Lessons from Machine Learning

Last time...

Exam 2

Today

- Lessons from **machine learning**

Explanation vs. Prediction

Explanation: describe causal underpinnings of behaviors/outcomes

Prediction: Accurately forecast behaviors/outcomes

Similarities

- Both are goals of science
- Good predictions can help us develop theories of explanation and vice versa

Explanation vs. Prediction

Statistical Tensions

- Statistical models that accurately describe causal truths often have poor prediction and are complex
- Predictive models are often very different from the "true", underlying data generating processes

Explanation vs. Prediction

What do we do typically do in Psychology?

How far have we come in
the past 10 years? 20 years?
30 years?

How do you *feel* about it?

Yarkoni & Westfall (2017) argue that we should spend more time creating predictive models

At the very least, maybe there are some lessons we can take away from the ML world that can potentially help us

Maybe Helpful Definitions

Machine Learning (ML) is the process of feeding previous observations into a computer and using the computer to generate predictions for new observations. AKA:

- machine inference
- pattern recognition (Google Photos!)
- statistical estimation
- prediction modeling
- statistical learning (ish)

Maybe Helpful Definitions

Training occurs by extracting patterns from the observed data; think of this as *learning*

Testing occurs by verifying predictions on previously unobserved data; think of this as *evaluating*

> How might this work in regression land?

A note

Artificial Intelligence is not the same as machine learning. AI is the simulation of human intelligence by computers; AI systems are generally trained with machine learning approaches.

Last class of the semester, we will hear more of this from Josh Oltmanns.

Some more definitions

Supervised Learning: using known patterns between input and output observations to train a mapping between the two

- **Regression!** learning the mapping between a continuous input feature variable and a continuous output target variable
- **Classification:** learning the mapping between a continuous input feature variable and a categorical output target variable (i.e., a label)

Some more definitions

Unsupervised Learning: determining patterns in observations without guiding referents

- **Dimensionality Reduction:** decreasing the overall number of features considered in a learning procedure (i.e., PCA, ICA etc.)
- **Clustering:** grouping features together that are similar as determined by *some* metric

@Wash U

- **Reinforcement Learning:** determining a mapping between input and output observations using only a measure of training quality
- ML classes in CS or SDS department
- Wouter Kool for reinforcement learning
- ACCSN with Dennis Barbour and myself for high level discussions on these topics
- Poli Sci department for dealing with categorical outcomes

