SOSC 4300/5500: Prediction Evaluation

Han Zhang

Outline

Logistics

How to compare different predictions?

- "Soviet Union will collapse one day"
  - There is only one prediction; our prediction is ultimately either right or wrong
  - It's nearly always to be true (because there is no time constraint!)
- In computational social sciences/modern data sciences, usually the prediction goals are precise and falsifiable (Hofman, Sharman and Duncan, 2017)

How to compare different predictions?

- Add scope conditions
- E.g., we make 10 predictions, adding constraint of time
    - Will Societ Union collapse in 1980? Yes or No?
    - ...
    - Will Societ Union collapse in 1989? Yes or No?
- If your theory/model is good, it should be able to predict in a more detailed way

Prediction difficulty vary by prediction goals

- Jake M. Hofman, Amit Sharma, and Duncan J. Watts, *Prediction and explanation in social systems*, Science **355** (2017), no. 6324, 486–488
- Given 852 million tweets, here is a list of things you can predict
- Predict whether each tweet will have more than 5 retweets: easier
- Predict the exact number of retweets for each tweet: harder
- Predict the size of cascades (number of retweets, retweets of retweets,..): hardest

Prediction evaluations for continuous outcomes

- It's common to use $\hat{Y}$ to denote the predicted value of Y
- For continuous outcomes:
- R^2: for linear regression
  - The larger the R^2, the better the model
- *MSE* (mean squared error): $1/n\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$
  - The most widely used metric
  - The smaller the MSE, the better the model
  - Sometimes we also use RMSE $= \sqrt{MSE}$
  - For regression: $R^2 = 1 - MSE/var(Y)$
- *MAE* (mean absolute error): $1/n\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$

Prediction evaluations for categorical outcomes

- For categorical outcomes, evaluation is more complex
- Cross-entropy loss is a common choice (used by some decision tree algorithms and some deep learning)
  - Also called log-loss or entropy loss
- For binary classification:

$$-\sum_{i=1}^{N} y_i \cdot \log P(\hat{y}_i = 1) + (1 - y_i) \cdot \log\left(1 - P(\hat{y}_i = 1)\right)$$

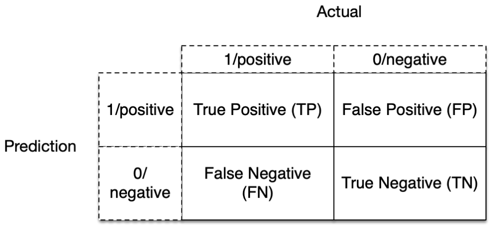
Prediction evaluation for categorical outcomes

- Another set of evaluation is based on tabulating predictions and actual values and is more intuitive
- Let us assume that there are 10,000 students/employees at HKUST, and there are 10 infected cases
- We has an algorithm predicting COVID infection (positive = 1 vs. negative = 0)
- We found that 99% of our predictions are correct. Yeah!
- But wait, is that good enough?

Prediction evaluations for categorical outcomes

- In fact, for any classification task, one of the simplest baseline is to predict every data point as belonging to the majority class
- Here, we know most people are not positive
- So the simplest baseline just predict that every one is negative
- What's the accuracy for this simplest baseline prediction?
- Accuracy = 9990 / 10000
- If class is imbalanced, it is very easy to achieve a high accuracy by predicting the majority class all the time
    - But it is misleading

Prediction evaluations for categorical outcomes



		Actual	
		1/positive	0/negative
Prediction	1/positive	True Positive (TP)	False Positive (FP)
	0/ negative	False Negative (FN)	True Negative (TN)

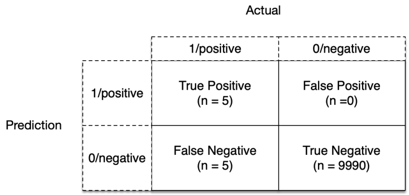
- It's better to use confusion matrix
- Each cell is the number of observations fall into the corresponding category
- accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$
- precision $= \frac{TP}{TP+FP}$
  - Interpretation: what proportion of predicted positives are actual positive?
- recall $= \frac{TP}{TP+FN}$
  - interpretation: what proportion true positives are identified by predictions?

