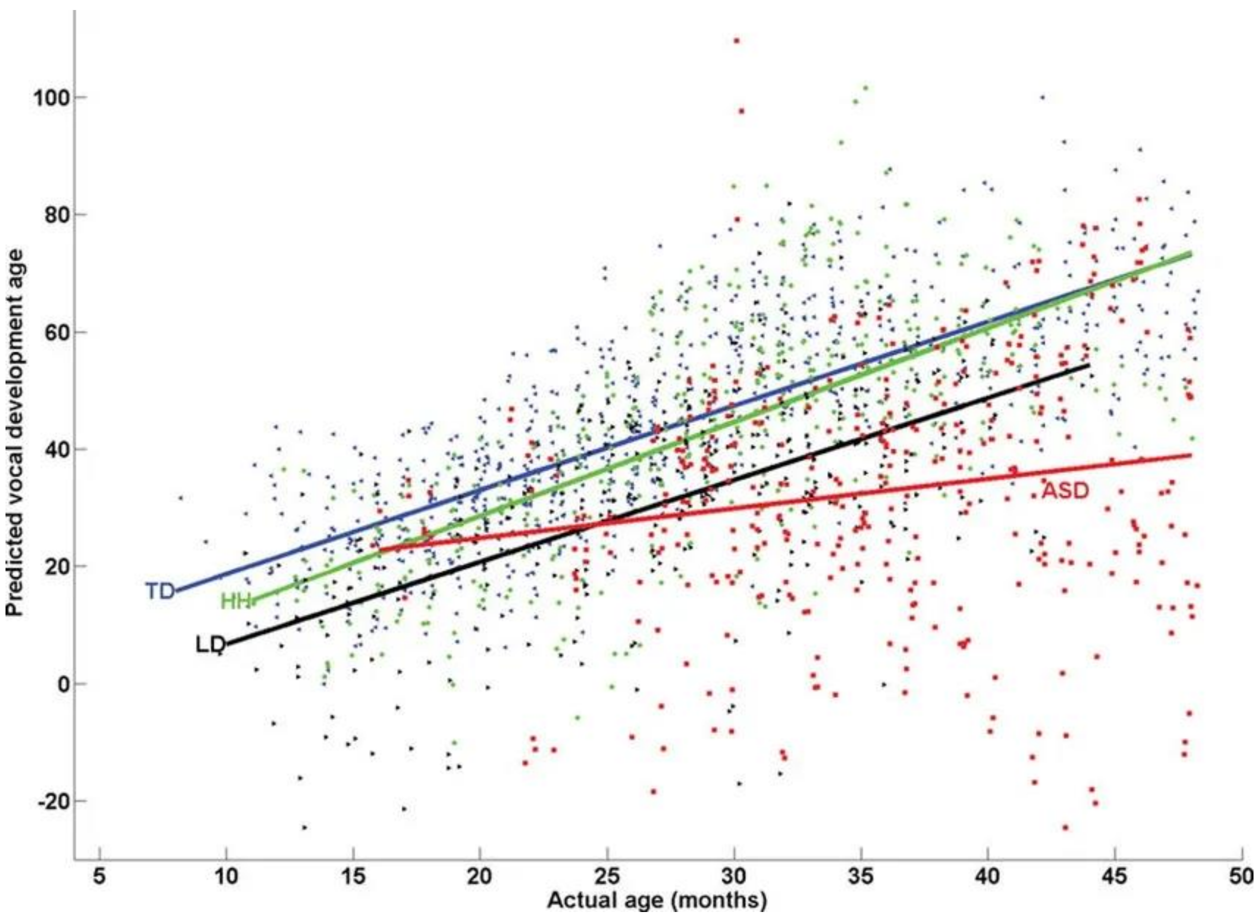
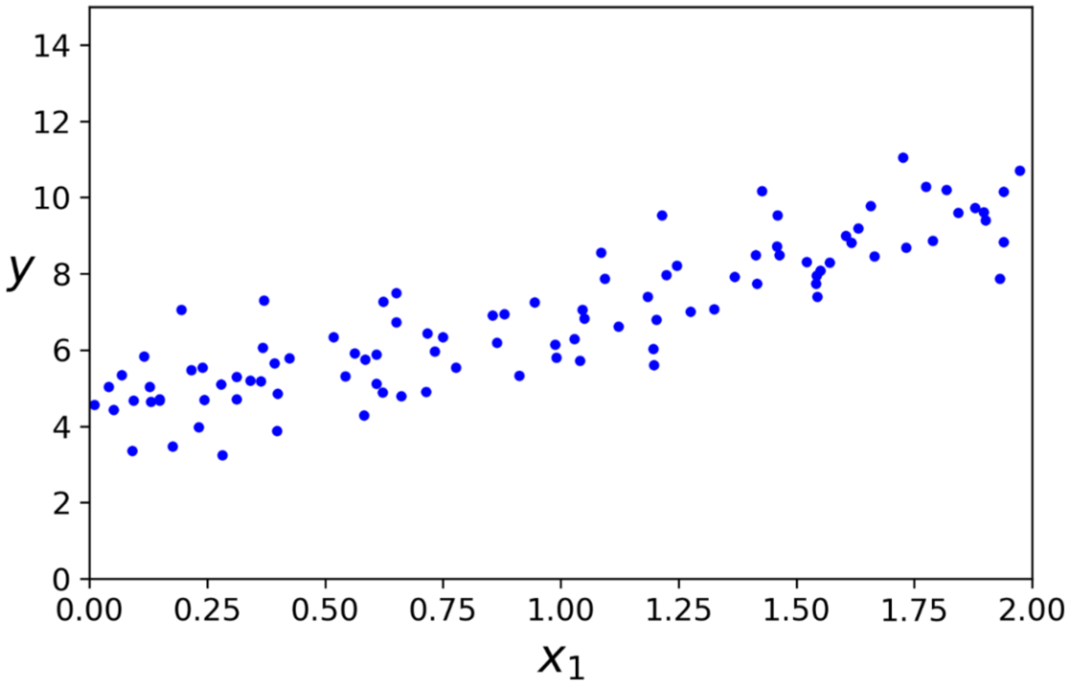
4. Training Models



