Producing and evaluating machine learning models

John Mount
Nina Zumel

All examples: http://winvector.github.io/DS/ModelTesting/

"Essentially, all models are wrong, but some are useful."

– George Box

Goal

- Learn about tools that allow you to produce, evaluate, and deploy powerful state of the art predictive models.
  - Data scientists become expert in all these steps.
  - Managers need to be expert in at least model evaluation.

Biography

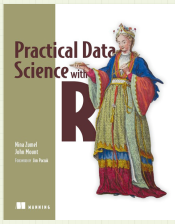
Nina Zumel
Win-Vector LLC
Dr. **Nina Zumel** is a principal consultant and founder at Win-Vector LLC a San Francisco data science consultancy and training company. Nina started her advanced education with an EE degree from UC Berkeley and holds a Ph.D. in Robotics from Carnegie Mellon University. Nina has worked as research scientist as SRI and developed revenue optimization platforms. She frequently writes and speaks on statistics and machine learning.

Nina is also the coauthor of the popular book of *Practical Data Science with R* (Manning Publications, 2014).

John Mount
Win Vector LLC
Dr. **John Mount** is a principal consultant and founder at Win-Vector LLC a San Francisco data science consultancy and training company. John has worked as a computational scientist in biotechnology and a stock-trading algorithm designer and has managed a research team for Shopping.com (now an eBay company). John started his advanced education in mathematics at UC Berkeley and holds a PhD in computer science from Carnegie Mellon.

John is also the coauthor of *Practical Data Science with R* (Manning Publications, 2014).

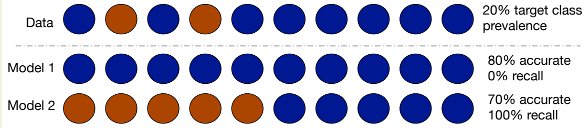
Please contact contact@win-vector.com for projects and collaborations. **http://win-vector.com/** Twitter: @WinVectorLLC .



What is a Good Model? Performance metrics for classifiers / decision procedures.

How do you measure model performance?

- Accuracy is not the only way



Which Metrics Are Appropriate?

Question	Metric	Example
Is it important that a positive classification is correct?	Precision	If the test comes back positive, is the patient really diabetic?
Is it important we find all positive cases?	Recall Sensitivity	Do we miss any diabetics through this test?
Are false positives expensive?	Precision Specificity	Diagnoses that lead to costly treatment
Are false negatives expensive?	Recall Sensitivity	Diagnosing conditions that are costly if untreated
Is it important to get everything right?	Accuracy	

Technical Metrics

- AUC (ROC), deviance, and others.
- Good metrics for data scientists and between data scientists
- Useful proxy measures for comparing candidate models
- Not always easily translatable to business goals

ROC/AUC

- Graph of trade-off between true positive and false positive rates as labeling threshold T is varied.
- AUC: area under the curve
  - Probability that a randomly chosen positive example will score higher than a randomly chosen negative example (with appropriate tie-breaking).
- Invariant to monotonic transformations of scoring function
- Independent of target class prevalence

Predictive modeling schematic

- Define a useful business goal.
- Choose a convincing performance measure.
- Collect input ("independent") variables.
- Build a model.
- Confirm you have a decisive model.
- Refine/repeat.

Big risks

- Not being able to produce a good model ("under fit").
- Not being able to falsify a bad model ("over fit").
- Model depending on variables that are really only available after the outcome is known ("data leakage").

Example: KDD2009

- KDD conference 2009 contest data.
- Predict from a few hundred features which credit accounts will "churn" or cancel.
- Training data: measurements from past accounts known to have cancelled or not cancelled in a fixed time interval.

Assume we have our data ready to go

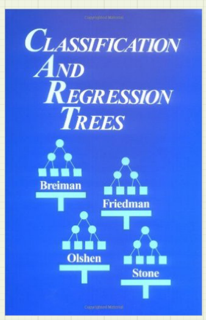
- Data acquisition, documentation, cleaning, and preparation is by far the largest most critical part of real world data science.
- For this demo we are going to assume this is done and move on to the part people always want to hear about: model construction.

Pre-packaged software

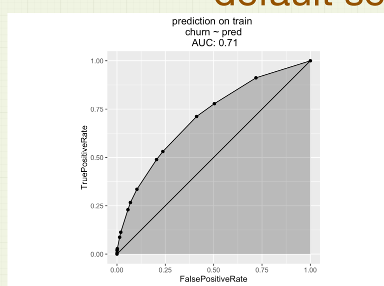
- You don't need a Ph.D. to perform machine learning, because people with Ph.D.s have already implemented and shared very powerful methods:
  - Gradient boosted trees.
  - Random forests.
  - Deep learning.
- For this demonstration we will exhibit R, decision trees, gradient boosting, and h2o deep learning.

