Data Science for Spatial Systems

目录

[Linear Regression 2](#_Toc189638214)

[Supervised learning 6](#_Toc189638215)

[DecisionTreeRegressor 6](#_Toc189638216)

[Metrics 13](#_Toc189638217)

[Metrics for regression 13](#_Toc189638218)

[Metrics for classification 13](#_Toc189638219)

[Tree-based Methods 14](#_Toc189638220)

[Decision trees 14](#_Toc189638221)

[Ensemble learning 17](#_Toc189638222)

[Random Forest 17](#_Toc189638223)

[GBDT 梯度提升决策树 18](#_Toc189638224)

[Permutation feature importance (PFI) 18](#_Toc189638225)

[Code 19](#_Toc189638226)

[Analysis Workflow 25](#_Toc189638227)

[Basic error analysis 25](#_Toc189638228)

[Machine learning strategy 26](#_Toc189638229)

[Techniques for reducing bias 26](#_Toc189638230)

[Techniques for reducing variance 27](#_Toc189638231)

[Data leakage 27](#_Toc189638232)

[Code 27](#_Toc189638233)

[Artificial Neural Networks 32](#_Toc189638234)

[Code 32](#_Toc189638235)

宝典：https://github.com/YYY677/PythonDataScienceHandbook

## Linear Regression

**Using VIF to deal with multicollinearity**

# calculating VIF

# This function is adjusted from: https://stackoverflow.com/a/51329496/4667568

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

from statsmodels.tools.tools import add\_constant

def **drop\_column\_using\_vif\_**(df, thresh=5):

    '''

    Calculates VIF each feature in a pandas dataframe, and repeatedly drop the columns with the highest VIF

    A constant must be added to variance\_inflation\_factor or the results will be incorrect

    :param df: the pandas dataframe containing only the predictor features, not the response variable

    :param thresh: (default 5) the threshould VIF value. If the VIF of a variable is greater than thresh, it should be removed from the dataframe

    :return: dataframe with multicollinear features removed

    '''

    while True:

        # adding a constatnt item to the data. add\_constant is a function from statsmodels (see the import above)

        df\_with\_const = add\_constant(df)

        #variance\_inflation\_factor 是 statsmodels 库中的一个函数，用于计算方差膨胀因子（VIF）

        vif\_df = pd.Series([variance\_inflation\_factor(df\_with\_const.values, i)

            for i in range(df\_with\_const.shape[1])], name= "VIF",

            index=df\_with\_const.columns).**to\_frame**()

        # drop the const

        vif\_df = vif\_df.**drop**('const')

        # if the largest VIF is above the thresh, remove a variable with the largest VIF

        # If there are multiple variabels with VIF>thresh, only one of them is removed. This is because we want to keep as many variables as possible

        if vif\_df.VIF.**max**() > thresh:

            # If there are multiple variables with the maximum VIF, choose the first one

            index\_to\_drop = vif\_df.index[vif\_df.VIF == vif\_df.VIF.**max**()].**tolist**()[0]

            '''

            tolist() 返回的是一个列表，[0] 表示从这个列表中获取第一个元素。因为这里可能存在

            多个特征具有相同的最大 VIF 值，但代码仅选择其中的第一个进行删除。

            '''

**print**('Dropping: {}'.**format**(index\_to\_drop))

            df = df.drop(columns = index\_to\_drop)

        else:

            # No VIF is above threshold. Exit the loop

            break

    return df

df\_predictors\_selected\_VIF = **drop\_column\_using\_vif\_**(bike\_rental\_final.**drop**('cnt', axis=1))

**print**("The columns remaining after VIF selection are:")

**print**(df\_predictors\_selected\_VIF.columns)

**Modeling**

model\_bike\_rental = sm.OLS(endog=bike\_rental\_final[['cnt']], exog=sm.add\_constant(df\_predictors\_selected\_VIF)).fit()

'''

endog 和 exog 是用来指定模型的因变量和自变量，endog 代表需要预测的目标，exog 代表预测该目标的特征。

sm.OLS() 提供更底层的灵活性，而 sm.formula.ols() 通过公式语法提供更高层次的用户友好性。

选择哪个方法取决于个人偏好和具体需求。

'''

model\_bike\_rental.summary()

**Linear relationship**

test\_rainbow = statsmodels.stats.diagnostic.linear\_rainbow(model\_bike\_rental)

# This function returns a tuple consisting of two values: the test statistic based on the F test and the pvalue of the test

**print**("The p value of the rainbow test: {:.4f}".**format**(test\_rainbow[1]))

This test assumes residuals are homoskedastic and may reject a correct linear specification if the residuals are heteroskedastic and the p value is less than or equal to the predefined significance level (usually 0.05).

**Independent errors**

test\_dw = statsmodels.stats.stattools.durbin\_watson(model\_bike\_rental.resid)

**print**("Durbin-Watson test statistic is: {:.4f}".**format**(test\_dw))

This statistic will always be between 0 and 4.

If there is no serial correlation, the test statistic equals 2.

The closer to 0, the more evidence for positive serial correlation.

The closer to 4, the more evidence for negative serial correlation.

**Normally distributed errors**

test\_JB = statsmodels.stats.stattools.jarque\_bera(model\_bike\_rental.resid)

**print**("The p value of the Jarque Bera test: {:.4f}".**format**(test\_JB[1]))

The null hypothesis of this test is that the data is normally distributed.

This test will return a tuple of four values, namely the Jarque-Bera test statistic, p value of the test statistic, estimated skewness of the data, and estimated kurtosis of the data.

if p value is less than 0.05, we will reject the null hypothesis and conclude that the residuals are not normally distributed.

**Equal variance**

statsmodels.stats.diagnostic.het\_goldfeldquandt(model\_bike\_rental.model.endog, model\_bike\_rental.model.exog)

The null hypothesis is that the variance in the two sub-samples are the same. The alternative hypothesis can be increasing (i.e. the variance in the second samples is larger than in the first), or decreasing or two-sided.

This test will return a tuple of four values, namely the Jarque-Bera test statistic, p value of the test statistic, estimated skewness of the data, and estimated kurtosis of the data.

**### Basics of the API**

Most commonly, the steps in using the Scikit-Learn estimator API are as follows

1. Choose a class of model by importing the appropriate estimator class from Scikit-Learn.

2. Choose model hyperparameters by instantiating this class with desired values.

3. Arrange data into a features matrix and target vector following the discussion above.

4. Fit the model to your data by calling the ``fit()`` method of the model instance.

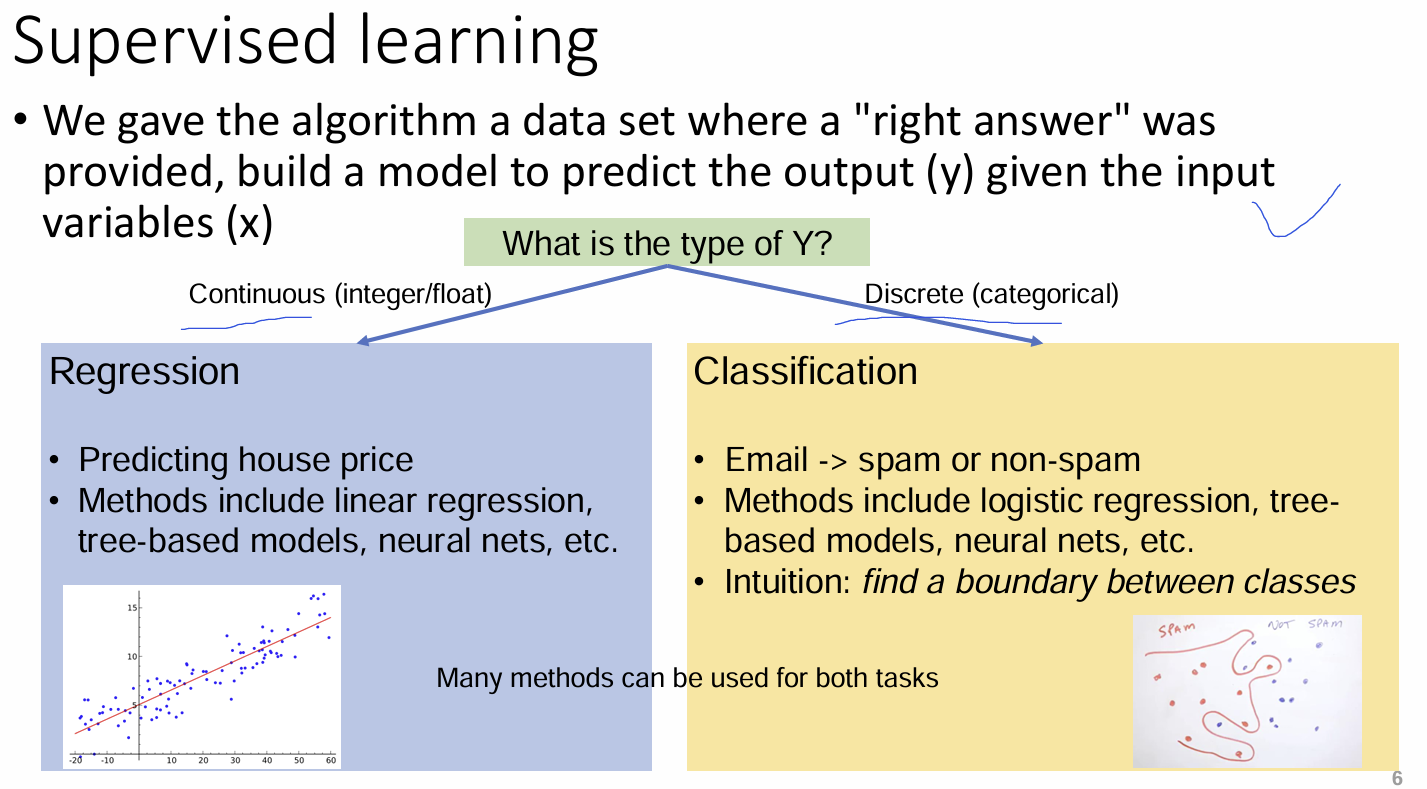
5. Apply the Model to new data:

   - For supervised learning, often we predict labels for unknown data using the ``predict()`` method.

   - For unsupervised learning, we often transform or infer properties of the data using the ``transform()`` or ``predict()`` method.

We will now step through several simple examples of applying supervised and unsupervised learning methods.

## Supervised learning



**Challenge of supervised learning: overfitting**

• ML is aimed at making accurate predictions for new unseen data.

• The model overfits the data, when the model is fitting the training data too well (accuracy of 100%), but does not generalise to new unseen data

• Overfitting is unavoidable: all machine learning methods tend to overfit, especially the complicated models.

• How to recognise overfitting? Use train-test split

• How to mitigate overfitting? Tune the hyperparameters carefully

• Workflow is crucial!

### DecisionTreeRegressor

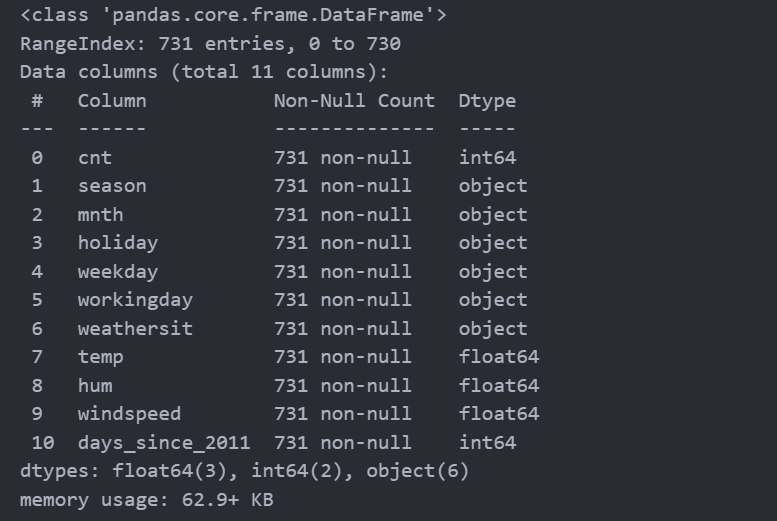
bike\_rental = pd.**read\_csv**('https://raw.githubusercontent.com/huanfachen/Spatial\_Data\_Science/main/Dataset/daily\_count\_bike\_rental.csv')

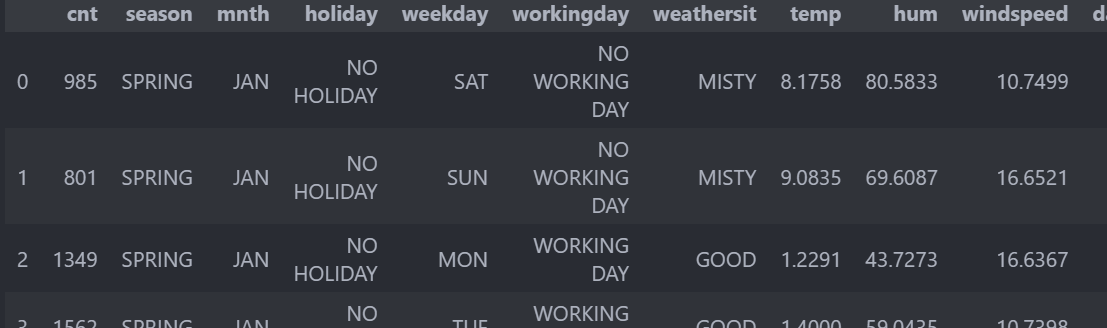
# drop the year variable as it is not useful

bike\_rental = bike\_rental.**drop**(['yr'], axis=1)

bike\_rental.**info**()

bike\_rental.**head**()





Converting categorical variables

bike\_rentail\_numeric = pd.**get\_dummies**(bike\_rental)

season、mnth、holiday、weekday、workingday、weathersit等被转换了，厉害。

bike\_rental\_final = bike\_rentail\_numeric.**drop**(['season\_SPRING', 'mnth\_JAN', 'holiday\_NO HOLIDAY', 'weekday\_MON', 'workingday\_WORKING DAY', 'weathersit\_GOOD'], axis=1) # K-1定律

Splitting data into random train and test subsets

random\_state\_split = 100

train\_x, test\_x, train\_y, test\_y = **train\_test\_split**(bike\_rental\_final.**drop**(['cnt'], axis = 1), bike\_rental\_final.cnt, random\_state=random\_state\_split)

 **train\_test\_split**:

* 是 scikit-learn 中的一个函数，用于将数据随机划分为训练集和测试集。
* 默认比例为 **75%训练集** 和 **25%测试集**。
* 如果希望修改训练集和测试集的比例，如：test\_size=0.3

 **输入参数**:

* bike\_rental\_final.drop(['cnt'], axis=1):
  + 去除目标变量 cnt，只保留特征数据作为输入。
* bike\_rental\_final.cnt:
  + 目标变量 cnt，表示自行车租赁量。
* random\_state=random\_state\_split:
  + 固定随机种子，确保拆分结果一致。

