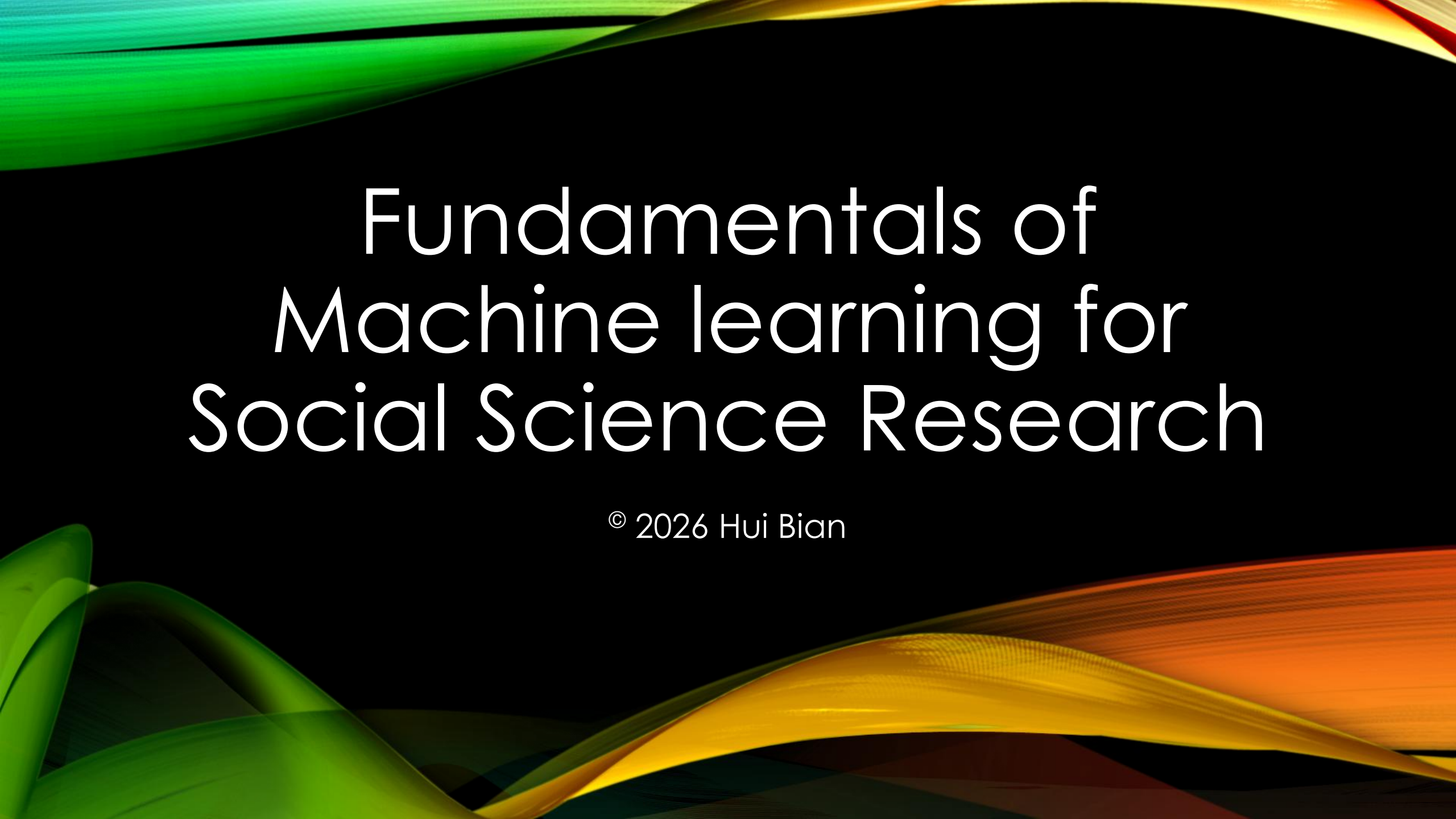




Fundamentals of Machine learning for Social Science Research

© 2026 Hui Bian



MACHINE LEARNING

- Software used is **Python** (Spyder from Anaconda).
- Main package used in Python is **Scikit-learn**.

MACHINE LEARNING

- What is **machine learning** (ML)?
 - IBM: “Machine learning is the subset of artificial intelligence (AI) focused on **algorithms** that can “learn” the patterns of **training** data and, subsequently, make accurate **inferences** about **new** data.”
 - Machine learning algorithms analyze the data, recognize underlying patterns, and make accurate predictions.
 - For example: spam detection, image recognition, etc.



SOCIAL SCIENCE RESEARCH

- Human and behavioral data are often messy, non-linear, and clustered!
- Machine learning provides good solutions to dealing with complex relationships.

SOCIAL SCIENCE RESEARCH

- Why machine learning (ML) instead of traditional statistics?
 - ML excels in analyzing **massive** text and image data.
 - ML excels in discovering complex and **non-linear** patterns: **random forests**, **gradient boosters**, **neural networks**, etc., can learn non-linearities, thresholds, and interactions directly from the data.

SOCIAL SCIENCE RESEARCH

- Why machine learning instead of traditional statistics?
 - ML can handle **high-dimensional** data. It is easy to deal with thousand variables.
 - ML can generate fresh data-driven theories for human researchers to test. For example, **decision trees**.

SOCIAL SCIENCE RESEARCH

- Machine learning, especially **non-parametric** ones like Decision trees, KNN, Random Forests, Support Vector Machines, or Neural Networks. They **do not care about the** assumptions in traditional statistical modeling.
- They are designed to find patterns in data that is highly **non-linear, correlated, and messy**.
- However, when we ask which predictor is the most important one, **multicollinearity** matters.