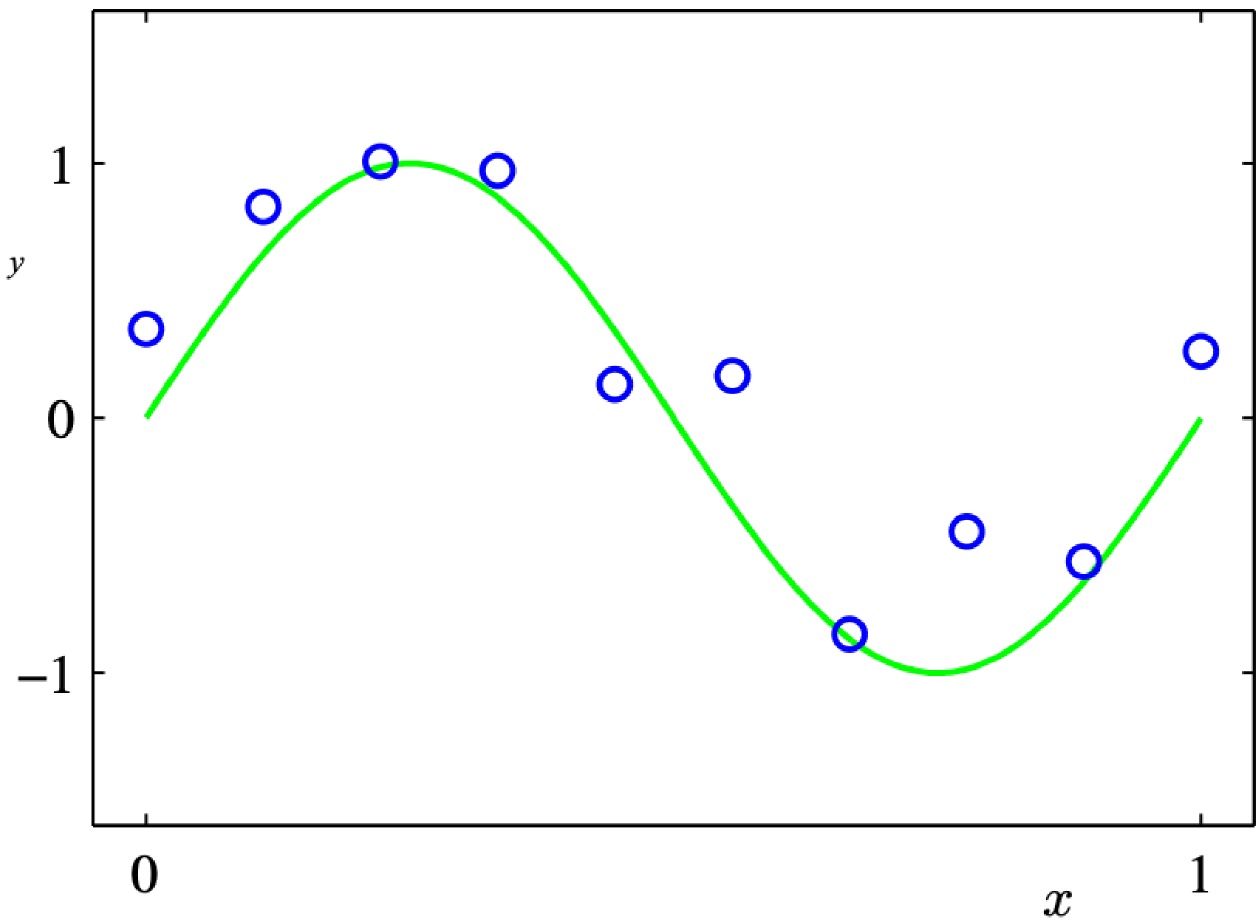
Terminology that matters

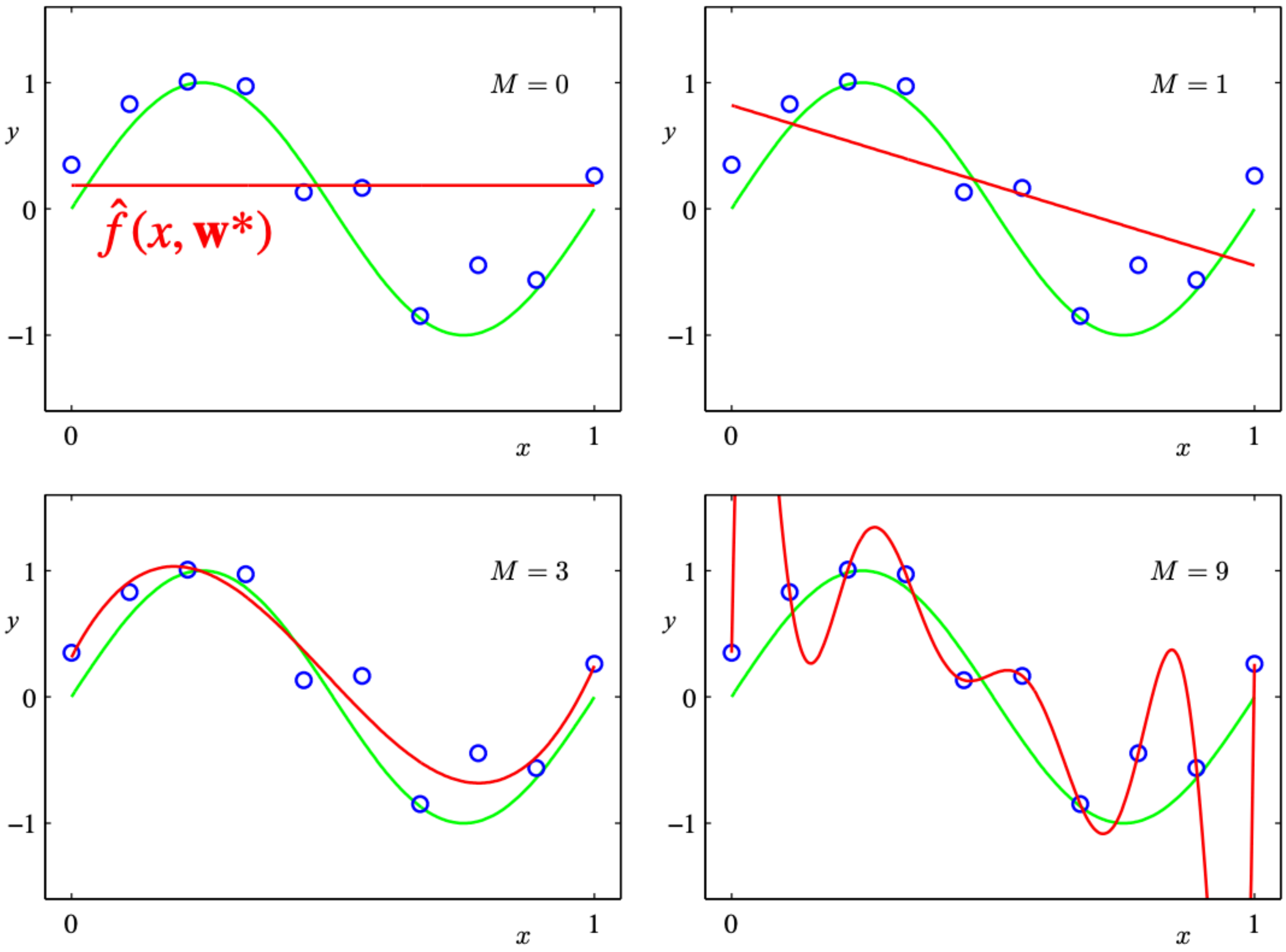
Overfitting is when we *mistakenly* fit sample-specific noise as if it were actually a signal.

