Logistic regression

This observation gives rise to another class of plugin methods, the most important of which is logistic regression, which implements the following strategy

1. Assume $\eta(\mathbf{x}) = \dfrac{1}{1 + e^{-(\mathbf{w}^T\mathbf{x}+b)}} \quad (\mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R})$

2. Directly estimate $\mathbf{w}, b$ (somehow) from the data

3. Plug the estimate

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

   into the formula for the Bayes classifier

Estimating the parameters

Challenge: How to estimate the parameters for

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

One possibility: $\mathbf{w} = \widehat{\Sigma}^{-1}(\widehat{\boldsymbol{\mu}}_1 - \widehat{\boldsymbol{\mu}}_0)$

$$b = \tfrac{1}{2}\widehat{\boldsymbol{\mu}}_0^T\widehat{\Sigma}^{-1}\widehat{\boldsymbol{\mu}}_0 - \tfrac{1}{2}\widehat{\boldsymbol{\mu}}_1^T\widehat{\Sigma}^{-1}\widehat{\boldsymbol{\mu}}_1 + \log\frac{\widehat{\pi}_1}{\widehat{\pi}_0}$$

Alternative: *Maximum likelihood estimation*

For convenience, set $\boldsymbol{\theta} = (b, \mathbf{w})$

Note that $\eta(\mathbf{x})$ is really a function of both $\mathbf{x}$ and $\boldsymbol{\theta}$, so we will use the notation $\eta(\mathbf{x}; \boldsymbol{\theta})$ to highlight this dependence

The *a posteriori* probability of our data

Suppose that we knew $\boldsymbol{\theta}$. Then we could compute

$$\begin{aligned}\mathbb{P}[y_i|\mathbf{x}_i;\boldsymbol{\theta}] &= \mathbb{P}[Y_i = y_i|X_i = \mathbf{x}_i;\boldsymbol{\theta}] \\ &= \begin{cases}\eta(\mathbf{x}_i;\boldsymbol{\theta}) & \text{if } y_i = 1 \\ 1-\eta(\mathbf{x}_i;\boldsymbol{\theta}) & \text{if } y_i = 0\end{cases} \\ &= \eta(\mathbf{x}_i;\boldsymbol{\theta})^{y_i}(1-\eta(\mathbf{x}_i;\boldsymbol{\theta}))^{1-y_i}\end{aligned}$$

Because of independence, we also have that

$$\begin{aligned}\mathbb{P}[y_1,\ldots,y_n|\mathbf{x}_1,\ldots\mathbf{x}_n;\boldsymbol{\theta}] &= \prod_{i=1}^{n}\mathbb{P}[y_i|\mathbf{x}_i;\boldsymbol{\theta}] \\ &= \prod_{i=1}^{n}\eta(\mathbf{x}_i;\boldsymbol{\theta})^{y_i}(1-\eta(\mathbf{x}_i;\boldsymbol{\theta}))^{1-y_i}\end{aligned}$$

Maximum likelihood estimation

We don't actually know $\boldsymbol{\theta}$, but we do know $y_1, \ldots, y_n$

Suppose we view $y_1, \ldots, y_n$ to be fixed, and view $\mathbb{P}[y_1, \ldots, y_n | \mathbf{x}_1, \ldots \mathbf{x}_n; \boldsymbol{\theta}]$ as just a function of $\boldsymbol{\theta}$

When we do this, $\mathcal{L}(\boldsymbol{\theta}) = \mathbb{P}[y_1, \ldots, y_n | \mathbf{x}_1, \ldots \mathbf{x}_n; \boldsymbol{\theta}]$ is called the ***likelihood*** (or likelihood function)

The method of ***maximum likelihood*** aims to estimate $\boldsymbol{\theta}$ by finding the $\boldsymbol{\theta}$ that ***maximizes*** the ***likelihood*** $\mathcal{L}(\boldsymbol{\theta})$

In practice, it is often more convenient to focus on maximizing the ***log-likelihood***, i.e., $\log \mathcal{L}(\boldsymbol{\theta})$

The log-likelihood

To see why, note that the likelihood in our case is given by

$$\mathcal{L}(\boldsymbol{\theta}) = \prod_{i=1}^{n} \eta(\mathbf{x}_i; \boldsymbol{\theta})^{y_i} (1 - \eta(\mathbf{x}_i; \boldsymbol{\theta}))^{1-y_i}$$

Thus, the log-likelihood is given by

$$\begin{aligned}
\ell(\boldsymbol{\theta}) &= \log \mathcal{L}(\boldsymbol{\theta}) \\
&= \sum_{i=1}^{n} y_i \log \eta(\mathbf{x}_i; \boldsymbol{\theta}) + (1 - y_i) \log(1 - \eta(\mathbf{x}_i; \boldsymbol{\theta})) \\
&= \sum_{i=1}^{n} y_i \boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i - \log(1 + e^{\boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i})
\end{aligned}$$

$$\widetilde{\mathbf{x}} = [1, x(1), \ldots, x(d)]^T \quad \boldsymbol{\theta} = [b, w(1), \ldots, w(d)]^T$$

Maximizing the log-likelihood

How can we maximize

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} y_i \boldsymbol{\theta}^T \tilde{\mathbf{x}}_i - \log(1 + e^{\boldsymbol{\theta}^T \tilde{\mathbf{x}}_i})$$

with respect to $\boldsymbol{\theta}$?

Find a $\boldsymbol{\theta}$ such that $\nabla \ell(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_1} \\ \vdots \\ \frac{\partial \ell(\boldsymbol{\theta})}{\partial \theta_{d+1}} \end{bmatrix} = 0$

(i.e., compute the partial derivatives and set them to zero)

Computing the gradient

It is not too hard to show that

$$\nabla \ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \nabla \left(y_i \boldsymbol{\theta}^T \widetilde{\mathbf{x}}_i - \log(1 + e^{\boldsymbol{\theta}^T \tilde{\mathbf{x}}_i}) \right)$$

$$= \sum_{i=1}^{n} \widetilde{\mathbf{x}}_i \left(y_i - \frac{1}{1 + e^{-\boldsymbol{\theta}^T \tilde{\mathbf{x}}_i}} \right) = 0$$

This gives us $d + 1$ equations, but they are ***nonlinear*** and have no closed-form solution

How can we solve this problem?

Optimization

Throughout signal processing and machine learning, we will very often encounter problems of the form

$$\underset{\mathbf{x}\in\mathbb{R}^d}{\text{minimize}}\, f(\mathbf{x})$$

(or $\underset{\boldsymbol{\theta}\in\mathbb{R}^{d+1}}{\text{minimize}}\, -\ell(\boldsymbol{\theta})$ for today)

In many (most?) cases, we cannot compute the solution simply by setting $\nabla f(\mathbf{x}) = 0$ and solving for $\mathbf{x}$

However, there are many powerful ***algorithms*** for finding $\mathbf{x}$ using a computer

Gradient descent

A simple way to try to find the minimum of our objective function is to iteratively *"roll downhill"*

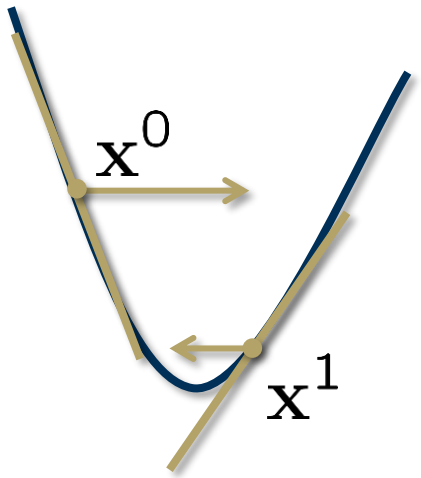
From $\mathbf{x}^0$, take a step in the direction of the negative gradient

$$\mathbf{x}^1 = \mathbf{x}^0 - \alpha_0 \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^0}$$

$$\mathbf{x}^2 = \mathbf{x}^1 - \alpha_1 \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^1}$$

$$\vdots$$

α_0 : "step size"



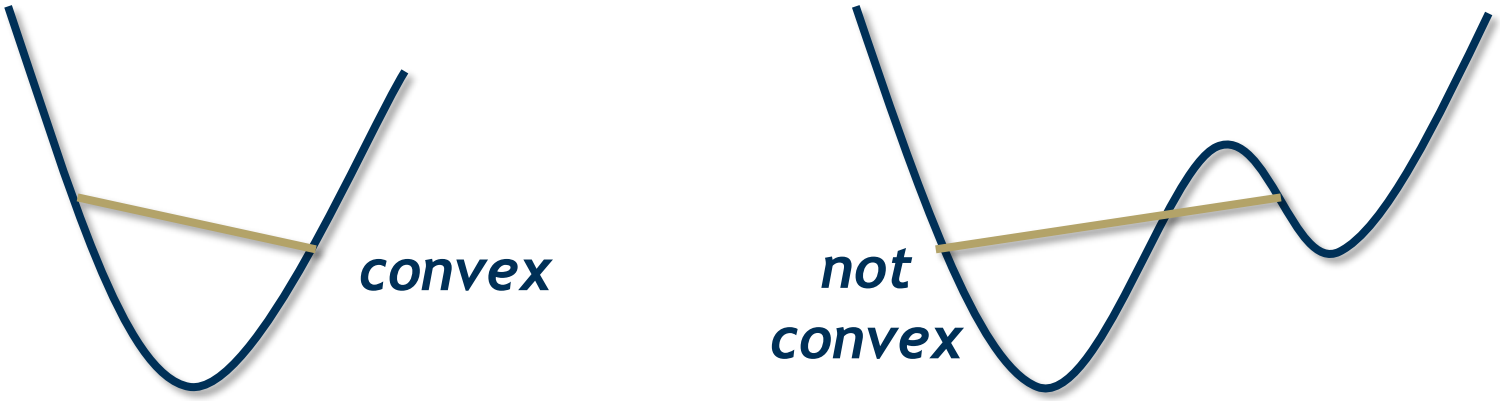
Convergence of gradient descent

The core iteration of gradient descent is to compute

$$\mathbf{x}^{j+1} = \mathbf{x}^j - \alpha_j \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^j}$$

