# CSCI 470: Machine Learning

Case Study 2, Part 1
September 23, 2021

# Case Study 2: Part 1

- Program a Python class that implements a type of (not Naive) Bayesian classifier
  - 2D Gaussian distribution on features
- Implement the class methods, like those of scikit-learn models
  - `fit()`
  - `predict()`
  - `fit_predict()`
- Build, train, and test your Bayesian classifier
- Evaluate your model
  - Confusion matrix
  - Accuracy
  - Precision, Recall, and F-score

# Scikit-Learn model functions

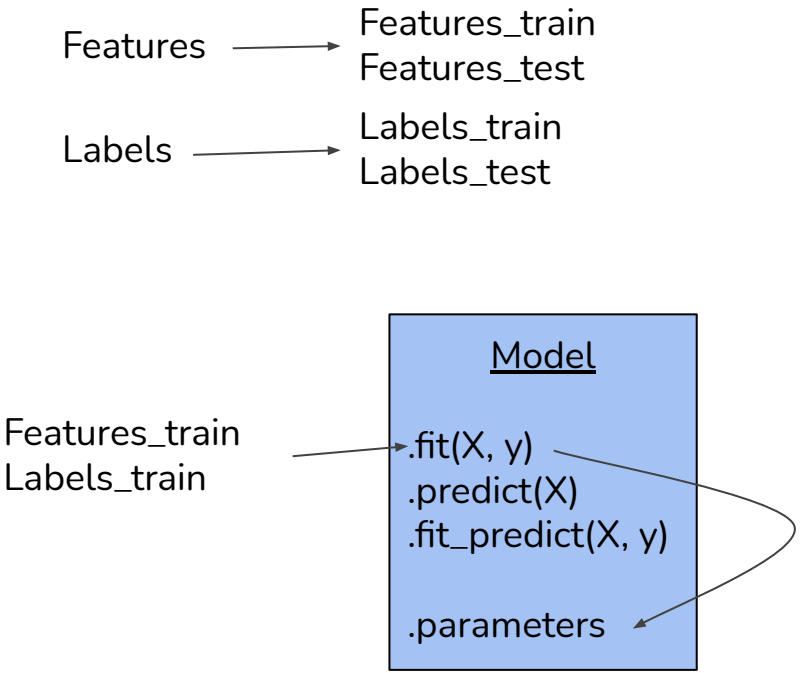
Methods common to many of the scikit-learn model classes

- fit(X, y)
  - Trains a model using features, X and labels/targets, y
  - labels/targets may be discrete-valued (classification) or continuous-values (regression)
  - Model parameters are stored as attributed of the class instantiation
- predict(X)
  - Using the trained model, make label/target predictions for the features, X, and return those predictions (e.g., y_hat)
  - X could be any set of features -- those used the train the model in fit(), those of a validation or test set, or something different altogether
- fit_predict(X, y)
  - Simply calls fit(X,y), to train the model, then calls predict(X) and return the output. Note that X is the features used in both, such that the returned predictions are those of the samples used to train the model.
  - Just a convenience function