Linear Regression:

Types:

- **Direct "Closed-Form" Equation**
- **Gradient Descent Variations**:
  - Batch Gradient Descent
  - Mini-batch Gradient Descent
  - Stochastic Gradient Descent



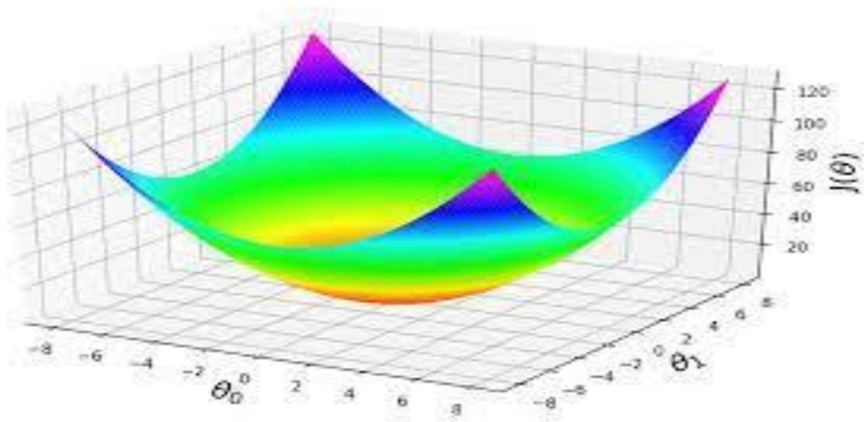
Model Equation:

- **General Form**: y=θ0+θ1x1+θ2x2+⋯+θnxn*y*=*ϑ*0+*ϑ*1*x*1+*ϑ*2*x*2+⋯+*ϑnxn*

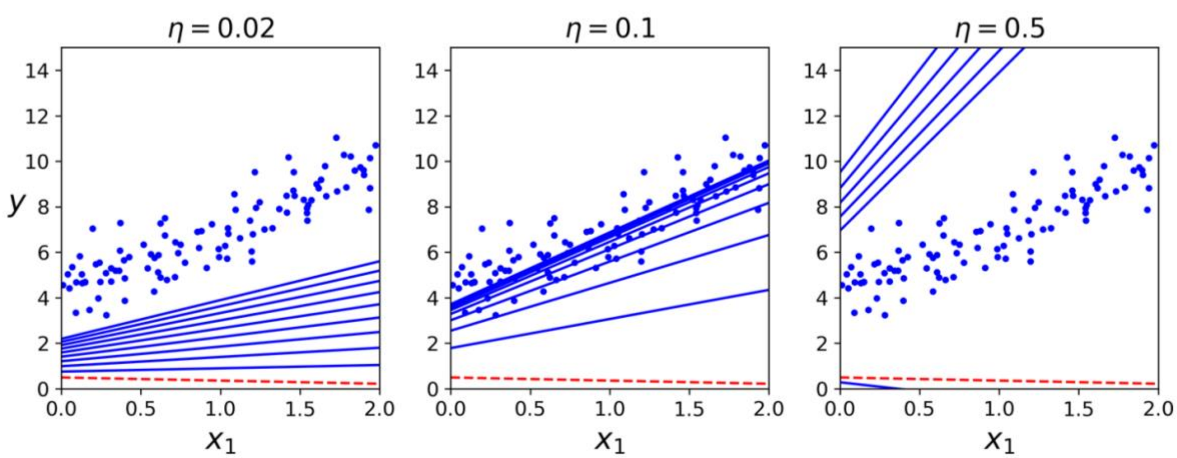
Training Approach:

1. **Objective**: Set parameters to best fit the training set. Find θ*ϑ* that minimizes the Root Mean Square Error (RMSE).
2. **MSE Cost Function**: MSE(X,hθ)=1m∑(θTx(i)−y(i))2MSE(**X**,*hϑ*)=*m*1∑(*ϑTx*(*i*)−*y*(*i*))2
3. **Solution Methods**:
   - **Normal Equation**: θ=(XTX)−1XTy*ϑ*=(**X***T***X**)−1**X***T***y**
   - **Singular Value Decomposition (SVD)**: More efficient computational approach.

Gradient Descent:

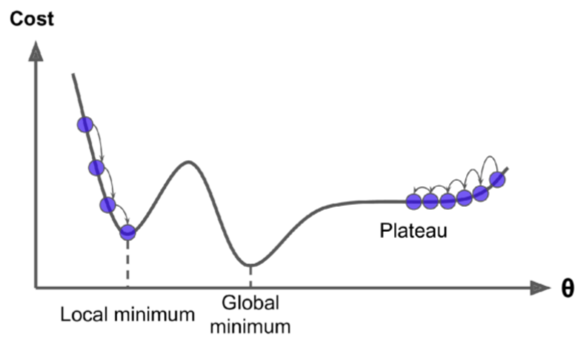


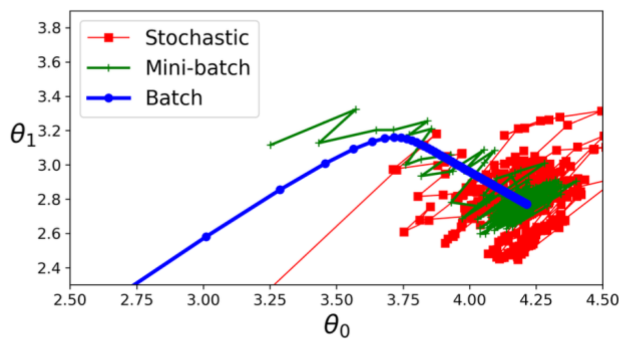
- **Basic Idea**: Iteratively tweak parameters to minimize the cost function, analogous to descending a mountain in the steepest direction. The height is the RMSE
- **Parameter Initialization**: Begin with random values (random initialization).
- **Learning Rate**: Crucial hyperparameter determining the size of each step. Too small leads to slow convergence, too large can overshoot the minimum.



