DIY Analytics for HMOs: How You Can And Why You Should

Roy Pardee
David Cronkite
KP Washington Health Research Institute

KAISER PERMANENTE

Your oral abstract presentation is scheduled for a 15-minute talk plus 3-minute Q&A on **Tuesday, April 9** at **8:54:00 AM - 9:12:00 AM**. Please note: following the oral presentations, there will be an 18-minute moderated Q&A session for all session presenters.

“Data is the new oil”

- Hype is inescapable.
- No shortage of vendors willing to help you spend money turning your data into “insights”.
- But the market seems to be bifurcated:
  - **Insurers** have claims, which should be comprehensive but imprecise with respect to patient health status.
  - **Providers** have clinical observations, which are detailed but often not comprehensive.
- As hybrid orgs, HMOs usually have a combination of these types of data.
  - So we can use *either.*
  - We should use **both.**
  - DIY approach allows this.

2

(Gonna give the ‘why’ before the ‘how’.)

Surely, none of you are going to escape the hype that I’m about to lay down here.

TODO: compile lists of products aimed at the 2 markets

Insurers
    Johns Hopkins ACG
Providers
    Epic systems’ voluminous offerings

Of course we’ve been doing this way before it was cool
    But mostly in a research context
    Catnip is mostly contributions to the academic literature—publications.

    This is a different orientation—this is brute-force prediction for fun and profit. We are not looking to unlock secrets of medicine or science here. If we do, great, but the cheese here is to make a prediction that we can act on in order to save $$.

The Good News

- No longer necessary to purchase software to get access to cutting-edge analytical methods.
- Free and open-source analytics software is:
  - High quality.
  - Fairly easy to use.
- Generally the choice is between R and Python.
- What follows is a case study of our experience with Python.

3

So we have the data. And the good news is there is high-quality software available *completely free of charge*.

TBH a large part of my motivation here was just to see if I could make these python bits work at all. It can be hard to know with open-source software what the knowledge pre-reqs are just to get to a functioning installation sometimes. I'm here to tell you that python—and specifically the Anaconda distribution thereof (https://www.anaconda.com/distribution/) is really quite functional out of the box.

Context: Learning Health System

- Partnership between Research and Care Delivery
  - Not Research
  - Aims at leveraging data to optimize care delivery
- One Goal: reduce inpatient admissions by 2%
- By optimizing the ops of our Care Management (CM) department.

4

KPWA's Complex Case Management program "coordinates the care and services of members with multiple chronic conditions and complicated medical/social needs often resulting in the extensive use of resources. The CCM program is designed to comply with the standards set for the by the National Committee on Quality Assurance (NCQA) and is integral to the accreditation for the health plan. The RN case managers are at minimum bachelor's level educated and are certified by the Commission for Case Management Certification (CCMC). The social work case managers are licensed independent clinical social workers (LICSW)."

Strategy: Have CM concentrate on patients likely to be hospitalized

- Tactic: Use Johns Hopkins' Adjusted Clinical Groups (ACG) software.
- ACG process takes the preceding 12 months worth of claims data as grist for its many useful indices and predictors.
  - Uses Professional, Institutional and Pharmacy claims.
  - But no clinical data.
  - Because: Marketed to Insurers.

5

https://www.hopkinsacg.org/

Context for this was our Learning Health System initiative, which is in part a collaboration between the Research Institute and the Health Plan @ KaPoW. I was brought on to wrangle data generally.

BTW—I do not mean to impugn the usefulness of ACG—it is definitely useful.

Side Quest: Can we predict hospitalizations our own selves?

- Cohort is everyone who had indication of CHF, Diabetes or Renal disease as of April 2016. N = 70,231.
- Data Gathered over the **three** months prior to April 2016:
  - Age/Sex
  - All pharmacy fills (by RxCUI)
  - All dx codes
  - All px codes
  - All VDW lab tests, including the abnormal indicator
  - All BMI measures, categorized
  - All blood pressure measures, categorized

6

That inspired me to dip a toe in the Data Science waters.

