Multinomial Logistic Regression

Emmanuel Pedernal

MSDS

[MSS603M_G01]

Let’s Review……

We have the **dependent variable**

the **independent variables**

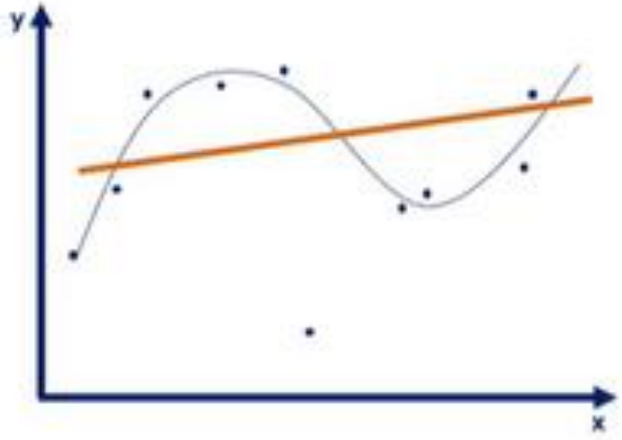
$$\boxed{\hat{y}} = \boxed{b_1} \cdot \boxed{x_1} + \boxed{b_2} \cdot \boxed{x_2} + \ldots + \boxed{b_k} \cdot \boxed{x_k} + \boxed{a}$$

and the **regression coefficients.**

Compute Mean Square Error (Loss)

$$L = \frac{1}{m} \sum_{i=1} \left(y_i - (w^T x_i + b) \right)^2$$

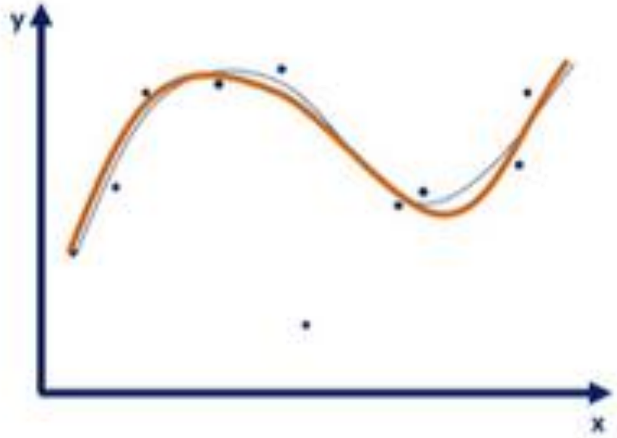
An **underfitted** model



Doesn't capture any logic

- High loss
- Low accuracy

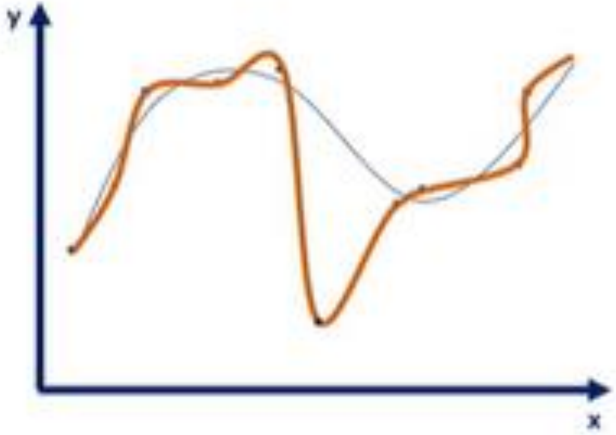
A **good** model



Captures the underlying logic of the dataset

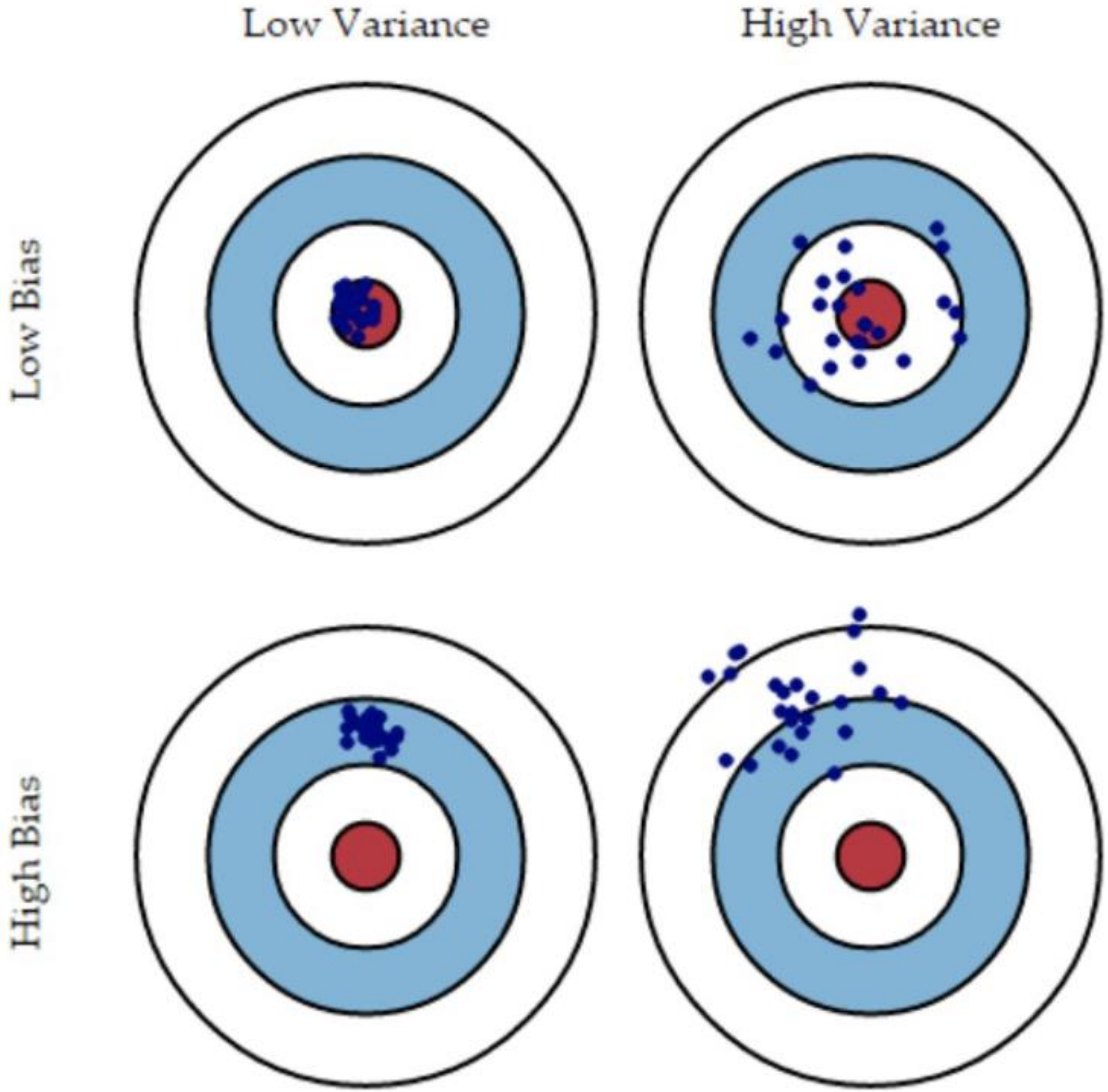
- Low loss
- High accuracy

An **overfitted** model



Captures all the noise, thus "missed the point"

- Low loss
- Low accuracy

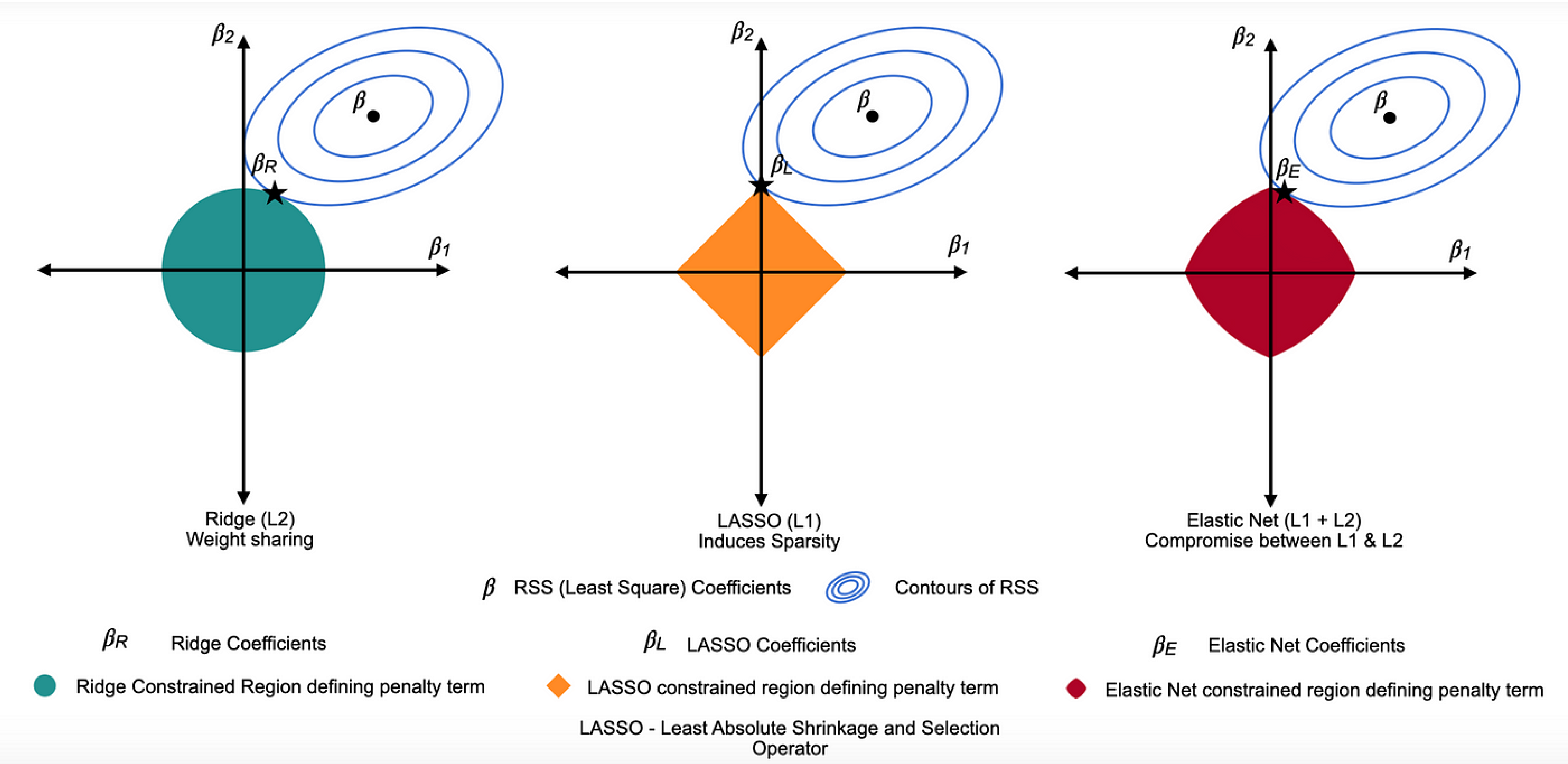

Low Variance
High Variance
Low Bias
High Bias

	Training error	Test error
Underfit model	HIGH	HIGH
Overfit model	LOW	HIGH

Overfit models fit the data + the noise, and prioritizes noise over general trend.

