^{N} \text{log\_loss}(y_n, \hat{p}(x_n, w, b))$$

Algorithm: Gradient descent

Today!

Avoid overfitting: Use L2 or L1 penalty on weights

Logistic Regr: Predicted Probas



Function is monotonically increasing in one direction

Decision boundaries will be linear

Summary of Methods

	Function class flexibility	Knobs to tune	Interpret?
Logistic Regression	Linear	L2/L1 penalty on weights	Inspect weights
Decision Tree Classifier	Axis-aligned Piecewise constant	Max. depth Min. leaf size Goal criteria	Inspect tree
K Nearest Neighbors Classifier	Piecewise constant	Number of Neighbors Distance metric How neighbors vote	Inspect neighbors

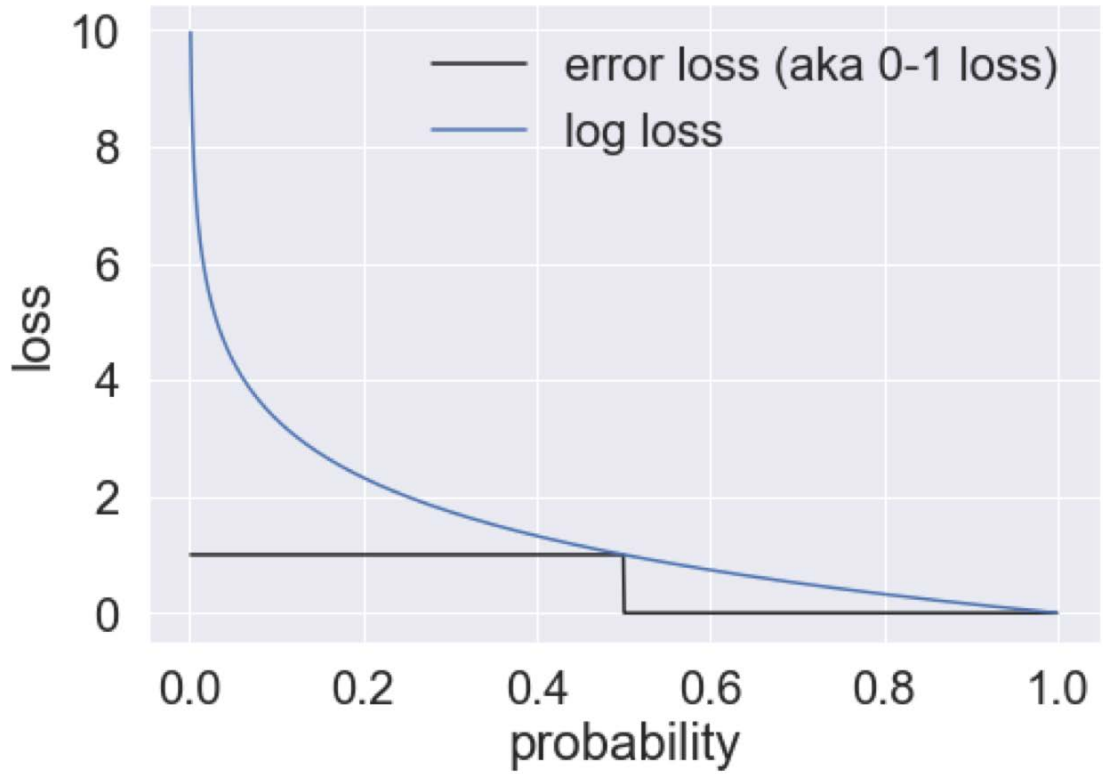
Optimization Objective
Why minimize log loss?

An upper bound justification

Log loss upper bounds error rate

$$\text{error}(y, \hat{y}) = \begin{cases} 1 & \text{if } y \neq \hat{y} \\ 0 & \text{if } y = \hat{y} \end{cases}$$

$$\text{log\_loss}(y, \hat{p}) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$



Plot assumes:

- True label is 1
- Threshold is 0.5
- Log base 2

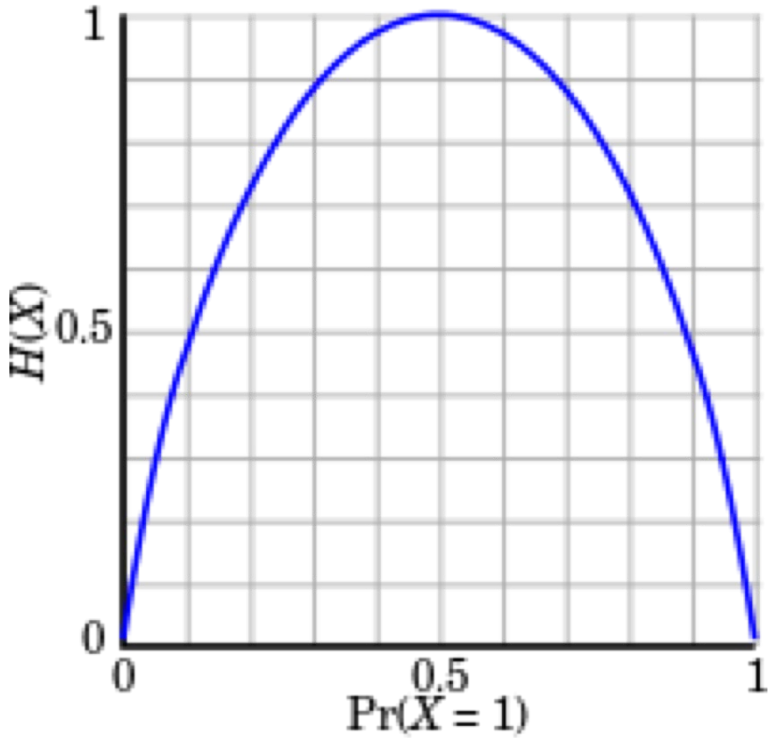
Optimization Objective
Why minimize log loss?

