# More on Model Fitting and Some Final Remarks

Statistical Machine Learning: 36-462/36-662

April 24, 2025

Final exam

- Friday May 2, 8:30-11:30am, Scaife Hall 105.
- See Canvas for the list of topics/questions, and the syllabus for details on coverage and rules (*note*: again, the formula sheet may be two-sided)

Additional Office Hour

- Tuesday, April 29, 4:15-6:00pm, BH 132M1.

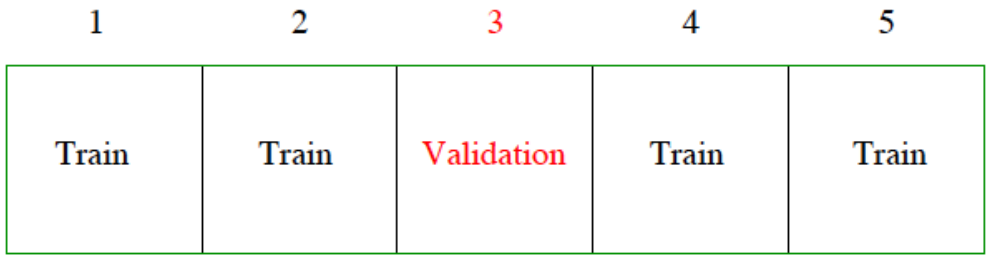
Today:

- More on the data analysis process
- Project prediction performance
- A bonus topic, if we have time
- I will post a review of the second half of the semester in slide form, with the details filled out, also under the last class!

Remember: Please respond to online course evaluation, at `https://cmu.smartevals.com`

# Cross-validation

One approach to avoid splitting too much is cross-validation. This is an efficient way to select our model without wasting too much data.




However, there is still some concern of over-fitting on the cross-validation set. This is especially an issue if we experiment with many models.

# Validation plan

A realistic approach:

1. Split the data into a fitting set and a testing set.
2. You can further split the fitting set into a train set and a validation set (this is easy to understand conceptually but rarely used in practice). More common: use cross-validation or out-of-bag errors only on the fitting set to pick your model.
3. Fit your chosen model on the whole fitting set.
4. At the very end, use your testing set to get an accurate picture of how well you actually do with that model.

You should never touch the testing set during your model experimentation and selection.

*Why do we need the test set?*

# Some Warnings about Cross-validation

Cross-validation seems like a magical solution to an impossible problem. There are a few things to beware of:

1. Can have a high computational cost, especially as the number of parameters/models grows

2. When the number of models is very large, it can give misleading results. It is not immune to the usual winner's curse problem

3. The folds need to include the entire estimation pipeline. A frequent mistake is to carry out part of the model selection before breaking the data into folds, which invalidates the result

# Warning: High Computational Cost

Suppose it costs $c$ to fit an estimator once.

To fit it across a grid of 100 $\lambda$ values, it would cost $100c$.

If we do 10-fold cross-validation across this grid, it would cost $1000c$.

If we do LOOCV, it would cost $100nc$, which could potentially be tremendously large

If we decide to tune two parameters with 10-fold cross-validation (each with a 100-value grid), it would cost $10^6c$...

We want to be somewhat judicious in what we choose to tune with CV (see, especially, the next slide). There is also a literature in statistics and computer science which works on tricks to avoid these high costs.

# Example: High Number of Models

If we search a very large number of models, one of them may look better completely by chance.

Imagine we are in a binary classification problem ($y_i \in \{0, 1\}$). If each model were terrible and just flipped coins to make the test set prediction, eventually we would try one that looked quite good. In particular, we would eventually find one that performed better in cross-validation error than the true best model. This is the winner's curse.

This is just something to be aware of when the set of possible models is expanded dramatically. It is less of a risk in more constrained settings.

# Example: Excluding Part of the Pipeline

This mistake happens a lot, and looks slightly different each time.

Remember: For cross validation (or test sets) to work, every data-driven decision about the model has to exclude the held-out fold (and, of course, the test set).

