Machine Learning for Economists

PECN 6100 Guest Lecture

Nicholas Lacoste

Tulane University

2024

Lecture Goals

1. Discuss the differences between Econometrics and Machine Learning
    - Specifically, how econometrics *thinks* about and approaches problems vs. how machine learning does it
2. Introduce ML lingo and discuss statistical issues that are unique to ML
3. Introduce some of the most common ML algorithms and basic ideas of their uses
4. Demonstration for LASSO/Ridge Regression – one of the most common ML approaches used in economic research

Overall: I want to introduce things you'll probably see if you pursue a career/further studies in data science, so that you know the topics employers will expect you to eventually learn about.

The Usual Econometric Problem

- The usual econometric problem has 3 ingredients:

  1. **Target** – usually a parameter of some statistical model (e.g. a conditional mean $E[Y_i|\boldsymbol{X}_i]$, a regression coefficient, etc.)
  2. **Estimator** – we estimate the target parameter by optimizing some objective function. For example, the "Least Squares" estimator (how we get regression coefficients) tries to minimize the Sum of Squared Errors (SSE)
  3. **A joint distribution** – we have data (i.e. we have some outcome Y, and a group of explanatory variables $\boldsymbol{X} = \{x_1, ..., x_k\}$). We assume some distribution (usually the Normal distribution) guides the relationship between Y and $\boldsymbol{X}$

The Usual Econometric Problem

- Here's an example you've seen before – a linear regression:
    1. We have a dataset of some outcome Y_i and covariates $\boldsymbol{X}_i$. We want the mean of Y_i among obervations i that have the same set of $\boldsymbol{X}_i$ values.
    2. This regression will do the trick:

    $$\begin{aligned} Y_i &= \alpha + \beta_1 x_1 + ... + \beta_k x_k + \varepsilon \\ &= \alpha + \boldsymbol{\beta' X} + \varepsilon \end{aligned}$$

    3. We can just plug in values for $\boldsymbol{X}_i$ and it will give us the conditional mean of Y_i
    4. We assumed that Y_i was (conditional on $\boldsymbol{X}_i$) distributed Normally, it had a constant variance σ^2, and that it can be described as a linear equation (three strong assumptions!)

The Usual Econometric Problem

- In that regression, we care about estimating the parameter α and the parameters $\beta_1, ..., \beta_k$. These are the **targets**
- Notice that we can rearrange the regression to isolate the error term:

$$Y_i - \alpha - \boldsymbol{\beta' X} = \varepsilon$$

- Our **estimator** for these parameters is “Ordinary Least Squares” – the terms that minimize total error:

$$(\hat{\alpha}_{ols}, \hat{\boldsymbol{\beta}}_{ols}) = \arg\min_{\alpha,\beta} \sum_{i=1}^{N} (Y_i - \alpha - \boldsymbol{\beta' X})^2$$

- Our parameter estimates are useful if the assumptions we made are correct

The Usual Econometric Problem

- All of that was known as **statistical inference** – that is, we cared about if our β estimates were good/informative, NOT if our predictions of Y_i were accurate.
- But what if we *do* care about prediction? What if Y_i is sales of product i, and $\boldsymbol{X}_i$ are characteristics of product i? If you're a business owner, you might care more about predicting Y_i accurately than understanding the effects of each characteristic on sales (though both are probably interesting).
- The objective function we <u>should</u> care about is minimizing the **mean squared prediction error** on <u>new</u>[1] data: $(Y_{n+1} - \hat{Y}_{n+1})^2$
- Fun fact: if there are more than 2 variables in $\boldsymbol{X}_i$, the OLS estimator is <u>never</u> going to produce the best possible prediction for $\hat{Y}_{n+1}$

[1] The OLS objective function minimizes the <u>in sample</u> error, here we care about <u>out of sample</u> error

Intro to Machine Learning

- Prediction is <u>fundamentally different</u> than inference, and we go about it differently when it is the emphasis.

- The next slides will discuss key ML terms and concepts to unlock the art of prediction!

Intro to Machine Learning

- **Training Sample** – data used to estimate our ML model
- **Test Sample** – data used to validate predictive capabilities of the model
- **Features** – the ML name for covariates/regressors/predictors (i.e. the $\boldsymbol{X}$ variables)
- **Weights** – ML often referrs to regression parameters as "weights"
- **Supervised Learning** – algorithms for cases where we observe both $\boldsymbol{X}$ and Y
- **Unsupervised Learning** – algorithms for cases where we observe only $\boldsymbol{X}$. We usually try to group ("cluster") observations in some way by similarity in these cases[2]
- **Classification Problems** – prediction problems of some discrete outcome (e.g. image recognition, binary choice "yes/no" problems, etc.)