SOCIAL SCIENCE RESEARCH

- Linear based models: assumptions

Requirement	Linear Reg	Logistic Reg	SVM	Neural Net	KNN
Feature Scaling	✓ Helpful	✓ Helpful	✓ Critical	✓ Critical	✓ Critical
No Missing Values	✓ Required	✓ Required	✓ Required	✓ Required	✓ Required
Independence	✓ Required	✓ Required	✓ Required	✓ Required	✓ Required
Multicollinearity	✗ Problem	● Manageable	● Manageable	● Manageable	✗ Not an issue
Linearity	✗ Required	✗ Not required	✗ Not required	✗ Not required	✗ Not required
Normality	✗ Required	✗ Not required	✗ Not required	✗ Not required	✗ Not required
Homoscedasticity	✗ Required	✗ Not required	✗ Not required	✗ Not required	✗ Not required

SOCIAL SCIENCE RESEARCH

- Applications of machine learning in social science research
 - **Text/image analysis**: discover patterns and structures: automating coding task.
 - **Predictions and classifications**: predict outcomes and classify cases.
 - **Dimensionality deduction** and pattern discovery.
 - **Theory testing**.

SOCIAL SCIENCE RESEARCH

- Interpretability vs. Accuracy
 - In traditional statistical modeling, for example, we want to see the relationship between a predictor and our outcome and focus on if this relationship is significant.
 - Machine learning focuses on the **accuracy** of the model we propose. The algorithm doesn't care **why** a pattern exists.
 - It only cares if the pattern is reliable enough to make accurate predictions on new, **unseen** data.

SOCIAL SCIENCE RESEARCH

- Interpretability vs. Accuracy
 - However, we still can get **feature** importance results from machine learning.
 - We use SHAP (SHapley Additive exPlanations, a Python package) to forcefully extract which features mattered most.

SOCIAL SCIENCE RESEARCH

- **Model specification**: choosing what goes into the model and how it is structured is just as vital in ML as it is in traditional statistical modeling.
- An algorithm is never a one-size-fits-all solution. It is **our** job to choose the algorithm. However, we can test several models or use **ensemble method** (it is the strategy of combining multiple models to make a prediction, rather than relying on a single model).

SOCIAL SCIENCE RESEARCH

- Some Python packages are useful for the social science research.
 - EconML
 - LLM (Large Language Model)
 - Emergent Theme Discovery (Topic Modeling)

COMPUTATIONAL POWER AND SAMPLE SIZE

- Machine learning requires computational power and **enough** data, especially when using **deep learning (e.g., neural networks)**.
- **Sample size**: There is no gold standard. It depends on the algorithm. However, large sample is better.

GOALS OF MACHINE LEARNING

- Main goal: Build models that can **generalize** well to **unseen data**.
 - Building a model
 - Evaluate a model
 - Apply a model
 - Maintain a model
- In the real world, we can use this model to solve some problems such as spam detection, screening students (operational utility).

TASKS OF MACHINE LEARNING

- **Regression**: Predict a continuous variable.
- **Classification**: Predicts a categorical variable.
- **Clustering**: Groups data points based on similarity.
- **Anomaly detection**: Identifies rare items, events, or observations.



TERMS

- **Labels/targets:** Dependent variables
- **Features/attributes:** Independent variables or predictors.

TERMS

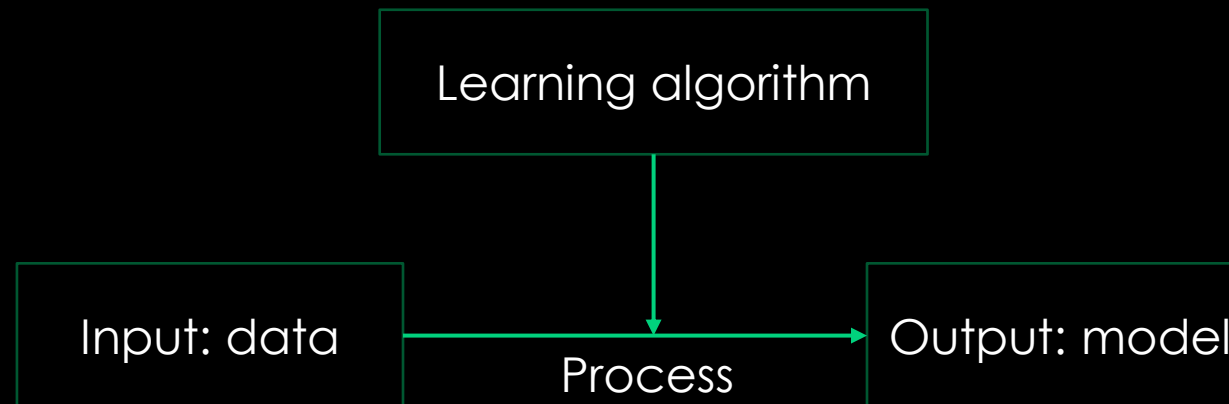
- **Black box:** A "black box" is any system where we can see the inputs and outputs, but we cannot see or understand what happens inside to connect them.
 - Neural networks
 - Random Forest (can get a single tree, white box)
 - Gradient Boosting (XGBoost, LightGBM, CatBoost)
 - Support Vector Machines (SVM)

TERMS

- **White box:** we know what happened between inputs and outputs, good for generating theories.
 - Linear regression
 - Logistic regression
 - Decision trees
 - K nearest neighbors (KNN)
 - Naive Bayes

TERMS

- Learning algorithms: also called **learners**, are the specific **mathematical rules and statistical procedures** that allows a computer to “learn” patterns from data.