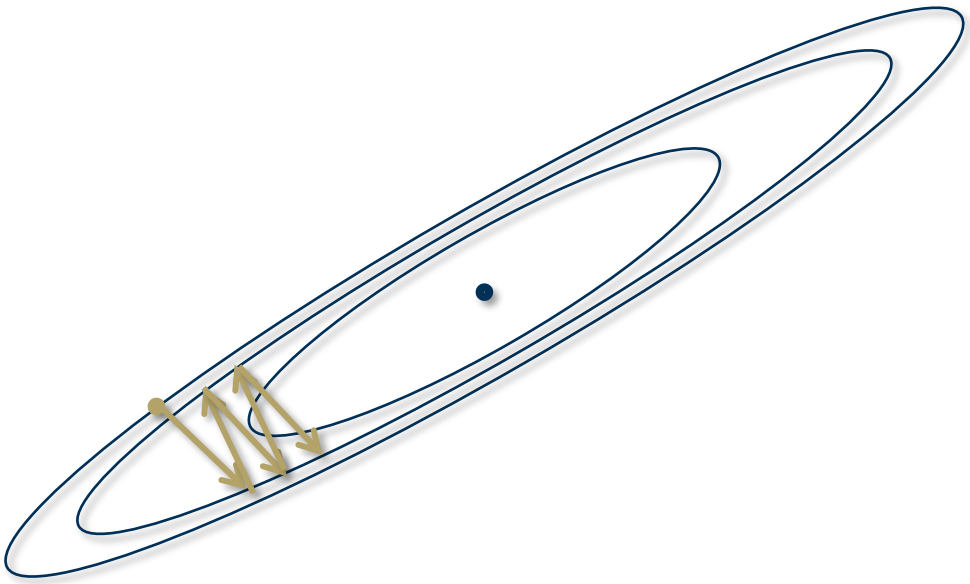
Note that if $\nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^j} = 0$, then we have found the minimum and $\mathbf{x}^{j+1} = \mathbf{x}^j$, so the algorithm will terminate

If f is convex and sufficiently smooth, then gradient descent (with a fixed step size α) is guaranteed to converge to the global minimum of f



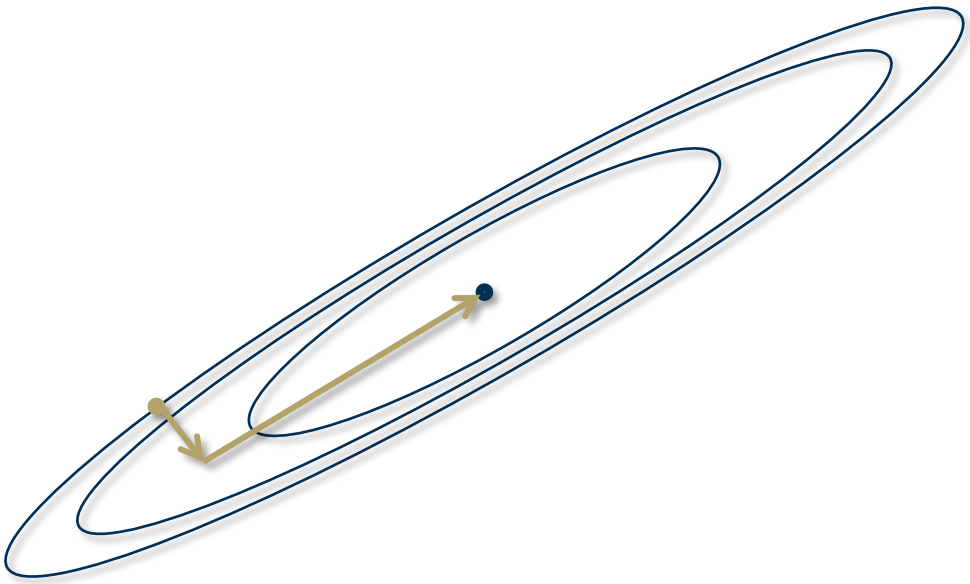
Step size matters!

Even though gradient descent provably converges, it can potentially take a while



Step size matters!

Even though gradient descent provably converges, it can potentially take a while



Newton's method

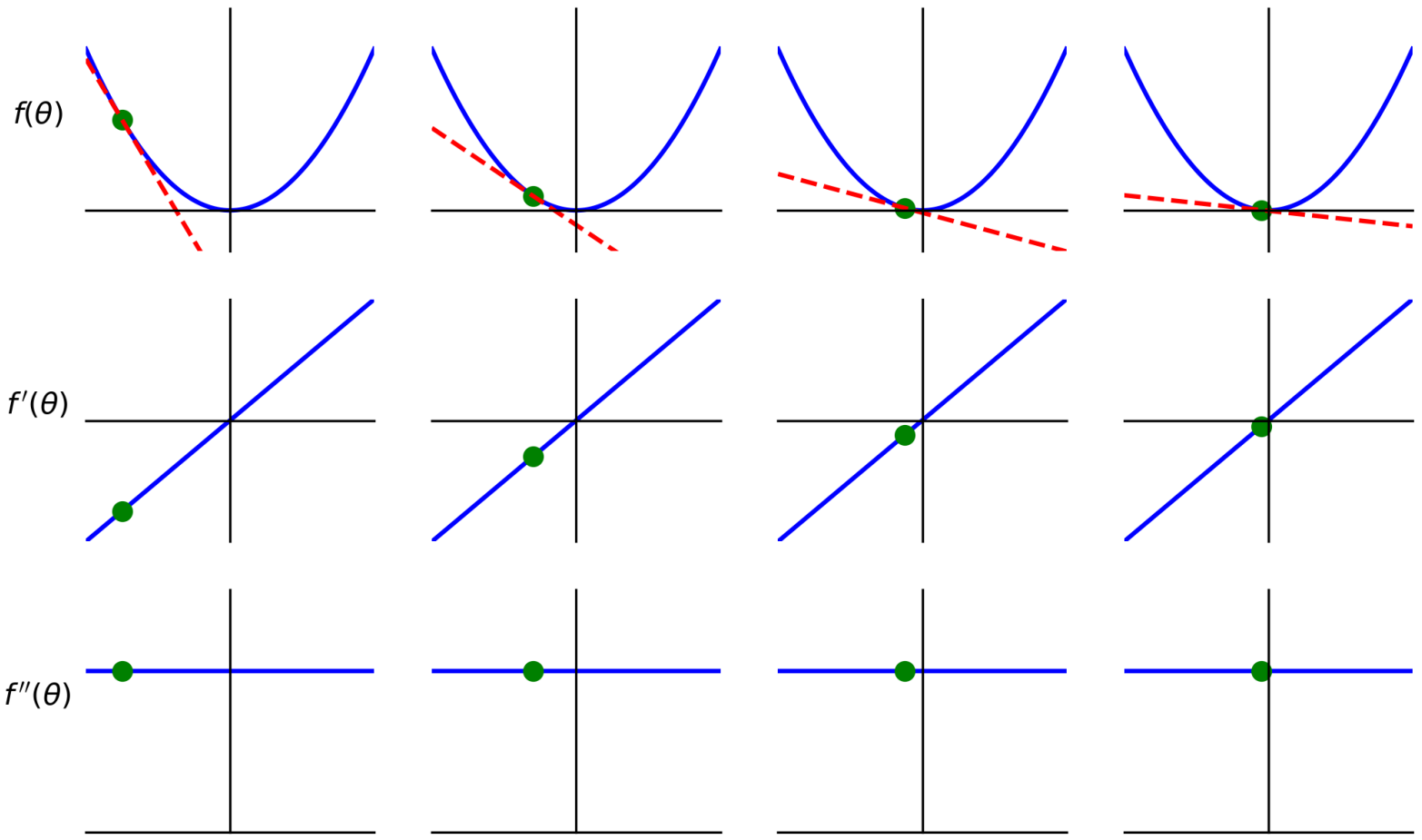
Also known as the Newton-Raphson method, this approach can be viewed as making a ***quadratic*** approximation to f at each iteration (instead of a linear one)

$$\mathbf{x}^{j+1} = \mathbf{x}^j - \left(\nabla^2 f(\mathbf{x})\right)^{-1} \nabla f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}^j}$$

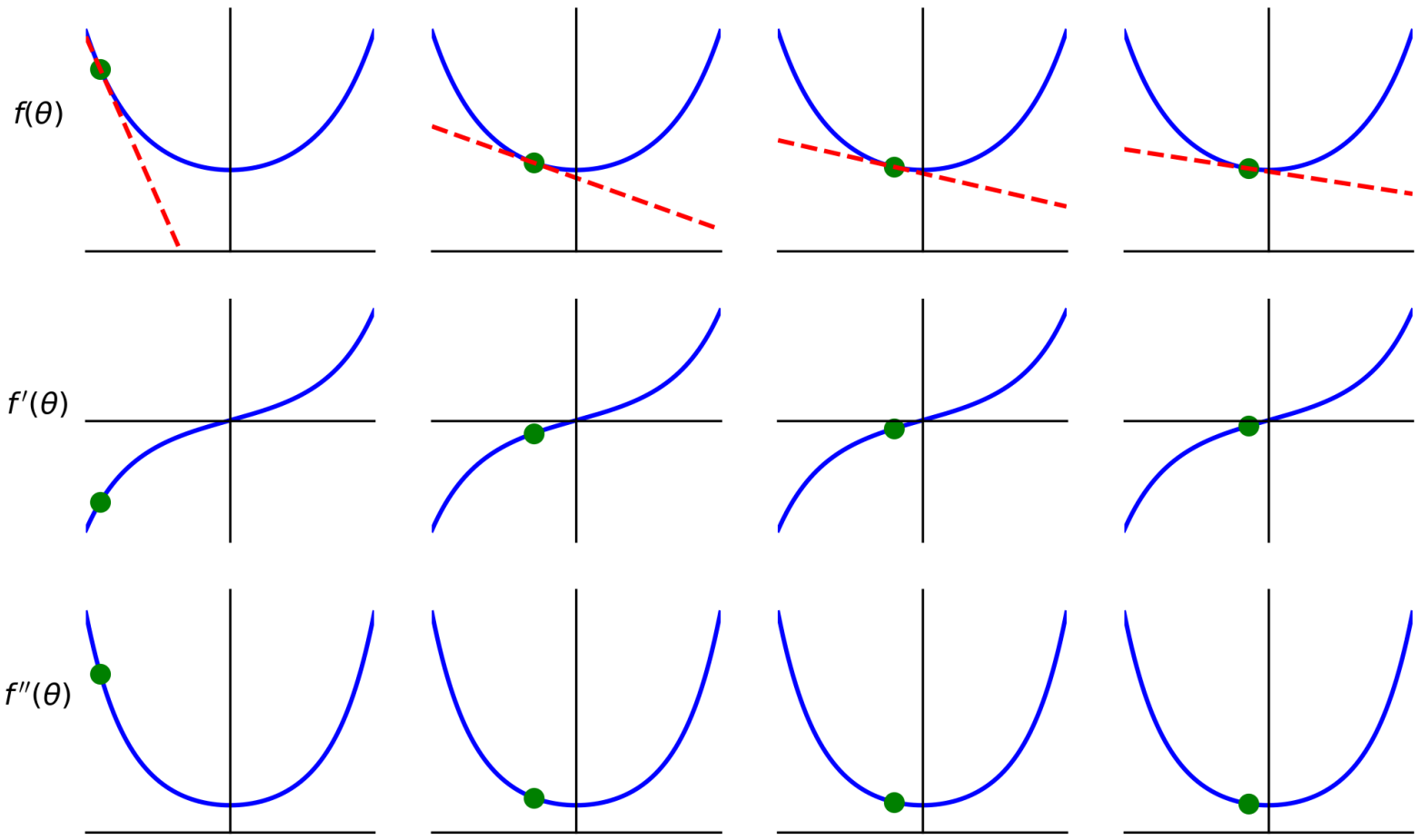
Hessian matrix

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_d} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_d \partial x_1} & \frac{\partial^2 f}{\partial x_d \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_d^2} \end{bmatrix}$$