Avoid overfitting & better generalize unseen data

- L1 (Lasso)– uses absolute value of weights as penalty
  - May reduce weight to zero (dead weight) like feature selection
- L2 (Ridge) – uses squared values as penalty
  - Encourages small weights but not zero
  - Retains all features but reduces impact
  - Relevant if you think all selected features are important
- Elastic Net - uses both L1 and L2
  - Combines the benefits of both Lasso and Ridge, allowing for feature selection and shrinkage simultaneously.
  - Elastic Net is often used when there are many correlated features, which can be problematic for Lasso.



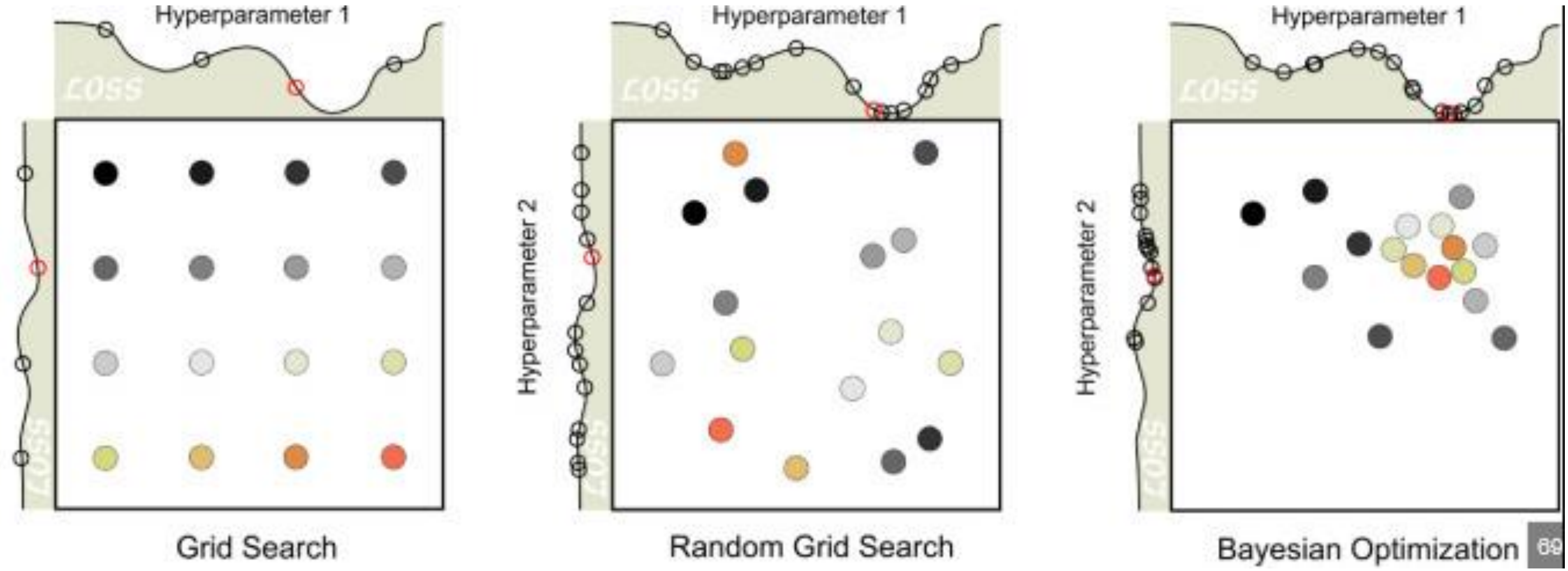
β2
β
βR
β1
Ridge (L2)
Weight sharing
β2
β
βL
β1
LASSO (L1)
Induces Sparsity
β2
β
βE
β1
Elastic Net (L1 + L2)
Compromise between L1 & L2
β RSS (Least Square) Coefficients
Contours of RSS
βR Ridge Coefficients
βL LASSO Coefficients
βE Elastic Net Coefficients
Ridge Constrained Region defining penalty term
LASSO constrained region defining penalty term
Elastic Net constrained region defining penalty term
LASSO - Least Absolute Shrinkage and Selection Operator

Hyperparameter Tuning

- Grid Search:
  - Use if small and well-defined search space.
  - have the computational resources to afford trying all combinations.
- Random Search:
  - Use when the search space is large and want to avoid the inefficiency of Grid Search.
  - Computational resources are limited, and want to find a good solution quickly.
- Bayesian Optimization:
  - Use when the objective function is expensive to evaluate (e.g., deep learning training, simulations).
  - You want to optimize hyperparameters with fewer evaluations and smarter exploration

Parameters

- Epochs – How many times the entire dataset pass through the model
- Learning Rate - Controls how much to change the model in response to the error each time the model weights are updated.
- Batch Size - The number of training examples used in one forward/backward pass. Small batch sizes can make the model generalize better, but large batch sizes may lead to faster convergence.
- Regularization Strength (L1, L2, Elastic Net) – Penalty against weights


Hyperparameter 1
Hyperparameter 1
Hyperparameter 1
Hyperparameter 2
Hyperparameter 2
Grid Search
Random Grid Search
Bayesian Optimization

Reduce loss through Gradient Descent

- Updates parameters iteratively

$$w \coloneqq w - \eta \frac{\partial L}{\partial w_k}$$

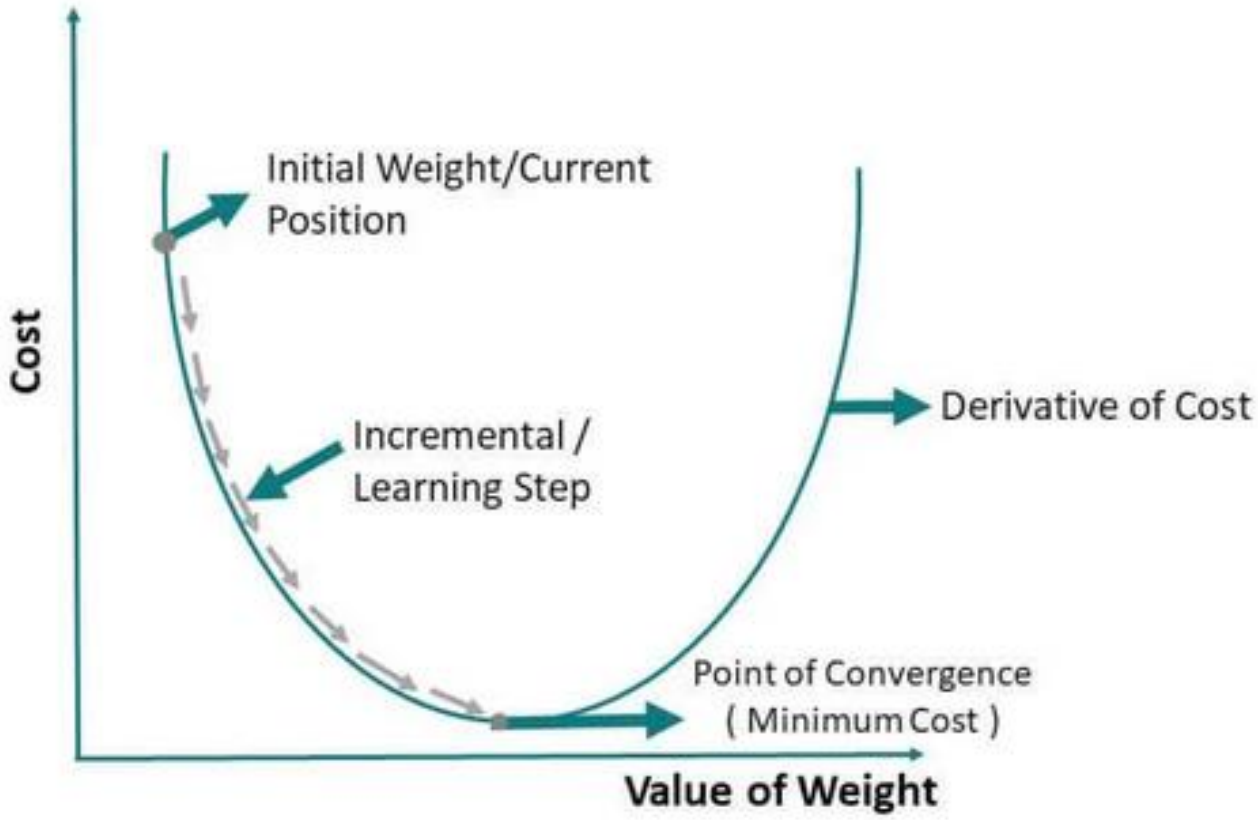
$$b \coloneqq b - \eta \frac{\partial L}{\partial b_k}$$

