Advanced Models II

Practical Machine Learning (with R)

UC Berkeley

Fall 2015

Topics

- Review and Expectations
- Questions
- New Topics

Remaining Topics

Today

- Custom loss functions
- Boosting
- Deployment
- Neural Networks

Next Time:

- svm
- regularization/shrinkage : ridge and lasso
- strategies and “patterns”

REVIEW AND EXPECTATIONS

REVIEW

- Trees
  Tree Variants
  Surrogate Splits (missing Data)

- Rules
  (Related to Trees, relaxes the Path constraint)

- Treatment of Categorical Variables
  - Grouped vs Independent

- Tuning Parameters

Two Big Ideas

- **Wisdom of the crowds**
  It is better to make estimates from multiple models (***ensembles***) than individual models
  - Better predictions
  - Lower variance for the same model

- **Greed is bad. Patience is good.**
  It is better to slowly approach your solution than arrive at an answer directly

Model Improvement

- Ensembles / Model Averaging
  Effect: reduce variance of model

  - M5 Trees (Regression in nodes)
  - Bagging (Bootstrap Aggregation, any learner)
  - Random Forest
    - Advantage / Disadvantages

Random Forest Advantages

- No overfitting
- More trees better (limited by computation time/power only)
- In caret, parameters are considered independently
- Because each learner is selected independently of all previous learners, randomforests is robust to a noisy response
- Computationally efficient -- each tree built on subset of predictors at each split.
- Use any tree variants as "base learner": CART, ctree, etc

Custom Loss/Cost Functions

- Most methods use "0-1" or "simple" loss, where all errors treated equal
  - Overly simple
- Real world

$$\begin{matrix} 0 & L_{FN} \\ L_{FP} & 0 \end{matrix}$$

Binary/two-class

$$\begin{matrix} 0 & \varepsilon_{12} & \varepsilon_{13} \\ \varepsilon_{21} & 0 & \varepsilon_{23} \\ \varepsilon_{31} & \varepsilon_{32} & 0 \end{matrix}$$

N-class

- Mostly applies to classification problems
- Not to be confused with error metric for training

Custom Loss:

Native Cost/loss Matrix

Used during training

- For example

```
rpart( parms = list(loss=L) )
```

➲ **Best results**: all aspects optimization are optimized through the custom costs

➲ Unfortunately, not every methods support custom loss functions

Custom Loss:
Class Probabilities

- Use class probabilities … adjust class based on probabilities
- Difficult/time consuming to implement in practice

Custom Loss:
Two-class Classification Problem

- Use case weights based on the benefit/cost of the correct classification

```
Fraud ~ . , weight=dollars
```

  - Emphasis given to cases that are more important, less likely of erroring here
  - Not all methods support case "weights" argument
  - Works well for rare-event detection problem
  - Lose 0-1 predictability

Custom Loss:

Two-class Classification Problem

- Define "benefit" against naïve case
  - e.g. assume all transactions are good.

$$
\begin{array}{ccc}
. & F & G \\
F & 0 & -\alpha \\
G & -x & +\delta
\end{array}
$$

outcome	benefit
F	$-x$
T	$+\delta$

- Models "expected" benefit from alt. decision
- Sign determines classification /action
- Does not account for all error terms (e.g. FN)

CUSTOM LOSS:
TWO-STEP MODEL

- Use two models
  - **Model 1:** Unmodified Classification model
    - Goal best classification prediction available
    - Assume this is the how the class will be identified
  - **Model 2:** Evaluation Model
    - **Model 1** provides both the response and predicted response
    - Calculate benefit (–error) of each poss. class, given the predicted class
    - New Response = class with the highest benefit for each case
  - **Benefit:**
    Separates classification model from benefit

