Machine Learning for Economists

Class 6: Inside a Model

葛雷

中国人民大学经济院

2025 年 3 月 27 日

中國人民大學
RENMIN UNIVERSITY OF CHINA

Inside a Model

Revisit Linear Regression

Loss function & function form

Optimize the model

Feature Engineering

Lasso

Inside a Model

Revisit Linear Regression

Loss function & function form

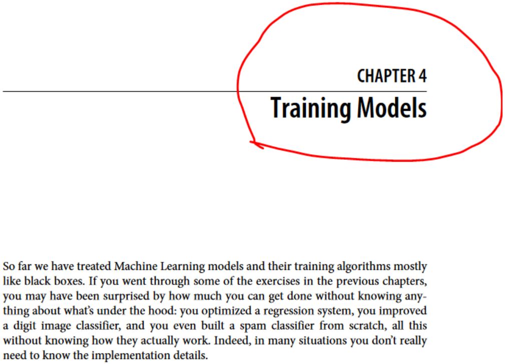
Optimize the model

Feature Engineering

Lasso

Textbook Chapter 4

Please Read Chapter 4 of Textbook



CHAPTER 4

Training Models

So far we have treated Machine Learning models and their training algorithms mostly like black boxes. If you went through some of the exercises in the previous chapters, you may have been surprised by how much you can get done without knowing anything about what's under the hood: you optimized a regression system, you improved a digit image classifier, and you even built a spam classifier from scratch, all this without knowing how they actually work. Indeed, in many situations you don't really need to know the implementation details.

Questions about linear regression?

- Linear regression is outdated?
- We always need fancy models like neural network, ensemble learning?

We use linear regression this class

1. **Fast:** Linear regression is fast → with C++ applied in stock trading (HFS)
2. **Unbiased:** Unbiased Estimation, good for policy estimation and causal study (BLUE)
3. **Simple:** Simple structure to learn the model training process

We explain model training in this class

- Linear regression is simple and transparent
- So, use Linear regression to explain model structure in general
- This model training structure also apply to ensemble learning and artificial neural network

Inside a Model

Revisit Linear Regression

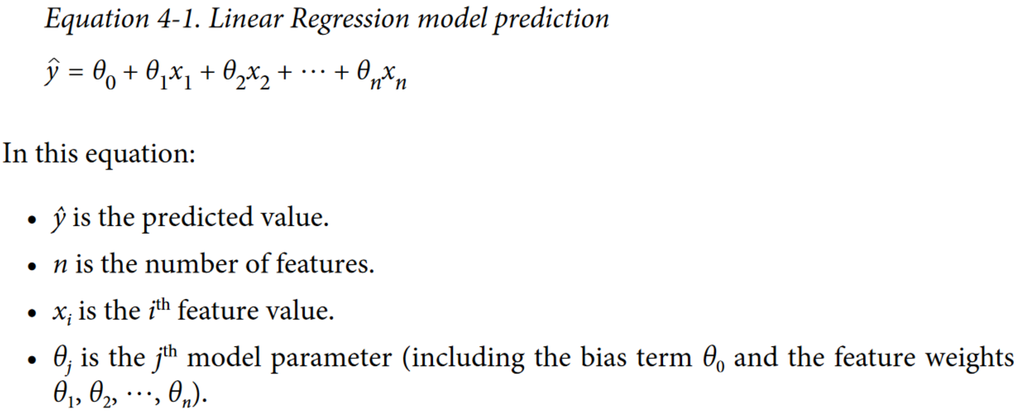
Loss function & function form

Optimize the model

Feature Engineering

Lasso

Linear Regression



Equation 4-1. Linear Regression model prediction

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

In this equation:

- $\hat{y}$ is the predicted value.
- n is the number of features.
- x_i is the i^{th} feature value.
- θ_j is the j^{th} model parameter (including the bias term θ_0 and the feature weights $\theta_1, \theta_2, \cdots, \theta_n$).

Put them into Matrix Form

Given a linear model:

$$y = X\beta + \epsilon$$

where:

- y is the $n \times 1$ vector of observed responses,
- X is the $n \times p$ design matrix (with a column of 1s for the intercept),
- β is the $p \times 1$ vector of unknown coefficients,
- ϵ is the $n \times 1$ error vector.