LEARNING ALGORITHMS

- There are many algorithms. Commonly used learning algorithms:
 - Linear and polynomial regression
 - Logistic regression
 - K-nearest neighbors (KNN)
 - Support vector machines (SVM)
 - Decision trees
 - Random forest
 - Ensemble methods

TYPES OF LEARNING SYSTEM

- Types of learning system/learning algorithms
 - Supervised learning
 - Unsupervised learning
 - Semi-supervised learning
 - Reinforcement learning
 - Batch and online learning
 - Instance-based learning and model-based learning

SUPERVISED LEARNING

- **Supervised learning**: the training data is **labelled**, which means datasets have both features and label (target).
 - Predictive model: The learning algorithm in a predictive model discovers and models the relationships between the target variable (the dependent variable) and features/attributes (predictors).

SUPERVISED LEARNING

- Supervised learning
 - Classification: The learning algorithm in a classification predicting a binary or multinomial response measure.

SUPERVISED LEARNING

- Supervised learning algorithms
 - K nearest neighbors (KNN)
 - Linear regression
 - Logistic regression
 - Support vector machines (SVM)
 - Decision trees and random forest
 - Neural networks

UNSUPERVISED LEARNING

- **Unsupervised learning**: the training data is **unlabeled**. It performs the analysis **without** a target variable.
- **Clustering**: The goal of clustering is to **segment** observations into **similar groups** based on the observed variables.
 - K-means
 - Hierarchical cluster analysis
 - Expectation maximization

UNSUPERVISED LEARNING

- **Dimension reduction**: reducing the number of **variables** in a data set.
 - Principal component analysis (PCA)
 - Kernel PCA
 - Locally-linear embedding (LLE)
 - T distributed stochastic neighbor embedding (t-SNE)

UNSUPERVISED LEARNING

- **Association rule learning**: discover associations between features.
 - Apriori
 - Eclat

EXAMPLE PROJECT

- Use YRBSS 2023 data (from CDC).
- We want to create a **logistic regression** model, a screen test.
- We want to use age (Q1), gender(Q2), sad/hopeless (Q26), current alcohol use (Q42), and feeling connected (Q103) to predict electronic vapor product use (Q35r), a binary variable.

EXAMPLE PROJECT

- I chose logistic regression for the data analysis demonstration. Other algorithms (e.g., Support vector machines, XGBoost) may be better.

EXAMPLE PROJECT

- Possible research questions
 - Does my proposed model perform well? Or is my model good enough to be useful?
 - Which predictor (age, gender, feeling connected or sadness/hopelessness) has the strongest marginal effect on electronic smoking probability?
 - Does the importance of sadness/hopelessness differ by gender or by age group? (using tree-based algorithms is better: random forest, XGBoost).

EXAMPLE PROJECT

- Research question
 - Does my proposed model perform well? Or is my model good enough to be useful?
 - We focus on model **performance**.

TYPICAL ML PROJECT

- A typical ML project workflow
 - Study data
 - Select model/algorithm
 - **Train** the model on the **training** data: running logistic regression on training data.
 - **Evaluate** model, **tune** the model, and determine a final, **optimal** model.
 - Apply the model on new cases using testing data (**test** model)
 - Launch, monitor, and maintain the system.

CHOOSE AN ALGORITHM

- Choose an **algorithm**
 - Prediction
 - Classification
 - Clustering
 - Dimensionality reduction

HOW TO CHOOSE A MODEL

- It is important to choose a **good** model and have a high quality of training data set.
- For example, we want to choose between linear model and a polynomial model.
 - We train **both models** and see which one is better in predicting new cases.

TRAINING A MODEL

- We train our model using **training data** and find **parameters** (e.g., b_0 and b_1) that make the logistic model fit best to our data. This procedure is called **Training the model**.

TRAINING DATA

- We **split** our raw data into two parts: training set (commonly **70%** or **80%** of total sample) and test set (**30%** or **20%** of total sample, treated as new cases).
 - We **train** our model using **training set**.
 - We **test** our model using **test set**.

TRAINING DATA

- Training data is the **dataset** that a machine learning algorithm uses to learn patterns, relationships, and rules. It is the **input data** from which the model builds its understanding.
 - Training data has both inputs (features) and outputs (labels) in supervised learning.
 - Training data has only inputs (no labels) in unsupervised learning.

TRAINING DATA

- Unbalanced target:
 - **Down-sampling** balances the dataset by reducing the size of the **majority** class(es) to match the size in the **minority** class.
 - Target has two categories: Yes include 5% data points, No includes 95% data points.
 - Keeping all samples in the minority class and **randomly** selecting an equal number of samples in the majority class.

TRAINING DATA

- Unbalanced target:
 - **Up-sampling/oversampling** means increasing the number of samples in the **minority** class of an imbalanced dataset.
 - **Randomly duplicates** existing samples from the minority class.
 - Create new, synthetic samples (SMOTE).
 - Use **class weight** parameter (XGBoost)
 - **Bootstrapping oversampling**: Draw samples from the minority class with replacement.

TRAINING DATA

- **Warning:** Do **NOT** down-sample and up-sample **before** splitting your data!
 - We must keep our **test** set completely untouched, imbalanced, and representative of **real-world** conditions!
 - Do it **after** splitting the data and only for **train** data!

TRAINING DATA

- Why do we need to do down/up sampling **after** splitting data? To prevent **data leakage**!
- Data leakage:
 - Train-Test contamination: When information from test dataset accidentally leaks into training dataset during the **feature engineering** or **preprocessing** phase: **X** we **standardize** features on whole data set.

