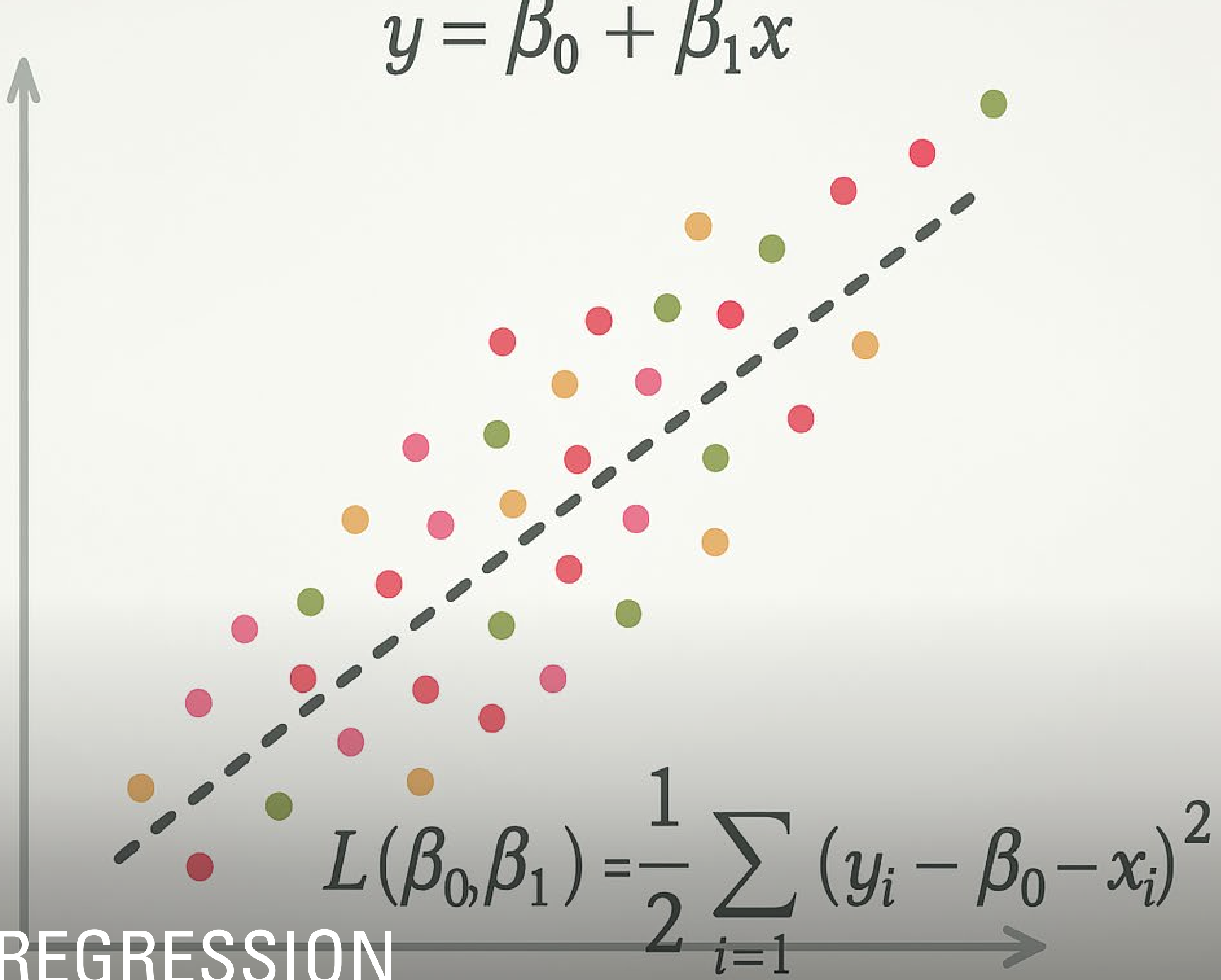


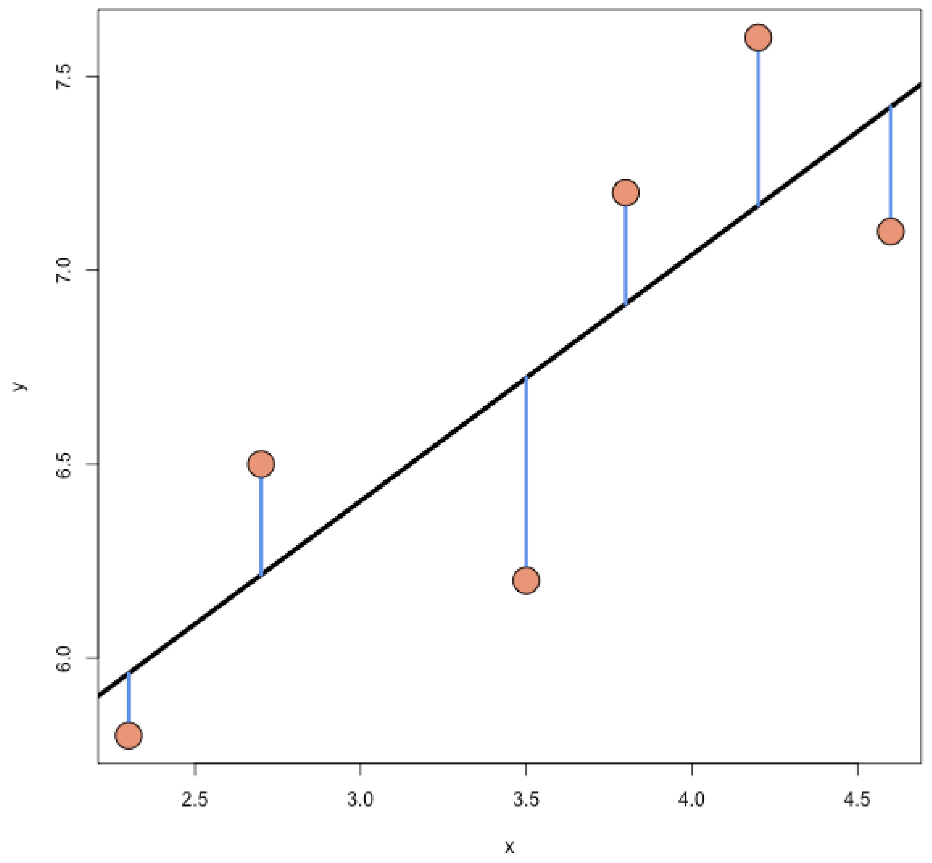
INTRO TO REGRESSION

Linear Regression

LINEAR REGRESSION

- Linear regression is a **supervised** learning approach for predicting a **quantitative** response
- Linear regression has been around for a long time
  - Still very widely used
  - Best method for truly linear relationships
  - Many "fancier" methods are based on linear regression

LEAST SQUARES



Least squares line is the line that minimizes

$$\frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Training a Linear Regression Model

GENERAL MODEL-BASED TRAINING

- Define an appropriate **objective function**
  - Cost/Loss functions are minimized (machine learning)
  - Utility/Fitness functions are maximized (statistics)
    - For example, the likelihood in MLE is a utility function
- "Train" the model by solving the optimization problem using an appropriate optimization **method**
  - a "closed-form" solution (rare)
  - a numerical method (usually)
    - **Gradient descent or variants (**Adam, Newton's method)

