Data Scientists Summit (FY16): Data Preparation Using R:

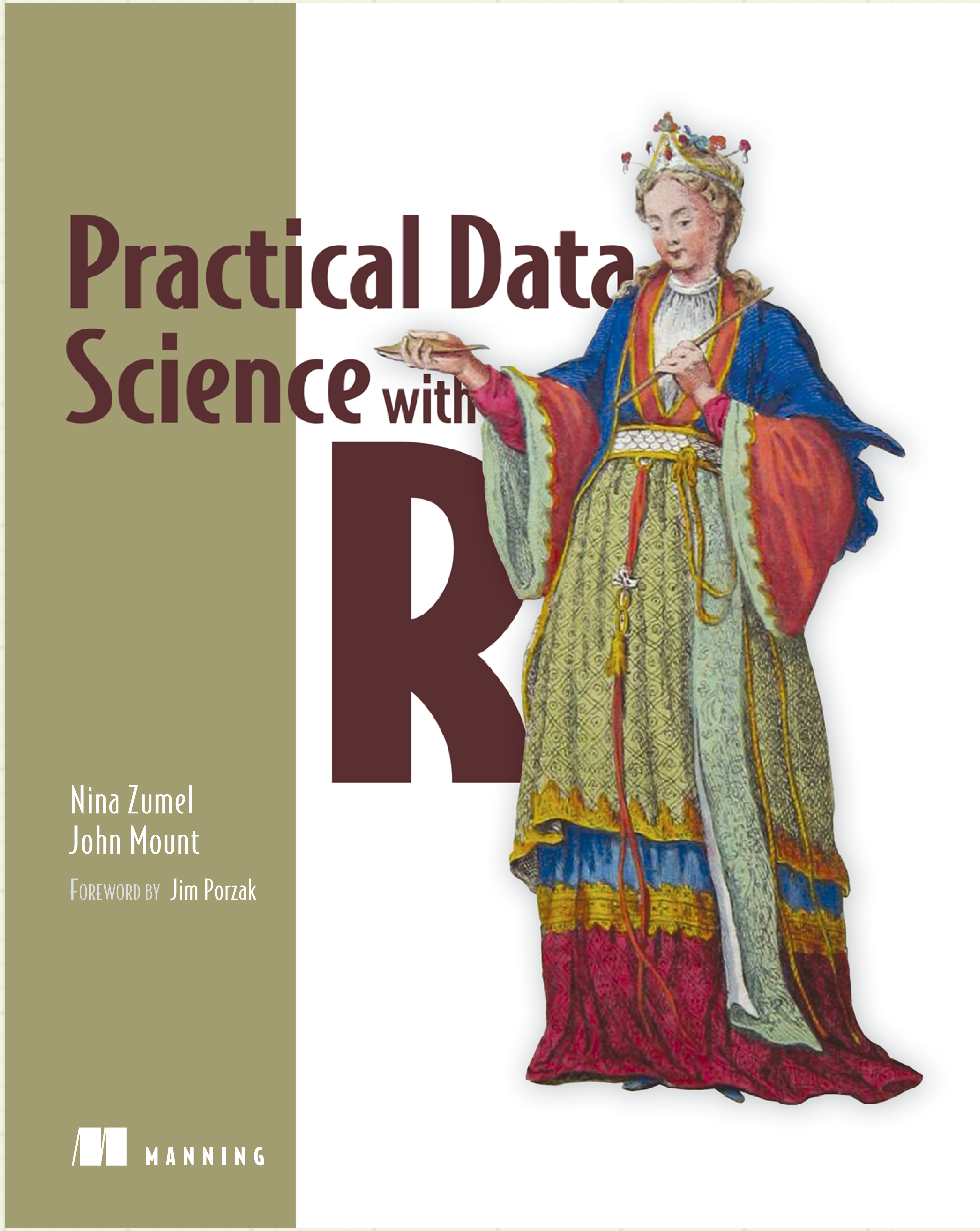
Basic through Advanced Techniques

John Mount & Nina Zumel
Win-Vector, LLC

All materials: https://github.com/WinVector/PreparingDataWorkshop

Who I am

- John Mount
- Principal Consultant at Win-Vector LLC
- One of the authors of Practical Data Science with R



Context

- We are going to discuss everything in the context of supervised learning. That is trying to learn a function f(x) such that f(x) ~ y on new data given many (x,y) training pairs.
- Machine learning view: predictive power coming from preserving statistical exchangeability of training or test data and future instances (not from actually inferring correct casual relations).
- Examples:
    - Regression
    - Classification
- Intended to be an exciting “not in front of the civilians” “what goes wrong and how to fix it” presentation.
- Illustrated with examples in “R” a command based analysis platform, but concepts valid/ important in any system.

Data preparation

- You do it because you have to prevent *possible* downstream modeling failures
  - Overfitting
  - Loss of signal
- *Not* because you are told to!

The 5 stages of data preparation grief

- **Denial**: "I don't need to pre-process data"
- **Anger**: "I shouldn't *have* to pre-process data as I am using sophisticated machine learning and cross-validation."
- **Bargaining**: "If I restrict myself to simpler models, no-effect priors, and regularization perhaps I won't see the problem in production."
- **Depression**: "Even if you need to pre-process, you can't as it introduces nested model issues and isn't statistically justifiable."
- **Acceptance**: "Nested models are everywhere and out of sample simulation is an effective extension of cross validation procedures."

Outline

- Part 1: data preparation as an operational issue.
- Part 2: data preparation as a statistical issue.

Part of the problem

- Statisticians reason about variables: well curated entities with known meaning, relevance, scale, and units.
- Data scientists tend to work with “columns”: messy undocumented, unproven modeling opportunities.

Outline of Part 1

- Data Preparation
    - Typical data problems & possible solutions
- `vtreat`: Automating variable treatment in R
    - An R based examples of automated variable treatment
    - Can be used for free or re-implemented in your own system
- Conclusion

Throughout this workshop

- We will keep an idealized goal in mind: using machine learning to build a predictive model.
- We assume we can delegate the modeling or machine learning to a library, and take on the responsibility for data preparation and cleaning.
- Having a single ideal goal allows us to apply seemingly “ad-hoc” fixes in a principled manner.
    - We can check if our “fixes” are for good or for bad.
    - We are not limited to mindlessly combining prior “name brand” procedures.

Data Preparation

Why Prepare Data at All?

- To facilitate modeling/analysis
  - Clean dirty data
  - Format data the way machine learning algorithms expect it
- Not a substitute for getting your hands dirty
  - But some issues show up again and again. Well worth automating treatment of these issues.

Typical Data Problems

- “Bad” numerical values (NA, NaN, sentinel values)
- Categorical variables: missing values, missing levels
- Categorical variables: too many levels
- Invalid values
    - Out of range numerical values
    - Invalid category levels

First Example: Bad/missing Numeric Values

Bad Numerical Values

Miles driven	Gas Consumption
100	2
235	0
150	7.5
200	5.5
0	0
300	NA

MPG
50
Inf
20
36.4
NaN
NA

Electric car/bad calculation

Non-numeric typo/ bad calculation

Electric car

Whither Bad Values?

