MACHINE LEARNING FUNDAMENTALS

Amir Ghaderi, Ph.D., P.Eng.
February – March 2020
Tillyard Auditorium

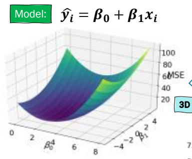
Session 2: Feb 26th

1

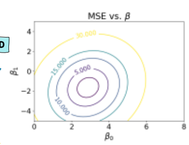
2

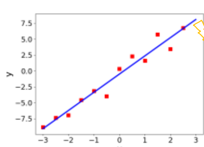
Gradient Descent: A closer look at cost function for simple linear regression

Model: $\hat{y}_i = \beta_0 + \beta_1 x_i$



$$\text{MSE} = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$





Line of best fit (min. cost)

2

3

Gradient Descent: Computational complexity of normal equation

This equation is linear with regards to the number of instances in the training set (it is O(m))

To obtain coefficients, the normal equation should compute the inverse of $X^T.X$, which is an $n \times n$ matrix (n: number of features)

The *computational complexity* of inverting such a matrix is typically about $O(n^3)$

For very large number of n or m we should find a better way to train a linear regression model

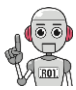
Gradient Descent is the most used learning algorithm in Machine Learning

Gradient descent is an optimization algorithm used to find the values of coefficients of a function that minimizes a cost function

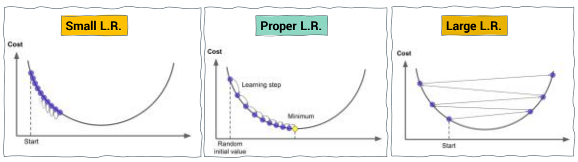
3

4

Gradient Descent: Concepts



Concretely, you start by filling β with random values (this is called *random initialization*), and then you improve it gradually, taking one baby step (called learning rate η) at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm *converges* to a minimum.

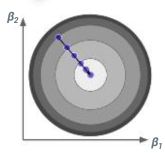


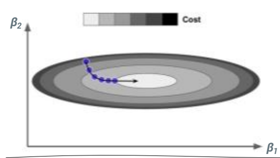
4

5

Gradient Descent: Feature scaling is a must

The cost function has the shape of a bowl, but it can be an elongated bowl if the features have very different scales





***With** scaled features, GD algorithm goes straight toward the minimum, thereby reaching it quickly*

Without feature scaling, GD first goes in a direction almost orthogonal to the direction of the global minimum, and it ends with a long march down an almost flat valley. It will eventually reach the minimum, but it will take a long time

5

6

Gradient Descent: GD in math language

$$\frac{\partial}{\partial \beta_j} MSE(\beta) = \frac{2}{m} \sum_{i=1}^{m} (\beta^T . x^{(i)} - y^{(i)}) x_j^{(i)}$$

Partial derivatives of the cost function

How to deal with j index?

❶ Batch GD

❷ Stochastic GD

❸ Mini-batch GD

6

7

Gradient Descent: Batch GD

$$\nabla_\beta MSE(\beta) = \begin{bmatrix} \frac{\partial}{\partial \beta_0} MSE(\beta) \\ \frac{\partial}{\partial \beta_1} MSE(\beta) \\ \vdots \\ \frac{\partial}{\partial \beta_n} MSE(\beta) \end{bmatrix} = \frac{\mathbf{2}}{\boldsymbol{m}} \boldsymbol{X}^T . (\boldsymbol{X}.\boldsymbol{\beta} - \boldsymbol{y})$$

Closed-form equation for gradient vector of the cost function (X is the design matrix)

$$\beta^{(next\ step)} = \beta - \eta \nabla_\beta \mathrm{MSE}(\beta)$$

GD step

η is called learning rate and it is the model's *hyperparameter*

What the heck is model's hyperparameter and what is its difference with model's parameter?

"Premature optimization is the root of all evil"
— Donald Ervin Knuth

7

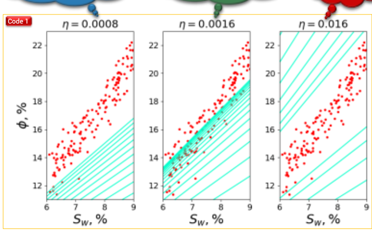
8

Gradient Descent: Importance of hyperparameters in iterative optimization

Very small L.R. will take a long time to converge

A suitable L.R. is sometimes time consuming to find!

Large L.R. will never converge



Main drawback: Batch GD uses the whole training set to compute the gradients at every step, which makes it very slow

How to set the number of iterations?

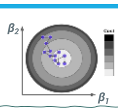
A simple solution is to set a very large number of iterations but to interrupt the algorithm when the gradient vector becomes tiny – that is

$$\|\nabla_\beta MSE(\beta)\| < \epsilon$$

8

9

Gradient Descent: Stochastic GD



The algorithm much faster since it has very little data to manipulate at every iteration

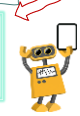
It makes it possible to train on huge training sets

Randomness is good to escape from local optima

Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradients based only on that **single instance**

During optimization, the cost function will bounce up and down and once the algorithm stops, the final parameter values are good, but not optimal

- One solution to this dilemma is to gradually reduce the learning rate
- The function that determines the learning rate at each iteration is called the *learning schedule*



9

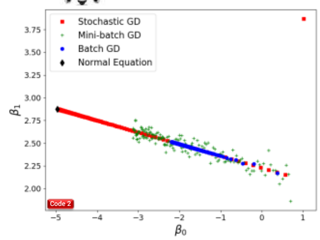
10

Gradient Descent: Mini-batch GD

At each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Mini-batch GD computes the gradients on small random sets of instances called *minibatches*

The main advantage of Mini-batch GD over Stochastic GD is that you can get a performance boost from hardware optimization of matrix operations, especially when using GPUs.

