Training Linear Models

From Linear Regression to Regularization Techniques

Introduction to Linear Models

- **Definition:** Models that make predictions using a linear function of input features.
- **Types Covered:**
  - Linear Regression
  - Logistic Regression (for classification)
  - Regularized Models (Ridge, Lasso, Elastic Net)
- **Key Concepts:**
  - Cost functions (MSE, Log Loss)
  - Gradient Descent

Linear Regression

- **Equation:**

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

  - **Vectorized form:**

$$\hat{y} = \theta \cdot x = \theta^T x$$

- **Cost Function:** Mean Squared Error (MSE) (convex function)

$$MSE = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}^i - y^i \right)^2$$

- **The Normal Equation:** A closed-form solution to find the value of θ that minimizes the cost function.

$$\hat{\theta} = (X^T X)^{-1} \cdot X^T \cdot y$$

Gradient Descent

Goal: Minimize cost function by iteratively adjusting model parameters.

Types:

Batch GD (uses full dataset)
Stochastic GD (uses one random instance)
Mini-batch GD (uses small random batches)

Gradient Descent

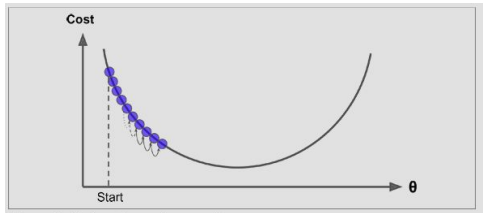


Figure 4-4. The learning rate is too small

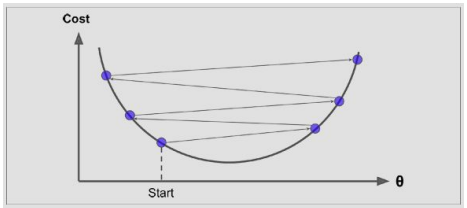


Figure 4-5. The learning rate is too large

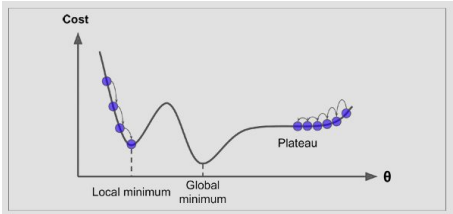
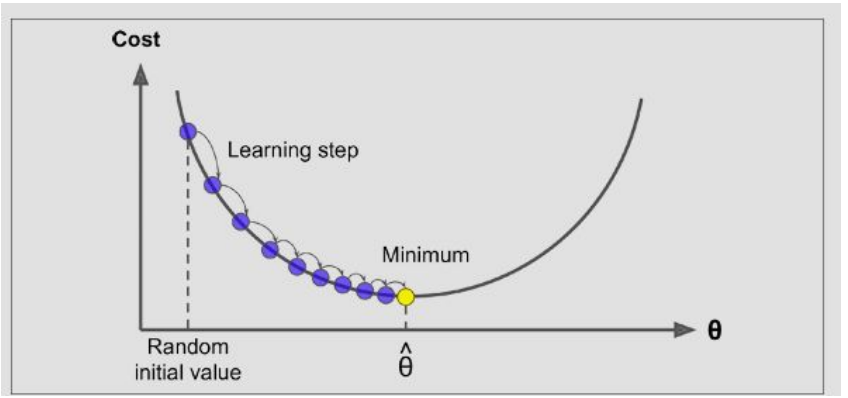


Figure 4-6. Gradient Descent pitfalls



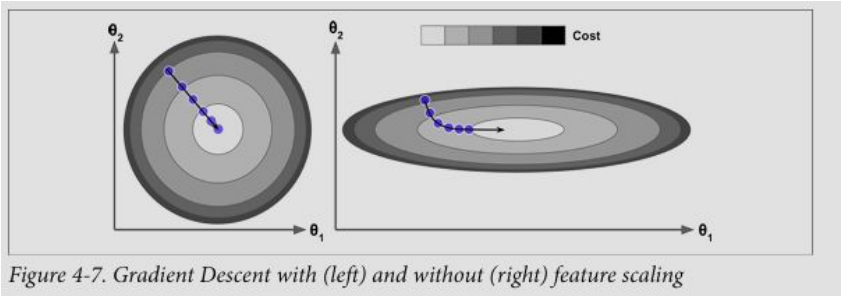


Figure 4-7. Gradient Descent with (left) and without (right) feature scaling

Batch Gradient Descent

Input:

- Dataset D = {(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)}
- Learning rate $\eta > 0$
- Number of epochs E
- Loss function $L(\theta, X, Y) \rightarrow$ returns average loss for dataset
- Gradient function $\nabla L(\theta, X, Y) \rightarrow$ returns gradient over the whole dataset

Initialize θ randomly

For epoch = 1 to E do:

$g \leftarrow \nabla L(\theta, X, Y)$ // Compute gradient for the entire dataset

$\theta \leftarrow \theta - \eta * g$ // Update parameters using the average gradient

End For

Output: Optimized parameters θ

- **Partial derivative of the cost function:**

$$\frac{\partial}{\partial \theta_j} MSE(\theta) = \frac{2}{m} \sum_{i=1}^{m} (\theta^T x^i - y^i) x_j^i$$

- **Gradient vector of the cost function**

$$\nabla_\theta MSE(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} MSE(\theta) \\ \frac{\partial}{\partial \theta_1} MSE(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} MSE(\theta) \end{pmatrix} = \frac{2}{m} X^T (X\theta - y)$$

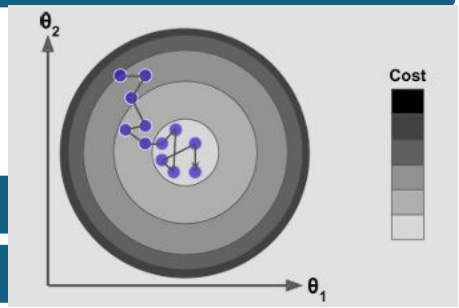
- **Gradient descent step**

$$\theta^{new} = \theta^{old} - \eta \nabla_\theta MSE(\theta)$$

Stochastic Gradient Descent

Input:

- Dataset D = {(x_1, y_1), (x_2, y_2), ..., ($x_□$, $y_□$)}
- Learning rate η > 0
- Number of epochs E
- Loss function L(θ, x, y) → returns loss for parameters θ
- Gradient function ∇L(θ, x, y) → returns gradient w.r.t. θ



Initialize θ randomly

For epoch = 1 to E do:

    Shuffle dataset D

    For each (x_i, y_i) in D do:

        g ← ∇L(θ, x_i, y_i) // Compute gradient for a single sample

        θ ← θ - η * g // Update parameters

    End For

End For

Output: Optimized parameters θ

- Batch Gradient Descent uses the whole training set to compute the gradients: **Slow when the training set is large**.
- Stochastic Gradient Descent picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- SGD is less regular than Batch Gradient Descent: **Higher oscillations.**
- Define **Learning Schedule**: $\frac{T_0}{t+T_1}$, where $T_0 = 5$ and $T_1 = 50$
- Randomness is good to escape from local optima.

Mini-batch Gradient Descent

Input:

- Dataset D = {(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)}
- Learning rate $\eta > 0$
- Number of epochs E
- Mini-batch size m $(1 < m < n)$
- Loss function $L(\theta, X\_batch, Y\_batch)$
- Gradient function $\nabla L(\theta, X\_batch, Y\_batch)$

Initialize θ randomly

For epoch = 1 to E do:

    Shuffle dataset D

    Partition D into batches of size m

    For each batch B in D do:

