Classification

- **Supervised learning** with categorical/qualitative outcomes

(in contrast to regression, with numeric outcomes)

- Often called "labels", K = number of unique classes
- Binary: positive/negative or 0/1 or yes/no or success/fail etc

Label names not mathematically important - e.g. use $1, \ldots, K$

- Limitations: labels already defined (not learned from data--that would be unsupervised learning), K is fixed
- Plots: often use color/point shape for categorical variables

Interpretable classification

Logistic regression

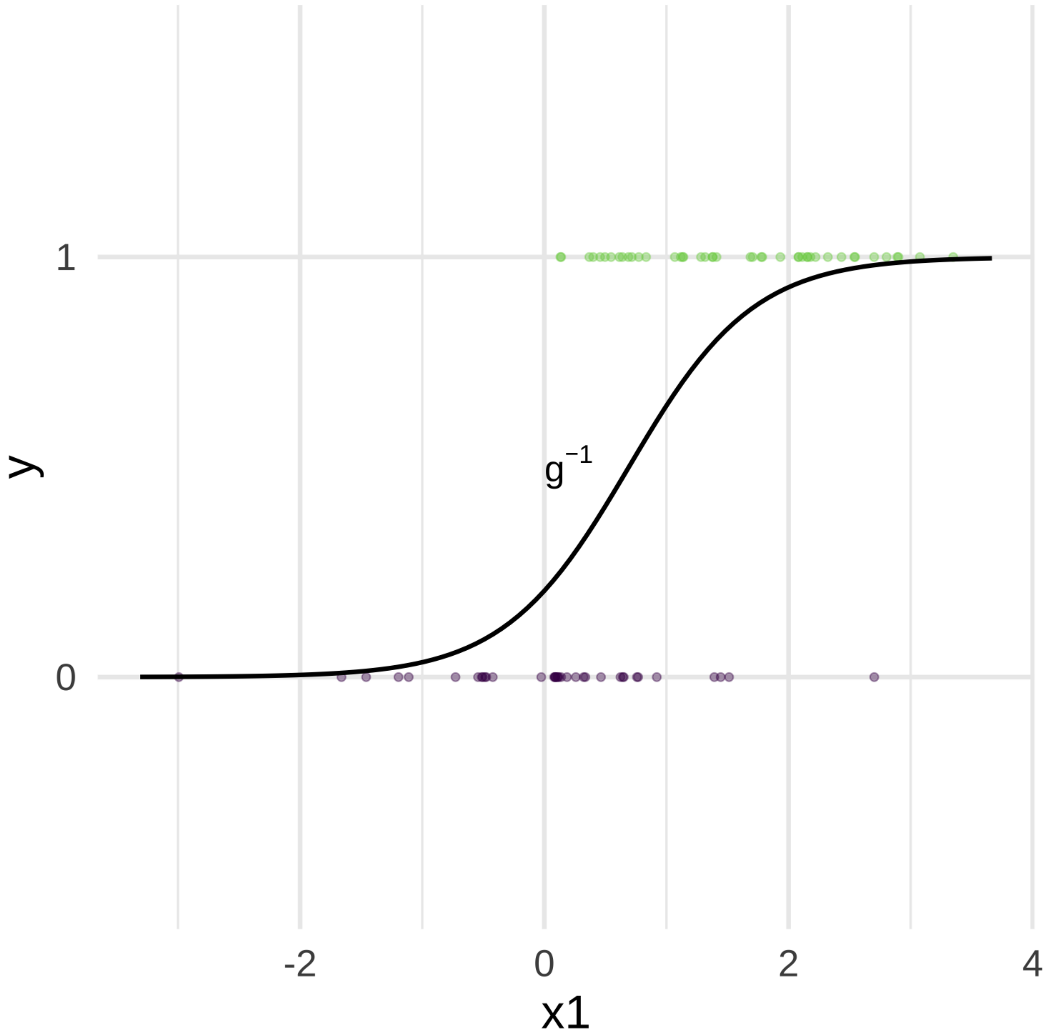
$$\mathbb{E}(Y|\mathbf{X} = \mathbf{x}) = g^{-1}(\mathbf{x}^T\beta)$$