[2] tbh these are pretty limited use, but data scientists love to ask about them in interviews

Intro to Machine Learning

Key machine learning concepts

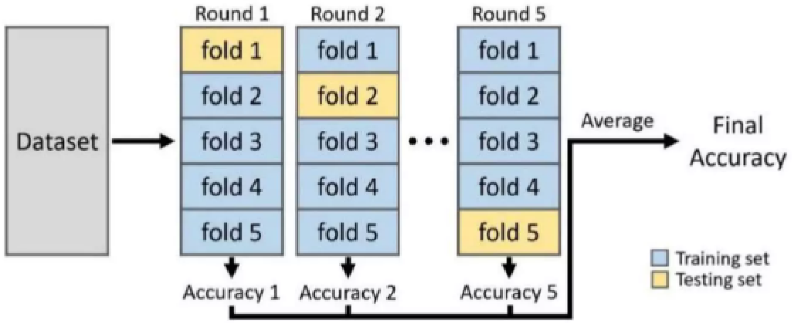
- K-fold cross-validation
- Overfitting, regularization, hyperparameter tuning
- Sparsity (not going to cover)
- Computational limitations and scalability of different ML approaches
- Ensemble methods (i.e. averaging predictions across many models)
- Inference

K-fold cross-validation

- Cross-validation is a strategy in ML to evaluate an algorithm's ability to predict on data it hasn't seen
- Sometimes we'll be comparing multiple different algorithms (e.g. a random forest vs. a neural network), other times we'll be comparing different versions of the same algorithm that we tweaked a bit (called "hyperparameters" – more later)
- The basic idea is to **split** the full data into "folds" (i.e. K evenly sized splits). Then we use all but 1 fold to train a model, then make predictions/evaluate on the held-out fold. We repeat until each fold is held out once, and we average prediction errors across folds

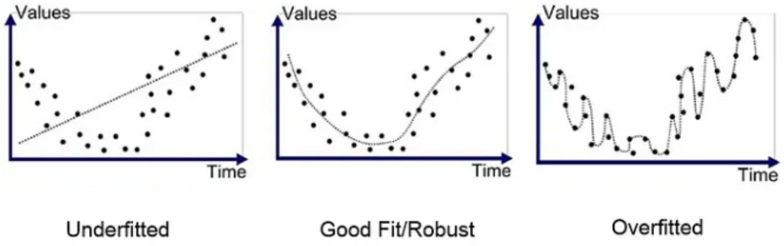
K-fold cross-validation

Example of 5-fold cross-validation



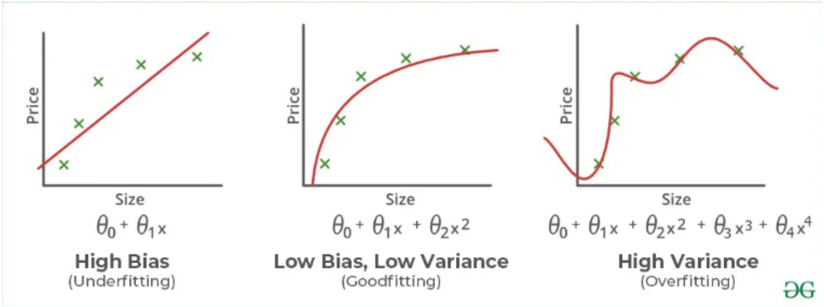
Overfitting, Regularization, and Hyperparameters

- One reason we use cross-validation and emphasise out-of-sample prediction quality is the notion of **overfitting**
- Overfitting is when we abuse the training sample too much. It occurs when the model is so hyper-tuned to the variation in the training data that it's actually capturing randomness. This doesn't lead to accurate predictions on out-of-sample data.



Overfitting, Regularization, and Hyperparameters

- The **Bias-Variance Tradeoff** is a big deal in both Econometrics and ML
- Bias is basially prediction accuracy (in-sample). You can reduce bias, but this increases the variance of your predictions. Too much variance or bias = bad out-of-sample prediction



Overfitting and Regularization

- One approach to avoid overfitting is called **regularization**
- Regularization is the process of penalizing a model when it starts to fit too-well on in-sample data
- For example, say we have a regression with 1000 variables in the $\boldsymbol{X}$ vector and we don't know which ones to include/disclude. If we just blindly include them all, we will probably overfit our training sample unless we have a very large dataset.
- One option is to add a penalty parameter to our regression that shrinks all of our coefficient estimates towards 0. This mechanically reduces variance by focusing on only the most predictive variables while allowing all of them to enter the model
    - I've just described LASSO, Ridge, and Elastic Net regressions! More later on these