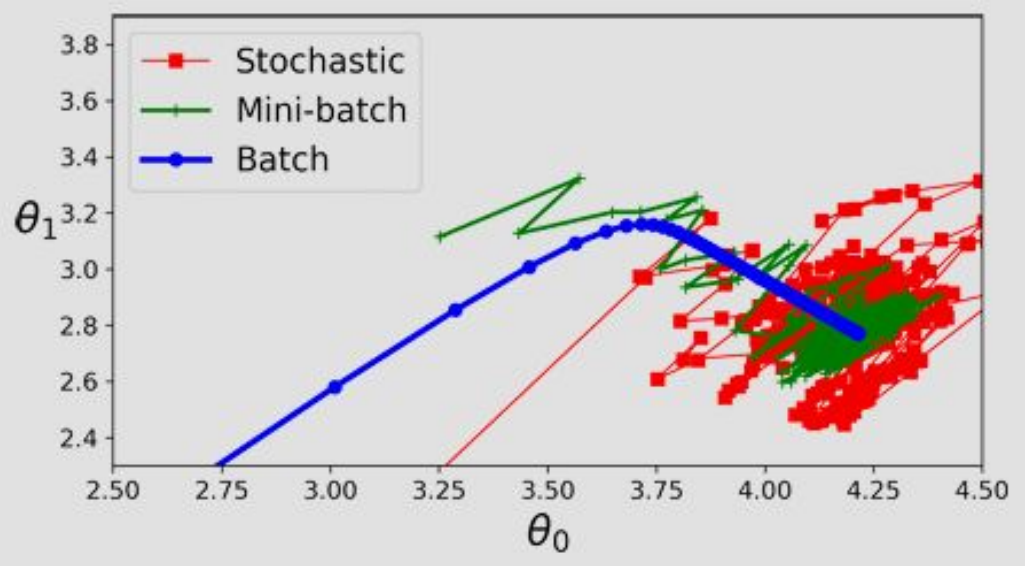
        $g \leftarrow \nabla L(\theta, X\_B, Y\_B)$ // Compute gradient
        $\theta \leftarrow \theta - \eta * g$ // Update parameters

    End For

End For

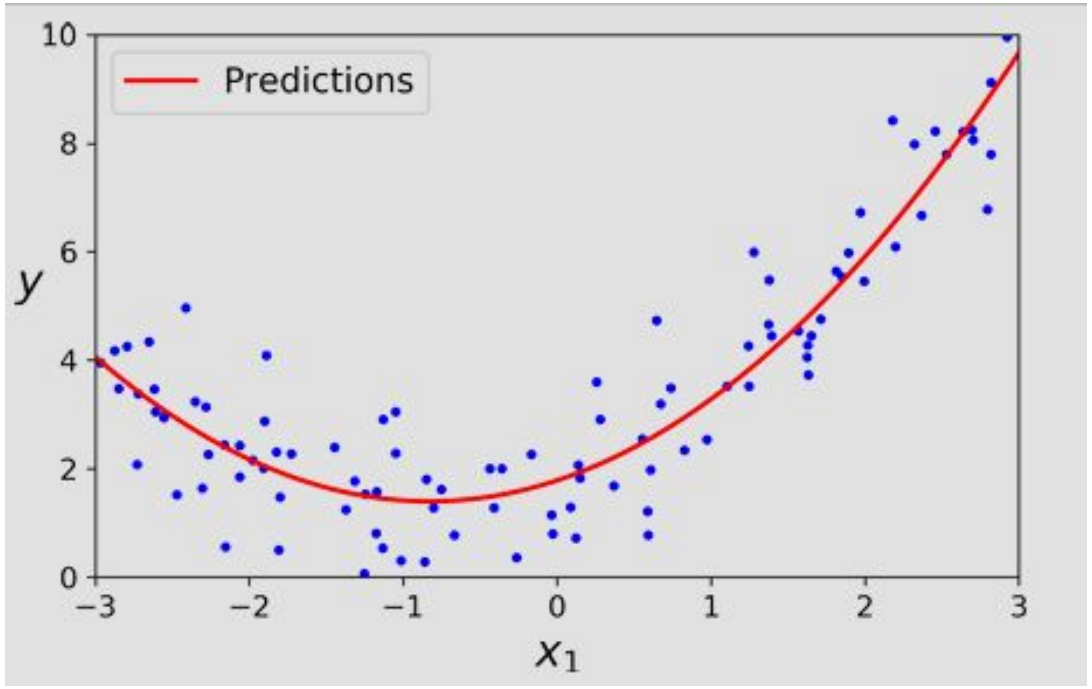
Output: Optimized parameters θ

- Mini-batch GD computes the gradients on small random sets of instances called mini-batches.