Simplest baseline: majority class

		Actual	
		1/positive	0/negative
Prediction	1/positive	True Positive (n = 0)	False Positive (n = 0)
	0/negative	False Negative (n = 10)	True Negative (n = 9900)

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
  - $\frac{9990}{10000} = 99.9\%$
- precision $= \frac{TP}{TP+FP}$
  - $\frac{0}{0+0} =$ not defined
- recall $= \frac{TP}{TP+FN}$
  - $\frac{0}{0+10} = 0\%$
- From precision/recall, we see that this prediction is very bad, which suggests that precision/recall can recognize this majority guess as a bad prediction

Case 1: high precision/ low recall



		Actual	
		1/positive	0/negative
Prediction	1/positive	True Positive (n = 5)	False Positive (n =0)
	0/negative	False Negative (n = 5)	True Negative (n = 9990)

- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
  - $\frac{5+9990}{10000} = 99.95\%$
- precision $= \frac{TP}{TP+FP}$
  - $\frac{5}{5+0} = 100\%$
- recall $= \frac{TP}{TP+FN}$
  - $\frac{5}{5+5} = 50\%$
- So every predicted infected case is indeed infected
- But we missed 50% of actual infected cases

Case 2: high recall/low precision

		Actual	
		1/positive	0/negative
Prediction	1/positive	True Positive (n = 9)	False Positive (n = 4)
	0/negative	False Negative (n = 1)	True Negative (n = 9986)

- We lower the threshold to be considered as infection case
- Accuracy: $\frac{TP+TN}{TP+TN+FP+FN}$
  - $\frac{9+9986}{10000} = 99.95\%$; the same
- precision $= \frac{TP}{TP+FP}$
  - $\frac{9}{9+4} = 69.23\%$
- recall $= \frac{TP}{TP+FN}$
  - $\frac{9}{9+1} = 90\%$
- Our prediction captures 90% of actual infected cases
- But less than 70% predicted cases are actually infected

Precision-recall trade-off

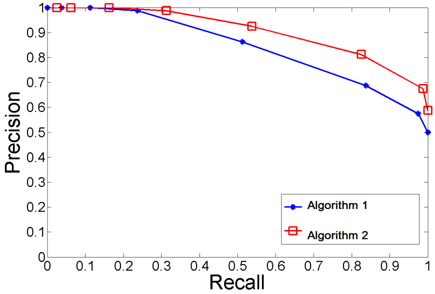
- In evaluting perdiction performances for categorical outcome, do not use accuracy
- Instead, use precision and recall
- Depending on tasks, we may emphasize one or the other
- An ideal algorithm will have both high precision and recall
- In real life, one always comes at the cost at the other
- This is called precision-recall trade-off
- [in class activities]: can you think of cases when high precision is more important? Cases when high recall is more important?

Precision-recall trade-off and decision threshold

- Many ML algorithms give you predicted probability, and then transform this probability into a binary prediction
- If $P(Y = 1) > \phi$; predicted probability is larger than a threshold
  - Predicted value $\hat{Y} = 1$
- If $P(Y = 1) <= \phi$;
  - $\hat{Y} = 0$
- Large threshold ϕ -> high precision
- Small threshold ϕ -> high recall
- Often software will choose threshold to be 0.5 by default
- You can generate new predictions based on whether you want high precision or high recall

Precision-recall curve

- Precision-recall curve is a way to visualize the trade-off
- Imagine you choose many different thresholds
- For each thresholds, obtain binary predictions, and calculate precision/recall
- Then plot the precision against recall



- Algorithm 2 is better than 1

Precision-recall and F1 score

- If you do not have a specific reason preferring one or the other
- $F1$ score is a single-number measure of how good your predictions are
  - $2 * \frac{precision * recall}{precision + recall}$

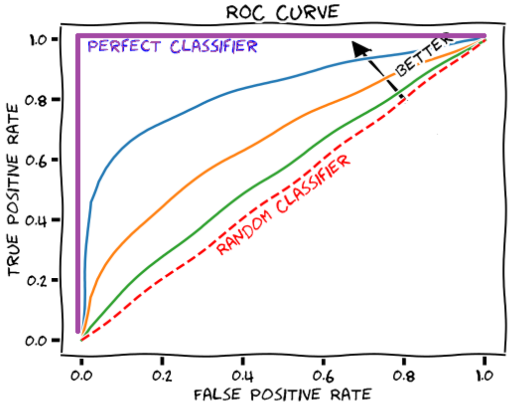
False positive and false negative rates

		Actual	
		1/positive	0/negative
Prediction	1/positive	True Positive (TP)	False Positive (FP)
	0/ negative	False Negative (FN)	True Negative (TN)

- True positive rate = recall: $\frac{TP}{TP+FN}$
- False negative rate: 1 - true positive rate = $\frac{FN}{TP+FN}$
    - For COVID example, what percentage of people were infected but predicted as not
- False positive rate: $\frac{FP}{FP+TN}$
    - For COVID example: what percentage of people were not infected but predicted as infected

ROC curve

- Similar to precision-recall curve case, vary decision thresholds and obtain false positive and false negative rates
- Then plot TPR = 1 - FNR against FPR