$$g(p) = \log\left(\frac{p}{1-p}\right)$$

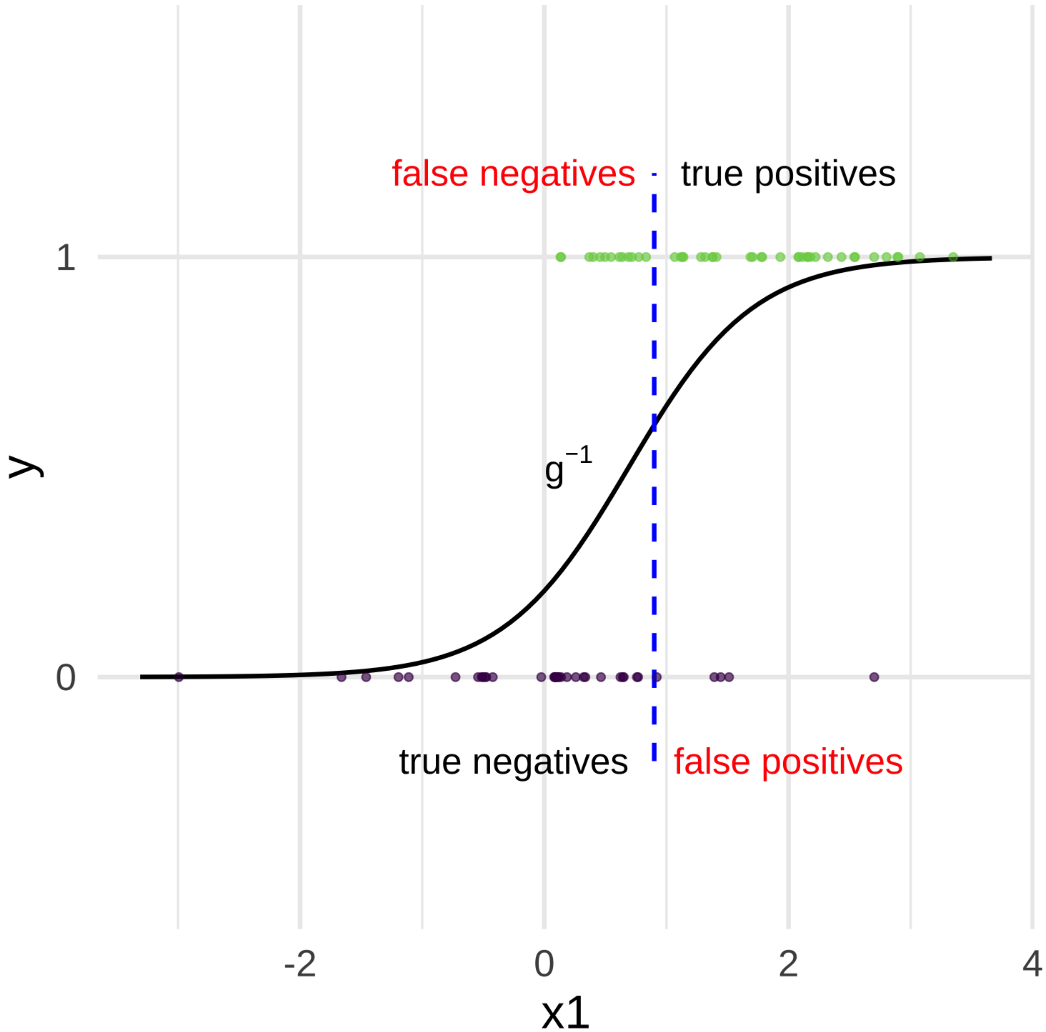
Generalized linear models (GLMs)

- Various "link" functions g
- Linear regression is a special case with $g = \mathrm{id}$
- Logistic in `R`: `glm(..., family = binomial())`
- Others: Poisson, multinomial, ..., see `?family` in `R`