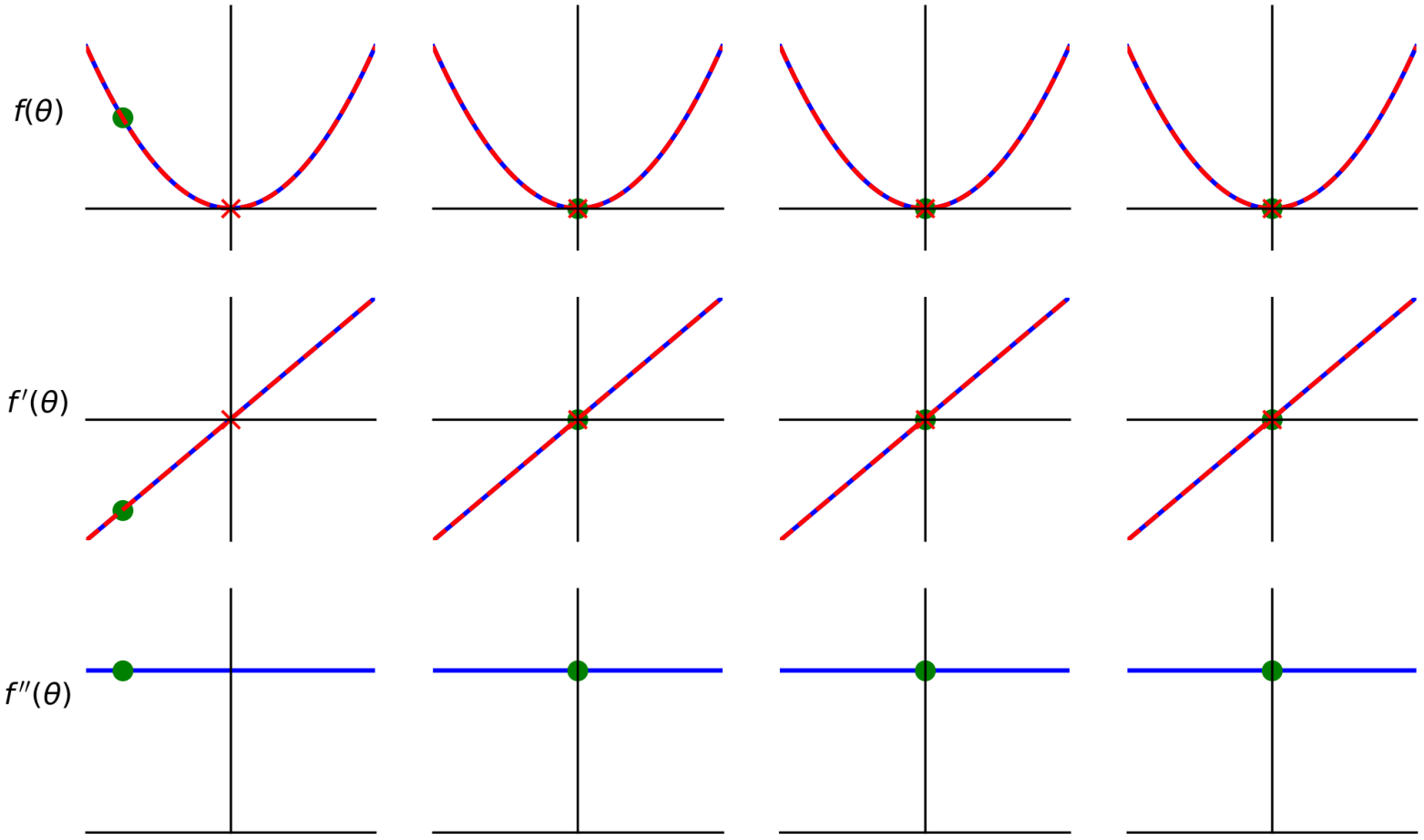
Gradient Descent for $f(\theta) = \theta^2$



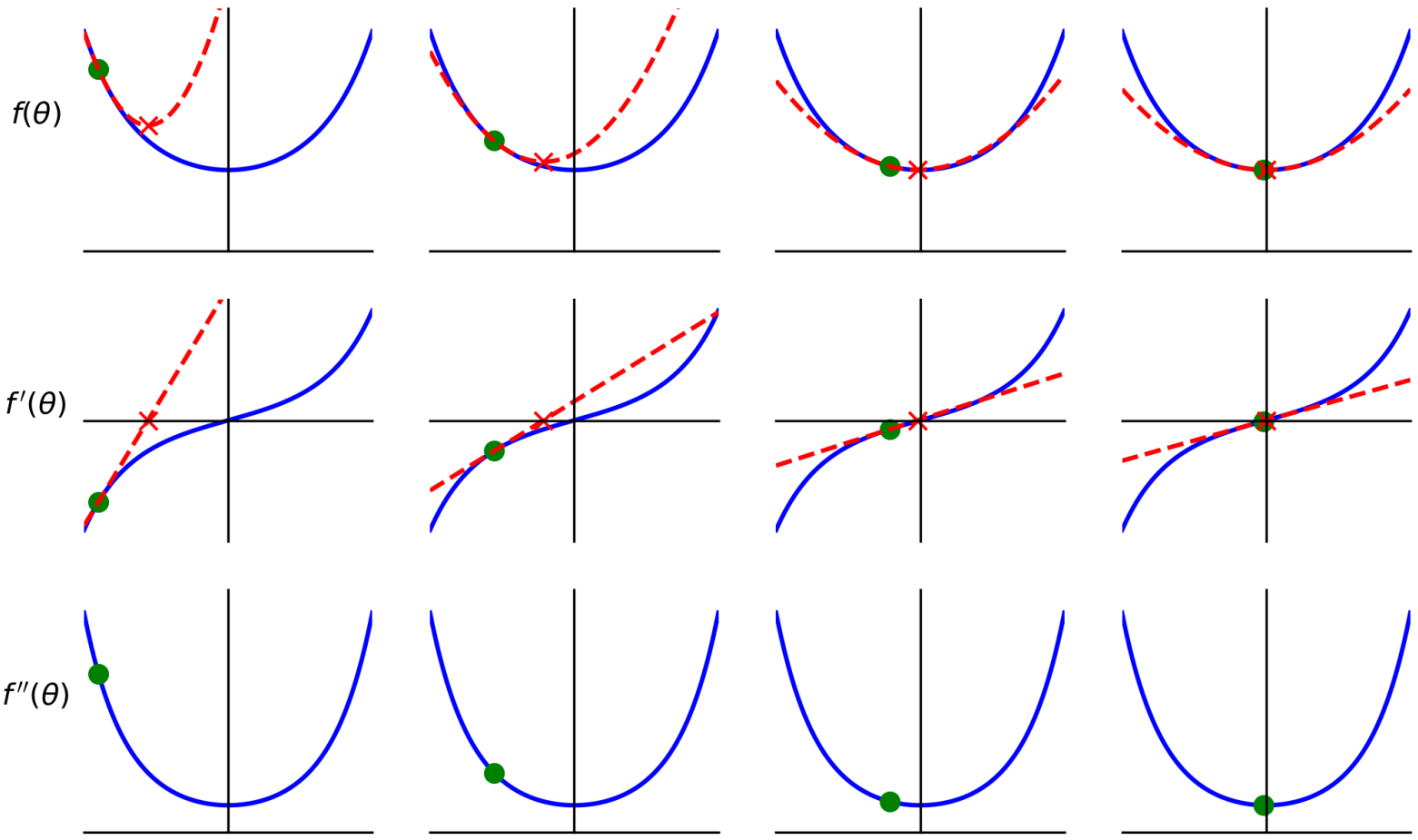
Gradient Descent for $f(\theta) = \exp(\theta^2)$



Newton's Method for $f(\theta) = \theta^2$



Newton's Method for $f(\theta) = \exp(\theta^2)$



Optimization for logistic regression

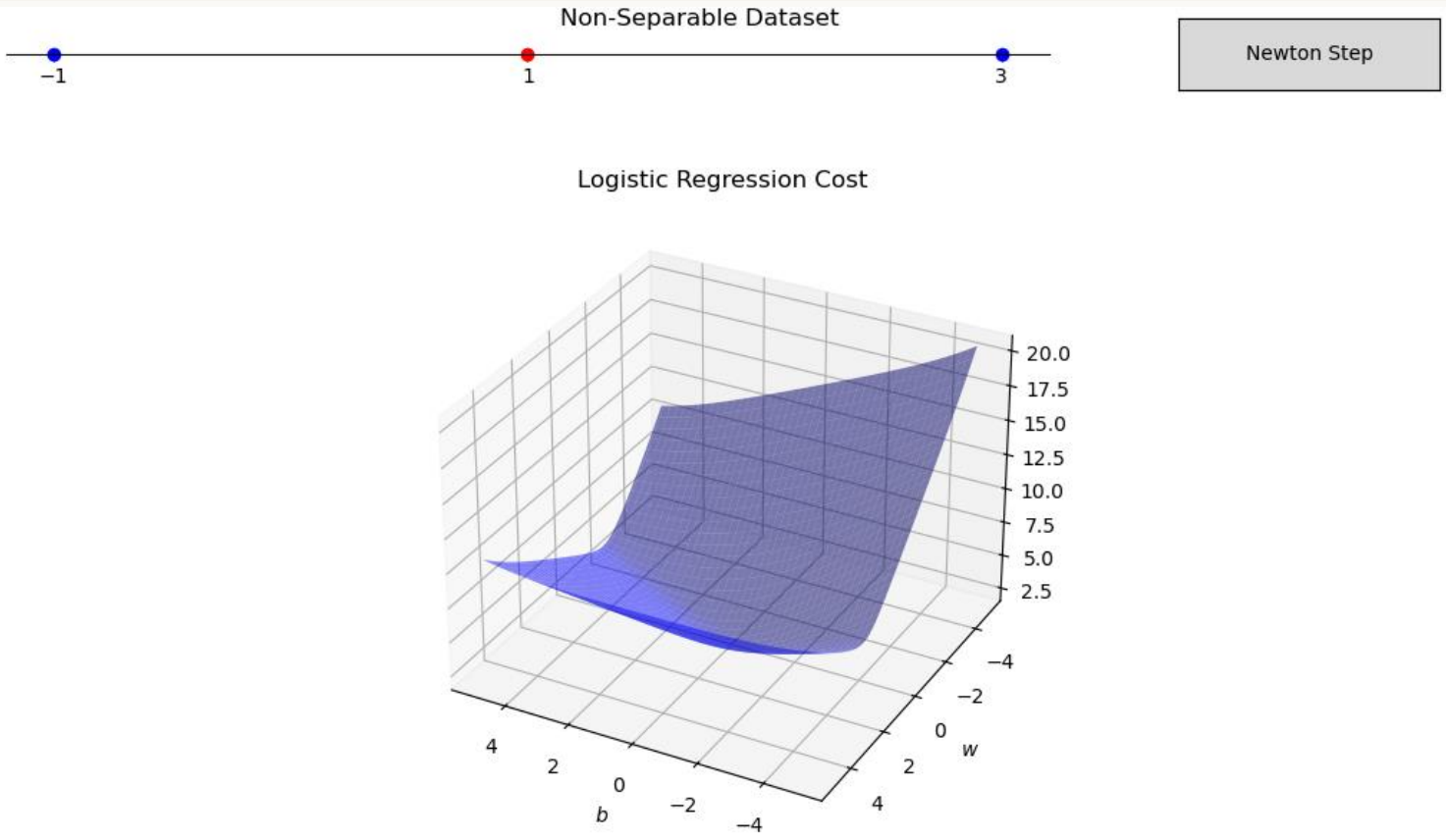
The negative log-likelihood in logistic regression is a convex function