First try

- Party like it is 1984.
  - Classification and Regression Trees

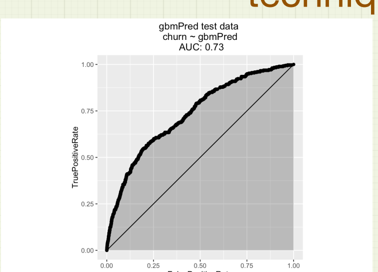


Decision tree model with default settings



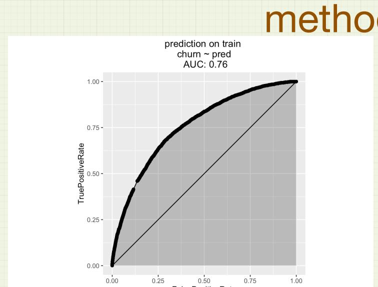
- Performance seems so-so.
- Winners of KDD contest reported AUC of 0.76.

Try more powerful modeling techniques



- gradient boosted trees
  - Based on Jerome H. Friedman's "gradient boosting machine" (1999).
  - All code:
    - https://github.com/WinVector/PreparingDataWorkshop/tree/master/KDD2009

Or even more powerful methods



- Neural Net / Connectionist / Deep Learning
  - In continuous development from the 1960s through now.
  - Shown here: h2O.ai "deeplearning" implementation.

Step back

- We have shown the typical “left to their own devices” data scientist workflow concentrating on improving models “in the lab” only on our training data.
- In a real application
  - The biggest modeling improvements come from commissioning new measurements and features.
  - High performance on training is not the true end goal, and must be viewed with suspicion (issues like over-fit, and data-leaks are important).
- What evidence do we have we actually solved the problem?

(switch)
Please stand by….

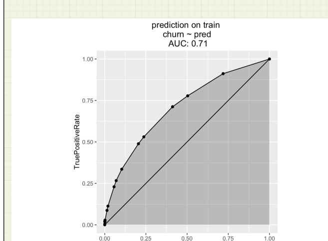
Is it *really* a Good Model? Estimating *out of sample* performance

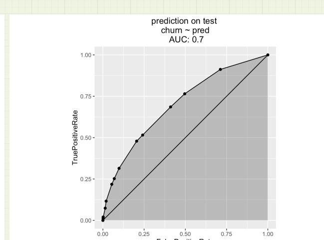
Question #1: Which model is the best?

	Performance on Training Data
Decision Tree	0.71
Gradient Boosting	0.73
Neural Net	**0.76**

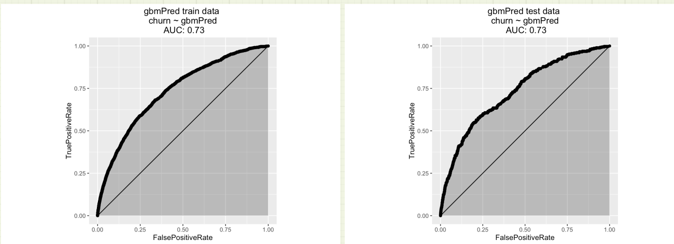
Now give the models new data:

Decision tree results

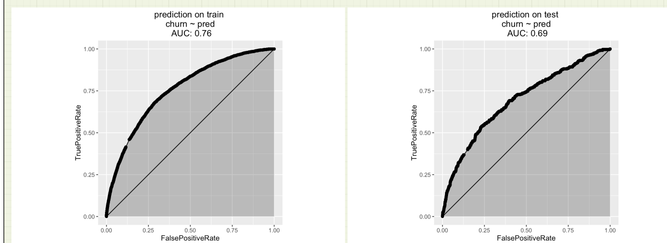




Gradient Boosting results



Neural Net / Deep Learning results



Best on train ≠ Best in future

	Performance on Training Data	Performance on Test Data
Decision Tree	0.71	0.70
Gradient Boosting	0.73	**0.73**
Neural Net	**0.76**	0.69

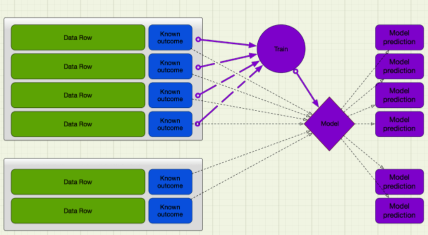
The problem

- Performance measures on training data tend to be upwardly biased or optimistic.
- We want a model that works well on future or new application data.

