Data Science: Street Fighting Statistics

John Mount, Win Vector LLC

Introduction

I am a Principal Consultant at Win Vector LLC. I have a Ph.D in computer science from Carnegie Mellon University, using probabilistic methods to prove convergence rates of Markov chains in optimization and sampling applications.

Co-author *Practical Data Science with R*

Co-author of several R packages

- vtreat
- wrapr
- cdata
- WVPlots

Author of several Python data science packages

- vtreat
- data_algebra
- wvpy

Statistics versus Data Science

There is a tension between statistics and data science.

- Statistics emphasizes model identification and inference.
- Data Science emphasizes quality of model predictions.

Either field can be made to look bad by judging it in terms of the other's concerns. Neither field can claim priority in consulting (that would be operations research!).

I will show what it looks like to rush from data to building a predictive model (supervised machine learning). Also, I will demonstrate some of the joy of working fast in idiomatic R.

Supervised Machine Learning

The task data scientists tend to be interested in is “supervised machine learning.”

- You are shown a table with rows of the form `(X, y)`: try to find a function that that `f(X) ~ y`.

Examples:

- Predicting childrens’ heights from parents’ heights.
- Predicting cancer risk.
- Predicting the probability of clicking on an online advertisement.
- Mapping a picture to a description (f(X) ~ y)
- Mapping a description to picture (f(y) ~ X)

(tends to ignore issues of cause!)

Problems and Practices

Judging a process merely by the results can simplify things, but also obscure important distinctions. This is the risk of “fit to finish” processes that are judged only on their final output.

It can seem that the practice of data science is memorizing an infinite number of arcane rituals. Many of the tool suppliers *would like it to be that way*.

Instead of this magical thinking, organize procedures by what they purport to fix.

Gregor Sailer, The Potemkin Village

The Fixes by what they Purport to Fix

defensive ritual(s)	legitimate fear(s)
regularization, variable pruning, principal components analysis	over-fit, **co-linear variables**
out of sample evaluation, cross methods, ensemble methods, bagging, maximum entropy methods	over-fit
re-weighting data	heteroskedastic errors, concept drift, **unbalanced classification classes**
non-linear outcome transforms, neural nets, tree methods	heteroskedastic errors, non-linear response structure
interactions, machine learning, neural nets, tree methods, boosting, stacking	under-expressive models, don't know structure of relation
hyper parameter search	unstable models, under-specified modeling techniques
Bayesian inference	neglecting prior knowledge and unobserved state

This Talk

In this talk I'll discuss predictive modeling, and two of the earlier bugbears:

- co-linear variables
- unbalanced classification classes

This is a chance to review some "street fighting statistics in R."

All slides and material here: https://github.com/WinVector/Examples/tree/main/Street_Fighting_Statistics.

Co-Linear Variables

Variables that imitate each other, which to use?

Co-Linear Variables, some data

Consider the following famous data set, Galton's height data. Galton didn't start with multi-variate regression, so he introduced a variable called "mid_parent".

```
d <- read.table("galton-stata11.tab", header = TRUE)
d$mid_parent = (d$father + 1.08 * d$mother) / 2
d |> head() |> knitr::kable()
```

family	father	mother	gender	height	kids	male	female	mid_parent
1	78.5	67.0	M	73.2	4	1	0	75.43
1	78.5	67.0	F	69.2	4	0	1	75.43
1	78.5	67.0	F	69.0	4	0	1	75.43
1	78.5	67.0	F	69.0	4	0	1	75.43
2	75.5	66.5	M	73.5	4	1	0	73.66
2	75.5	66.5	M	72.5	4	1	0	73.66

Held Out Evaluation

In deployment or production most data we see was not available when the model was trained! We try to simulate this for our evaluation by using held-out data.

```
# build a per-family test/train split
families <- unique(d$family)
train_families <- sample(
  families,
  size = 0.8 * length(families),
  replace = FALSE)
d_train <- d[d$family %in% train_families, ]
d_test <- d[!(d$family %in% train_families), ]
```

A Model

Here is a standard model. Notice `mid_parent` is suppressed as it is co-linear with some combination of `father` and `mother`.

```
model_1 <- lm(  # standard OLS model
  height ~ father + mother + mid_parent,
  data = d_train)
summary(model_1)$coefficients |> knitr::kable()
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	19.7779649	4.5749414	4.323108	1.76e-05
father	0.3932891	0.0513849	7.653786	0.00e+00
mother	0.3076535	0.0524606	5.864472	0.00e+00

Picking one variable over another

Another Model

```
var_cols <- c("father", "mother", "mid_parent")
model_2 <- glmnet::glmnet(  # L2 regularized model
    x = as.matrix(d_train[ , var_cols]),
    y = d_train[["height"]],
    alpha = 0,
    lambda = 1e-3)
model_2$beta |> as.matrix() |> knitr::kable()
```

	s0
father	0.4096798
mother	0.3254056
mid_parent	-0.0328211

How *effectively* different are the models?

Indistinguishable.