OLS = > Minimize RSS

Minimize the **residual sum of squares (RSS)**:

$$\text{Loss function} = \text{RRS} = \|\epsilon\ |^2 = \|y - X\beta\|^2 = (y - X\beta)^T (y - X\beta)$$

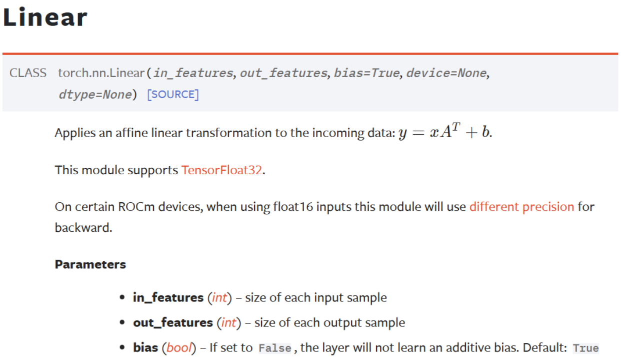
Derivation of the Normal Equation

1. Expand the RSS:
$$\text{RSS} = y^T y - 2\beta^T X^T y + \beta^T X^T X\beta$$
2. Take the gradient with respect to β and set it to zero:
$$\nabla_\beta \text{RSS} = -2X^T y + 2X^T X\beta = 0$$
3. Solve for β:
$$X^T X\beta = X^T y \quad \Rightarrow \quad \beta = (X^T X)^{-1} X^T y$$

   This is the **normal equation aka theoretical solution** (not all algorithms have theoretical solution).

It is linear transformation in Pytorch

https://pytorch.org/docs/stable/nn.html



Linear

CLASS torch.nn.Linear(*in_features*, *out_features*, *bias=True*, *device=None*, *dtype=None*) [SOURCE]

Applies an affine linear transformation to the incoming data: $y = xA^T + b$.

This module supports TensorFloat32.

On certain ROCm devices, when using float16 inputs this module will use different precision for backward.

Parameters

- **in_features** (*int*) – size of each input sample
- **out_features** (*int*) – size of each output sample
- **bias** (*bool*) – If set to `False`, the layer will not learn an additive bias. Default: `True`

Inside a Model

Revisit Linear Regression

Loss function & function form

Optimize the model

Feature Engineering

Lasso

L function and f function

This semester we will use $\boxed{\text{two important functions } L \text{ and } f \text{ function}}$

- L is the loss function (distance btw Y and $\hat{Y}$)
- $f(x;\theta)$ is the prediction function, where x is feature and θ is model parameters (OLS, Random Forrest, Artificial Neural Network and ...)

The most common L loss function is MSE

- The most common L loss function is MSE
- $Loss = (Y - \hat{Y})^2 = (Y - f(\theta))^2 = (Y - X\theta)^2$
- Question: Y not follow which distribution, should we use MSE loss

L loss function is not only MSE

- L loss function is not only MSE
- What if Y not follow normal distribution (Yep, pretty common in stock, housing, mortgage default, bank risk and etc)
- Gamma Loss, Median Loss, Quantile Loss, Cross Entropy Loss, etc.
- **Lets go to the blackboard !!!**

Inside a Model

Revisit Linear Regression

Loss function & function form

Optimize the model

Feature Engineering

Lasso

How to find optimal parameter *theta* for our model

- Our Target: my best prediction function $f(x;\theta)$, so that we predictions $\hat{Y}$ are good

- Aka, we should find θ to minimize the Loss over the sample: $L(Y,\hat{Y}) = L(Y, f(x;\theta))$

- Do we always have normal form math solution by using OLS? Nope

- But How to find optimal parameter *theta* for other models such as LASSO, Gradient Boost, ANN? → Gradient Descent Method

Gradient Descent Method

Definition

The gradient descent method is an optimization algorithm to find a local minimum of a differentiable function. The update rule is:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta L(\theta_t)$$

where:

- θ_t is the current point the parameters of our model
- L here is cost function, which is the sum of all loss functions across all sample
- $\nabla L_\theta(theta_t)$ is the gradient at $theta_t$
- $\eta > 0$ is the learning rate

We use iterations to find optimal θ

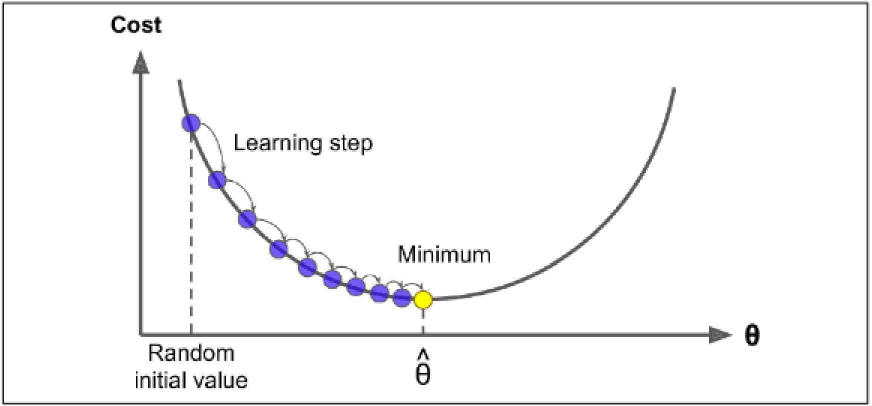


Figure 4-3. In this depiction of Gradient Descent, the model parameters are initialized randomly and get tweaked repeatedly to minimize the cost function; the learning step size is proportional to the slope of the cost function, so the steps gradually get smaller as the parameters approach the minimum

Gradient Descent vs Stochastic Gradient Descent

- Gradient Descent: each iteration use all data sample
- Stochastic Gradient Descent: each iteration use only subsample of data sample
- Stochastic Gradient Descent is the method we use often, why ???