Both gradient descent and Newton's method are common strategies for setting the parameters in logistic regression

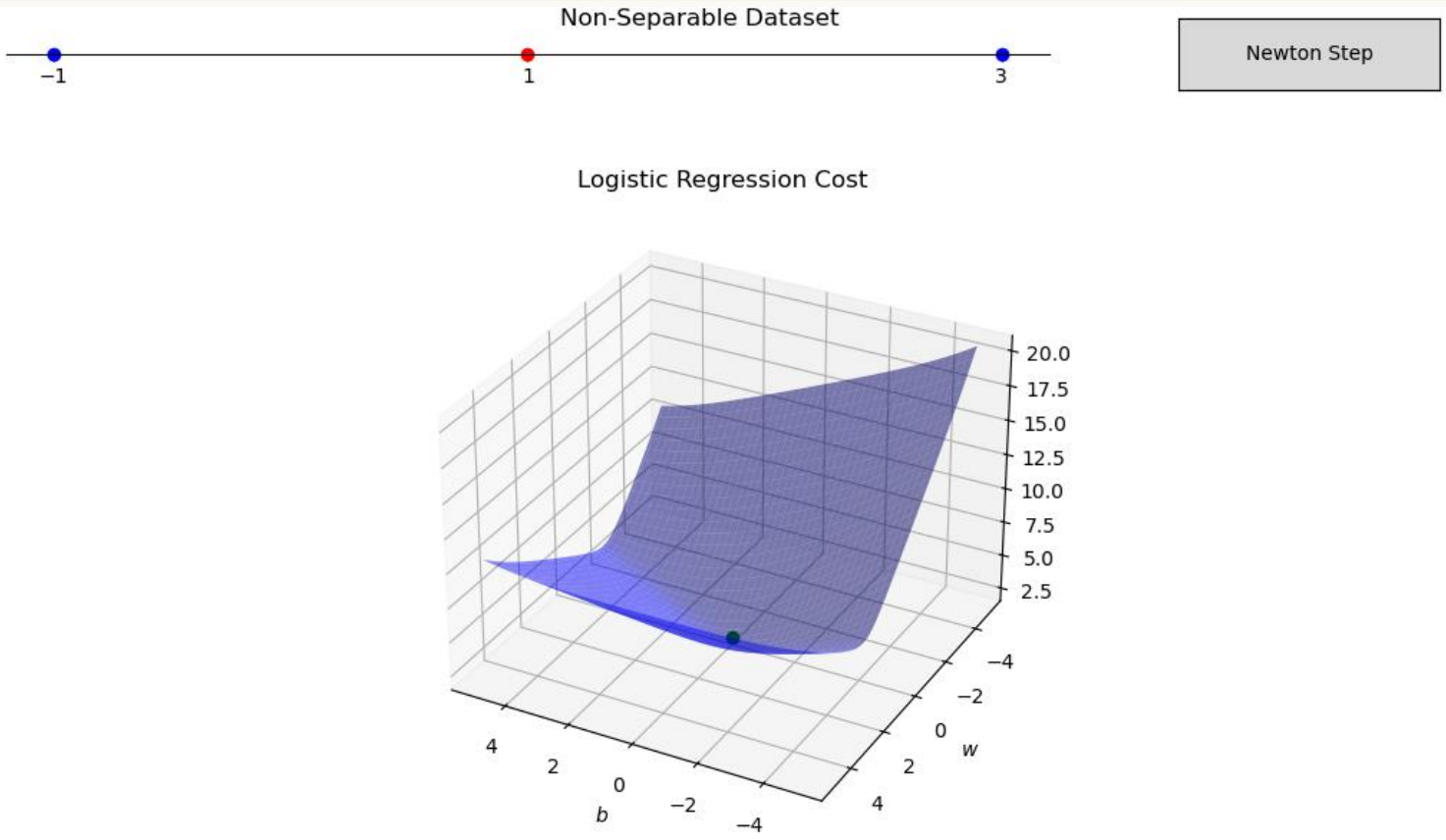
Newton's method is much faster when the dimension d is small, but is impractical when d is large

Why?