Types of Gradient Descent:

1. **Batch Gradient Descent**: Uses the entire training set, can be slow on large datasets.
2. **Stochastic Gradient Descent (SGD)**: Uses random instances; faster but less precise. Because it's random, never stops, hence the use of simulated annealing for stopping.
3. **Mini-batch Gradient Descent**: Uses small random sets of instances; balances speed and precision.

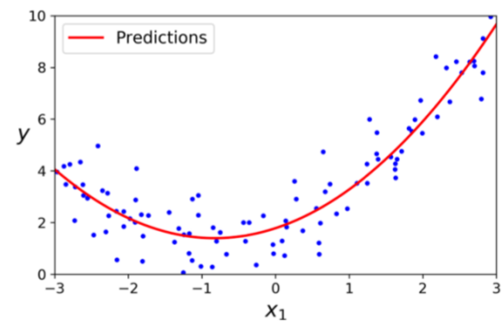




Algorithm	Large *m*	Out-of-core support	Large *n*	Hyperparams	Scaling required	Scikit-Learn
Normal Equation	Fast	No	Slow	0	No	N/A
SVD	Fast	No	Slow	0	No	`LinearRegression`
Batch GD	Slow	No	Fast	2	Yes	`SGDRegressor`
Stochastic GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`
Mini-batch GD	Fast	Yes	Fast	≥2	Yes	`SGDRegressor`

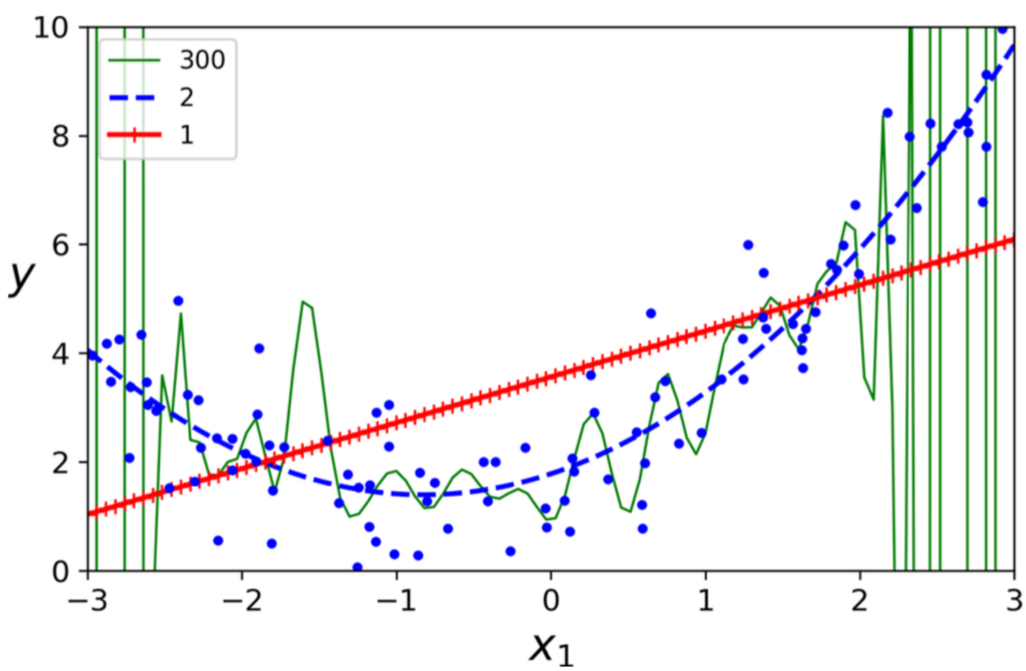
Polynomial Regression:

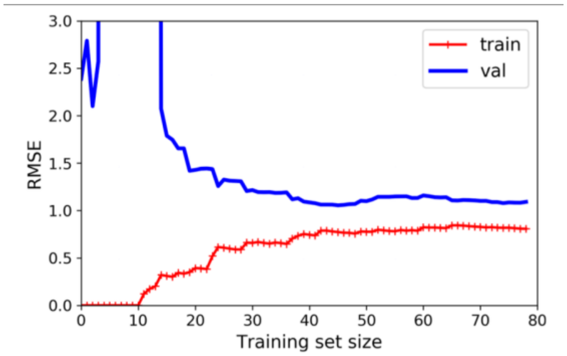
- **Concept**: Models relationships that are not linear by using polynomial features.