TRAINING DATA

- **Nonrepresentative training data**
 - The training data should be **representative** of new cases we want to predict.
 - When using nonrepresentative training data, we trained a model to make **inaccurate** prediction (generalization problem).
 - **Sample size** is a concern. We will have sampling bias when the sample size is too small.
 - If sampling method is flawed, the sample is not representative.
 - Poor quality of data.

FEATURE ENGINEERING

- Feature engineering: feature transformation.
 - We must do it **after** splitting data to prevent data leakage!!!
 - **Except**, we can remove missing for target **before** splitting data.

FEATURE ENGINEERING

- Data transformations:
 - Standardize features (after data splitting).
 - Recode categorical features into dummy variables (after data splitting).
 - Impute missing values (after data splitting).
- Feature creation:
 - Combine features (before data splitting). For example, create a new feature c using feature a and b (across columns).
 - Compute new features.

FEATURE ENGINEERING

- When having categorical features, tree-based algorithms do **NOT** need dummy coding, but linear-based algorithms do.
 - **Tree-based algorithms:** XGBoost, Random Forest, LightGBM, or CatBoost. We should avoid dummy-coding and use **native** categorical features instead.
 - **Linear-based algorithms:** Linear/Logistic Regression, SVMs, Neural Networks, or K-Nearest Neighbors.

FEATURE ENGINEERING

- When we need to impute missing values.

Model Type	Can Handle NaN?	Need Imputation?
XGBoost	✓ Yes (native)	✗ No
LightGBM	✓ Yes (native)	✗ No
CatBoost	✓ Yes (native)	✗ No
Random Forest	👉 Some implementations	🟡 Sometimes
Decision Trees	👉 Some implementations	🟡 Sometimes

FEATURE ENGINEERING

- When we need to impute missing values.

Model Type	Can handle Nan	Need imputation
Logistic Regression	✗ No	✓ Yes
Linear Regression	✗ No	✓ Yes
SVM	✗ No	✓ Yes
Neural Networks	✗ No	✓ Yes
KNN	✗ No	✓ Yes
Naive Bayes	✗ No	✓ Yes

FEATURE ENGINEERING

- Features in the training data
 - **Feature selection**: for example, in a logistic regression model, it should include important predictors.
 - **Feature extraction**: combine existing features to create new features. Especially, we have a lot of predictors. We may use **dimensionality reduction** to reduce number of features.

RESCALING

- Feature scaling: important in ML!
 - Features are predictors in our logistic regression model.
 - If features have different scales, ML performs bad.
 - We need to standardize our variables.
 - However, it is not required to rescale the dependent variable.



RESCALING

- When features are binary variables, we do **NOT** need to standardize them.

RESCALING

- Two ways to rescale features
 - **Normalization**: $(\text{original value} - \text{minimum value}) / (\text{Maximum value} - \text{minimum value})$. The transformed values will be between 0 and 1.
 - **Standardization**: $(\text{original value} - \text{mean}) / \text{standard deviation}$. Mean = 0 and standard deviation = 1.

RESCALING

- We only rescale our **training** dataset.
- A common mistake is to apply some data **preprocessing** such as normalization **before** splitting the entire data into training and test sets. This causes **data leakage**.

OVERFITTING

- **Overfitting of train model:** model overfits the data. Training performance is **much higher** than **test** performance: The model performs excellently on the training data, but fails badly on new, unseen data.
- The model detects pattern in the noise, outliers, and random fluctuations, rather than learning the underlying **true** pattern.

OVERFITTING

- Solutions to overfitting
 - When model is **too complex**: simplify the model or reduce the number of features.
 - When training data is too small: collect more data.
 - Reduce noise in the training data: remove outliers.

UNDERFITTING

- **Underfitting of train model**: the model is too simple to learn the pattern of the data: Both training and test performance are **poor**.
- Solutions:
 - Select more **complex model** with more parameters.
 - Include **good** features in the model.
 - Reduce the constraints on the model (e.g., adjust **hyperparameters**).

HYPERPARAMETER

- **Hyperparameters**: These are the external **configurations** we **set** before training begins and keep it constant during the training.
- They are **NOT** parameters of a model, but influence model parameters. So, they control how the model learns and the structure of the model itself.
 - Mathematically, they change the **formula** used to calculate model parameters.

HYPERPARAMETER

- The differences between model parameters and hyperparameters

	Model Parameters	Hyperparameters
Definition	Learned from the training data	Set manually before training
Who sets them?	The learning algorithm	The data scientist / engineer
When are they determined?	During training	Before training starts
Change during training?	Yes, they update continuously	No, they remain fixed
Examples	Regression coefficients, Decision tree split values, Neural network weights	Learning rate, Tree depth, Number of clusters (K), Regularization strength

HYPERPARAMETER

- We can train 100 different models using 100 hyperparameter values. For example:
 - In K nearest neighbors: we can change **K**
 - In Decision trees: we can change the **depth** of our tree.

