

ML Methods Overview

STAT3009 Recommender Systems

by Ben Dai (CUHK)
on Department of Statistics

» Recall: RS

- * **Training dataset:** [userID, itemID, rating]
- * **Testing dataset:** [userID, itemID, ?]
- * **Evaluation:** Given a testing index set Ω^{te} (set of user-item pairs we want to predict),

$$RMSE = \left(\frac{1}{|\Omega^{\text{te}}|} \sum_{(u,i) \in \Omega^{\text{te}}} (\hat{r}_{ui} - r_{ui})^2 \right)^{1/2}.$$

- * **Goal:** Find predicted ratings $(\hat{r}_{ui})_{(u,i) \in \Omega^{\text{te}}}$ such that **minimizes RMSE**

This is typical Machine Learning (ML) task.

» Machine learning (ML): RS

Using ML methods to build RS:

- Step 1. Introduce a model with some **parameters**
- Step 2. Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set** (replace **test data** in Evaluation by **training data**)
- Step 3. Use the estimated model to **predict**
 - Idea **Learning from Data**: A model that performs well on the **training set** tends to perform well on the **testing set**.

» Machine learning (ML): RS

Some MATH

- * Training dataset $(\text{feat}_i, \text{out}_i)_{i=1}^n$
- * Testing dataset $(\text{feat}_j)_{j=1}^m$:

Step 1. Introduce a model with some **parameters**: f_θ

Step 2. Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set** (replace **test data** in Evaluation by **training data**)

$$\hat{f}_\theta = \underset{f_\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n L(\text{out}_i, f_\theta(\text{feat}_i)).$$

Step 3. Use the estimated model to **predict**

$$\widehat{\text{out}}_j = \hat{f}_\theta(\text{feat}_j).$$

» Components in ML

Data (feat, label) is a pair of **input features** and its **outcome**

Model f_{θ} : a **parameterized** function to map features to label

Loss $L(\cdot, \cdot)$: the measure of how good the **predicted** outcome compared with the **true** outcome

Opt The **algorithm** for solving the problem

» Case study: Linear regression in California housing dataset

Data The California housing data frame has 20640 rows and 8 columns (it was divided into train and test sets)

Feats **MedInc** - median income in block group

HouseAge - median house age in block group

AveRooms - average number of rooms per household

AveBedrms - average number of bedrooms per household

Population - block group population

AveOccup - average number of household members

Latitude - block group latitude

Longitude - block group longitude

outcome **MedHouseVal** - median value of owner-occupied homes (target).

» Case study: Linear regression in California housing dataset

Data The California housing data frame has 20640 rows and 8 columns (it was divided into train and test sets)

Feats 8 feats

outcome **MedHouseVal** - median value of owner-occupied homes (target).

split train and test splits

Python:

- * consider the data type in Python
- * preprocessing data:
sklearn.preprocessing.StandardScaler

InClass demo: **loading/overview/preprocessing data**

» Case study: Linear regression in California housing dataset

Loss Evaluated by **RMSE** on a test set ($feat_j, out_j$)

$$RMSE(f_{\theta}) = \sqrt{\frac{1}{m} \sum_{j=1}^m (out_j - \widehat{out_j})^2}$$

Model Let's first use the **linear model** to make prediction for the problem

» Case study: Linear regression in California housing dataset

Recall the steps

Step 1. Introduce a model with some **parameters**: f_{θ}

$$f_{\theta}(\text{feat}_i) = \sum_{j=1}^d \theta_j \text{feat}_{ij} + \theta_0$$

InClass demo: **model**

» Case study: Linear regression in California housing dataset

Recall the steps

Step 1. Introduce a model with some **parameters**: f_{θ}

$$f_{\theta}(\text{feat}_i) = \sum_{j=1}^d \theta_j \text{feat}_{ij} + \theta_0$$

InClass demo: **model**

Step 2. Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set** (replace **test data** in Evaluation by **training data**)

$$\hat{f}_{\theta} \leftarrow \underset{\theta}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^n (\text{out}_i - \sum_{j=1}^d \theta_j \text{feat}_{ij} - \theta_0)^2 \right)^{1/2}.$$