- “Faulty Sensor” — values are missing at random
  - Assume they come from the same distribution as the other values
  - The mean of the “good” values is a reasonable stand-in
- Systematically missing
  - Electric cars
  - They WILL behave differently from gas or hybrid cars
  - The mean of the good values is not a valid stand-in

A number of possible solutions

- Naive: skip rows with missing values
- Multiple models: build many models using incomplete subsets of the columns.
- Imputation: build additional models that guess values for missing variables based on other variables.
- Statistical: sum-out or integrate-out missing values.
- Pragmatic: replace with harmless stand-ins and add notation so the machine learning system is aware of the situation.

Missingness as signal

- In business analytics missing data is often an indicator of where the data came from and how it was processed.

- Consequently it is often one of your most informative signals when modeling!

One Pragmatic Solution

MPG
50
Inf
20
36.4
NaN
NA

MPG	MPG_isBad
50	FALSE
35.5	TRUE
20	FALSE
36.4	FALSE
35.5	TRUE
35.5	TRUE

Critique

- Only gives a point estimate.
- Not as sophisticated as missing value imputation.
- With enough data nearly optimal for:
  - Linear models over independent predictors: extra degree of freedom enough to encode any non-interaction correction.
  - Tree based models (decision trees, random forests, boosted trees): extra variable enough to encode any interaction.

MPG
50
Inf
20
36.4
NaN
NA

MPG	MPG_isBad
50	FALSE
35.5	TRUE
20	FALSE
36.4	FALSE
35.5	TRUE
35.5	TRUE

Missing values are “good”

- Much easier to fix than undocumented “sentinel values”
    - 99 as a stand in for “age not known”
    - 38°N 97°W as “US address unknown” in MaxMind IP to geolocation database.
        - Associated 600 million IP addresses (including many thieves and scammers) to actual address of Joyce Taylor’s farm.
        - Many instances of attempted revenge and harassment.
        - http://fusion.net/story/287592/internet-mapping-glitch-kansas-farm/

Win-Vector LLC

Let's try to remove NAs from real data

- PreparingDataWorkshop/KDD2009/KDD2009vtreat.html
- PreparingDataWorkshop/KDD2009/KDD2009naive.html

Second Example: Unexpected or Novel Categorical Levels

Categorical Variables:
Missing Values and Novel Levels

TrainingData

Residence
CA
NV
OR
CA
CA
NA
WA
OR
WA

NewData

Residence
NV
OR
NV
WY
CA
CA
NV
NA
OR

Novel Levels - Model Failure

```
> model = lm("premium~age+sex+residence",
data=TrainingData)

> predPremium = predict(model,
                         newdata=NewData)

Error in model.frame.default(Terms, newdata,
na.action = na.action, xlev = object$xlevels) :
factor residence has new levels WY
```


Win-Vector LLC

On the Way to the Solution: Dummy/Indicator Variables

Residence
CA
NV
OR
CA
CA
NA
WA
OR
WA

Res_NA	Res_CA	Res_NV	Res_WA	Res_OR
0	1	0	0	0
0	0	1	0	0
0	0	0	0	1
0	1	0	0	0
0	1	0	0	0
1	0	0	0	0
0	0	0	1	0
0	0	0	0	1
0	0	0	1	0

Three Possible Solutions

Training Data Proportions

NA	CA	NV	WA	OR
1/9	1/3	1/9	2/9	2/9

1) A novel level is weighted proportional to known levels

Residence
WY

→

Res_NA	Res_CA	Res_NV	Res_WA	Res_OR
1/9	1/3	1/9	2/9	2/9

2) A novel level is treated as "no level"

Residence
WY

→

Res_NA	Res_CA	Res_NV	Res_WA	Res_OR
0	0	0	0	0

3) A novel level is treated as uncertainty among rare levels

Residence
WY

→

Res_NA	Res_CA	Res_NV	Res_WA	Res_OR
1/2	0	1/2	0	0

`vtreat` solution

Residence	# of occurrences
CA	2000
NV	1100
OR	1000
WA	1500
WY	18
ID	14
CO	8

Residence	# of occurrences
CA	2000
NV	1100
OR	1000
WA	1500
RARE	40

- Levels that appear fewer than N times (N user specified) : pooled to **`rare`**
- Levels (including `rare`) that don't achieve statistical significance code to **`zap`**
  - `zap` codes to "no level" (no model effect)
  - novel levels code to `rare` (if available), otherwise to `zap`

Converse problem: missing levels

- Suppose there levels that occur only in training and not in application (due to training set being larger or having been collected over a longer time scale).

- How to do we know built in dummy/indicator encoders guarantee identical code books?
  - Formula interface through models likely gets this right.
  - Separate direct calls to `model.matrix` likely get this wrong.

Missing levels

```
> model.matrix(~state,data.frame(state=c('CA','NV','CA','WY','AL')))
  (Intercept) stateCA stateNV stateWY
1           1       1       0       0
2           1       0       1       0
3           1       1       0       0
4           1       0       0       1
5           1       0       0       0
attr(,"assign")
[1] 0 1 1 1
attr(,"contrasts")
attr(,"contrasts")$state
[1] "contr.treatment"

> model.matrix(~state,data.frame(state=c('CA','NV','CA')))
  (Intercept) stateNV
1           1       0
2           1       1
3           1       0
attr(,"assign")
[1] 0 1
attr(,"contrasts")
attr(,"contrasts")$state
[1] "contr.treatment"
```

- Consider the first `model.matrix()` the training step, and the second the application.
- Notice ‘AL’ codes to the reference level (not coded) in training and then ‘CA’ codes to the reference level in application.
- Common state systematically mis-coded.

Third Example: Categorical Variables with Very Many Levels

Categorical variables: Too many levels

ZIP	SalePriceK
94127	725
94564	402
90011	386
94704	790
94127	1195
94109	903
94124	625
94124	439
94564	290

- Too many levels is a computational problem for some machine learning algorithms.

- You will inevitably have a novel level

Win-Vector LLC

The Best (but not always possible) Solution

Use as join key into domain knowledge.

San Francisco County ZIP codes	Avg. listing price Week ending Aug 13	Median sales price Date range: May-Aug '14
Name	Amount	Amount
94124	$571,667	$625,000
94134	$619,495	$640,000
94132	$713,583	$835,000
94102	$768,558	$605,000
94112	$771,234	$728,250
94111	$877,000	$959,000
94116	$904,071	$1,025,000
94107	$1,019,113	$908,500
94117	$1,057,000	$1,125,000
94131	$1,057,160	$1,200,000
94110	$1,128,511	$1,082,000
94122	$1,227,482	$930,000
94114	$1,405,793	$1,452,000
94103	$1,406,597	$850,000
94109	$1,408,431	$903,500
94105	$1,549,047	$1,107,500
94127	$1,569,846	$1,300,000

Pragmatic Solution: “Impact/Effects Coding”

ZIP	avgPriceK	ZIP_impact
90011	386	-253.4
94109	903	263.6
94124	532	-107.4
94127	960	320.6
94564	346	-293.4
94704	790	150.6
globalAvg	639.4	0