Bad CV example: Suppose that we observe $y$ and $x_1, \ldots, x_p \in \mathbb{R}^n$, with $p \gg n$. We would like to test our magical prediction algorithm $\widehat{f}_\lambda$. Because $p$ is so large, we first filter for $x_i$ which have some reasonable correlation with $y$, building $\widetilde{X}$ with columns $\{x_i : |\mathrm{cor}(x_i, y)| > \delta\}$.

Now we estimate $\widehat{f}_\lambda$ on $\widetilde{X}, y$ and use cross validation to tune $\lambda$ and estimate the error.

What goes wrong?

# Feature Generation

Again, several over-generalizations:

- The general lesson most data-scientists learn through many applications/contests/etc. is that, for prediction tasks, the data and features are often more important than the exact algorithm that you apply.
- Tweaking the algorithm gives small gains. Developing new features to feed into your algorithms gives big gains.
- This is where the power of forests, boosting, and deep learning come in. They have an ability to use features more flexibly.

# Feature Generation

It is difficult to give specific advice about feature generation, since it is so problem-dependent.

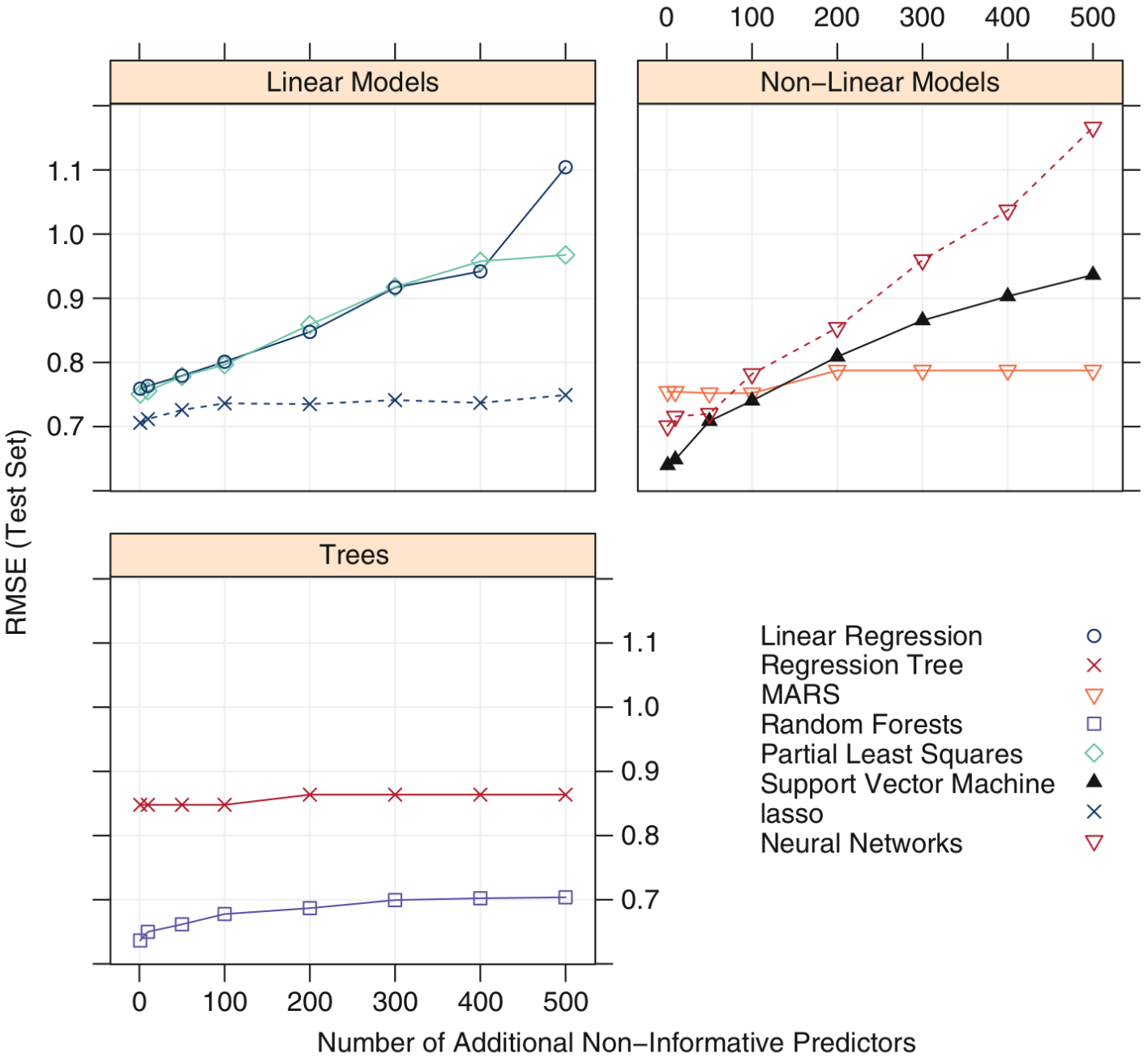
A few comments:

- With modern algorithms (e.g., Lasso, random forests), having extra features does not hurt much. You can afford to generate many useless features in pursuit of a few really good ones.

  This means you can consider multiple transformations or versions of the same information.

- Make sure to only use information you would have in new samples!

# Feature Generation



# Time as a Feature

You'll often have time stamps or dates. First, these are often annoyingly formatted, so you need to make them load correctly/pre-process.

You also often want to featurize these in a way that your algorithm can make use of all the information they contain. Some example additional features to extract:

- Work hours
- Work day
- Month, quarter, season
- Holiday

# You're ready to *learn*!

You've seen many models, and hopefully obtained an intuition for how they behave. You can now start to really think about which models are good for your situation.

- Linear/logistic regression
- Ridge linear/logistic regression
- Lasso linear/logistic regression
- Trees
- Random forests
- Boosted trees
- SVM
- LDA/QDA
- Naïve Bayes

You should consider: which of these are flexible, non-linear, interpretable, good at dealing with irrelevant features, and so on...

# First Models

The quick and dirty approach:

- Something simple and interpretable. Often Lasso (regression) or Logistic Lasso (classification).
- Something flexible and more powerful. Often random forests.

Differences between the two models can hint at what you might be ignoring with the simple model. Explore the difference and try to see why!

Variable importance and partial dependence plots from the random forest can suggest important variables, new transformations, and important interactions.

# Now what?

You have your data. You have some features. You've fit a couple models and have estimates of their error.

Now what? How do you improve your error?

(Always keeping in your mind: what error should you actually care about?)

# Now what?

One approach: Think about the Bias-Variance Tradeoff! Remember, all the error you can correct is either in your bias or your variance.