An information-theory justification

Entropy of Binary Random Var.

Goal: Entropy of a distribution captures the amount of uncertainty

$$\text{entropy}(X) = -p(X=1)\log_2 p(X=1) - p(X=0)\log_2 p(X=0)$$



Log base 2: Units are “bits”
Log base e: Units are “nats”

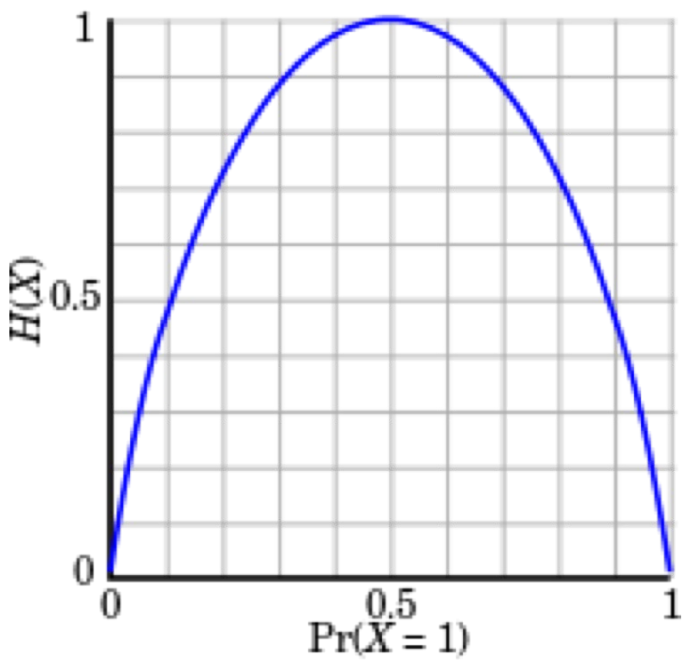
1 bit of information is always needed to represent a binary variable X

Entropy tells us how much of this one bit is uncertain

Entropy of Binary Random Var.

Goal: Entropy of a distribution captures the amount of uncertainty

$$\text{entropy}(X) = -p(X=1)\log_2 p(X=1) - p(X=0)\log_2 p(X=0)$$



$$H[X] = -\sum_{x \in \{0,1\}} p(X=x)\log_2 p(X=x)$$

$$= -\mathbb{E}_{x \sim p(X)}\left[\log_2 p(X=x)\right]$$

Entropy is the **average** number of bits needed to encode an outcome

Want: low entropy
(low cost storage and transmission!)

Cross Entropy

Goal: Measure cost of using estimated q to capture true distribution p

$$\text{Entropy}[p(X)] = -\sum_{x \in \{0,1\}} p(X = x) \log_2 p(X = x)$$

$$\text{Cross-Entropy}[p(X), q(X)] = -\sum_{x \in \{0,1\}} p(X = x) \log_2 q(X = x)$$

Info theory interpretation:

Average number of bits needed to encode
samples from a true distribution p(X)
with codes defined by a model q(X)

Goal: Want a model q that uses fewer bits! Lower entropy!

Log loss is cross entropy!

Let our "true" distribution p(Y) be **empirical** distribution of labels in the training set

Let our "model" distribution q(Y) be the estimated probabilities from logistic regression

$$\text{Cross-Entropy}[p(Y), q(Y)] = \mathbb{E}_{y \sim p(Y)}\left[-\log q(Y = y)\right]$$
$$= \frac{1}{N}\sum_{n=1}^{N} -y_n \log \hat{p}_n - (1 - y_n)\log(1 - \hat{p}_n)$$

Same as the "log loss"!

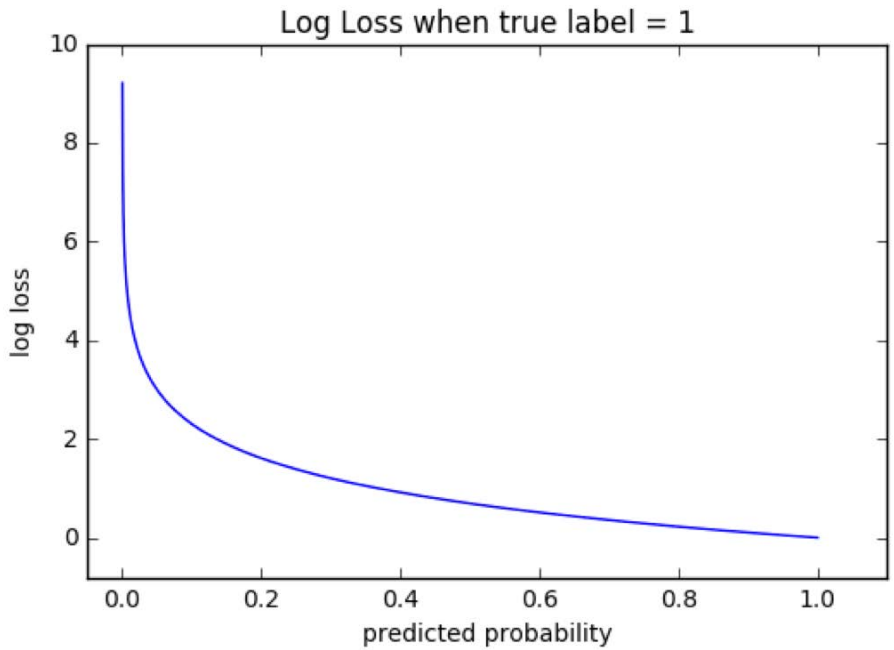
Info Theory Justification for log loss:
Want to set logistic regression weights to provide best encoding of the training data's label distribution

The log loss metric

Log loss (aka "binary cross entropy")

```
from sklearn.metrics import log_loss
```

$$\text{log\_loss}(y, \hat{p}) = -y \log \hat{p} - (1 - y) \log(1 - \hat{p})$$



Lower is better!