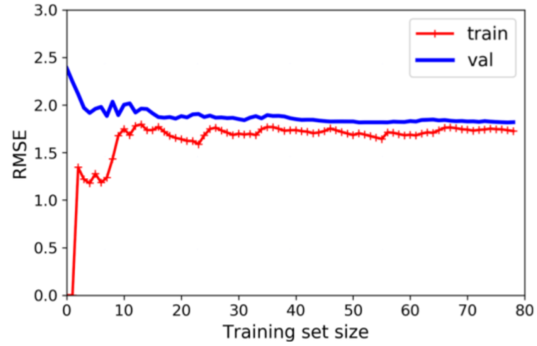


Learning Curves:

- **Interpretation:** Used to determine if a model is overfitting or underfitting.
- **Overfitting:** Model performs well on training data but poorly on validation data.
- **Underfitting:** Poor performance on both training and validation data.

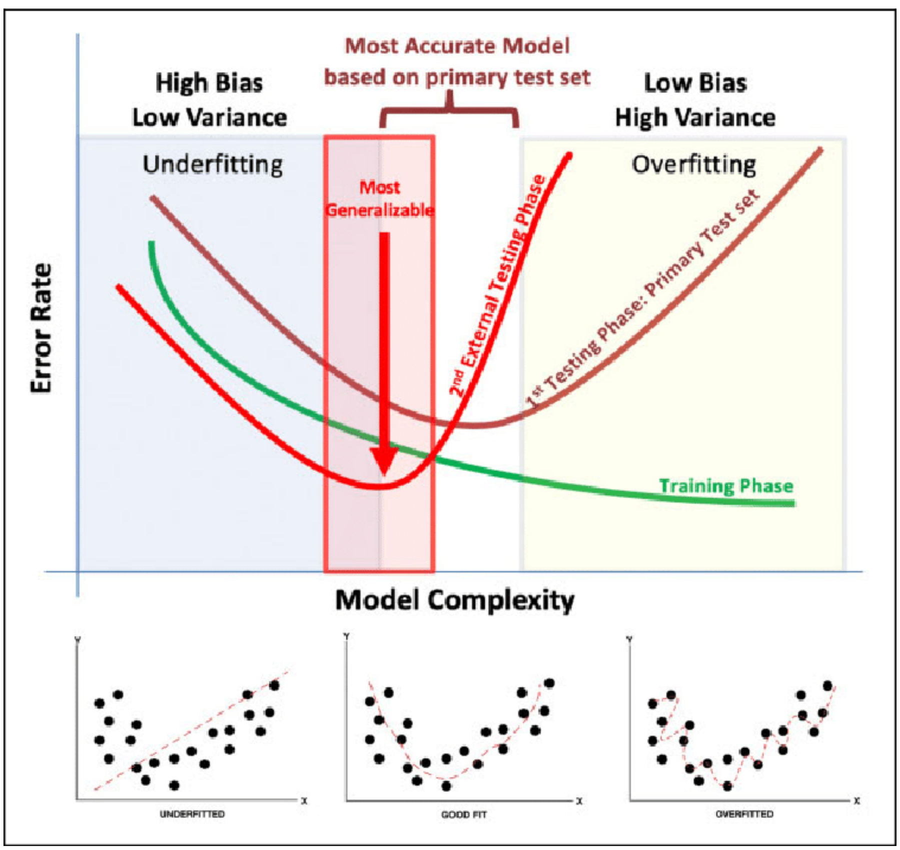






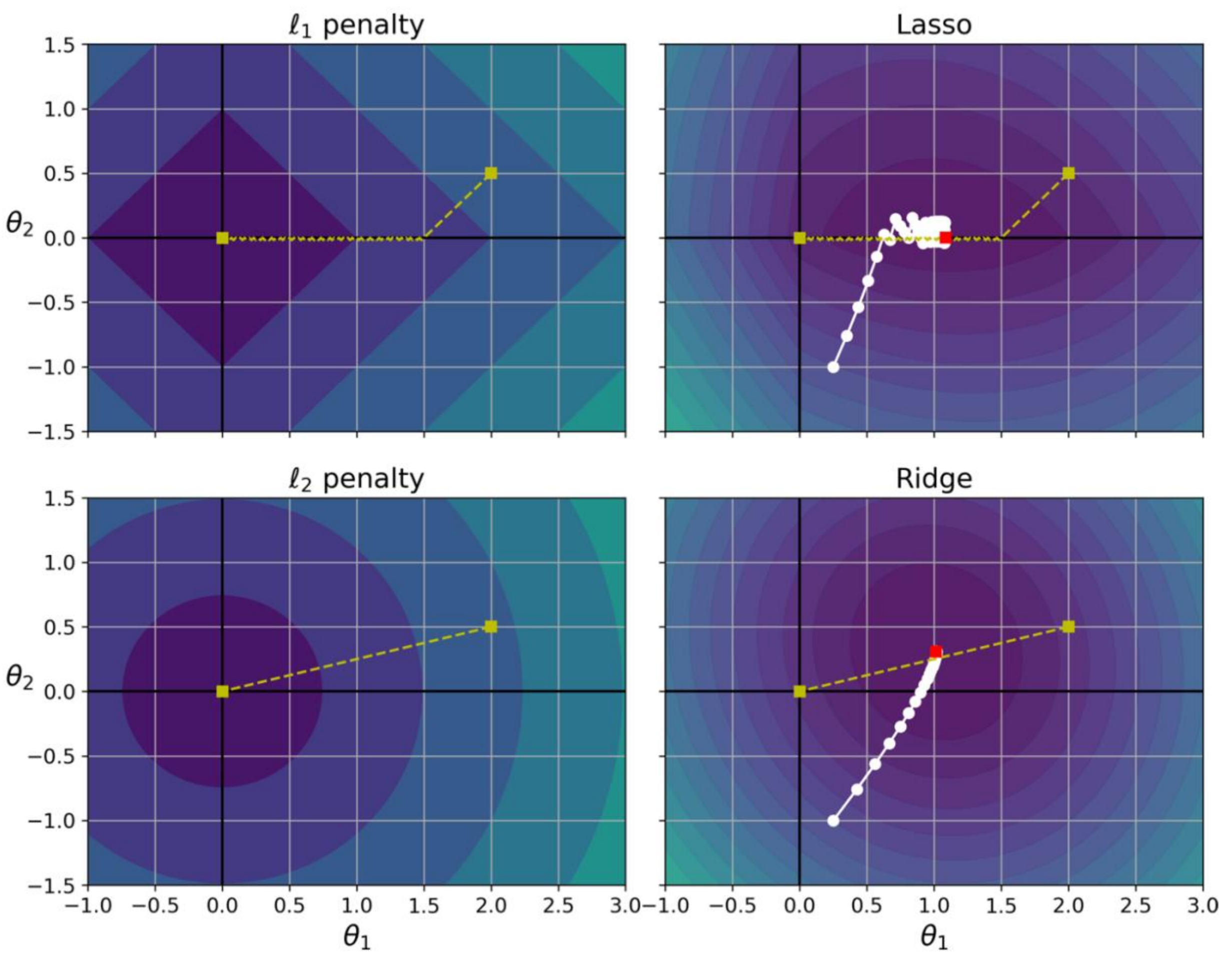
The Bias/Variance Trade-off:

- **Bias:** Error due to wrong assumptions, like assuming linearity in a non-linear context.
- **Variance:** Error due to the model's excessive sensitivity to small variations in training data.
- **Irreducible Error:** Due to noisiness in the data itself.
- **Complexity Trade-off**: Increasing model complexity increases variance and reduces bias, and vice versa.



