ML Method Overview

STAT3009 Recommender Systems

by Ben Dai (CUHK-STAT)
on November 17, 2022

» 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),

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

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

This is typical *supervised* 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 works well in **Training Set**, tend to work well in **Testing Set**

» Machine learning (ML): RS

⚠ 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)

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

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

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

» 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

- The California housing dataset 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

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

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

* 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 $\searrow$ $\Longrightarrow$ (i) train error $\searrow$ (ii) test error $\searrow$ + $\nearrow$

* When #neighbors is too small, we have **overfitting**
* This is so-called **bias-variance** trade-off

» Overfitting in ML

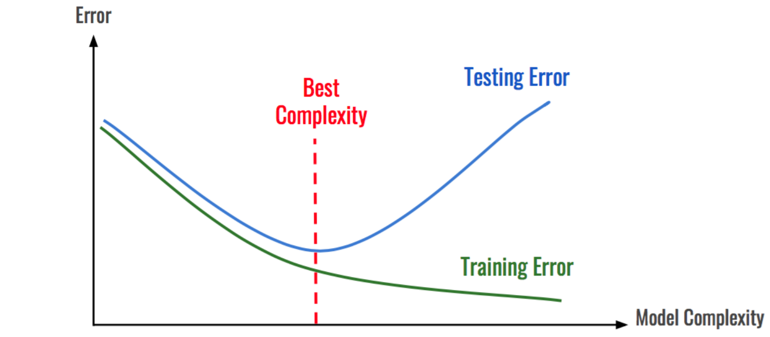
[1]

- **Overfitting**: fit the noise
- Too many parameters (**model complexity**) leads to **overfitting**
- In kNN, when #neighbors ↘, 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



[2]

* **Complexity** too large $\Longrightarrow$ Low Training loss but high Testing loss

[2] Image 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 *psychic* AI.
High *Testing* Error	OVERFITTING	The model is not learning.

Source[3]

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 $\nearrow \implies$ 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 $\nearrow \implies$ complexity $\searrow$
- Examples
  - kNN models: #neighbors
  - Ridge regression: weight λ for the l_2-penalty

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(\mathrm{out}_i - \theta^\intercal(\mathrm{feat}_i)\right)^2 + \lambda \|\theta\|_2^2.$$

» Overfitting: solution

Q: How to address the issue of overfitting?

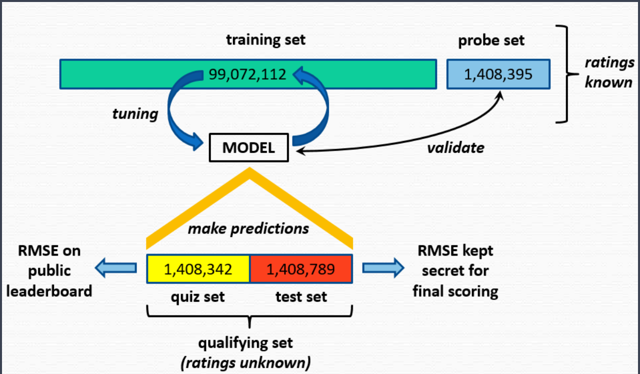
* Introduce a **hyperparameter** (hp) to control the complexity of the model
  * Typical hyperparameters are #params, magnitude of params
  * Control the **complexity** of the model
  * Smoothness $\nearrow$ $\Longrightarrow$ complexity $\searrow$
* Examples
  * kNN models: #neighbors
  * Ridge regression: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

$$\widehat{\theta} = \underset{\theta}{\operatorname{argmin}} \ \frac{1}{n} \sum_{i=1}^{n} \left(\mathrm{out}_i - \theta^{\mathsf{T}}(\mathrm{feat}_i)\right)^2 + \lambda \|\theta\|_2^2.$$

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

* Tune by **cross-validation** (CV)