HYPERPARAMETER

- In logistic regression, hyperparameters include

Hyperparameter	What It Does	Common Values / Notes
<code>C</code>	Inverse of regularization strength. Smaller values = stronger regularization (simpler model). Larger values = less regularization (can overfit) 1 2 7 .	Typically tested on a log scale: <code>[0.001, 0.01, 0.1, 1, 10, 100]</code> 1 4 .
<code>penalty</code>	Specifies the type of regularization to use 3 6 .	<code>'l2'</code> (Ridge, default), <code>'l1'</code> (Lasso, for feature selection), <code>'elasticnet'</code> (combines both), or <code>'none'</code> 3 7 .
<code>solver</code>	The algorithm used for optimization. Different solvers support different types of penalties 1 2 3 .	<code>'lbfgs'</code> (good default), <code>'liblinear'</code> (supports L1/L2), <code>'saga'</code> (supports all penalties incl. Elastic-Net) 1 2 7 

HYPERPARAMETER

- In logistic regression, hyperparameters include

<code>max_iter</code>	Maximum number of iterations for the solver to converge <code>1</code> <code>3</code> .	Increase this if you see a convergence warning. Often set to <code>1000</code> or <code>5000</code> <code>1</code> <code>3</code> .
<code>class_weight</code>	Adjusts weights for each class. Useful for imbalanced datasets.	<code>'balanced'</code> automatically gives more weight to the minority class <code>3</code> <code>4</code> <code>5</code> .
<code>fit_intercept</code>	Whether to include an intercept (bias) term <code>3</code> <code>4</code> .	<code>True</code> (default) or <code>False</code> .
<code>tol</code>	Tolerance for stopping the optimization <code>3</code> .	Smaller values mean the algorithm runs for longer to find a more precise solution.

HYPERPARAMETER

- Purpose of using hyperparameters:
 - Prevent overfitting
 - Control model complexity
 - To adapt to different problems: for example, we use $c = 1$ for one dataset, we may use $c = 0.1$ for other datasets.

HYPERPARAMETER

- How to choose right hyperparameters: we need to **experiment**. This process is called **Hyperparameter Tuning**.
- Keep this in mind: Hyperparameters are **algorithm-specific**.

REGULARIZATION

- **Tuning models:** manipulate hyperparameters.
 - **Regularization** is one of the strategies to **tune** hyperparameters. it reduces overfitting by panelizing complexity of a model.
 - The amount of regularization applied can be controlled by a **hyperparameter**.
 - For example, in logistic regression, regularization keeps model coefficients small.

REGULARIZATION

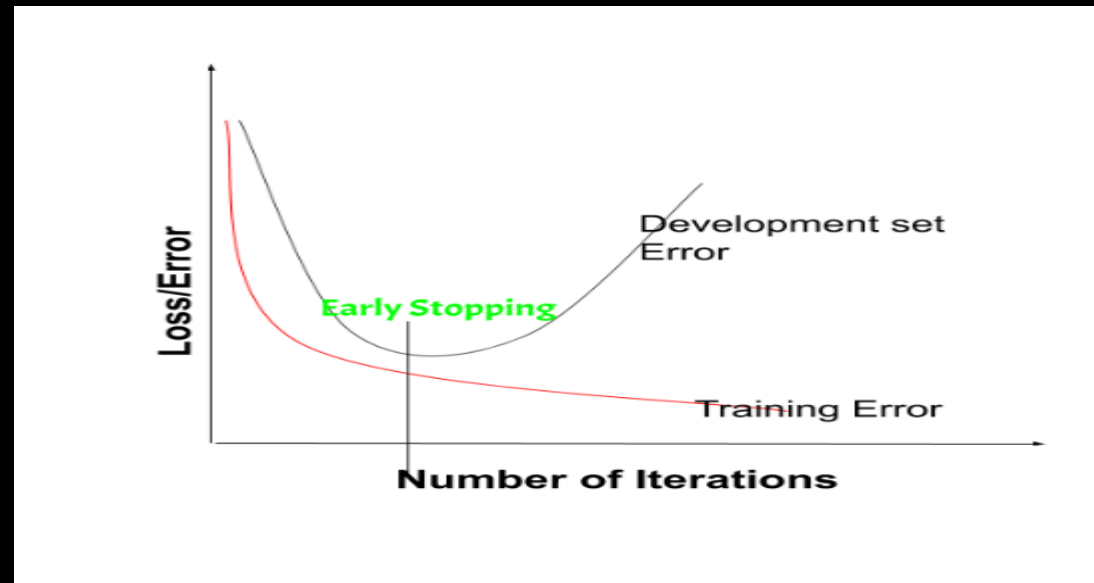
- Techniques of regularization
 - **Weight penalties**: They directly punish the model for having **large** numeric weights (parameters).
 - L2 Regularization (Ridge): It forces the weights to be **small** and evenly distributed.
 - L1 Regularization (Lasso): It forces the smallest weights to become exactly zero.
 - Used in linear/logistic regression, neural network, tree boosting.

REGULARIZATION

- Techniques of regularization: **Early stopping** (It prevents **overfitting and saves time**).
 - It stops a machine learning model early by monitoring **validation** performance, halting training when progress stalls, and restoring the best weights.
 - It stops the model from studying too long so it does not ruin its ability to handle new data.
 - Used in **gradient boosting algorithms** and **neural networks**.

REGULARIZATION

- **Early stopping:** loss/error is a performance measure.



REGULARIZATION

- How **Early stopping** works: During iterative training, the algorithm monitors performance on **two** distinct datasets:
 - **Training set**: The loss/error continually decreases as the model fits the data more closely.
 - **Validation set**: The loss/error decreases initially, hits an optimal **sweet spot**, and then begins to increase as the model starts to overfit. Validation set is from training data.
- Early stopping acts as an automated trigger that saves the model weights at that optimal sweet spot and terminates **training** immediately once the validation loss stops improving.

REGULARIZATION

- In logistic regression, hyperparameters include `c`, `penalty`, and `solver`. L1, L2 are **regularization** techniques.