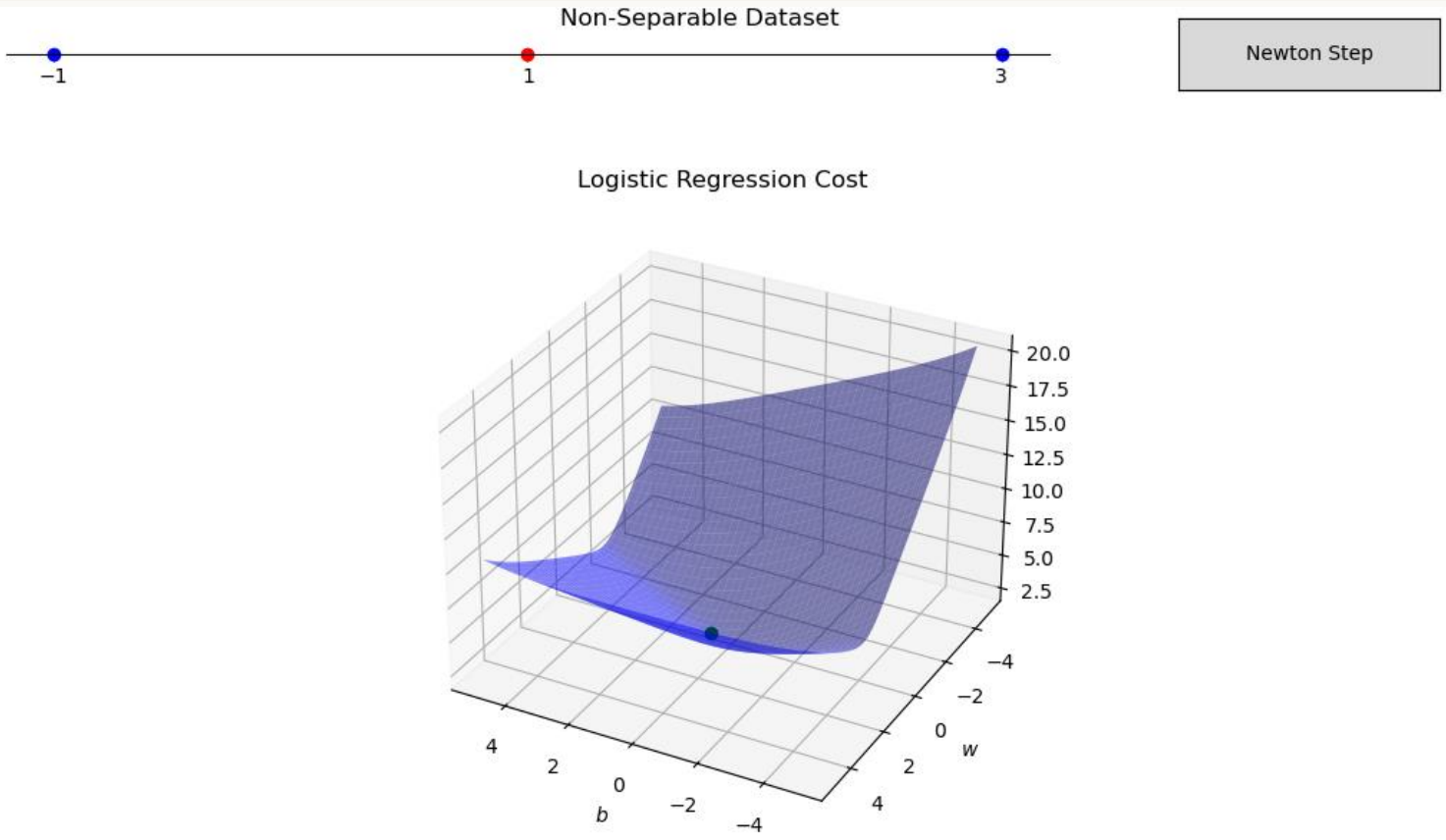
More on this on next week's homework



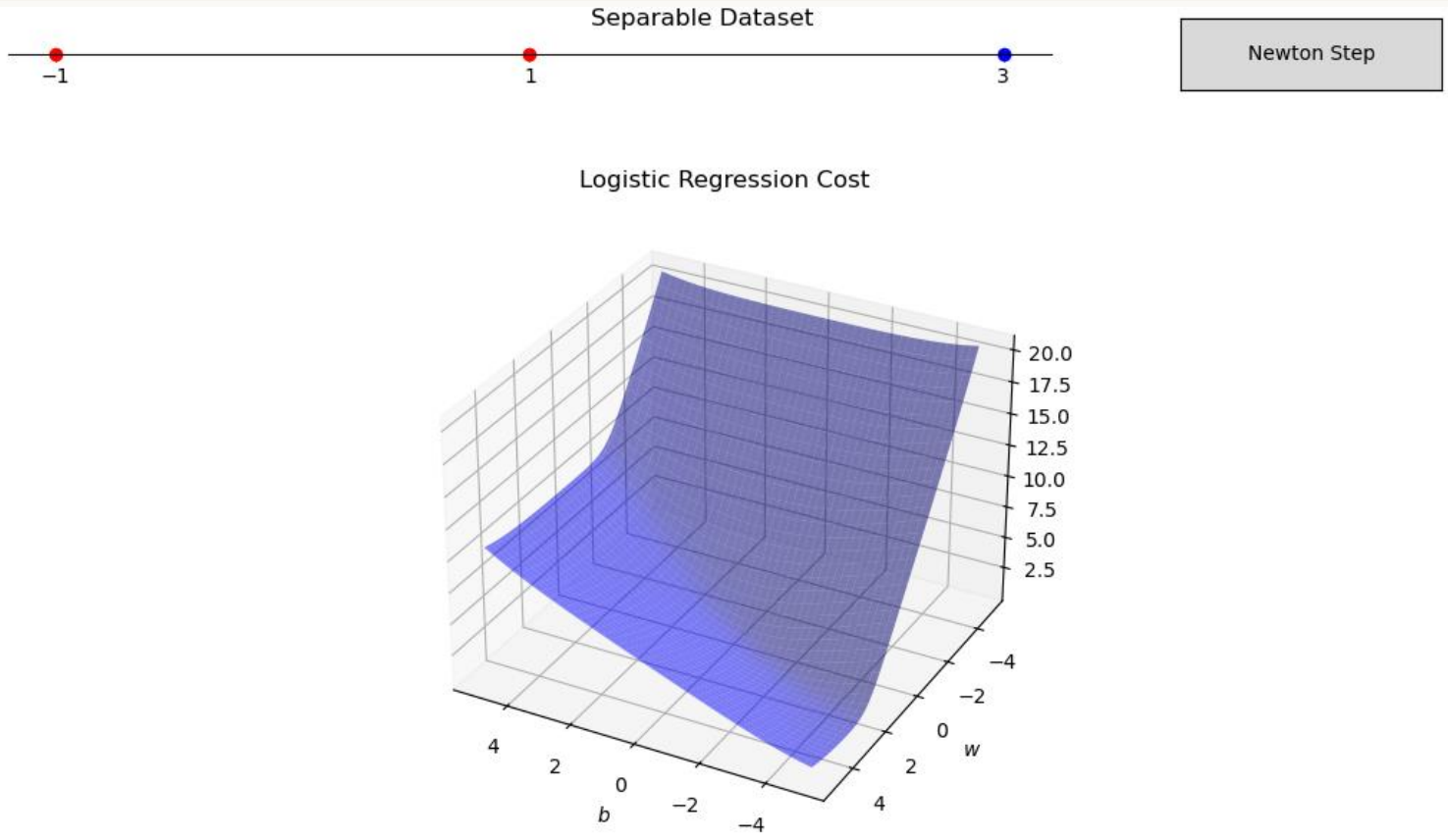
Landscape looks quadratic near solution, Newton's method works well



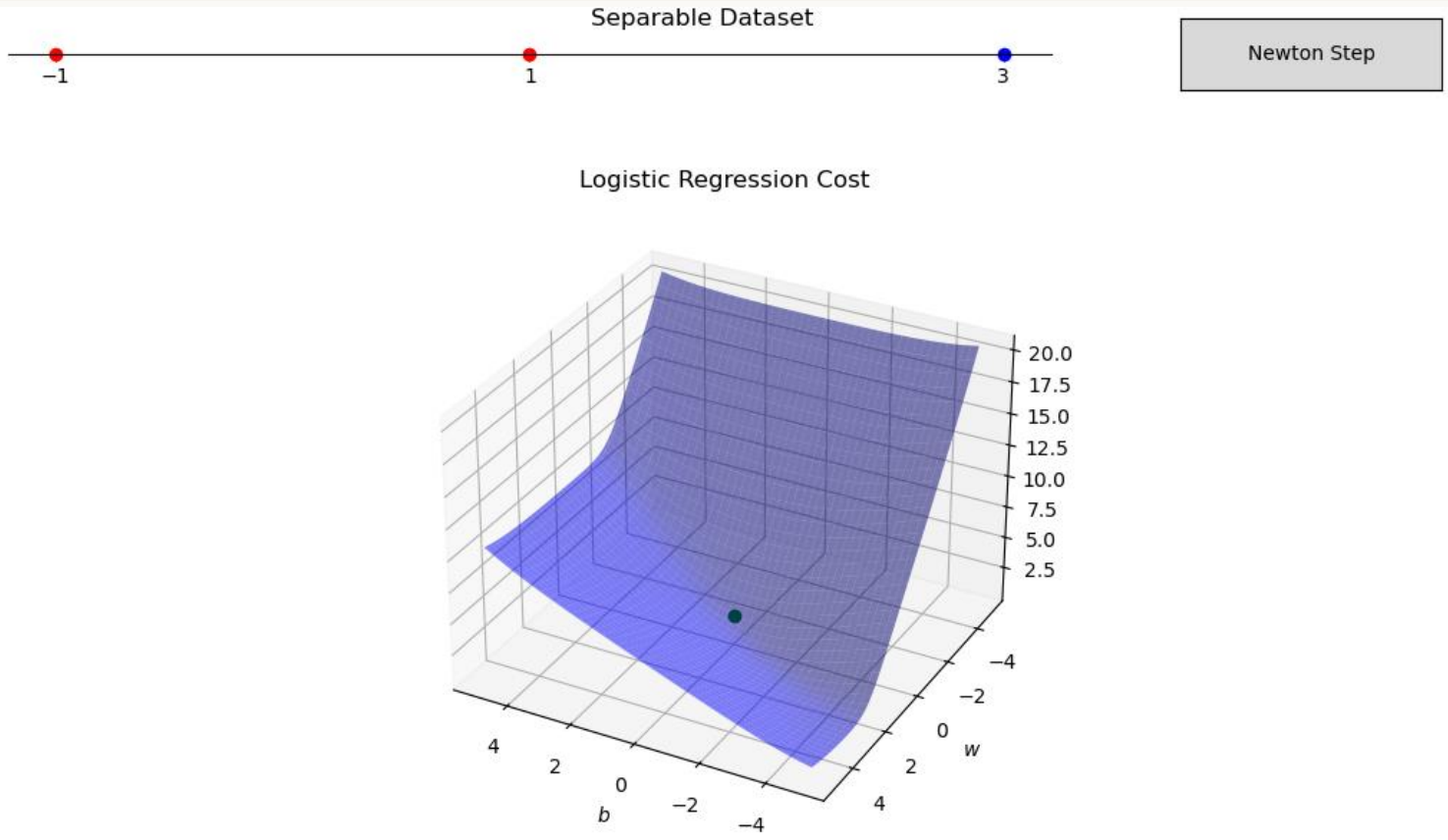
Landscape looks quadratic near solution, Newton's method works well



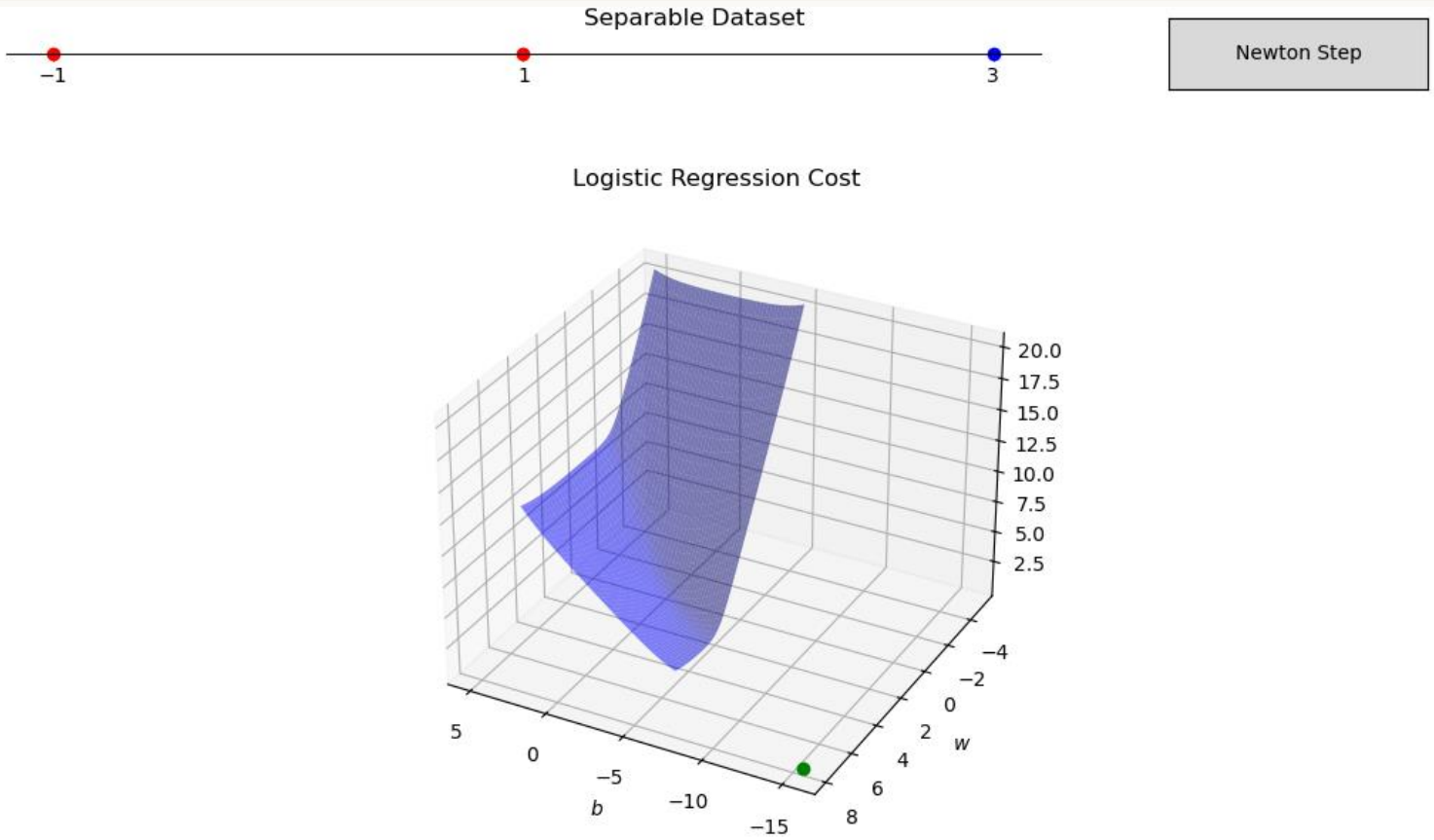
Landscape looks quadratic near solution, Newton's method works well