» Case study: Linear regression in California housing dataset

Step 2(i). How to solve the minimization problem:

$$\min_{\theta} \left(\frac{1}{n} \sum_{i=1}^n (\text{out}_i - \sum_{j=1}^d \theta_j \text{feat}_{ij} - \theta_0)^2 \right)^{1/2},$$

is equivalent to

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (\text{out}_i - \sum_{j=1}^d \theta_j \text{feat}_{ij} - \theta_0)^2.$$

Denote $y_i = \text{out}_i$, and $\mathbf{x}_i = \text{feat}_i$, then it becomes **linear regression!**

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \mathbf{x}_i - \theta_0)^2$$

» Case study: Linear regression in California housing dataset

Step 2(ii). Linear regression:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^T \mathbf{x}_i - \theta_0)^2$$

Here, we give two ways to solve the problem in Python.

`sklearn` (use `sklearn.linear_model.LinearRegression`)

- * define a model: `model = LinearRegression()`
- * feed **data** to `model.fit(X_train, y_train)`
- * pred with `model.predict(X_test)`

PS most methods in `sklearn` follow a consistent pipeline that typically involves two main methods: `fit` and `predict`

- * InClass demo: **sklearn LR solver**

» Case study: Linear regression in California housing dataset

Step 2(ii). Linear regression:

$$\min_{\theta} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \mathbf{x}_i - \theta_0)^2$$

Here, we give two ways to solve the problem in Python.

M Manually solve the problem via the **matrix form** (after data **pre-processing**):

$$\min_{\theta} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\theta - \theta_0\|_2^2$$

$$\implies \hat{\theta} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \bar{\mathbf{y}}, \quad \bar{\mathbf{y}} = \mathbf{y} - \hat{\theta}_0, \quad \hat{\theta}_0 = \frac{1}{n} \sum_{i=1}^n y_i$$

InClass demo: **Manually LR solver**

» Case study: Linear regression in California housing dataset

Recall the steps

Step 1. Introduce a model with some **parameters**: f_{θ}

$$f_{\theta}(\text{feat}_i) = \sum_{j=1}^d \theta_j \text{feat}_{ij} + \theta_0$$

Step 2. Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set** (replace **test data** in Evaluation by **training data**)

$$\hat{f}_{\theta} \leftarrow \underset{\theta}{\operatorname{argmin}} \left(\frac{1}{n} \sum_{i=1}^n (\text{out}_i - \sum_{j=1}^d \theta_j \text{feat}_{ij} - \theta_0)^2 \right)^{1/2}.$$

Step 3. Use the estimated model to **predict**

$$\widehat{\text{out}}_j = \hat{f}_{\theta}(\text{feat}_j).$$

» Rethink baseline methods: RS

The ML learning paradigm for recommender systems.

Data **feat** = (user_id, item_id) → **rating**

Loss **RMSE**: root mean squared error

$$\text{RMSE}(f_{\theta}) = \sqrt{\frac{1}{|\Omega^{\text{te}}|} \sum_{(u,i) \in \Omega^{\text{te}}} (r_{ui} - \hat{r}_{ui})^2}$$

Model Baseline models

Glob $f_{\theta}(u, i) = \mu_0$; (μ ; 1 **param**)

User $f_{\theta}(u, i) = a_u$; ($\mathbf{a} = (a_1, \dots, a_n)^{\top}$; n **params**)

Item $f_{\theta}(u, i) = b_i$; ($\mathbf{b} = (b_1, \dots, b_m)^{\top}$; m **params**)

Opt Can we solve the **optimal** parameters for the baseline models from supervised ML formulation?

» Rethink baseline methods: RS

Step 1. **Model.** Introduce a model with some **parameters**: f_θ

Step 2. **Opt.** Estimate the **parameters** by minimizing (maximizing) the **Evaluation Loss** in **Training Set**, i.e., find θ such that

$$\min_{\theta} \left(\frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (f_\theta(u,i) - r_{ui})^2 \right)^{\frac{1}{2}} \\ \iff \min_{\theta} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (f_\theta(u,i) - r_{ui})^2$$

Step 3. **Predict.** Use the estimated model to **predict**

$$\hat{r}_{ui} = \hat{f}_\theta(u,i), \quad (u,i) \in \Omega^{\text{te}}.$$

» Rethink baseline methods: Opt

Steps 1 and 3 are clear, let's focus on Step 2.