Note that we're using 9 fewer months than ACG. That was an arbitrary decision on our part—it's very possible we would have gotten around to extending that if my results weren't so good (and of course nothing stops us from extending it to see if we can get even better).

Ironically(?) we used ACG's assessment of those 3 conditions to identify the cohort.

Note how indiscriminate we are about the data that go into the modeling. The vast majority of that will be irrelevant. The Data Science Way is to let the rock tumbler of machine learning sort the wheat from the chaff.

Details on both cohort definition & feature assembly can be seen in the program get_features.sas in the github repository named on the thank-you slide (

Machine Learning Approach

- Algorithms “learn” from the data (“features”) to make predictions.
- Python’s scikit-learn library (https://scikit-learn.org) provided a wide variety of classifier algorithms for us to try.
- We used actual hospitalizations in the 12 months starting in April 2016 as our gold standard.
- In General:
  - Feed a subset of your data (“features”, “variables”) into a classifier along with the gold standard.
  - The classifiers “learn” to make predictions from this data.
  - You then sic them on a held-back subset of data.
  - If the predictions match your gold standard, you’ve got a good classifier.

7

Dig how brute-force this is.

Machine Learning Boogeyman: Overfitting

- Even when you know what you're doing, there's a ton of iterating in this work.
  - Parameter tweaking.
  - Feature engineering.
- Poses the risk that you will torture your models until they essentially "memorize" your training data.
  - Which makes for excellent predictions—on your training data.
  - And horrible performance in new samples.
- Traditional stats hypothesis testing relies on mathematical argument.
- But with data science you justify belief *empirically*.
  - You should believe my predictions **because I'm demonstrating that they work**.

Ultimately, we need an answer to the question "why should I believe your predictions?"

In the olden days, you'd have to be **very** stingy with the amount of iterating you could do and still rely on the logic of significance testing to make the argument that your results should generalize to new samples.

(Harken back to your stats professors railing against stepwise regression. That was one of the first iterative, automated model selection techniques. The primary reason that was bad was that it involved doing waaaay too many significance tests w/out protecting the overall experiment-wise error rate.

For a representative screed against stepwise, see e.g. https://towardsdatascience.com/stopping-stepwise-why-stepwise-selection-is-bad-and-what-you-should-use-instead-90818b3f52df.)

Data Science Solution: Partition Into Development and Validation Samples

Table 1: Cohort Partition by Hospitalization Status

Frequency Row Percent Col Percent	Hospitalized?		
	No	Yes	Total
Development	49,411 87.93 80.00	6,781 12.07 80.05	56,192 80.01
Validation	12,349 87.96 20.00	1,690 12.04 19.95	14,039 19.99
Total	61,760 87.94	8,471 12.06	70,231 100.00

But we're data rich now!

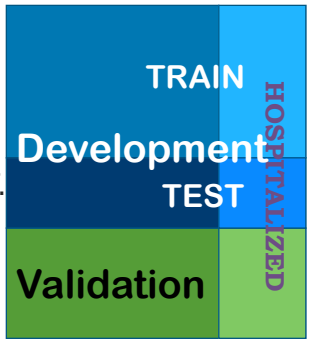
80% dev and 20% validation.
Note that 12% of our cohort has been hospitalized. These are pretty sick people.

'Hospitalization' here excludes accidents & pregnancies—see the list of excluded DRGs starting on line 72 of get_features.sas.

So—dev is the playground. We can torture that for as long as we like, commit unspeakable statistical crimes upon it etc. Validation is our safety net—that's the basis of our argument later on that whatever we put together to predict in dev will be useful beyond that set of data.

Development Process

1. Read the Development data out of the database.
2. Split Development into two-thirds TRAIN and one-third TEST (stratified by Hospitalization status).
3. Train various classifiers (random forest, support vector, etc.) on TRAIN
4. Use those trained classifiers to predict the cases in TEST.
5. Evaluate performance.
6. If performance is unsatisfactory **and** we can think of something to tweak, do those tweaks and return to #1.
7. When done iterating, evaluate classifiers in the held-back Validation data.