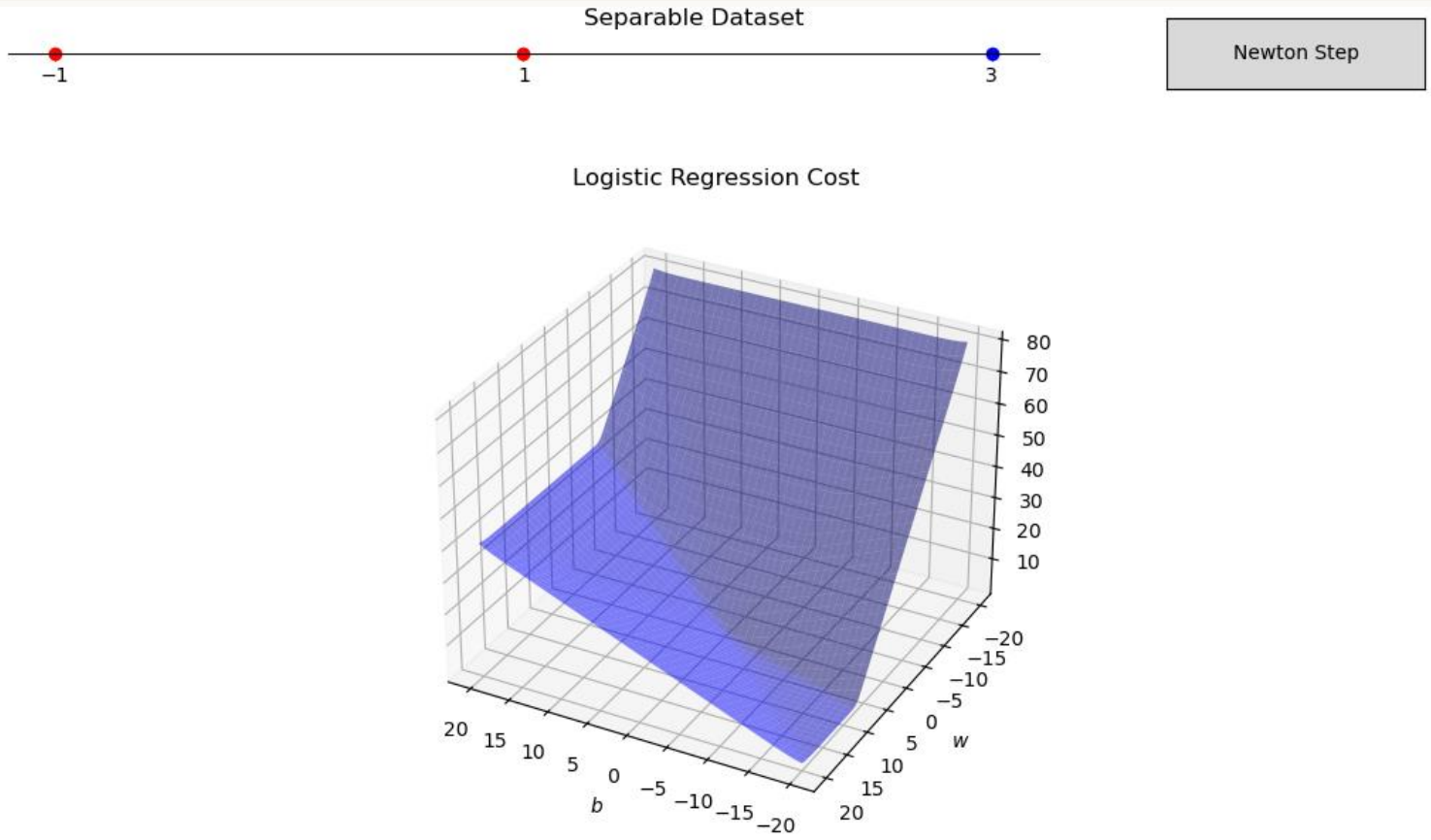
Separable dataset has no unique minimizer, Newton's method sends us to infinity



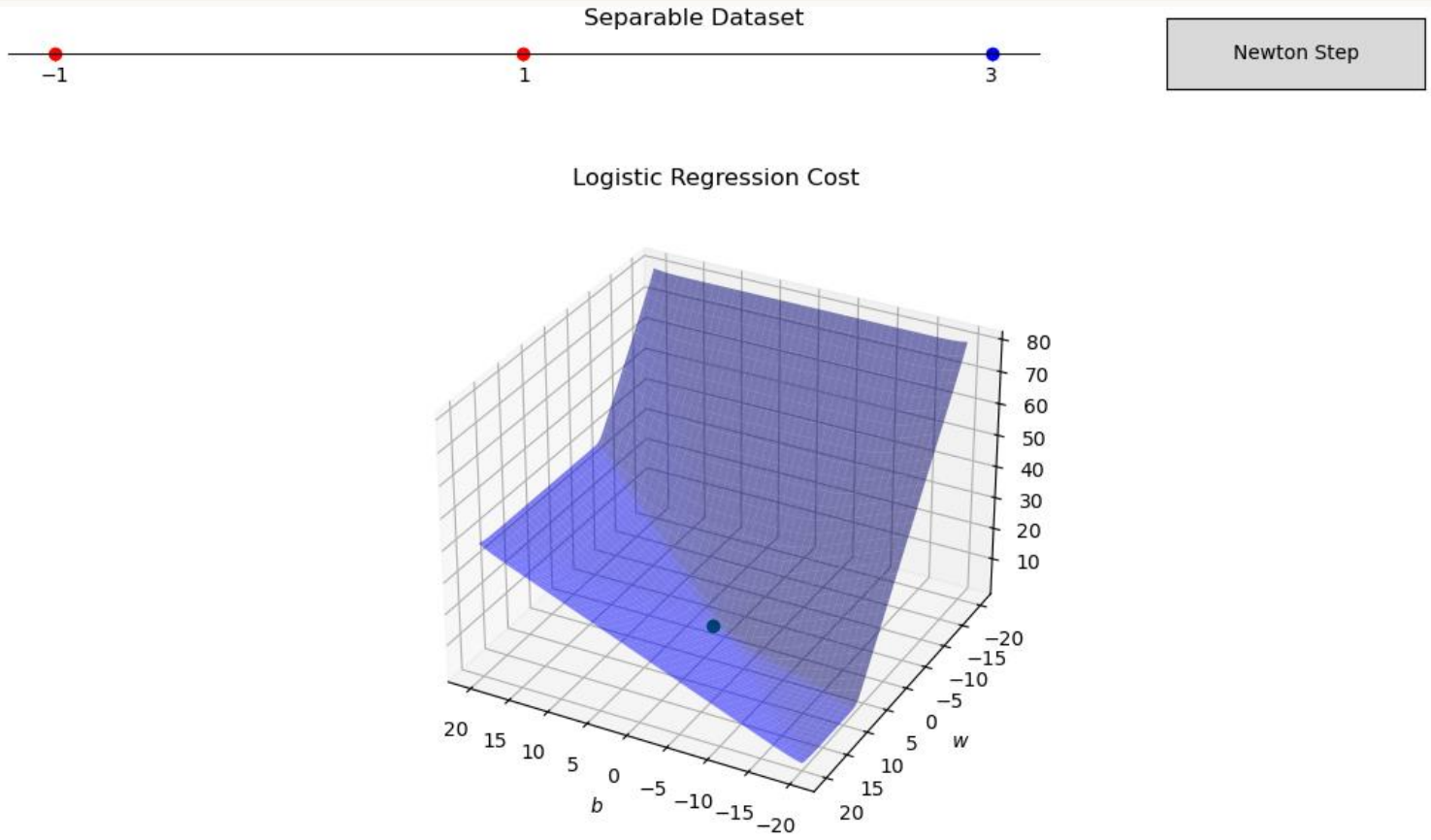
Separable dataset has no unique minimizer, Newton's method sends us to infinity



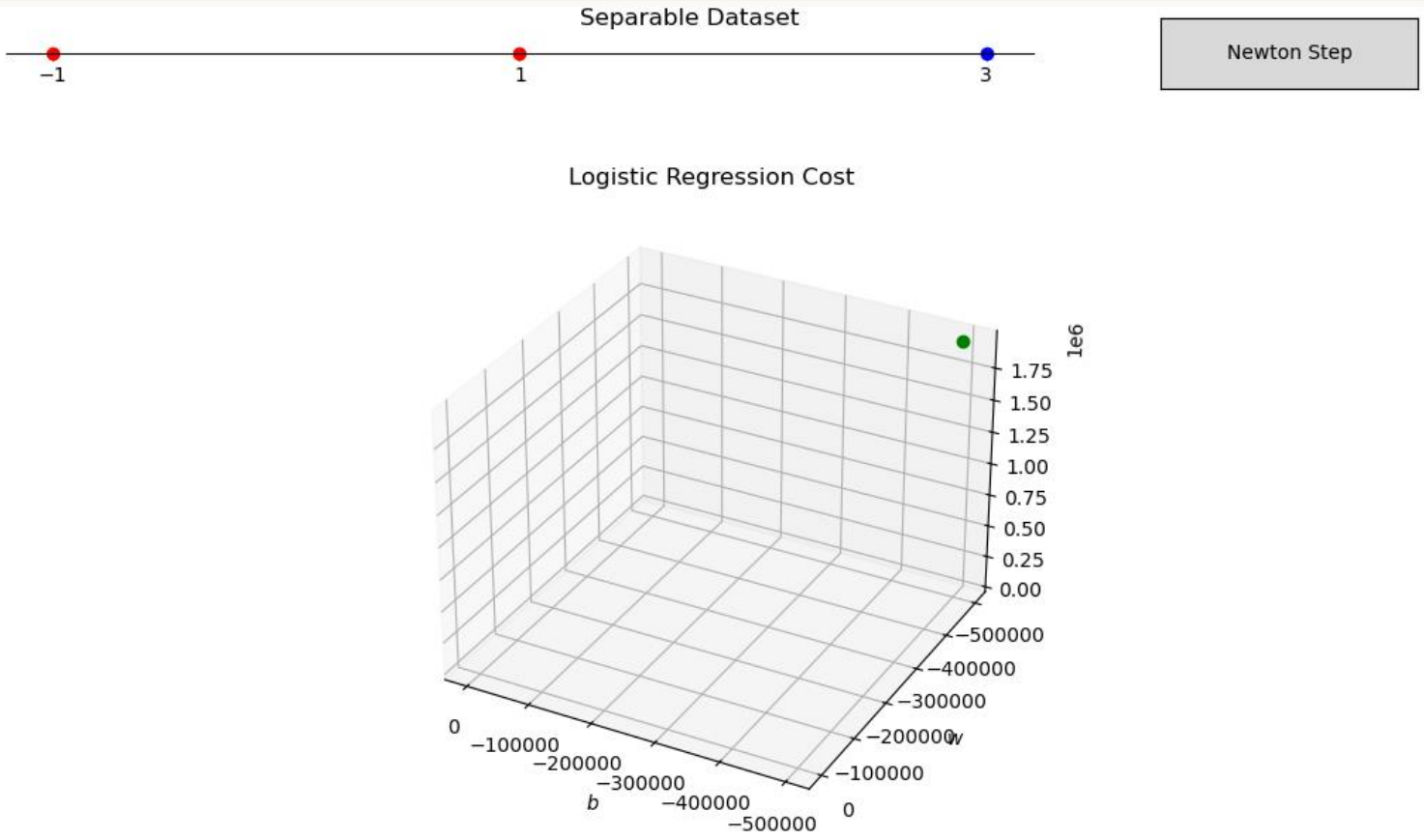
Separable dataset has no unique minimizer, Newton's method sends us to infinity