# Model training and testing: Review

THIS PROCESS IS THE MOST FUNDAMENTAL AND IMPORTANT CONCEPT IN MACHINE LEARNING

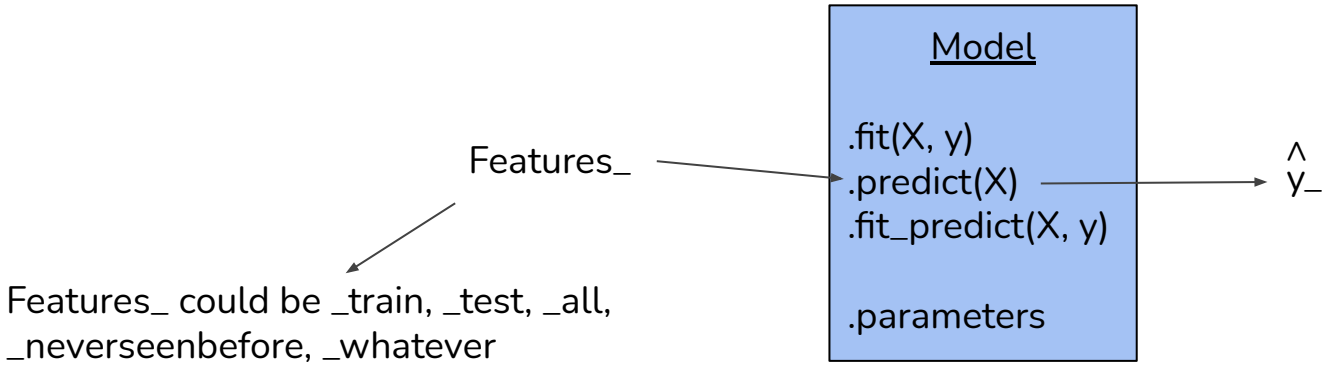
# Training


Features
Features_train
Features_test
Labels
Labels_train
Labels_test
Model
Features_train
Labels_train
.fit(X, y)
.predict(X)
.fit_predict(X, y)
.parameters

# Prediction

Features → Features_train
Features_test

Labels → Labels_train
Labels_test

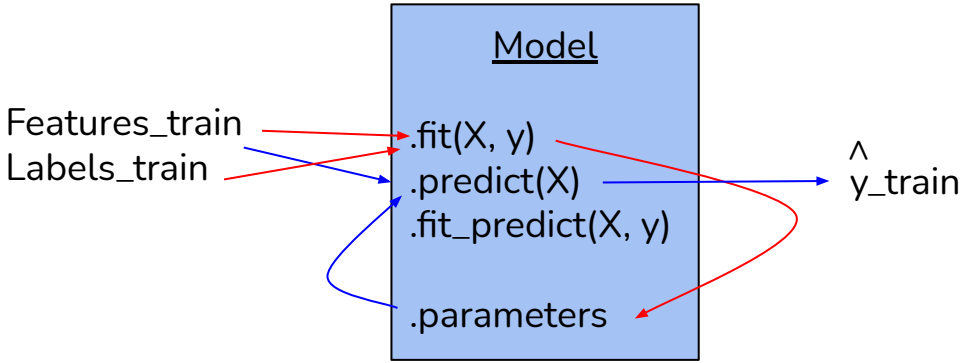


# Training, and prediction on train set

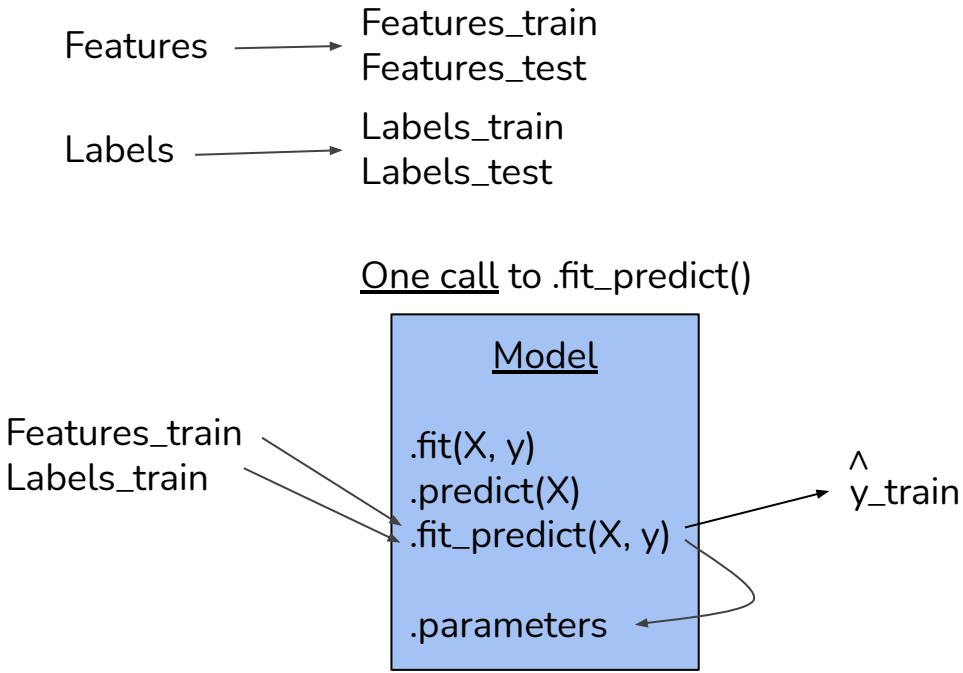
Features → Features_train
Features_test

Labels → Labels_train
Labels_test

Two separate calls to .fit() and .predict()

