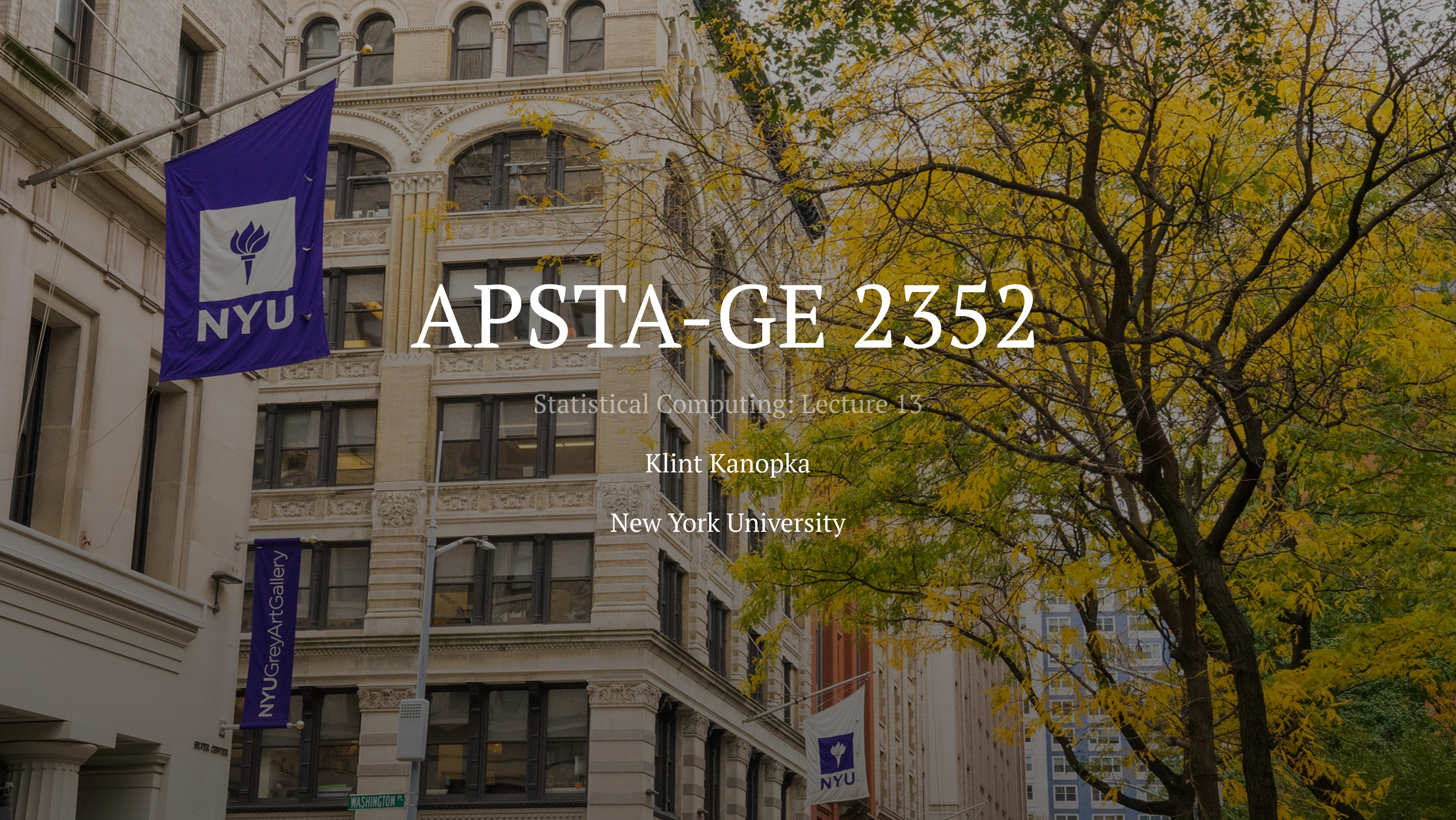

APSTA-GE 2352

Statistical Computing: Lecture 13

Klint Kanopka

New York University

Table of Contents

1. Statistical Computing - Week 13
   1. Table of Contents
   2. Announcements
2. Evaluation Theory
   1. Model Evaluation
   2. Data Splitting
3. Cross Validation
   1. k-Fold Cross Validation in Practice
   2. Selecting k and Tradeoffs
   3. Implementing Cross Validation
4. Wrap Up

Announcements

- PS7 is posted
  - It’s your last one!
  - **Must be submitted on Gradescope!**
- After today:
  - One more lecture
  - One more lab
- Next week:
  - Final exam review
  - Course evals

Final Exam Details

- Exam schedule:
  - Thursday, December 18
  - 4-5.50p
  - 60 5th Ave, Rm C10
- Exam format:
  - Entirely hand written
  - You can bring one page of **handwritten** notes
  - You will:
    - Interpret and trace code
    - Implement algorithms written in plain English
    - Answer questions about how and why we do things the way we do
- No surprises, but may be challenging!
- Will be scanned and graded on Gradescope, so you’ll get feedback relatively quickly!