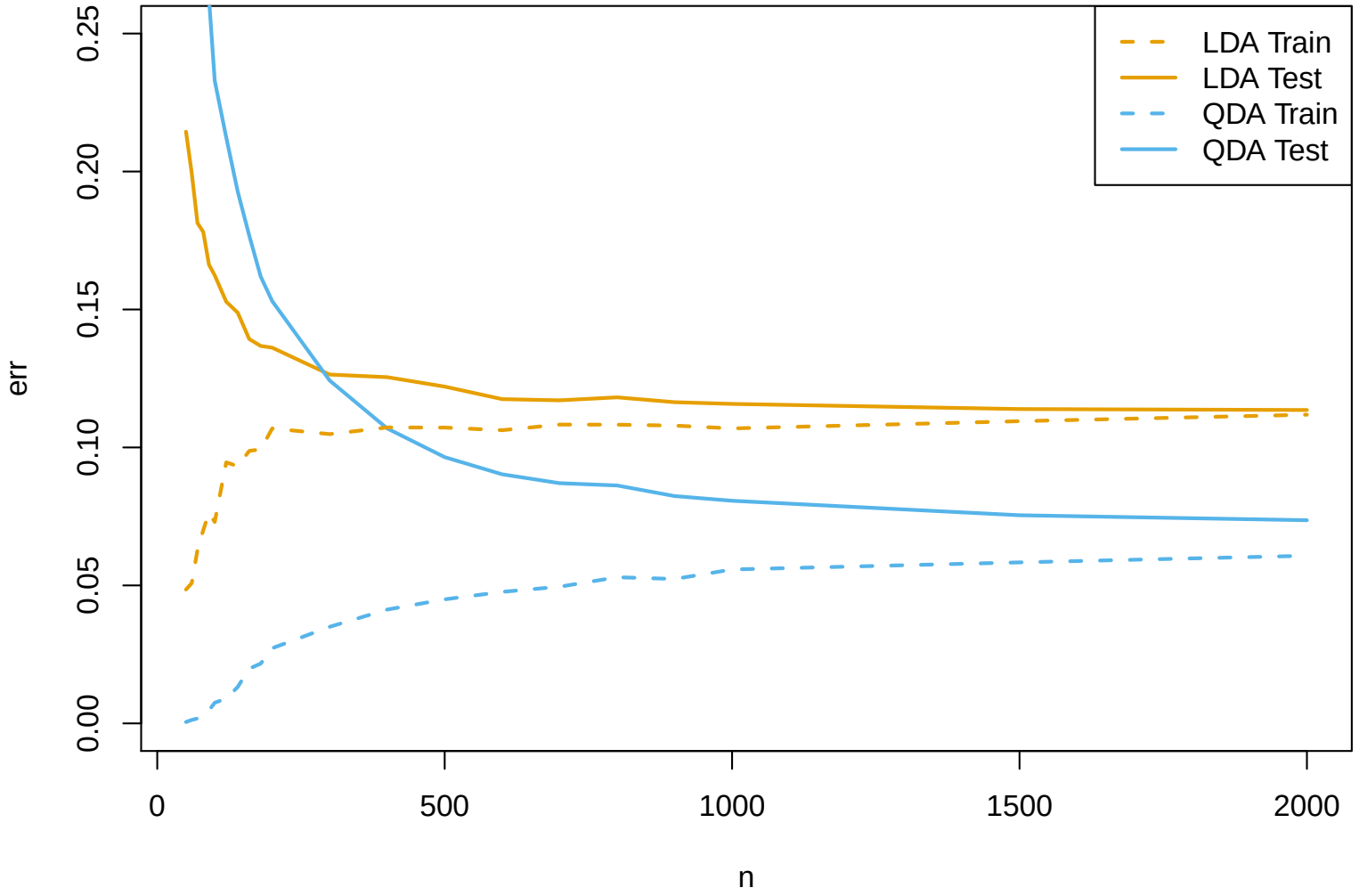
Is your current model too biased or too variable?

One way to get a sense: look at the difference between training and test error.

What does it mean if:

- The training error is much smaller than the test (CV) error?
- The training error is about the same as the test (CV) error?

# Bias-Variance




LDA with fixed $p$ and varying $n$. True model fits QDA assumptions, but it *can be* too high variance.

# Now what?

What can you do to change your model? What do each of the following do to the bias and variance?

- Get more data:
- Make more/better features:
- Use a more flexible model:
- Use a more regularized model:

# Tweaking Features

You can look at the variables that are currently important to give some hints of better variables.

If a variable is important, how can you make it more useful? What other variables might it suggest are useful?

If it is not important but you thought it should be, what is going wrong?

Variable importance plots can hint at useful variables when included with transformations or interactions. Partial dependence plots can reveal better transformations and even interactions.

# What errors do you care about?

In regression problems: is least squares error reasonable for your problem?

In classification: is misclassification rate appropriate? Do you care about false positives and false negatives equally? How should you trade them off?

# Correcting for Badly Imbalanced Samples

When one class is much smaller than the other, most of our classifiers don't do very well. Suppose that you have 90% class 1, and 10% class 2; there are a few approaches that address this issue:

- Downsampling: sample one (or a few) elements of class 1 for each element of class 2 to make it more balanced
- Upsampling: duplicate elements of class 2 to make it more balanced.
- Artificially change your prior weights, class weights, case weights, etc. Appropriate tools depend on which method you are using.

Always keep an eye on your base rate: the fraction of each class in your data. That keeps you aware of this problem, and lets you know what good performance really is: 10% misclassification in the problem above is not particularly impressive...