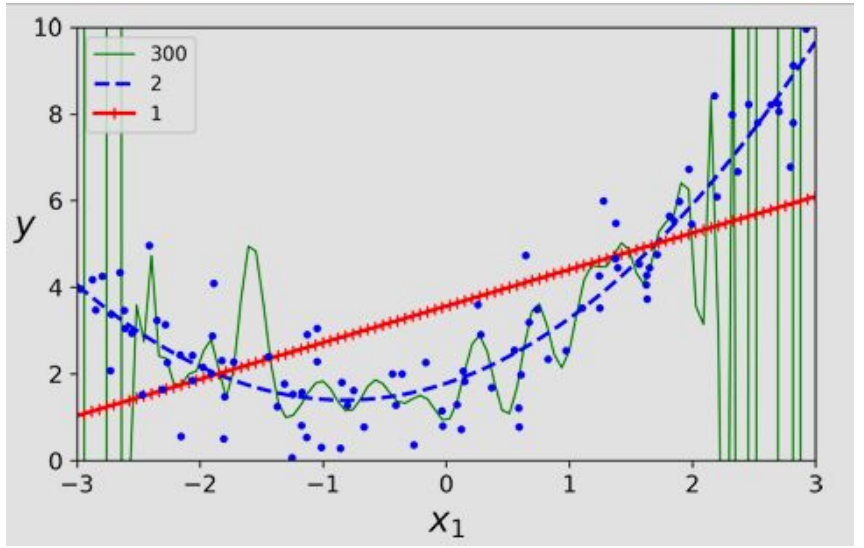
Polynomial Regression

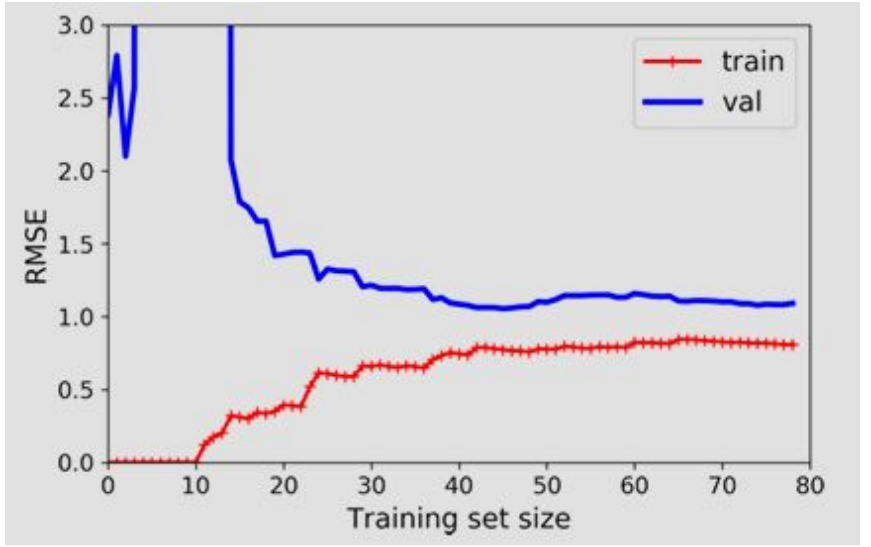
- **Idea:** Fit nonlinear data by adding powers of features.
- **Example:**
  $\hat{y} = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_n x^n$
- **Warning:** High-degree polynomials can overfit!



Learning Curves

- To check the underfitting and overfitting.





Regularized Linear Models

A good way to reduce overfitting.

A simple way to regularize a polynomial model is to reduce the number of polynomial degrees.

For a linear model, regularization is typically achieved by constraining the weights of the model.