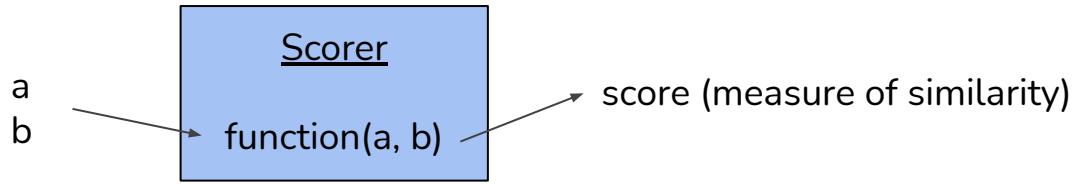


# Training, and prediction on train set



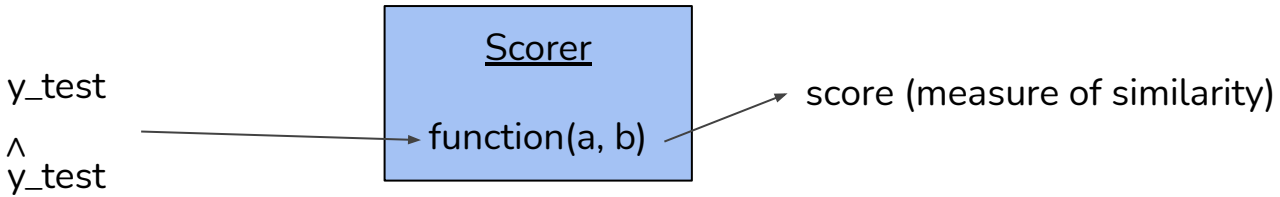
# Scoring

A metric is a function of two variables, which provides a measure, or score, of similarity of the values of those variables.



# Evaluating (via the test set)

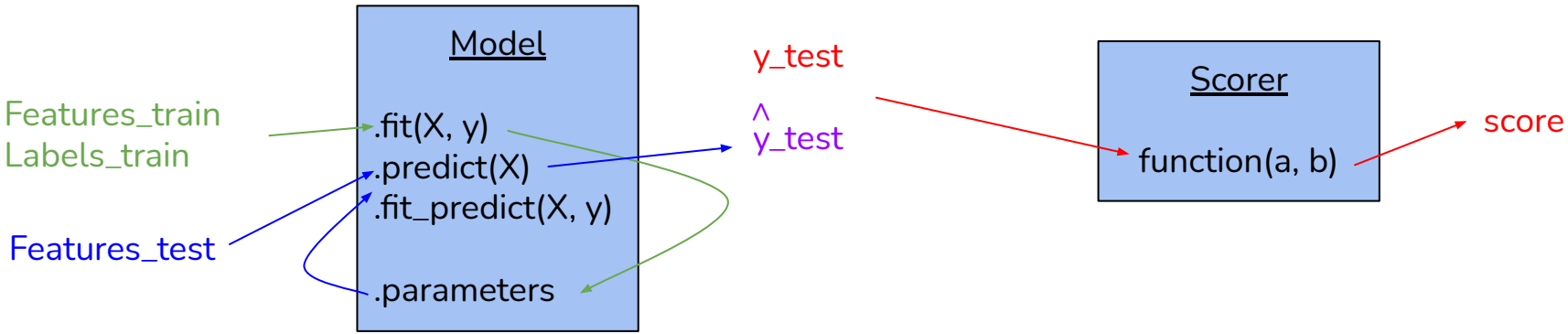
In supervised learning, we specify a **metric** that provides a measure of the similarity of two values, or two vectors of values. These values could come from anywhere, but when evaluating a SL model the values are (a) test set's labels, *y_test*, and (b) the predictions the trained model generates from the test set's features, *y_test_hat*.