- Where η is the learning rate and partial derivates are as follows

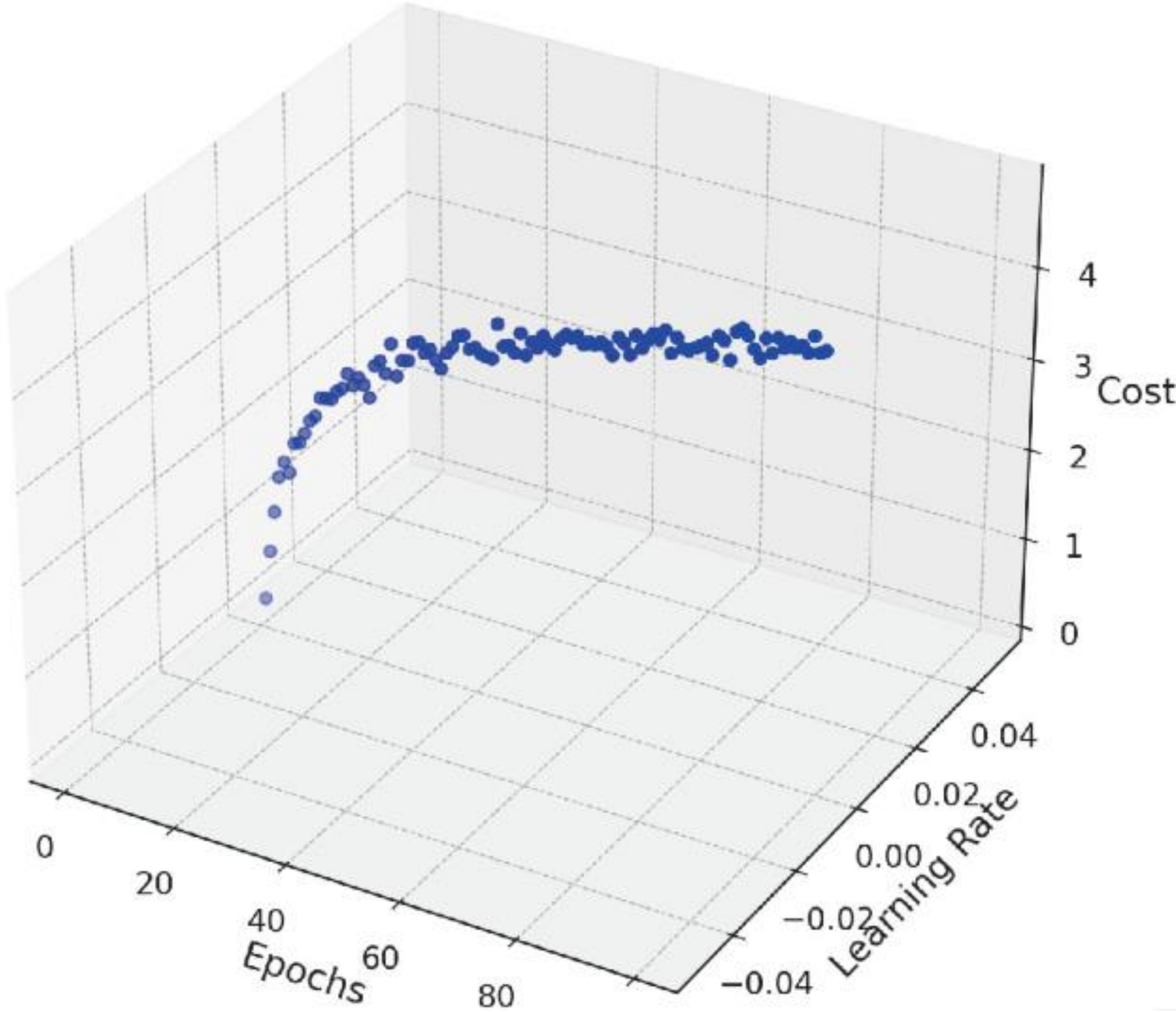
$$\frac{\partial L}{\partial w} = \frac{2}{m} \sum_{i=1}^{m} (y_i - (w^T x_i + b^1)(-x_i)$$

$$\frac{\partial L}{\partial b} = \frac{2}{m} \sum_{i=1}^{m} (y_i - (w^T x_i + b^1)(-1)$$

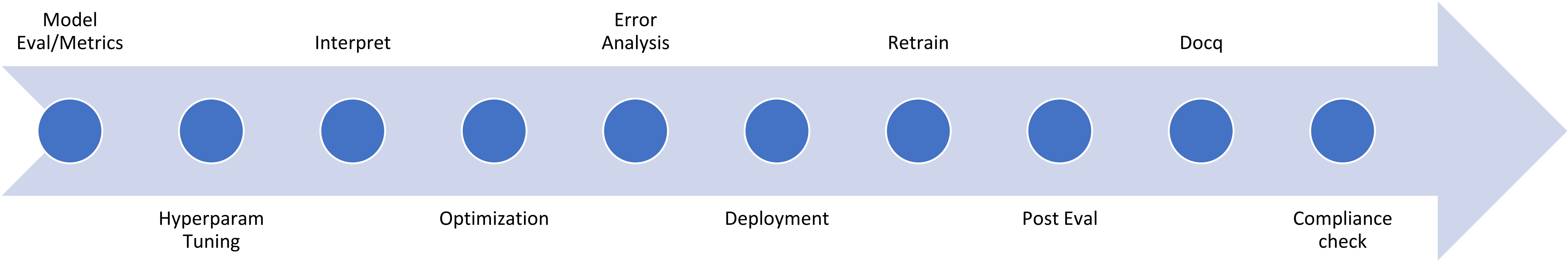
- Convergence - model weights have stopped changing significantly, and the loss function has reached plateau
    - Reduce Learning rate or
    - Use Early stopping



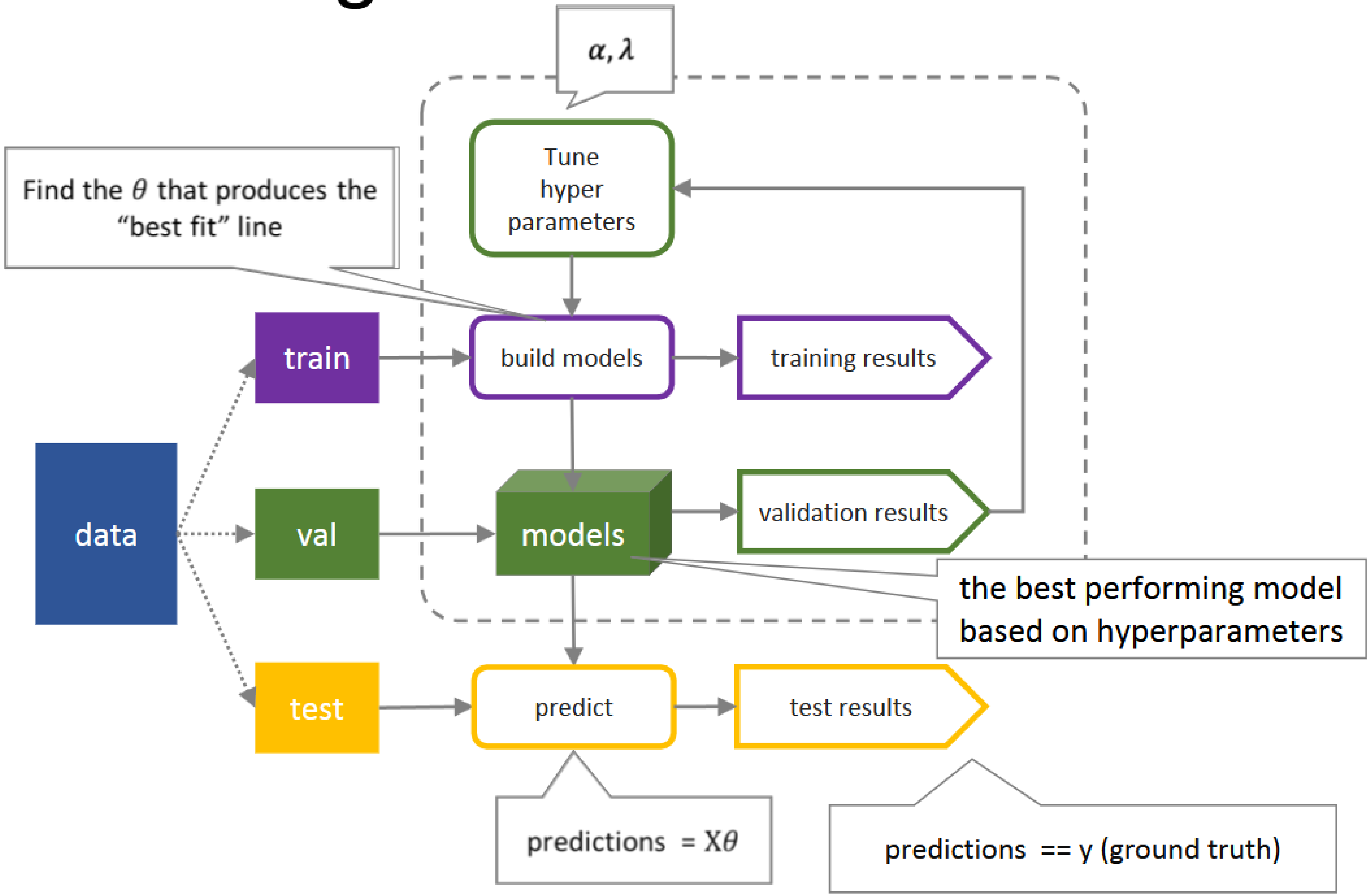
- After doing mentioned steps, if not satisfied repeat with different parameters.
- Finally, we can validate the model again.



Post-Training Steps



Pipeline Linear regression



Credits to Dr. Thomas Tiam Lee

Definitions

- **Probability** - Chance of an event happening over total number of all outcomes.
  - If you roll a die, what's the probability of getting a 4?
    - P(4)=1/6
- **Odds** - Ratio of probability of an event happening to it not happening.
  - If the probability of rain is 0.75.
    - Odds in favor of rain = 0.75 / 1-0.75 = 3:1
- **Odds-ratio** - Compares the odds of the event happening for two different values of a predictor variable.

Group	Got Better	Did Not Get Better
Drug Group	40	10
Placebo Group	20	30

odds ratio = (40)(10)/ (30)(20) = 6; People in the drug group are 6 times more likely to get better compared to those in the placebo group.

Multiple Logistic Regression supports the same….