 **返回值**:

* train\_x: 训练集的特征数据。
* test\_x: 测试集的特征数据。
* train\_y: 训练集的目标变量数据。
* test\_y: 测试集的目标变量数据。

**初始化回归模型**：DecisionTreeRegressor 是一个用于回归任务的模型，适用于预测连续值。

from sklearn.tree import DecisionTreeRegressor

reg\_tree = DecisionTreeRegressor(random\_state=0)

reg\_tree.**fit**(train\_x, train\_y)

**print**("R2 on the training data:")

**print**(reg\_tree.**score**(X=train\_x, y=train\_y))

**print**("R2 on the testing data:")

**print**(reg\_tree.**score**(X=test\_x, y=test\_y))

R2 on the training data:

1.0

R2 on the testing data:

0.7229453566227575

It looks like the R2 on the testing data is much lower than that on the training data. This indicates the overfitting problem, meaning that the model fits very well to the training data but doesn't generalise well to unseen data.

from sklearn.metrics import **mean\_squared\_error**

**print**("RMSE on the training data:")

**print**(**mean\_squared\_error**(train\_y, reg\_tree.**predict**(train\_x), squared=False))

**print**("RMSE on the testing data:")

**print**(**mean\_squared\_error**(test\_y, reg\_tree.**predict**(test\_x), squared=False))

 **训练集 RMSE**：衡量模型拟合训练数据的精度。

 **测试集 RMSE**：衡量模型在新数据上的预测能力。

 通常关注测试集 RMSE。如果测试集 RMSE 大大高于训练集 RMSE，说明模型可能过拟合训练数据。*均方根误差（root-mean-square error）*

Tuning hyperparameters of the ML model

是指调整和优化机器学习模型的超参数（hyperparameters）以提升模型的性能。

**Holdout Validation vs Cross Validation**

|  |  |  |
| --- | --- | --- |
| 特性 | 保留验证 | Cross Validation交叉验证 |
| 数据利用率 | 部分数据用于训练，部分用于测试。 | 全部数据都被用于训练和测试。 |
| 评估稳定性 | 可能受到数据划分的随机性影响。 | 更稳定，评估结果基于多次测试的平均值。 |
| 计算成本 | 较低，只需训练模型一次。 | 较高，需要训练模型多次。 |
| 适用场景 | 数据量大，时间成本较高。 | 数据量较小，需要更稳健的评估。 |
| 典型比例或设置 | 一般按 80:20 或 70:30 分割。 | 常用 k=5 或 k=10，称为 5-fold 或 10-fold。 |

**Holdout Validation**

# 将训练集中的前 75% 行作为训练子集，后 25% 行作为验证子集

n\_row\_training = np.floor(train\_x.shape[0]\*0.75).astype(int) #四舍五入

ind\_train = list(range(n\_row\_training)) # 前 75% 的索引

ind\_val = list(range(n\_row\_training, train\_x.shape[0])) # 后 25% 的索引

# 候选的超参数值

hyperparameters = {'max\_depth':[10,20,30,40,50]}

# 定义一个随机种子用于保持结果可复现

randomState\_dt = 10000

#创建一个决策树回归器实例 dt

dt = DecisionTreeRegressor(random\_state=randomState\_dt)

# 使用网格搜索，结合自定义的训练集和验证集索引.

clf = GridSearchCV(dt, hyperparameters, cv=[(ind\_train, ind\_val)])

‘’’

GridSearchCV 是用于超参数调优的工具。

cv=[(ind\_train, ind\_val)]：自定义交叉验证，指定训练集和验证集的索引。

hyperparameters：定义网格搜索的参数范围（max\_depth）。

‘’’

clf.**fit**(train\_x, train\_y)

# 输出最佳超参数值和对应的验证集分数

**print** ("The best parameter value is: ")

**print** (clf.best\_params\_)

**print** ("The best score is: ")

**print** (clf.best\_score\_)

The best parameter value is:

{'max\_depth': 30}

The best score is:

0.7253171537913115

# 使用找到的最佳超参数重新训练决策树模型

dt\_final = DecisionTreeRegressor(max\_depth=clf.best\_params\_['max\_depth'], random\_state=randomState\_dt)

dt\_final.**fit**(train\_x, train\_y)

**# 评估模型在训练集和测试集上的分数**

**#score() 方法返回 R² 分数，用于衡量模型的拟合效果。**

**print**('The score on the training data:')

**print**(dt\_final.**score**(train\_x, train\_y))

**print**('The score on the testing data:')

**print**(dt\_final.**score**(test\_x, test\_y))

The score on the training data:

1.0

The score on the testing data:

0.7337776508078722

**Cross Validation**

# values of max\_depth

hyperparameters = {'max\_depth':[10,20,30,40,50]}

randomState\_dt = 10000

dt = DecisionTreeRegressor(random\_state=randomState\_dt)

# 使用 GridSearchCV 进行超参数调优，默认使用 5 折交叉验证

clf = GridSearchCV(dt, hyperparameters)

clf.**fit**(train\_x, train\_y)

# we can query the best parameter value and its accuracy score

**print** ("The best parameter value is: ")

**print** (clf.best\_params\_)

**print** ("The best score is: ")

**print** (clf.best\_score\_)

The best parameter value is:

{'max\_depth': 10}

The best score is:

0.7451947631296068

max\_depth\_range = [10,20,30,40,50]

# 使用 validation\_curve 绘制验证曲线，查看不同 max\_depth 对模型性能的影响

train\_scores, valid\_scores = **validation\_curve**(estimator=DecisionTreeRegressor(),

                                              X=train\_x, y=train\_y,

                                              param\_name="max\_depth",

                                              param\_range=max\_depth\_range,

                                              cv=5)

validation\_curve 是一个用于评估超参数（这里是 max\_depth）对模型性能影响的函数。它执行以下操作：

estimator=DecisionTreeRegressor()：使用决策树回归模型。

X=train\_x, y=train\_y：使用训练集 train\_x 和 train\_y 来训练模型。

param\_name="max\_depth"：超参数是 max\_depth，它控制决策树的最大深度。

param\_range=max\_depth\_range：提供 max\_depth 值的范围，评估这些值对模型性能的影响。

cv=5：使用 5 折交叉验证进行评估。即将数据分为 5 个部分，每次使用 4 个部分训练，剩余 1 个部分验证，轮流进行。

# plotting the validation curve

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

valid\_scores\_mean = np.mean(valid\_scores, axis=1)

valid\_scores\_std = np.std(valid\_scores, axis=1)

plt.**title**("Validation Curve with RF")

plt.**xlabel**(r"maximum tree depth")

plt.**ylabel**("R2 Score")

plt.**ylim**(0.6, 1.1)

lw = 2

plt.**semilogx**(max\_depth\_range, train\_scores\_mean, label="Training score",

             color="darkorange", lw=lw) # **semilogx**使用对数坐标绘制

plt.**fill\_between**(max\_depth\_range, train\_scores\_mean - train\_scores\_std,

                 train\_scores\_mean + train\_scores\_std, alpha=0.2,

                 color="darkorange", lw=lw) #填充

plt.**semilogx**(max\_depth\_range, valid\_scores\_mean, label="Cross-validation score",

             color="navy", lw=lw)

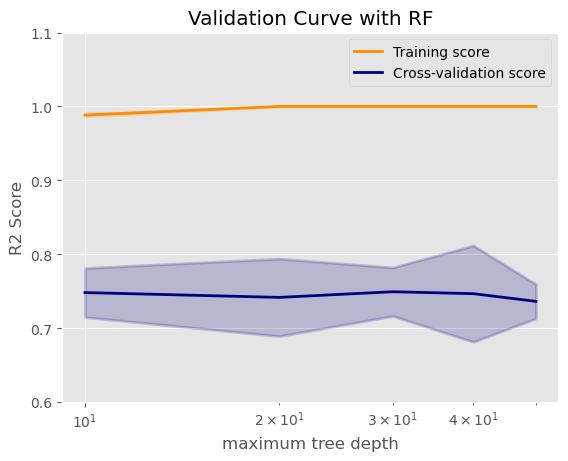
plt.**fill\_between**(max\_depth\_range, valid\_scores\_mean - valid\_scores\_std,

                 valid\_scores\_mean + valid\_scores\_std, alpha=0.2,

                 color="navy", lw=lw)

plt.**legend**(loc="best")

plt.**show**()



# 使用找到的最佳 max\_depth 值创建最终的决策树回归模型

dt\_final = DecisionTreeRegressor(max\_depth=clf.best\_params\_['max\_depth'], random\_state=randomState\_dt)

dt\_final.**fit**(train\_x, train\_y)

**print**('The score on the training data:')

**print**(dt\_final.**score**(train\_x, train\_y))

**print**('The score on the testing data:')

**print**(dt\_final.**score**(test\_x, test\_y))

The score on the training data:

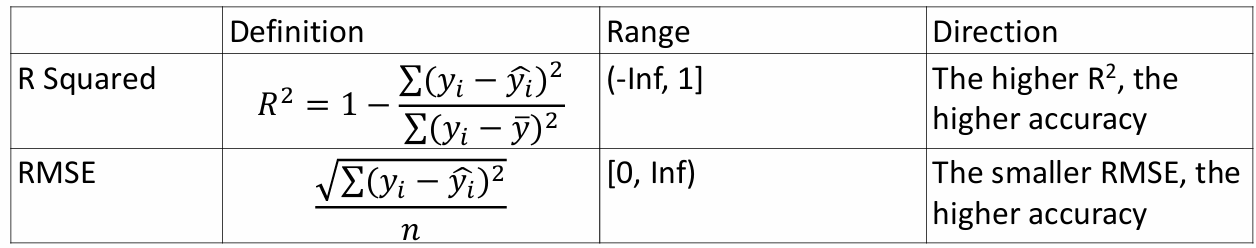
0.9890472695830228

The score on the testing data:

0.7350074821879649

## Metrics

### Metrics for regression



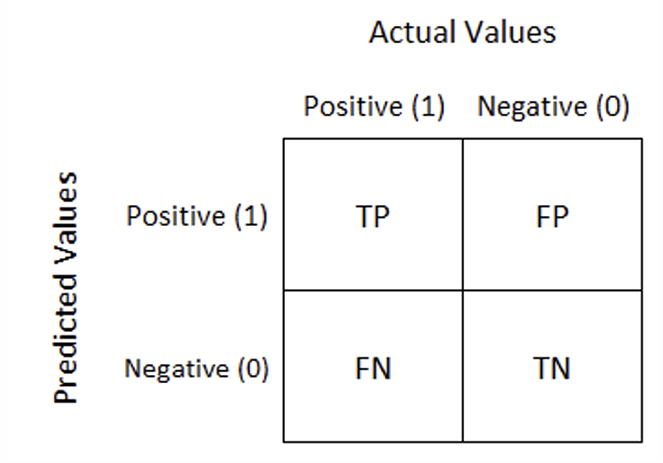
1. According to the definition, R2 is not the square of a value.

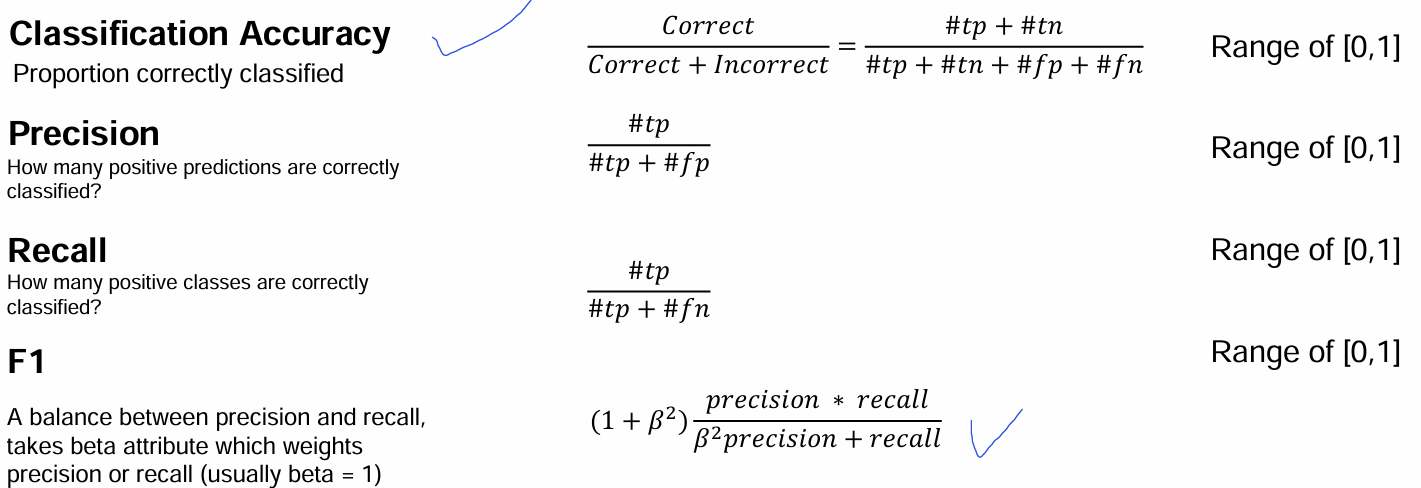
2. Two possibilities of a negative R2: the prediction algorithm is not suitable or not well tuned; the input variables are not related to y variable.

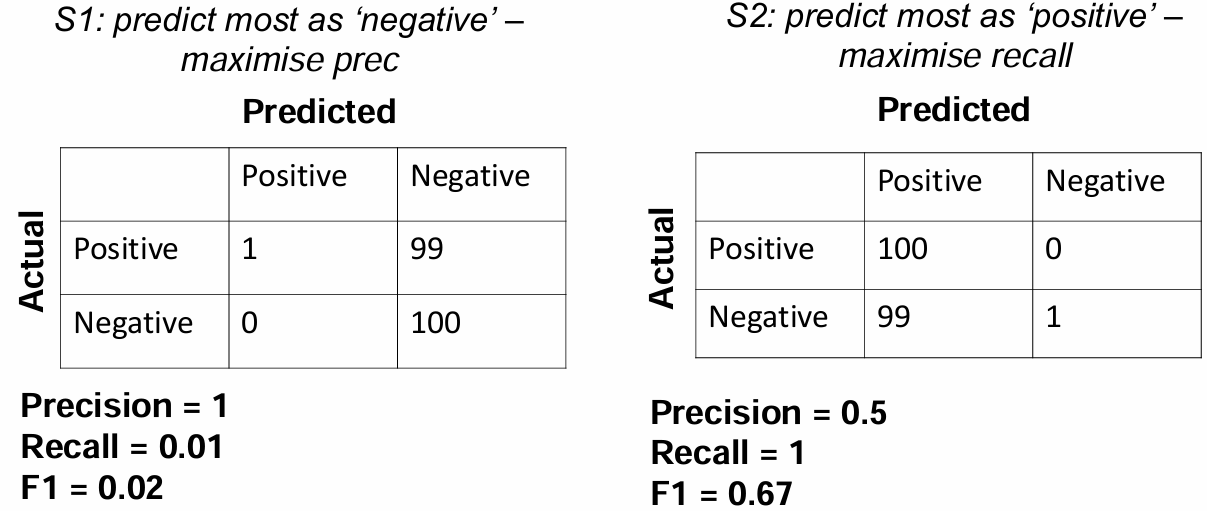
3. A special case of R2 is that linear regression has a R2 in the range of [0,1].