QUESTIONS

NEW TOPICS

Boosting

- Single models work;
  - Multiple models work better

- Idea is simple:
  - **Fit first** model: $\hat{y}_1 \sim f_1\,(x)$
  - **Fit** errors/residuals: $$\begin{aligned}\hat{y}_2 &= f_2(y - \hat{y}_1)\\ &= f_2\big(y - f_1(x)\big)\\ &= f_2\,(x)\end{aligned}$$
  - **Iterate:** $\hat{y}_i = (y - \hat{y}_{i-1}) \sim f_i\,(x)$
  - **Predict**: $\hat{y} \sim \sum_i f_i\,(x)$

Boosting Notes

- Additive models
- Works best with "weak learners"
  - i.e. ungreedy, low bias, low variance
  - ~~Any~~ Most models with a tuning parameter can be a weak learner
  - Trees are excellent weak learners
    - Weak → "restricted depth"
- Residuals or errors define a gradient
- Interpreted as forward step-wise regression with exponential loss

Simple Gradient Boosting

1 Select tree depth, D, and number of iterations, K

2 Compute the average response, $\overline{y}$, and use this as the initial predicted value for each sample

3 **for** k = *1 to* K **do**

4   Compute the residual, the difference between the observed value and the *current* predicted value, for each sample

5   Fit a regression tree of depth, D, using the residuals as the response

6   Predict each sample using the regression tree fit in the previous step

7   Update the predicted value of each sample by adding the previous iteration's predicted value to the predicted value generated in the previous step

8 **end**

Simple Gradient Boosting – Comparison To Random Forest

Similarities	Differences

Stochastic Gradient Boosting

- Gradient Boosting Susceptible to Overfitting
  - Apply "regularization/shrinkage"
    - Use λ ("Learning Rate")
      Rather than add the entirety of the residuals, add a fraction of the residuals at each iteration.

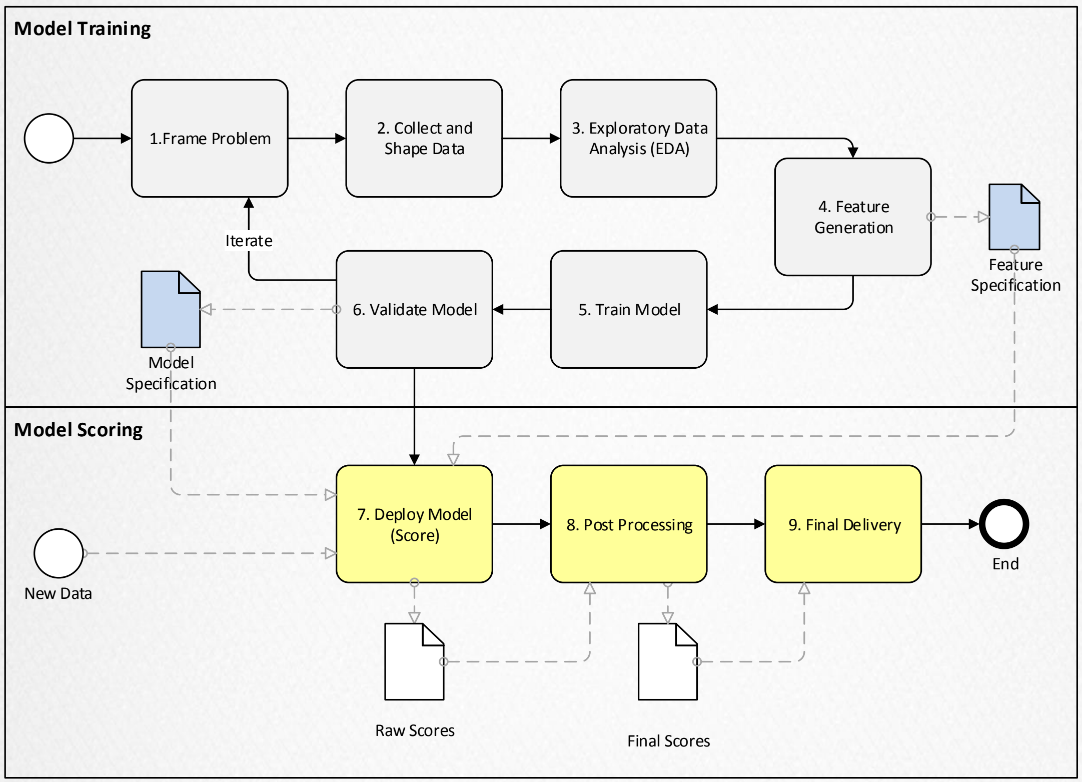
$$\hat{y} \sim \lambda \sum_i f_i(x) \qquad 0 < \lambda \leq 1$$