Ways to constrain weights:

Ridge Regression
Lasso Regression
Elastic Net

Ridge Regression (L2 Regularization)

- Ridge Regression cost function:

$$J(\theta) = MSE(\theta) + \alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$$

  - The hyperparameter α controls how much you want to regularize the model.
  - The bias term θ_0 is not regularized.
- Ridge Regression closed-form solution

$$\hat{\theta} = (X^T X + \alpha A)^{-1} \cdot X^T \cdot y$$

  - Where A is the $(n + 1) \times (n + 1)$ identity matrix
- **Effect:** Shrinks coefficients but never to zero.

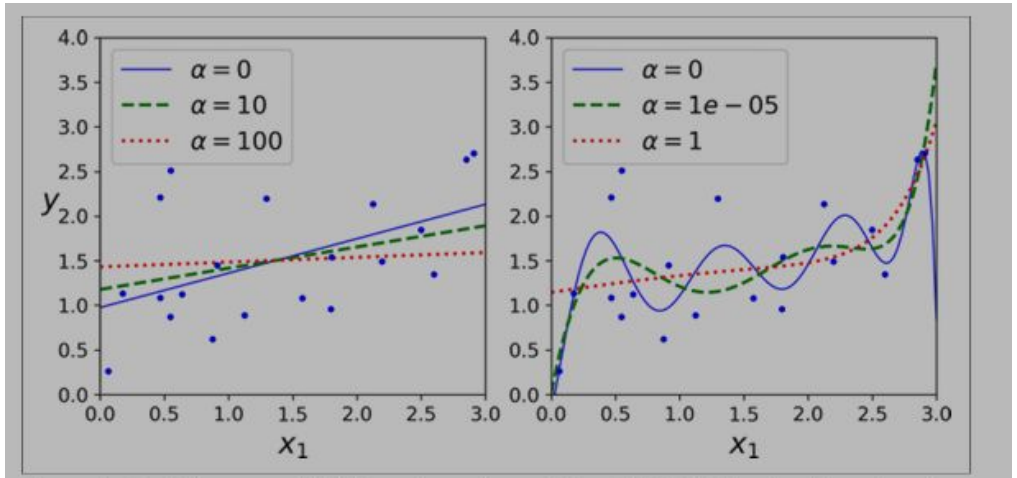


Figure 4-17. A linear model (left) and a polynomial model (right), both with various levels of Ridge regularization

Lasso Regression (L1 Regularization)

- Least Absolute Shrinkage and Selection Operator (Lasso) Regression cost function:

$$J(\theta) = MSE(\theta) + \alpha \sum_{i=1}^{n} |\theta_i|$$

- **Effect:** Can shrink coefficients to zero (feature selection).

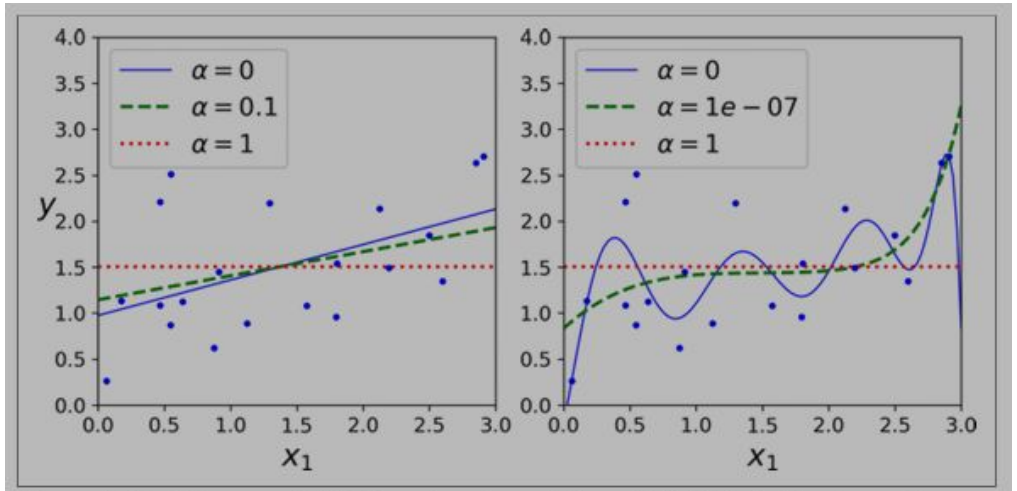


Figure 4-18. A linear model (left) and a polynomial model (right), both using various levels of Lasso regularization