Target Class

$$\log \frac{P(Y = \boxed{k} | X)}{P(Y = \boxed{K} | X)} = b_1 x_1 + b_2 x_2 + \cdots + b_n x_n + a$$

Reference Class

- WITH activation function [converts logits to probabilities] (Softmax)

$$P(y = c_k | x) = \frac{e^{z_k}}{\Sigma_j^C = 1 e^{z_j}}$$

Suspose: [0.1,1.0,2]

(logits scores) * e (Euler's number)

$$e^{1.0} \approx 2.718$$
$$e^{0.1} \approx 1.105$$
$$e^{2} \approx 7.389$$

$$\sum_{j=1}^{3} e^{z_j} \approx 11.212$$

- $P(y = Cat|z) = \frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} = \frac{2.718}{11.212} \approx 0.242$
- $P(y = Hamster|z) = \frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} = \frac{1.105}{11.212} \approx 0.099$
- $P(y = Dog|z) = \frac{e^{z_1}}{\sum_{j=1}^{3} e^{z_j}} = \frac{7.389}{11.212} \approx \mathbf{0.659}$

Choosing your metric

Accuracy

% of examples correctly predicted

Not ideal for data with rare classes

Precision

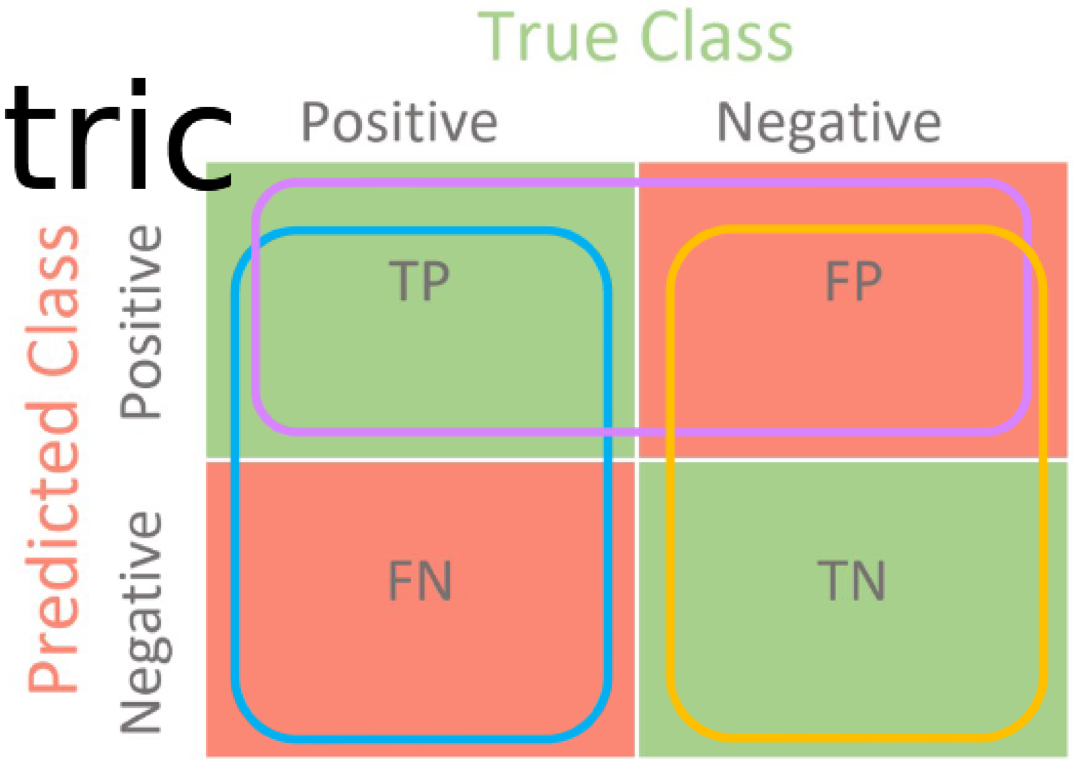
of all the predicted as dogs, what % are dogs

Recall/sensitivity

what % of dogs were predicted as dogs

Specificity

what % of not dogs were preidicted as not dogs



Dataset

- Iris Dataset
- 150 samples
- 3 Categories

Iris Versicolor Iris Setosa Iris Virginica

Feature Transformation

- StandardScaler() – Mean = 0 ,Stdv= 1 (normal distribution)
- Drop petal width (cm) – Multicollinearity
- Remove outliers– IQR shown through Box plot

Assumptions

1. Nominal Dependent Variable with Multiple Categories

- Assumption: The dependent variable must be categorical with three or more *unordered* classes.
- If violated:
  - If your dependent variable is ordinal (ordered categories), consider using ordinal logistic regression instead.
  - If you have only two classes, use binary logistic regression.

PASSED

```
target
0    50
1    50
2    50
Name: count, dtype: int64
```

2. Adequate Sample Size (Features x 10 – 20 x No.Class)

- **4 * 10 * 3 = 120, /we have 150/**
- Assumption: Enough cases per predictor variable.
- If violated:
  - Collect more data if possible.
  - Reduce the number of predictors (feature selection).
  - Use penalized regression methods to stabilize estimates.

PASSED

```
[106]:  X_reduced.shape

[106]:  (150, 3)
```

3. No Perfect Multicollinearity Among Independent Variables

- Assumption: Predictors should not be perfectly or highly correlated.
- Test using correlation matrix or Variance Inflation Factor (VIF) > 10
- If violated:
  - Remove or combine highly correlated variables.
  - Use dimensionality reduction techniques (e.g., PCA) or regularization methods (e.g., Lasso, Ridge regression).

PASSED

- VIF < 10

```
print(vif_data)
```

```
             Feature        VIF
0  sepal length (cm)   7.072722
1   sepal width (cm)   2.100872
2  petal length (cm)  31.261498
3   petal width (cm)  16.090175
```

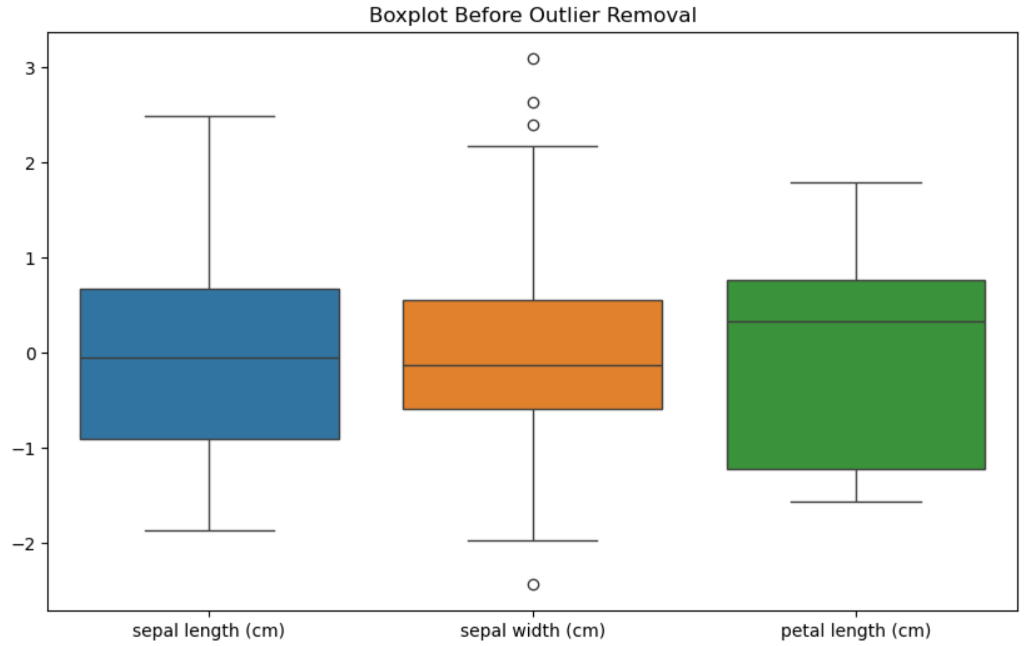
```
print(vif_data)
```

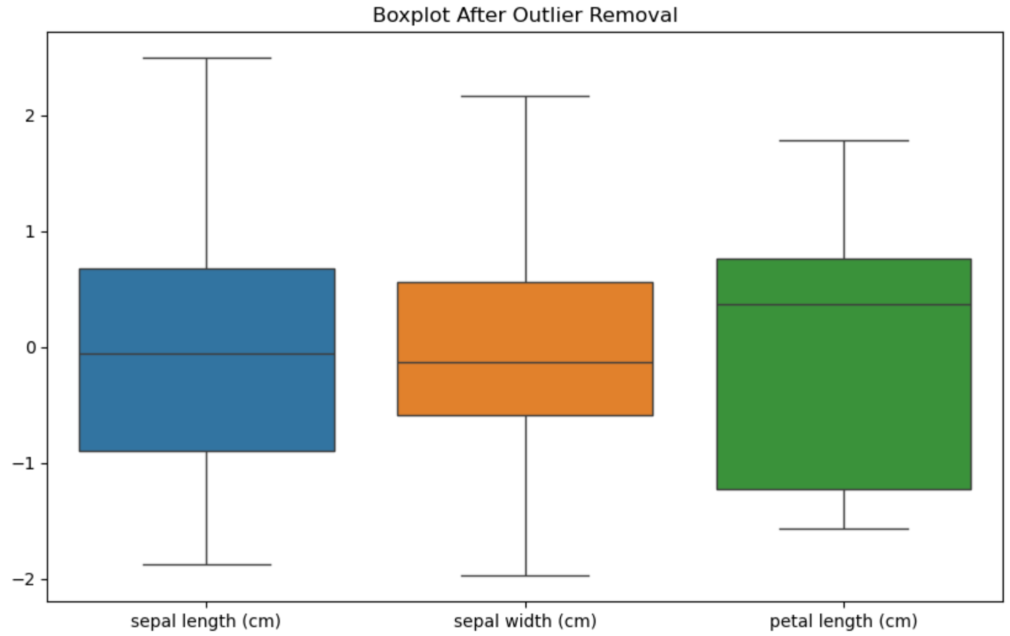
```
             Feature       VIF
0  sepal length (cm)  6.256954
1   sepal width (cm)  1.839639
2  petal length (cm)  7.557780
```

4. Absence of Outliers and High Leverage Points

- Assumption: No influential extreme data points distort the model.
- Test using Cook's distance 4/N
- If violated:
  - Detect outliers with residual plots, Cook's distance, or leverage statistics.
  - Investigate and clean or remove problematic points if justified.
  - Consider robust regression techniques.