AUC of ROC

- Similar to $F1$ score is a single-number measure based on precision-recall curve
- Area under the curve (AUC) is a single-number measure based on ROC curve
- Larger AUC -> better prediction performance

ROC vs Precision/Recall

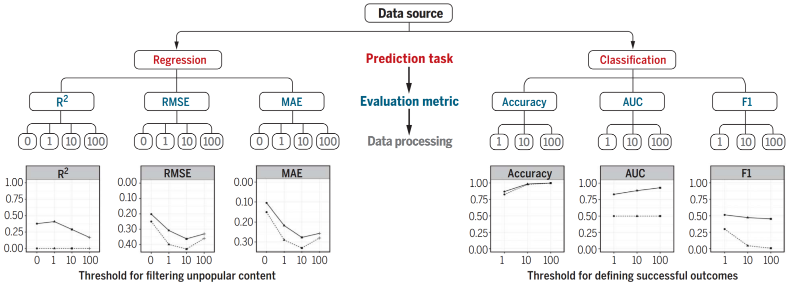
- Precision-recall curve and ROC curve are much better measure of algorithm performances than ROC curve
- Use ROC curve, if you care both positive and negatives
- Use precision/recall curve, if you care one class more than the other
    - e.g., you care about positive class more than negative class
- Use precision/recall curve, if your data is highly imbalanced, and you only care about one class
    - Often in text analysis, this is the case: you want to extract some information from documents

From binary to categorical

- If you have more than 2 categories
- Calculate precision/recall; FPR/FPR for each category
- Macro-average: treat each category as the same; take the average precision of each category
  - can be problematic if you have imbalanced data.
- Micro-average: take into consideration of the size of each category; preferred

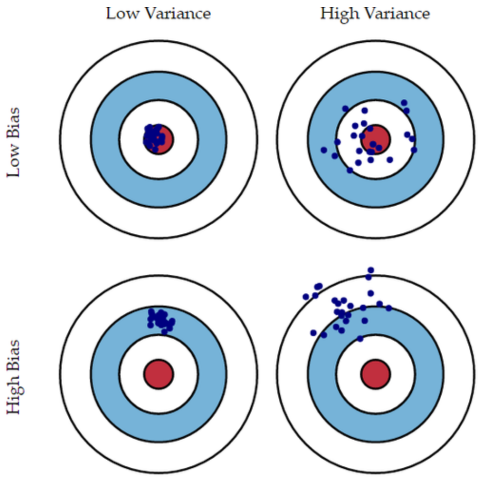
Summary of evaluation characteristics

- Jake M. Hofman, Amit Sharma, and Duncan J. Watts, *Prediction and explanation in social systems*, Science **355** (2017), no. 6324, 486–488



Bias Variance Trade-Off

- Typically, more complex prediction algorithm gives less biased predictions, but their variance is also huge



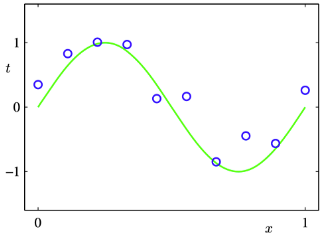
Bias Variance Trade-off

- One implication of high prediction algorithm variance is that it does not generalize well on new data that have not been seen by the ML algorithm
    - When this occurs, people call this overfitting
    - It means that your algorithm learn training data very well, but it is highly unstable on new data and can make a lot of errors
    - Intuition: you remember every thing taught in a class so well, but professor gives you some new exercises you have not seen and your knowledge does not handle new questions

Bias Variance Trade-off (example)

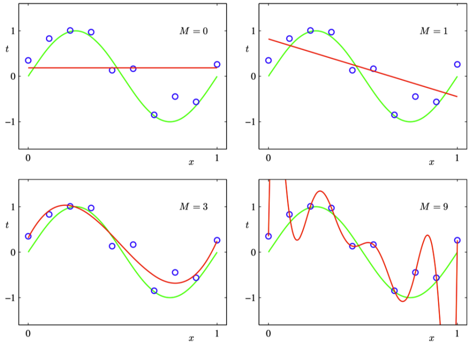
- We used $Y = sin(X)$ (the green curve) to generate some data (in blue dots)
- We use a polynomial regression with only one variable X for prediction:
    - Polynomial regression: keep adding higher order terms to independent variables
- When M is larger, we get models that fit the data better and better

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \cdots + X^M$$



Bias Variance Trade-off (cont'd)

- $M = 1$, OLS gives lots of estimation bias
- $M = 9$, the model fits the data so well, but it is highly sensitive to small changes in observations
- $M = 3$, it achieves a good balance between estimator bias and estimator variance



Bias Variance Trade-off

- There are typically two ways to achieve bias variance trade-off
  - One focus on algorithm side, and the other focus on data side
- Regularization: explicitly make the model less complex to balance bias and variance
  - Note that this is not commonly used in linear regression: people often tell you to add as much variables as possible
- Use train-test split:
  - Test your model's performance on new data.
  - Evidence-based argument:
    - There are many good algorithms to choose from; each of them claims that they have some good properties
    - The ultimate evaluation is how an algorithm performs on new data that the algorithms have never seen before.