Here's the iteration I was talking about.

Note that we're not bringing any actual clinical or even statistical knowledge to bear on this problem. This is very much a throw-it-against-the-wall-and-see-what-sticks approach.

One very nice thing about the scikit-learn libs I was able to use is that their programmatic interfaces are pretty uniform. So I was able to write a single function that accepted a generic classifier and the training/test data and would spit out predictions & statistics.

Note also that we don't touch the validation data in this process at all. That is our ultimate backstop on overfitting—if we overfit development, validation will smack us upside the head.

How'd We Do?

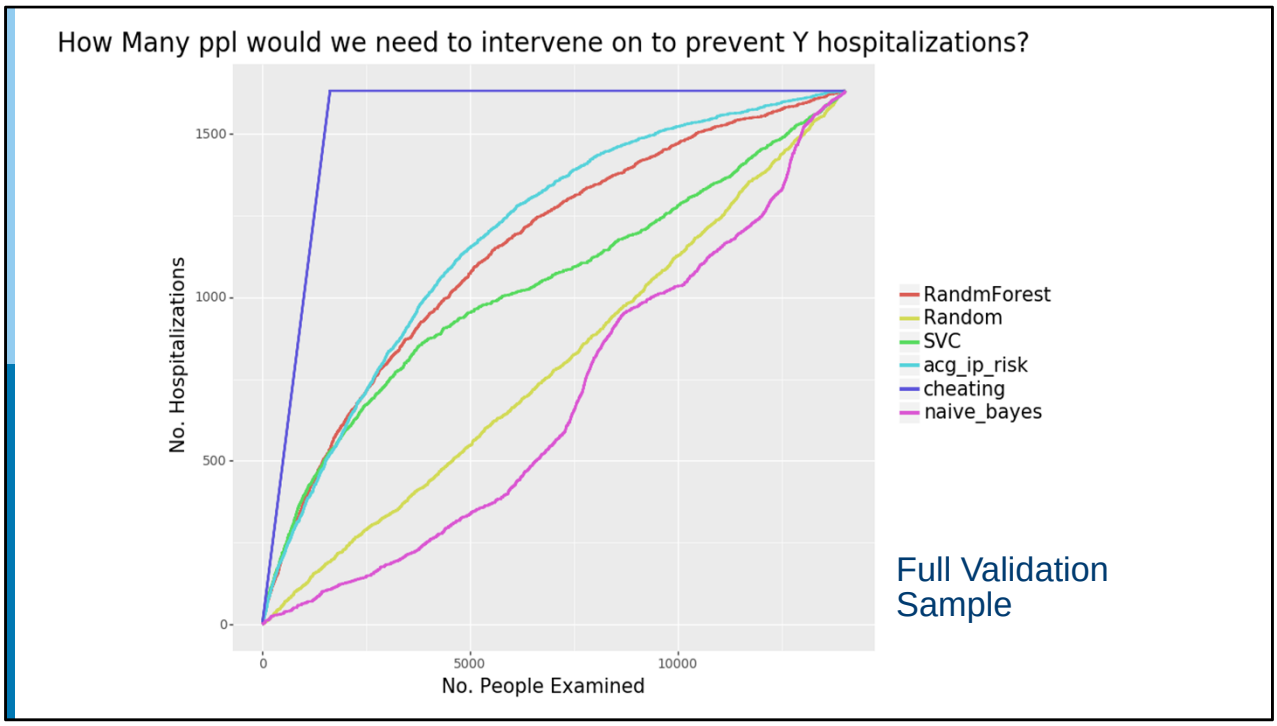
- This is a **triage** task—goal is to help Care Managers *prioritize* their efforts.
- So what we really want is a good *ranking* of most-needy to least-needy.
- To evaluate a given predictor:
  1. Sort the data from most-likely-to-be-hospitalized to least-likely.
  2. Run down the line counting the cumulative number of people who were in fact hospitalized.
  3. Plot the number of people examined by the number of hospitalizations detected.