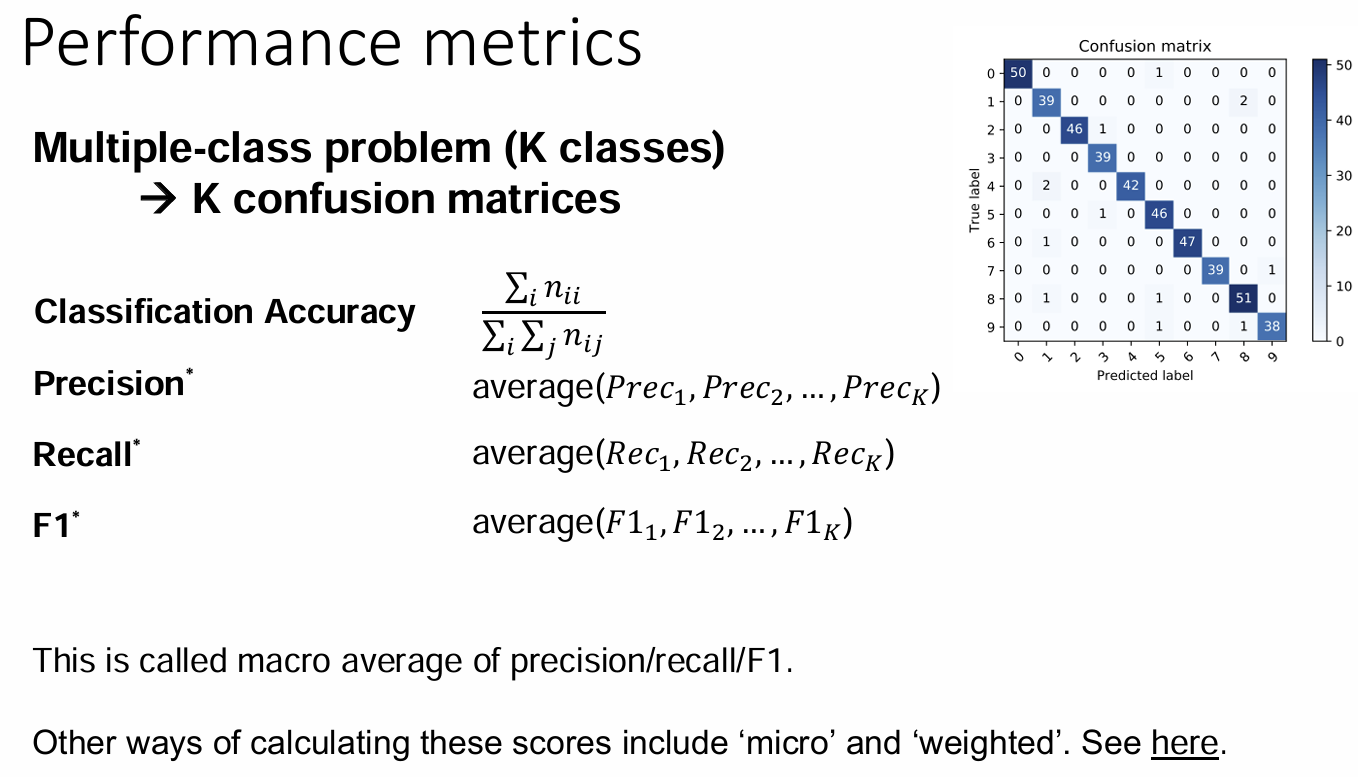
### Metrics for classification

Confusion Matrix: comparing predicted against observed classes









# Tree-based Methods

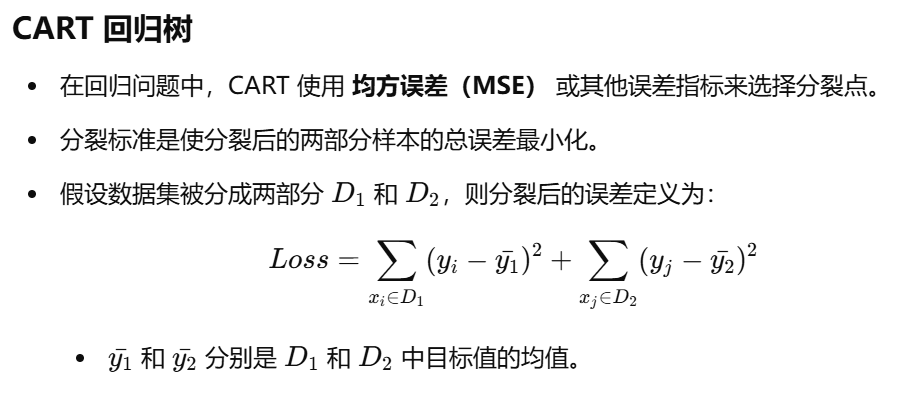
## Decision trees

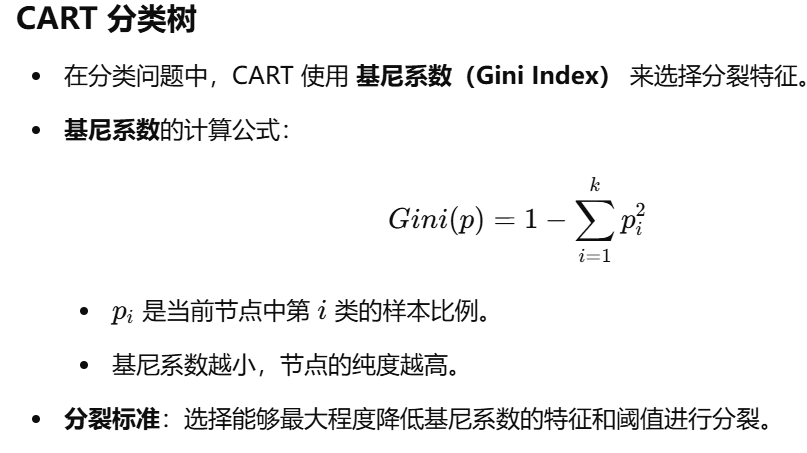
There are different types of decision trees (CART, ID3, others). We focus on the CART (**Classification and regression Tree 分类回归树**)

CART 是一种重要的决策树算法，因其支持分类和回归问题，且结构简单直观，被广泛用于各种场景。它在很多集成算法（如随机森林和梯度提升树）中也发挥了核心作用，是构建机器学习模型的基础工具之一。

**CART 的核心特点**

1. **二叉树结构**：
   * CART 决策树始终生成二叉树，每个节点只有两个子节点（左子树和右子树）。这是 CART 的一个显著特点，与一些可以生成多叉树的算法（如 ID3）不同。
2. **支持分类和回归**：
   * 分类问题：CART 会输出离散的类别标签。
   * 回归问题：CART 会输出连续的数值。
3. **递归分裂数据集**：
   * CART 使用特定的评价指标来选择最优划分特征和阈值，并不断地递归分裂数据集，直至满足停止条件。
4. **后剪枝**：
   * CART 使用后剪枝的方法来简化模型，避免过拟合。



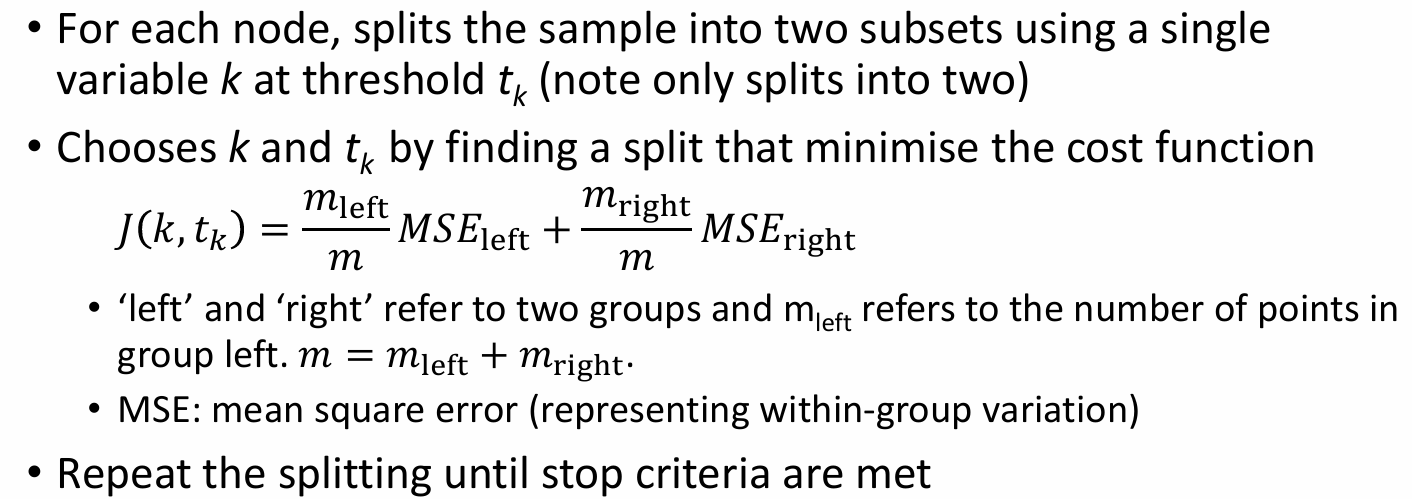


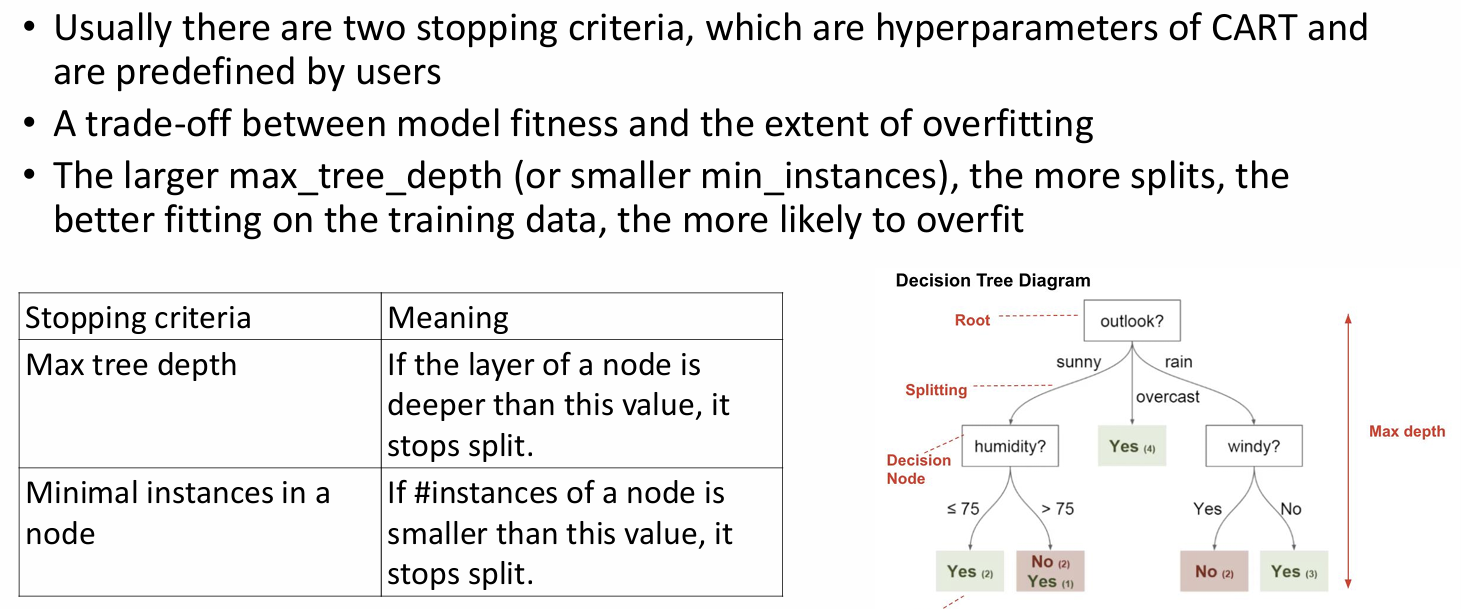
How to **train** a CART

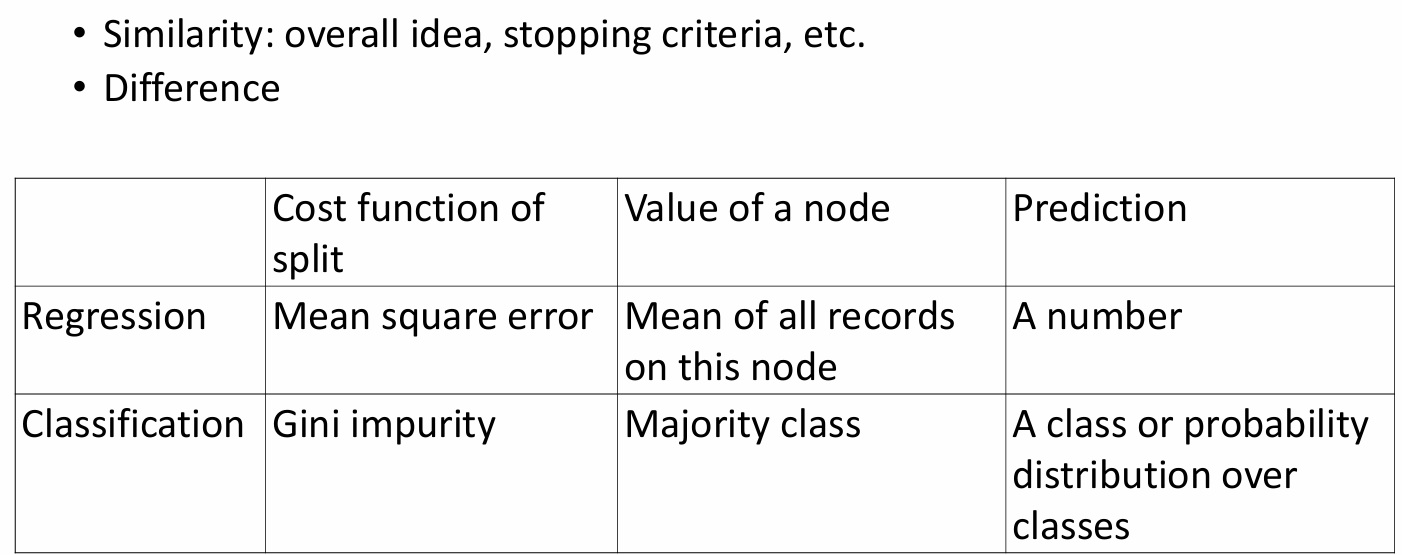
• Q1: how to decide which split to adopt? (metrics)