    - Small values for λ (~0.01) work best
    - λ ~ 1/computational time ~ 1/storage time
- Use bagging, as well
  - Bagging Fraction: a sample of data in each loop iteration

DEPLOYMENT

Deployment

- Making a model available to scoring to a larger group of users



General Consideration

Users	How many? Technical proficiency
Data	Single case < set of cases < universe
Frequency	How often scores accessed?
Latency	How much time tolerable to score?
Interface	Web UI (shiny) Command-line(optigrab) Rserve OpenCPU
Resources Req'd	Memory, disk

General Consideration

- Three steps
  - accept inputs from interface
  - apply logic
  - present / render findings

- Accept inputs /render findings
  - Dependent upon Interface
  - R / command line / shiny / web application

General Considerations : Application Logic

- Training / scoring data must be ***similar***
  - Every transformation made to create training data must be "replayed" on `newdata`
  - `scale` must use `center` and `scale` parameters from training
  - `impute` must use same distributions/data
  - `Etc.`

- standardize feature development

  [fetch data] → build features → build frame

  Separate functions

Native R

- Most flexible
- Requires R knowledge, tech. proficiency

- Best Practices
  - standardize model location and usage
    use R package features(?)
    - `data`
    - `fetch`
    - `featurize`
    - `build_frame`

    Deploying of model is simple as `library(mypackage)`

Command-line application

users access models from command prompt

```
> Rscript score.R --sepal.length 2.7 –sepal.width
```

Assumptions

- No specific R knowledge required
- Some technical proficiency

- Data passed in as **optigrab**
  - Actions bound to parameters
- Best with single-case scoring

- Can work with data sets if you can specify them

Shiny

➲ *Key Features*

- ***Reactive*** programming
  - Variable have dependency on other values
  - Update values when dependencies change

- Separated concerns
  - **`ui/ui.R`** (presentation) : high-level functions for widgets and layout
  - **`server / server.R`** (application) : application logic

ASSIGNMENT

Evaluating Variable Contributions In Complex Models

- How do understand the contribution of each variable?

- Retrain the model leaving out the variable: change is the contribution
- Randomly permit values in evaluation set and compare the performance (Brieman)
- Aggregate performance of variable across each model