The faster the Y values climb, the better the predictor.

11

A good predictor will pack the head of the line with people who were in fact hospitalized.

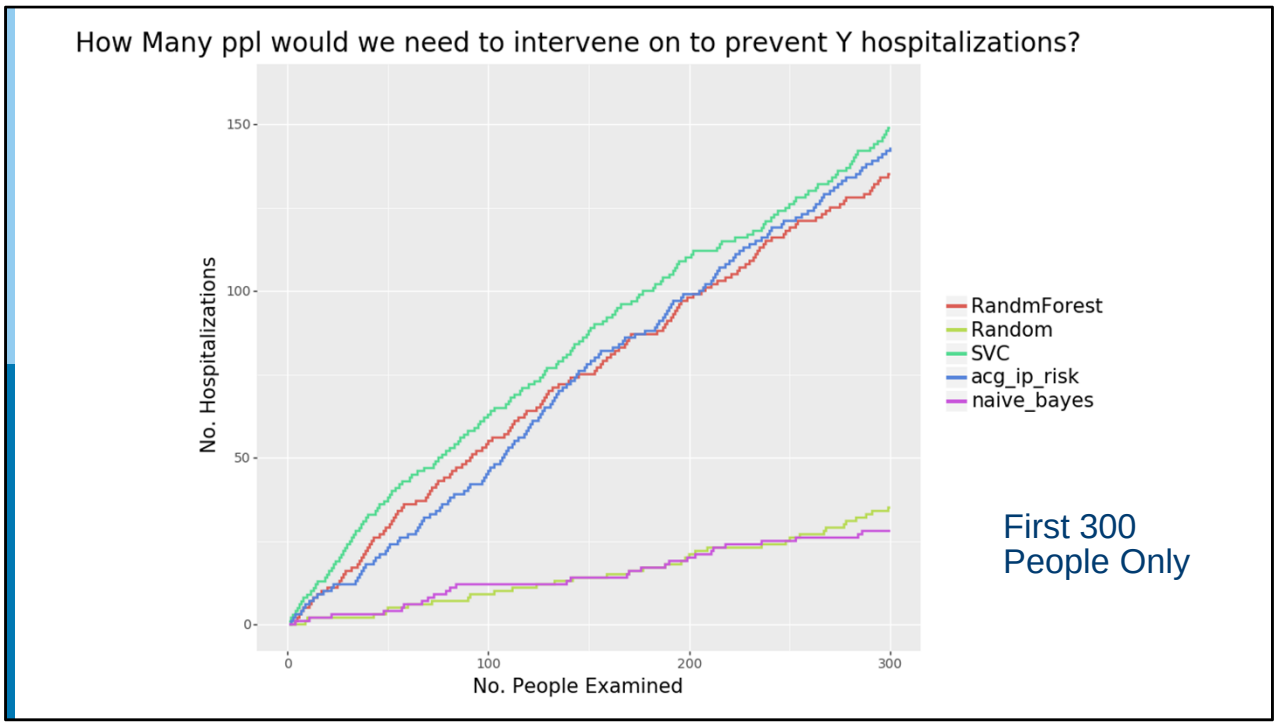
(Would it be enough to say "number needed to treat" here?)



Couple of lines of particular interest:

1. The purple 'cheating' line is what we get if we order people by their true status—we put everyone who was in fact hospitalized up to the beginning of the line, and have godlike, perfect 'prediction'. That represents the absolute best that any predictor could possibly do. Nobody is in that neighborhood—not even ACG.
2. The dull yellow-green 45-degree line in the middle is what we get if we order people randomly. That's what we should expect to see from a complete bullshit, snake-oil predictor. Astoundingly, Naïve Bayes manages to be worse than this. Possibly we're using it incorrectly somehow—not sure.
3. The turquoise line is of course ACG—that's what we're shooting at. It's a pretty clear favorite over the entire range of the development sample here. Pre-2,500 people there are two candidates that give ACG a run for its money, but after that ACG pulls away.
4. BUT! On the left hand side of the axis, we have two candidate scores that look pretty darn good—Support Vector Classifier and Random Forest. That's a pretty important part of the graph actually, because just like you don't normally have time to page through the 2d, 3rd, 4th pages of google results that you get on a

search, CM folks don’t have time to get involved with 2500 people. So all is not lost here. If we can deliver say, the top 300 people most likely to be hospitalized, we’ve got a win.