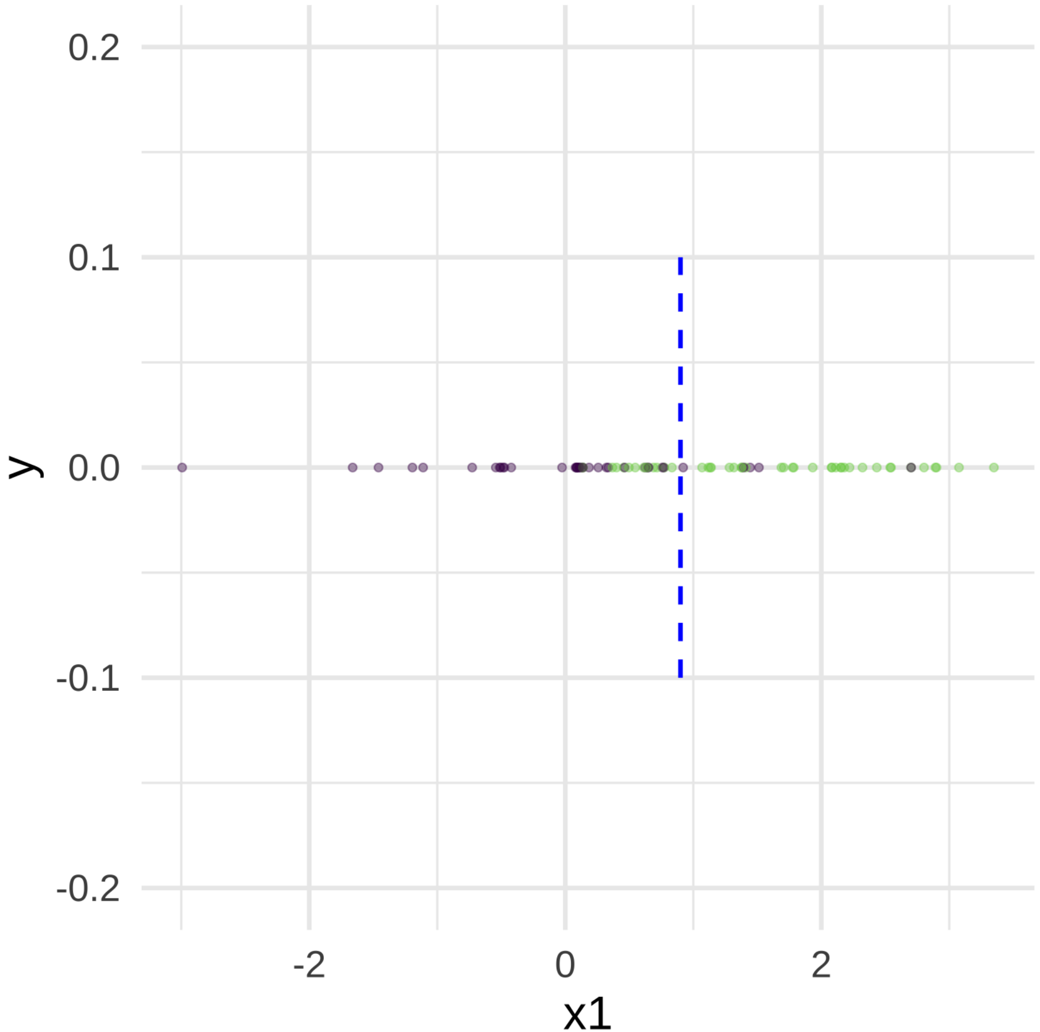
One predictor, "S curve"