- If our model has a $R^2 = .9$, we do an excellent job of explaining variance *in our sample*.
- OLS models tend to be overfit because they minimize error for a specific sample

Following slides courtesy of Dennis Barbour

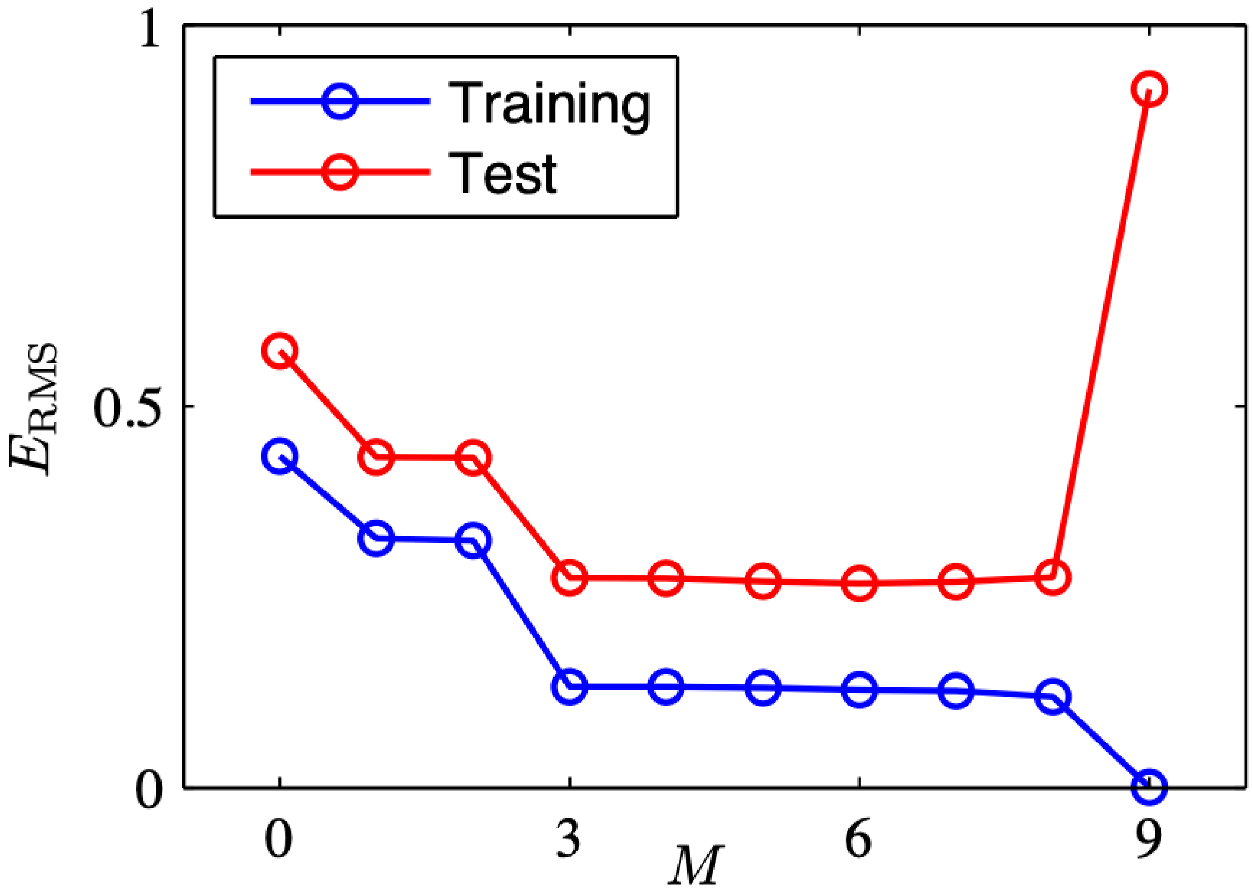
Let's say we have these data





M = parameters, red = line we are fitting, green = "truth"**Why *shouldn't* we use all 9 parameter?**

Why not use all 9 parameters?