```
preds_1 <- predict(model_1, newdata = d_test)
sqrt(mean((d_test[["height"]] - preds_1)^2))
```

```
[1] 3.416253
```

```
preds_2 <- as.numeric(
  predict(model_2,
          as.matrix(
            d_test[ , var_cols])))
sqrt(mean((d_test[["height"]] - preds_2)^2))
```

```
[1] 3.416255
```

```
sqrt(mean((preds_1 - preds_2)^2))
```

```
[1] 7.62149e-05
```

How *semantically* different are the models?

Very Different.

- The traditional model has made the decision to suppress `mid_parent`.
  - This means the model is immune to any harm this variable could inflict in the future. Say for example some day in the future it is calculated or stored wrong in our data source.
  - Actually a super important decision, perhaps best left to the practitioner to decide which of `father`, `mother`, or `mid_parent` should be dropped.
- The L2 regularized (or Ridge Regression or Tikhonov Regularized) model keeps all variables, but tries to enforce small (near zero) coefficients.
  - This is a “fire and forget strategy.”
  - Works well if the average of the variables is more stable than the individual variables, as it often is.

The Fit to Finish Dilemma

Machine learning or AI-training builds a model that is *superficially indistinguishable from a correct model*.

- Often best possible on training data. In the absence of over-fit issues may be nearly best possible on hold out data.
- The statistics view
  - May still be the wrong model, with wrong coefficients inferred.
  - Model identification is important, as the model may be called on an example not similar to the training data.
- The data science view
  - "I am paid to call `predict()`, we are done here." (For many business situations, this is in fact the right answer!)
  - Model identification can be pointless when don't have the right model structure, and without that we can't expect reliable predictions for distributionally different examples.

Some Commentary

- The claimed above statistical view hopes to estimate the correct *conditional* distribution `P[y | X]` for *all* `X`.
- The claimed above data science view hopes to estimate the correct *joint* distribution `P[X, y]` for *only* `(X, y)` exchangeable with the training data.

Not the same problem (despite Bayes' Law)! Either can be warped into a criticism.

Unbalanced Classification Classes

Tools deforming the hands

Data scientist over-worry about unbalanced classes in classification problems.

- This distinction is a bit stronger in the Python data science community as the Python sklearn `.predict()` interface returns class labels instead of links or probabilities.
- The R `predict()` interface doesn't have this issue.
- R users are more comfortable with statistical concepts such as uncertainty, probabilities, odds-ratios, and log-odds-ratios.
- Some modeling tooling exists to support fixes that repair problems introduced by other fixes (complexity death spiral).

A Classification Example

For our classification example, let's take a BCSC breast cancer data set.

For this model `breast_cancer_history` codes to 0 (no cancer), or 1 (cancer) for all instances with known cancer status. Each row is weighted by how many examples like the given row are in a larger ideal data set.

```
d_orig <- read.csv(
  "bcsc_risk_factors_summarized1_092020.csv.gz")
d <- d_orig[d_orig$breast_cancer_history %in% c(0, 1), ]
sum(d["breast_cancer_history"] * d["count"]) /
  sum(d["count"])
```

```
[1] 0.1034583
```

The prevalence of cancer, among those instances with known cancer determination, is about 10.3%.

Preparing the data

All the variables in this model are re-coded categoricals, so we need to force them to be treated as strings.

```
vars <- c(
  "year", "age_group_5_years", "race_eth",
  "first_degree_hx", "age_menarche", "age_first_birth",
  "BIRADS_breast_density", "current_hrt", "menopaus",
  "bmi_group", "biophx")
for(v in vars) {
  d[v] <- as.character(d[[v]])
}
```

Building a classification model

We can fit a logistic regression classification model on this data.

```
model <- glm(
  wrapr::mk_formula(
    "breast_cancer_history",
    vars,
    extra_values = list(d = d)  # to find weights!
    ),
  data = d,
  family = binomial(link = "logit"),
  weights = d[["count"]])
```

Using the model

Let's see what sort of cancer risk the model associates with each of our example situations.

```
predict(model, newdata = d, type="response")[1: 3]
```

```
          1           2           3
0.006407362 0.109373072 0.006369305
```

Remember each outcome we are trying to match is `0` or `1` (and each row in the data set can represent multiple individuals).

```
library(data.table)

data.table(d)[, .(count = sum(count)), by = breast_cancer_history] |> knitr::kable()
```

breast_cancer_history	count
0	1497378
1	172793

Probabilities

By default `predict.glm` returns probabilities or frequencies. We can see these match actual incidence on training data.