Advantages:

- smooth
- easy to take derivatives!

Optimization Objective

Why minimize log loss?

A **probabilistic** justification

Likelihood of labels under LR

We can write the probability for each outcome of Y as:

$$p(Y_i = 1 | x_i) = \text{sigmoid}(w^T x_i + b)$$
$$p(Y_i = 0 | x_i) = 1 - \text{sigmoid}(w^T x_i + b)$$

We can write the probability mass function of Y as:

$$p(Y_i = y_i | x_i) = \left[\sigma(w^T x_i + b)\right]^{y_i} \left[1 - \sigma(w^T x_i + b)\right]^{1-y_i}$$

Interpret: p(y | x) is the "likelihood" of label y given input features x
Goal: Fit model to make the training data as likely as possible

Maximizing likelihood

$$\max_{w,b} \prod_{n=1}^{N} p(Y_n = y_n | x_n, w, b)$$

Why might this be hard in practice?

Think about datasets with 1000s of examples N

Maximizing **log** likelihood

The **logarithm** (with any base) is a monotonic transform

a > b implies log (a) > log (b)

Thus, the following are equivalent problems

$$w^*, b^* = \arg\max_{w,b} \quad \prod_{n=1}^{N} p(Y_n = y_n | x_n, w, b)$$

$$w^*, b^* = \arg\max_{w,b} \quad \log \left[\prod_{n=1}^{N} p(Y_n = y_n | x_n, w, b) \right]$$

Log likelihood for LR

We can write the probability mass function of Y as:

$$p(Y_i = y_i | x_i) = \left[\sigma(w^T x_i + b)\right]^{y_i} \left[1 - \sigma(w^T x_i + b)\right]^{1-y_i}$$

Our training objective is to maximize log likelihood

$$w^*, b^* = \arg\max_{w,b} \underbrace{\boxed{\log \left[\prod_{n=1}^{N} p(Y_n = y_n | x_n, w, b)\right]}}_{J(w,\, b)}$$

Pair Exercise: Simplify the training objective J(w,b)!

Can you recover a familiar form?

Minimize **negative** log likelihood

Two equivalent optimization problems:

$$w^*, b^* = \arg\max_{w,b} \quad \sum_{n=1}^{N} \log p(Y_n = y_n | x_n, w, b)$$

$$w^*, b^* = \arg\min_{w,b} \quad -\sum_{n=1}^{N} \log p(Y_n = y_n | x_n, w, b)$$

Summary of "Likelihood interpretation"

- LR defines a probabilistic model for Y given x
- We want to maximize probability of training data (the "likelihood") under this model
- We can show that an another optimization problem (”maximize log likelihood”) is easier numerically but produces the same optimal values for weights and bias
- Turns out, minimizing log loss is precisely the same thing as minimizing negative log likelihood

Computing gradients

Simplified LR notation

- Feature vector with first entry constant

$$\tilde{x}_n = [1 \; x_{n1} \; x_{n2} \; \ldots \; x_{nF}]$$

- Weight vector (first entry is the "bias")

$$w = [w_0 \quad w_1 \quad w_2 \ldots w_F]$$

- "Score" value z (real number, -inf to +inf)

$$z_n = w^T \tilde{x}_n$$

Simplifying the log likelihood

$$
\begin{aligned}
J(z_n) &\triangleq \log p(Y_n = y_n | z_n) \\
&= y_n \log \sigma(z_n) + (1 - y_n) \log(1 - \sigma(z_n)) \\
&= y_n \log \sigma(z_n) + (1 - y_n) \log \sigma(-z_n) \\
&= y_n \log \frac{e^{z_n}}{1 + e^{z_n}} + (1 - y_n) \log \frac{1}{1 + e^{z_n}} \\
&= y_n z_n - \log\left(1 + e^{z_n}\right)
\end{aligned}
$$

Gradient of the log likelihood

$$\boxed{z_n = w^T \tilde{x}_n}$$

Log likelihood

$$J(z_n(w)) = y_n z_n - \log(1 + e^{z_n})$$

Gradient w.r.t. weight on feature f

$$\frac{d}{dw_f} J(z_n(w)) = \frac{d}{dz_n} J(z_n) \cdot \frac{d}{dw_f} z(w)$$

Simplifying yields: $= (y_n - \sigma(z_n))\, x_{nf}$

Partner Activity

Try the notebook here:

https://github.com/tufts-ml-courses/comp135-19s-assignments/blob/master/labs/LogisticRegressionDemo.ipynb

Goals: Build understanding

- What is the optimal w for the 1D example?
- What is the optimal w for the 2D example?
- Why is regularization important here?

