# Lecture 4:
# OLS Numerical Properties
# Github Tools

## Big Data and Machine Learning for Applied Economics
## Econ 4676

Ignacio Sarmiento-Barbieri

Universidad de los Andes

August 20, 2020

# Agenda

# Recap

- We started shifting paradigms
- Linear Regression
- Prediction vs Estimation
- Train and Test Samples
- Example in `R`

# Motivation

- Working model is

$$y = f(X) + u \tag{1}$$

- Linear regression is the "work horse" of econometrics and (supervised) machine learning.

$$y = X\beta + u \tag{2}$$

  - All the interest is on $\beta$
  - Gauss-Markov Theorem says that under classical assumptions it is BLUE

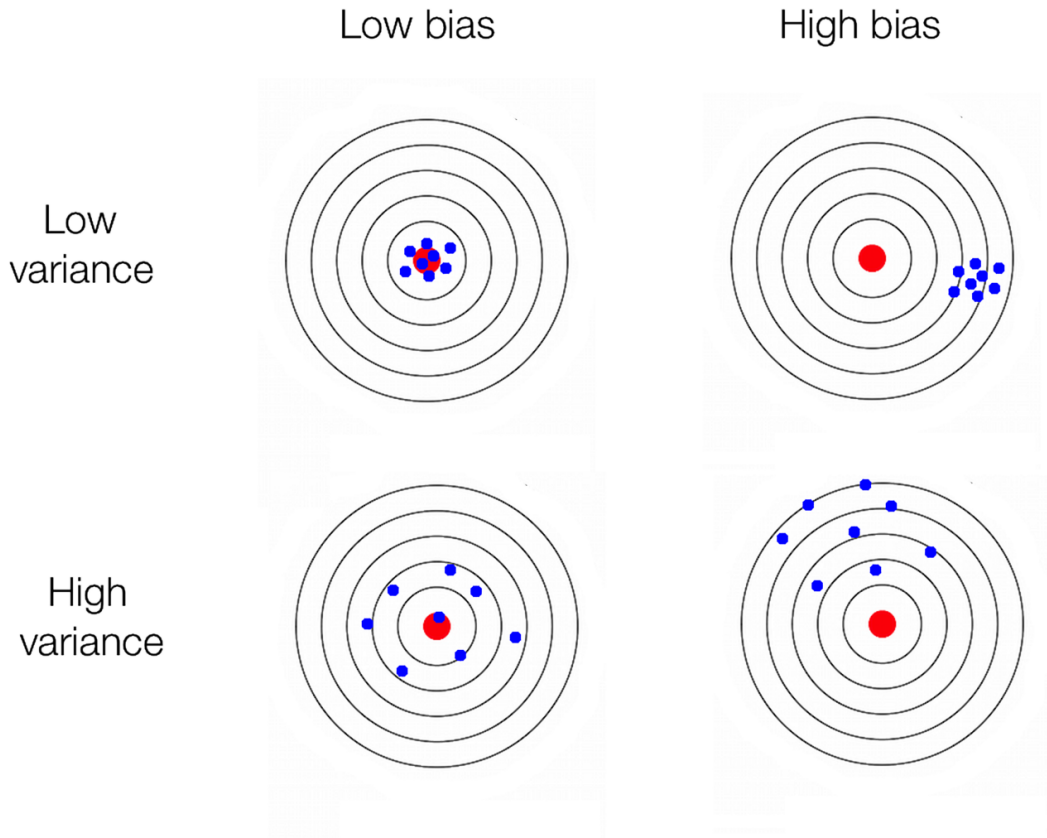
# Mean Square Error

$$MSE(\beta) = E(\hat{\beta} - \beta)^2 \tag{3}$$

$$= E(\beta - E(\hat{\beta}))^2 + Var(\hat{\beta}) \tag{4}$$

- Intuitively, the result says that how wrong is the estimate (MSE) depends on:
  - how uncentered it is (bias) and
  - how dispersed it is around its center (variance).