Overfitting is a big problem!

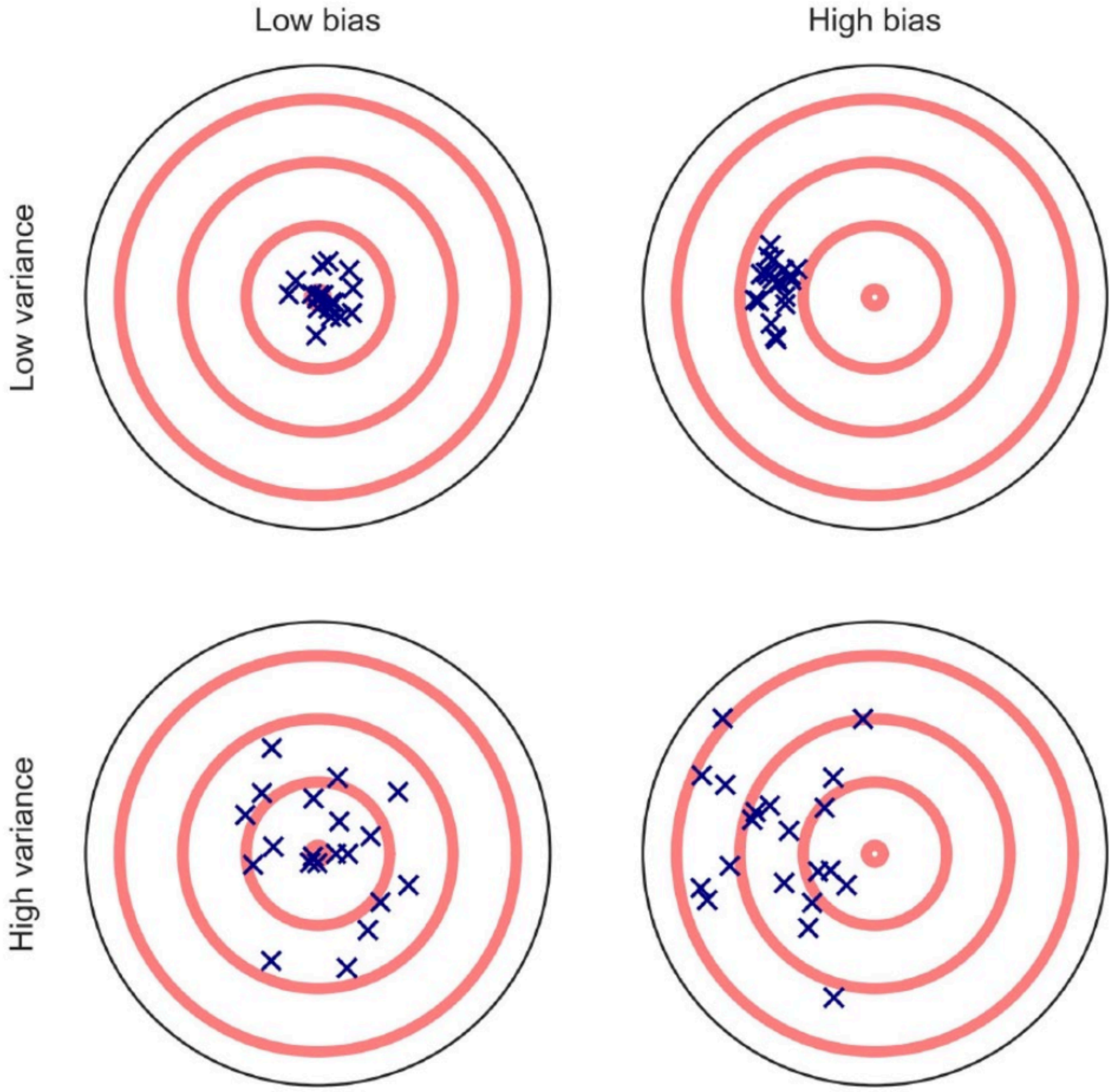
Underfitting means we can't capture a relationship at all -- not as big of a problem for us

Bias and Variance

Bias refers to systematically over- or under-estimating parameters.

Variance refers to how much estimates tend to jump around

Bias-Variance Tradeoff we can reduce variance around our estimates but at the expense of increasing bias of estimates and vice versa


Low bias
High bias
Low variance
High variance

You should be afraid...very afraid

How do we take what we know from overfitting & the bias-variance tradeoff and incorporate it? What can we learn from the ML world?