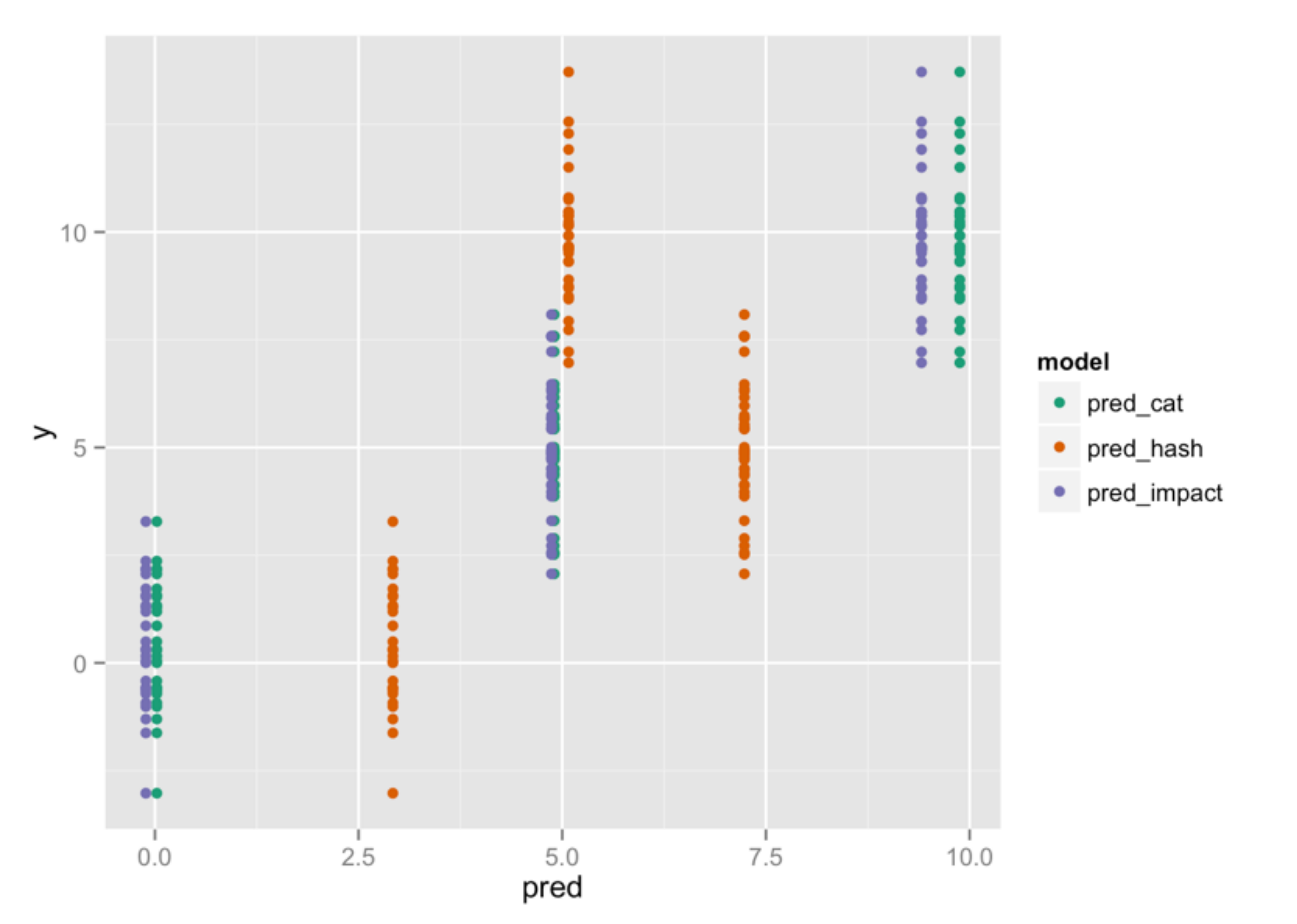
Impact-coding the ZIP variable

ZIP
94127
94564
90011
94704
94127
94109
94124
94124
93401

ZIP_impact
320.6
-293.4
-253.4
150.6
320.6
263.6
-107.4
-107.4
0

Sidebar: Impact-Code; DON'T Hash!

- `Python/scikit-learn`: only takes numerical variables
- Hashing loses information!
- Impact-code, or convert to indicators: `OneHotEncoder()`
- If you must hash, use Random Forest



http://www.win-vector.com/blog/2014/12/a-comment-on-preparing-data-for-classifiers/

Win-Vector LLC

Automating Variable Treatment in R: `vtreat`

Overall: a two-step Process

- Design the data treatment plans
  - Numeric outcome:

```
tPln = designTreatmentsN(train, xv, y)
```

  - Binary class outcome

```
tPln = designTreatmentsC(train, xv, y, target)
```

- Prepare the data sets
  - `train.treat = prepare(tPln, train, pruneSig=0.05)`
  - `test.treat = prepare(tPln, test, pruneSig=0.05)`

vtreat documentation

- http://winvector.github.io/vtreathtml/
  - **vtreat.html**: Overall documentation.
  - **vtreatVariableTypes.html**: The types of derived variables vtreat returns and how to filter to specific variable treatment types.
  - **vtreatSignifcance.html**: How to use estimated significances to prune variables.
  - **vtreatOverfit.html**: Example of why you need to either have separate calibration data or simulate having separate calibration data.
  - **vtreatCrossFrames.html**: How to use cross validation frames to simulate separate calibration data.
  - **vtreatScaleMode.html**: The intended use of scale mode (getting all variables into y-units prior to something like PCA or kmeans clustering).

Designing the Treatment Plans: Numeric Output

salePrice ~ ZIP + homeType + numBed + numBath + sqFt

```
treatPlan = designTreatmentsN(train,
      c("ZIP", "homeType", "numBed", "numBath", "sqFt"),
      "salePrice")
```


Win-Vector LLC

Example Input

ZIP	homeType	numBed	numBath	sqFt	salePrice
94499	condo	4	4	1025	815678
94403	condo	2	3	1082	600635
94361	townhouse	1	3	751	444609
94115	condo	2	3	1093	349433
94217	<NA>	NA	3	914	692468

many-level categorical

categorical

numeric

```
treatPlan = designTreatmentsN(train,
   c("ZIP", "homeType", "numBed", "numBath", "sqFt"),
   "salePrice")
```

Using the treatment plan to prepare data

```
df.treat = prepare(treatPlan, df, pruneSig=0.2)
```

`df` is any frame of appropriate format (training or test)

ZIP_catN	homeType_lev_NA	homeType_lev_x.condo	homeType_lev_x.loft
190033.174	0	1	0
-5320.826	0	1	0
35596.174	0	0	0
-119202.826	0	1	0
-94775.326	1	0	0

homeType_lev_x.single.family	homeType_lev_x.townhouse	numBed_clean	numBed_isBAD
0	0	4.000000	0
0	0	2.000000	0
0	1	1.000000	0
0	0	2.000000	0
0	0	2.456325	1

numBath_clean	numBath_isBAD	sqFt_clean	salePrice
4.000000	0	1025	815678
3.000000	0	1082	600635
3.000000	0	751	444609
3.000000	0	1093	349433
3.000000	0	914	692468

Win-Vector LLC

Designing the Treatment Plans: Binary Classification

loanApproved ~ ZIP + loanType + income + homePrice + FICO

```
treatPlan = designTreatmentsC(train,
   c("ZIP", "loanType", "income", "homePrice", "FICO"),
   "loanApproved", TRUE)
```


Win-Vector LLC

Open Issues

- Overfit from too many variables
  - Variable Selection
  - yAware PCA for dimension reduction
- False fit: upward biased model evaluations from nested models
  - Calibration sets
  - Data fuzzing (differential privacy techniques), more on this later!