Support Vector classifier beats ACG over this whole range!
Random Forest is ahead for the first 135 people or so.
Both are worthy alternatives to ACG.

This looks like success to me, and it's likely that we could improve this significantly from here if we spent more time/added more data.

Conclusions…

- Created a predictor just as good (for our purposes) as an expensive third party tool, using fewer months of input data.
  - Very little expertise, medical or otherwise.
  - $0 spent on software/hardware.
- Python + scikit-learn very usable.
- Ditto VDW Data.
- No reason to operate with one hand (insurance/clinical data) tied behind our back.

…and Caveats

- Retrospective nature of the data means all claims were in.
  - Real world low-latency data will be spottier.
  - Predictions may not be as good.
- Classifiers are generally Black Boxes.
- Not *Research*, really.
  - Not unlocking mysteries of the underlying mechanisms here.

Machine was Intel Core i-7 w/16GB of Ram, running 64-bit Windows 7.

Thank You!

https://github.com/rpardee/hcsrn_2019

roy.e.pardee@kp.org
david.j.cronkite@kp.org

KAISER PERMANENTE

Bonus slides follow that *may* be useful if I get questions.

Future Directions

- Switch
  - from using binary feature flags to counts of feature occurrences.
  - to ingredient-level RxCUIs (or supplement with?)
  - to lower-latency sources for dx/px (other sources are already daily).
- Try additional classifiers.
- Tweak Parameters.
- Expand population, either to additional diseases, or to the general population.
- Create additional predictors for Care Managers' ratings of success/impact.

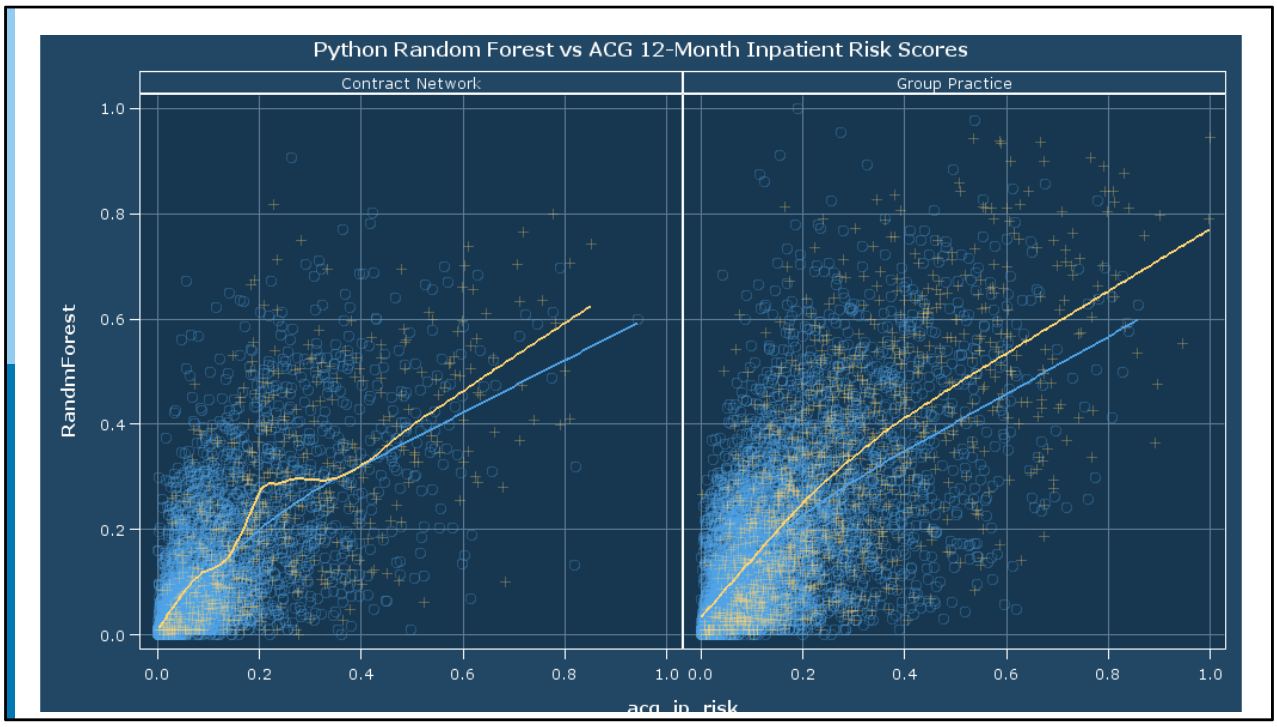
Unfortunately it's not clear I'll get to do any of this—LHS leadership was not enthusiastic about it, for reasons I still don't understand.