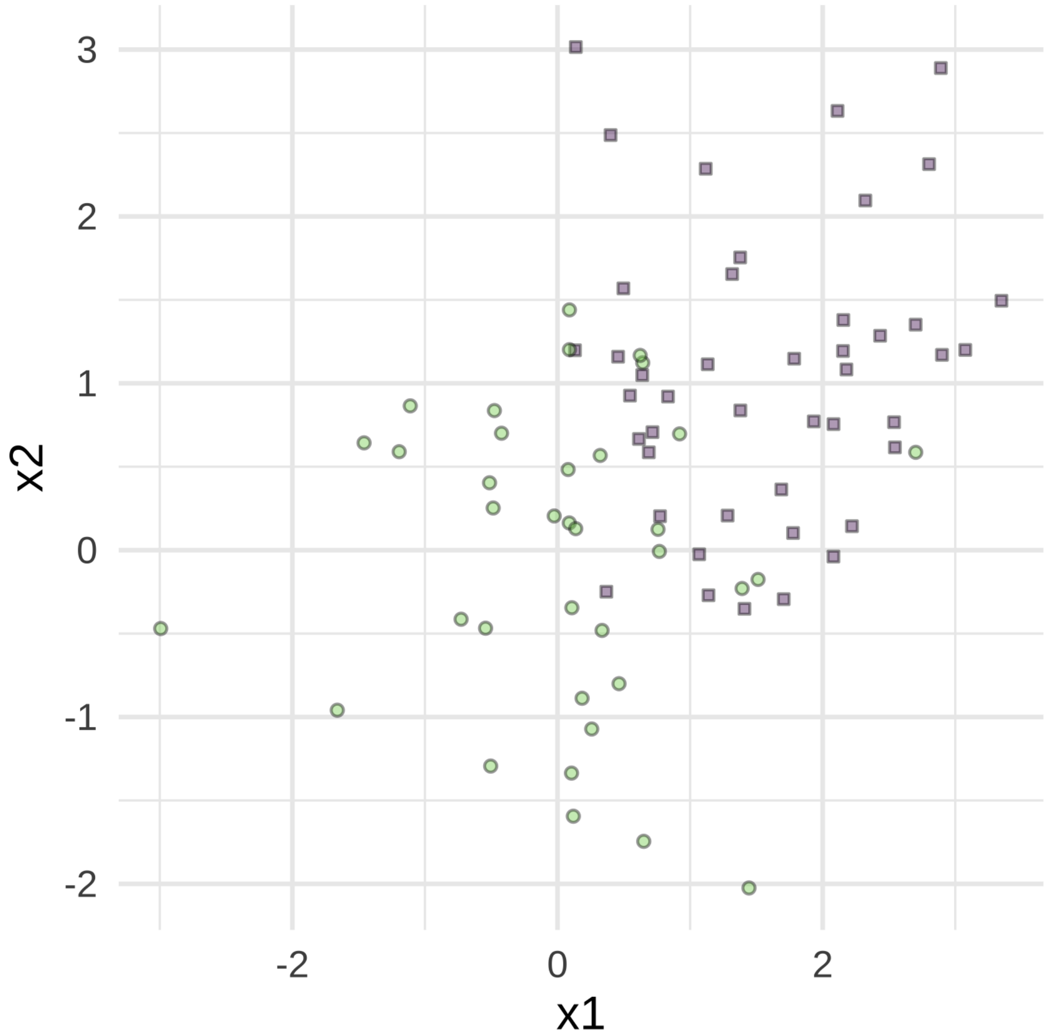
Classifications/decisions: threshold probability