```
sum(predict(model, newdata = d, type="response") *
d["count"]) /
  sum(d["count"])
[1] 0.1034583
sum(d["breast_cancer_history"] * d["count"]) /
  sum(d["count"])
[1] 0.1034583
```

The Fallacy

In Python `.predict()` returns the class label, which is defined as “1” if the probability is at least `0.5` and “0” otherwise. There is absolutely no reason for the prevalence of this label to match the data set.

```
sum(
 (predict(model, newdata = d, type="response") >= 0.5) *
 d["count"]
) /
 sum(d["count"])
```

```
[1] 0.03301997
```

For very imbalanced outcomes, this prevalence may in fact be zero.

The pointless fix

Because of the above problem many practitioners re-weight their training data to have the same number of positive and negative cases.

- This fix is unnecessary.
  - However, it is a popular fix and writing articles how to perform this fix is a popular activity.
- The *imagined* need for the fix likely comes from thinking the useful output of a classifier is the class-label, when in fact the estimated probability is *far* more useful.
- *Always* insist on probabilities from classifiers, not "most likely class label."
  - One can always convert probabilities to labels later, but you can't do the reverse.

More on the evils of binning when you don't have a reason: https://discourse.datamethods.org/t/categorizing-continuous-variables/3402 (though I have used the technique as "poor man's GAM").

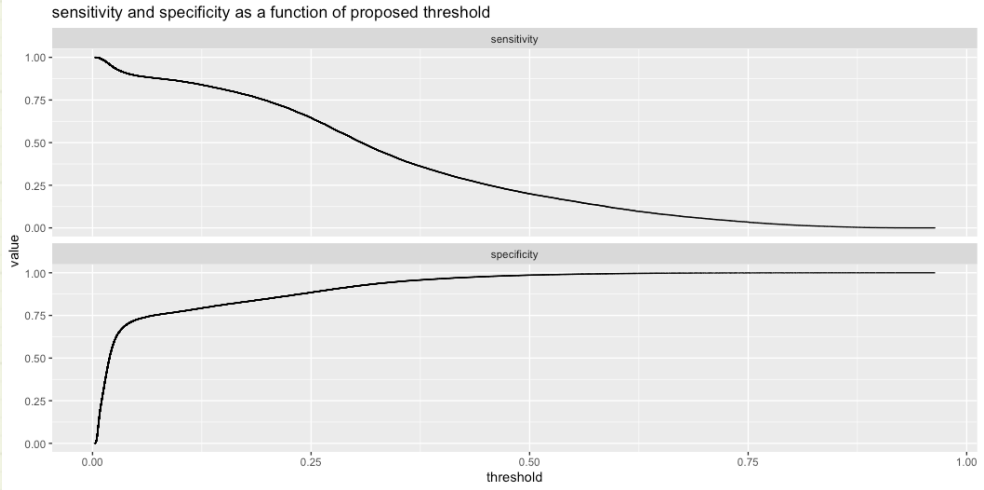
The Underlying Problem

- The view that classifiers should return categories is pernicious nonsense.
- In English “accuracy” is a synonym for “quality.”
- Accuracy is the most commonly asked for goodness metric for classifiers.
  - Accuracy is *almost never* the right metric when outcome classes are imbalanced
- Learn to use ROC/AUC as it tells you how often your classifier’s probability orders the outcomes correctly.

ROC plot

Don’t bother learning the ROC plot until you are comfortable with sensitivity and specificity. All the information in the ROC plot is also in this easy to read plot.

```
d["prediction"] <- predict(model, newdata = d, type="response")
d_expanded <- d[rep(row.names(d), d[["count"]]), ]
WVPlots::ThresholdPlot(
  d_expanded,
  xvar = "prediction",
  truthVar = "breast_cancer_history",
  truth_target = 1,
  title = "sensitivity and specificity as a function of proposed threshold")
```



Time to Wrap Up

Summary

- Be wary of applying too many or too few tricks in your statistical or data science practice.
  - You may need some
  - You should justify each one you use
- Statistics is the formal study of when samples correctly represent the population they are drawn from.
- Data Science is the optimistic application of machine learning methods at scale.
- Or: statisticians like to call `summary()` and `anova()`, data scientists are content to call `predict()`.
- In either case, the fears of the practitioner don't always match the fears of the theorist, *and* vice versa.

Some of our articles on these topics

Consider reading one, not all!!!

- Nina Zumel: “I don’t think that means what you think it means;” Statistics to English Translation, Part 1: Accuracy Measures.
- Nina Zumel: Squeezing the Most Utility from Your Models.
- Nina Zumel: Does Balancing Classes Improve Classifier Performance?
- Nina Zumel: Link Functions versus Data Transforms.
- Nina Zumel: The Simpler Derivation of Logistic Regression.
- vtreat for R, vtreat for Python (our own super trick for re-coding high cardinality categorical variables!)
- John Mount: Can a classifier that never says “yes” be useful?
- John Mount: When Cross-Validation is More Powerful than Regularization.

Thank You!