We Mashed The Data All Together In A Single Table

Person	Date	Type	Code	Result	Comments
Patty	::DOB::	agegender	60to64_F		Woman between ages of 60 & 64
Patty	5/3/2016	lab	PT:NL	15	Prothrombin time: normal range
Patty	5/3/2016	lab	INR:NL	1.2	International ratio: normal range
Patty	5/3/2016	bp	high	130/76	Blood pressure: high category
Patty	6/8/2016	Rx	310429		Furosemide 20mg tab
Patty	6/8/2016	Dx	Z85.820		Personal history of malignant melanoma of skin

17

The features we fed in to the ML algorithms was a concatenation of the type & code fields. Each one was treated as a distinct 'variable' if you will. You can see the actual implementation in ml_hospitalizations.py in the github repository cited in the thank-you slide.



In case anybody is curious about how the predictions compare on individuals. The paneling here is done on the basis of delivery system. At KaPoW we have basically 2 DSs—Group Practice patients are people who get the vast majority of their care from KaPoW owned/operated clinics. Contract Network patients experience KaPoW as pretty much just an insurer (so ACG should feel right at home in that pop).

It would be interesting to see NNT diagrams that were stratified by delivery system.

Bonus: What is a Random Forest?

- A Classification Tree is a simple method for making a decision.
- A Random Forest:
  - Creates B different bootstrapped samples from the input training data.
  - Creates a classification tree for each one.*
  - When predicting new data, it gets run through *every one of the B bootstrap-generated trees*.
  - The prediction becomes the classification reported by the majority of trees in the forest.

* The twist is that at each level of the tree, when choosing a feature to split on, only a random subset of features are considered.

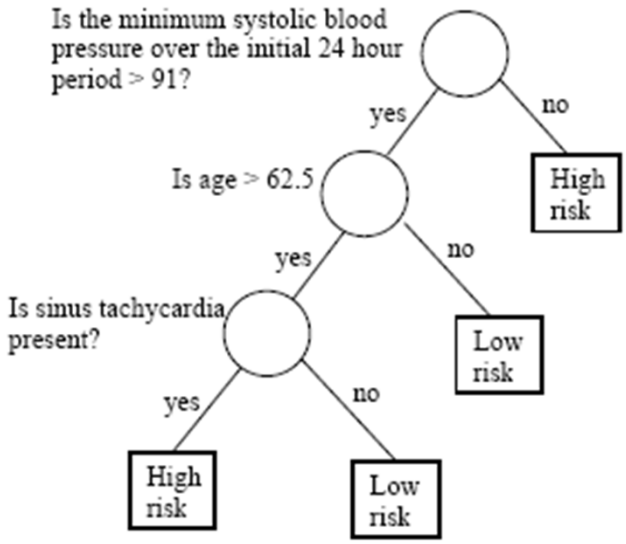


Image from https://onlinecourses.science.psu.edu/stat857/node/22/

In my case I accepted the default number of trees, which is 10. Thinking on it I should probably switch to an odd number of trees so there's no chance of ties.