Let's work on y-aware PCA

- PreparingDataWorkshop/YAwarePCA/

Conclusions

- There's no substitute for getting your hands on the data
- Nonetheless, some variable treatments are reusable again and again
- y aware data treatment mitigates loss of signal, but requires some care to avoid introducing over-fit
- We've presented our go-to data treatments, and an R implementation for them: `vtreat`

Further References

- Impact Coding
  - http://www.win-vector.com/blog/2012/07/modeling-trick-impact-coding-of-categorical-variables-with-many-levels/
  - http://www.win-vector.com/blog/2012/08/a-bit-more-on-impact-coding/
- Converting Categorical Variables to Numerical (No Hashing)
  - http://www.win-vector.com/blog/2014/12/a-comment-on-preparing-data-for-classifiers/
- PRESS statistic
  - http://www.win-vector.com/blog/2014/09/estimating-generalization-error-with-the-press-statistic/

Where to get `vtreat`

- vtreat on CRAN
  - https://cran.r-project.org/package=vtreat
- vtreat code on GitHub
  - https://github.com/WinVector/vtreat

Additional Issues: Overfitting and False Fitting

Data Scientists Summit (FY16): Data Preparation Using R:

Basic through Advanced Techniques
Part 2: statistical issues

John Mount & Nina Zumel
Win-Vector, LLC

All materials: https://github.com/WinVector/PreparingDataWorkshop

Outline of Part 2

- Overfit
- Nested model fallacies
- The need for y-aware procedures
- General simulation of out of sample data (or cross-frame procedures)

Statistical issues

- Up until now we have largely been working around real-world *operational* issues
    - Missing values
    - Novel levels
    - Categorical variables with very many levels
- Even more dangerous are statistical issues, both those obvious and those unnoticed

Statistical issues we are worried about

- Loss of signal
  - Losing information about the outcome
- Bias
  - Systematic mis-predictions that are function more of the fitting process than the data.
- Overfitting
  - Models that perform well on training data and then fail in production.

Why we care

- For “wide data” (very many variables) we can *not* safely leave all variable selection to common machine learning software.
  - Huge multiple comparison problem
- We can accidentally introduce one issue ourselves when treating variables.
  - Need to at least fix our own mistakes.

Is Variable Selection a Data Treatment Problem?

- For very wide data sets: yes!
  - Too many variables slow down model fitting
  - Can result in misleading models
- So should at least prune variables with no obvious signal

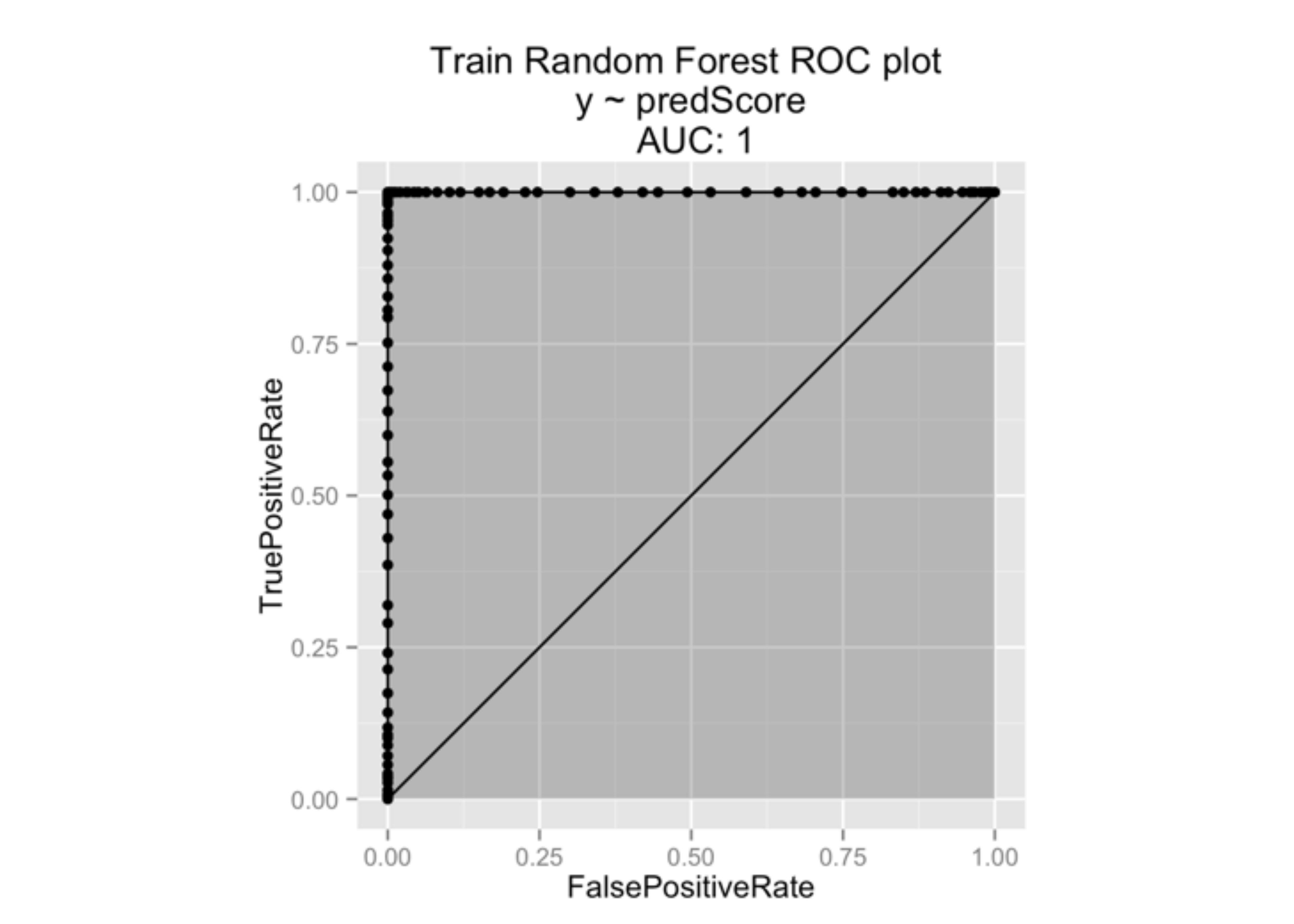
Machine Learning procedures assume curated variables