PASSED

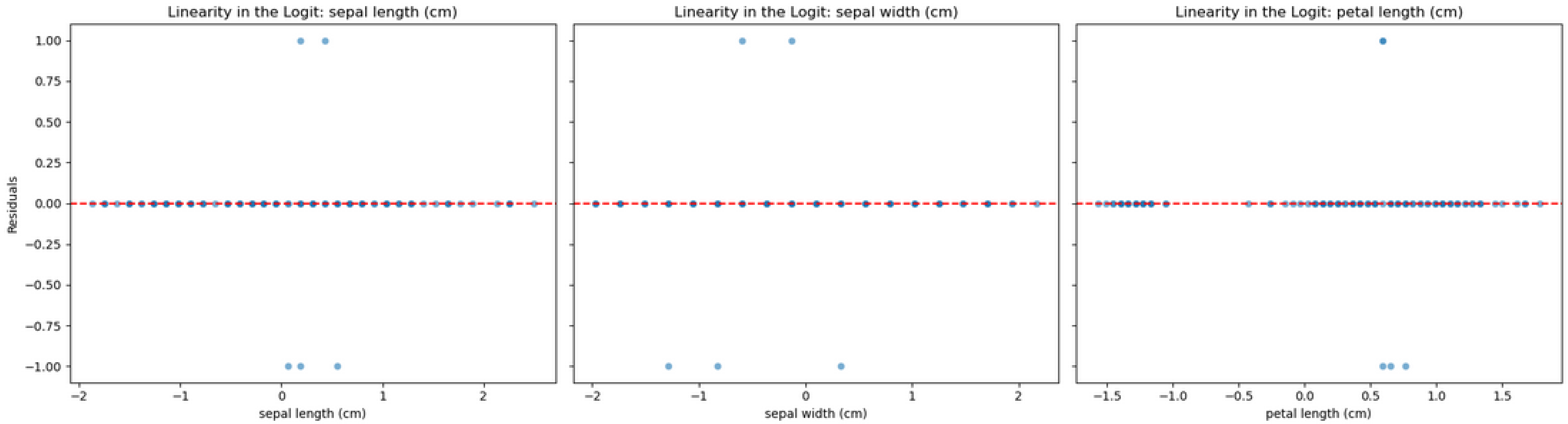

Boxplot Before Outlier Removal
3
2
1
0
−1
−2
sepal length (cm)
sepal width (cm)
petal length (cm)


Boxplot After Outlier Removal
2
1
0
−1
−2
sepal length (cm)
sepal width (cm)
petal length (cm)

5. Linearity in the Logits

- Assumption: Continuous predictors have a linear relationship with the log-odds.
- Test using Scatter plot, randomly scattered points around 0 with no discernible pattern (i.e., no curves, no clear fan shape, no systematic trends).
- If violated:
  - Use polynomial terms or splines to model non-linear relationships.
  - Transform variables (e.g., log, square root).
  - Use generalized additive models (GAMs) if many nonlinear relationships are expected.

PASSED

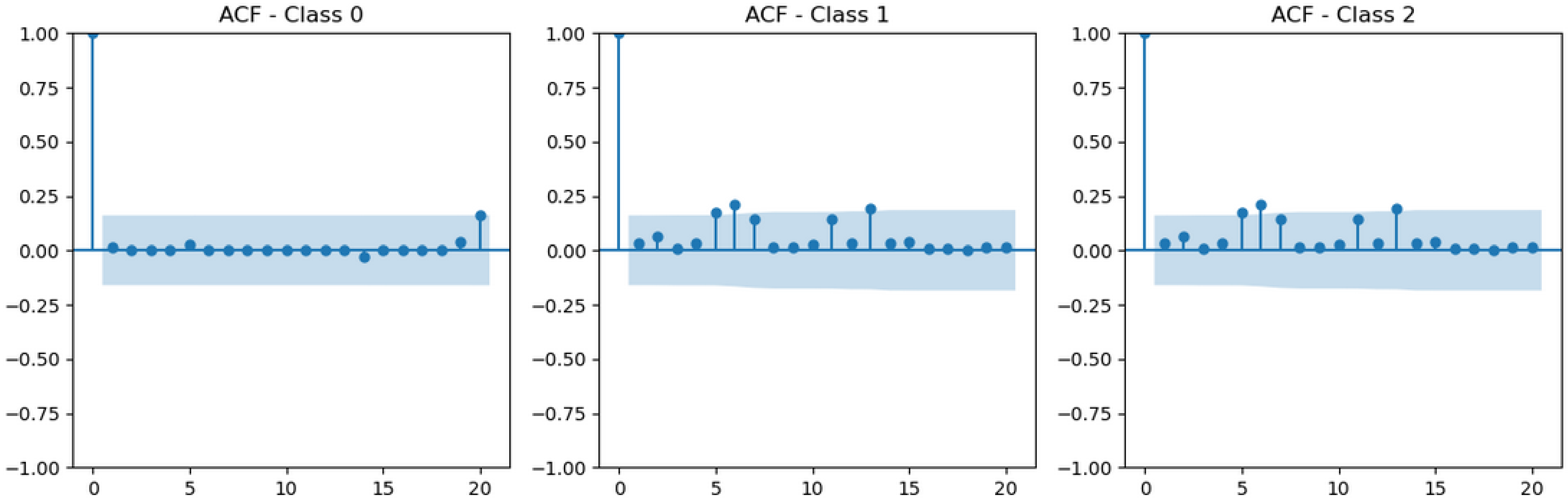


Linearity in the Logit: sepal length (cm)
Linearity in the Logit: sepal width (cm)
Linearity in the Logit: petal length (cm)
Residuals
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−2
−1
0
1
2
sepal length (cm)
sepal width (cm)
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
petal length (cm)

6. Independence of Observations

- Assumption: The outcome of one observation does not influence the outcome of another
- If violated:
  - If data are clustered or repeated measures (e.g., longitudinal data), use mixed-effects (multilevel) models or generalized estimating equations (GEE) to account for correlation within clusters.

PASSED


ACF - Class 0
ACF - Class 1
ACF - Class 2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20

- **7. Independence of Irrelevant Alternatives (IIA)**
- Assumption: Odds between any two outcomes are independent of other alternatives.
- Test for IIA with Hausman-McFadden or Small-Hsiao tests
- If violated:
  - Use nested logit models, multinomial probit models, or mixed logit models which relax the IIA assumption.

PASSED

```
> result_table <- data.frame(
  `Test` = "Hausman-McFadden",
  `Statistic (H)` = round(H[1,1], 3),
  `df` = df,
  `p-value` = round(p_value, 3)
)
result_table
              Test Statistic..H. df p.value
1 Hausman-McFadden        -5.566  3       1


> if (p_value > 0.05) {
  cat("IIA assumption holds. Multinomial logistic regression is appropriate.\n")
} else {
  cat("IIA assumption violated. Consider nested logit or alternatives.\n")
}
IIA assumption holds. Multinomial logistic regression is appropriate.
```

```
> library(nnet) #because mlogit is not available natively ☹

data(iris) #load data, sandboxed
iris$Species <- as.factor(iris$Species)


# Fit full model (3 classes)
full_model <- multinom(Species ~ Sepal.Length + Sepal.Width + Petal.Length, data = iris)


# Restricted model (drop virginica) or restrict whichever
iris_restricted <- subset(iris, Species != "virginica")
iris_restricted$Species <- factor(iris_restricted$Species)
restricted_model <- multinom(Species ~ Sepal.Length + Sepal.Width + Petal.Length, data = iris_restricted)


# extract coef and remove intercept
b_full_all <- coef(full_model)["versicolor", ]
b_full <- b_full_all[-1] # remove intercept


b_restrict_all <- coef(restricted_model)
b_restrict <- b_restrict_all[-1] # remove intercept
```

```
# extract 3x3 matrix
vcov_full_all <- vcov(full_model)
vcov_restrict_all <- vcov(restricted_model)


# Indices for versicolor (2nd row) predictors in 8x8 vcov:
# Rows/cols 5–7 = Sepal.Length, Sepal.Width, Petal.Length for versicolor
vcov_full <- vcov_full_all[5:7, 5:7]
vcov_restrict <- vcov_restrict_all[-1, -1] # remove intercept row and column


# compute hausman test
b_diff <- b_restrict - b_full
V_diff <- vcov_restrict - vcov_full


# Compute test only if shapes match
if (length(b_diff) == 3 && all(dim(V_diff) == c(3,3))) {
H <- t(b_diff) %*% solve(V_diff) %*% b_diff
df <- length(b_diff)
p_value <- pchisq(H, df = df, lower.tail = FALSE)


list(Hausman_statistic = H[1,1], df = df, p_value = p_value)
} else {
"Still dimension mismatch!"
}
```

Loss is calculated using cross-entropy loss

$$L = -1/m \sum_{i=1}^{m} \left(\sum_{k=1}^{C} \mathbb{I}(y_i = C_k) \log P(y_i = C_k | \mathbb{x}_i) \right)$$

Hyperparameter tuning

Grid Search / Random Search / Bayesian

- **Learning rate η**: Too high, and you risk overshooting; too low, and training might be slow.
- **Regularization strength**: Prevent overfitting by adding regularization terms like L1, L2 or elastic net.
- **Number of iterations or epochs**: How many times the model iterates through the training data.
- **Batch size** (for mini-batch gradient descent): How many samples are processed at once during each step.

Gradient Decent

- Updates weights and bias iteratively

$$w \coloneqq w - \eta \frac{\partial L}{\partial w_k}$$

$$b \coloneqq b - \eta \frac{\partial L}{\partial b_k}$$

- Where η is the learning rate and partial derivates are as follows

$$\frac{\partial L}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} (P(y_i = C_k | \mathbb{x}i) - \mathbb{I}(y_i = C_k))\mathbb{x}i$$

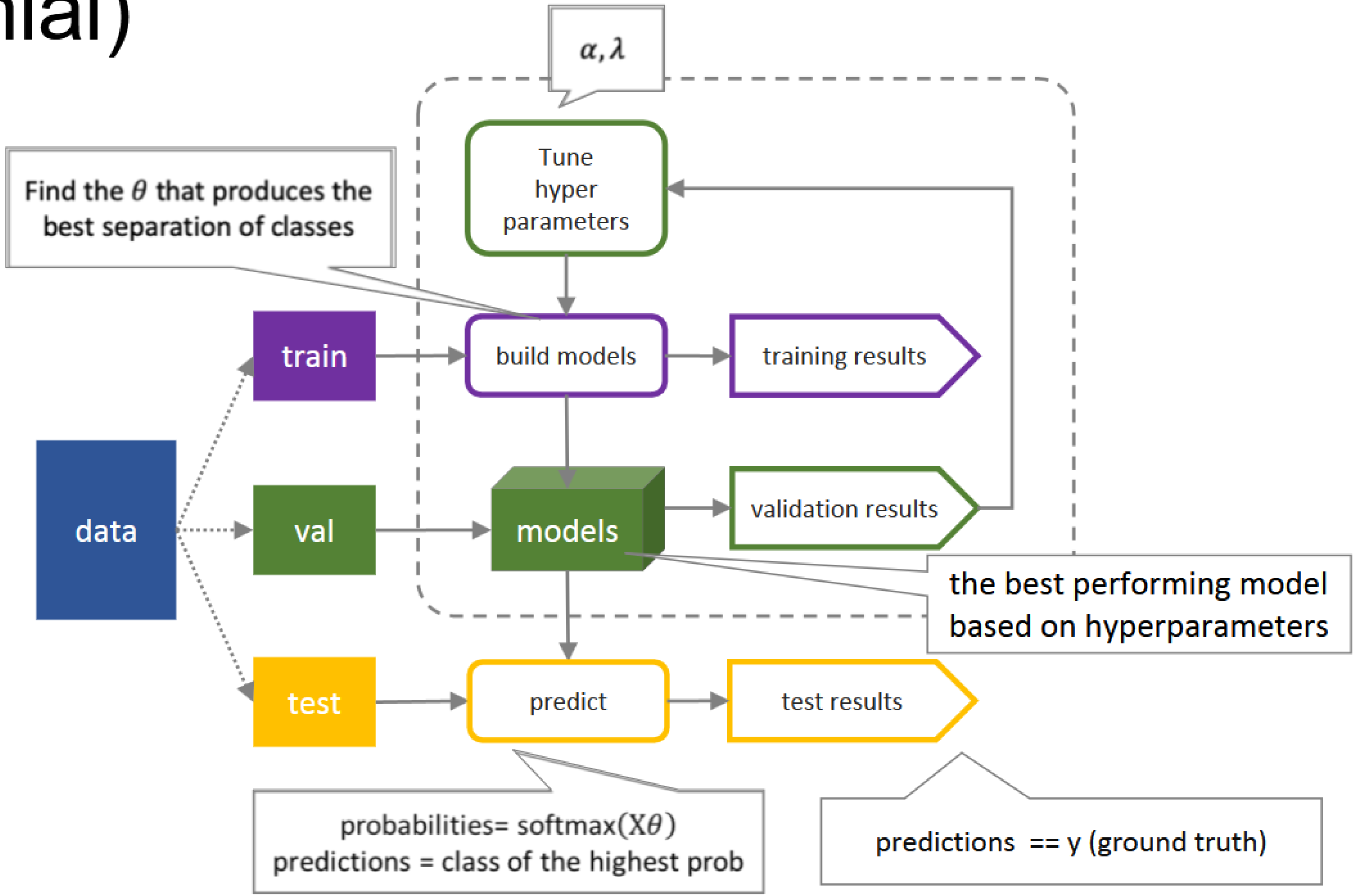
$$\frac{\partial L}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (P(y_i = C_k | \mathbb{x}i) - \mathbb{I}(y_i = C_k))$$