# Summary

1. Train-validation-test data splitting
2. Featurization
3. Quick and dirty (fit a couple of models). Understand base rates (fit naive predictors), understand what error metric you should be using
4. Diagnose bias/variance problems (use sample-size curves, model-complexity curves, regularization curves, compare different models)
5. Fix bias/variance problems (different set of fixes in each case). Iterate 2,4,5. Think carefully about how to not get bogged down in tweaking tuning parameters (use smaller data sets, be parsimonious in choices to try out; especially important, though often infeasible, for NNs)
6. Error analysis (diagnose points on which you are predicting poorly, are they outliers? Can you design useful features for them specifically?)
7. Sometimes you just don't have enough data; collecting more training data is often a data analyst's last resort

# Our Top Predictors

| Team Name | Test Data Accuracy |
|---|---|
| Aidan Bradshaw and James Lauer | 0.7990236 |
| nothing_but_net | 0.7941416 |
| TEAMLJ | 0.7903057 |
| IGuessed | 0.7882134 |

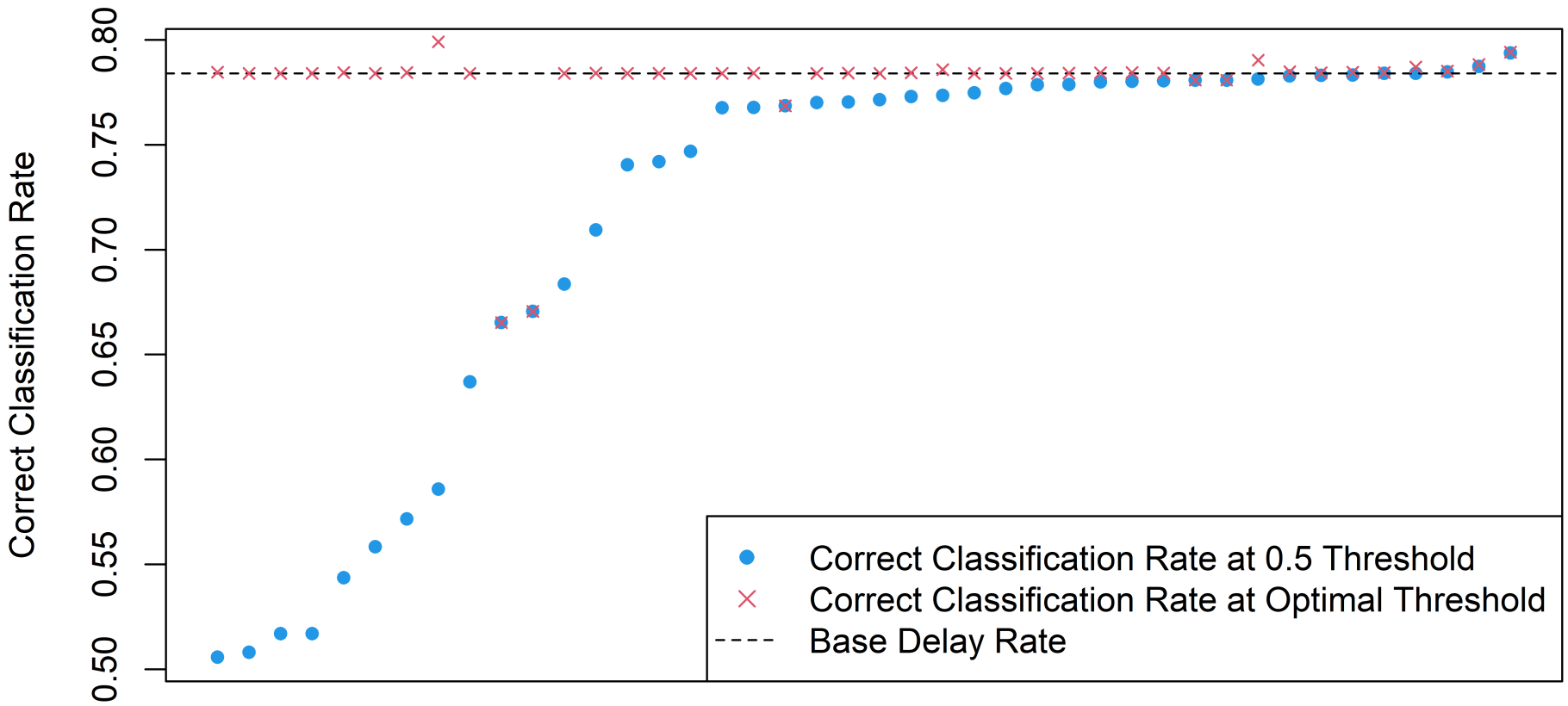
+ 18 other teams with a test accuracy above the constant predictor of most likely class.