OPTIMIZING THE OBJECTIVE FUNCTION

- Minimum vs. Minimizer
- Given a function, *f*, how do we minimize the function?
- What if there is no analytic solution? (I.e., no closed form?)

- A minimizer is the value *x* that produces the smallest output *f(x)*.
- Differential calculus!
- Numerical approaches.

ORDINARY LEAST SQUARES OBJECTIVE

- What does it mean to "fit" a line to data?
- How do we measure goodness?
- Why use $(y - \hat{y})^2$and not $|y - \hat{y}|$?
- The objective function:

- We want $\hat{y} = X\beta$ to be a *good* approximation.
- We want $y - \hat{y}$ to be *small.*
- Differentiability/punishing larger errors.

$$J(\beta) = \frac{1}{2n}(y - X\beta)^T(y - X\beta)$$

METHOD 1: NORMAL EQUATIONS

- Matrix notation cost function:

$$J(\beta) = \frac{1}{2n}(y - X\beta)^T(y - X\beta)$$

- Derivative of Cost Function:

$$\nabla J(\beta) = \frac{1}{n}X^T(X\beta - y)$$

- Set equal to 0 an rearrange (normal equations):

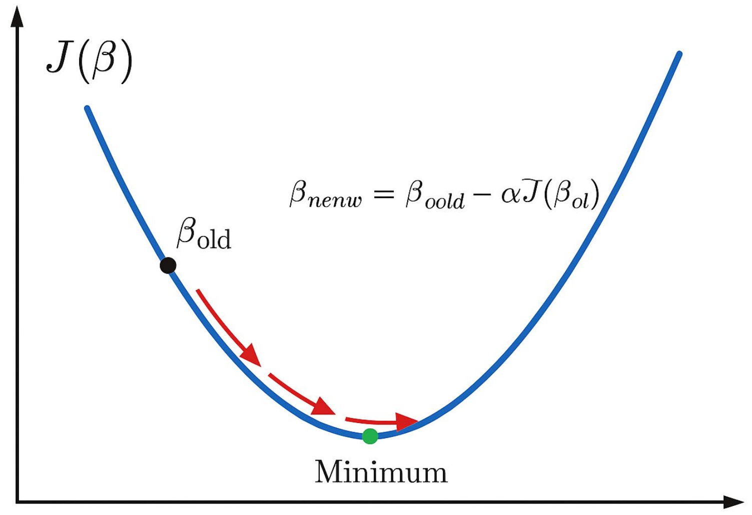
$$(X^TX)\hat{\beta} = X^Ty$$

- Solution for $\hat{\beta}$:

$$\hat{\beta} = (X^TX)^{-1}X^Ty$$

METHOD 2: GRADIENT DESCENT

- Numerical optimization algorithm that tweaks parameters iteratively until the minimum is found
  - “Descend” the gradient of the cost function in the steepest direction until the minimum is reached
- What is the gradient?
  - The vector that points in the direction of the greatest rate of increase for a function
  - It is the vector of partial derivatives


$J(\beta)$
$\beta_{nenw} = \beta_{oold} - \alpha\mathcal{J}(\beta_{ol})$
β_{old}
Minimum

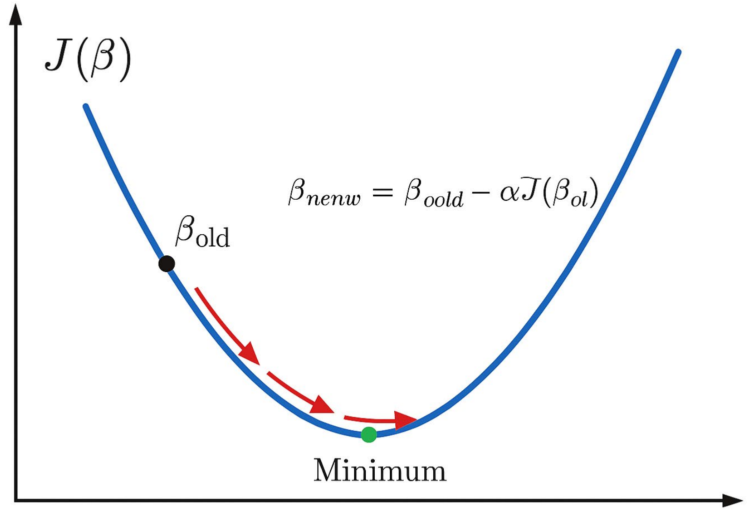
METHOD 2: GRADIENT DESCENT

- **Batch Gradient Descent Algorithm:**
  - Choose initial values for the $\hat{\beta}$ vector (that is, choose a value for each $\hat{\beta}_j$)
  - Adjust $\hat{\beta}$ by moving in the opposite direction of the gradient:

$$\hat{\beta}_{new} = \hat{\beta}_{old} - \alpha \nabla J(\hat{\beta}_{old})$$

    - Where α is the *learning rate*, a hyperparameter, that controls the step size
  - Repeat previous step until convergence is reached

- The term “Batch” refers to the fact that all the training data is used



$J(\beta)$
$\beta_{nenw} = \beta_{oold} - \alpha \mathcal{J}(\beta_{ol})$
β_{old}
Minimum

LEARNING RATE

- If the learning rate is too high, the gradient descent algorithm will bounce around the minimum and never converge
    - (See Figure 4-5 in HOML)
- If the learning rate is too low, it could take a very long time to find the minimum
    - (See Figure 4-4 in HOML)
- It is common to start with a small learning rate (~0.01) and adjust based on the performance of the model

GRADIENT DESCENT VARIATIONS

- Stochastic Gradient Descent (SGD)
  - How it works:
    - Chooses one random instance at each step.
    - Computes the gradient based on that single instance.
  - **Pros**:
    - Faster per update (small computation).
    - Randomness helps escape local minima on complex cost surfaces.
  - **Cons**:
    - Updates are noisy; the algorithm never fully "settles."
    - True minimum is rarely reached—oscillates around it.

GRADIENT DESCENT VARIATIONS

- Mini-Batch Gradient Descent
  - How it works:
    - Chooses a small random subset (mini-batch) of training data at each step.
    - Computes the gradient based on that subset.
  - **Pros**:
    - More stable than SGD (less noisy updates).
    - Faster than full batch gradient descent for large datasets.
    - Leverages vectorized operations for efficiency.
  - **Cons**:
    - Still introduces some randomness, so convergence is not perfectly smooth.
    - Requires tuning batch size (too small = noisy, too large = slow).