Comparison of different GDs with similar iterations



10

11

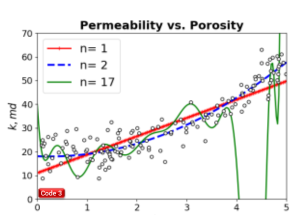
Linear Regression: Comparison of algorithms

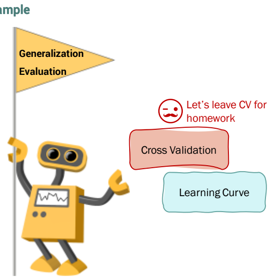
Algorithm	Large m	Large n	Hyperparameters	Scaling required	Scikit-learn
Normal Eq.	Fast	Slow	0	No	LinearRegression
Batch GD	Slow	Fast	2	Yes	n/a
Stochastic GD	Fast	Fast	>=2	Yes	SGDRegressor
Mini-batch GD	Fast	Fast	>=2	Yes	n/a

11

12

Model's Generalization Performance: Polynomial Example





12

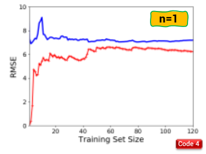
13

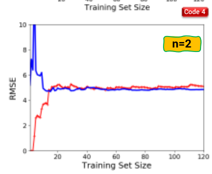
Model's Generalization Performance: Learning curves

- These are plots of the model's performance on the training set and the validation set as a function of the training set size.
- To generate the plots, simply train the model several times on different sized subsets of the training set


If your model is underfitting the training data, adding more training examples will not help. You need to use a more complex model or come up with better features


One way to improve an overfitting model is to feed it more training data until the validation error reaches the training error





13

14

Regularized Linear Model: Modify cost function

"A good way to reduce overfitting is to regularize the model (i.e., to constrain it): the fewer degrees of freedom it has, the harder it will be for it to overfit the data"

Regularization

- ✓ Ridge
- ✓ Lasso
- ✓ Elastic Net

It is important to scale the data before performing Ridge Regression, as it is sensitive to the scale of the input features. This is true of most regularized models

$$J(\beta) = MSE(\beta) + \alpha\frac{1}{2}\sum_{i=1}^{n}\beta_i^2$$

$$J(\beta) = MSE(\beta) + \alpha\sum_{i=1}^{n}|\beta_i|$$

Note that the bias term β_0 is not regularized (the sum starts at $i = 1$, not 0).

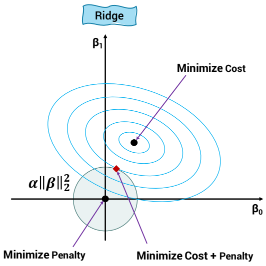
$$J(\beta) = MSE(\beta) + \alpha\frac{1-r}{2}\sum_{i=1}^{n}\beta_i^2 + r\alpha\sum_{i=1}^{n}|\beta_i|$$

14

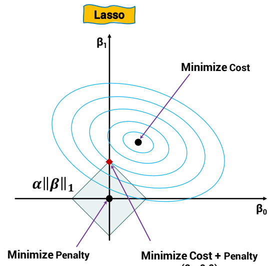
15

Regularized Cost Functions: Graphical illustration

Ridge



Lasso

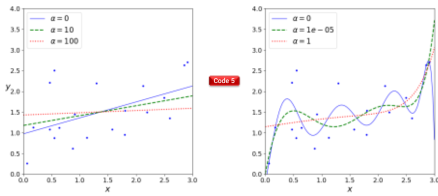


15

16

Regularized Cost Functions: Ridge Regularization Example

- The regularization term should only be added to the cost function during training
- Once the model is trained, you want to evaluate the model's performance using the unregularized performance measure



16

17

Regularized Cost Functions: Elastic-Net Example

```
m = 20
X = 3 * np.random.rand(m, 1)
y = 1 + 0.5 * X + np.random.randn(m, 1) / 1.5

elastic_net = ElasticNet(alpha=0.001, l1_ratio=0.5)
elastic_net.fit(X, y)
prediction = elastic_net.predict([[1.5]])
print(prediction[0])
```

Code 6

In general, Elastic Net is preferred over Lasso since Lasso may behave erratically when the number of features is greater than the number of training instances or when several features are strongly correlated

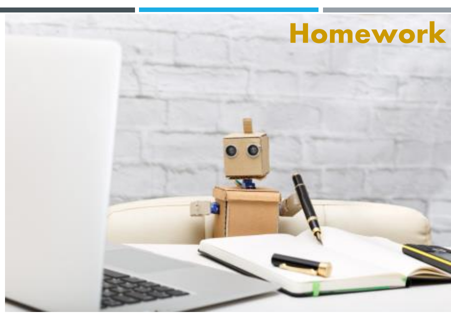
when should you use Linear Regression, Ridge, Lasso, or Elastic Net?

- It is almost always preferable to have at least a little bit of regularization, so avoid plain LR
- Ridge is a good default
- If you suspect that only a few features are actually useful, you should prefer Lasso or Elastic Net

17

18

Homework



18