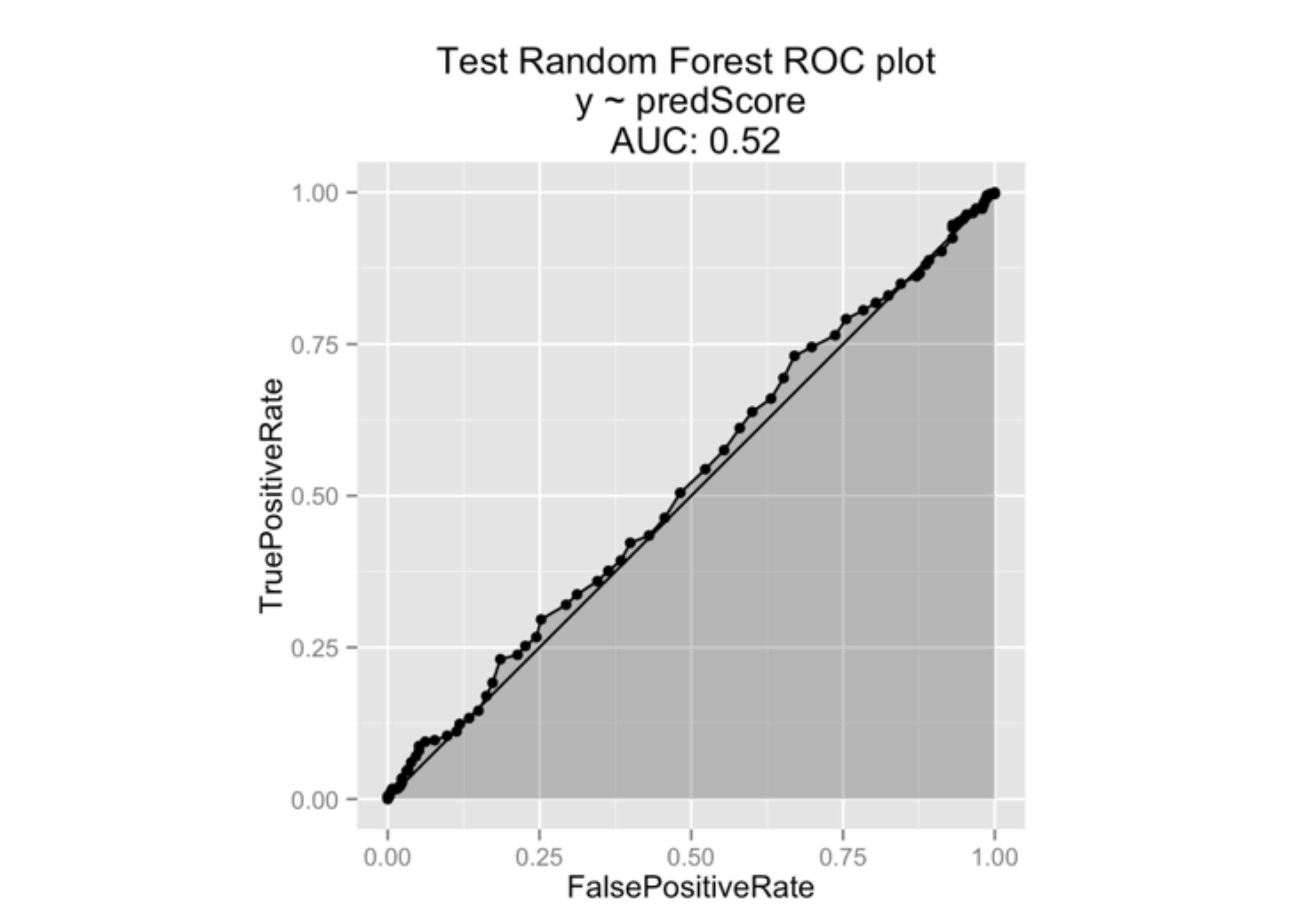
- For *at least* the following common popular machine learning algorithms we can design a simple data set where we get arbitrarily high accuracy on training even when the dependent variable is generated completely independently of all of the independent variables.

- Decision Trees
- Logistic Regression
- Elastic Net Logistic Regression
- Gradient Boosting
- Naive Bayes
- Random Forest
- Support Vector Machine

Can't we keep at least some of our training performance?

- Common situation:
    - Near perfect fit on training data.
    - Model performs like random guessing on new instances.
    - Extreme over fit.
- One often hopes some regularized, ensemble, or transformed version of such a model would have at least some use on new instances.
- Can only keep "some of training performance" for simple models (more on this later).





- PreparingDataWorkshop/BadModels/BadModels.html

Win-Vector LLC

What causes this?

- “Regression to the mean”
  - Some fraction of your “top performers” performance is actually noise.
  - This “luck” isn’t repeated later in production.
  - With enough variables to choose from, your top performers can be completely due to noise (and cut in front of all true variables).
- Also called “Freedman's paradox”
  - Freedman, D. A. (1983) "A note on screening regression equations." The American Statistician, 37, 152–155.

Why doesn't bagging, regularization, or priors fix this?

- Bagging can help eliminate modeling variance, this is bias.
  - We make similar mistakes in each sub-model.
  - Duplicate or near duplicate variables defeat sampling based sub-model diversity (such as Random Forest's variable controls.
- Regularization (or controlling model complexity) can help
  - Most regularization ends up looking like some sort of prior
  - Often defeated when you have a lot of data (Bernstein–von Mises theorem) and when you have a lot of multiple comparison bias (many models/variables to pick from).

Win-Vector LLC

What to do

- Variable selection by significance
  - Pick significance 1/“number of proposed variables”
    - With this significance only a few completely useless variables should leak into the model
    - Downstream model system should be able to deal with these
  - Notice significance pick is different than “always use p=0.05” nonsense

How does significance pruning work?

- Essentially it imitates a really neat permutation test experiment.

Thought Experiment: What does No Signal look like?