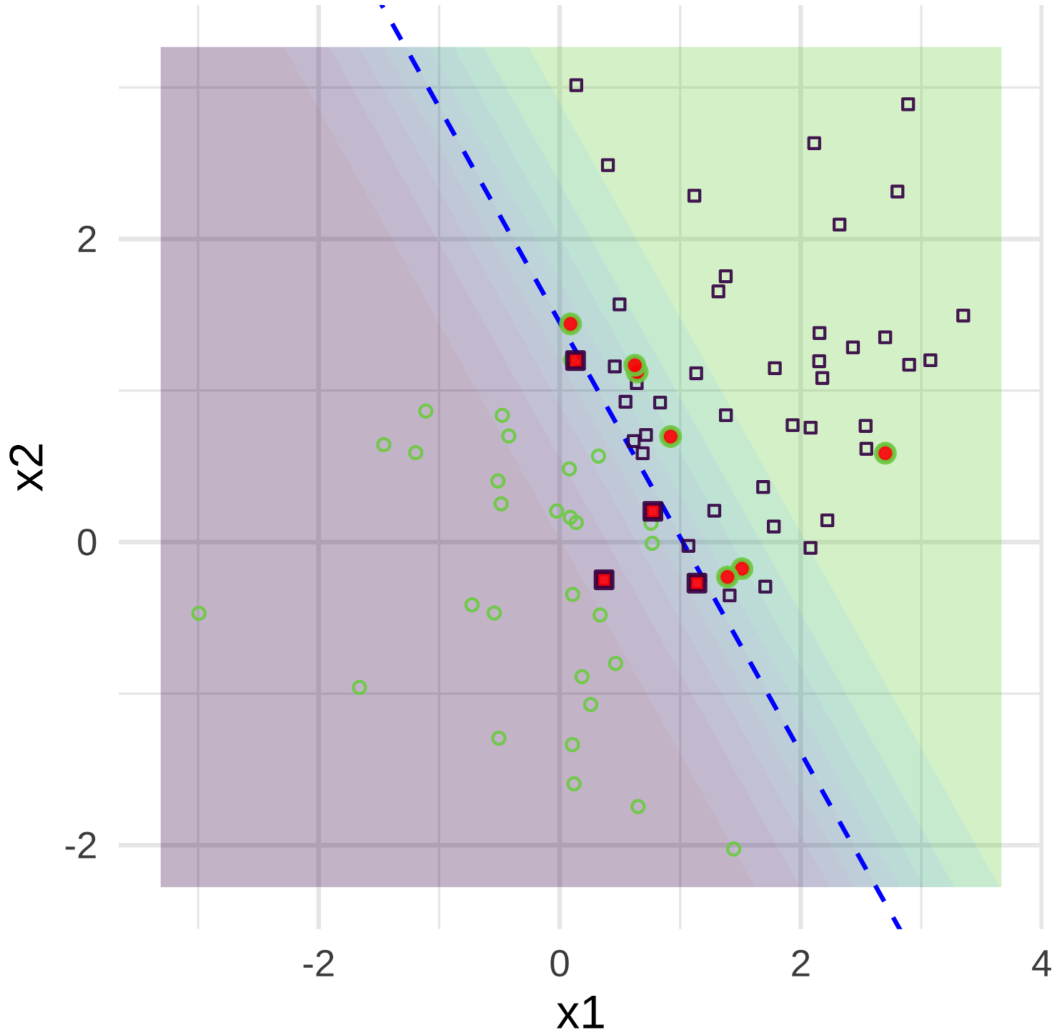
Without giving y a spatial dimension



Two predictors, binary outcome



Contours of GLM-predicted class probabilities



Classification boundaries with

$p = 3$ predictors

Boundary = plane

$p > 3$ predictors

Boundary = hyperplane

(In practice, "high-dimensional" = can't easily plot it)

Interpretation: coefficients

```
model_fit <- glm(y ~ x1 + x2, family = "binomial", data = train)
broom::tidy(model_fit)
```

```
## # A tibble: 3 × 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    -2.24     0.656     -3.42 0.000635
## 2 x1              2.17     0.584      3.72 0.000198
## 3 x2              1.53     0.499      3.07 0.00215
```

Coefficient scale: log-odds? Exponentiate $\longrightarrow$ odds

```
broom::tidy(model_fit, exponentiate = TRUE)
```

```
## # A tibble: 3 × 5
##   term        estimate std.error statistic  p.value
##   <chr>          <dbl>     <dbl>     <dbl>    <dbl>
## 1 (Intercept)    0.106     0.656     -3.42 0.000635
## 2 x1             8.78      0.584      3.72 0.000198
## 3 x2             4.62      0.499      3.07 0.00215
```

Interpretation: inference and diagnostics

- MLEs $\longrightarrow$ asymptotic normality for intervals/tests

`summary()`, `coef()`, `confint()`, `anova()`, etc in `R`

- "Deviance" instead of RSS

```
broom::glance(model_fit)
```

```
## # A tibble: 1 × 8
##   null.deviance df.null logLik   AIC   BIC deviance df.residual  nobs
##           <dbl>   <int>  <dbl> <dbl> <dbl>    <dbl>       <int> <int>
## 1          110.      79  -25.8  57.5  64.7     51.5          77    80
```

- Because y is 0 or 1, residual plots will show patterns, not as easy to interpret geometrically

Challenges

Separable case (guaranteed if $p > n$)

If classes can be perfectly separated, the MLE is undefined, fitting algorithm diverges as $\hat{\beta}$ coordinates $\to \pm\infty$

Awkwardly, classification is *too easy*(!?) for this probabilistic approach

Curse of dimensionality

Biased MLE and wrong variance/asymp. dist. if $n/p \to \text{const}$, even if > 1

See Sur and Candès, (2019)

Classification summary

- Numeric prediction $\to$ classification

$$\hat{y} = \mathbb{I}(\hat{p} > c) = \begin{cases} 0 & \text{if } \hat{y} \leq c \\ 1 & \text{if } \hat{y} > c \end{cases}$$

Log-odds function is monotonic, so (hyperplanes)

$$\hat{p} > c \leftrightarrow x^T\beta > c'$$

- More classes: transform to binary, predict using largest $\hat{p}_k$
- Non-linear boundaries: transformation of predictors, or use methods other than GLMs (we'll learn more soon)
- Some classification methods output categorical classes, not probabilities (or other numeric scores)

Fitting logistic regression

How do we estimate β? **Maximum likelihood**:

$$\text{maximize } L(\beta; \mathbf{y}|\mathbf{X}) = \prod_{i=1}^{n} L(\beta; y_i|\mathbf{x}_i)$$

(assuming the data is i.i.d.)