# Mean Square Error




Source: https://tinyurl.com/y3xlh87o

# Prediction Error

- Now suppose that the goal is to predict $Y$ with another random variable $\hat{Y}$.
- The *prediction error* is defined as:

$$Err(\hat{Y}) \equiv E\left(Y - \hat{Y}\right)^2 \tag{5}$$

- Conceptually the prediction error is equal to the MSE
  - MSE compares a RV ($\hat{\beta}$) with a parameter ($\beta$)
  - $Err(\hat{Y})$ involves two RV

# Prediction Error

Then

$$Err(\hat{Y}) = E(Y - \hat{f})^2 \tag{6}$$

$$= MSE(\hat{f}) + \sigma^2 \tag{7}$$

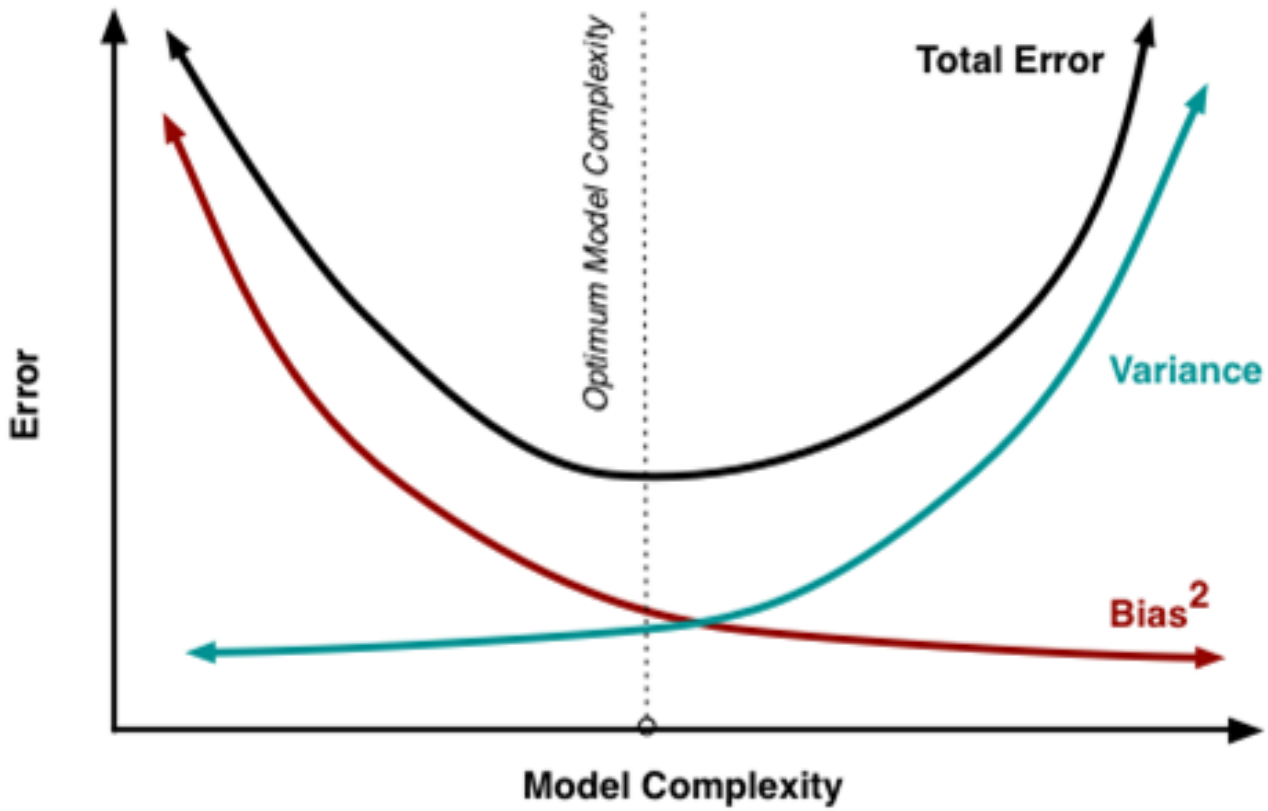
$$= Bias^2(\hat{f}) + V(\hat{f}) + \sigma^2 \tag{8}$$

Two parts:

- the error from estimating $f$ with $\hat{f}$. (*reducible*)
- the error from not being able to observe $u$. (*irreducible*)

This is an important result, predicting $Y$ properly we need a good estimate of $f$.