Data set: y depends on x

x	y
x1	y1
x2	y2
x3	y3
x4	y4
…	…
xn	yn

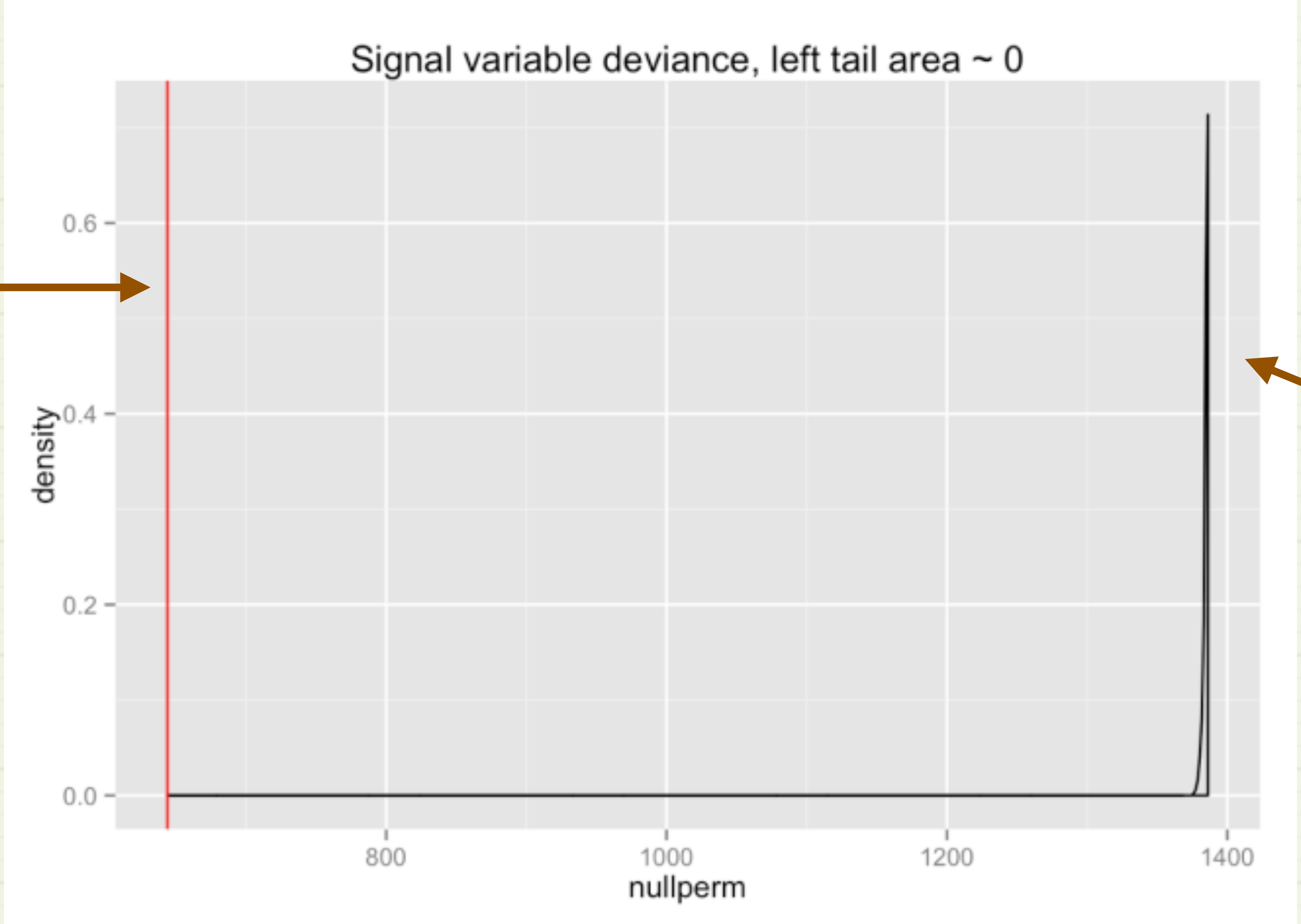
Permute y: now y has no relation to x

x	y
x1	y4
x2	y10
x3	y7
x4	y1
…	…
xn	yk

Do this several times…

Fit models, and compare: When y depends on x

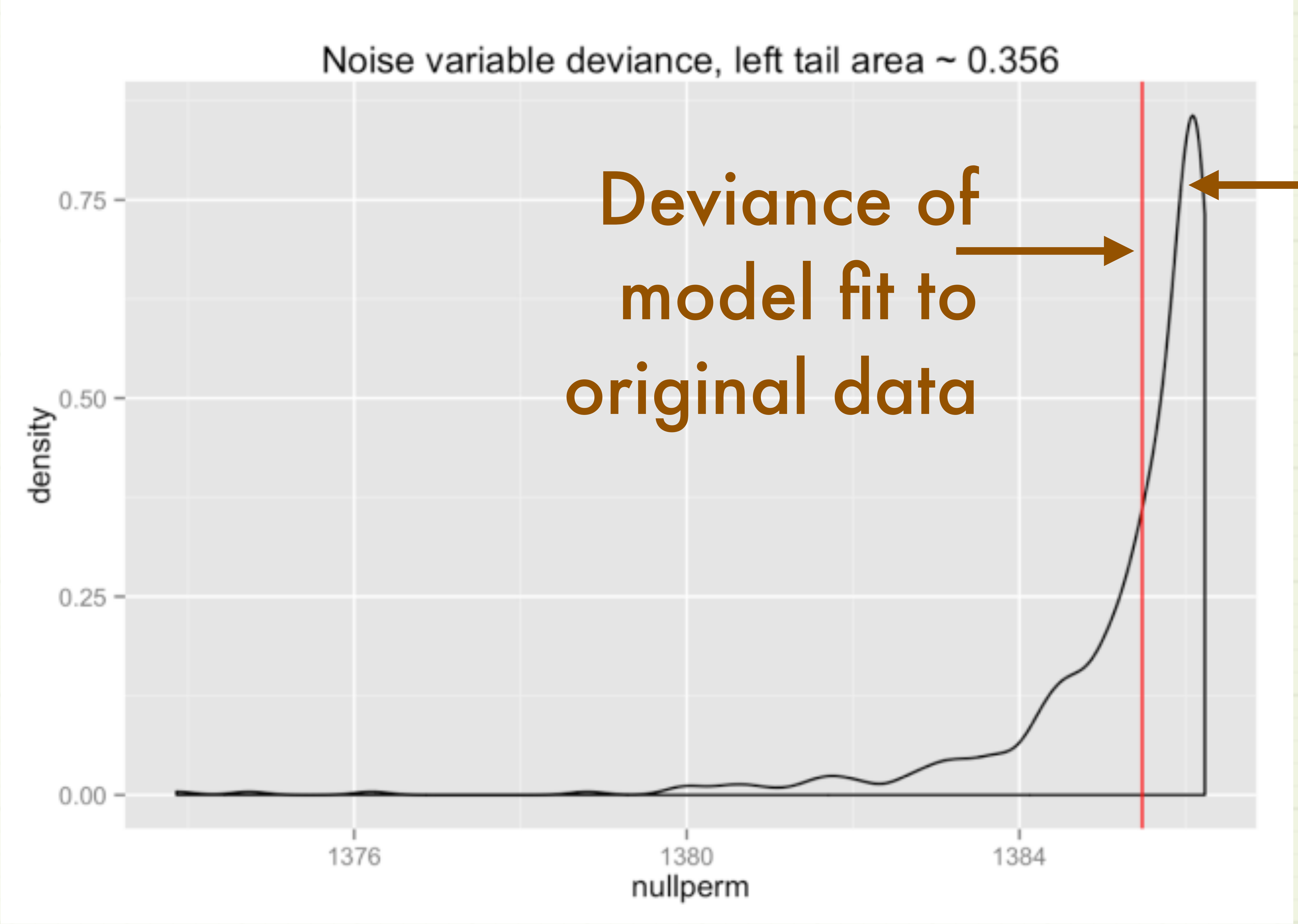
Deviance of model fit to original data



Distribution of deviances of models fit to permuted data

Win-Vector LLC

When y has no relation to x



Distribution of deviances of models fit to permuted data

Left tail area:
The significance of the model fit to original data

Win-Vector LLC

To test a variable v for signal

- Ideally: Build a one-variable model on v, check its significance by a permutation test.
- In practice: use a chi-squared or F-test
    - Model significance of logistic or linear regression, respectively