Evaluation Theory

Model Evaluation

- How do we know that we have a model that's any good?
- There exists a litany of retrospective fit statistics, metrics, and statistical tests
  - These are sometimes specific to certain types of models
  - These are often not comparable between model types or across datasets
  - These all have one killer flaw, however

An Analogy: Course Grading

- This course used to be graded exclusively on problem set performance
  - What are the advantages/disadvantages?
- Some courses have final exams and problem sets
  - What are the advantages/disadvantages?
- What if you distribute practice final exams?
  - What are the advantages/disadvantages?

An Analogy: Course Grading

- Problem set only grading is working with already observed data
  - Upside: Easier for everyone!
  - Upside: Students get time to work through problem sets, find resources, and do their best
  - Downside: You don’t have a good estimate of how students will do on new tasks that you didn’t design for them (you’ve evaluated the outcome of a course on the process of learning)
  - Think of this as *in-sample evaluation*
- Adding a final exam adds a new test challenge to the end!
  - Upside: You now have a new, novel task that students have to complete using only what they’ve learned
  - Downside: Students could prepare for the wrong things and totally bomb it (overfitting)
  - You’re evaluating course learning on new, novel observations. This is *out-of-sample (OOS) evaluation*
- How do practice finals change this?
  - They give students a chance to adjust to what the new task might be like (reducing overfitting)
  - They have to devote time spent studying to time spent taking practice tests (less training)
- What do you think is *best*?

Back to Models

- The fatal flaw of traditional model evaluation is that it only uses the data in hand and tells you nothing about if your model will generalize to new data or is overfitting to the data you used to fit it
- When fitting models, we typically only have the data in hand and the common wisdom is to use as much of it as humanly possible!
- Implementing one of these evaluation schemes requires splitting up the data you have, so you don't use all of it to fit models. We call these *sets*

Data Splitting

- The *training set* is the set of data points used to fit the model. This should **always** be your largest subset of data
- The *testing set* (or *test set* or *holdout set*) is the set of data points used to make your final evaluation
  - The model fit on the training set makes new predictions for the OOS data in the testing set
  - Then you compute some fit metric (like MSE or accuracy)
  - The test set should be as small as possible while still giving you a sample of points to evaluate on
  - If you have model parameters to tune, like regularization parameters or decisions between which model to use, you can't make that selection based on test set performance, because then it becomes in sample data!
- The *validation set* (or *eval set*, *development set*, or *dev set*) is not used to train the model, but used to pick the best values for user-tunable parameters
  - Things like learning rates in gradient descent, or amounts of regularization, or balance between LASSO and Ridge regularization
  - Should be the same size as your test set

In Practice

- The gold standard when you don't have any tunable model parameters is a train/test split, often 80% of data in the training set and 20% in the testing set (called an 80/20 split)
  - If you have a huge amount of data, you can push this to a 90/10 split or even smaller for the test set
- If you have tunable model parameters or modeling decisions, the gold standard is a train/dev/test split
  - Often this is 80/10/10
- The problem is that you want to have as much data as possible inside your model
  - This is especially true of more flexible machine learning models!
- There is another way…

Cross Validation

Cross Validation

- Big idea: What if, instead of holding out your dev set, you divide your train set into chunks, and use those chunks to make your modeling decisions?
- This is worse than holding out a dev set, but *much* better than not using a dev set at all!
- Once you've made your modeling decision based upon the cross validated (CV) performance, you fit a final model using those parameters on your training set and then test
- If there are no modeling decisions, you can also use CV instead of a test set
  - This is worse than using a test set
  - Only provides an estimate of OOS performance
  - Might be your best bet in low data situations

k-Fold Cross Validation in Practice

1. Divide your training set into k buckets, called *folds* (5 or 10 are commonly used)
2. For each fold, $i \in \{1, \ldots, k\}$:
   - Fit the model to the data **not** in fold i
   - Use the model to predict outcomes for the data in fold i
3. Evaluate performance on these CV predictions
4. Fit a model to the entire training set
5. Test performance

Selecting k and Tradeoffs

- Higher values of k put fewer observations into each test fold, and more observations into each model
- This requires fitting $k + 1$ models, however! If model fitting is slow, you want to balance that in your selection of k
- If you have N observations, what happens if $k = N$?
  - This is called *Leave One Out Cross Validation (LOO-CV)*
  - Each model is fit on $N - 1$ data points to predict an outcome for 1 observation
  - Requires fitting N models
- **In the final Problem Set, you'll implement parallelized LOO-CV!**

Implementing Cross Validation

Data and code available here

Wrap Up

Recap

- Final exam
  - You can bring one 8.5×11 sheet of **handwritten** notes
  - The focus is less on memorizing specific algorithms and more on writing and understanding efficient and effective `R` code
  - Think about stuff we've hammered all semester!
- Out of sample (OOS) evaluation is always the gold standard for understanding the ability of a model to generalize
  - This becomes increasingly important as you implement more complex and flexible machine learning models
  - This is the primary way to *honestly* communicate about model performance
- k-fold cross validation
  - Sometimes we don't have enough data to make fully OOS evaluation reasonable
  - Splitting your data, refitting models, and estimating performance is a good compromise
  - Cross validation is also good for tuning parameters

Final Thoughts

- PollEv.com/klintkanopka