# Prediction Error




Source: https://tinyurl.com/y4lvjxpc

A very interesting discussion in a recent Twitter thread by Daniela Witten:

https://twitter.com/daniela_witten/status/1292293102103748609?s=20

# Prediction Error

- Suppose that we have a linear regression model $y = X\beta + u$. The prediction error
- $\hat{y} = \hat{\beta}X$ is the prediction
- The estimated prediction error is

$$\hat{Err}(\hat{Y}) = \sum (y_i - \hat{y}_i)^2 \tag{9}$$

- Common alternatives involve: the mean or the square root
- In Econometrics

$$\hat{Err}(\hat{Y}) = \sum_{i=1}^{n} e_i^2 \tag{10}$$

- $R^2 = 1 - \frac{Err(\hat{Y})}{TSS}$
- OLS minimizes $\hat{Err}(\hat{Y})$ and maximizes $R^2 \rightarrow$ minimizing the predictive error is to maximize fit in the sample

# Prediction Error

Challenge:

- The goal of machine learning is *out of sample* prediction
- Minimize the prediction error outside of the sample
- OLS designed to minimize inside the sample
- Predicting well in sample doesn't mean that it would work outside
- There are estimators that work very well in sample but very badly outside (Overfit) more on this later

# Train and Test Samples

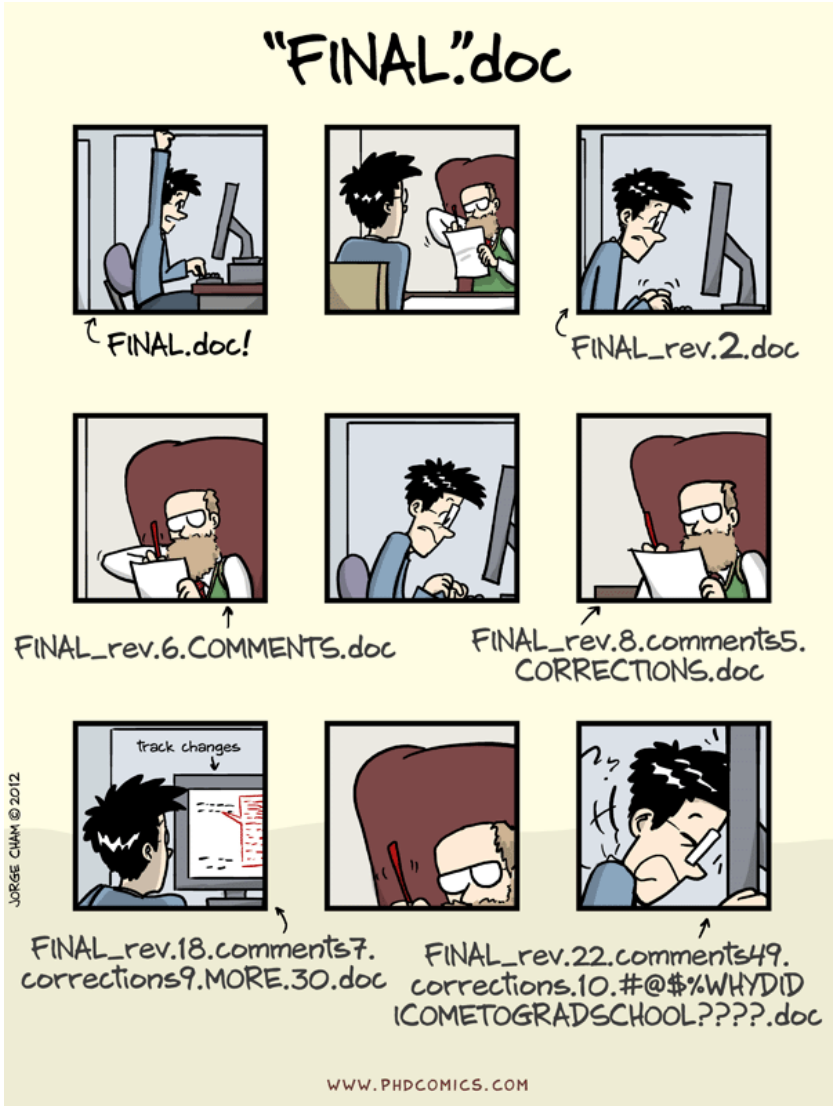
- Problem with OLS (and other estimators)
  - Minimize the prediction error outside of the sample
  - OLS designed to minimize inside the sample
- A workaround: split the data
  - Training sample: to build/estimate/train the model
  - Test sample: to evaluate its performance