Glob $f_{\theta}(u, i) = \mu_0$:

$$\hat{\mu}_0 = \underset{\mu_0 \in \mathbb{R}}{\operatorname{argmin}} \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} (r_{ui} - \mu_0)^2$$

Taking the derivative to μ_0 :

$$2 \sum_{(u,i) \in \Omega} (\mu_0 - r_{ui}) = 0, \implies \mu_0 = \frac{1}{|\Omega|} \sum_{(u,i) \in \Omega} r_{ui} = \bar{r}$$

* The **best** global constant prediction is nothing but **global mean**!

» Rethink baseline methods: Opt

User **model**: $f_{\theta}(u, i) = a_u$; **all params**: $\mathbf{a} = (a_1, \dots, a_n)^T$

$$\min_{\mathbf{a} \in \mathbb{R}^n} \sum_{(u,i) \in \Omega} (a_u - r_{ui})^2 \iff \min_{\mathbf{a} \in \mathbb{R}^n} \sum_{u=1}^n \sum_{i \in I_u} (a_u - r_{ui})^2$$

- * The loss function is **separable**, thus it suffices to consider user-wise minimization: for $u = 1, \dots, n$

$$a_u = \operatorname{argmin}_{a_u \in \mathbb{R}} \sum_{i \in I_u} (a_u - r_{ui})^2 = \frac{1}{|I_u|} \sum_{i \in I_u} r_{ui} = \bar{r}_u,$$

- * The **best** user-specific constant prediction is nothing but **user mean**!

» Rethink baseline methods: Opt

InClass practice.

Item **model**: $f_{\theta}(u, i) = b_i$; **all params**: $\mathbf{b} = (b_1, \dots, b_n)^{\top}$

» Rethink baseline methods: Opt

InClass practice.

Item **model**: $f_{\theta}(u, i) = b_i$; **all params**: $\mathbf{b} = (b_1, \dots, b_n)^{\top}$

$$\min_{\mathbf{b} \in \mathbb{R}^m} \sum_{(u, i) \in \Omega} (b_i - r_{ui})^2 \iff \min_{\mathbf{b} \in \mathbb{R}^m} \sum_{i=1}^m \sum_{u \in \mathcal{U}_i} (b_i - r_{ui})^2$$

- * The loss function is **separable**, thus it suffices to consider item-wise minimization: for $i = 1, \dots, m$

$$\hat{b}_i = \operatorname{argmin}_{b_i \in \mathbb{R}} \sum_{u \in \mathcal{U}_i} (b_i - r_{ui})^2 = \frac{1}{|\mathcal{U}_i|} \sum_{u \in \mathcal{U}_i} r_{ui} = \bar{r}_i,$$

- * The **best** item-specific constant prediction is nothing but **item mean**!

» Rethink baseline methods: Opt

Step 1. Introduce a method with some **params**

| method | MATH | parameters |
|-------------|------------------------|---------------------------------------|
| Global pred | $\hat{r}_{ui} = \mu_0$ | μ_0 |
| User pred | $\hat{r}_{ui} = a_u$ | $\mathbf{a} = (a_1, \dots, a_n)^\top$ |
| Item pred | $\hat{r}_{ui} = b_i$ | $\mathbf{b} = (b_1, \dots, b_m)^\top$ |

Step 2. Estimate the **parameters** by minimizing **RMSE**

Global $\hat{f}_\theta(u, i) = \bar{r}$

User $\hat{f}_\theta(u, i) = \bar{r}_u$

Item $\hat{f}_\theta(u, i) = \bar{r}_i$

Step 3. Make a prediction

InClass demo: **Make Baseline Methods as sklearn Estimators.**

» Case study: knn regression in California housing dataset

New model for California housing dataset?