Big Data

- Reduce the likelihood of overfitting -- more data means less error

Cross-validation

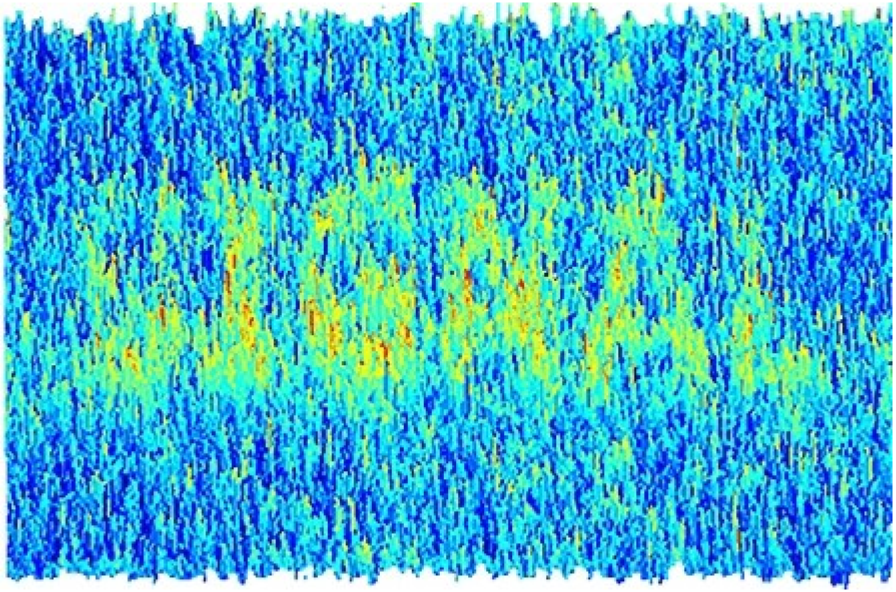
- Is my model overfit?

Regularization

- Constrain the model to be less overfit

"Every pattern that could be observed in a given dataset reflects some...unknown combination of signal and error"

Error is random, so it cannot correlate with anything; as we aggregate many pieces of information together, we reduce error.



Cross-validation

Cross-validation is a family of techniques that involve testing and training a model on different samples of data.

Cross-validation: Hold-out Samples

- Split into training and testing sets
- Fit your model on the *training* set
- Predict outputs for your *testing* set

Pros

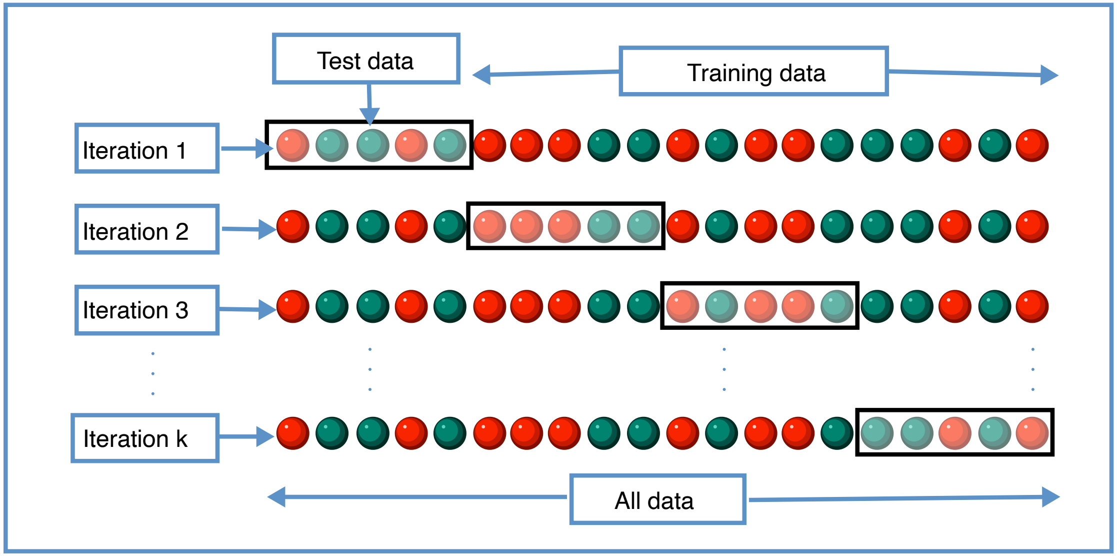
- Straightforward; computationally easy

Cons

- Which data go into which set?
- What if the vast majority of group A fall into the training set and the vast majority of group B fall into the testing set?

