Statistical Computing

Michael Mayer

March 2023

Statistical Computing: What will we do?

Chapters

1. R in Action
2. Statistical Inference
3. Linear Models
4. Model Selection and Validation
5. Trees
6. Neural Nets

Remarks

- Chapters 3 to 6:
  Statistical ML in Action
- Two weeks per chapter
- Exercises at end of chapter notes

Model Selection and Validation

Two Questions

→ prevent overfitting (more params → more overfit

- "How good is our model?"
- "Which model to choose among alternatives?"

Problem and solution

- "In-sample" performance is biased
  evaluating model with train data
- Overfitting should not be rewarded
- Use data splitting to get fair results

Notation

Data

loss fct.

- Total loss $Q(f, D) = \sum_{(y_i, \boldsymbol{x}_i) \in D} L(y_i, f(\boldsymbol{x}_i))$
- Average loss $\bar{Q}(f, D) = Q(f, D)/|D|$ Avg Bernoulli deviance
- Performance measure or evaluation metric $S(f, D)$ of interest, often $S = \bar{Q}$ or a function of it for logistic function

if $L(y_i, f(x_i)) = \|y_i, f(x_i)\|^2$

→ S might be $\sqrt{\bar{Q}}$ to conserve the dimension

Outline

- Nearest-Neighbor
- Simple Validation
- Cross-Validation
- Test Data and Final Workflow
- Excursion: SQL and Spark

Excursion: k-Nearest-Neighbor (k-NN)

"Black box model"

- Alternative to linear model
- How does it work?
- Classification and regression
- Standardization?

select covariates
look at objects with similar covariates
take average response variable

$$\frac{\text{covariates} - \text{mean}}{\text{stdev.}}$$

maybe weighting the
covariates, measure the distance with euclidian dist.

Example

categorys ordinal measuring
clarity is ordered
1
2
3
:
contours are not
→ dummy variables

Simple Validation

- In-sample, 1-NN would win any comparison!? → overfitting
- Split data into training and validation sets D_{train} and D_{valid}, e.g., 80%/20%
- Use performance $S(\hat{f}, D_{\text{valid}})$ on validation set to make decisions (choose models, choose parameters like k)
- Measure amount of overfitting/optimism by

$$S(\hat{f}, D_{\text{valid}}) - S(\hat{f}, D_{\text{train}})$$

Example

packages

caret
MLR3 (R6)
tidymodels very strange

K-fold Cross-Validation (CV)

Simple validation is neither economic nor robust, except for large data

not the k-NN
range 5 - 10

Algorithm

1. Split the data into K pieces $D = \{D_1, \ldots, D_K\}$ called "folds". Typical values for K?
2. Set aside one of the pieces (D_k) for validation
3. Fit model $\hat{f}_k$ on $D \setminus D_k$
4. Calculate performance $\hat{S}_k = S(\hat{f}_k, D_k)$
5. Repeat Steps $2 - 4$ for each k
6. Calculate CV performance $\hat{S}_{CV} = \frac{1}{K} \sum_{k=1}^{K} \hat{S}_k$

Remarks

fit k-NN on the whole data set for a final mo

- How to choose and fit best/final model?
- What means «best»?
- Stability of results? calculate std dev of $\hat{S}_k$
- Repeated CV?

Example

repeat CV with different seeds, if small dataset

Hyperparameter Tuning

- Choosing k in k-NN is example of "hyperparameter tuning"
- Algorithms with more than 1 hyperparameter? xg boost
- Grid Search CV grid of hyp. params
- Randomized Search CV if too big grid
  → randomised search

Test Data and Final Workflow

Problematic consequence of model tuning?

- Overfitting on validation data or on CV!
- Performance of final model? → Test data

Workflow A

simple validation

1. Split data into train/valid/test,
   e.g., by ratios 60%/20%/20%
   70 20 10
2. Train different models on training data and assess performance on validation data. Choose best model, re-train on (training + validation) data, and call it "final model" (Simplification?) by not retraining on both
3. Assess performance of final model on test data

Workflow B

cross validation

1. Split data into train/test,
   e.g., by ratios 80%/20%.
2. Evaluate and tune different models by K-fold CV on training data. Choose best model, re-train on full training data
3. Assess performance of final model on test data

Example of Workflow B

we can include the test data to fit the model

When test data not necessary?

Ridge Regression

High dim data, L^1 penalty (more cols?)
some or most of the
β are zero
→ feature selection

- Example of penalized regression
- Model equation similar to usual linear regression

$$\mathbb{E}(Y \mid \boldsymbol{x}) = f(\boldsymbol{x}) = \beta_0 + \beta_1 x^{(1)} + \cdots + \beta_p x^{(p)}$$

- But with penalized least-squares objective

can be tuned

effect: make
β closer to zero

$$Q(f, D_{\text{train}}) = \sum_{(y_i, \boldsymbol{x}_i) \in D_{\text{train}}} (y_i - f(\boldsymbol{x}_i))^2 + \underbrace{\lambda \sum_{j=1}^{p} \beta_j^2}_{L^2 \text{ penalty}}$$

- *L*2 penalty pulls coefficients slightly towards 0, fighting overfitting
- $\lambda_{\text{opt}} \geq 0$ with best (cross-)validation result → use to fit final model
- Intercept? Standardization? L1? Elastic-net? — both, L^1 and L^2 penalty

normally not penalised

scaling happens internally for most packages

abs(β_j)

Example

Using Independent Partitions is Essential

Random splits

generate indep. partitions

Grouped splits

rows from different clients
overly optimistic it have leakage
→ data from same client
in train and test data
→ avoid by random split by groupid

Time-Series splits

different logic

Stratified splits

in binary response:
less 1's than 0's → keep the distribution
in splits!

Excursion: SQL and Spark

Data science is 80% preparing data, 20% complaining about preparing data.

Typical preprocessing steps?

Good moment to learn

- data structure
- meaning of columns
- sources of bias

How to do preprocessing?

Data = files on disk or tables in database

- If small $\rightarrow$ R/Python
- If large? $\rightarrow$ Database Management System (DBMS) or Spark
- Communication via SQL

SQL

Structured Query Language

- Pronounced?
- Important in data science
- In DBMS or R/Python
- ISO norm $\leftrightarrow$ dialects
- SQL queries

DuckDB (since 2018)

- In-process, open-source DBMS
- Easy to install in R/Python
- No dependencies (Java etc.)
- Fast
- Out-of-core capabilities

Learn SQL with examples

- Diamonds (from memory)
- Taxi (from Parquet)

Apache Spark

- Distributed, open-source cluster computing system for big data
- Apache project since 2013
- Heavily used in industry
- Written in Scala
- Contains SQL engine
- Can be used from R/Python

Examples

- Diamonds (from memory)
- Taxi (from Parquet)