Inside a Model

Revisit Linear Regression

Loss function & function form

Optimize the model

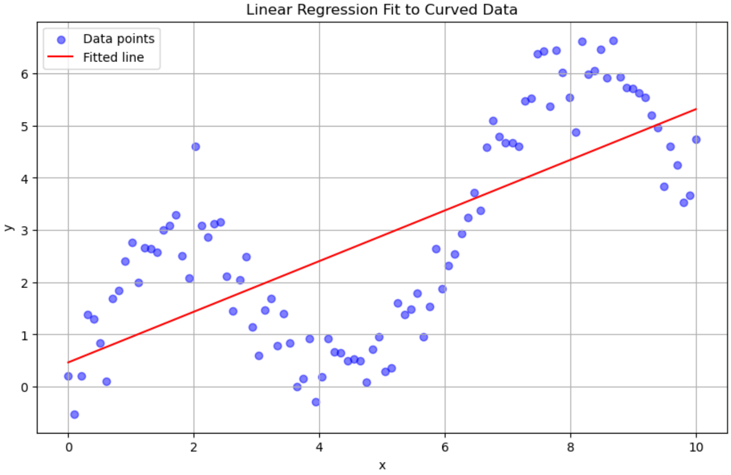
Feature Engineering

Lasso

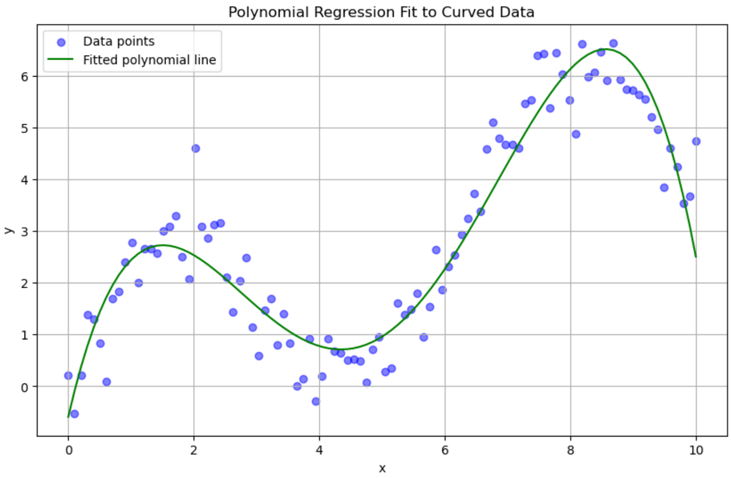
Feature Engineering

- Add nonliearity in model
- Add interaction to model

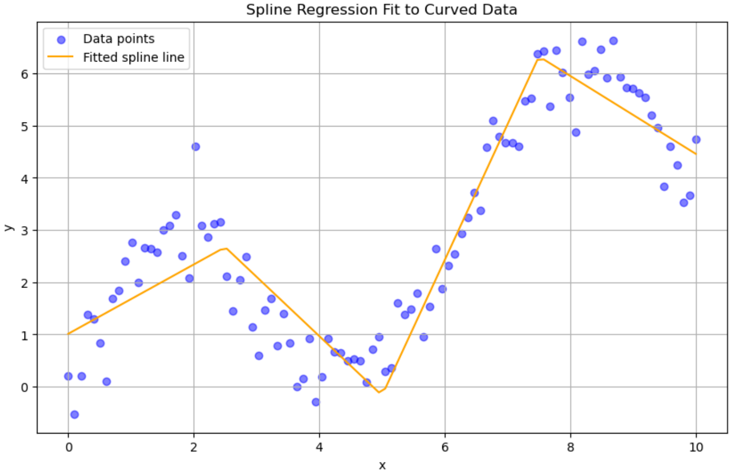
Why we need nonliearity: simple linear regression not fit



Polynomial features to fit



Spline features to fit (More practical in Industry)



Feature Engineering: Interaction

- Why we need features interactions?
- Feature with interact each other for the price prediction
- Please refer to our Homework 3

Inside a Model

Revisit Linear Regression

Loss function & function form

Optimize the model

Feature Engineering

Lasso

Lasso, Ridge, Elastic Net

- **Lasso:** add L1 regularization to the loss function of linear regression
- **Ridge:** add L2 regularization to the loss function of linear regression
- **Elastic Net:** add L1+L2 regularization to the loss function
- How to add? Please follow me to the blackboard

Why we need LASSO

- **Problems:** Without regularization, models may fit noise in training data, leading to poor generalization. (It is the famous over-fitting problem)
- Regularization: selects features from a correlated group and shrink out others.
- Please refer to the 04_training_linear_models.ipynb

Reference

1. Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow (3rd edition)
2. Kaggle
3. Wikipedia
4. ChatGPT
5. DeepSeek