Types of Gradient Decent

- **Batch Gradient Descent**: Uses the entire dataset.
  - use with small dataset
- **Stochastic Gradient Descent (SGD)**: Uses a single training example.
  - Dataset > computer memory
- **Mini-batch Gradient Descent**: A compromise between the two.

We can validate the model again, repeat and change parameters if not satisfied

Pipeline Logistic regression (multinomial)



Credits to Dr. Thomas Tiam Lee

Python Demo

```python
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, log_loss
import pandas as pd
import numpy as np
```

```python
df = pd.read_csv('./iris.csv')
```

```python
df = df.drop(['Unnamed: 0', 'petal_width'], axis=1) # drop col 1 (index) drop petal width due to multicolinearity
```

```python
le = LabelEncoder()
df['species_encoded'] = le.fit_transform(df['species'])
```

Scale and Split dataset

```python
X = df.drop(['species', 'species_encoded'], axis=1)
y = df['species_encoded']

# Standardize
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split: 80/10/10
X_temp, X_test, y_temp, y_test = train_test_split(X_scaled, y, test_size=0.1, random_state=25, stratify=y)
X_train, X_val, y_train, y_val = train_test_split(X_temp, y_temp, test_size=0.1111, random_state=25, stratify=y_temp)
```

Grid Search

```
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2', 'elasticnet'],
    'solver': ['saga', 'lbfgs', 'newton-cg'],
    'l1_ratio': [0.1, 0.5, 0.9] # add this if elasticnet is present
    'max_iter': [1000, 5000, 10000],
}
grid = GridSearchCV(LogisticRegression(), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)
best_model = grid.best_estimator_
```

```
best_model
```

```
LogisticRegression
LogisticRegression(C=10, max_iter=5000, solver='saga')
```

Hyperparameter	Description	Example Values
C	Regularization strength.	0.01, 1.0, 100
penalty	Type of regularization.	'l1', 'l2', 'elasticnet', 'none'
solver	Optimization algorithm to use.	'liblinear', 'saga', 'lbfgs', 'newton-cg'
max_iter	Maximum number of iterations for optimization.	100, 5000
tol	Tolerance for stopping criteria.	1e-4, 1e-6
multi_class	Strategy to handle multiclass classification.	'auto', 'ovr' (one-vs-rest), 'multinomial'
fit_intercept	Whether to include an intercept in the model.	True, False
intercept_scaling	Scaling of the intercept term (used for liblinear solver).	1.0, 10.0
class_weight	Weights for each class (useful for imbalanced datasets).	None, {'class_0': 1, 'class_1': 10}, balanced
warm_start	Whether to reuse the solution of the previous call to fit.	True, False
n_jobs	Number of CPU cores to use during computation.	-1 (use all cores), 2, 4
random_state	Seed for the random number generator, for reproducibility.	42, None
verbose	Verbosity level. Controls logging output.	0, 1, 2
max_fun	Maximum number of function evaluations for solvers like 'lbfgs'.	150, 500
l1_ratio	Used when penalty='elasticnet', controls the mix between L1 and L2 regularization.	0.2, 0.5, 0.8

Prediction and Store Metrics

```
y_test_pred = best_model.predict(X_test)
y_test_pred_proba = best_model.predict_proba(X_test)


test_accuracy = accuracy_score(y_test, y_test_pred)
test_log_loss = log_loss(y_test, y_test_pred_proba)
test_conf_matrix = pd.DataFrame(
    confusion_matrix(y_test, y_test_pred),
    index=le.classes_,
    columns=[f"Predicted {label}" for label in le.classes_]
)

{
    "Test Accuracy": test_accuracy,
    "Test Log Loss": test_log_loss,
    "Test Confusion Matrix": test_conf_matrix
}
```

```
{'Test Accuracy': 0.9333333333333333,
 'Test Log Loss': 0.10076315561858003,
 'Test Confusion Matrix':                  Predicted Iris-setosa  Predicted Iris-versicolor  \
 Iris-setosa                         5                          0
 Iris-versicolor                     0                          4
 Iris-virginica                      0                          0

                  Predicted Iris-virginica
 Iris-setosa                            0
 Iris-versicolor                        1
 Iris-virginica                         5  }
```

```
test_conf_matrix
```

	Predicted Iris-setosa	Predicted Iris-versicolor	Predicted Iris-virginica
Iris-setosa	5	0	0
Iris-versicolor	0	4	1
Iris-virginica	0	0	5

Python

```
coef_df
```

	sepal_length	sepal_width	petal_length	Intercept
Iris-setosa	-2.141514	1.813595	-5.249076	-0.337984
Iris-versicolor	1.728591	-0.681104	-2.638715	3.292326
Iris-virginica	0.412922	-1.132491	7.887791	-2.954342

JASP

Regression Coefficients

	Coefficient (β)	Standard Error	z	p
(Intercept) : Iris-setosa	−1.609	968.121	−0.002	0.999
(Intercept) : Iris-versicolor	12.357	4.137	2.987	0.003
sepal_length : Iris-setosa	11.475	1885.828	0.006	0.995
sepal_length : Iris-versicolor	3.333	1.659	2.010	0.044
sepal_width : Iris-setosa	2.897	281.357	0.010	0.992
sepal_width : Iris-versicolor	−0.492	1.096	−0.450	0.653
petal_length : Iris-setosa	−55.336	1663.012	−0.033	0.973
petal_length : Iris-versicolor	−22.847	7.902	−2.891	0.004

Note. The regression coefficients for numeric features are standardized.

Sample Prediction

```python
# sample prediction should be class Iris setosa
sample_raw_df = pd.DataFrame([[5.1, 3.5, 1.4]], columns=X.columns)
sample_scaled = scaler.transform(sample_raw_df)

# logits
logits = best_model.coef_ @ sample_scaled.T + best_model.intercept_.reshape(-1, 1)
logits = logits.flatten()

# Softmax
exp_logits = np.exp(logits)
probs = exp_logits / np.sum(exp_logits)
```

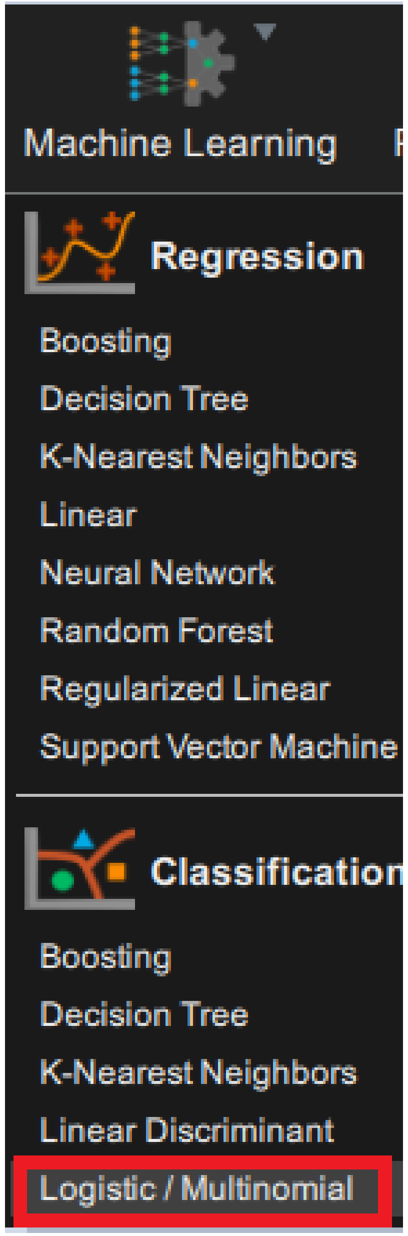
```python
# echo results
prediction = {
    "Raw Input": sample_raw_df.values.flatten().tolist(),
    "Standardized Input": sample_scaled.flatten().tolist(),
    "Logits": logits.tolist(),
    "Exp(Logits)": exp_logits.tolist(),
    "Probabilities": probs.tolist(),
    "Predicted Class Index": int(np.argmax(probs)),
    "Predicted Class Label": le.classes_[int(np.argmax(probs))]
}

{
    "Confusion Matrix": conf_matrix_df,
    "Coefficients and Intercepts": coef_df,
    "Prediction for [5.1, 3.5, 1.4]": prediction,
    "Validation Accuracy": accuracy_score(y_val, y_pred),
    "Log Loss": log_loss(y_val, y_pred_proba)
}
```