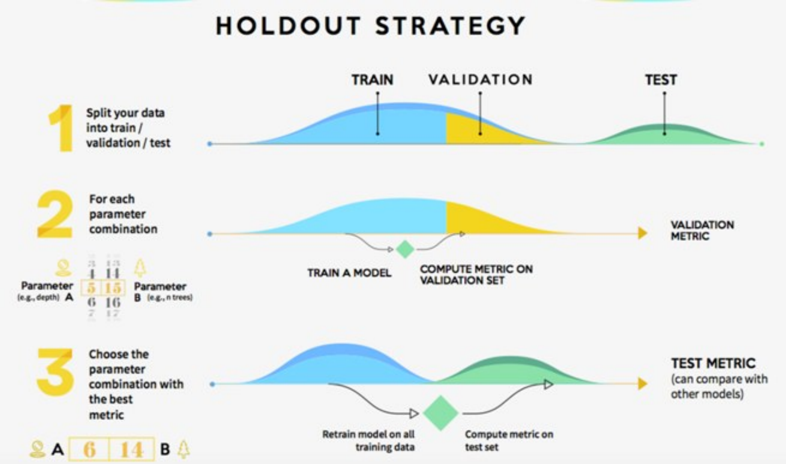
» Cross-validation: validation dataset



- Further split train set to { train set and valid set }
- One hyperparameter $\rightarrow$ 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

» Cross-validation: validation dataset



[4]

---

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

» 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

» Cross-validation: Ridge regression

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

* $\lambda \nearrow \Longrightarrow$ 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

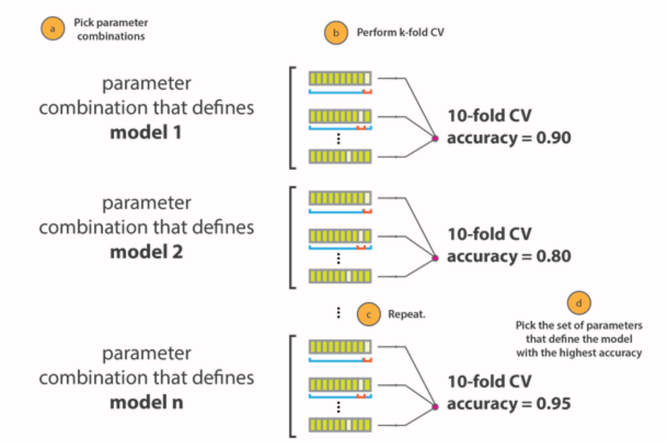
- Use the optimal hp to refit the model with ALL

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 (**Homework**)



[5]

[5] Image 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 valid loss for each hp based on *a* valid set 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

» Rethink baseline methods: RS

The ML learning paradigm for recommender systems.

Data feat = (user_id, item_id) → rating

Loss **RMSE**: root mean squared error

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

Model Baseline models

- Glob $f_\theta(u,i) = \mu_0;\quad (\mu;\ 1$ param)
- User $f_\theta(u,i) = a_u;\quad (\mathbf{a} = (a_1, \cdots, a_n)^\intercal;\ n$ params)
- Item $f_\theta(u,i) = b_i;\quad (\mathbf{b} = (b_1, \cdots, b_m)^\intercal;\ m$ params)
  - No hp

Opt Can we solve the optimal parameters for the baseline models from supervised ML formulation? [Page 3]

» 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

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

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

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

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

» Rethink baseline methods: Opt

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

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

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

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: $\boldsymbol{a} = (a_1, \cdots, a_n)^\intercal$

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

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

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

* 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: $\mathbf{b} = (b_1, \cdots, b_n)^\intercal$

» Rethink baseline methods: Opt

InClass practice.

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

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

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

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

- 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	$\hat{r}_{ui} = a_u$	$\boldsymbol{a} = (a_1, \cdots, a_n)^\intercal$
Item pred	$\hat{r}_{ui} = b_i$	$\boldsymbol{b} = (b_1, \cdots, b_m)^\intercal$

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

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

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

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

Step 3. Make a prediction

» Discussion: baseline methods

"All models are wrong, but some are useful." — George E. P. Box

We need to figure out the **assumptions** for each method!

* **Global average** assumes that all users and items are essentially same
* **User average** assumes that a user has equal preference to all items
* **Item average** assumes that all users like "good" items

Motivation: I just don't like Action movies, even their ratings is quit high. We need to model the user-item **interaction**.