Simple Linear Regression

Simple Linear Regression = **predict a number using one input with a straight line**.

Key ideas:

- Input (X): The value that affects the output
- Output (Y): The value we want to predict
- The model finds the best straight line:

Y=mX+b

Where:

- Y = predicted output
- X = input
- m = slope of the line (how much Y changes if X changes)
- b = intercept (where the line crosses Y-axis)

```
# Create model

model = LinearRegression()

# Train / fit model

model.fit(X, Y)
```

Multi Linear Regression

Multi Linear Regression = **predict a number using multiple inputs with a flat plane**.

Key ideas:

- Inputs (X_1, X_2, X_3…): Multiple values that affect the output
- Output (Y): The value we want to predict
- The model finds the best flat plane:

Y=b+m1X1+m2X2+m3X3+…

Where:

- Y = predicted output
- X_1, X_2, X_3… = input values
- m_1, m_2, m_3… = slopes (how much Y changes if each input changes)
- b = intercept (where the plane crosses the Y-axis)

# Create model

model = LinearRegression()

# Train / fit model

model.fit(X, Y)

hyper parameter combination

fit_intercept	normalize / scaler	positive	n_jobs
TRUE	StandardScaler(True)	FALSE	None
TRUE	StandardScaler(False)	FALSE	None
TRUE	StandardScaler(True)	TRUE	None
FALSE	StandardScaler(True)	FALSE	None

Support Vector Regression (SVR)

SVR = **predicting numbers using a line or curve that fits the data, ignoring tiny errors**, and you can make it flexible with different kernels.

Key ideas:

- You have **inputs** → X (one or many things that affect the output)
- You have **output** → Y (what you want to predict)
- The model tries to **fit a line or curve** that predicts Y as close as possible
- You can choose **linear or non-linear curve** using a **kernel**:

- linear → straight line
- poly → polynomial curve
- rbf → flexible curved line

Extra points:

- **C** → How much the model tries to fit all the points perfectly (bigger C → fits tighter)
- **epsilon** → A small range where errors are ignored (helps avoid overfitting)

# Create model

# Use RBF kernel, C=100, epsilon=0.1

model = SVR(kernel='rbf', C=100, epsilon=0.1)

# Train / fit model

model.fit(X, Y)

hyper parameter combination

hyper parameter

Kernel	C	gamma	epsilon	degree	coef0
rbf	1	0.1	0.1	-	-
rbf	10	0.01	0.05	-	-
linear	1	-	0.1	-	-
poly	1	0.1	0.1	3	0
poly	10	0.01	0.05	2	0.1
sigmoid	1	0.1	0.1	-	0.1

<u>**Decision Tree Regression**</u>

Decision Tree Regression = predicting numbers by repeatedly splitting data into groups like a tree until the prediction is accurate.

Key ideas:

- You have inputs → X_1, X_2… (features that affect the output)
- You have output → Y (what you want to predict)
- The model splits the data at each step based on the input values to make the output more similar within each group

- The final prediction is the average value of the outputs in the leaf node (the end of the branch)

Extra points:

- max_depth → How deep the tree can go (deeper → more complex)
- min_samples_split → Minimum samples needed to split a node
- min_samples_leaf → Minimum samples in a leaf node

# Create model

model = DecisionTreeRegressor(max_depth=3)

# Train / fit model

model.fit(X, Y)

Decision Tree Regression Hyperparameter Combinations

max_depth	**min_samples_split**	**min_samples_leaf**	**max_features**	**criterion**	**splitter**
None	2	1	None	squared_error	best
5	2	1	sqrt	squared_error	best
7	5	2	log2	absolute_error	best
10	10	4	None	friedman_mse	random
3	2	1	sqrt	squared_error	best

Random Forest Regression

Random Forest Regression = many decision trees working together to predict a number more accurately.

Key ideas:

- You have **inputs** → X_1, X_2… (features that affect the output)
- You have **output** → Y (what you want to predict)
- The model builds **many decision trees** using **random samples of the data and features**

- The **final prediction** is the **average** of all tree predictions

Extra points:

- **n_estimators** → Number of trees in the forest
- **max_depth** → Maximum depth of each tree
- **min_samples_split** → Minimum samples needed to split a node
- **min_samples_leaf** → Minimum samples in a leaf node
- **max_features** → Maximum features considered for each split

# Create model

model = RandomForestRegressor(n_estimators=100, max_depth=3, random_state=42)

# Train / fit model

model.fit(X, Y)

Random forest Regression Hyperparameter Combinations

n_estimators	**max_depth**	**min_samples_split**	**min_samples_leaf**	**max_features**	**bootstrap**	**criterion**
100	None	2	1	sqrt	TRUE	squared_error
100	10	2	1	log2	TRUE	squared_error
200	15	5	2	auto	TRUE	absolute_error
200	None	2	2	sqrt	FALSE	squared_error
500	10	5	1	log2	TRUE	friedman_mse
500	15	10	4	auto	TRUE	squared_error

<u>Boosting Algorithm</u>

Boosting builds a strong model by combining many simple models, learning from each model's mistakes

Adaboost -AdaBoost builds a strong model by fixing the mistakes of many small models step by step

Gradient Boosting - Gradient Boosting improves predictions step by step. Each new small model fixes the mistakes of the models before it
XGBoost- XGBoost makes predictions stronger by combining many small models, fixing errors step by step, and doing it very efficiently

adaboost Hyperparameter Combinations

Base Estimator	**n_estimators**	**Learning Rate**	**Loss**
Tree(depth=1)	50	0.1	linear
Tree(depth=1)	100	0.1	linear
Tree(depth=2)	100	0.05	square
Tree(depth=2)	200	0.1	linear
Tree(depth=3)	200	0.1	exponential
Tree(depth=3)	500	0.05	square

gradient Hyperparameter Combinations

n_estimators	**learning_rate**	**max_depth**	**min_samples_split**	**min_samples_leaf**	**subsample**	**max_features**	**loss**
100	0.1	3	2	1	1	None	squared_error
100	0.05	3	2	2	0.8	sqrt	squared_error
200	0.1	5	5	2	1	log2	absolute_error
200	0.05	5	2	1	0.8	sqrt	huber
500	0.01	7	10	4	0.6	None	squared_error

xg boost
Hyperparameter
Combinations

n_estimators	learning_rate	max_depth	min_child_weight	subsample	colsample_bytree	gamma	reg_alpha	reg_lambda	objective
100	0.1	3	1	1	1	0	0	1	reg:squarederror
200	0.05	5	3	0.8	0.8	0.1	0.01	1.5	reg:absoluteerror
200	0.1	7	5	0.8	0.8	0.1	0.01	2	reg:squarederror
500	0.01	10	1	0.6	0.6	0.5	0.1	2	reg:gamma
500	0.05	7	3	0.8	1	0.1	0.01	1.5	reg:squarederror