Loss Evaluated by RMSE on a test set ($feat_j, out_j$)

$$RMSE(f_{\theta}) = \sqrt{\frac{1}{m} \sum_{j=1}^m (out_j - \widehat{out}_j)^2}$$

* Consider kNN regression.

Opt Using `sklearn.neighbors.KNeighborsRegressor`

* InClass demo: Colab

» Overfitting in ML

Results kNN regression with different **#neighbors**

```
##### 1-NN regression #####  
train_mse: 0.000; test_mse: 0.670  
##### 5-NN regression #####  
train_mse: 0.273; test_mse: 0.434  
##### 10-NN regression #####  
train_mse: 0.330; test_mse: 0.420  
##### 20-NN regression #####  
train_mse: 0.373; test_mse: 0.424  
##### 50-NN regression #####  
train_mse: 0.420; test_mse: 0.446  
##### 100-NN regression #####  
train_mse: 0.453; test_mse: 0.469
```

- Obs **#neighbors** $\searrow \implies$ (i) train error \searrow (ii) test error $\searrow + \nearrow$
- * When **#neighbors** is too large, we have **overfitting**
 - * This is so-called **bias-variance trade-off**

» Hyperparameter (HP)

Parameter: A parameter is a variable that is learned from the training data and is used to make predictions on new, unseen data.

$$y = \beta_0 + \beta_1 x + \varepsilon \quad (1)$$

where β_0 and β_1 are **parameters** learned from the training data.

Hyperparameter: A hyperparameter is a variable that is set **before training a model**, and its value is used to control the learning process.

$$\min_{\beta_0, \beta_1} \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2 + \lambda (\beta_0^2 + \beta_1^2) \quad (2)$$

where λ is a **hyperparameter** that is set before training the model, controlling the strength of regularization.

» Overfitting in ML

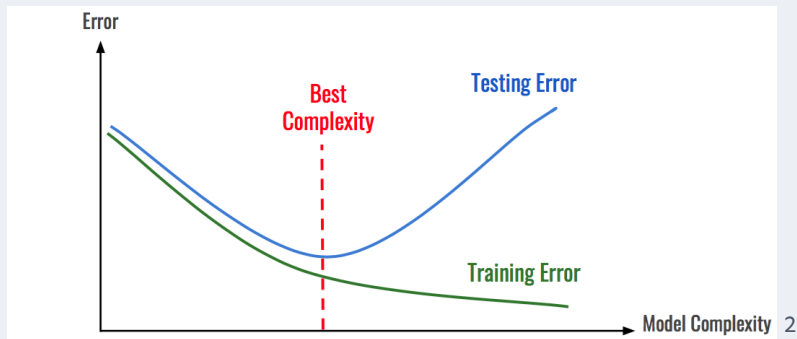


1

- * **Overfitting**: fit the noise
- * Too many **parameters** (**model complexity**) leads to **overfitting**
- * In kNN, when **#neighbors** \searrow , the model becomes more complicate

¹<https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Overfitting in ML



* **Complexity** too large \Rightarrow Low Training loss but high Testing loss

²Image source: <https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Overfitting in ML: Latent Factor Model

| | | |
|--------------------|------------------------|--|
| | Low Training Error | High Training Error |
| Low Testing Error | The model is learning! | Probably some error in your code. Or you've created a <i>psychic</i> AI. |
| High Testing Error | OVERFITTING | The model is not learning. |

Source³

³<https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42>

» Overfitting: solution

Q: How to address the issue of **overfitting**?

- * Introduce a **hyperparameter** (hp) to control the complexity of the model
 - * Typical hyperparameters are **#params**, **magnitude of params**
 - * Control the **complexity** of the model
 - * Smoothness $\nearrow \Rightarrow$ **complexity** \searrow

» Overfitting: solution

Q: How to address the issue of **overfitting**?

- * Introduce a **hyperparameter** (hp) to control the **complexity** of the model
 - * Typical **hyperparameters** are **#params**, **magnitude of params**
 - * Control the **complexity** of the model
 - * **Smoothness** ↗ \implies **complexity** ↘
- * **Examples**
 - * kNN models: **#neighbors**
 - * Ridge regression: **weight** λ for the l_2 -penalty

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\text{out}_i - \theta^\top (\text{feat}_i))^2 + \lambda \|\theta\|_2^2.$$

» Overfitting: solution

Q: How to address the issue of **overfitting**?