GRADIENT DESCENT VARIATIONS

- Adam Optimizer
- How it works:
  - Maintains two moving averages:
  - First moment (m): average of gradients (like momentum).
  - Second moment (v): average of squared gradients
  - Applies bias correction to both moments.
  - Update rule:

$$\beta_{new} = \beta_{old} - \alpha \frac{\hat{m}}{\sqrt{\hat{v}} + \epsilon}$$

LEARNING SCHEDULE

- A way to help “settle” stochastic gradient descent
- Gradually reduce the learning rate according to a function called the “learning schedule”
- Additional Points:
  - Common schedules include:
  - Step decay: Reduce learning rate by a factor after a fixed number of epochs.
  - Exponential decay: Multiply learning rate by a constant factor each epoch.
  - Polynomial decay: Decrease learning rate following a polynomial function.
- Modern techniques:
  - Cosine Annealing: Learning rate follows a cosine curve, often combined with restarts.
  - One-cycle policy: Increase learning rate initially, then decrease sharply for better convergence.
  - Adaptive methods (Adam, RMSProp): Adjust learning rate per parameter automatically based on gradient history.

REGULARIZATION

REGULARIZATION

- Adding constraints to model to "simplify" the model to help prevent overfitting
- **Regularization**: Techniques that constrain a model to reduce overfitting and improve generalization.
  - Explicit Regularization
  - Implicit Regularization

EXPLICIT REGULARIZATION

- Intentionally and directly add a regularization term to the loss function
- Explicitly modify the optimization objective
- Examples (not exhaustive, but these are the most common):
    - L1 Regularization (Lasso)
    - L2 Regularization (Ridge)
    - Elastic Net (combine L1 and L2 regularization)

IMPLICIT REGULARIZATION

- Any regularization method that does not explicitly modify the cost or objective function
- Can be a byproduct of an algorithm, the learning process, or other choices made in training
- Examples (not exhaustive):
  - Early stopping (in gradient-based optimization, stop training before convergence is reached)
  - Dropout (randomly ignoring some parts of a neural network)
  - Pruning (simplifying a grown decision tree)
  - Data augmentation (adding synthetic data to increase the sample size)

Example of Explicit Regularization in Linear Regression

REGULARIZED COST FUNCTIONS

- Least squares cost function: finds the β's that minimize

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

L1 PENALTY: LASSO

- Least squares cost function

$$J(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2$$

Tuning parameter that controls relative impact of two criteria

What happens if $\alpha = 0$ or $\alpha \approx \infty$?

- **Lasso** linear regression cost function

$$J_{lasso}(\beta) = \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2}_{} + \underline{\alpha}\underbrace{\sum_{j=1}^{p}|\beta_j|}_{}, \qquad (\alpha > 0)$$

Tries to fit the data well
(find βs that move the prediction close to the observed data)

Constrains the model
(pushes βs to be close to or exactly equal to 0)

REGULARIZED COST FUNCTIONS

ℓ_1 penalty

- **Lasso** linear regression cost function

$$J_{lasso}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \alpha\sum_{j=1}^{p}|\beta_j|, \quad \text{(where } \alpha > 0\text{)}$$