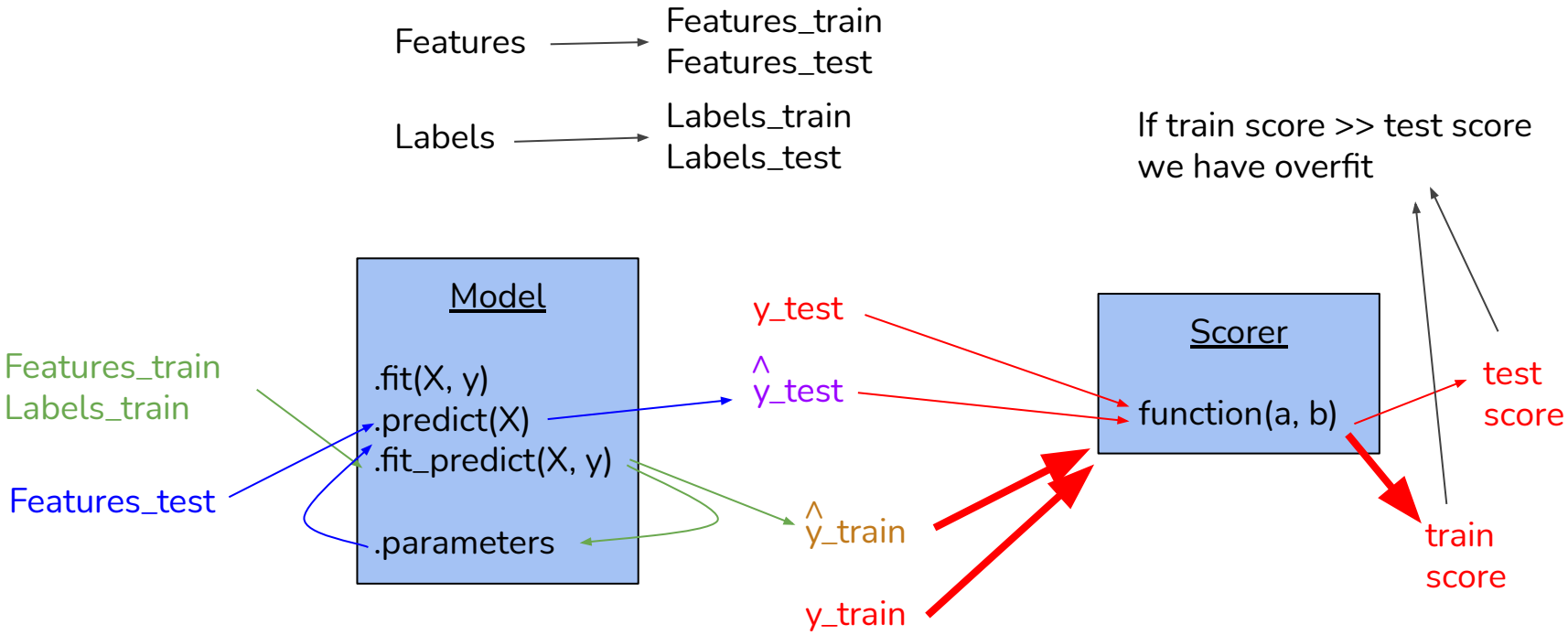
# Train, test, and evaluate

Features → Features_train
Features_test

Labels → Labels_train
Labels_test



# Train, test, and evaluate
## ... with assessment of overfitting



# Model training and testing: Terminology

- **Samples** - Individual samples of data. Some of which you will use to train your model, some of which you will use to test your model.
- A trained model ingests sample **features** and outputs sample **predictions**
- To train a supervised learning model we use the model's training algorithm, which ingests **features** and **labels**, and outputs (or stores) the trained model's **parameters**. The training algorithm attempts to minimize a **loss function**.
- Synonyms
  - **Training** == **fitting** == **learning**
  - **Labels** == **Targets**
    - **Labels/targets** may be discrete values (**classes**, in a classification problem)
    - **Labels/targets** may be continuous values (in a regression problem)
  - **Parameters** == **weights** == **coefficients**
  - **Loss function** == **objective function**
- Variable names (commonly used, but they can be anything)
  - **X** - a matrix of features, usually with samples in rows, features in columns
  - **y** - a vector of targets/labels