# Predicting House Prices in R

Example: Predicting House Prices in R
(switch to RStudio)

# Intro to Git(Hub)

## Motivation




Source: http://phdcomics.com/comics/archive_print.php?comicid=1531

# Intro to Git(Hub)

## Motivation

- Git
  - Git is a distributed version control system
  - It is optimized for the things that economists and data scientists spend a lot of time working on (e.g. code).
- GitHub
  - It's important to realize that Git and GitHub are distinct things.
  - GitHub is an online hosting platform built on top of the Git system. (Like Bitbucket and GitLab.)
  - Just like we don't *need* Rstudio to run R code, we don't *need* GitHub to use Git

# Repositories: Init

- Let's get hands on
- Create a Repo: go to github and clicking the green "new repository"
  - Step 1: Name your repository
  - Step 2: Public or private
  - Step 3: Add a README
  - Step 4: License e.g. `https://choosealicense.com/`
  - Step 5: gitignore
- Congratulations!

# Repositories: Clone

1. First you need to get your repo url
2. Open up your command line tool, and write `git clone` with the url you copied. For example:

```
$ git clone https://github.com/ignaciomsarmiento/test_repo.git
```

3. Go to you click on your file explorer if you are mousey, type `ls` (unix based) or `dir` (windows) in a terminal window if you are penguinesque.

```
$ ls
```

4. Check what you cloned!

# Repositories: Useful commands

- See commit history (hit spacebar to scroll down or q to exit).

```
$ git log
```

- or what has changed?

```
$ git status
```

- to add files

```
$ git add NAME-OF-FILE-OR-FOLDER
```

# Repositories: Useful commands

You can use wildcard characters to stage a group of files (e.g. sharing a common prefix).

- Stage all files.

```
$ git add -A
```

- Stage updated files only (modified or deleted, but not new).

```
$ git add -u
```

- Stage new files only (not updated).

```
$ git add .
```

- Commit your changes.

```
$ git commit -m "Helpful message"
```

# Repositories: Useful commands

Pull from the upstream repository (i.e. GitHub).

```
$ git pull
```

Push any local changes that you've committed to the upstream repo (i.e. GitHub).

```
$ git push origin master
```

`origin` is a shorthand name for the remote repository that a project was originally cloned from. and `master` the branch we are pushing to.

# Collaboration time

- I'm going to create "breakout rooms" of two people, and they are going to last 5 min
- This is what I want you to do
  - Partner 1: Invite Partner 2 to join you as a collaborator on the `test_repo` GitHub repo that you created earlier. (See the `Settings` tab of your repo.)
  - Partner 2: Clone Partner 1's repo to your local machine. Make some edits to the README (e.g. delete lines of text and add your own). Stage, commit and push these changes. Make sure you change into a new directory first, with a different name so you don't have conflicts with your own
  - Partner 1: Make your own changes to the README on your local machine. Stage, commit and then try to push them (*after* pulling from the GitHub repo first).

# Collaboration time

... and we are back

- Did Partner 1 encounter a 'merge conflict' error?

# Collaboration time

... and we are back

- Did Partner 1 encounter a 'merge conflict' error?
- Check what is going on

```
$ gitstatus
```

```
Unmerged paths:
  (use "git add <file>..." to mark resolution)

    * both modified:   README.md
```

- Git is protecting P1 by refusing the merge. It wants to make sure that you don't accidentally overwrite all of your changes by pulling P2's version of the README.