STOP: End of class 2/11

Gradient descent for LR

Gradient descent for L2 penalized LR

$$\min_{\boldsymbol{w}, w_0} \boxed{\begin{array}{c} -\sum_i \log\ p(y_i \mid \boldsymbol{x}_i; \boldsymbol{w}, w_0) \\ J(w, w_0) \end{array}} + \frac{\lambda}{2} \|\boldsymbol{w}\|_2^2$$

Start with $\boldsymbol{w}^0 = 0, w_0^0 = 0$, step size s

for $t = 0, \ldots, (T-1)$

$\quad \boldsymbol{w}^{t+1} = \boldsymbol{w}^t - \mathrm{s}\, \nabla J(\boldsymbol{w}^t, w_0^t) - \lambda \boldsymbol{w}^t$

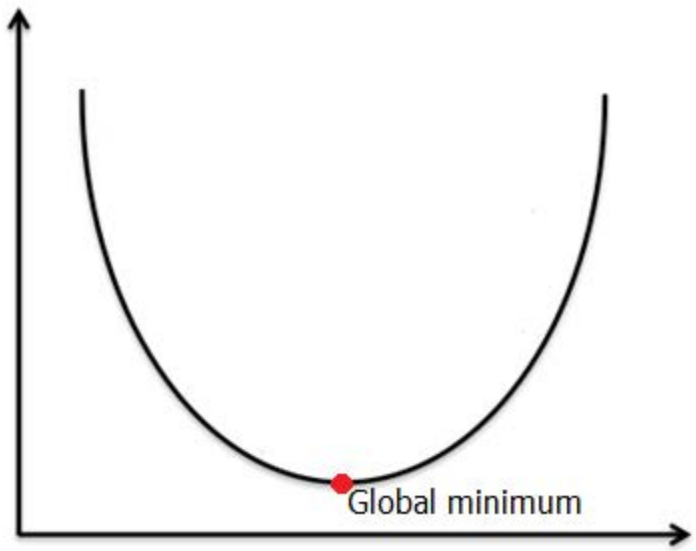
$\quad w_0^{t+1} = w_0^t - \mathrm{s}\, \nabla J(\boldsymbol{w}^t, w_0^t)$

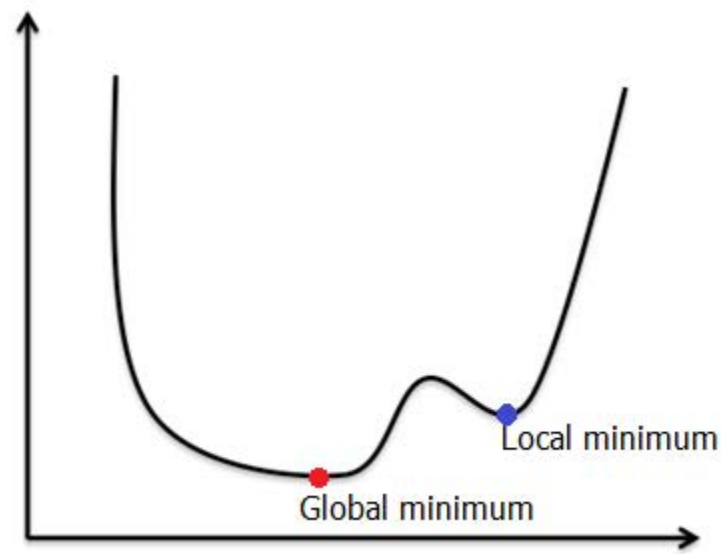
$\quad$ if $L(\boldsymbol{w}^{t+1}, w_0^{t+1}) - L(\boldsymbol{w}^t, w_0^t) < \delta$

$\quad\quad$ break

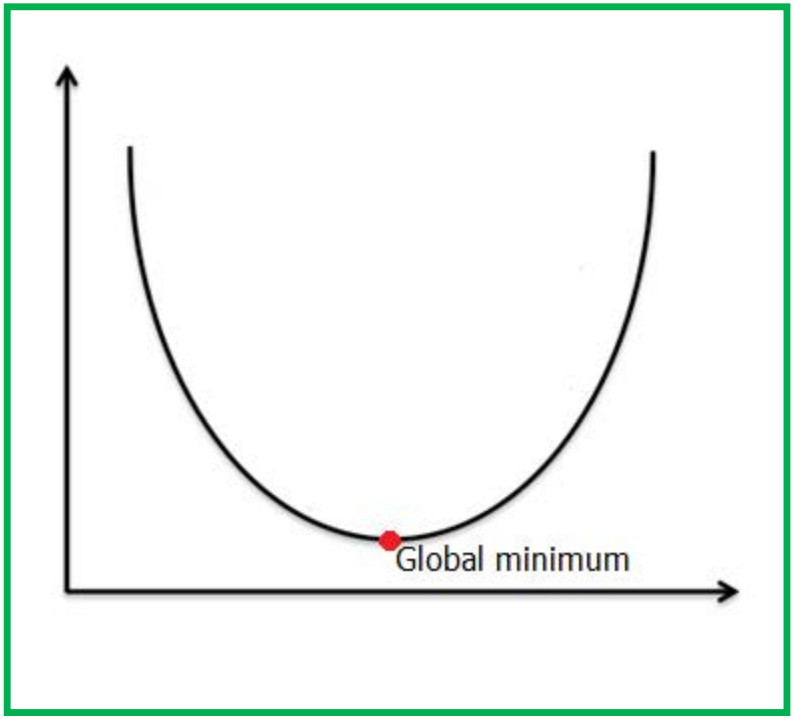
return $\boldsymbol{w}^T, w_0^T$

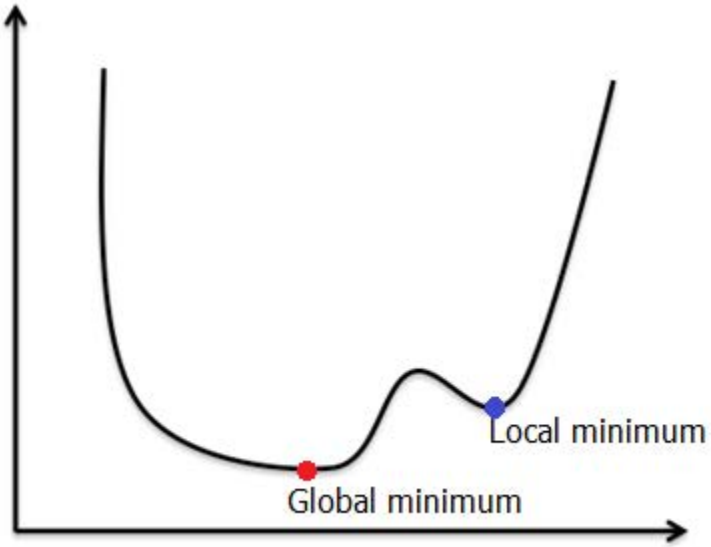
Will gradient descent always find same solution?