Next slide: a bit of mathematics

MLE

$$L(\beta;\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{n}\left(\frac{1}{1+e^{-x_i\beta}}\right)^{y_i}\left(1-\frac{1}{1+e^{-x_i\beta}}\right)^{1-y_i}$$

$$\ell(\beta;\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n} y_i \log\left(\frac{1}{1+e^{-x_i\beta}}\right) + (1-y_i)\log\left(1-\frac{1}{1+e^{-x_i\beta}}\right)$$

$$\frac{\partial}{\partial\beta}\ell(\beta;\mathbf{y}|\mathbf{x}) = \sum_{i=1}^{n} y_i\left(\frac{x_i e^{-x_i\beta}}{1+e^{-x_i\beta}}\right) + (1-y_i)\left(\frac{-x_i}{1+e^{-x_i\beta}}\right)$$

$$= \sum_{i=1}^{n} x_i\left[y_i - \left(\frac{1}{1+e^{-x_i\beta}}\right)\right] = \sum_{i=1}^{n} x_i[y_i - \hat{p}_i(\beta)]$$

Set this equal to 0 and solve for β using Newton-Raphson

Newton-Raphson

- Find the roots of a function
- Iteratively approximating the function by its tangent
- Root of the tangent line is used as starting point for next approximation
- See the animation on Wikipedia

Exercise: using result from previous slide, compute the second derivative of ℓ and derive the expressions needed to apply Newton-Raphson

Logistic regression fitting: multivariate case

Newton-IRLS (equivalent) steps:

$$\begin{aligned}
\hat{\mathbf{p}}_t &= g^{-1}(\mathbf{X}\hat{\beta}_t) && \text{update probs.} \\
\mathbf{W}_t &= \operatorname{diag}[\hat{\mathbf{p}}_t(1 - \hat{\mathbf{p}}_t)] && \text{update weights} \\
\hat{\mathbf{y}}_t &= g(\hat{\mathbf{p}}_t) + \mathbf{W}_t^{-1}(\mathbf{y} - \hat{\mathbf{p}}_t) && \text{update response}
\end{aligned}$$

and then update parameter estimate (LS sub-problem)

$$\hat{\beta}_{t+1} = \arg\min_{\beta} (\hat{\mathbf{y}}_t - \mathbf{X}\beta)^T \mathbf{W}_t (\hat{\mathbf{y}}_t - \mathbf{X}\beta)$$

Note: larger weights on observations with $\hat{p}$ closer to 1/2, i.e. the most difficult to classify (***look for variations of this theme***)

See Section 4.4.1 of ESL

Optimization algorithms

Downside of Newton-Raphson: requires second derivatives, including *inverting the $p \times p$ Hessian matrix* when optimizing over $p > 1$ parameters

If p is large, **second-order** optimization methods like Newton's are very costly

First order methods only require computing the $p \times 1$ gradient vector

Recall that the gradient is a vector in the *direction of steepest increase* in the parameter space

Gradient (steepest) descent

i.e. skiing as fast as possible. Notation, let

$$L(\beta) = L(\mathbf{X}, \mathbf{y}, g_\beta) \text{ (loss function)}$$

1. Start at an initial point $\beta^{(0)}$
2. For step $n = 1, \ldots$
   - Compute $\mathbf{d}_n = \nabla L(\beta^{(n-1)})$ (gradient)
   - Update $\beta^{(n)} = \beta^{(n-1)} - \gamma_n \mathbf{d}_n$
3. Until some **convergence criteria** is satisfied

Where the **step size** $\gamma_n > 0$ is made small enough to not "overshoot" and increase the loss, i.e. the loss only decreases

Optimization more generally

- Components: objective functions, algorithms, local/global optima, approximate solutions
- Computational cost: speed, storage (time and space)

Closed form / analytic solutions

e.g. OLS formula for $\hat{\beta}$ (remember?)

Iterative algorithms (e.g. Newton-Raphon)

- Rates of convergence
- Might have guarantees, e.g. if objective is **convex**

Machine learning = optimization algorithms applied to data

Understanding optimization is very important!

- Intuition (challenge: dimensionality)
- Mathematical guarantees (challenge: relevance)
- Empirical evaluation (challenge: overfitting...)