Elastic Net

- **Combination:** Mix of L1 and L2 penalties

$$J(\theta) = MSE(\theta) + r\alpha \sum_{i=1}^{n} |\theta_i| + \frac{1-r}{2} \alpha \sum_{i=1}^{n} \theta_i^2$$

- When $r = 0$, Elastic Net is equivalent to Ridge Regression, and when $r = 1$, it is equivalent to Lasso Regression.
- Elastic Net is preferred over Lasso because Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated.

Early Stopping

- A different way of regularization.
- Stopping training as soon as the validation error reaches a minimum.
- Geoffrey Hinton called it a “beautiful free lunch.”

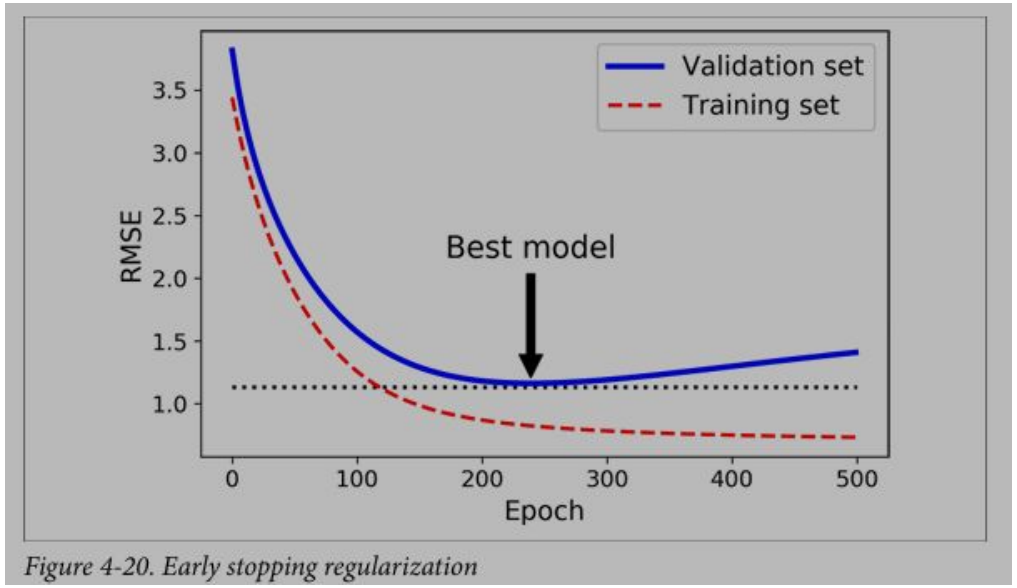


Figure 4-20. Early stopping regularization

Logistic Regression (Classification)

- Logistic Regression model estimated probability (vectorized form):

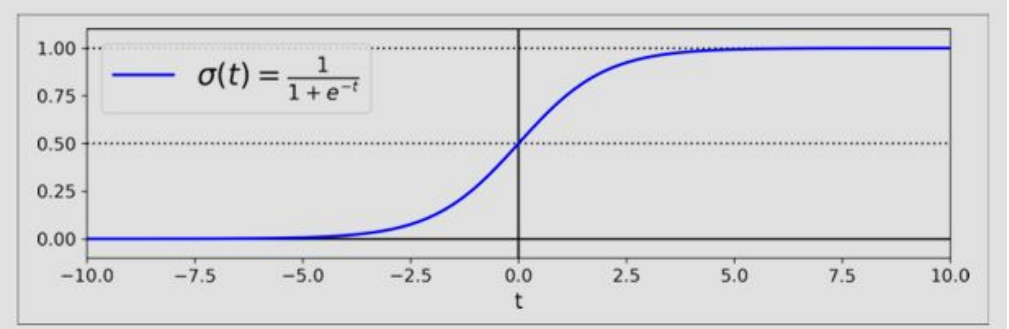
$$\hat{p} = h_\theta(x) = \sigma(x^T\theta)$$