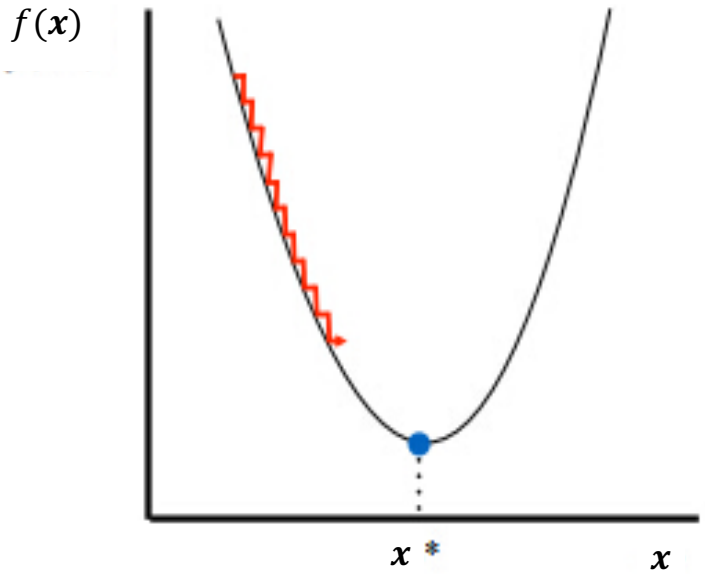


Log loss is convex!