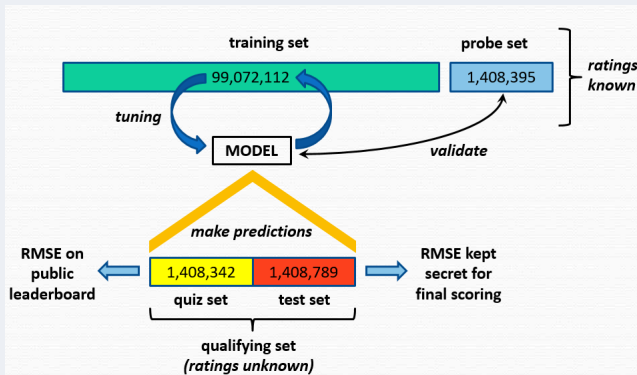
- * Introduce a **hyperparameter** (hp) to control the complexity of the model
 - * Typical hyperparameters are **#params**, **magnitude of params**
 - * Control the **complexity** of the model
 - * **Smoothness** $\nearrow \Rightarrow$ **complexity** \searrow
- * Examples
 - * kNN models: **#neighbors**
 - * Ridge regression: **weight λ** for the l_2 -penalty

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\text{out}_i - \theta^\top (\text{feat}_i))^2 + \lambda \|\theta\|_2^2.$$

Q: How to determine the **optimal** hyperparameter?

- * **GridSearch** + **cross-validation** (CV)

» CV: Holdout



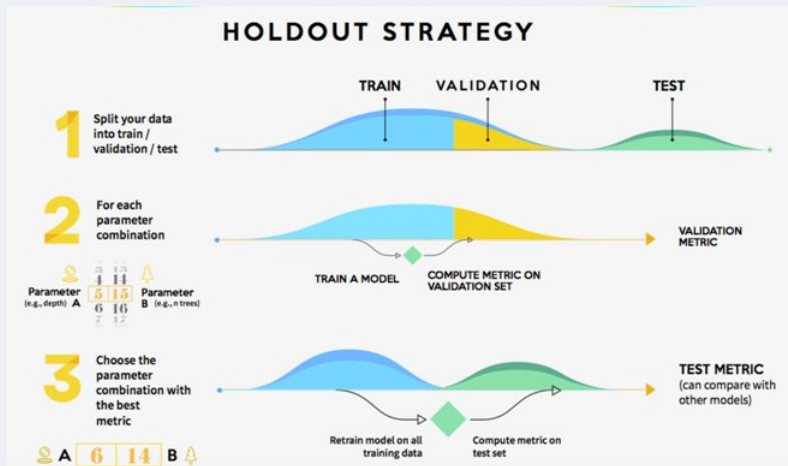
- * Further split **train set** to { **train set** and **valid set** }

- * One **hyperparameter** → **perf** on **valid set**

- * Select the optimal **hyperparameter** based on **valid perf**

Idea Good model in **valid set** tends to perform well in **test set**

» GridSearch + Holdout CV



4

⁴<https://medium.com/@sanidhyaagrawal08/what-is-hyperparameter-tuning-cross-validation-and-holdout-validation-and-model-selection-a818d225998d>

» Holdout Cross-validation: kNN regression

Results Cross-validation kNN regression:

```
k: 1; train_mse: 0.000; valid_mse: 0.648
k: 5; train_mse: 0.287; valid_mse: 0.435
k: 10; train_mse: 0.345; valid_mse: 0.426
k: 20; train_mse: 0.387; valid_mse: 0.433
k: 50; train_mse: 0.436; valid_mse: 0.457
k: 100; train_mse: 0.474; valid_mse: 0.487
```

* **optimal** #neighbors = 10

Refit Use the **optimal** hp to refit the model with **ALL** data

Golden Rule More data is better

» Holdout Cross-validation: Ridge regression

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^n (\text{out}_i - \theta^\top(\text{feat}_i))^2 + \lambda \|\theta\|_2^2.$$