# Our Most Realistic Predictors

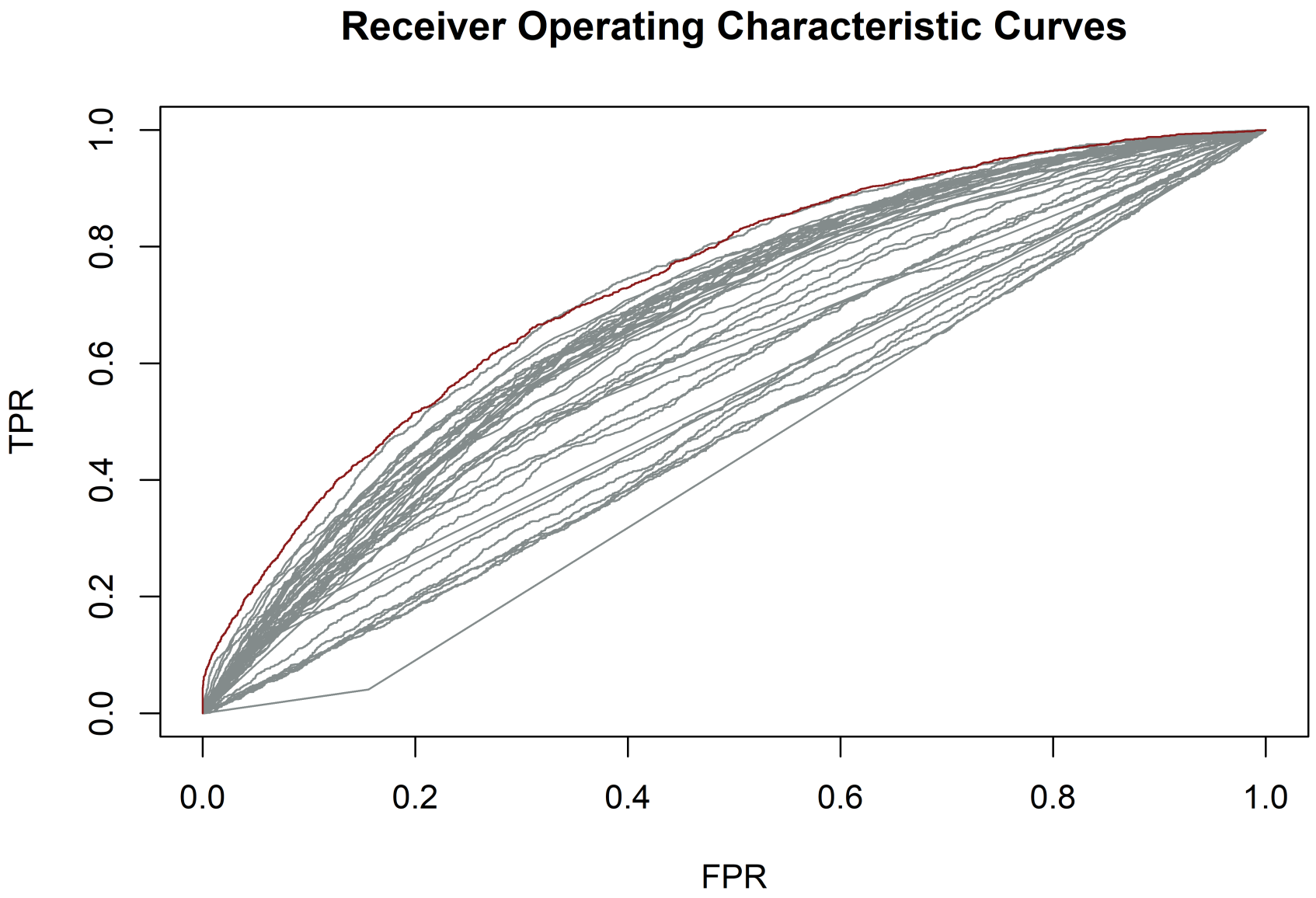
| Team Name | Test Data Accuracy | MAD Estimation Error |
|---|---|---|
| Stat7Team | 0.7842613 | 0.004261304 |
| horse | *Base Rate* | 0.005971173 |
| Phoenix | *Base Rate* | 0.007157334 |
| IGuessed | 0.7882134 | 0.007713414 |

+ 21 other teams with a test accuracy within 0.08 of their predicted.