Regularization of linear regression: LASSO and Ridge

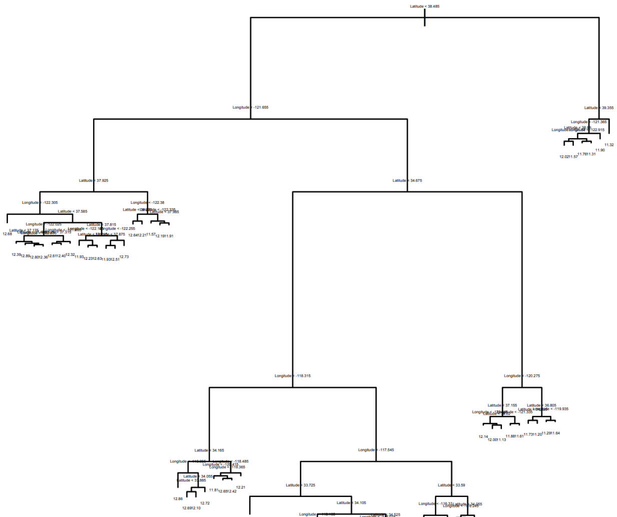
- When there is more regressors than independent variables
- LASSO: force some variables to be zero
- Ridge: force some variable to be closer to zero (weaker form or regularization)

Decision Tree: Bias vs Variance

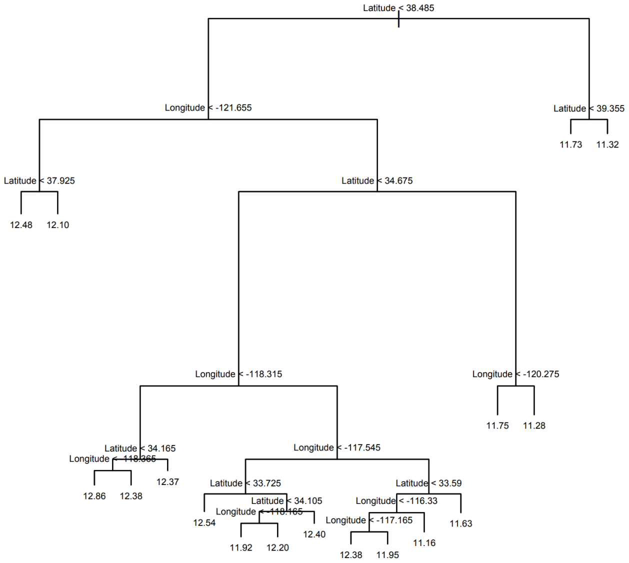
- A decision tree, without any restrictions, can be quite complex
- We can always make a very complex tree by:
    - Try your best to make every single leaf contains only one Y
- Bias-variance trade-off:
    - Complex trees fit the data nearly perfect (low predictor MSE)
- We need to regularize to make tree simpler
    - This is the same principle of LASSO/Ridge
        - To make better predictions, we sometimes have to make the algorithm less complicated

Decision Tree: un-pruned

- In decision tree, regularization is called pruning
- It's like cutting leaves and making the tree smaller



Decision Tree: pruned



Random Forest: regularization

- Bagging: simply fit many trees over the entire data
- The key innovation of random forests:
- For each sample from the original training data, randomly select $m < p$ variables, and grow a tree;
  - A common choice: $m = \sqrt{p}$
- In other words, we just force $p - m$ predictors to be non-relevant each time

Google Flu Trends Example

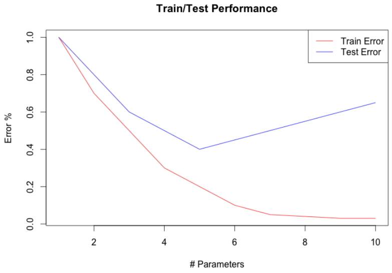
- Principle: use new data to evaluate your algorithm
- Training data: CDC counts and search queries from 2003 to 2007
  - Training Procedure:
    1. Select a model: e.g., linear regression, CDC counts $\sim \beta \times$ (Google search queries)
    2. Training (fitting) model: obtain the value of regression coefficient β
- New data: CDC counts and search queries in 2008
  - Evaluating procedures:
    1. calculate predicted CDC counts, based on β and search queries in 2018
    2. compare actual CDC counts in 2008, and predicted CDC counts from the above
- Evidence-based way of choosing the best model:
  - if you have another algorithm, just test it on the exact same new data
  - If the new algorithms gives better performance, then use the new algorithm