Results

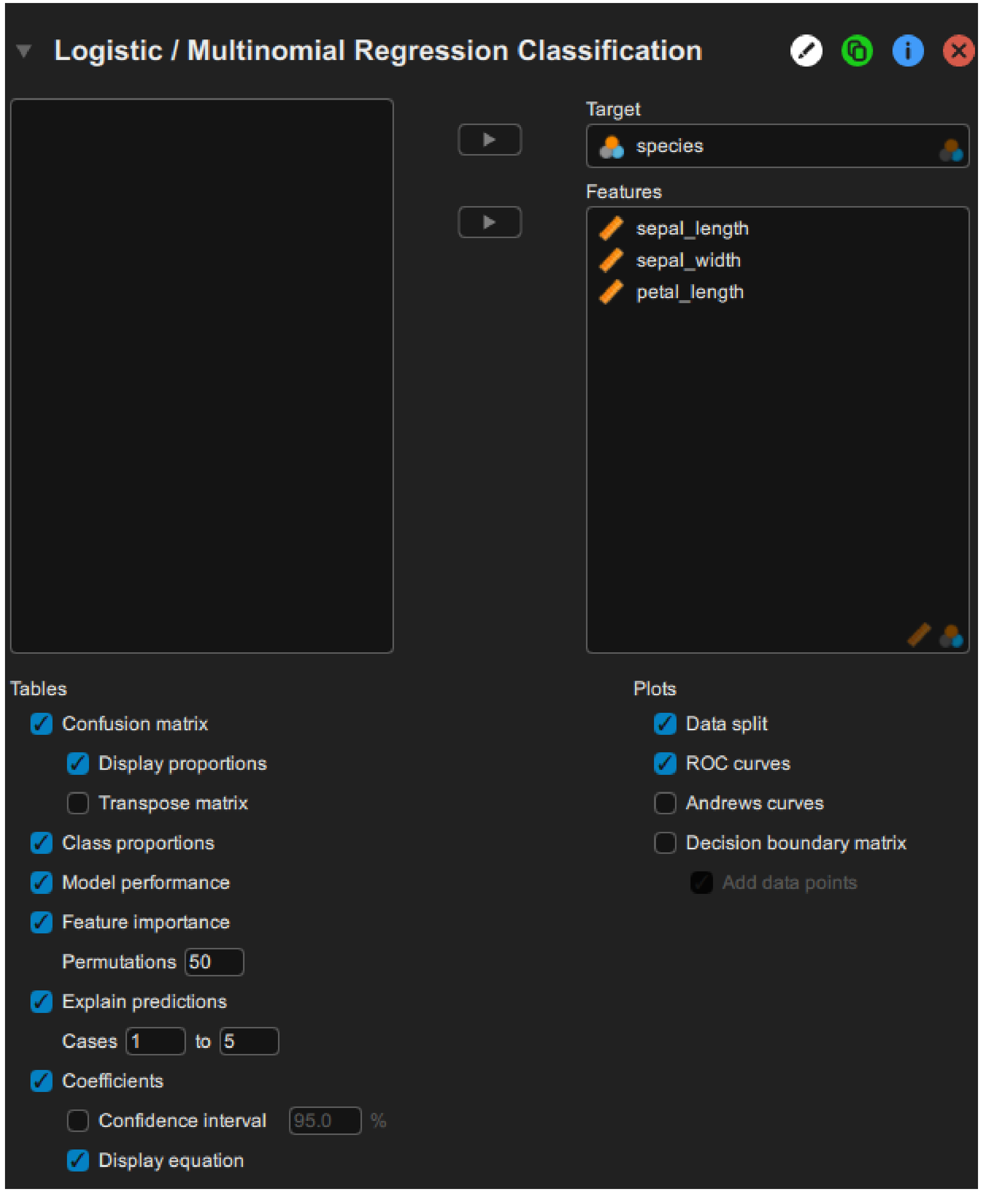
```
pd.set_option('display.precision', 16)
pd.set_option('display.max_colwidth', None)

prediction_df = pd.DataFrame(list(prediction.items()),columns=['Metric', 'Value'])
prediction_df
```

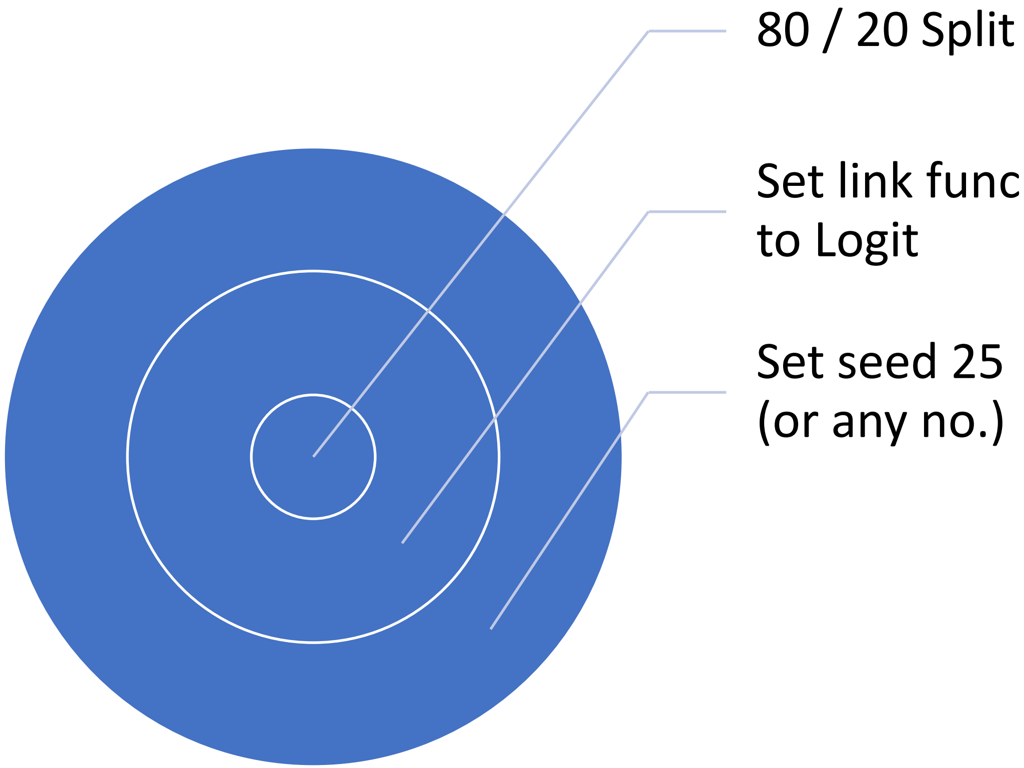
	Metric	Value
0	Raw Input	[5.1, 3.5, 1.4]
1	Standardized Input	[-0.9006811702978088, 1.0320572244889565, -1.3412724047598314]
2	Logits	[10.503011718427075, 4.571712975218585, -15.074724693645765]
3	Exp(Logits)	[36425.03960724319, 96.70962916522113, 2.838770276908608e-07]
4	Probabilities	[0.9973519989740267, 0.0026480010182004074, 7.772821226362788e-12]
5	Predicted Class Index	0
6	Predicted Class Label	Iris-setosa

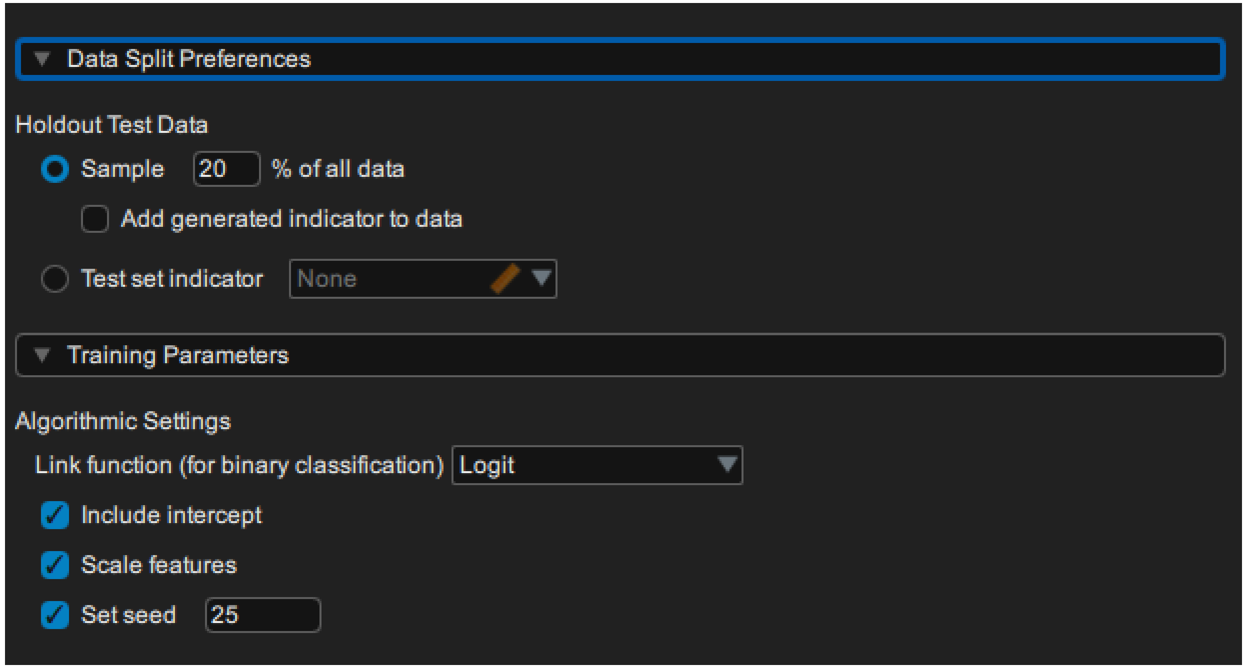
JASP DEMO


Machine Learning
Regression
Boosting
Decision Tree
K-Nearest Neighbors
Linear
Neural Network
Random Forest
Regularized Linear
Support Vector Machine
Classification
Boosting
Decision Tree
K-Nearest Neighbors
Linear Discriminant
Logistic / Multinomial

Select the following




80 / 20 Split
Set link func to Logit
Set seed 25 (or any no.)


Data Split Preferences
Holdout Test Data
Sample
20
% of all data
Add generated indicator to data
Test set indicator
None
Training Parameters
Algorithmic Settings
Link function (for binary classification)
Logit
Include intercept
Scale features
Set seed
25

Results

100% of actual *Setosa* observations were correctly classified

Versicolor – 83% accuracy. 10 correctly classified 2 misclassified as virginica

100% of actual *Viginica* observations were correctly classified

Confusion Matrix

		Predicted		
		Iris-setosa	Iris-versicolor	Iris-virginica
Observed	Iris-setosa	10	0	0
	Iris-versicolor	0	10	2
	Iris-virginica	0	0	8

Model Summary: Multinomial Regression Classification

Family	Link	n(Train)	n(Test)	Test Accuracy
Multinomial	Logit	120	30	0.933

Metrics

Iris-Setosa – Perfect Prediction

Iris-Versicolor

- Lower recall (0.833) - missed some true Versicolor cases
- Specificity (1.000) — no falsely prediction.