K-fold Cross-validation

- Make *k* subsets of your data
- Repeat the hold-out method of test/train, but do it *k* times
- Get the model fit for all *k* iterations; take the average model fit



K-fold Cross-validation

Pros

- Doesn't matter much which data points fall into test or train since each subset can be both a test and a training set
- The more folds you do (larger k), the more you are able to decrease your variance around your averaged model fit

Cons

- Can take a decent amount of computational power, depending on the dataset

Leave-One-Out Cross-validation

- Same as *k*-fold, but now *k* is equal to your N

Pros

- Good estimations

Cons

- Even more computationally expensive
- Especially if using "big data"

```r
library(here)
stress.data = read.csv(here("data/stress.csv"))
library(psych)
describe(stress.data, fast = T)
```

```
##          vars   n   mean     sd median  min    max  range  skew kurtosi
## id          1 118 488.65 295.95 462.50 2.00 986.00 984.00  0.10    -1.2
## Anxiety     2 118   7.61   2.49   7.75 0.70  14.64  13.94 -0.18     0.2
## Stress      3 118   5.18   1.88   5.27 0.62  10.32   9.71  0.08     0.2
## Support     4 118   8.73   3.28   8.52 0.02  17.34  17.32  0.18     0.1
## group       5 118    NaN     NA     NA  Inf   -Inf   -Inf    NA       N
```

```r
model.lm = lm(Stress ~ Anxiety*Support*group,
              data = stress.data)
summary(model.lm)$r.squared
```

```
## [1] 0.4126943
```

A brief aside:

Newer package called `tidymodels` is better for machine learning, but requires many more steps. For now, this is the simpler method.

`R` vs. `Python` 🐍. `R` is excellent for statistics and visualizing data. Most of what we do in Psychology. `Python` is better for machine learning and is more of a full suite language.

If you're going hard with ML, use `tidymodels` or Python. See the grad student resources on our website for tools to help you learn both.

Example: 10-fold cross validation

```
library(caret)
# set control parameters
ctrl <- trainControl(method="cv", number=10)
# use train() instead of lm()
cv.model <- train(Stress ~ Anxiety*Support*group,
                  data = stress.data,
                  trControl=ctrl, # what are the control paramet
                  method="lm") # what kind of model
cv.model
```

```
## Linear Regression
##
## 118 samples
##   3 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 107, 106, 106, 106, 106, 106, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1.529142  0.3615326  1.22586
```

Cross-Validation Summary

- Tempers your estimates to protect against overfitting
- You do not need fancy ML algorithms. You can do this with your classic regression!
- If your whole sample is not representative, this will not save you

  > Garbage in, garbage out

Regularization

Penalizing a model as it grows more complex.

- Usually involves ***shrinking*** coefficient estimates -- the model will fit less well in-sample but may be more predictive

LASSO regression: balance minimizing sum of squared residuals (OLS) and minimizing smallest sum of absolute values of coefficients

- Penalize the size of the coefficients
- The larger the penalty applied, the more estimates are go towards zero (or "shrink" -- the first "S")
- Coefficients are more biased (tend to underestimate coefficients) but produce less variability in results
- Also helps with variable selection

The coefficient λ is used to penalize the model.

Least Absolute Shrinkage and Selection Operator

LASSO Regression

The `glmnet` package has the tools for LASSO regression. One small complication is that the package uses matrix algebra, so you need to feed it a matrix of predictors -- specifically, instead of saying "find the interaction between A and B", you need to create the variable that represents this term.

Luckily, the function `model.matrix()` can do this for you.

```r
# provide your original lm model to get matrix of predictors
X.matrix <- model.matrix.lm(model.lm)
head(X.matrix)
```

```
##   (Intercept)  Anxiety Support groupTx Anxiety:Support Anxiety:groupTx
## 1           1 10.18520  6.1602       1        62.74287         10.1852
## 2           1  5.58873  8.9069       0        49.77826          0.0000
## 3           1  6.58500 10.5433       1        69.42763          6.5850
## 4           1  8.95430 11.4605       1       102.62076          8.9543
## 5           1  7.59910  5.5516       0        42.18716          0.0000
## 6           1  8.15600  7.5117       1        61.26543          8.1560
##   Support:groupTx Anxiety:Support:groupTx
## 1          6.1602                62.74287
## 2          0.0000                 0.00000
## 3         10.5433                69.42763
## 4         11.4605               102.62076
## 5          0.0000                 0.00000
## 6          7.5117                61.26543
```

```r
library(glmnet)
lasso.mod <- glmnet(x = X.matrix[,-1], #don't need the interc
                    y = stress.data$Stress)
```