• Q2: when to stop the split? (the stopping criteria)

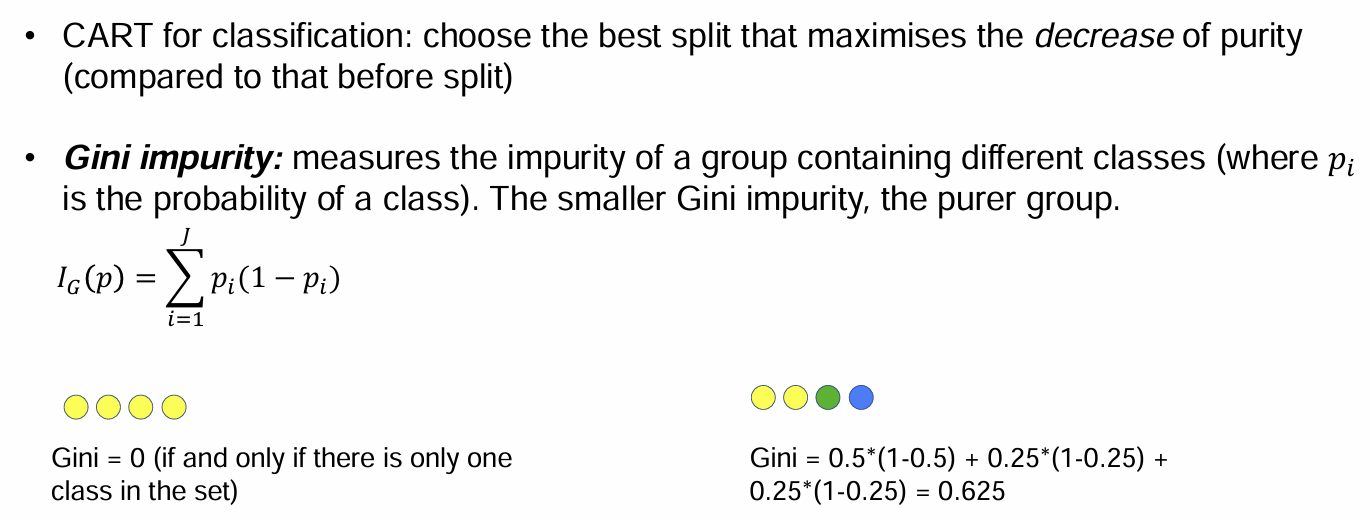
• Q3: what is the similarity and difference between CART for regression and for classification?

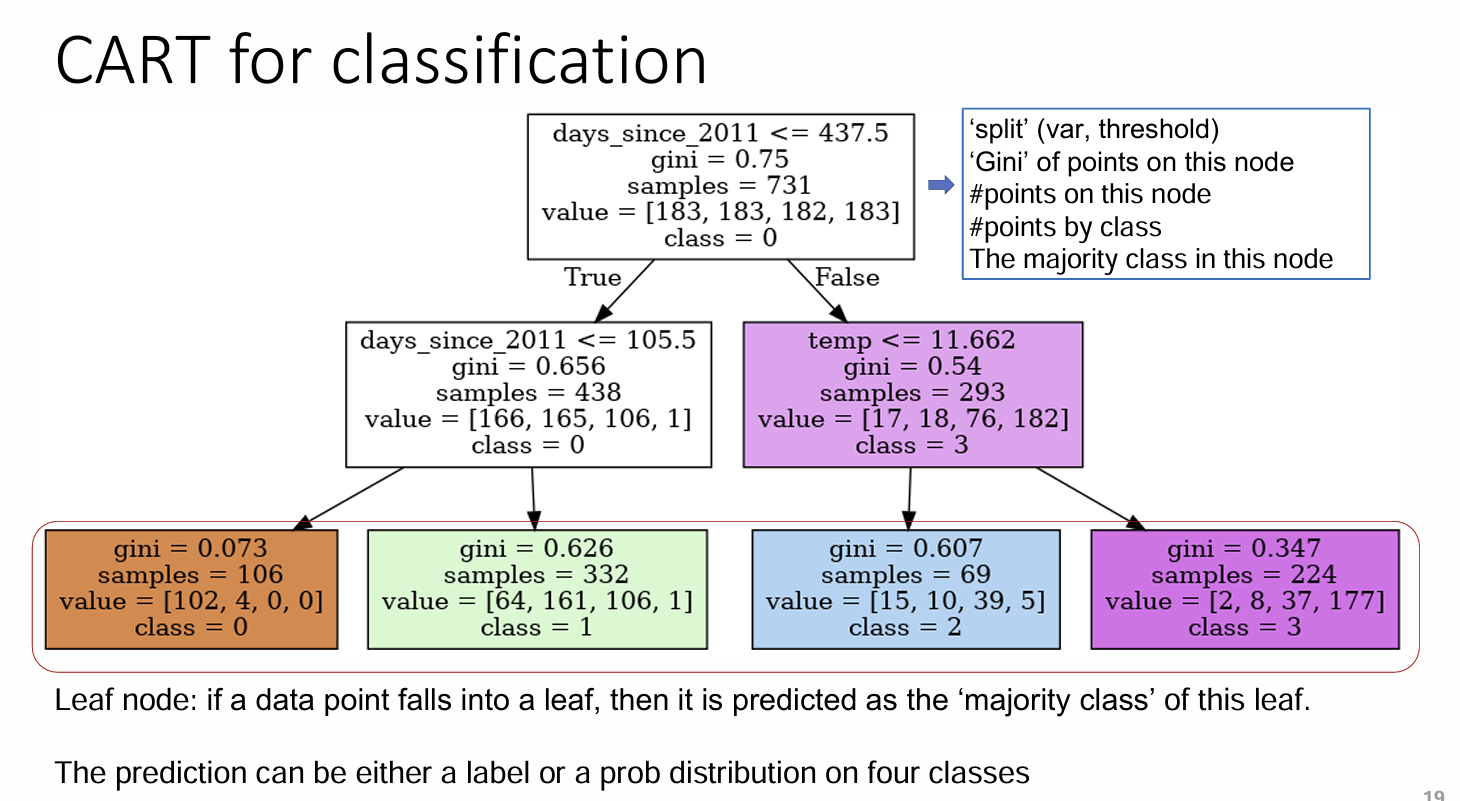
**Q1:**

**Q2:**

**Q3:** ****

**Gini impurity**





**Advantages of CART**

• Interpretability: relatively easy to understand

• Flexibility: no assumptions of data distribution and no transformations needed

**Disadvantages**

• Lack of smoothness: Slight changes in the predicators have a big impact on response

• Tendency of overfitting

**Key points**

• CART can be used for both regression and classification

• The issues associated with CART will be tackled by RF or GBDT

• It is uncommon to use CART to directly make predictions. Rather, CART is used to construct RF or GBDT

## Ensemble learning

集成学习：在统计学和机器学习中，集成学习方法使用多种学习算法来获得比单独使用任何单独的学习算法更好的预测性能。不像统计力学中的系综通常是无限的，机器学习集合仅由一组具体的有限的可替代模型组成。

CART is a good unit for ensemble learning

• Training a CART is relatively easy and cheap

• CART makes no assumptions on input data

Two common approaches of ensemble learning

• Bagging (random forest)

• Boosting (gradient boosting decision tree, GBDT)

### Random Forest

• RF is a collection of many different CARTs.

• Given an input, the prediction of RF is a combination (e.g. average or the majority votes) of the output of all trees

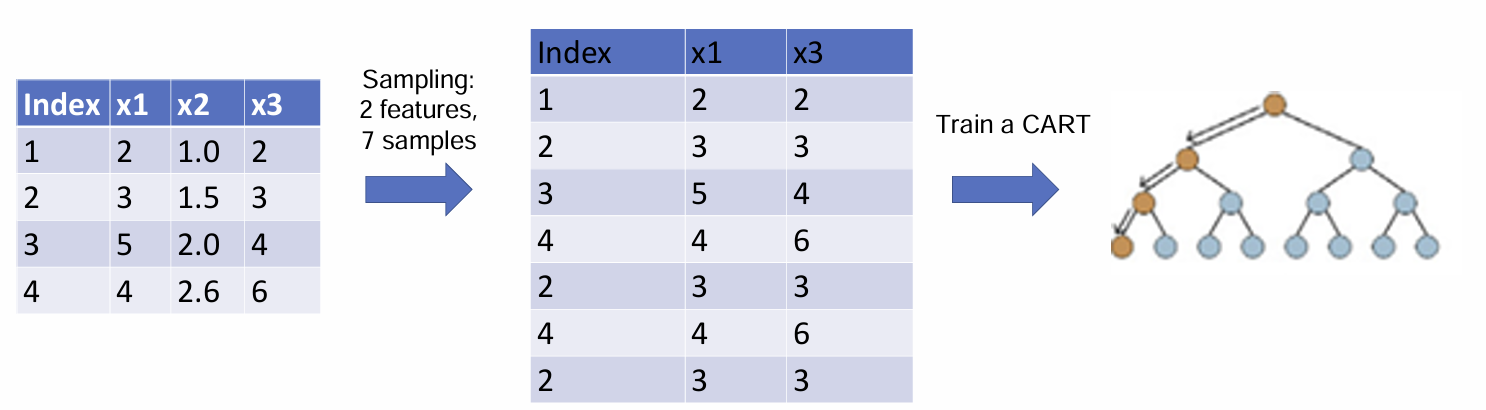
• Two techniques to grow different and diverse trees (the beauty of randomness)

1. Bagging (short for bootstrap aggregating): sampling instances (‘rows’)2. Random feature selection: sampling features (‘columns’)

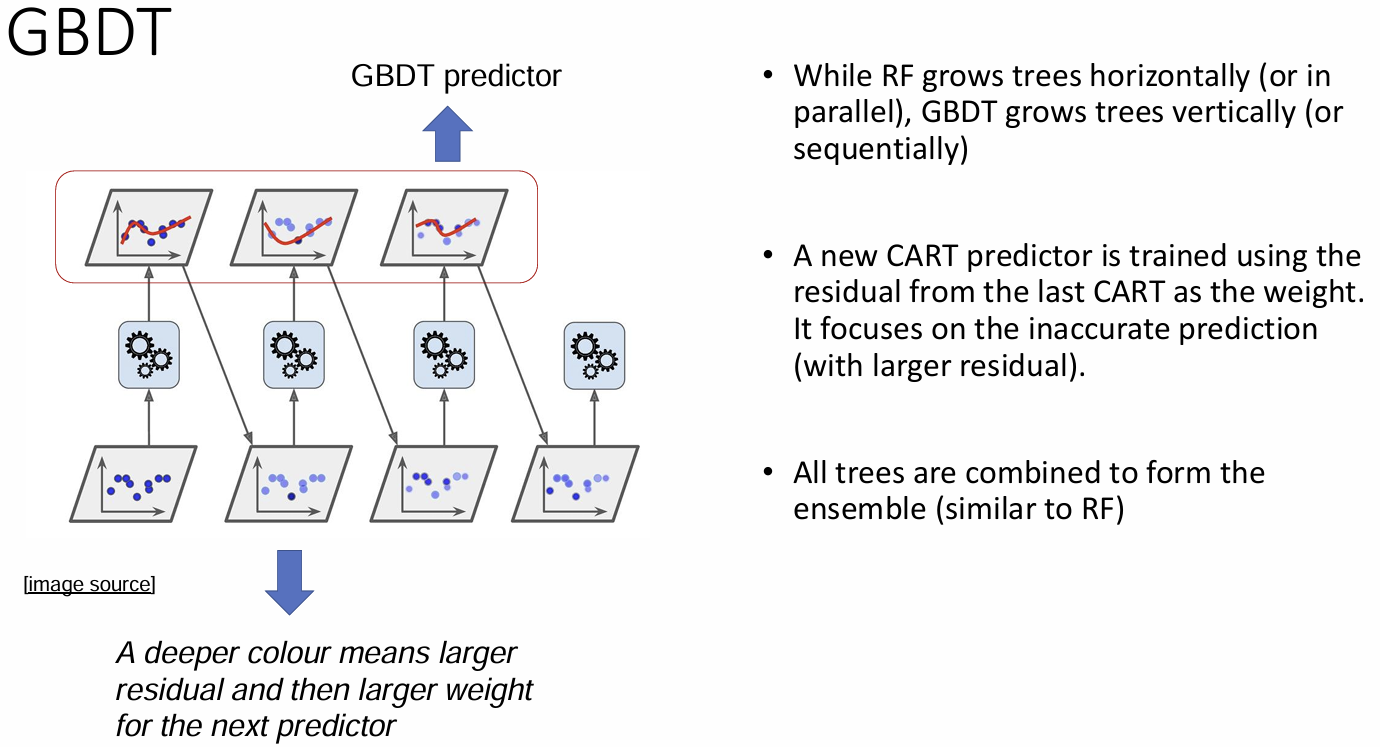
• As each CART sees different training data, the trees are different.

• Bootstrap: sampling with replacement. It guarantees that the sample has the same distribution as population; some instances may be sampled repeatedly.

• Example of bagging and random feature selection



### GBDT 梯度提升决策树



RF and GBDT

• Advantages

• No assumptions on data distribution

• Able to model non-linear relationship and feature interactions

• Good predictive performance (especially for tabular data)

• Good generalisation

• Disadvantages: not intuitive, although there are some interpretation methods

Model interpretation

• ‘Interpretation of ML models’ is an emerging field and there are many new methods coming out every year.

• One of the classic methods for interpreting tree-based models is **permutation feature importance (PFI)**

### Permutation feature importance (PFI)

• The idea is straightforward. We measure the importance of a feature by calculating

the increase in the model’s prediction error after permuting the feature.