- PreparingDataWorkshop/TestingForSignal/PermutationSelection.html

- PreparingDataWorkshop/TestingForSignal/PermutationSelection.html

Assumptions

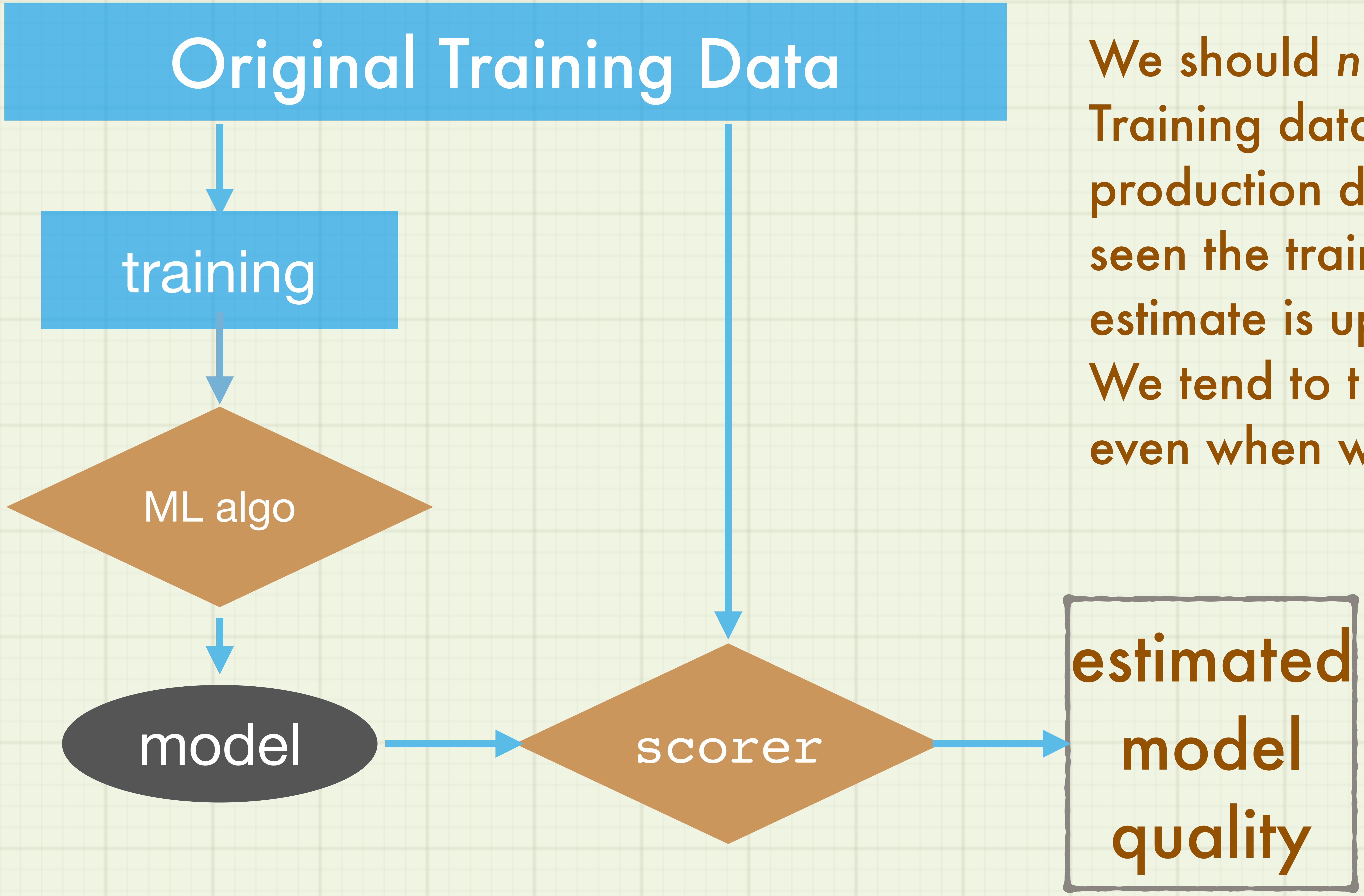
- A variable with signal will manifest it through a linear model or a Bayes model.
- We are not considering interactions among variables when looking for signal
    - That is the left to the machine learning modeling
    - Need to explicitly exclude variables that are needed for use in interactions (such as level-counts) from pruning.

Are we done?

We can accidentally introduce issues when treating variables

- Impact/effect coding is *not* completely safe
  - An impact/effect coded categorical is essentially a model
  - So any model using such variables is a nested model
  - Nested models require some additional care

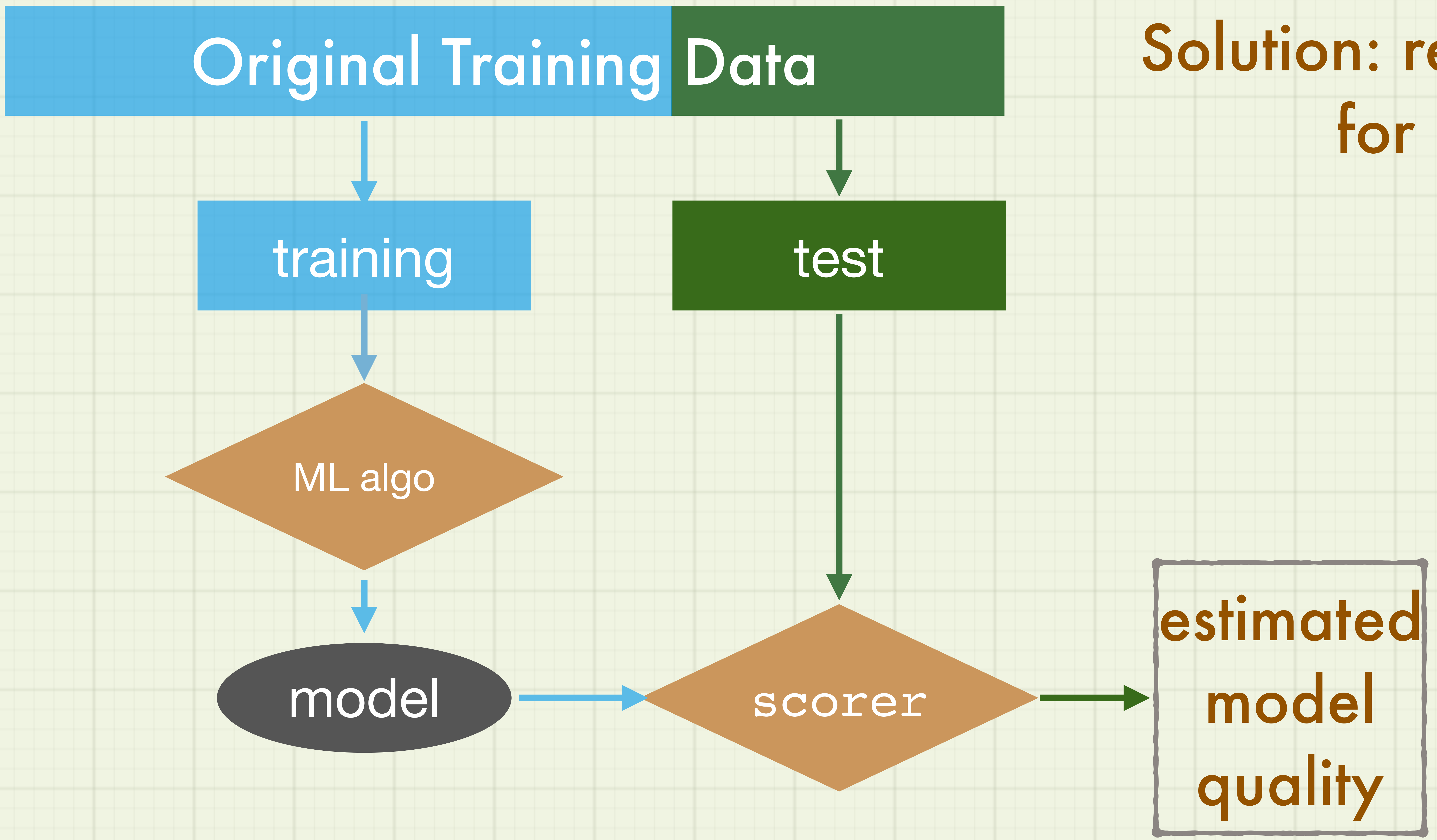
Naive machine learning practice



We should *never* do this.
Training data isn't exchangeable with future production data (due to the trainer having seen the training data), so the model quality estimate is upwardly biased.
We tend to think we have a good model, even when we do not.