Hyperparameter	What It Does	Common Values / Notes
<code>c</code>	Inverse of regularization strength. Smaller values = stronger regularization (simpler model). Larger values = less regularization (can overfit) 1 2 7 .	Typically tested on a log scale: <code>[0.001, 0.01, 0.1, 1, 10, 100]</code> 1 4 .
<code>penalty</code>	Specifies the type of regularization to use 3 6 .	<code>'l2'</code> (Ridge, default), <code>'l1'</code> (Lasso, for feature selection), <code>'elasticnet'</code> (combines both), or <code>'none'</code> 3 7 .
<code>solver</code>	The algorithm used for optimization. Different solvers support different types of penalties 1 2 3 .	<code>'lbfgs'</code> (good default), <code>'liblinear'</code> (supports L1/L2), <code>'saga'</code> (supports all penalties incl. Elastic-Net) 1 2 7 v

FINE-TUNE MODEL

- Common strategies to **tune** the model: finding the best hyperparameters that maximize the model's **accuracy and performance** on new, unseen data:
 - **Grid search**: we define a set of possible values for the hyperparameters (e.g., $K = 3, 5, 7, \dots, 49$) and try every single combination.
 - **Random search**: We pick random combinations from a defined range of hyperparameters.

FINE-TUNE MODEL

- Common **strategies** to tune the model:
 - **Bayesian optimization**: A smart, probabilistic approach that learns from past attempts to guess which hyperparameters will work best next rather than trying options blindly.
 - **Automated tuning** (AutoML): Tools like Optuna or Hyperopt that run the experiments for us automatically.



FINE-TUNE MODEL

- Common strategies to tune the model:
 - **Ensemble method**: combines multiple machine learning models to create a single, more powerful model.

FINE-TUNE MODEL

- Workflow of Grid search with Cross-validation. We only use training data.

```
For each hyperparameter combination in param_grid:  
    For each fold in cv=10:  
        1. Split data into train_fold and val_fold  
        2. Train model on train_fold with current hyperparameters  
        3. Predict on val_fold  
        4. Calculate accuracy  
    Average accuracy across 10 folds → store as score for this combination  
  
After trying all combinations:  
    Find combination with HIGHEST average accuracy  
    Store it as best_params_  
    Retrain final model on ALL training data using best_params_
```

FINE-TUNE MODEL

- Tuning mode is very important in ML.
 - Hyperparameters directly control the overfitting vs. underfitting.
 - Tuning hyperparameters can result in high predictive accuracy. The goal is to find an optimal model.
 - We need hyperparameters almost every time when using a machine learning model.
 - For example: in k -nearest neighbor models, we can change K to get different models.

OPTIMIZATION

- **Optimization** is used to calculate the model **parameters** (e.g., weights) for a given fixed set of hyperparameters.
- Fit() in Python. `grid_search.fit(X_train, y_train)`

MODEL EVALUATION

- **Model performance** measures (generalization performance)
 - **Utility function** (fitness function): measures how **good** the model fits the data.
 - **Cost function**: measures how **bad** the model fits the data.

MODEL EVALUATION

- In Linear regression model
 - Mean squared error (MSE): the average of the squared error.
 - Root mean squared error ($RMSE$): the square root of the MSE metric.
 - Mean absolute error (MAE)
 - R^2

MODEL EVALUATION

- In classification model
 - **Accuracy**: how model **correctly** predicts the outcome.
 - **Cross-entropy**: similar to MSE, we need to **minimize** it.
 - **Gini index**: mainly in **decision trees**, a small value indicates that a node contains predominantly observations from a single class. We need to minimize it.
 - **Confusion matrix**: 2*2 table of predicted class and observed class.

MODEL EVALUATION

- Confusion matrix: for example, in logistic regression, we can get a table below.

	predicted events	predicted non-events
actual events	correctly forecasted events	missed events
actual non-events	missed non-events	correctly forecasted non-events

	predicted events	predicted non-events
actual events	True Positive	False Negative
actual non-events	False Positive	True Negative

MODEL EVALUATION

- Based on the table, we can estimate:
 - **Accuracy**: % of correctly predicted = $(TP+TN)/total$
 - **Precision**: % of correctly predicting **positive** cases (the event) = $TP/(TP+FP)$
 - **Recall (sensitivity)**: ability to detect an individual with event (true positive) = $TP/(TP+FN)$
 - **Specificity**: ability to detect an individual without event (true negative) = $TN/(TN+FP)$
 - **F1 score**: harmonic mean of Precision and Recall.