- Sigmoid or Logistic or S-shaped Function:

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

- Logistic Regression model prediction:

$$\hat{y} = \begin{cases} 0 \; if \; \hat{p} < 0.5 \\ 1 \; if \; \hat{p} \geq 0.5 \end{cases}$$



Training and Cost Function

- Logistic Regression cost function (log loss)

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y^i \log\left(\hat{p}^i\right) + \left(1 - y^i\right)\log\left(1 - \hat{p}^i\right)\right]$$

- No known closed-form equation exists but it is a convex function.
- Use Gradient Descent (or any other optimization algorithm) to find the global minimum.
- Logistic cost function partial derivatives

$$\frac{\partial}{\partial\theta_j}J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma\left(\theta^T x^i\right) - y^i\right)x_j^i$$

Softmax Regression (Multinomial Logistic Regression)

- Generalization of Logistic Regression for multiple classes.
- Softmax score for class k

$$s_k(x) = x^T \theta^k$$

- Each class has its own dedicated parameter vector θ^k. All these vectors are typically stored as rows in a $parameter\ matrix\ \Theta$.
- Estimate the probability $\hat{p}^k$ that the instance belongs to class k by running the scores (logits or log-odds) through the Softmax function:

$$\hat{p}^k = \sigma\big(s(x)\big)_k = \frac{e^{s_k(x)}}{\sum_{j=1}^{K} e^{s_j(x)}}$$

- Softmax Regression classifier prediction

$$\hat{y} = \underset{k}{argmax}\ \sigma\big(s(x)\big)_k$$

Softmax Regression (Multinomial Logistic Regression)

- **Cross entropy** cost function is used to measure how well a set of estimated class probabilities matches the target classes.

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^i \log\left(\hat{p}_k^i\right)$$

- y_k^i is the target probability that the i-th instance belongs to class k. In general, it is either equal to 1 or 0, depending on whether the instance belongs to the class or not.
- Cross entropy gradient vector for class k

$$\frac{\partial}{\partial \theta^k} J(\Theta) = \frac{1}{m}\sum_{i=1}^{m}\left(\hat{p}_k^i - y_k^i\right)x^i$$

KL Divergence (Kullback-Leibler Divergence)

- What is KL Divergence?
  - A measure of how one probability distribution differs from a second, reference probability distribution.
  - Often used in statistics, machine learning (e.g., variational inference), and information theory.
- Given two probability distributions P (true) and Q (approximation):

$$D_{KL}(P||Q) = \sum_{x} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

Entropy

- Entropy quantifies how much "surprise" is in the data.
- High entropy = more uncertainty (e.g., uniform distribution).
- Low entropy = more certainty (e.g., one class dominates).
- Decision Trees (e.g., ID3, C4.5)
  - Entropy is used to measure the purity of a split.
  - Lower entropy → better split (more homogeneous classes).
- Classification Tasks
  - Cross-entropy loss (based on entropy) measures how far the predicted distribution is from the true distribution.
- Uncertainty Quantification
  - Entropy helps assess how confident a model is in its predictions.

KL Divergence in Terms of Cross-Entropy

- Entropy of a distribution P:

$$H(P) = -\sum_{x} P(x)\log P(x)$$

- Cross-Entropy between distributions P and Q:

$$H(P,Q) = -\sum_{x} P(x)\log Q(x)$$

- KL Divergence as a Difference:

$$D_{\mathrm{KL}}(P||Q) = H(P,Q) - H(P)$$

  - If Q approximates P well, cross-entropy is close to entropy ⇒ small KL divergence.
  - If Q is far from P, cross-entropy is much larger ⇒ large KL divergence.