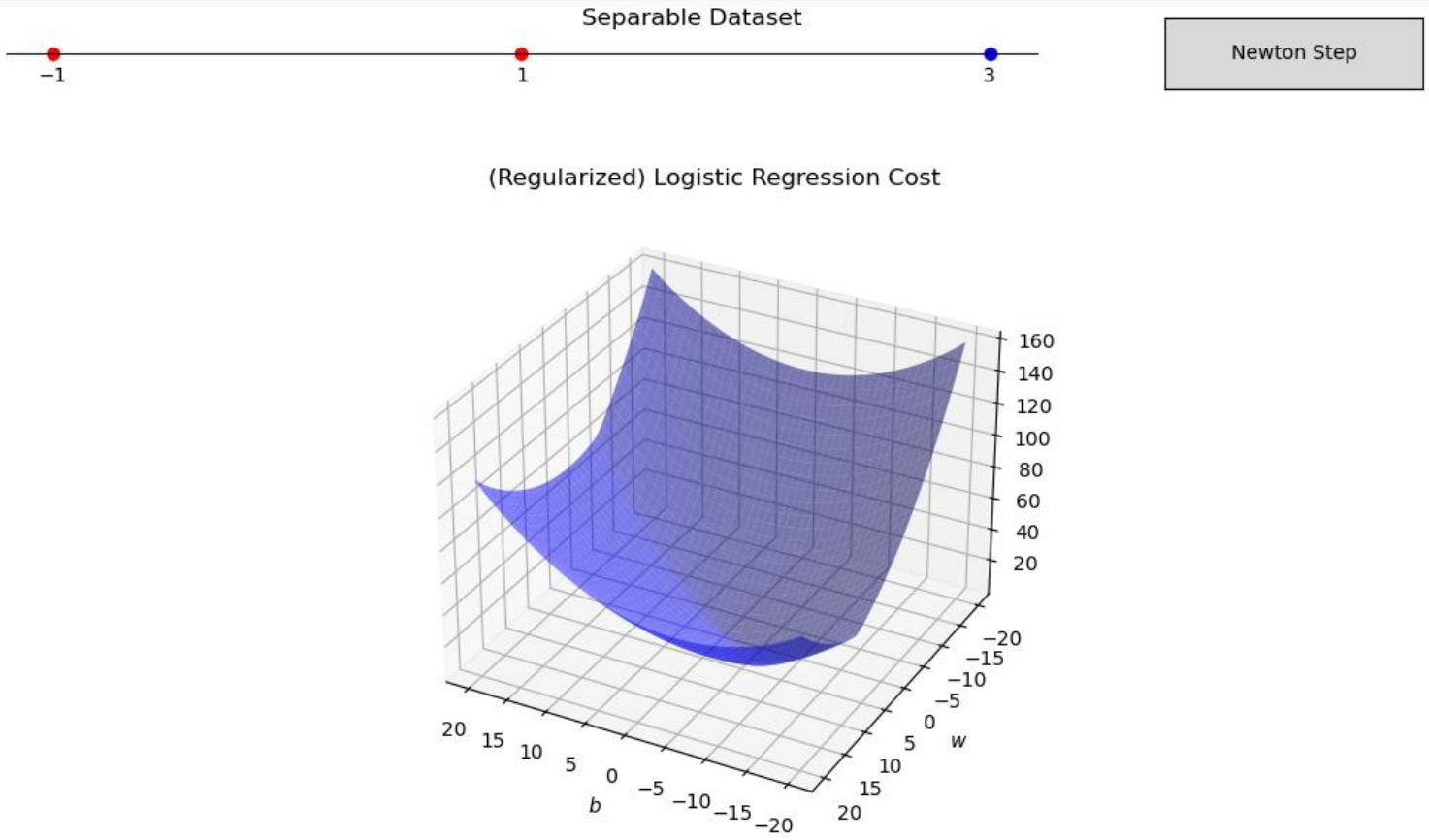
Landscape is many almost linear regions stitched together by quadratic seams. Newton's method blows up in linear regions.



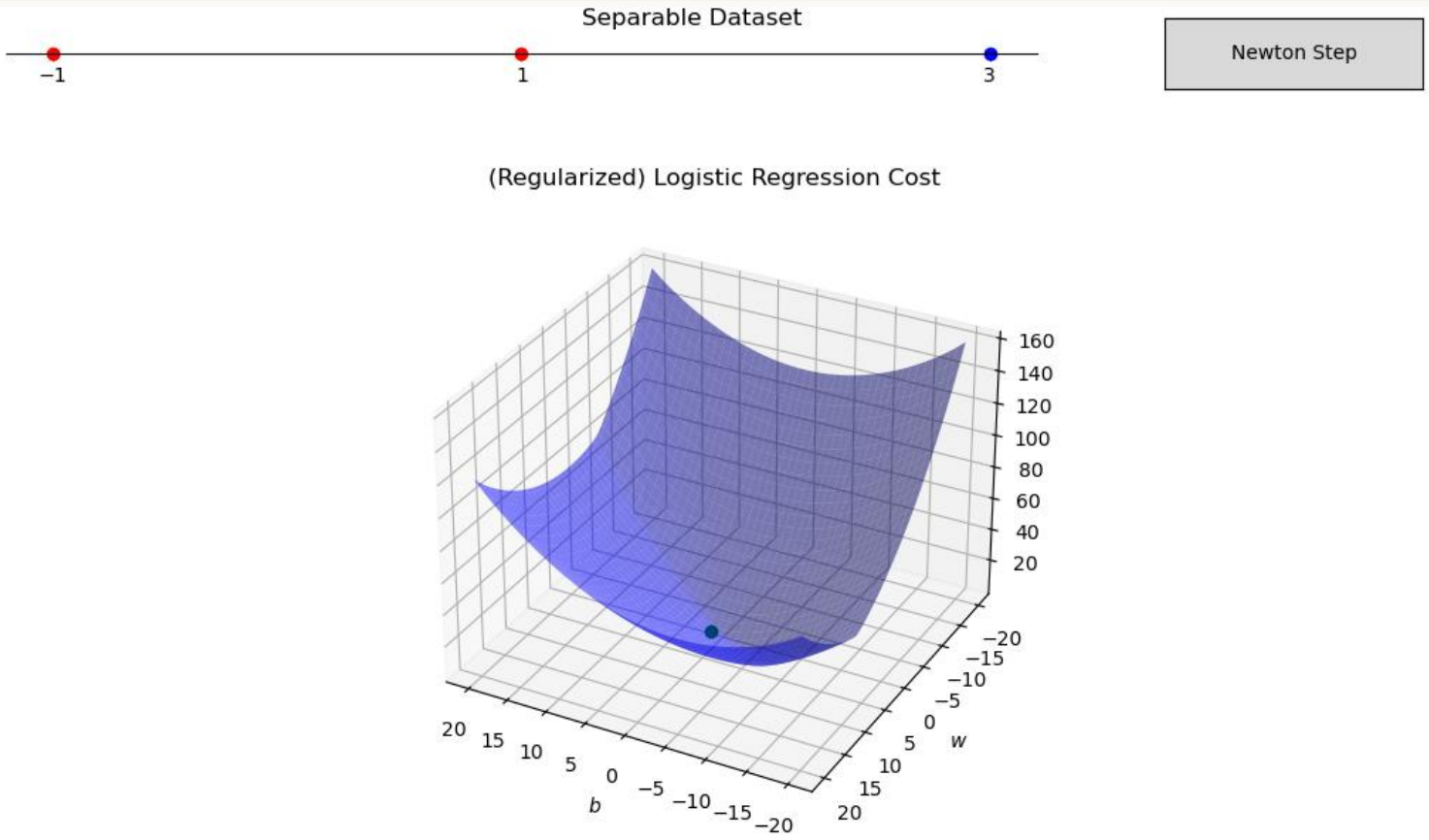
Landscape is many almost linear regions stitched together by quadratic seams. Newton's method blows up in linear regions.



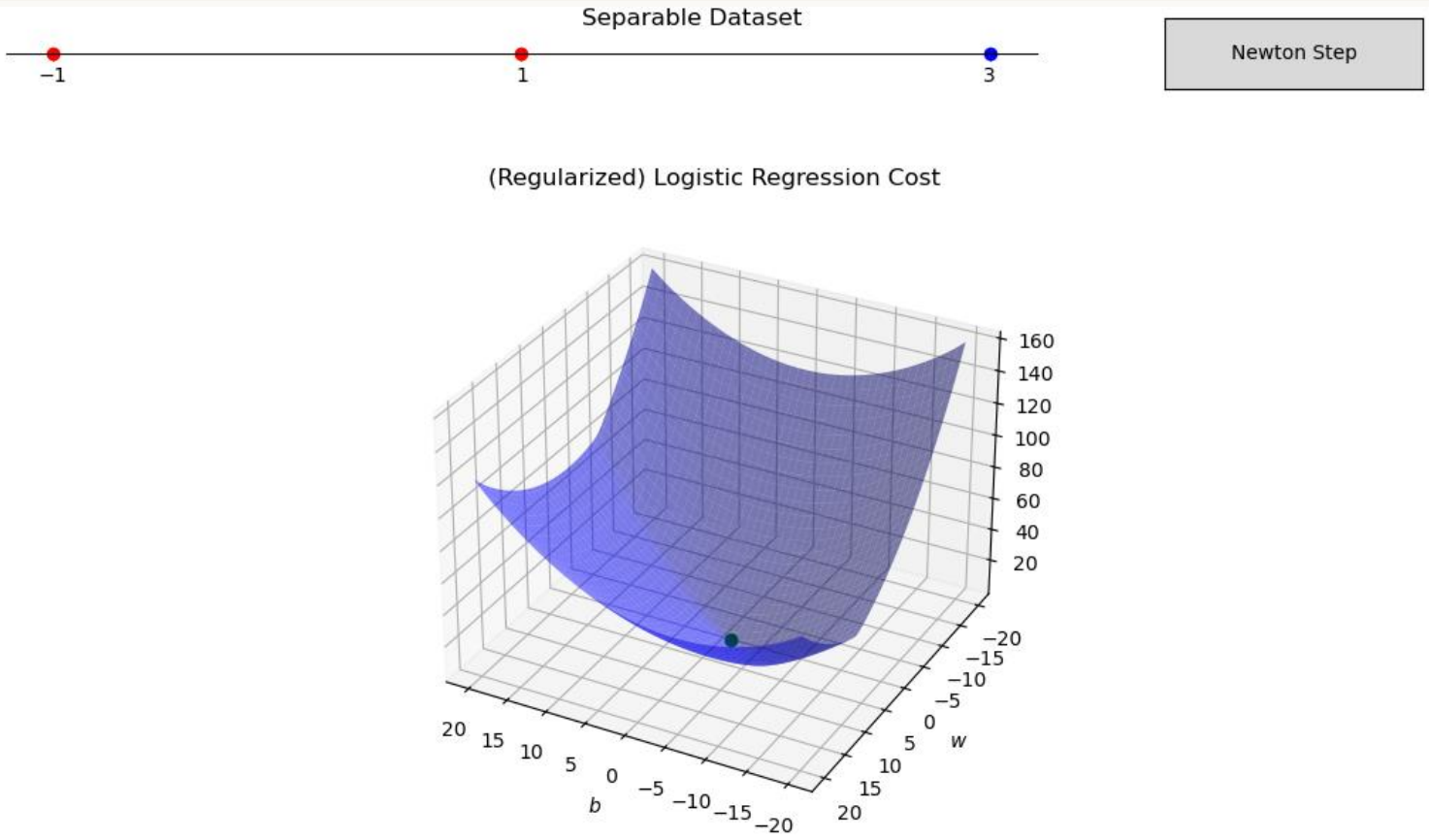
Landscape is many almost linear regions stitched together by quadratic seams. Newton's method blows up in linear regions.