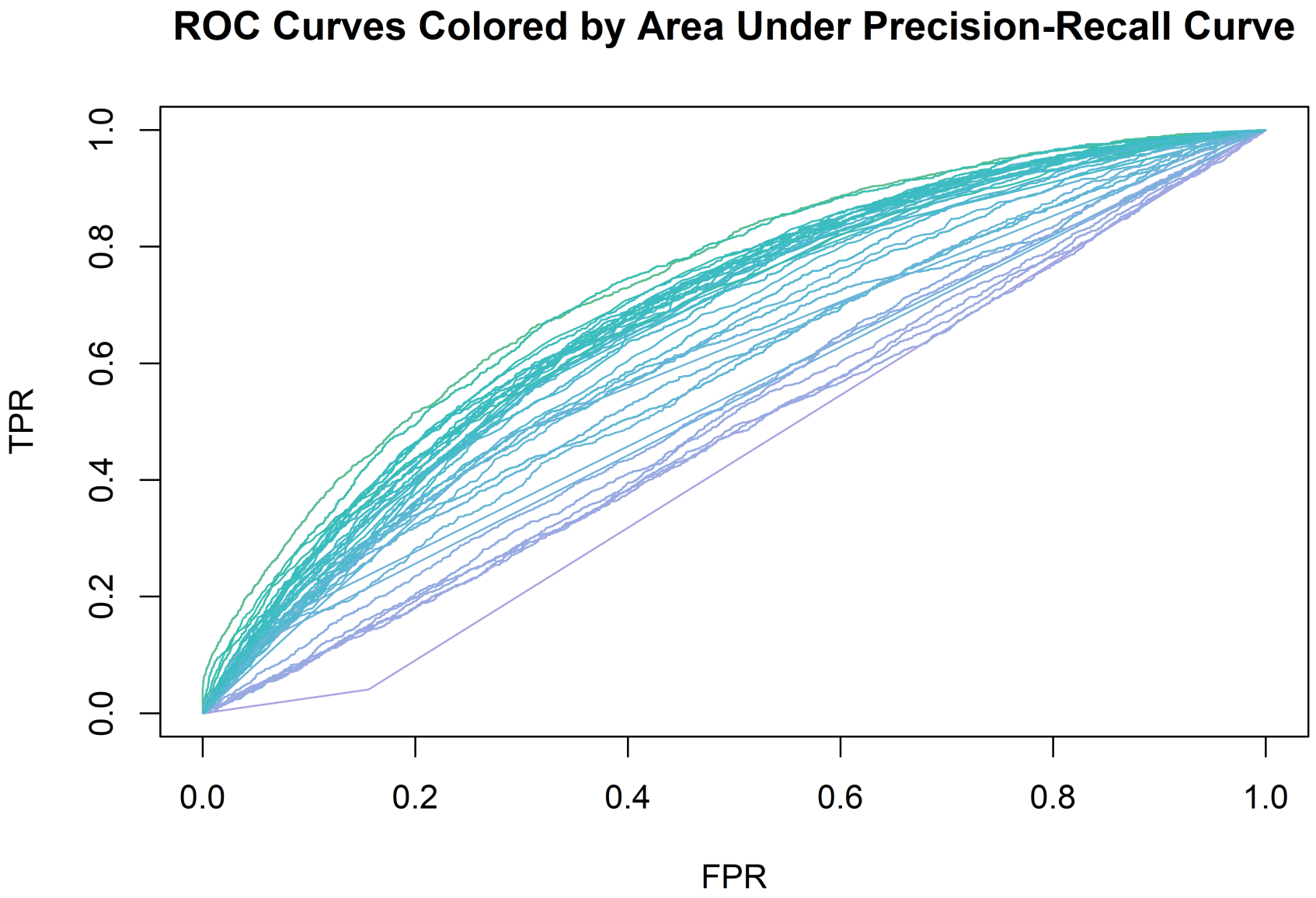
# Test Accuracy by Team



# ROC by Team

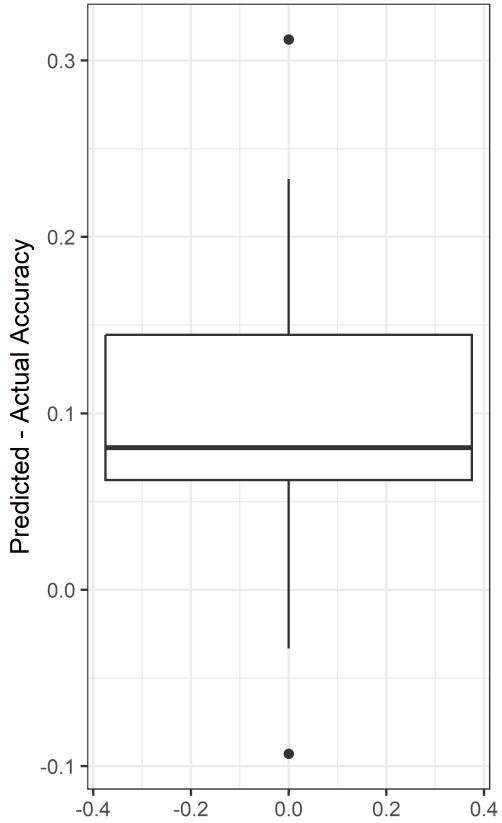

Receiver Operating Characteristic Curves
TPR
0.0
0.2
0.4
0.6
0.8
1.0
FPR
0.0
0.2
0.4
0.6
0.8
1.0

# AUC-PR by Team



# Summary of Test Data Accuracy / Estimate

Box plot of "Your Guess of Test Accuracy - Test Set Accuracy".




There is strong evidence suggesting that nearly all teams used some form of in-sample prediction error as the guess of test error.

# A Further Question

- How confident are we when using the test data error rate to decide the winners?
- Arguments in favor of this decision:
  - The test error rate is an unbiased risk estimate
  - The test error rate has small variance when the test sample size is large
- Cautionary thoughts:
  - The test data error rate still has randomness in it.
  - Could it be that the runners-up got an unlucky split of their fitting sample?

# Uncertainty Quantification

- UQ is a big topic in statistics and machine learning: essentially the statistical learning algorithms' outputs are educated guesses. But how confident are we about these guesses?
- Sources of uncertainty:
  - The natural randomness in the training and testing data
  - Imperfect knowledge about the data generating mechanism:
    - the models we choose to use
    - the parameter values we estimate