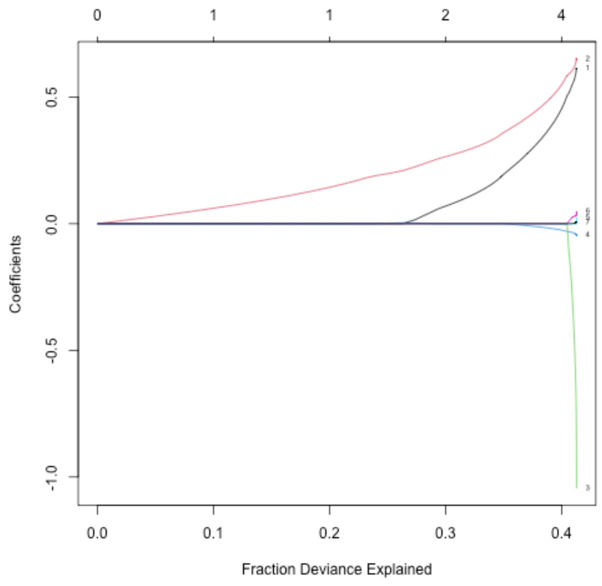
```r
lasso.mod
```

```
##
## Call:  glmnet(x = X.matrix[, -1], y = stress.data$Stress)
##
##    Df  %Dev  Lambda
## 1   0  0.00 0.97920
## 2   1  4.66 0.89220
## 3   1  8.54 0.81300
## 4   1 11.75 0.74080
## 5   1 14.42 0.67500
## 6   1 16.64 0.61500
## 7   1 18.48 0.56040
## 8   1 20.00 0.51060
## 9   1 21.27 0.46520
## 10  1 22.32 0.42390
## 11  1 23.20 0.38620
## 12  2 24.11 0.35190
## 13  2 24.98 0.32070
## 14  2 25.70 0.29220
## 15  2 26.29 0.26620
## 16  3 27.40 0.24260
## 17  2 29.39 0.22100
## 18  2 30.44 0.20140
## 19  2 31.31 0.18350
## 20  2 32.03 0.16720
```

- DF = number of non-zero coefficients
- dev = R^2
- lambda = complexity parameter
  - how much to down-weight
  - between 0 & 1

What value of λ to choose?

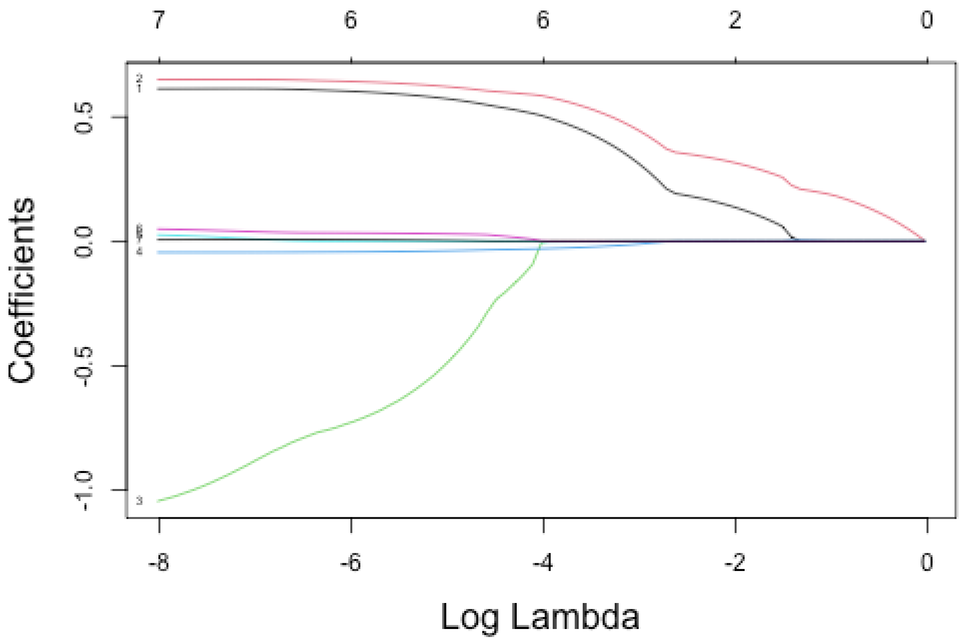
```
plot(lasso.mod, xvar = "dev")
```



Looks like coefficients 1, 2, and 3 have high values even with shrinkage.