L2 regularization changes those almost linear regions to "shallow" quadratic regions without changing the minimizer too much. Newton's method now works anywhere.



L2 regularization changes those almost linear regions to "shallow" quadratic regions without changing the minimizer too much. Newton's method now works anywhere.



L2 regularization changes those almost linear regions to "shallow" quadratic regions without changing the minimizer too much. Newton's method now works anywhere.

Plugin methods so far

Linear discriminant analysis and logistic regression both

- take a parametric model for the distribution/a posteriori probabilities
- estimate those parameters
- plug these into the formula for the Bayes classifier

Can we take a nonparametric approach?

There are lots of approaches to estimating the distribution of our data…

Curse of dimensionality is a major obstacle…

One more plugin method: Naïve Bayes

The Naïve Bayes classifier is an approach built on estimating the distribution of the data and then plugging this into the Bayes classifier

Makes a (probably ***naïve***) assumption:

Let $X = [X(1), \ldots, X(d)]^T \in \mathbb{R}^d$ denote the random feature vector in a classification problem and Y the corresponding label

The naïve Bayes classifier assumes that, given Y, $X(1), \ldots, X(d)$ are ***independent***

Sometimes known as “Idiot Bayes”

What does the NB assumption buy us?

The major advantage of NB is that we only need to estimate ***scalar/univariate*** densities

Let $f_{X|Y}(\mathbf{x}|y)$ be the probability law (density or mass function) of $X|Y = y, y = 1, \ldots, K$

By the NB assumption

$$f_{X|Y}(\mathbf{x}|y) = \prod_{j=1}^{d} f_{X(j)|Y}(x(j)|y)$$

where $f_{X(j)|Y}(x(j)|y)$ denotes the probability law of $X(j)|Y = y$

Naïve Bayes classifier

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ be training data and set

- $\widehat{\pi}_y = \dfrac{|\{i : y_i = y\}|}{n}$
- $\widehat{f}_{X(j)|Y}(x(j)|y) =$ any estimate of $f_{X(j)|Y}(x(j)|y)$ based on $\{x_i(j) : y_i = y\}$

The NB classifier is then given by

$$\widehat{h}(\mathbf{x}) = \arg\max_y \ \widehat{\pi}_y \prod_{j=1}^{d} \widehat{f}_{X(j)|Y}(x(j)|y)$$

So, how can we determine $\widehat{f}_{X(j)|Y}(x(j)|y)$?

Continuous features

Suppose that $X(j)|Y = y$ is a continuous random variable

Options for estimating $\widehat{f}_{X(j)|Y}(x(j)|y)$ include

- parametric estimate (e.g., Gaussian MLE)
- kernel density estimate (more on this later!)
- quantize to a discrete random variable

Discrete features

Now suppose that $X(j)|Y = y$ takes only the values $z_1, \ldots, z_L$

Define $n_y = |\{i : y_i = y\}|$

$$n_y(j, z_\ell) = |\{i : y_i = y \text{ and } x_i(j) = z_\ell\}|$$

for $y = 0, \ldots, K-1$

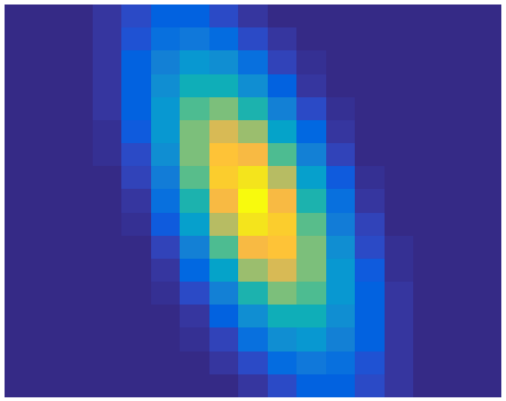
Then a natural (maximum likelihood) estimate of $\mathbb{P}[X(j) = z_\ell | Y = y]$ is

$$\widehat{f}_{X(j)|Y}(z_\ell|y) = \frac{n_y(j, z_\ell)}{n_y}$$

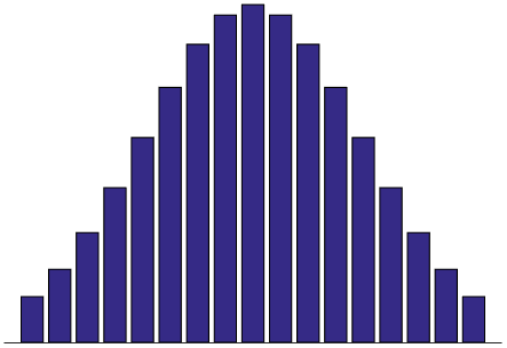
(AKA: a ***histogram***!)

Example

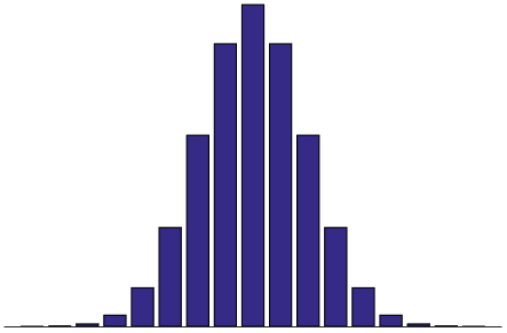
$f_{X|Y}(\mathbf{x}|y)$



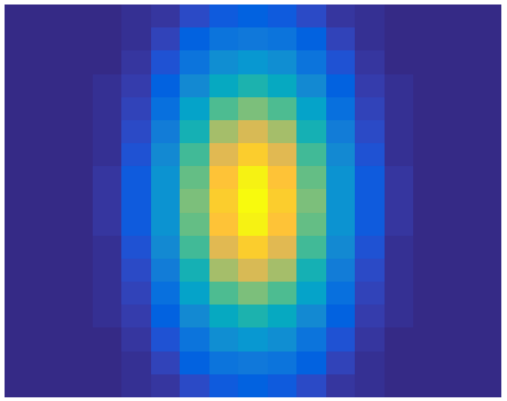
$\widehat{f}_{X(2)|Y}(x(2)|y)$



$\widehat{f}_{X(1)|Y}(x(1)|y)$



$f_{X|Y}(\mathbf{x}|y)$



Example: Document classification

Suppose we wish to classify documents into categories

- 0 = politics
- 1 = sports
- 2 = finance
- …

One simple (but useful) way to represent a document is as $\mathbf{x} = [x(1), \ldots, x(d)]^T$ with d representing the number of words in our "vocabulary" and $x(j)$ representing the number of times word j occurs in the document

"bag of words" model

Multinomial Naïve Bayes

We can think of each document as consisting of n words which are distributed among the d choices in the vocabulary independently at random

- ***multinomial distribution***

All that is required in this model is to estimate the probability of each word under the different categories:

- For each class y, compute the number of times each word appears (across all documents); denote this $n_y(\text{word})$
- Let $n_y^{\text{total}} = \sum_{j=1}^{d} n_y(\text{word}_j)$ denote the total number of words appearing in all documents labelled class y
- Compute the estimate

$$\mathbb{P}(\text{word}_j|y) = \frac{n_y(\text{word}_j)}{n_y^{\text{total}}}$$

Multinomial Naïve Bayes

With this estimate, for a new document summarized by the bag-of-words model $\mathbf{x} = [x(1), \ldots, x(d)]^T$, we can compute

$$\mathbb{P}(\mathbf{x}|y) = \frac{(\sum_{j=1}^{d} x(j))!}{x(1)! \cdots x(d)!} \prod_{j=1}^{d} \mathbb{P}(x(j)|y)$$

$$= \frac{(\sum_{j=1}^{d} x(j))!}{x(1)! \cdots x(d)!} \prod_{j=1}^{d} \mathbb{P}(\text{word}_j|y)^{x(j)}$$

This gives us the classifier

$$\widehat{h}(\mathbf{x}) = \arg\max_{y} \ \widehat{\pi}_y \prod_{j=1}^{d} \mathbb{P}(\text{word}_j|y)^{x(j)}$$

Undesirable property

It is possible that after training our classifier, we may be given an input $\mathbf{x}$ where some $x(j) \neq 0$ for some word that was not observed in the training data for some class y

For such a class, our previous estimate would yield $\mathbb{P}(\text{word}_j|y) = 0$, and hence

$$\mathbb{P}(\mathbf{x}|y) \propto \prod_{j=1}^{d} \mathbb{P}(\text{word}_j|y)^{x(j)} = 0$$

In this case, class y will never be predicted

This is rather unfortunate…

Laplace smoothing

It could happen that every training document in “sports” contains the word “ball”

What happens when we get an article about hockey?

To avoid this problem, it is common to instead use

$$\mathbb{P}(\text{word}|y) = \frac{n_y(\text{word}) + 1}{\sum_{j=1}^{d} n_y(\text{word}_j) + d}$$

We can think of this as the result of adding d “pseudo-samples” to our data to ensure that each word occurs at least once

Related to Laplace’s rule of succession: ***Lapalce smoothing*** (Bayesian estimate with a Dirichlet prior)

Other variants

- Bernoulli Naïve Bayes
  - used when the features are binary-valued
  - example: word occurrence vectors (vs word count vectors)
  - simply need to estimate probability of 1 vs 0 for each feature
- Gaussian Naïve Bayes
  - models continuous features as univariate Gaussian densities
  - estimates mean and variance of data to fit a Gaussian to each feature
- Many other extensions
  - Gaussian mixture models
  - kernel density estimates
  - …

Comparison of plugin methods

LDA, logistic regression, and naïve Bayes, are all ***plugin methods*** that result in ***linear*** classifiers

Linear discriminant analysis

- better if Gaussianity assumptions are valid

Logistic regression

- models only the distribution of $Y|X$, not (X, Y)
- valid for a larger class of distributions
- fewer parameters to estimate

Naïve Bayes

- plugin method based on density estimation
- scales well to high-dimensions and naturally handles mixture of discrete and continuous features

Beyond plugin methods

Plugin methods can be useful in practice, but they are also somewhat limited

- There are always distributions where our assumptions are violated
- If our assumptions are wrong, the output is totally unpredictable
- Can be hard to verify whether our assumptions are right
- Require solving a more difficult problem as an intermediate step

For most of the remainder of this course will focus on (nonparametric) methods that avoid making such strong assumptions about the (unknown) process generating the data