Train/test split

- Time-series data allow very natural way to choose new data
- Train/test split is an alternative way if you do not have time-series data
- Typically, you don't use all of your data to train an algorithm
    - Again, very different from running linear regression, where you almost always use all data in your regression
- Instead, you split your data into two parts (typically 80%/20% or 90%/10%)
    - training data
    - (out-of-sample) test data
- Train your model only with training data
- And evaluate your model with test data
    - With precision/recall or ROC curves
- During training, ML algorithm must not see the test data

Train/test split

- Test error (e.g., MSE or FPR/FNR) typically is larger than training error
- When test error begins to increase, overfitting occurs
- Error/performance metrics based on test data gives more faithful evaluation of how the algorithm will perform in real-world, unseen new data



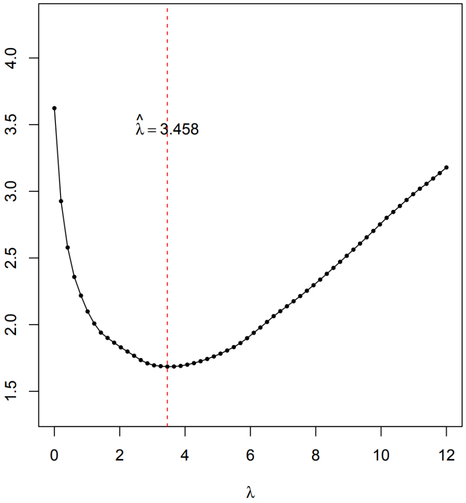
Example: Use train/test split to choose a LASSO model

- Assume we have a LASSO model, and we think our λ can take values 0,1,2,3, $\cdots$, 10
- For each possible value of λ:
  - Use the training data to fit your statistical model (here: LASSO)
  - Use the estimated parameters to predict Y for the test data
  - Calculate MSE on the test data

$$1/n \sum_{i \in \text{test}} \left(Y_i - \hat{\beta} X \right)$$

- We should select λ that yields the smallest MSE on the test data

Example: Use train/test split to choose a LASSO model



Example: Use train/test split to prune a decision tree

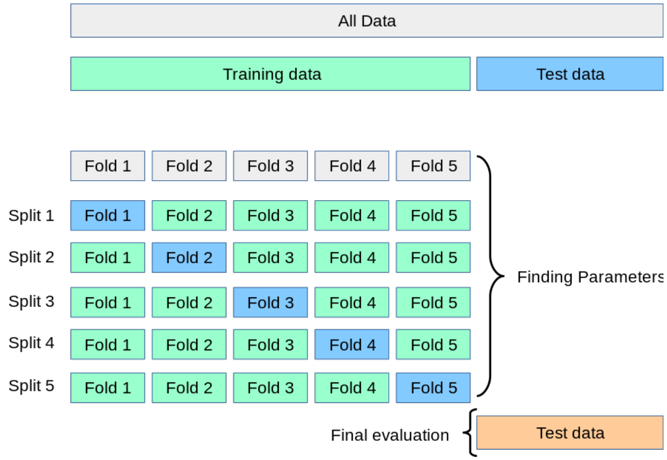
- We can also use cross-validation to prune a tree
- Again, leave out some data as test data
- Let the algorithm select the next split for you (grow one branch)
- then predict use the new and old tree for test data
  - if the new MSE is smaller, keep it
  - Otherwise, go back and delete the most recent split (pruning)

Cross-validation

- Imagine you have chosen a particular three-way split:
  - 80% as training
  - 20% as validation
- Sometimes the concern is that you the particular 80% may be different from the entire sample
- A more complex and popular approach is to use K-fold cross validation

Cross-validation

- K-fold cross validation (below example shows $K = 5$)



Summary

- ML algorithm evaluations:
  - Continuous outcome: MSE
  - Binary: precision/recall; false positive/false negative rates; ROC curve
  - No more “this is the best, magic, computer algorithm that does blah blah”
  - Instead, present empirical evidence
- Bias-variance trade-off
  - Definition; overfitting
  - Regularization
  - Train-test split
    - Allows more faithful evaluation of prediction performance
    - Allow you to select tuning parameters
    - Allow you to select the best algorithm