- * $\lambda \nearrow \implies$ **less weight** in fitting or **reduce** the model complexity

Results Cross-validation **ridge regression**:

```
alpha: 0.5; train_mse: 0.519; valid_mse: 0.5231
alpha: 1.0; train_mse: 0.519; valid_mse: 0.5231
alpha: 10.0; train_mse: 0.519; valid_mse: 0.5230 (best)
alpha: 50.0; train_mse: 0.520; valid_mse: 0.5233
alpha: 100.0; train_mse: 0.522; valid_mse: 0.5246
alpha: 1000.0; train_mse: 0.575; valid_mse: 0.5784
```

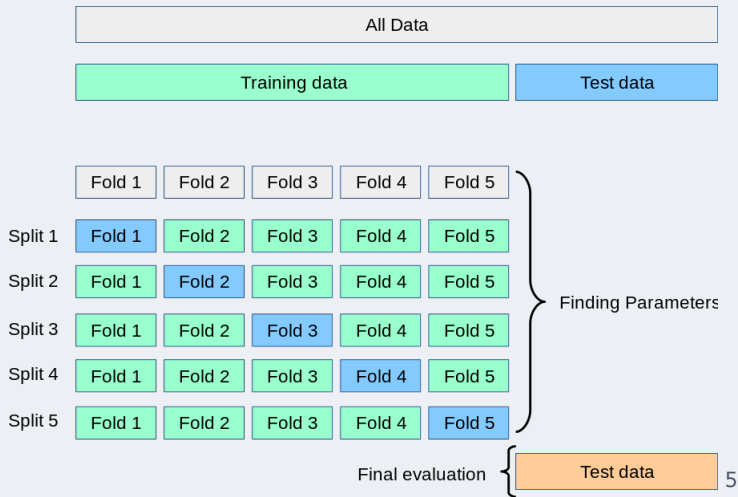
- * **optimal** penalty weight = 10

» Cross-validation: rule of thumb

- R1 Design your **Grid**: optimal hp **INSIDE** your grid
- R2 Breakdown the **local** mini-hp to get a better one
- R3 **More data is better**
 - * Use the **optimal** hp to refit the model with **ALL**
- R4 CV based on (only) **ONE** validation set is somehow risky...
 - * random splitting many times
 - * k -fold CV

» k -Fold Cross-Validation

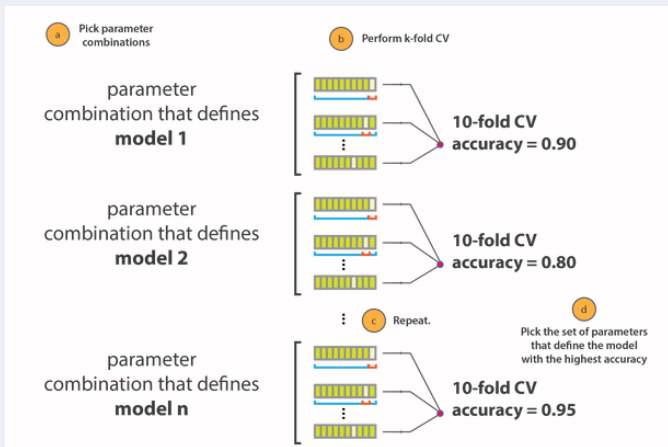
* Typical splitting method: k -fold CV



⁵ Image Source: https://scikit-learn.org/stable/modules/cross_validation.html/

» k -Fold Cross-Validation

* GridSearch + k -fold CV



6

⁶Image Source: <https://cambridgecoding.wordpress.com/2016/04/03/scanning-hyperspace-how-to-tune-machine-learning-models/>

» Summary

Let's summarize:

- Step 1 Design your **model** (**param** & **hp**); **Grid** for **hp**
- Step 2 Train **param** based on training set with different **hp**
- Step 3 Compute **score** for each **hp** based on a **holdout CV** or **k-fold CV**; and select the **optimal** **hp**
- Step 4 Refit the model with **optimal** **hp** based on **ALL** data
- Step 5 Make **prediction** for test set

InClass demo: **Implement GridSearch + CV in Python**