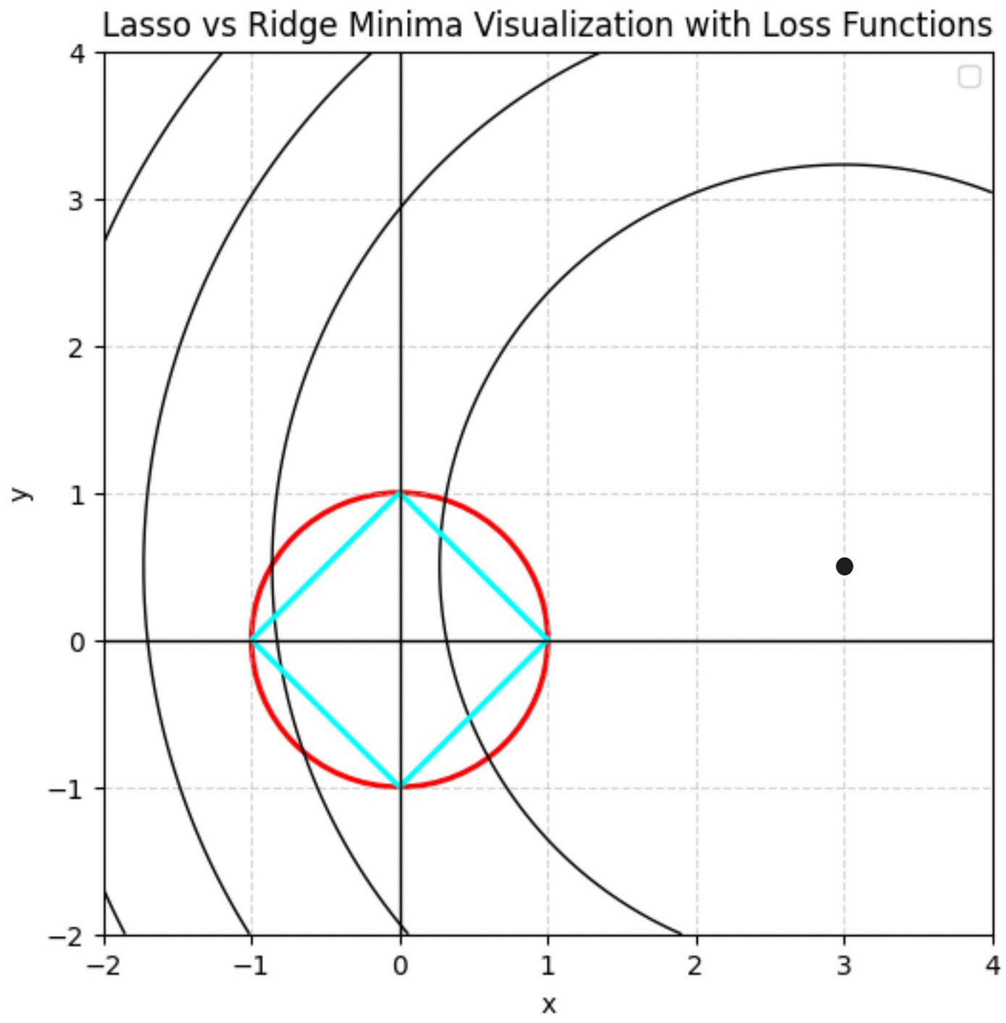
ℓ_2 penalty

- **Ridge** linear regression cost function

$$J_{ridge}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \alpha\underline{\sum_{j=1}^{p}(\beta_j)^2}, \quad \text{(where } \alpha > 0\text{)}$$

Constrains the model
(pushes βs to be close to but NOT exactly equal to 0)

LASSO V. RIDGE VISUALIZATION



Let's explore this further in **Desmos**: https://www.desmos.com/3d

https://www.vizuaranewsletter.com/p/l1-regularization-is-easy-can-you: This is a great source for an intuitive description!

RIDGE VS LASSO

- Lasso
    - Forces some coefficients to be exactly 0 (advantage)
    - Built in variable selection
    - Coefficients can be very different for correlated features (disadvantage)
- Ridge
    - Coefficients will never be exactly 0 (fact)
    - Tends to estimate similar coefficient for correlated features (advantage)
- Neither method will universally dominate
- Lasso generally performs better when many of the features don't contribute to predictability
- Ridge generally performs better when many of the features contribute to the prediction and when features are correlated

ELASTIC-NET REGULARIZATION

- Combination of Ridge and Lasso
    - The idea is to capitalize on the strengths of both methods

$$J_{en}(\beta) = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \alpha_1 \sum_{j=1}^{p}\left|\beta_j\right| + \alpha_2 \sum_{j=1}^{p}\left(\beta_j\right)^2$$

- If $\alpha_1 = 0$ (and $\alpha_2 \neq 0$), elastic net becomes Ridge
- If $\alpha_2 = 0$ (and $\alpha_1 \neq 0$), elastic net becomes Lasso
- If both are 0, we're back to vanilla least squares

REGULARIZATION GENERALLY

- We’ve just looked at Lasso, Ridge, and Elastic net (explicit regularization) in the context of linear regression
- Regularization is a general set of methodologies, not limited to linear regression
  - Used in many models: logistic regression, neural networks, decision trees, etc.
  - Goal: improve generalization by reducing overfitting
  - Can be applied to weights, complexity, or even the training process