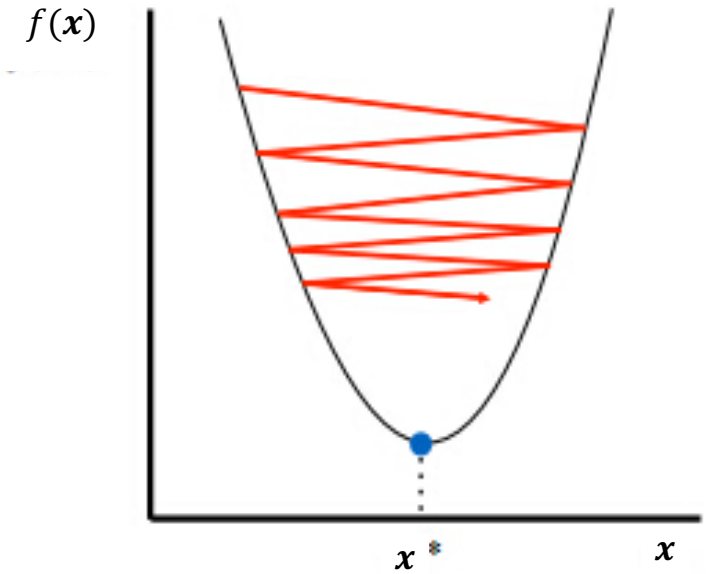




Intuition: 1D minimization

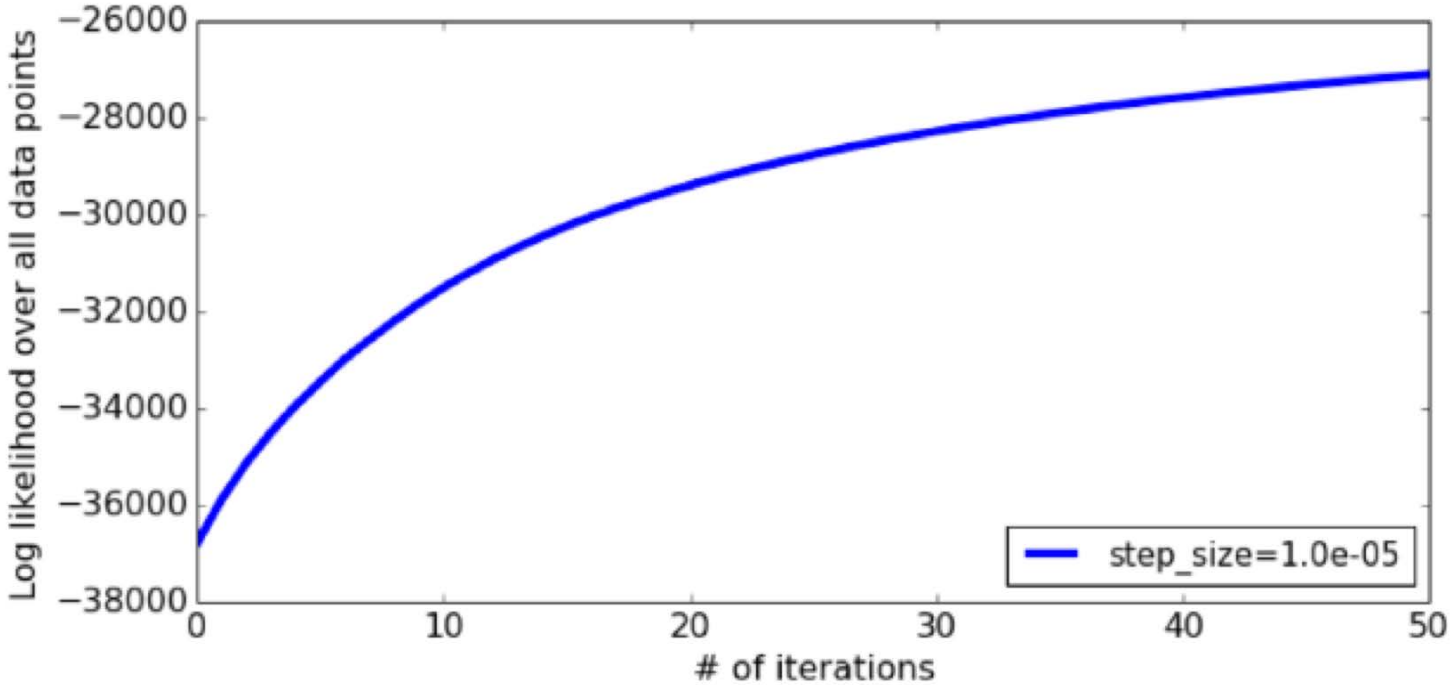


Too small: converge very slowly

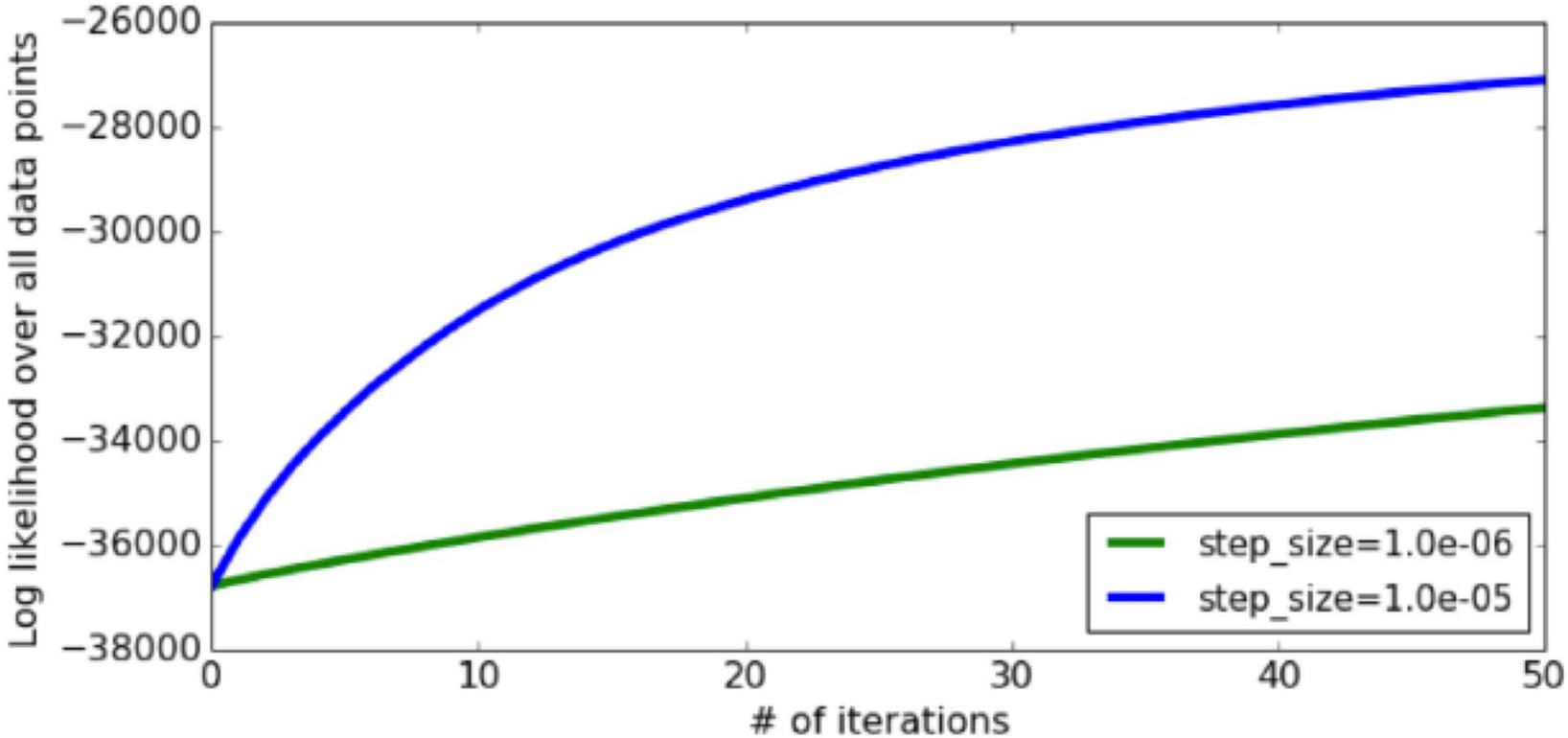


Too big: overshoot and even diverge

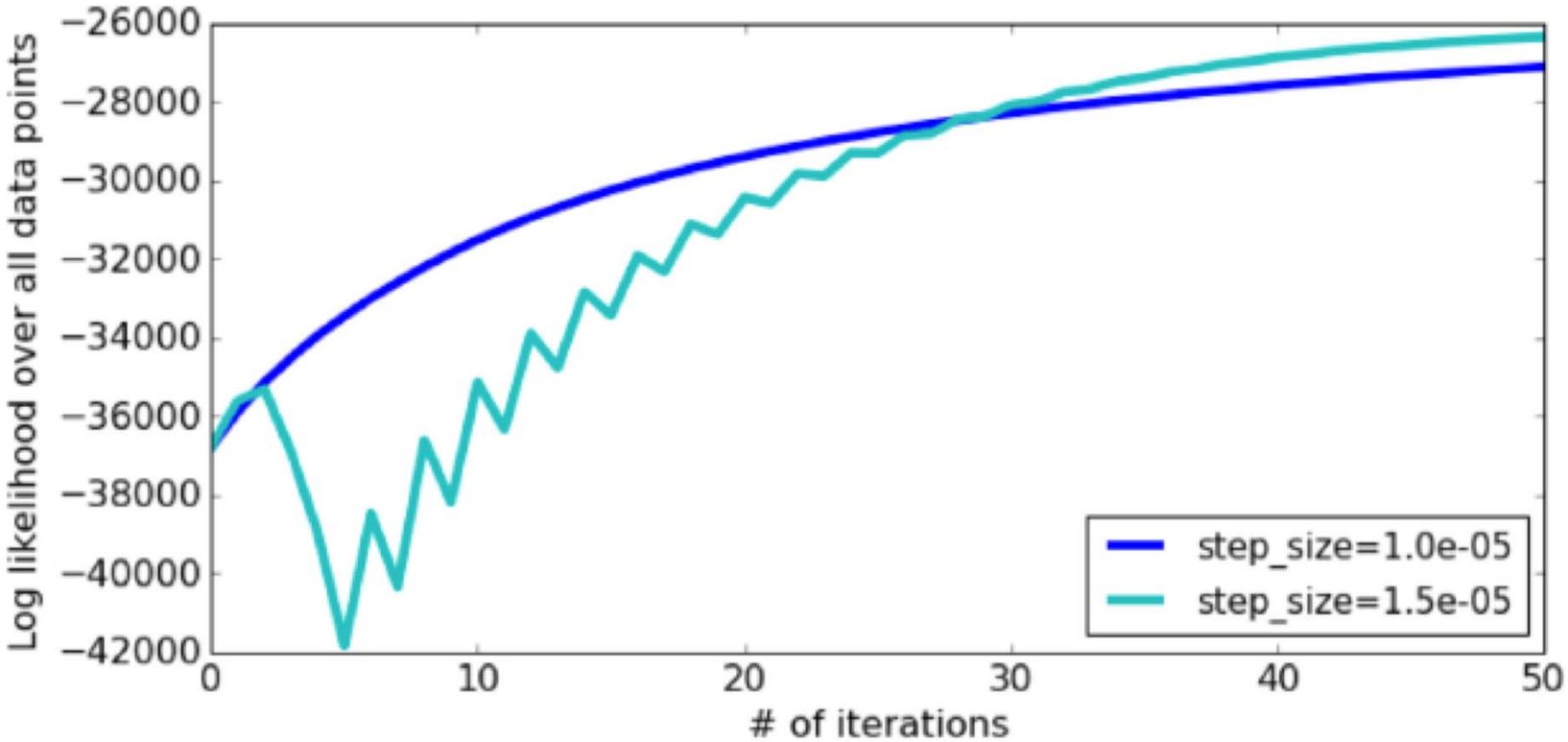
Log likelihood vs iterations



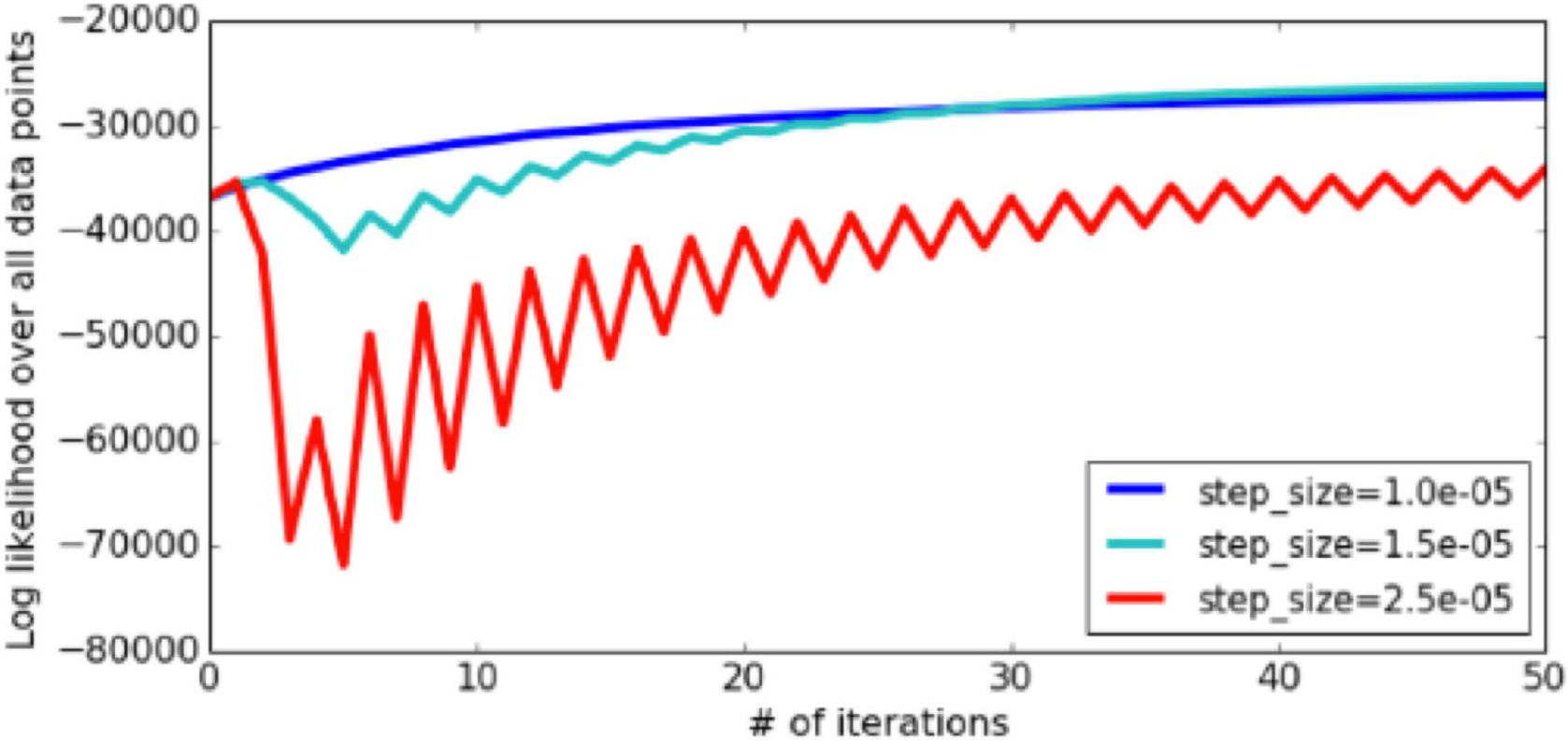