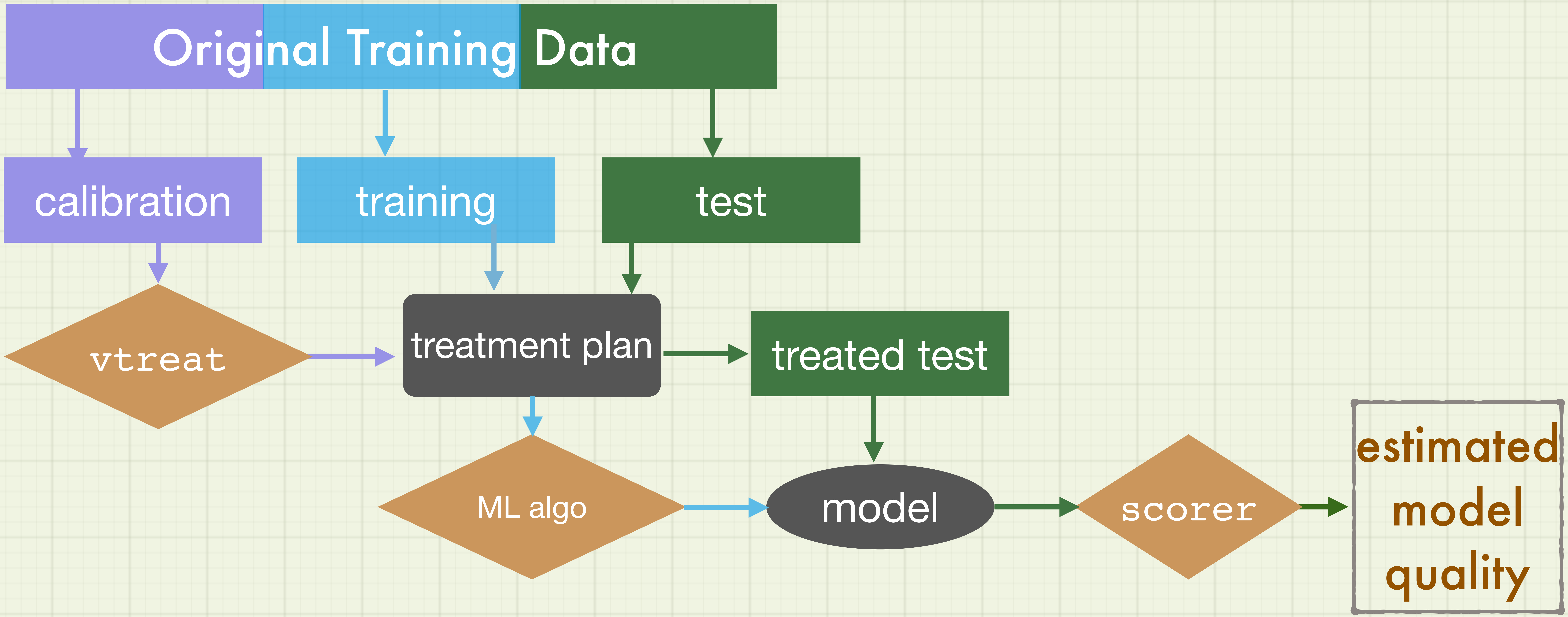
Win-Vector LLC

Standard machine learning practice



Solution: reserve some data for evaluation

Nested model solution: use a calibration set to fit impact models



Win-Vector LLC

Why so much machinery?

- vtreat is essentially building models for the large categorical columns.
- If data that was used in designing the treatment plan is used in training- the training system tends to think these variables are way more reliable than they actually are, and also vastly under estimate the number of degrees of freedom such variables consume.
- This is because the next stage model sees effects codes as having one degree of freedom, when they actually have a number of degrees of freedom equal to the number of distinct levels in the original categorical variable.

Learning: mathematical justification

- Standard modeling situation:
  - learn() is our learning function, mapping data to models
  - model = learn(olddata)
    - model maps data to predictions
  - loss(,) is our criticism of fit, mapping data and predictions to a score
  - Define f(A,B) = loss((learn(A))(B))
    - “the loss of the model trained on A when applied to B”

Generalization theorem

- *If* the range complexity of learn() is not too high (in the sense of VC dimension / PAC learning) then for large exchangeable independent draws of olddata and newdata:
  - f(olddata,olddata) ~distributed as~ f(olddata,newdata)
  - “typically simple models behave in production not much worse than on training”

Win-Vector LLC

Nested model justification

- prelearn(): another learner that returns a function mapping data to data
    - g = prelearn(calibrationdata)
    - calibrationdata: a data set drawn independently of olddata and newdata, but exchangeable with them
    - prelearn() may be complex, and not meet the pre-conditions of the previous slide
- Theorem (under reasonable conditions):
    - For a fixed g=prelearn(calibrationdata) the data sets g(olddata) and g(newdata) behave as exchangeable independent draws (under re-draws of olddata and newdata).
    - So previous theorem applies to conditioned data:
        - olddata’ = g(olddata)
        - newdata’ = g(newdata)

Back from the theory

- Single use calibration sets allow correct fitting of nested models
    - Used by vtreat to build data conditioners
    - Use in super-learning/stacking to build ensemble models
- Can simulate single use calibration sets through cross validation methods.
    - Statistically more efficient
    - Computationally more work
    - Implemented in vtreat as `mkCrossFrameNExperiment` and `mkCrossFrameCExperiment`
    - Special notation treatment in `PreparingDataWorkshop/CrossFrames/CrossOperators.html`

Conclusions

- You must prepare data prior to analysis, even when using sophisticated modern machine learning methods.
- Even though you may be preparing your data for mere operational reasons (data cleaning), you soon run into statistical issues.
- Think of columns and variables as single variable models.
- There are many good techniques to correctly and efficiently build sub-models.

Further Reading

- vtreat
  - https://cran.r-project.org/package=vtreat
- Model testing procedures
  - http://www.win-vector.com/blog/2015/09/isyourmodelgoingtowork/
- Permutation tests
  - http://www.win-vector.com/blog/2015/08/how-do-you-know-if-your-data-has-signal/
- Differential privacy
  - http://www.win-vector.com/blog/2015/11/our-differential-privacy-mini-series/

Thank You

All materials: https://github.com/WinVector/PreparingDataWorkshop

Win-Vector LLC