True estimate of out-of-sample performance: holdout data

Test-train split

- Subset of data only used for model evaluation
  - Simulates future application
- Can help find some issues such as over-fit.
- Blind to other issues such as concept drift and data leakage.



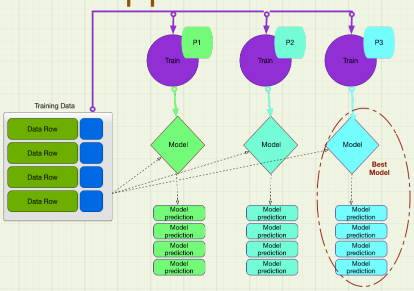
Question #2: How do you tune the modeling algorithm?

- How many trees for gradient boosting? How deep?
- What's the best learning rate for NN?
- How many iterations before you stop updating the NN?

Naive approach

Use Training Data to Pick Parameter

- Train one model for each candidate parameter P
- Pick the P the performs the best



Which gradient boosting model (xgboost) is the best?

Number of trees	Performance on Training Data
50	0.93
100	0.98
200	**1.0**

Again: Best on train ≠ Best in future

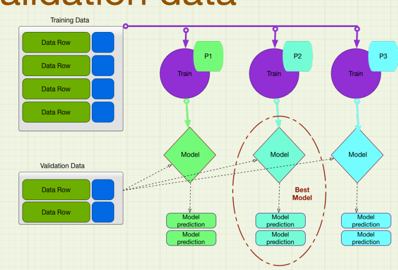
Number of Trees	Performance on Training Data	Performance on Test Data
50	0.93	**0.73**
100	0.98	0.72
200	**1.0**	0.70

Overfit!

Tune model with validation data

Train-Val-Test Split

- Fit candidate model(P) on Train
- Evaluate candidate model(P) on Val
- Re-evaluate final model on Test (not shown)



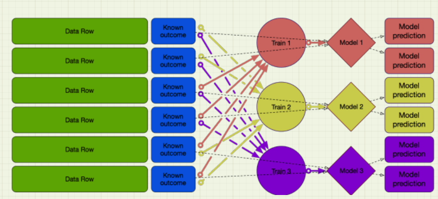
This is better, but….

- Validation performance estimates only unbiased the first time ("peeking problem")
- You need a LOT of data to split three ways
  - Bigger training set = better model
  - Smaller validation set = more peeking problems
  - "Statistically inefficient"
- Computationally efficient
  - One model per parameter setting

Estimating Out-of-sample performance with training data

Cross-validation

- No point is evaluated on a model it helped to train
- Use cross-val estimates to pick P
- Train final model with best P and ALL the training data



Cross-validation

- Less biased estimate of out-of-sample performance
  - "Peeking" issue slowed down
- Statistically efficient
  - Largest possible validation set for a given training set size
  - Final model is trained on ALL the training data
- Computationally inefficient
  - Fit N models for every possible parameter
- **Evaluates *modeling procedure* — NOT the performance of the final model.**

Cross-val vs. Holdout

	Statistically Efficient	Computationally Efficient	Evaluates Model	Evaluates Procedure
Cross-val	✓			✓
Holdout		✓	✓	

Data Science : Data-rich

- Generally, we will prefer train-validation-test split
  - Lots of data to spare for holdout
  - Large data sets make computational efficiency attractive
  - Possible exception: very rare target class

When to Consider Cross-val

- Data sets too small for train-validation-test split
- Lots of modeling parameters
- Rare target or rare features of interest

Cross validation and existing packages

- Many modeling procedures have cross-validation or validation set use baked in
  - Picking parameters, stopping criteria, etc.
  - gradient boosting with gbm, h2o neural nets….
- Reduces upward bias, but doesn't eliminate it
  - Still need a holdout set

Holdout: No peeking! (Or not too much)

- In practice: fit model->evaluate->tweak model …
  - Too many iterations and performance estimates are upwardly biased again
  - Especially if the holdout (or validation) set is small
- Recent differential-privacy related results to alleviate this
  - **http://www.win-vector.com/blog/2015/10/a-simpler-explanation-of-differential-privacy/**

Takeaways

- Knowing how a model will perform in the field is critical
- Data used in model construction may not be suitable for estimating this
- Train-Val-Test split and Cross-validation techniques can be used to fix some of the issues.

Thank you

All examples: http://winvector.github.io/DS/ModelTesting/