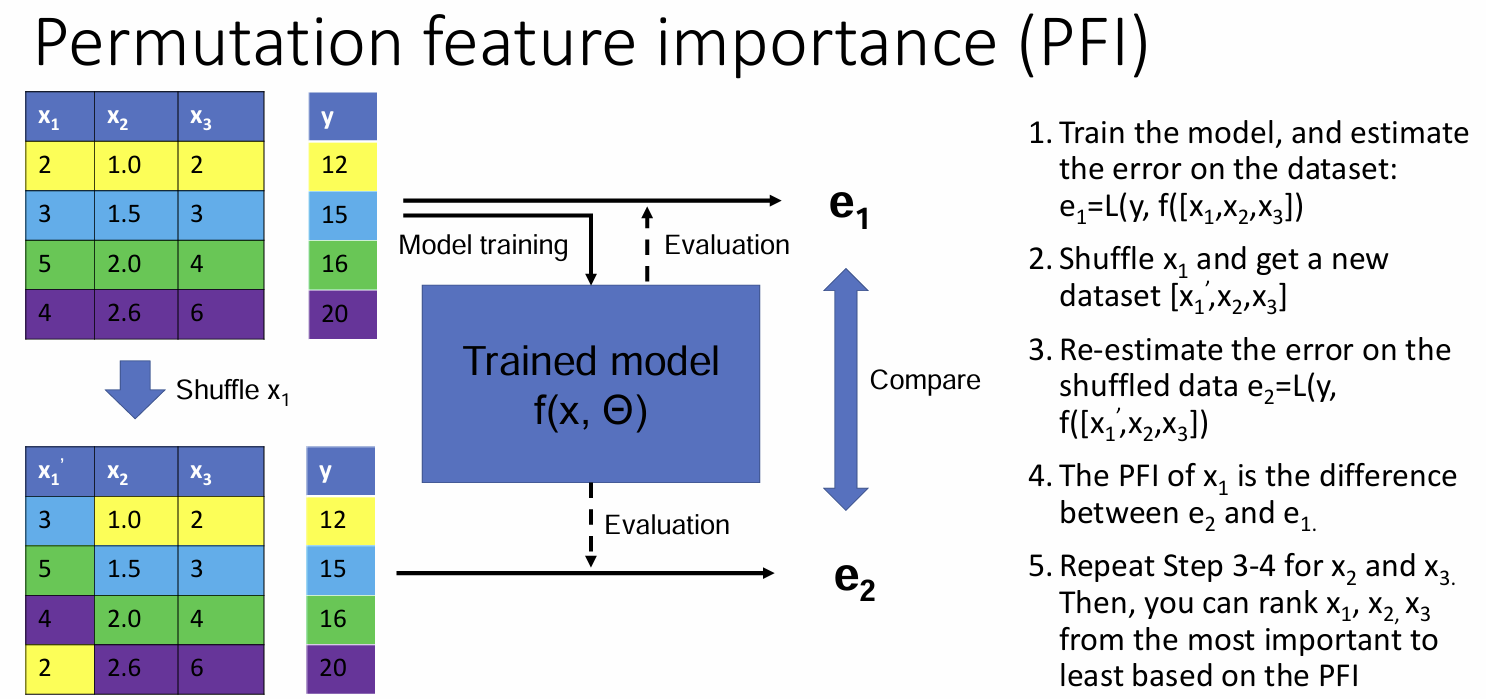
• A feature is “important” if shuffling its values increases the model error, because in

this case the model relied on the feature for the prediction.

• In contrast, a feature is “unimportant” if shuffling its values leaves the model error

unchanged, because in this case the model ignored the feature for the prediction.

• This method is model-agnostic: Applicable to linear regression, CART, RF, GBDT, etc; Applicable to regression and classification task



## Code

Build a CART

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

from sklearn.model\_selection import **train\_test\_split**, GridSearchCV, **validation\_curve**

from sklearn.metrics import **mean\_squared\_error**

# CART

from sklearn.tree import DecisionTreeRegressor

# random forest

from sklearn.ensemble import RandomForestRegressor

# feature importance

import rfpimp

# xgboost

import xgboost

from xgboost import XGBRegressor

pd.**set\_option**('display.max\_rows', 300)

pd.options.display.float\_format = '{:40,.4f}'.**format** # specifies default number format to 4 decimal places

plt.style.**use**('ggplot') # specifies that graphs should use ggplot styling

%matplotlib inline

bike\_rental = pd.**read\_csv**('https://raw.githubusercontent.com/huanfachen/Spatial\_Data\_Science/main/Dataset/daily\_count\_bike\_rental.csv')

# drop the year variable as it is not useful

bike\_rental = bike\_rental.**drop**(['yr'], axis=1)

bike\_rentail\_numeric = pd.**get\_dummies**(bike\_rental)

bike\_rental\_final = bike\_rentail\_numeric.**drop**(['season\_SPRING', 'mnth\_JAN', 'holiday\_NO HOLIDAY', 'weekday\_MON', 'workingday\_WORKING DAY', 'weathersit\_GOOD'], axis=1)

random\_state\_split = 100

train\_x, test\_x, train\_y, test\_y = **train\_test\_split**(bike\_rental\_final.**drop**(['cnt'], axis = 1), bike\_rental\_final.cnt, random\_state=random\_state\_split)

**Tuning the hyperparameters using cross validation**

# values of max\_depth and min\_samples\_split

hyperparameters = {'max\_depth':[10,20,30,40,50], 'min\_samples\_split':[2,4,6,8,10]}

/\*min\_samples\_split\* refers to the minimum number of samples required to split a node. If a node contains less samples than min\_samples\_split, this node becomes a leaf node and would not split.\*/

randomState\_dt = 10000

dt = DecisionTreeRegressor(random\_state=randomState\_dt)

# cv=5 by default, which means 5-fold cross-validation

clf = GridSearchCV(dt, hyperparameters)

clf.**fit**(train\_x, train\_y)

# we can query the best parameter value and its accuracy score

**print** ("The best parameter value is: ")

**print** (clf.best\_params\_)

**print** ("The best score is: ")

**print** (clf.best\_score\_)

The best parameter value is:

{'max\_depth': 10, 'min\_samples\_split': 8}

The best score is:

0.7549920033546205

dt\_final = DecisionTreeRegressor(max\_depth=clf.best\_params\_['max\_depth'], min\_samples\_split=clf.best\_params\_['min\_samples\_split'], random\_state=randomState\_dt)

dt\_final.**fit**(train\_x, train\_y)

**print**("R2 on the training data:")

**print**(dt\_final.**score**(X=train\_x, y=train\_y))

**print**("R2 on the testing data:")

**print**(dt\_final.**score**(X=test\_x, y=test\_y))

R2 on the training data:

0.9727089518051003

R2 on the testing data:

0.757715812742848

**print**("RMSE on the training data:")

**print**(**mean\_squared\_error**(train\_y, dt\_final.**predict**(train\_x), squared=False))

**print**("RMSE on the testing data:")

**print**(**mean\_squared\_error**(test\_y, dt\_final.**predict**(test\_x), squared=False))

RMSE on the training data:

322.91814665797057

RMSE on the testing data:

922.8954973299212

# some attributes of the tree

**print**("Tree depth:{}".**format**(dt\_final.**get\_depth**()))

**print**("Number of leaves:{}".**format**(dt\_final.**get\_n\_leaves**()))

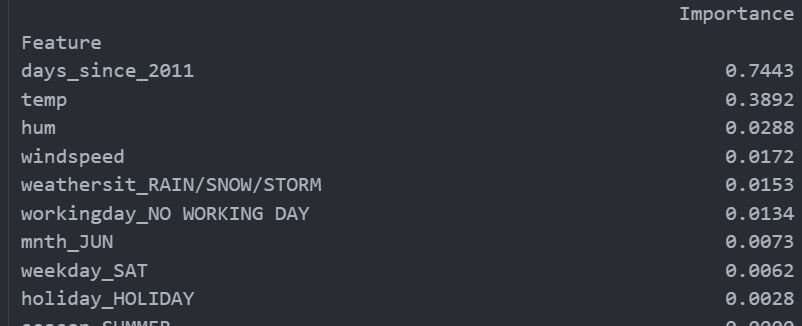
Tree depth:10

Number of leaves:114

**rfpimp** package for permutation features importance (**PFI**)

imp = rfpimp.importances(dt\_final, test\_x, test\_y)

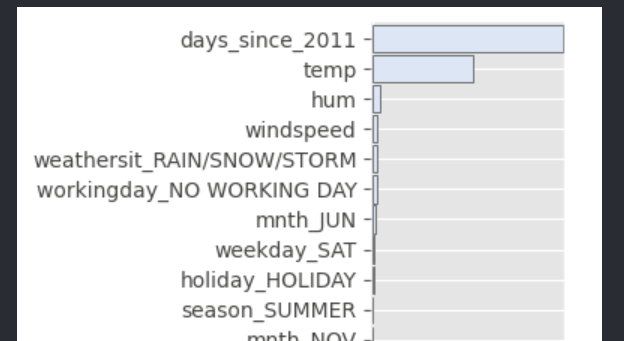
**print**(imp)



## Here is a barplot of the feature importance

viz = rfpimp.plot\_importances(imp)

viz.view()



**Build a random forest. 和CART基本上一摸一样**

# values of max\_depth and min\_samples\_split

hyperparameters = {'max\_depth':[10,20,30,40,50], 'min\_samples\_split':[2,4,6,8,10]}

randomState\_dt = 10000

rf = RandomForestRegressor(random\_state=randomState\_dt)

# cv=5 by default, which means 5-fold cross-validation

clf = GridSearchCV(rf, hyperparameters)

clf.**fit**(train\_x, train\_y)

# we can query the best parameter value and its accuracy score

**print** ("The best parameter value is: ")

**print** (clf.best\_params\_)

**print** ("The best score is: ")

**print** (clf.best\_score\_)

rf\_final = RandomForestRegressor(max\_depth=clf.best\_params\_['max\_depth'], min\_samples\_split=clf.best\_params\_['min\_samples\_split'], random\_state=randomState\_dt)

rf\_final.**fit**(train\_x, train\_y)

**print**("R2 on the training data:")

**print**(rf\_final.**score**(X=train\_x, y=train\_y))

**print**("R2 on the testing data:")

**print**(rf\_final.**score**(X=test\_x, y=test\_y))

**print**("RMSE on the training data:")

**print**(**mean\_squared\_error**(train\_y, rf\_final.**predict**(train\_x), squared=False))

**print**("RMSE on the testing data:")

**print**(**mean\_squared\_error**(test\_y, rf\_final.**predict**(test\_x), squared=False))

imp = rfpimp.importances(rf\_final, test\_x, test\_y)

**print**(imp)

viz = rfpimp.plot\_importances(imp)

viz.view()

Build XGBoost

# values of max\_depth and min\_samples\_split

hyperparameters = {'max\_depth':[10,20,30,40,50], 'n\_estimators':[50,100,150,200,250]}

/\*n\_estimators: The number of trees in the ensemble. Equivalent to the number of boosting rounds. The value must be an integer greater than 0. Default is 100 \*/

randomState\_xgb = 125

xgb = XGBRegressor(random\_state=randomState\_xgb)

# cv=5 by default, which means 5-fold cross-validation

gscv\_xgb = GridSearchCV(xgb, hyperparameters)

gscv\_xgb.**fit**(train\_x, train\_y)

# we can query the best parameter value and its accuracy score

**print** ("The best parameter value is: ")

**print** (gscv\_xgb.best\_params\_)

**print** ("The best score is: ")

**print** (gscv\_xgb.best\_score\_)

xgb\_final = XGBRegressor(max\_depth=gscv\_xgb.best\_params\_['max\_depth'], n\_estimators=gscv\_xgb.best\_params\_['n\_estimators'], random\_state=randomState\_xgb)

xgb\_final.fit(train\_x, train\_y)

**print**("R2 on the training data:")

**print**(xgb\_final.score(X=train\_x, y=train\_y))

**print**("R2 on the testing data:")

**print**(xgb\_final.score(X=test\_x, y=test\_y))

**print**("RMSE on the training data:")

**print**(**mean\_squared\_error**(train\_y, xgb\_final.predict(train\_x), squared=False))

**print**("RMSE on the testing data:")

**print**(**mean\_squared\_error**(test\_y, xgb\_final.predict(test\_x), squared=False))

imp = rfpimp.importances(xgb\_final, test\_x, test\_y) # permutation

**print**(imp)

viz = rfpimp.plot\_importances(imp)

viz.view()

**Comparing three models**

# create a list of models

list\_name\_models = ['CART', 'RF', 'XGBoost']

# use the models from above

list\_reg\_models = [dt\_final, rf\_final, xgb\_final]

dict\_models = dict()

for name, model in zip(list\_name\_models, list\_reg\_models):

    dict\_models[name] = [model.score(train\_x, train\_y), model.score(test\_x, test\_y), model.score(train\_x, train\_y) - model.score(test\_x, test\_y)]

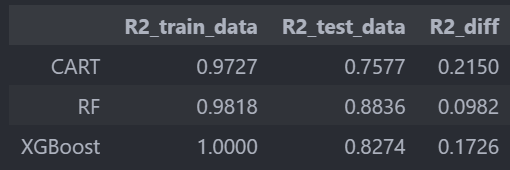
# transform dict\_models to dataframe

df\_models = pd.DataFrame.**from\_dict**(dict\_models, orient='index', columns=['R2\_train\_data', 'R2\_test\_data', 'R2\_diff'])

df\_models

# you can then export df\_models as a csv file and use it in MS Excel or Word

# df\_models.to\_csv(...)



**结论：**

The above result shows that the random forest model has a higher R2 on the testing data than CART and XGBoost, as well as a smaller R2 difference between training and testing data.

This indicates that the random forest model is less subject to the overfitting issue and has a better generalisation. The XGBoost also has a less extent of overfitting than CART.

For this reason, it is uncommon to use CART to directly make predictions. Rather, CART is used to construct RF or GBDT.

# Analysis Workflow

## Basic error analysis

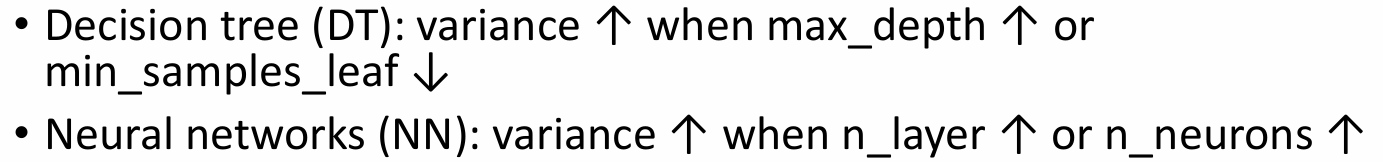
There is a tradeoff between a model's ability to minimise bias and variance (**called bias-variance tradeoff**) 偏见方差权衡

Why important? These two errors are linked to over- and under-fitting. Understanding these two types of error can help us diagnose model results and mitigate over- or under-fitting.