Iris-Virginica

- High recall (1.000) — all Virginica instances were correctly identified.
- But lower precision (0.800) — the model sometimes predicts Virginica when it's actually Versicolor (false positives).
- Specificity (0.909) indicates some non-Virginicas were wrongly predicted as Virginica.

Model Performance Metrics ▼

	Iris-setosa	Iris-versicolor	Iris-virginica	Average / Total
Support	10	12	8	30
Accuracy	1.000	0.933	0.933	0.956
Precision (Positive Predictive Value)	1.000	1.000	0.800	0.947
Recall (True Positive Rate)	1.000	0.833	1.000	0.933
False Positive Rate	0.000	0.000	0.091	0.030
False Discovery Rate	0.000	0.000	0.200	0.067
F1 Score	1.000	0.909	0.889	0.934
Matthews Correlation Coefficient	1.000	0.866	0.853	0.906
Area Under Curve (AUC)	1.000	0.569	0.955	0.841
Negative Predictive Value	1.000	0.900	1.000	0.967
True Negative Rate	1.000	1.000	0.909	0.970
False Negative Rate	0.000	0.167	0.000	0.056
False Omission Rate	0.000	0.100	0.000	0.033
Threat Score	∞	5.000	2.000	∞
Statistical Parity	0.333	0.333	0.333	1.000

Note. All metrics are calculated for every class against all other classes.

The model relies most heavily and significantly on:

- Petal Length (negatively) and Sepal Length (positively) to classify Versicolor vs Virginica

- Predicting Setosa may not rely on coefficients in the same way due to perfect classification — the model doesn’t need to "weigh" predictors statistically when there's no error.

- Sepal Width adds little to the model’s decision-making power.

Regression Coefficients

	Coefficient (β)	Standard Error	z	p
(Intercept) : Iris-setosa	−1.609	968.121	−0.002	0.999
(Intercept) : Iris-versicolor	12.357	4.137	2.987	0.003
sepal_length : Iris-setosa	11.475	1885.828	0.006	0.995
sepal_length : Iris-versicolor	3.333	1.659	2.010	0.044
sepal_width : Iris-setosa	2.897	281.357	0.010	0.992
sepal_width : Iris-versicolor	−0.492	1.096	−0.450	0.653
petal_length : Iris-setosa	−55.336	1663.012	−0.033	0.973
petal_length : Iris-versicolor	−22.847	7.902	−2.891	0.004

Note. The regression coefficients for numeric features are standardized.

Odds Ratio JASP

e^{β} **= Odds Ratio**

An increase with X feature will result in ± X Odds Ratio

petal_length : Iris-versicolor = -22.847

Odds Ratio= $e^{\beta} = e^{-22.847} \approx 1.2\times 10^{-10}$

1-unit increase in petal length **decreases** the odds of the class being *Iris-versicolor* by

0.00000000011958425

The longer the petal the further it is to be iris-versicolor

Predictor	Coefficient (β)	Std. Error	z	p	Odds Ratio (e^β)
(Intercept) : Iris-setosa	-1.609	968.121	-0.002	0.999	0.200
(Intercept) : Iris-versicolor	12.357	4.137	2.987	0.003	233,715.405
sepal_length : Iris-setosa	11.475	1885.828	0.006	0.995	96,075.185
sepal_length : Iris-versicolor	3.333	1.659	2.010	0.044	28.012
sepal_width : Iris-setosa	2.897	281.357	0.010	0.992	18.121
sepal_width : Iris-versicolor	-0.492	1.096	-0.450	0.653	0.611
petal_length : Iris-setosa	-55.336	1663.012	-0.033	0.973	~0.000 (very small)
petal_length : Iris-versicolor	-22.847	7.902	-2.891	0.004	~0.000 (very small)

Odds Ratio Python

1 unit increase in petal length **reduces** the odds of being classified as Iris-versicolor by about 92.85%

1−0.0715=0.93% **less likely** to be *Iris-versicolor*

Class	Feature	Coefficient (β)	Odds Ratio (e^β)
Iris-setosa	sepal_length	-2.141514	0.1179
	sepal_width	1.813595	6.1325
	petal_length	-5.249076	0.0053
	Intercept	-0.337984	0.7131
Iris-versicolor	sepal_length	1.728591	5.6342
	sepal_width	-0.681104	0.5063
	petal_length	-2.638715	0.0715
	Intercept	3.292326	26.9084
Iris-virginica	sepal_length	0.412922	1.5110
	sepal_width	-1.132491	0.3224
	petal_length	7.887791	2,662.6317
	Intercept	-2.954342	0.0522

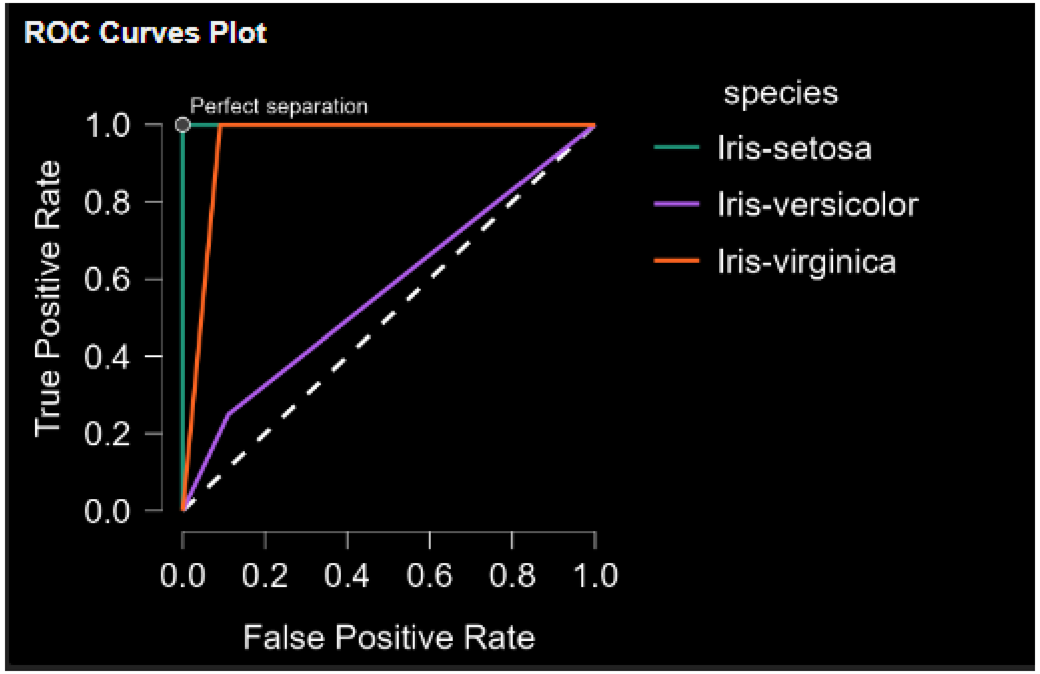
Odds Ratio	How to Interpret	Example
< 1	1 - OR → % decrease in odds	OR = 0.07 means 93% decrease in odds (Less 1, Reduce by)**[L1R]**
> 1	OR × increase in odds	OR = 2 means 2× ↑ in odds (Greater 1, Multiply by) **[G1M]**
= 1	No effect	OR = 1 → odds unchanged

The model has excellent ROC performance for Setosa and Virginica.

Versicolor is harder to distinguish:

- Lower recall
- Some misclassifications in confusion matrix

The ROC curves visually confirm the findings from accuracy, precision, and the coefficient significance.



- **Reference category:** *Iris-virginica*
  Multinomial logistic regression computes logits relative to this reference

- Setosa VS Virginica
  - $\log\left(\frac{P(virginica)}{P(setosa)}\right) = -1.609 + 11.475(sepal\_length)2.897(sepal\_width) - 55.336(petal\_length)$

- Versicolor vs Virginica
  - $\log\left(\frac{P(virginica)}{P(versicolor)}\right) = 12.357 + 3.333(sepal\_length) + 0.492(sepal\_width) - 22.847(petal\_length)$

Sample

- Unknown Iris (since our dataset is scaled, we also scale these)
  - Sepal length = 5.1 -> −0.9007
  - Sepal Width = 3.5 -> 1.0321
  - Petal length = 1.4 -> −1.3413
- Setosa vs Virginica **log-odds**:

$$\log\left(\frac{P(Setosa)}{P(Virginica)}\right) = -1.609 + 11.475(-0.9007\ ) + 2.897(1.0321) - 55.336(-1.3413)$$

$$\log\left(\frac{P(Setosa)}{P(Virginica)}\right) = -1.609 + 10.5743 + 2.9899937 - 74.2221768 \approx 65.27$$

- $\log\left(\frac{P(Versicolor)}{P(Virginica)}\right) =$
12.357+3.333(0.9007)−0.492(1.0321)−22.847(1.3413)

- $\log\left(\frac{P(Versicolor)}{P(Virginica)}\right) =$ 12.357−3.002+(−0.507)+30.644 ≈ 39.49

Exponentiate

$$odds_{setosa} = e^{-65.27} \approx 4.5\times10^{-29}$$

$$odds_{versicolor} = e^{-39.49} \approx 7.1\times10^{-18}$$

$$odds_{virginica} = e^{0} = 1(Ref\ Class)$$

SOFTMAX

$$Total\ odds = 4.5{\times}10^{-29} + 7.1{\times}10^{-18} + 1 \approx 1$$

$$P(Setosa) = \frac{4.5{\times}10^{-29}}{1} \approx 4.5{\times}10^{-29}$$

$$P(Versicolor) = \frac{7.1{\times}10^{-18}}{1} \approx 7.1{\times}10^{-18}$$

$$P(Virginica) = \frac{1}{1} \approx \mathbf{1}$$

Iris Virginica

On Average

- Iris Setosa
  - Sepal Length (5.0)
  - Sepal Width (3.4)
  - Petal Length (1.5)
  - Petal Width (0.2)
- Iris Versicolor
  - Sepal Length (5.9)
  - Sepal Width (2.8)
  - Petal Length (4.3)
  - Petal Width (1.3)
- Iris Verginica
  - Sepal Length (6.5)
  - Sepal Width (3)
  - Petal Length (5.5)
  - Petal Width (2.0)

Sepal length = 5.1
Sepal Width = 3.5
Petal length = 1.4

They differ because……

- The JASP model has
  - Coefficients with extremely large (e.g. −55.336)
  - No regularization (L1,L2,Elastic net)
  - No Solver/ uses default gradient decent if any

JASP isn’t bad it just leans more on Virginica Class due to factors mentioned fin

Linkedin

github.com/PedGit025