If step size is **too small**



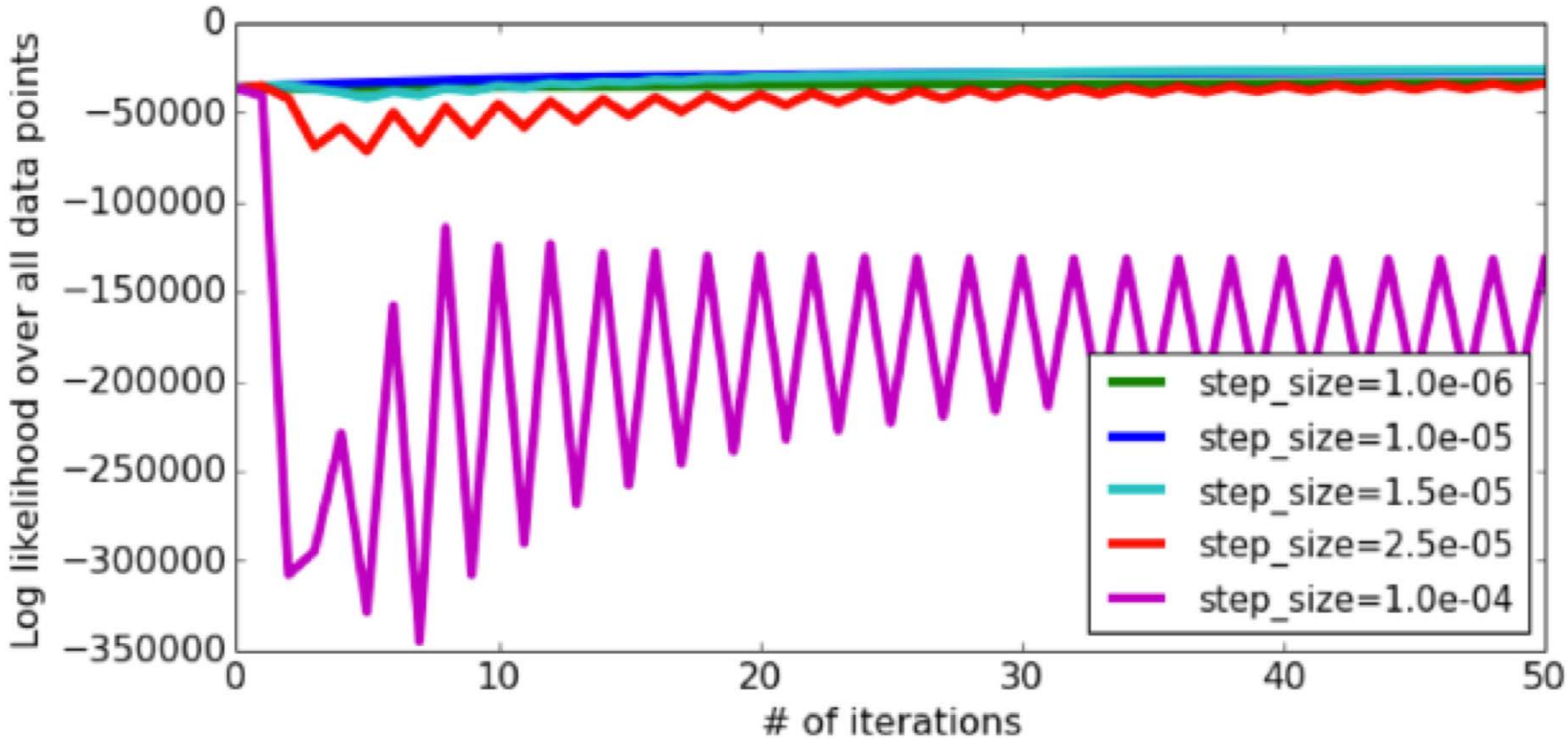
If step size is **large**



If step size is **too large**



If step size is **way too large**



Rule for picking step sizes

- Never try just one!
- Try several values (exponentially spaced) until
  - Find one clearly too small
  - Find one clearly too large (oscillation / divergence)
- Always make trace plots!
  - Show the loss, norm of gradient, and parameters

- Smarter choices for step size:
  - Decaying methods
  - Search methods
  - Adaptive methods

Decaying step sizes

- Decay over iterations

Searching for good step sizes

- Line Search

  scipy.optimize.line_search