**Bias**: how close the model can get to the true relationship between the predictors and the outcome

通常，线性回归模型的偏差大于决策树或神经网络，因为它有很强的假设关于y和x之间的线性关系，将无法拟合当实际关系是非线性的。

**Variance**: 如果我们使用不同的训练数据集进行估计，模型将发生变化的量。Given an algorithm, the higher complexity, the higher variance.



Note: we can’t directly compare variance of a CART and NN

**Bias-variance trade-off**

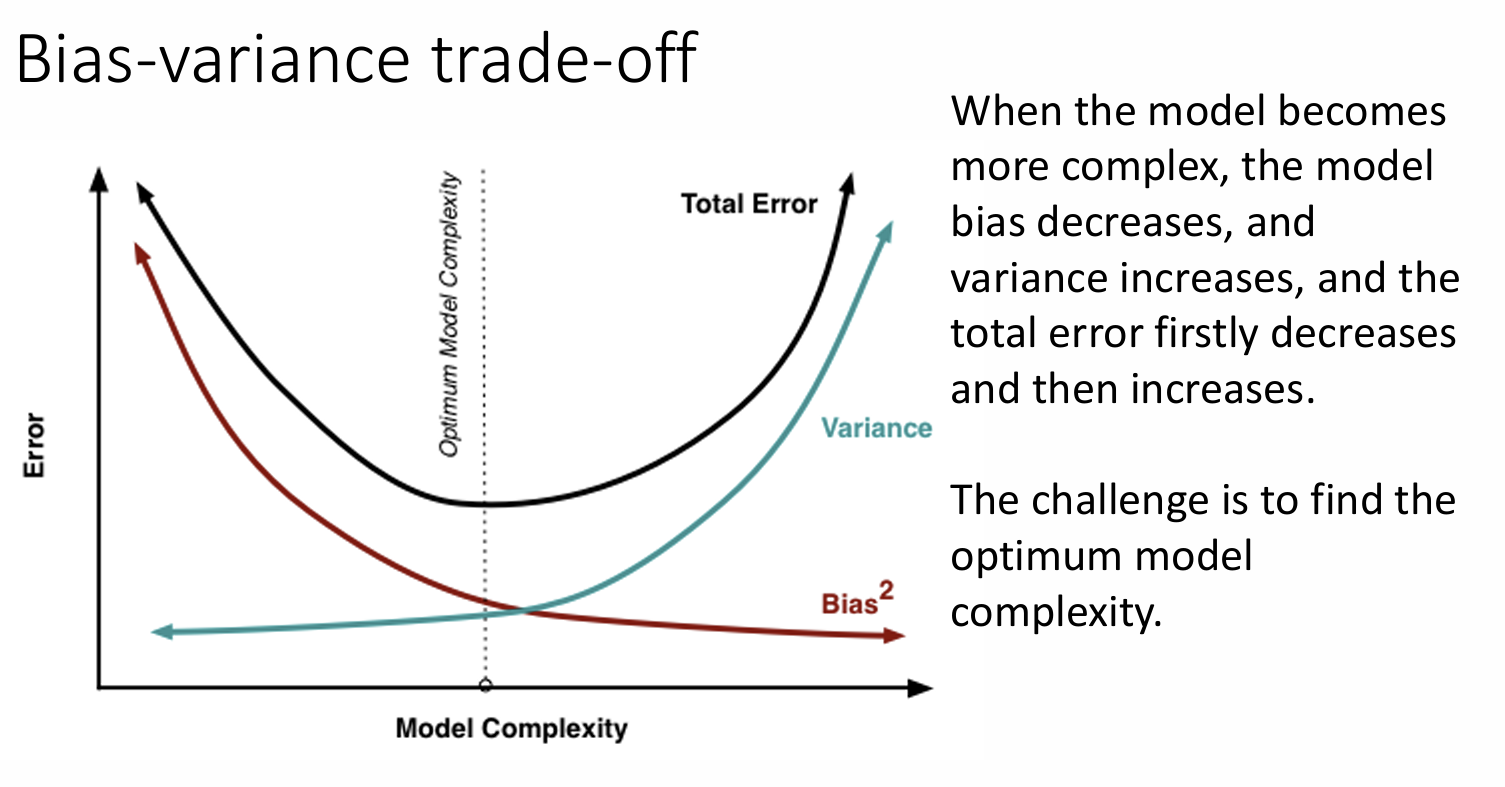
• Ideally, we want a model with low bias and low variance, but this is very challenging in practice. In fact, this could be described as the goal of applied machine learning for a given predictive problem.

• The bias-variance trade-off

• Reducing the bias can easily be achieved by increasing the variance.

• Reducing the variance can easily be achieved by increasing the bias.

• This is a good conceptual framework for thinking how to choose models and model configuration (or hyperparameters).



在实践中，人们不会直接使用偏差和方差。相反, 他们经常使用训练误差和测试误差来诊断故障模型（过拟合或欠拟合）

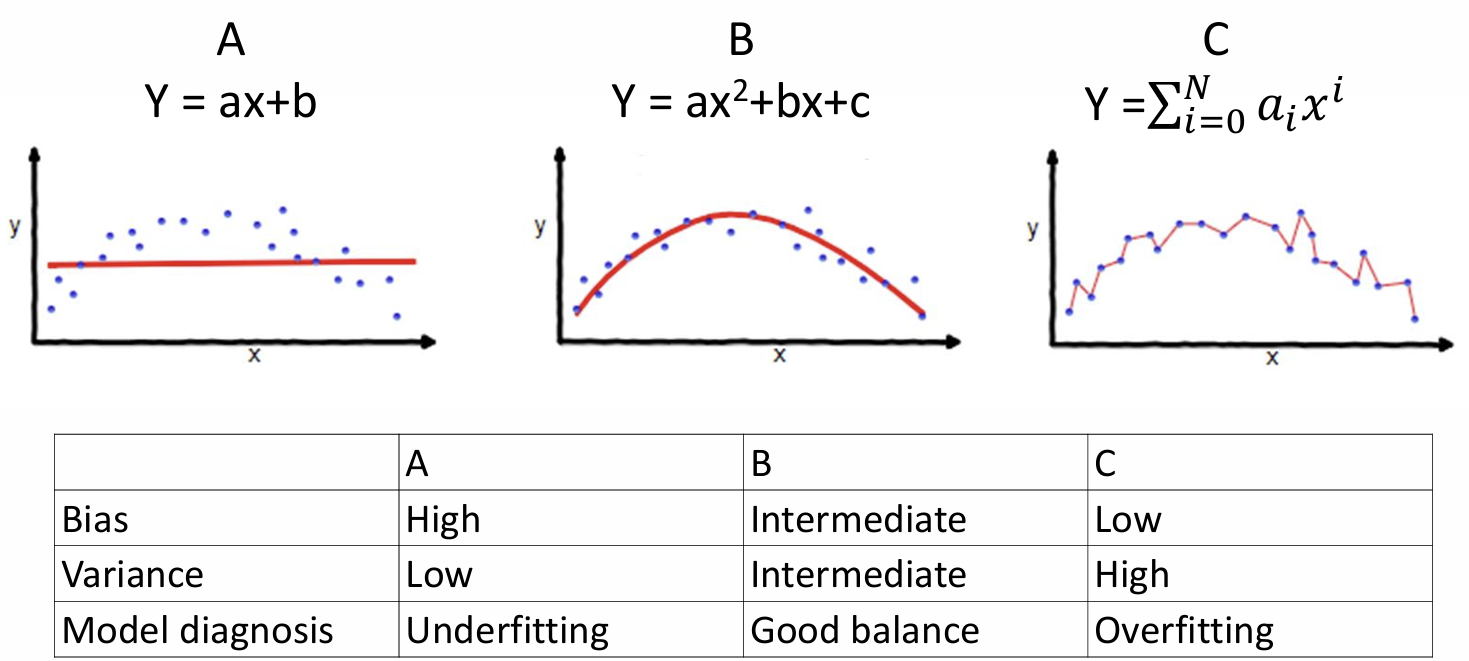
**Training error**: the error on the training data.

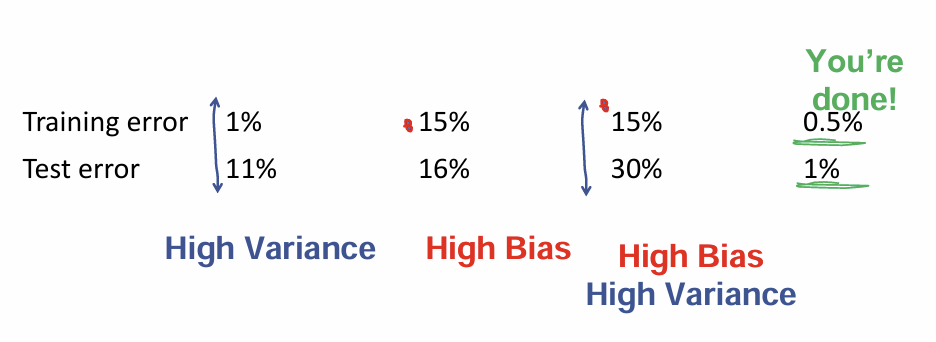
**Testing error**: the error on the testing data.

**Error diff**= testing error – training error

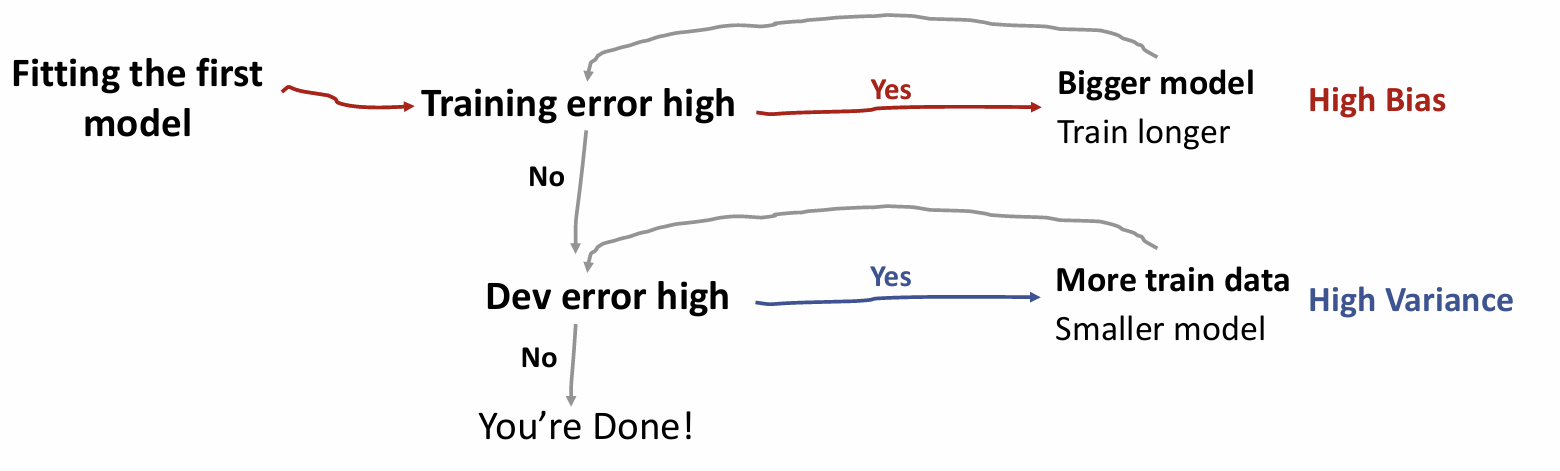


**INCORRECT statement**: training error is an estimate of bias; the error difference is an estimate of variance, but these are **roughly** of the same direction: when training error is high, the model bias is high; when testing error is high, the model variance is high.





## Machine learning strategy



### Techniques for reducing bias

**Increase model complexity**

• Replace a simple linear regression model with a more flexible model, such as random forest or deep learning.

• Add more neurons or layers in a deep learning model.

**Modify input features**

• Inspect检查 your training data to understand which examples your model is not doing well on.

• See if you can modify data features to eliminate these errors.

### Techniques for reducing variance

**Add more training data**

• This is the simplest and the most reliable way to address variance, so long as you have access to significantly more data.

**Reduce model size/complexity**

• Replace a large neural network with a random forest.

• Add regularization to your neural network.

• Decrease neural network size.

**Feature selection to decrease number/type of input features**

• This technique might help with variance problems, but it might also increase bias.

• In deep learning, there has been a shift away from feature selection, and we are now more likely to give all the data to the algorithm and let the algorithm sort out which ones to use.

## Data leakage

数据泄露：在统计学和机器学习中，模型训练过程中使用了在预测时不应该可用的信息，导致在生产环境中运行时，预测得分（指标）高估了模型的效用。

**Techniques to minimise data leakage**

1. **Hold back a testing dataset** for final sanity check of your developed models

2. **Perform data preparation without using testing dataset**. If you are using cross validation, perform data preparation within your cross validation folds

3. **Add Noise**. Add random noise to input data to try and smooth out the effects of possibly leaking variables.

4. **Use Pipelines**. Heavily use pipeline architectures that allow a sequence of data preparation steps to be performed within cross validation folds, such as the caret package in R and Pipelines in scikit-learn. (note: pipeline is not about parallel computing; it is a way of combining multiple steps in data analysis)

## Code

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

import sklearn

from sklearn.model\_selection import **train\_test\_split**, GridSearchCV, **validation\_curve**

from sklearn.metrics import **root\_mean\_squared\_error**, **r2\_score**

# preprocessors

from sklearn.preprocessing import StandardScaler, OneHotEncoder

from sklearn.impute import SimpleImputer

from sklearn.compose import ColumnTransformer

# pipeline

from sklearn.pipeline import Pipeline

# linear regression

from sklearn.linear\_model import LinearRegression

# CART

from sklearn.tree import DecisionTreeRegressor

# random forest

from sklearn.ensemble import RandomForestRegressor

# xgboost

import xgboost

from xgboost import XGBRegressor

pd.**set\_option**('display.max\_rows', 300)

pd.options.display.float\_format = '{:40,.4f}'.**format**

plt.style.**use**('ggplot')

%matplotlib inline

bike\_rental = pd.**read\_csv**('https://raw.githubusercontent.com/huanfachen/Spatial\_Data\_Science/main/Dataset/daily\_count\_bike\_rental.csv')

# drop the year variable as it is not useful

bike\_rental = bike\_rental.**drop**(['yr'], axis=1)

bike\_rental.**isnull**().**sum**()

random\_state\_split = 100

train\_x, test\_x, train\_y, test\_y = **train\_test\_split**(bike\_rental.**drop**(['cnt'], axis = 1), bike\_rental.cnt, random\_state=random\_state\_split)

#axis = 1 参数指定了删除操作应作用于列（axis=0 则作用于行）。

**The preprocessor of a pipeline**

# 数值特征的预处理步骤

numeric\_transformer = Pipeline(steps=[

       ('imputer', SimpleImputer(strategy='mean'))

      ,('scaler', StandardScaler())

])

# 分类特征的预处理步骤

categorical\_transformer = Pipeline(steps=[

       ('imputer', SimpleImputer(strategy='constant'))

      ,('encoder', OneHotEncoder(drop='first'))

])