Computational Limitations and Scalability

- In normal Econometrics, we don't care too much about computational feasibility – it's kindof rarely an issue
- But in ML, we're often performing some VERY computationally intensive tasks. Just look at cross-validation! By doing 5-folds, we are performing 5x as many computations as we would if we weren't worried about out-of-sample performance
- Again consider the regression example with 1000 covariates. The "brute force" approach to regularization would be to estimate a regression for every combination of covariates and take the one with the best out-of-sample prediction quality. It would take about 10 billion years to estimate that many regressions
- Another reason the LASSO/Ridge/EN approaches are useful

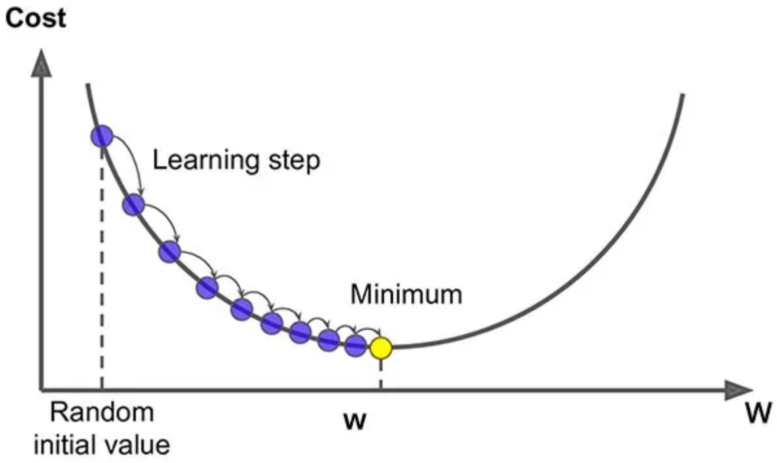
Classical Gradient Descent

- Something you'll see often in ML is **classical gradient descent** – it's good to know what this is and the scalable version **stochastic gradient descent**
- In ML, we don't always estimate parameters with OLS. In fact we usually don't in most algorithms (e.g. Neural Networks)
- But remember the goal in model training/estimation: we want the parameter that minimizes our average out-of-sample prediction error (a.k.a. a **loss function**)
- One way to guarantee we get the error-minimizing parameter is through "classical" gradient descent

Classical Gradient Descent

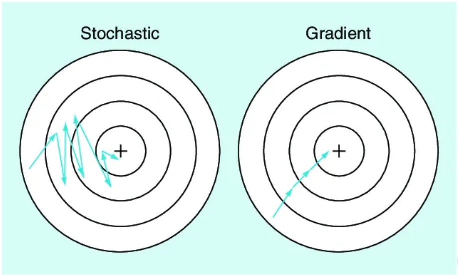
- The basic idea of "classical" gradient descent is that we start with some initial guess at our parameter(s), say we guess $\hat{\boldsymbol{\beta}}_1$
- From there, we calculate the gradient of the loss function (i.e. the derivatives with respect to each data point): $\nabla F(\boldsymbol{x}_i) = \begin{bmatrix} \frac{\partial F(\boldsymbol{x}_i)}{\partial x_1} \\ \vdots \\ \frac{\partial F(\boldsymbol{x}_i)}{\partial x_n} \end{bmatrix}$
- The gradient tells us the fastest direction down the loss function. So if we just update $\hat{\boldsymbol{\beta}}_1$ by changing it by the gradient (times a "step size" determining how big an adjustment to make), we'll move closer to the minimum:
$$\hat{\boldsymbol{\beta}}_2 = \hat{\boldsymbol{\beta}}_1 - \gamma \nabla F(\boldsymbol{x}_i)$$

Classical Gradient Descent



Stochastic Gradient Descent

- Notice to compute the full gradient, we needed to compute partial derivates for EVERY data point. What if we have millions or billions of points? It's not scalable
- **Stochastic gradient descent** is our shortcut: what if we just approximate the gradient by taking a small random subset of observations? Turns out, while each step is a bit noisier, this ultimately works (usually) – and MUCH faster
- This is just one very common example of methods that require alternate approaches to be computationally feasible



Ensemble Methods (Model Averaging)

- In 2007, Netflix held a $1 million prize competition to the team that could make the best predictive model for user movie ratings

Ensemble Methods (Model Averaging)

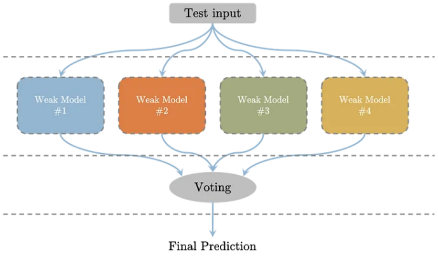
- In 2007, Netflix held a $1 million prize competition to the team that could make the best predictive model for user movie ratings
- Which algorithm did the winners choose? Neural Networks? Any guesses?