# Collaboration time

```
# README
Some text here.
<<<<<<< HEAD
Text added by Partner 2.
=======
Text added by Partner 1.
>>>>>>> 814e09178910383c128045ce67a58c9c1df3f558.
More text here.
```

# Collaboration time

- Fixing these conflicts is a simple matter of (manually) editing the README file.
  - Delete the lines of the text that you don't want.
  - Then, delete the special Git merge conflict symbols.

- Once that's done, you should be able to stage, commit, pull and finally push your changes to the GitHub repo without any errors.
  - P1 gets to decide what to keep because they fixed the merge conflict.
  - The full commit history is preserved, so P2 can always recover their changes if desired.
  - Another solution is using `branches`

# Branches

**Incomplete Section in e-TA:** the best pull requests for this section get some bonus points =)

- Branches are one of Git's coolest features.
  - Allow you to take a snapshot of your existing repo and try out a whole new idea *without affecting* your main (i.e. "master") branch.
  - Only once you (and your collaborators) are 100% satisfied, would you merge it back into the master branch.¡sup¿1¡/sup¿
  - This is how most new features in modern software and apps are developed.
  - It is also how bugs are caught and fixed.
  - But researchers can easily — and should! — use it to try out new ideas and analysis (e.g. robustness checks, revisions, etc.)
  - If you aren't happy, then you can just delete the experimental branch and continue as if nothing happened.

# Branch Shell Commands

Create a new branch on your local machine and switch to it:

```
$ git checkout -b NAME-OF-YOUR-NEW-BRANCH
```

Push the new branch to GitHub:

```
$ git push origin NAME-OF-YOUR-NEW-BRANCH
```

List all branches on your local machine:

```
$ git branch
```

# Branch Shell Commands

Switch back to (e.g.) the master branch:

```
$ git checkout master
```

Delete a branch

```
$ git -d NAME-OF-YOUR-FAILED-BRANCH

$ git push origin :NAME-OF-YOUR-FAILED-BRANCH
```

# Merging branches + Pull requests

You have two options:

1. Locally
   - Commit your final changes to the new branch (say we call it 'new_idea').
   - Switch back to the master branch: $ git checkout master
   - Merge in the new-idea branch changes: $ git merge new_idea
   - Delete the new-idea branch (optional): $ git branch -d new_idea

2. Remotely (i.e. *pull requests* on GitHub)
   - Notify collaborators — or yourself!
   - Write a summary of the changes
   - You can assign reviewers of your code

# Your first pull request

- You know that "new_idea" branch we just created a few slides back? Switch over to it if you haven't already.
- Remember:

```
$ gitcheckout -b new_idea
```

- Make some local changes and then commit + push them to GitHub.
  - The changes themselves don't really matter. Add text to the README, add some new files, whatever.

# Your first pull request

- After pushing these changes, head over to your repo on GitHub.
  - You should see a new green button with "Compare & pull request". Click it.
  - Add a meta description of what this PR accomplishes. You can also change the title if you want.
  - Click "Create pull request".
  - (Here's where you or your collaborators would review all the changes.)
  - Once satisfied, click "Merge pull request" and then confirm.

For more instructions
https://help.github.com/articles/creating-a-pull-request/

# Forking

- Forking is going to be the way to contribute to my lectures, e-TAs, etc, and get those precious participation points
- By forking a repo then you are really creating a copy of it.
- Once you fork a repo, you are free to do anything you want to it. (It's yours.)
- Combined with a pull request is how you contribute to code development

# Review & Next Steps

- Mean Square Error
- Prediction Error
- Train and Test Samples
- Example in `R`
- Intro to `Git(Hub)`

- **Next Class:** Big Data intro, OLS Numerical Properties Computation.

- Questions? Questions about software?

# Further Readings


- Davidson, R., & MacKinnon, J. G. (2004). Econometric theory and methods (Vol. 5). New York: Oxford University Press.
- James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). An introduction to statistical learning (Vol. 112, p. 18). New York: springer.
- Friedman, J., Hastie, T., & Tibshirani, R. (2001). The elements of statistical learning (Vol. 1, No. 10). New York: Springer series in statistics.
- `Git` tutorials from BDEEP group at NCSA. Mimeo.
- `Git` tutorial from Prof. Grant McDermott.