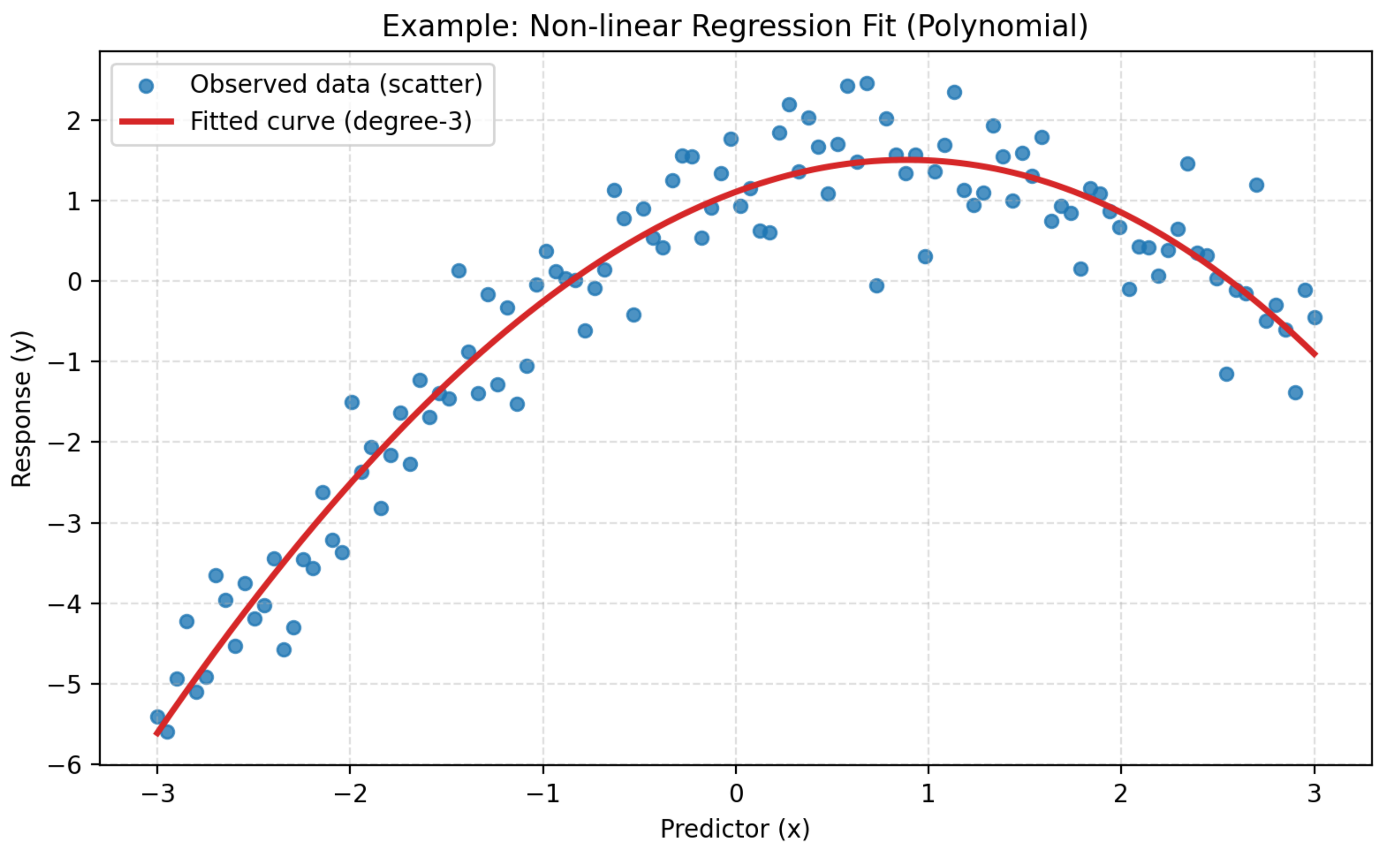
“NONLINEAR” LINEAR REGRESSION

POLYNOMIAL REGRESSION

- Sometimes the relationship between X and y is non-linear
- We can use linear regression to model non-linear relationships (between X and y)
  - Add quadratic, cubic, etc. terms to the model
  - This is sometimes called polynomial regression
  - (Now we are moving into the realm of multiple regression, where we have more than one predictor)

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

POLYNOMIAL REGRESSION



$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \epsilon_i$$

TERMINOLOGY

- Linear regression is "linear" with respect to the parameters (the model is a linear combination of the parameters)

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i$$

- True "nonlinear" regression is where the regression model is a nonlinear combination of the parameters, for example:

$$y_i = \frac{\beta_1 x_i}{\beta_0 + x_i} + \epsilon_i$$

MORE TERMINOLOGY

- Just FYI:
  - **Multiple** regression is a regression model with more than one predictor (or feature)
  - **Multivariate** regression is a regression model with more than one response (or target)

INTERACTION & POLYNOMIAL FEATURES

```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures

# Add squared and cubed features for every predictor plus interactions
poly = PolynomialFeatures(degree=3)
X = poly.fit_transform(X)

# Only include up to 3rd-order interactions
inter = PolynomialFeatures(degree=3, interaction_only=True)
X = inter.fit_transform(X)
```

REGRESSION METRICS

EVALUATING A REGRESSION MODEL

- In regression (we are predicting a numeric variable),
  what does a "good" prediction look like?
- Ideally, $\hat{y}$, (the prediction) is close to y (the observed value)

COMMON REGRESSION METRICS

- Lower is better:
  - Mean absolute error (MAE): $\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \widehat{y_i}|$
  - Mean squared error (MSE): $\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2$
  - Root mean squared error (RMSE): $RMSE = \sqrt{MSE}$
- Higher is better:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

