STA 141A Lecture 18a | ISLR 2.1, 2.2 | 2018年11月29日

Efficiency TARP 14

- Statistical Models & Machine Learning
  - So far mostly visualization and point statistics.
  - Modeling is another important part of EDA.
  - Other classes cover theory of models, so here the focus is implementation.
  - We saw performance was an issue for HW 5 with text; it can also be an issue with fitting models.
  - The basic idea of modeling is that we have some inputs $X_1, X_2, \dots$ and we want to predict or understand an output Y.

Tradeoff!

$$Y = f(X_1, X_2, \dots) + \varepsilon$$

ε ← error term

$$\hat{Y} = \hat{f}(X_1, X_2, \dots)$$

- Training Set

Test Set

X_{11}	Y_1
X_{21}	Y_2
$\vdots$	$\vdots$

$n =$ number of observations

Y: response, dependent

X: feature, predictor, independent

- Parametric vs non-parametric
  - Parametric models assume a shape form for f, then estimate parameters
  - Non-parametric models do not assume a shape for f, but do assume other properties, like smoothness. More obs needed
- Supervised (response) vs unsupervised (no response)
- Regression vs classification

- We use mean squared error to measure error in regression models:

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i-\hat{f}(x_i)\right)^2$$

Other measurements are possible and can change how the model fits.

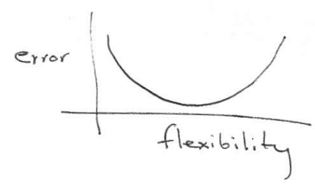
- We care about test error, not training error! This is where $\boxed{\text{overfitting}}$ can be a problem. } Revisit in CV example

- Bias-variance tradeoff:

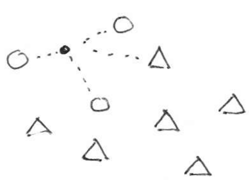
$$\mathbb{E}\left(y_0-\hat{f}(x_0)\right)^2 = \underbrace{Var\left(\hat{f}(x_0)\right)}_{\text{model variance}} + \underbrace{\left[Bias\left(\hat{f}(x_0)\right)\right]^2}_{\text{model bias}} + \underbrace{Var(\varepsilon)}_{\text{error variance}}$$

Variance — how $\hat{f}$ changes with different train set
Bias — error due to approximation

bias, inference $\longleftrightarrow$ variance, flexibility



- k-nearest neighbors



Use k-nearest training points to predict class

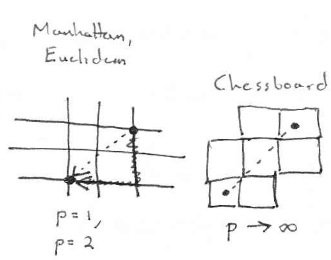
Ties!

$$\mathbb{P}\left(Y_\bullet=\Delta \mid X_{1\bullet}=x_{10}, X_2=x_{30}\right)=\frac{\#\,\Delta}{k}$$

Bayes classifier approximation

The choice of k makes a difference!
Similarly, how we measure distance makes a difference!

- Minkowski distances:
  - Euclidean $p=2$
  - Manhattan $p=1$



$$\left(\sum_{i=1}^{n}|a_i-b_i|^p\right)^{1/p} \qquad p\geq 1$$

STA 141A Lecture 18b

2018年11月29日

- How to actually implement kNN?
  - What are the inputs? The outputs?

    Input: $x_1, x_2, \ldots$ predictors
    k number of neighbors
    $\{(x,y)_i\}$ training set

    Output: $\hat{y}_0$ predicted label

  - So what are the steps?

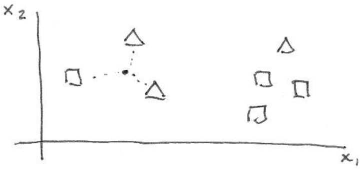
    1. Find k nearest neighbors to x_0
       - How do we tell which points are "near"?
       - Compute distances, find smallest distances
         dist() sort() order()
    2. Use neighbor labels to "vote"
       table()
       - Break ties
         sample()
    3. Return "winning" label

- Is kNN "fast"?
- Is kNN flexible? Where is bias-variance tradeoff?

2018年12月4日

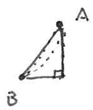
STA 141A Lecture 19

- References: ISLR 2, 5.1 / TARP 8, 14
- 5-min reminder of kNN



- Minkowski distance

$$\left(\sum_{i=1}^{d} |a_i - b_i|^p \right)^{1/p} \quad p \geq 1$$



Effect of k on model flexibility

- How to implement kNN?

predict_knn()

Inputs: new point, train points, k, metric

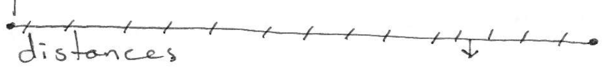
Outputs: class label

Steps:

Find:

1. Compute distances from new point(s) to all train points — dist(), as.matrix()

For each new point



2. Sort/order distances — order()
3. Take top k train points — [ ]

Vote:

4. Count classes for top k points — table()
5. Return winning class, possibly breaking ties. — which.max() / max()

Break the voting step (including ordering) into a separate function.

apply(new_points, 1, vote) ...

- Error rates: how can we tell how well the model works?

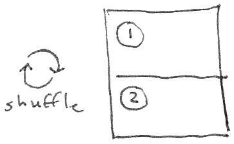
$$\sum \frac{\text{actual == pred}}{\text{total}}$$

Fit on train → predict on train → underestimate

Fit on train → predict on test → Why not use test to train?

* Cross-validation (m-fold) *

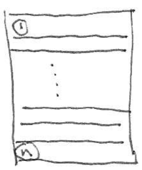
Idea: fit multiple times and average



Iteration	1	2
Train	1	2
Test	2	1
Error	e_1	e_2

$$\hat{e} = \frac{e_1 + \dots + e_m}{m}$$

Two folds doesn't give us much to average.
Error rate will vary depending on how we choose folds, and model doesn't get to see all the data } High bias



n error estimates Leave-one-out CV

Train all others
Test 1 2 ... n

Now we have to fit the model n times — not feasible for large n or slow model.
Estimates are highly dependent! → high variance

Usually we compromise and use $m = 5$ or $m = 10$

* Error rates

Bias - variance tradeoff

* How to implement CV?

cv_error_knn()

Inputs: m, train set, k, metric

Outputs: CV error estimate

Steps:

1. Shuffle the rows or sample row ids. sample()
2. Split rows into folds. rep() split()
3. Loop over test folds; remaining are train folds
   a. Fit model on train folds
   b. Predict for test fold
   c. Compute error rate
4. Return average of error rates. mean()

STA 141A Lecture 20

2018年12月6日

- Cross-validation (Lecture 19)
- Implementing CV (Lecture 19)
- Performance considerations for CV
  - Compute distances within training set once, <u>before</u> looping over the folds
  - Split row indexes instead of actual observations, and look up appropriate rows (distances) for each fold
- Recap designing functions / ideas to code:
  - List inputs, outputs, and steps — This requires careful thought
  - Draw pictures or make simple examples of the problem. Work these out by hand
  - Divide-and-conquer: complicated steps can be written as separate functions
  - Test on simple/small examples first
- Function examples?

  dist() order() table() split()
  sample() rep() indexing
- Where to go from here?
  - STA 141B focuses on collecting data, interactive visualizations, and uses Python
  - STA 141C focuses on programming for statistics and uses R (possibly with some C)
  - STA 135/108
  - ECS 171 machine learning