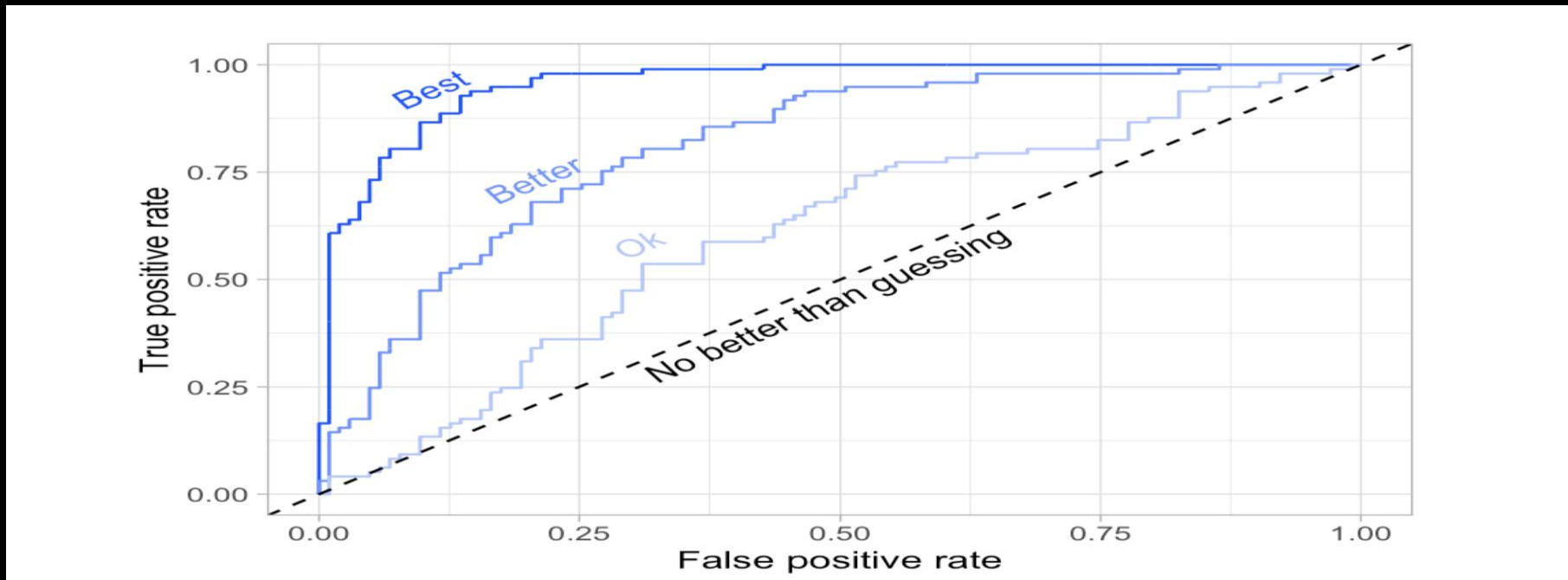
MODEL EVALUATION

- **F1 score**: balance precision and recall. Higher value means more successful our model is at managing the "yes" group (if yes is the focus).
- **Warning**: Standard accuracy can be highly misleading if our data is imbalanced.

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

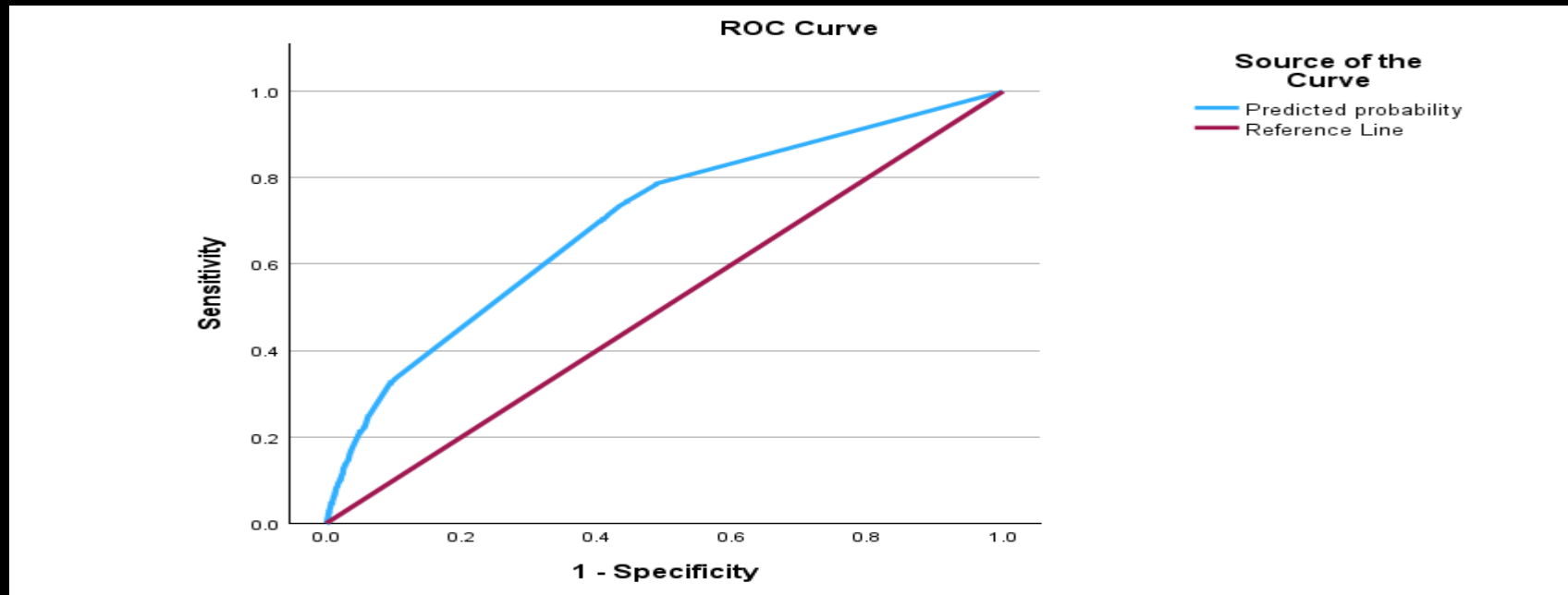
MODEL EVALUATION

- Based on the table, we can estimate:
 - **AUC**: Area under the curve. Use **ROC** curve to display it.