Regularization of Linear Models:

- **Purpose**: To prevent overfitting by constraining the model.
- **Types**:
  - **Ridge Regression**: Adds penalty equivalent to the square of the magnitude of coefficients.
  - **Lasso Regression**: Adds penalty equivalent to the absolute value of the magnitude of coefficients. Can completely eliminate some features' weights.
  - **Elastic Net**: Combines penalties of Ridge and Lasso.



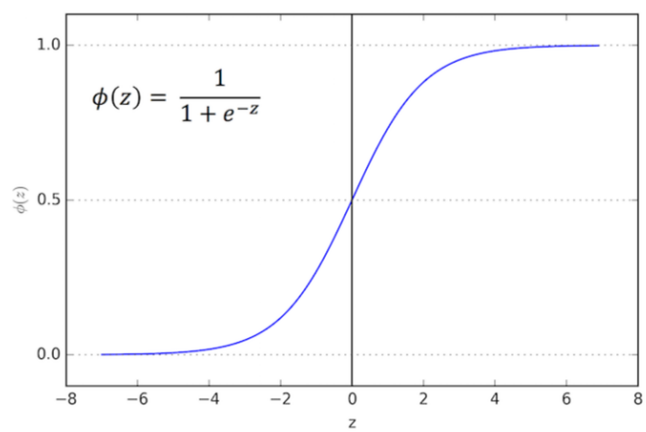
Logistic Regression

- **Concept**: Estimates the probability of an instance belonging to a particular class using the logistic function.

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma\left(\mathbf{x}^{\mathsf{T}}\boldsymbol{\theta}\right)$$

- **Model Equation**:

- **Logistic Function**:



$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{if } y = 1 \\ -\log(1-\hat{p}) & \text{if } y = 0 \end{cases}$$

Training and Cost Function:

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

Key Differences from Linear Regression:

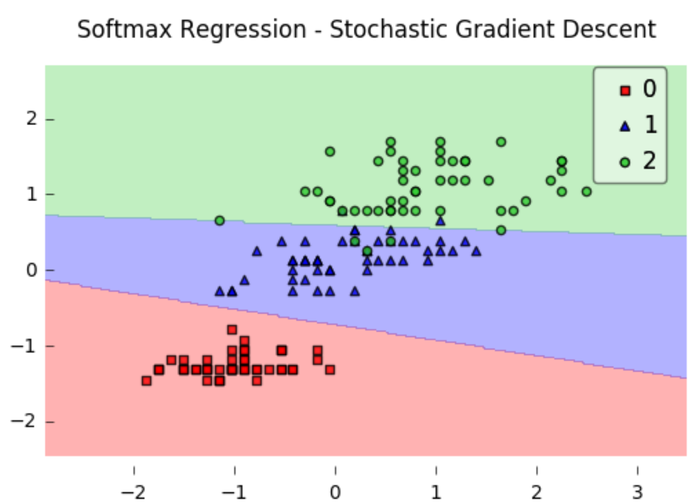
- **Output Range**: Probability values between 0 and 1 for Logistic Regression vs. unbounded continuous values for Linear Regression.
- **Objective Function**: Minimization of log loss for Logistic Regression vs. minimization of squared residuals for Linear Regression.
- **Nature of Prediction**: Suitable for binary classification tasks vs. predicting quantitative outputs.

Decision Boundaries in Logistic Regression:

- **Concept**: Determines the threshold where the probability of a class switches from one category to another.

Softmax Regression:

- **Application**: Extends Logistic Regression to multiple classes without needing multiple binary classifiers.
- **Score Calculation**: Computes a score for each class, then applies the softmax function to these scores to estimate probabilities.
- **Training**: Uses cross entropy as the cost function to measure the model's performance.



Cross Entropy in Classification:

- **Role**: Measures how well the predicted probability distribution of class labels matches the actual distribution.
- **Computation**: Sum of the negative log of predicted probabilities, weighted by the actual distribution.