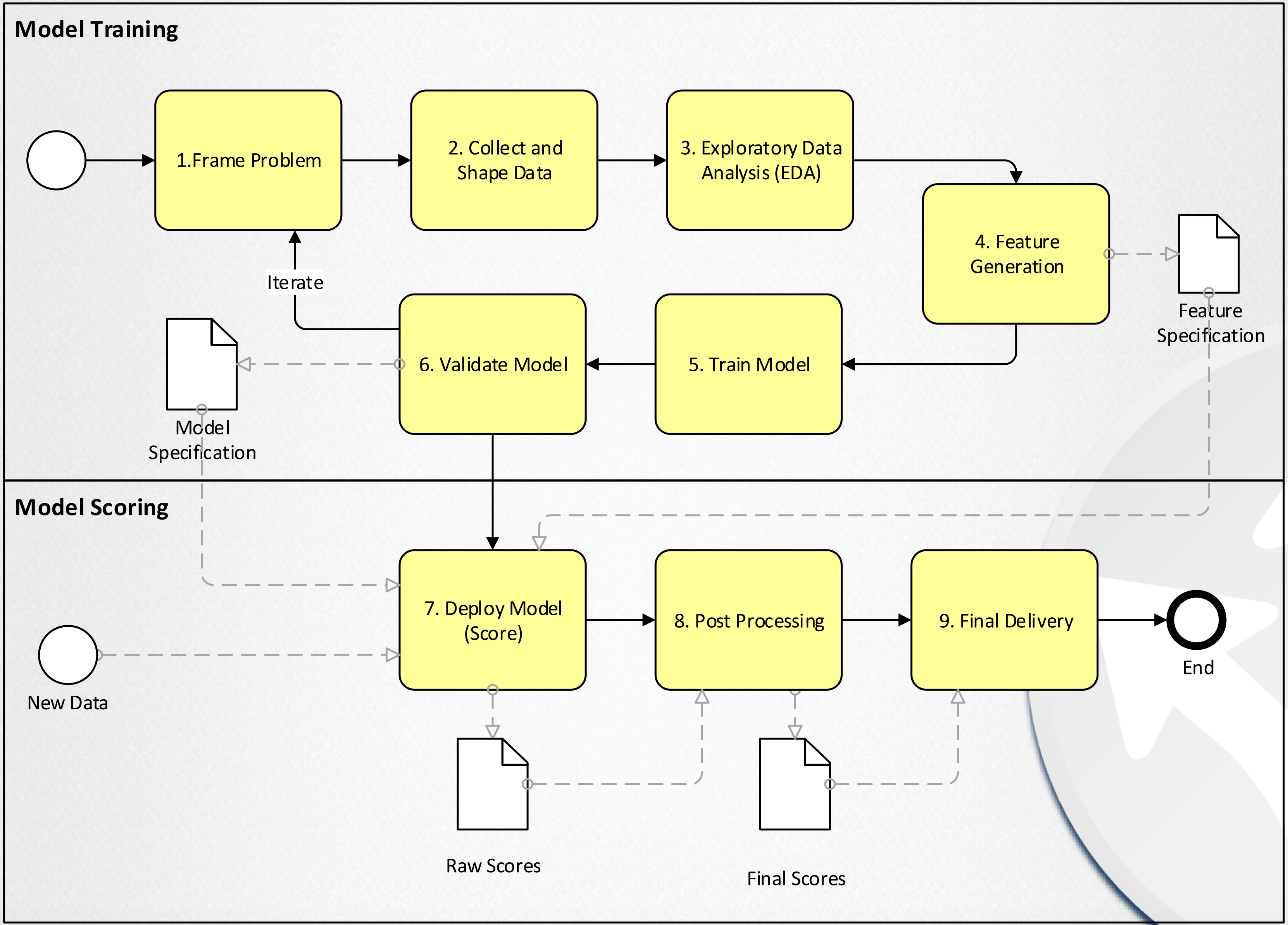
APPENDIX

ADABOOST (SHAPIRE/FREUND)

1 Let one class be represented with a value of +1 and the other with a value of -1
2 Let each sample have the same starting weight $(1/n)$
3 **for** k = *1 to* K **do**
4 Fit a weak classifier using the weighted samples and compute the kth model's misclassification error (err_k)
5 Compute the kth stage value as $\ln\left((1 - err_k)\,/err_k\right)$.
6 Update the sample weights giving more weight to incorrectly predicted samples and less weight to correctly predicted samples
7 **end**
8 Compute the boosted classifier's prediction for each sample by multiplying the kth stage value by the kth model prediction and adding these quantities across k. If this sum is positive, then classify the sample in the +1 class, otherwise the -1 class.

MULTI-CLASS PERFORMANCE

Comprehensive ML Process



TERMS

- Kappa Statistic,
- S-Statistics, F-Statistic

EXAMPLE OF ML ALGORITHM(S)

- Spam Filter
- handwriting recognition (svm)
- Traffic engineering (lights)
- Weather prediction
- Sentiment analysis (social media)
- Netflix Recommender
- Fraud detection (Visa)
- Imaging processing
- (network) Intrution detection
- Self-driving cars

Comparison of Models (Chart)