MODEL EVALUATION

- Example: AUC= 69.7% (our model correctly predict “Yes” 69.7% of the time)

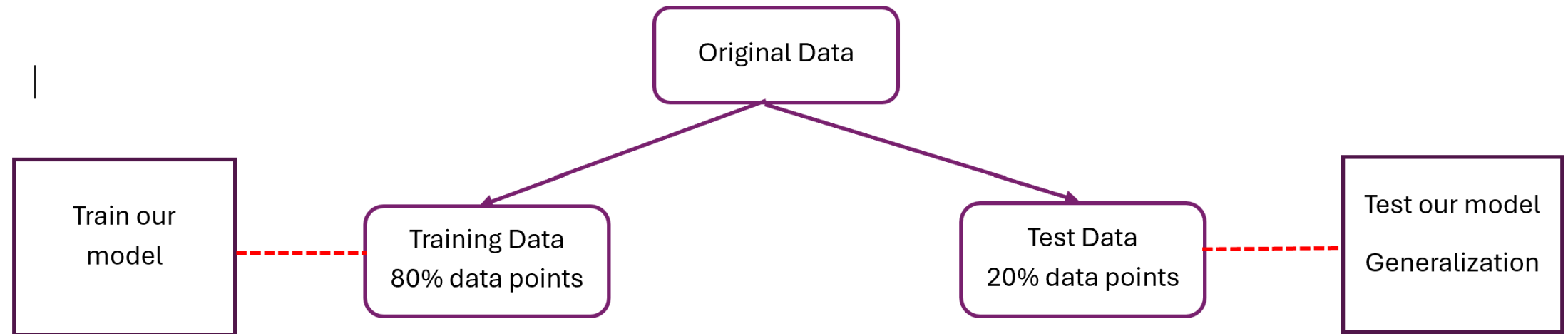




TESTING AND VALIDATION

- We want to know how well our model can generalize to new cases.
- We need to **test** our model using test data.

TESTING AND VALIDATION



TESTING AND VALIDATION

- We need to choose performance measures to evaluate our model:
 - For a linear regression model, we use Root Mean Square Error (**RMSE**). It tells us how much error the model makes in prediction.
 - **MAE** (Mean Absolute Error).
 - Both RMSE and MAE measure the average magnitude of **residuals**.

TESTING AND VALIDATION

- Cut-off values of NRMSE: RMSE relative to mean.
 - $\text{NRMSE} = \text{RMSE} / (\text{mean of actual values})$

NRMSE	Interpretation
< 0.1 (10%)	Excellent
0.1 – 0.2 (10–20%)	Good
0.2 – 0.5 (20–50%)	Fair
> 0.5 (50%)	Poor

TESTING AND VALIDATION

- We also can compare RMSE to standard deviation
 - If $RMSE < SD$: our model predicts better than the intercept only model (baseline model).
 - If $RMSE \approx SD$: Model is no better than the baseline model.
 - If $RMSE > SD$: Model is worse than the simplest possible guess.

TESTING AND VALIDATION

- For classification models:
 - Accuracy of prediction
 - Precision
 - Recall
 - AUC
 - Confusion matrix

TESTING AND VALIDATION

- Scoring metric in Python: use which metric to evaluate a model

Scoring Metric	Best For	What It Measures
<code>'accuracy'</code>	Balanced classification	Percentage of correct predictions
<code>'roc_auc'</code>	Imbalanced classification	Area under the ROC curve (handles class imbalance well)
<code>'f1'</code>	When false positives AND false negatives are costly	Harmonic mean of precision and recall
<code>'precision'</code>	When false positives are costly (e.g., spam detection)	Of all predicted positives, how many are actually positive?
<code>'recall'</code>	When false negatives are costly (e.g., cancer detection)	Of all actual positives, how many did we catch?
<code>'neg_mean_squared_error'</code>	Regression problems	Average squared error (negative because sklearn maximizes)



TESTING AND VALIDATION

- Validation: it is **complicated** and there are different strategies.
- We use **training data** for the Cross-validation (CV).

TESTING AND VALIDATION

Method	How it works	Best for
Train-Test Split	Single random split into train & test	Large datasets (>>10,000 samples)
K-Fold Cross-Validation	Split data into k folds; train on k-1, validate on 1; repeat k times	Small to medium datasets
Stratified K-Fold	Same as K-Fold, but preserves class percentages	Classification with imbalanced classes
Leave-One-Out (LOO)	k = number of samples; train on all but one, validate on one	Very small datasets
Time Series Split	Train on past, validate on future (no shuffling)	Time series forecasting

TESTING AND VALIDATION

- K-fold cross validation (resampling method):
for example, 10-fold cross validation.

TRAINING DATA (e.g., 70% of original)

|
└─ 10-fold CV splits it further:

Fold 1: [Train: folds 2-9] [Validate: fold 1]

Fold 2: [Train: folds 1,3-9] [Validate: fold 2]

...

Fold 10: [Train: folds 1-9] [Validate: fold 10]

Each validation fold is "unseen" within the training data

TEST DATA (never touched until the end)

TESTING AND VALIDATION

- Validation
 - We train **different** models using different hyperparameters in the **training** set.
 - Select the model and hyperparameter that perform **best** in the validation set.
 - Run the final test using test data to get generalization error.

TESTING AND VALIDATION

- If the training error is low and generalization error is high, it means our model is **overfitting** the training data set.

ISSUES

- Machine learning needs **huge data/examples** (20000 cases are considered as a small data set) to get the algorithm. It won't perform well if you have a small data set.
- High quality data are very important.
- The training data should be representative of the new cases we want to **generalize** to. If the data are not representative, the prediction won't be accurate.



ISSUES

- The model is not too simple (underfitting) or too complex (overfitting).
- After training a model, we need to evaluate it, fine-tune it if necessary.

EXAMPLE PROJECT

- Use YRBS 2023 data.
- We want to create a logistic regression model.
- We want to use age (Q1), gender (Q2), feeling sad/hopeless (Q26), current alcohol use (Q42), and feeling connected (Q103) to predict electronic vapor product use (Q35r).

EXAMPLE PROJECT

- We have a labelled data set. So, we have a supervised learning model.
- The target is a binary variable. So, we use logistic regression algorithm.



REFERENCE

- Géron, A. (2023). Hands-on machine learning with Scikit-Learn and TensorFlow: Concepts, tools, and techniques to build intelligent systems (3rd ed.). O'Reilly Media.