SimpleImputer 是 scikit-learn 中的一个类，用于处理缺失值。它可以用指定的策略（如均值、中位数或众数）来填补缺失值。

StandardScaler 是 scikit-learn 中的一个类，用于标准化特征。它通过减去均值并除以标准差来将特征缩放到均值为0、标准差为1的分布。

OneHotEncoder 是 scikit-learn 中的一个类，用于将分类特征转换为一组二进制（0 或 1）特征。每个类别对应一个二进制列。

在 Pipeline 中，单引号中的名字并不是固定的。它们是用户自定义的名称，用于标识流水线中的每个步骤。这些名称可以是任意字符串，但通常会选择具有描述性的名称，以便更容易理解每个步骤的作用。

如：imputer\_step = pipeline.named\_steps['imputer']

numeric\_features = ['temp', 'hum', 'windspeed', 'days\_since\_2011']

categorical\_features = ['season', 'mnth', 'holiday', 'weekday', 'workingday', 'weathersit']

preprocessor = ColumnTransformer(

   transformers=[

    ('numeric', numeric\_transformer, numeric\_features)

   ,('categorical', categorical\_transformer, categorical\_features)

])

ColumnTransformer 是 scikit-learn 中的一个类，用于对不同的列应用不同的预处理步骤。它允许对数据框中的不同列进行不同的转换。

**Adding an estimator to a pipeline**

pipeline = Pipeline(steps = [

   ('preprocessor', preprocessor),

   ('regressor',DecisionTreeRegressor())

])

cart\_model = pipeline.**fit**(train\_x, train\_y)

**print**("RMSE on the training data:")

**print**(**root\_mean\_squared\_error**(train\_y, cart\_model.**predict**(train\_x)))

**print**("RMSE on the testing data:")

**print**(**root\_mean\_squared\_error**(test\_y, cart\_model.**predict**(test\_x)))

**print**("R2 on the training data:")

**print**(**r2\_score**(train\_y, cart\_model.**predict**(train\_x)))

**print**("R2 on the testing data:")

**print**(**r2\_score**(test\_y, cart\_model.**predict**(test\_x)))

**Using pipeline for hyperparameter tuning**

cart\_pipeline = Pipeline(steps = [

  ('preprocessor', preprocessor),

  ('regressor', DecisionTreeRegressor(random\_state=123))

])

# grid\_params is the range of each hyperparameter

grid\_params = {

  'regressor\_\_max\_depth': [10,20,30,40,40],

  'regressor\_\_min\_samples\_split': [2,4,6,8,10]

}

One trick here: when specifying the grid\_params, the name of each hyperparameter should be in the format of [estimator]\_\_[hyperparameter], where the first part of [estimator] is the name of estimator in the Pipeline object, the second part is **two underscores**, and the third part is the hyperparameter in the model.

search = GridSearchCV(cart\_pipeline, grid\_params)

search.**fit**(train\_x, train\_y)

**print**("Best R2 Score: ", search.best\_score\_)

**print**("Best Params: ", search.best\_params\_)

Best R2 Score: 0.7578489598345406

Best Params: {'regressor\_\_max\_depth': 10, 'regressor\_\_min\_samples\_split': 8}

regressors = {

    'Linear': LinearRegression(),

    'CART': DecisionTreeRegressor(),

    'RF': RandomForestRegressor(),

    'XGB': XGBRegressor()

}

# a dict to store the R2 of training and testing data

dict\_results = dict()

for name, regressor in regressors.**items**():

    pipeline = Pipeline(steps = [

               ('preprocessor', preprocessor)

              ,('regressor', regressor)

           ])

    model = pipeline.**fit**(train\_x, train\_y)

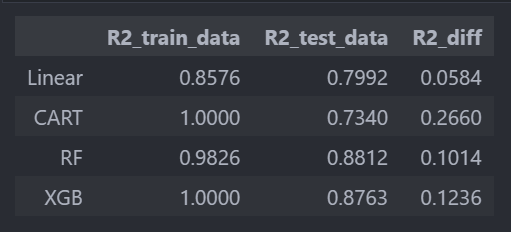
    predictions = model.**predict**(test\_x)

    dict\_results[name] = [model.**score**(train\_x, train\_y), model.**score**(test\_x, test\_y), model.**score**(train\_x, train\_y) - model.**score**(test\_x, test\_y)]

# transform dict\_models to dataframe

df\_models = pd.DataFrame.**from\_dict**(dict\_results, orient='index', columns=['R2\_train\_data', 'R2\_test\_data', 'R2\_diff'])

df\_models



1. The linear model is underfitting, as it has a relatively low R2 on the training data.

2. The CART may be subject to overfitting, as it has a relatively low R2 on the testing data.

3. Compared with the CART, the random forest and XGBoost model are less subject to overfitting, as their R2 score on the testing data is higher than CART. These two models have better performance than linear model or CART.

**总结对比**

| **模型** | **原理** | **适用于** | **优点** | **缺点** |
| --- | --- | --- | --- | --- |
| **线性回归 (Linear Regression)** | 线性关系，最小二乘法 | 线性数据 | 计算快、可解释性强 | 假设数据线性，易受异常值影响 |
| **决策树回归 (CART)** | 递归二分，基于均值划分数据 | 非线性数据，小数据集 | 直观、可解释性好 | 易过拟合，对噪声敏感 |
| **随机森林回归 (RF)** | 多个决策树投票，Bagging | 高维数据、非线性问题 | 抗过拟合、稳定 | 计算较慢，难以解释 |
| **XGBoost 回归 (XGB)** | 梯度提升，迭代学习 | 大数据、复杂特征交互 | 高精度、自动特征选择 | 计算量大，调参复杂 |

**推荐选择**

* 如果数据是线性可分的 ➝ 线性回归
* 如果数据较小且非线性较强 ➝ 决策树
* 如果数据较复杂，想要更稳健的模型 ➝ 随机森林
* 如果数据量大，追求高精度 ➝ XGBoost（基于**梯度提升决策树**）

1. **LinearRegression**：
   * 线性回归模型通常没有太多的超参数需要调优。可以考虑以下参数：
     + fit\_intercept：是否计算截距项。默认值为 True。
     + normalize：是否在回归前标准化数据。默认值为 False。
2. **DecisionTreeRegressor (CART)**：
   * max\_depth：树的最大深度。控制树的复杂度，防止过拟合。
   * min\_samples\_split：内部节点再划分所需的最小样本数。控制树的生长。
   * min\_samples\_leaf：叶子节点所需的最小样本数。控制树的生长。
   * max\_features：寻找最佳分割时考虑的最大特征数。可以是整数、浮点数、字符串 auto、sqrt、log2 或 None。
3. **RandomForestRegressor**：
   * n\_estimators：森林中树的数量。更多的树通常会提高性能，但也会增加计算成本。
   * max\_depth：树的最大深度。控制每棵树的复杂度。
   * min\_samples\_split：内部节点再划分所需的最小样本数。
   * min\_samples\_leaf：叶子节点所需的最小样本数。
   * max\_features：寻找最佳分割时考虑的最大特征数。
   * bootstrap：是否在构建树时使用自助法（bootstrap）抽样。
4. **XGBRegressor**：
   * n\_estimators：提升树的数量。
   * learning\_rate：每个树的权重缩减系数。较小的值通常需要更多的树。
   * max\_depth：树的最大深度。
   * min\_child\_weight：子节点中最小的样本权重和。用于控制过拟合。
   * subsample：用于训练模型的子样本比例。减少过拟合。
   * colsample\_bytree：每棵树使用的特征比例。减少过拟合。
   * gamma：节点分裂所需的最小损失减少。用于控制过拟合。

regressors = {

    'Linear': (LinearRegression(), {

        'regressor\_\_fit\_intercept': [True, False],

        'regressor\_\_normalize': [True, False]

    }),

    'CART': (DecisionTreeRegressor(), {

        'regressor\_\_max\_depth': [10, 20, 30, 40, 50],

        'regressor\_\_min\_samples\_split': [2, 4, 6, 8, 10],

        'regressor\_\_min\_samples\_leaf': [1, 2, 4]

    }),

    'RF': (RandomForestRegressor(), {

        'regressor\_\_n\_estimators': [50, 100, 200],

        'regressor\_\_max\_depth': [10, 20, 30],

        'regressor\_\_min\_samples\_split': [2, 4, 6],

        'regressor\_\_min\_samples\_leaf': [1, 2, 4],

        'regressor\_\_bootstrap': [True, False]

    }),

    'XGB': (XGBRegressor(), {

        'regressor\_\_n\_estimators': [50, 100, 200],

        'regressor\_\_learning\_rate': [0.01, 0.1, 0.2],

        'regressor\_\_max\_depth': [3, 6, 9],

        'regressor\_\_min\_child\_weight': [1, 3, 5],

        'regressor\_\_subsample': [0.6, 0.8, 1.0],

        'regressor\_\_colsample\_bytree': [0.6, 0.8, 1.0],

        'regressor\_\_gamma': [0, 0.1, 0.2]

    })

}

# Artificial Neural Networks

在监督学习（Supervised Learning）中，features（特征）**和**labels（标签）分别指：

**Features（特征）**：输入数据，也叫做**自变量（predictors）**，用于预测目标变量。

**Labels（标签）**：输出数据，也叫做**因变量（target）**，是模型要学习预测的目标。

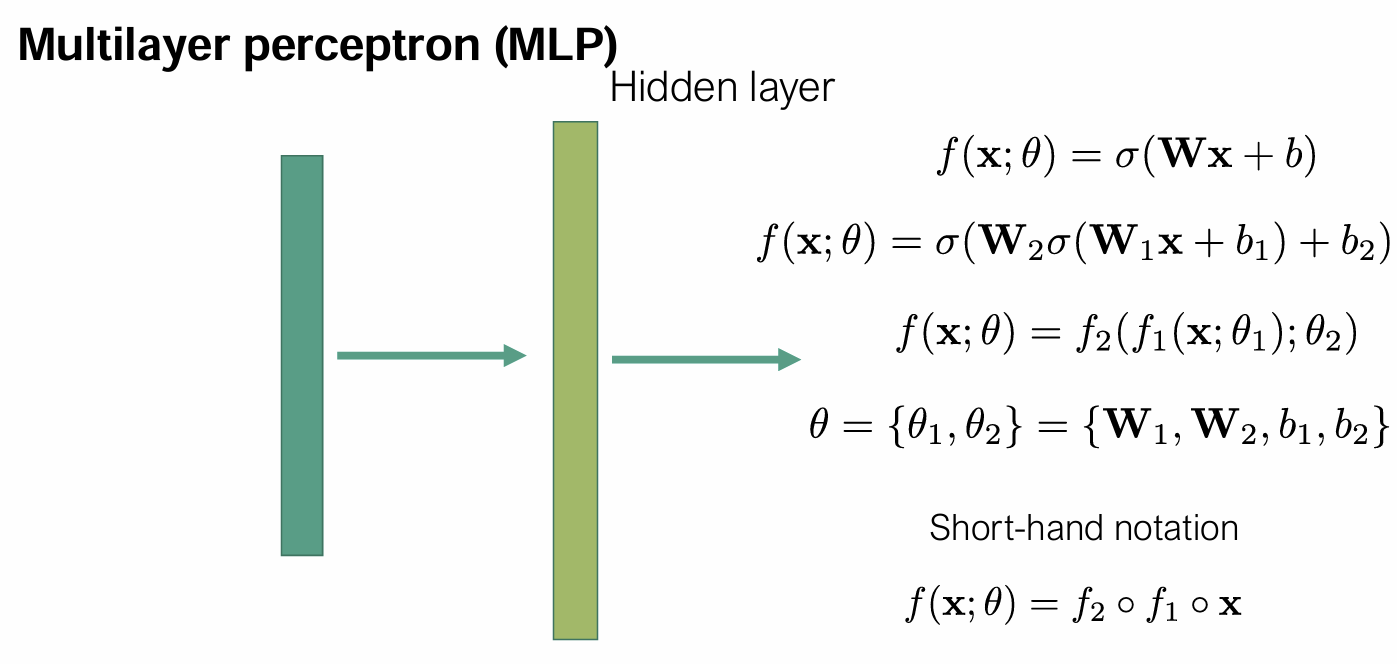
**Basic concepts** - Three sets: Training, validation and test

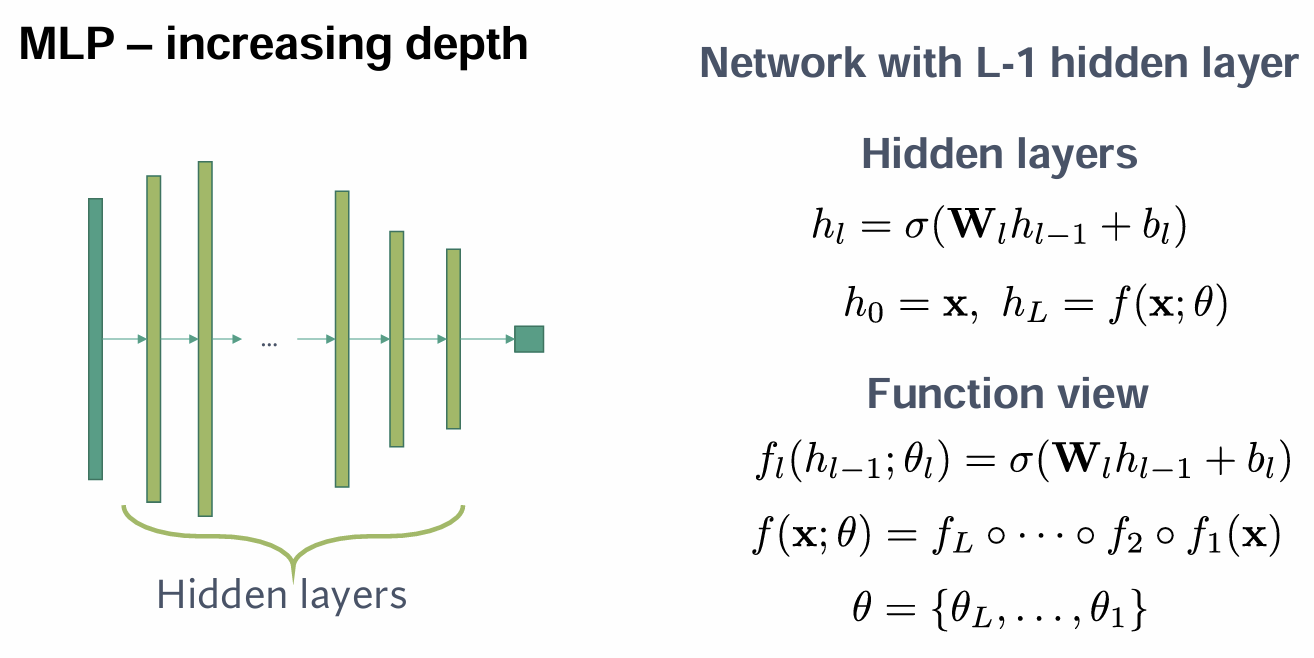
1. Determining the best model parameters A. Training set