What value of λ to choose?

```
plot(lasso.mod, xvar = "lambda", label = TRUE, cex.lab = 1.5)
```



Looking for λ values where those coefficients are still different from 0

```r
coef = coef(lasso.mod, s = exp(-5))
round(x = coef, digits = 2)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                                 s1
## (Intercept)                  -2.21
## Anxiety                       0.57
## Support                       0.62
## groupTx                      -0.49
## Anxiety:Support              -0.04
## Anxiety:groupTx               .   
## Support:groupTx               0.03
## Anxiety:Support:groupTx       0.00
```

```r
coef = coef(lasso.mod, s = exp(-4))
round(x = coef, digits = 2)
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                              s1
## (Intercept)               -1.85
## Anxiety                    0.50
## Support                    0.58
## groupTx                    0.00
## Anxiety:Support           -0.03
## Anxiety:groupTx            0.00
## Support:groupTx            0.00
## Anxiety:Support:groupTx   .
```

```r
coef = coef(lasso.mod, s = 0)
coef
```

```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                                     s1
## (Intercept)               -2.373766131
## Anxiety                    0.612812455
## Support                    0.650290859
## groupTx                   -1.044285345
## Anxiety:Support           -0.045146588
## Anxiety:groupTx            0.025299852
## Support:groupTx            0.048852216
## Anxiety:Support:groupTx    0.005913829
```

$\lambda = 0$ is pretty close to our OLS solution

```r
coef = coef(lasso.mod, s = 1)
coef
```

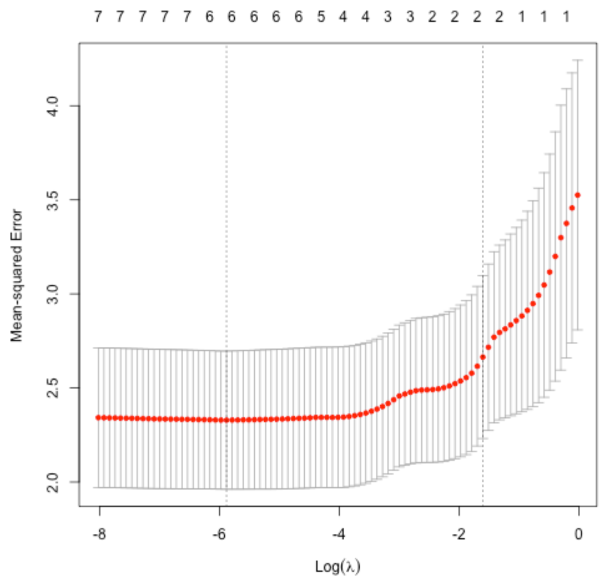
```
## 8 x 1 sparse Matrix of class "dgCMatrix"
##                                  s1
## (Intercept)               5.180003
## Anxiety                   .
## Support                   .
## groupTx                   .
## Anxiety:Support           .
## Anxiety:groupTx           .
## Support:groupTx           .
## Anxiety:Support:groupTx .
```

$\lambda = 1$ is a huge penalty

Choosing a lambda based on eyeballing can be hard. We can use cross-validation instead to help us choose!

```
cvfit <- cv.glmnet(x = X.mat
cvfit
```

```
plot(cvfit)
```




```
##
## Call:  cv.glmnet(x = X.matrix[, -1], y = stress.data$Stress, type.meas
##
## Measure: Mean-squared Error
##
##       Lambda Index Measure     SE Nonzero
## min 0.00279    64   2.329 0.3674       6
## 1se 0.20138    18   2.664 0.4330       2
```

Once you've imposed a shrinkage penalty on your coefficients, you've wandered far from the realm of NHST.

In general, you'll find that very few ML techniques are compatible with probability theory (including Bayesian), because they're focused on different goals.

Instead of asking, "how does random chance factor into my result?", machine learning optimizes (out of sample) prediction. Both methods explicitly deal with random variability.

Rather than estimating degree of randomness; in machine learning, we're trying to remove it.

Summary: Yarkoni and Westfall (2017)

Big Data

- Reduce the likelihood of overfitting -- more data means less error

Cross-validation

- Is my model overfit?

Regularization

- Constrain the model to be less overfit

Enjoy the rest of the semester!!

- Ran talking about Resampling Methods
- Josh Jackson talking about Bayes & MLM. *Taking over as instructor!*
- Amy Eyeler talking about Qualitative Methods
- Josh Oltmanns talking about AI
- *Hint for Exam 3: you do NOT get to forget p-values or confidence intervals.*

Machine Learning Algorithms

- ordinary least squares linear regression
- logistic regression
- k-means clustering
- nearest neighbor
- naive Bayes
- ridge regression
- LASSO regression
- support vector machine
- random forest
- Gaussian process estimator
- multilayer perceptron (deep net)
- convolutional network
- recurrent network
- generalized adversarial network