REGRESSION METRIC COMPARISONS

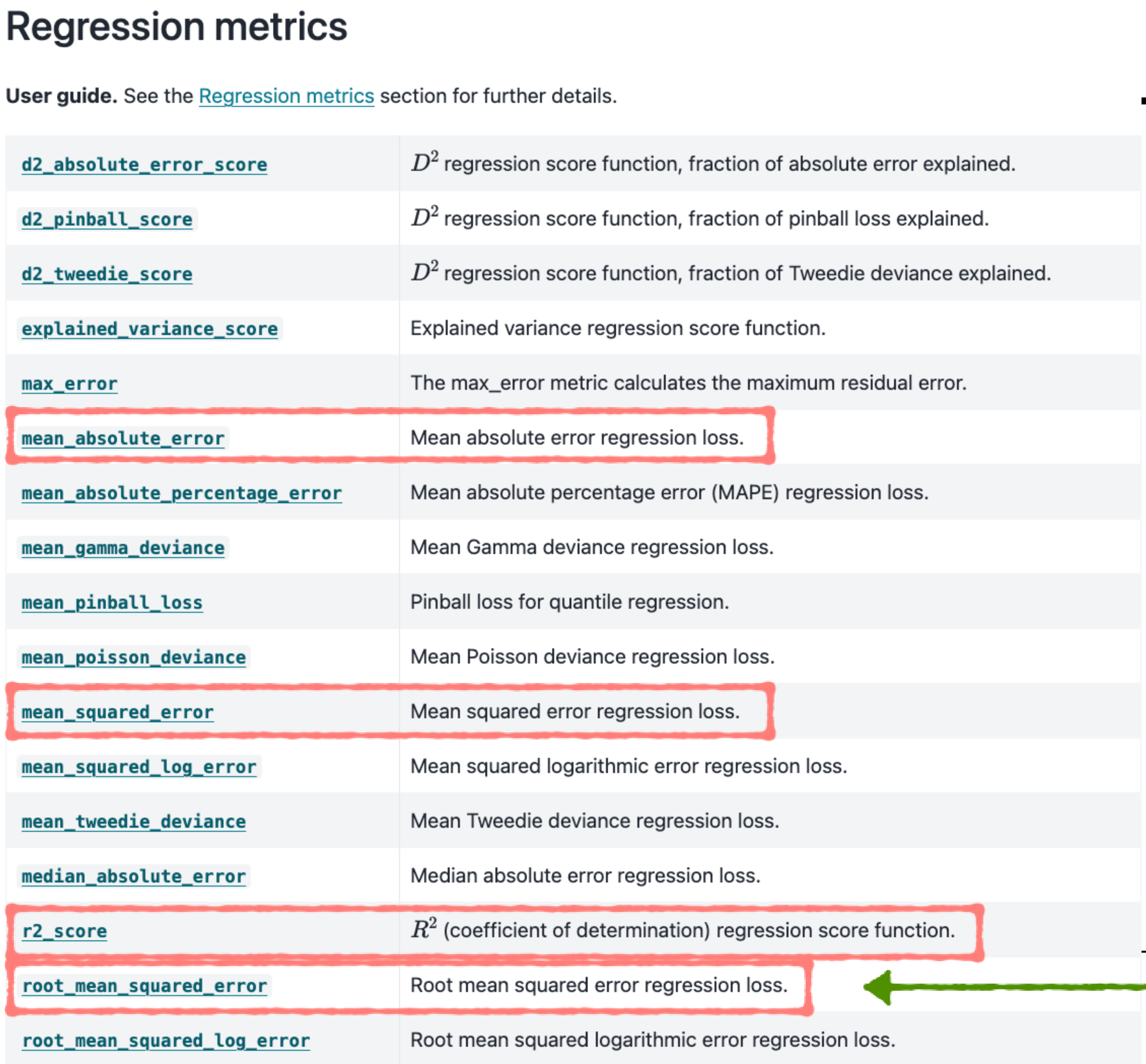
MAE (Mean Absolute Error)
✅ Easy to interpret (average absolute difference)
✅ Less sensitive to outliers than MSE
❌ Does not penalize large errors as strongly

R^2 (Coefficient of Determination)
✅ Measures proportion of variance explained
✅ Easy to compare across models
❌ Can be misleading for non-linear

MSE (Mean Squared Error)
✅ Penalizes large errors more (squared term)
✅ Commonly used in optimization (differentiable)
❌ Sensitive to outliers

RMSE (Root Mean Squared Error)
✅ Same units as target variable
✅ Highlights large errors clearly
❌ Still sensitive to outliers