  - Additional coin tosses inherent to the algorithm (e.g., sample splitting, stochastic gradient descent, random forest, ...)

# Dealing with Uncertainty

Natural randomness:

- $Y = f(X) + \epsilon$, where $\epsilon$ is some measurement error.
- Prediction interval: $\widehat{C}_\alpha(X) = [\widehat{f}(X) - \widehat{q}_{\alpha/2},\ \widehat{f}(X) + \widehat{q}_{1-\alpha/2}]$ where
  - $\alpha$ is a mis-coverage level
  - $\widehat{q}_\alpha$ is an estimate of the $\alpha$-quantile of $\epsilon$
- $\mathbb{P}(Y \in \widehat{C}_\alpha(X)) \approx 1 - \alpha$, if $\widehat{f}$ is accurate.

# Dealing with Uncertainty

Parameter uncertainty:

- $Y = f(X) + \epsilon$, where $\epsilon$ is some measurement error.
- Let's say $f(x) = f(x; \theta)$, parameterized by $\theta$
- $\widehat{f}(X) = f(x; \widehat{\theta})$
- Frequentist: construct confidence intervals for $\theta$
- Bayesian: use prior distributions on $\theta$

# Dealing with Uncertainty

Model uncertainty:

- This is basically asking: what can we say about our prediction when the model used could potentially be wrong?
- Perhaps the hardest one to deal with
- Resampling and randomization techniques
  - Bootstrap: generate additional random samples from a similar distribution.
  - Permutation, subsampling
  - Randomization: artificially tweak the data to create some variety in the output, with careful calibration (example: conformal prediction)
- Bayesian approach: create a distribution of models