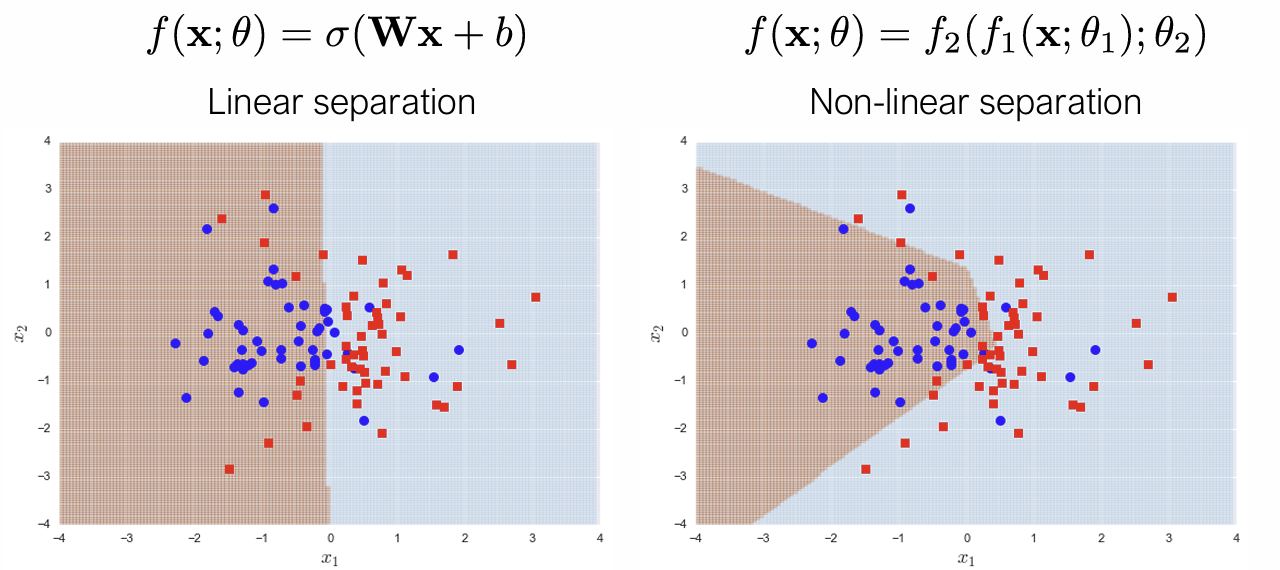
2. Determining the hyper-parameters B. Validation set

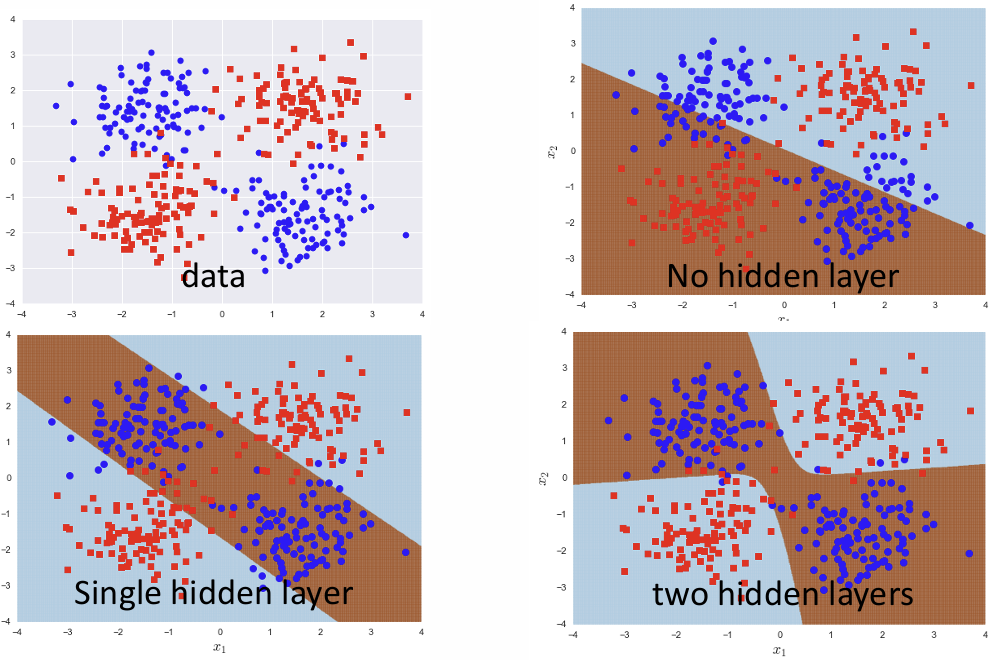
3. Estimating generalization accuracy C. Test set











这里的 **d** 代表的是**数据的原始维度（dimensionality）**，即原始特征的数量。



d1=d（仅非线性变换）

维度不变，只是对数据进行了非线性变换（例如通过激活函数、核方法等）。

d1 >d（映射到更高维度）

**增加维度**，通常通过某种映射方法（如**核方法**、特征扩展、深度学习**隐含层**）。

d1<d（降维）

**降低数据维度**，例如 PCA（主成分分析）、Autoencoder（自动编码器）等方法。

**MLP – notes on depth 多层感知机（MLP, Multi-Layer Perceptron）**

• No hidden layer – perceptron / logistic regression / linear

• Increasing depth

•Allows more complicated decision boundaries

• More powerful models

• Leads to larger number of model parameters

• Needs more samples to fit reliably

• May become difficult to train



**求导的目的**

在**深度学习**中，求导（计算梯度）主要用于**优化模型**，即调整神经网络的权重，使得**损失函数（Loss）最小化**。

神经网络的训练通常使用 **梯度下降（Gradient Descent）** 或其变种（如 Adam、RMSprop 等），核心步骤就是计算 **损失函数对模型参数的梯度**，然后\*\*反向传播（Backpropagation）\*\*来更新权重。

## Code

# ! pip install fastai==1.0.61

from fastai.basics import \*

from fastai.gen\_doc.nbdoc import \*

import fastai

from fastai.vision import \*

Let's begin with our sample of the MNIST\_TINY dataset.

We use the `untar\_data` to download data and the `get\_transforms` to create a list of flip, rotate, zoom, warp, lighting transforms on the images.

mnist = untar\_data(URLs.MNIST\_TINY)

tfms = get\_transforms(do\_flip=False)

**print**(URLs.MNIST\_TINY + '.tgz')#to see where the data downloaded from.

mnist #check where the data is saved in the local computer.

data = (ImageList.from\_folder(mnist)

        .split\_by\_folder()

        .label\_from\_folder()

        # .transform(tfms, size=32)

        .databunch(num\_workers=0)

        .normalize(imagenet\_stats))

 **从 mnist 文件夹加载图像**。

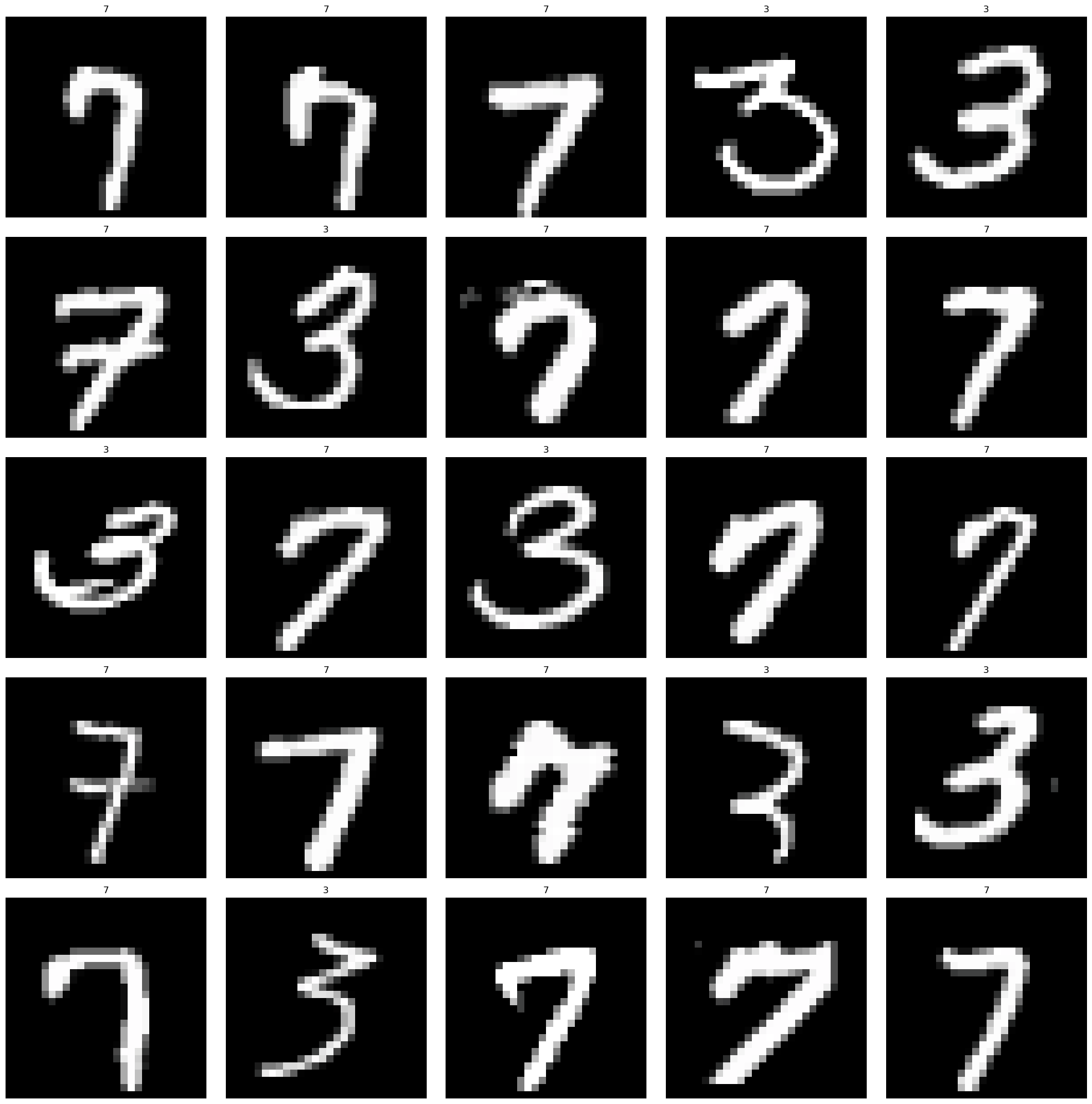
 **按照文件夹划分训练集和验证集**。

 **根据文件夹名称生成标签**（适用于分类任务）。

 **转换为 DataBunch 格式，方便 fastai 训练**。

 **归一化数据**，提高模型效果。

data.show\_batch()



data.show\_batch(rows=3, figsize=(4,4)) #阔以控制

learn = cnn\_learner(data, models.resnet18, metrics=accuracy)

cnn\_learner() 是 fastai 提供的快速创建 CNN 训练器的函数。

参数解析：

- data：数据加载器，通常是 ImageDataBunch 或 DataLoaders。

- models.resnet18：使用 ResNet-18 作为预训练模型（还有resnet34, resnet50）

- metrics=accuracy：使用准确率作为评价指标。

learn.fit\_one\_cycle(cyc\_len=50,max\_lr=1e-2)

fit\_one\_cycle()：使用 "1-cycle" 训练策略，在一定学习率范围内训练模型。

cyc\_len=50：训练 50 轮 (epochs)。

max\_lr=1e-2：最大学习率为 0.01。

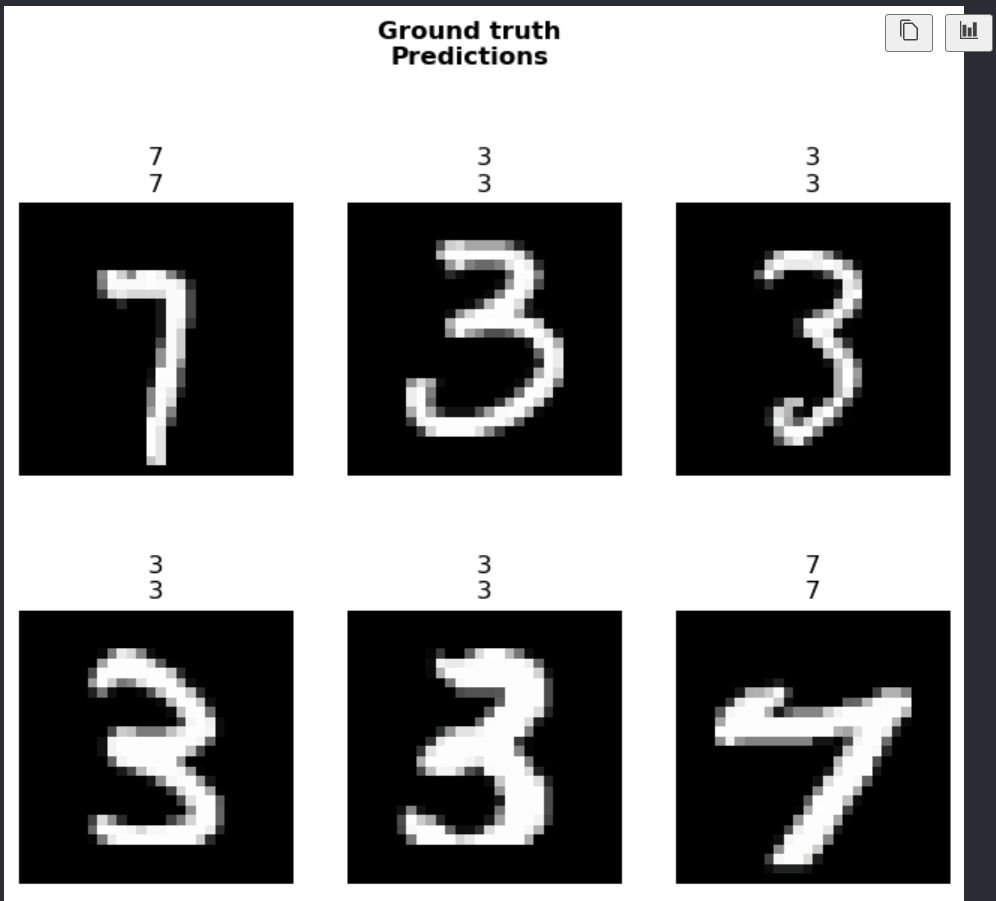
learn.save('mini\_train')

 保存训练好的模型，文件名为 'mini\_train'（会保存在 models 目录）。

 之后可以用 learn.load('mini\_train') 来加载模型。

learn.show\_results()

learn.show\_results(ds\_type=DatasetType.Train, rows=4, figsize=(8,10))

****