Ensemble Methods (Model Averaging)

- In 2007, Netflix held a $1 million prize competition to the team that could make the best predictive model for user movie ratings
- Which algorithm did the winners choose? Neural Networks? Any guesses?
- Turns out, they used **ensembling** – they trained many different algorithms and allowed them to "vote" for the correct prediction, then the final prediction was the popular vote!

Ensemble Methods (Model Averaging)

- This approach to prediction is very common (and successful) in ML
- There are 3 main strategies to implement it – which we sadly don't have time to cover today:
  1. **Bagging**
  2. **Boosting**
  3. **Stacking**
- A solid introductory article: https://towardsdatascience.com/what-are-ensemble-methods-in-machine-learning-cac1d17ed349



Common Algorithms

Algorithms to know about

Common Algorithms

4 broad classes of algorithms for regression (continuous Y) problems:

1. Regularized Linear Regression
   - **LASSO, Ridge, Elastic Nets**
2. Splitting/Partitioning Methods
   - Regression Trees, Random Forests
3. Neural Networks
   - Perceptrons (simplest), Feed-forward networks, recurrent networks (sequential processing – e.g. speech prediction), convolutional networks (image recognition), many more...
4. Boosting methods

Common Algorithms

5 broad classes of algorithms for classification (discrete Y) problems:

1. Regularized disscrete choice maximum likelihood models
    - Regularized logistic regression, regularized probit, etc.
2. Splitting/Partitioning Methods
    - Classification Trees, Random Forests
3. Support Vector Machines
4. Neural Networks and Boosting also work here

Common Algorithms

Some new algorithms that combine prediction and causal inference:

1. Causal Trees and Causal Forests (Athey and Wager, 2018)
2. Meta Learners
   - R-learner, T-learner, etc.
3. Double Machine Learning
4. DeepIV
5. Check out this paper: https://arxiv.org/abs/2110.04442 ; and the EconML package by Microsoft: https://econml.azurewebsites.net/

Intro to LASSO/Ridge/Elastic Nets

Overview of regularized linear regression methods

Intro to LASSO/Ridge/Elastic Nets

- LASSO is still probably the most common ML method used in economic research
- It's the go-to approach when economists have a prediction-based problem as part of their analysis, so it's important to know how to use it
- Remember: the main objective of any ML algorithm is out-of-sample prediction quality, and LASSO/Ridge/EN's aim to train linear models that are good at that goal (because regular OLS is <u>not</u> usually very good at that)

Intro to LASSO/Ridge/Elastic Nets

The key theory:

- We aim to predict an outcome Y. The specific prediction is the average of Y, conditional on our predictors $\boldsymbol{X}$. We'll call this value $g(x)$

$$g(x) = E[Y_i | \boldsymbol{X}_i = x]$$

- We estimate this with a linar regression that includes a **penalty term** that **shrinks our** β **estimates towards zero**. Remember the reason OLS is not good is that it generates noisy parameter estimates (i.e. it overfits). The penalty parameter helps us tune-out the noise:

$$g(x) = \beta_1 x_1 + ... + \beta_k x_k + \varepsilon$$

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \underbrace{\sum_{i=1}^{N} (Y_i - \beta_1 x_{i,1} - ... - \beta_k x_{i,k})^2}_{\text{Regular OLS}} + \underbrace{\lambda (\sum_{i=1}^{K} |\beta_i|^{\alpha})^{1/\alpha}}_{\text{Penalty}}$$

Intro to LASSO/Ridge/Elastic Nets

The terms α and λ are **hyperparameters**

- Hyperparameters affect how a ML model is trained, but we determine them ourselves
- Here, the selection of α determines the actual algorithm we choose:
  1. $\alpha = 1$ corresponds to LASSO
  2. $\alpha = 0$ corresponds to Ridge Regression
  3. $\alpha \in (0, 1)$ corresponds to Elastic Nets that combine LASSO and Ridge
- The key differences: LASSO is very strict in its' penalty – returns non-0 coefficients for only the most important variables in $\boldsymbol{X}$. Useful for understanding the most important predictors.
- Ridge is less restrictive and more conservative – allows more variables to be "important." Useful to prevent us from over-interpreting "important" coefficients
- λ is another hyperparameter determining strictness. As λ increases, shrinkage is stricter. We'll use cross-validation to pick this value

Intro to LASSO/Ridge/Elastic Nets

Coding Demo