REGRESSION METRICS IN SCIKIT-LEARN



Regression metrics

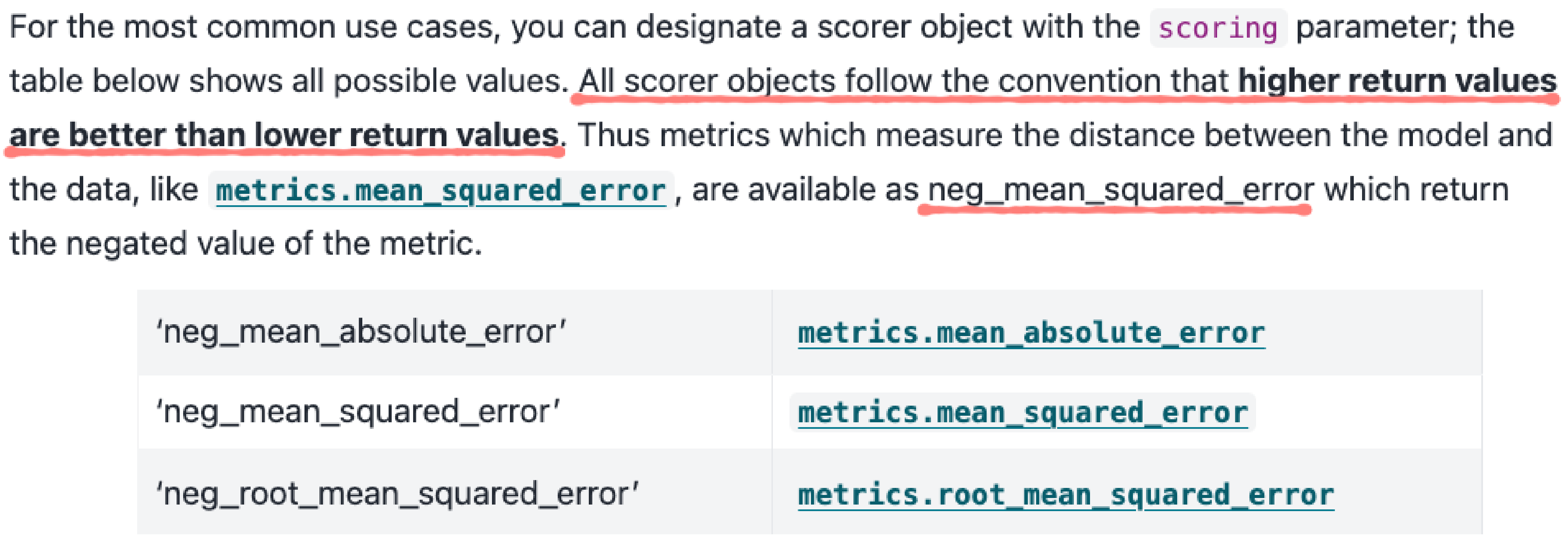
User guide. See the Regression metrics section for further details.

d2_absolute_error_score	D^2 regression score function, fraction of absolute error explained.
d2_pinball_score	D^2 regression score function, fraction of pinball loss explained.
d2_tweedie_score	D^2 regression score function, fraction of Tweedie deviance explained.
explained_variance_score	Explained variance regression score function.
max_error	The max_error metric calculates the maximum residual error.
mean_absolute_error	Mean absolute error regression loss.
mean_absolute_percentage_error	Mean absolute percentage error (MAPE) regression loss.
mean_gamma_deviance	Mean Gamma deviance regression loss.
mean_pinball_loss	Pinball loss for quantile regression.
mean_poisson_deviance	Mean Poisson deviance regression loss.
mean_squared_error	Mean squared error regression loss.
mean_squared_log_error	Mean squared logarithmic error regression loss.
mean_tweedie_deviance	Mean Tweedie deviance regression loss.
median_absolute_error	Median absolute error regression loss.
r2_score	R^2 (coefficient of determination) regression score function.
root_mean_squared_error	Root mean squared error regression loss.
root_mean_squared_log_error	Root mean squared logarithmic error regression loss.

In v1.4 or greater (previous versions have a "squared" argument in the MSE function)

MODEL SELECTION METRICS

- Models selection tools, such as "GridSearchCV" and "cross_val_score" use a scoring parameter



For the most common use cases, you can designate a scorer object with the `scoring` parameter; the table below shows all possible values. All scorer objects follow the convention that **higher return values are better than lower return values**. Thus metrics which measure the distance between the model and the data, like `metrics.mean_squared_error`, are available as neg_mean_squared_error which return the negated value of the metric.

'neg_mean_absolute_error'	`metrics.mean_absolute_error`
'neg_mean_squared_error'	`metrics.mean_squared_error`
'neg_root_mean_squared_error'	`metrics.root_mean_squared_error`