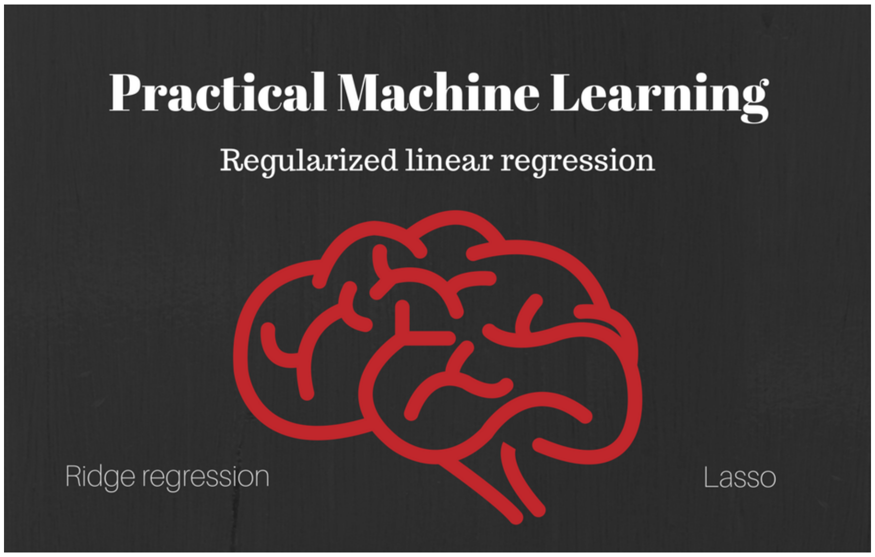
Images haven’t loaded yet. Please exit printing, wait for images to load, and try to print again.

Data hacker | entrepreneur | engineer | kitesurfer | https://codingstartups.com/
Aug 28 · 10 min read

1.

Practical machine learning: Ridge Regression vs. Lasso



For many years, programmers have tried to solve extremely complex computer science problems using traditional algorithms which are based on the most basic condition statement: if this then that. For example, if the email contains the word “free!” it should be classified as spam.

In recent years, with the rise of exceptional cloud computing technologies, the machine learning approach for solving complex problems has been magnificently accelerated. Machine learning is the science of providing computers the ability to learn and solve problems without being explicitly programmed. Sounds like a black magic? Maybe. In this post, I will introduce you to problems which can be solved using machine learning, as well as practical machine learning solutions for solving them.

Exactly like humans learn on a daily basis, in order to let a machine to learn, you need to provide it with enough data. Once it processed the data, it can make predictions about the future. Assuming you want to classify emails by whether they are spam emails or not. In order to solve this problem using machine learning, you need to provide the machine with many **labeled** emails—which are already classified in the correct classes of spam vs. not spam. The **classifier** will iterate over the samples and learn what are the **features** that define a spam email. Assuming you trained the machine learning model right, it will be able to **predict** whether a future email should be classified as spam or not, with high accuracy. In many cases, you'll not be able to completely understand how the model predicts the class.

Machine learning hierarchy

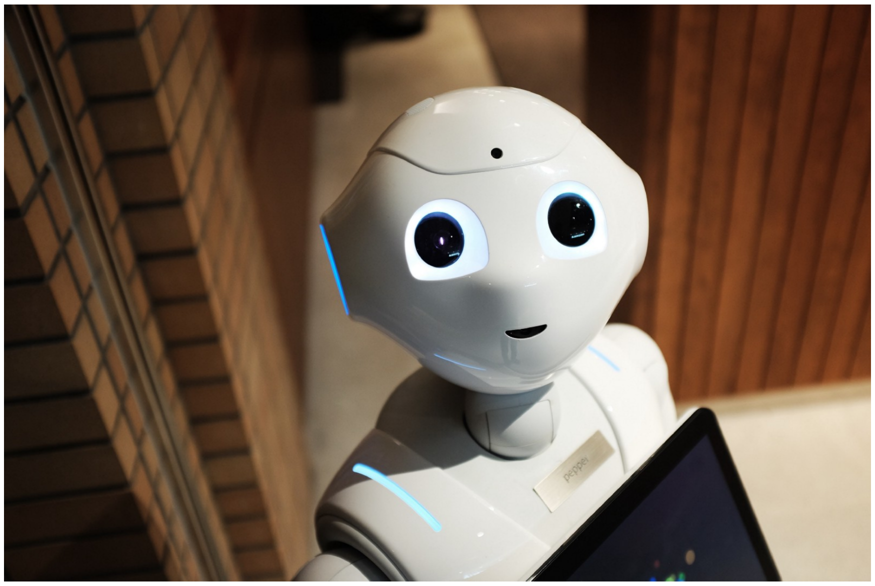
The world of machine learning can be divided into two types of problems: **supervised learning** and **unsupervised learning**. In this post, we will focus only on supervised learning, which is a subset of problems which contain **labeled** data (That is, every email is labeled as spam or not spam). For cases where you have unlabeled data, unsupervised learning might be a proper solution.

Underneath the supervised learning problems, there is another division of **regression** problems vs. **classification** problems. In regression problems, the value you wish to predict is **continuous**. For example, house price. In classification problems, on the other hand, the value you are about to predict is discrete, like spam vs. not spam.

The data you need to provide in order to train your model depends on the problem and the value you wish to predict. Let's assume you want to predict a house price based on different properties. So in this case, each row in your dataset should (for example) consist of:

> **features:** house size, the number of rooms, floor, whether elevator exists, etc.
>
> **label:** house price.

Choosing and collecting the features that best describe a house for predicting its price can be challenging. It requires market knowledge as well as access to big data sources. The features are the keys in which the prediction of the house price will be based upon.



Machine learning as an optimization problem

Every machine learning problem is basically an **optimization problem**. That is, you wish to find either a maximum or a minimum of a specific function. The function that you want to optimize is usually called the **loss function** (or **cost function**). The loss function is defined for each machine learning algorithm you use, and this is the main metric for evaluating the accuracy of your trained model.

For the house price prediction example, after the model is trained, we are able to predict new house prices based on their features. For each house price we predict, denoted as Ŷi, and the actual house price Yi we can calculate the loss by:

> l = (Ŷi- Yi)2

This is the most basic form of a loss for a specific data-point, That is used mostly for **linear regression** algorithms. The loss function as a whole can be denoted as:

> L = ∑(Ŷi- Yi)2

Which simply defines that our model's loss is the sum of distances between the house price we've predicted and the **ground truth**. This loss function, in particular, is called **quadratic loss** or **least squares**. We wish to **minimize** the loss function (L) as much as possible so the prediction will be as close as possible to the ground truth.

If you followed me up until now, you are familiar with the basic concept of every practical machine learning problem. **Remember,** every machine learning algorithm defines its own loss function according to its goal in life.

Linear regression

Linear regression is a basic yet super powerful machine learning algorithm. As you gain more and more experience with machine learning, you'll notice how **simple is better than complex** most of the time. Linear regression is widely used in different supervised machine learning problems, and as you may guessed already, it focuses on regression problem (the value we wish the predict is continuous). It is extremely important to have a good understanding of linear regression

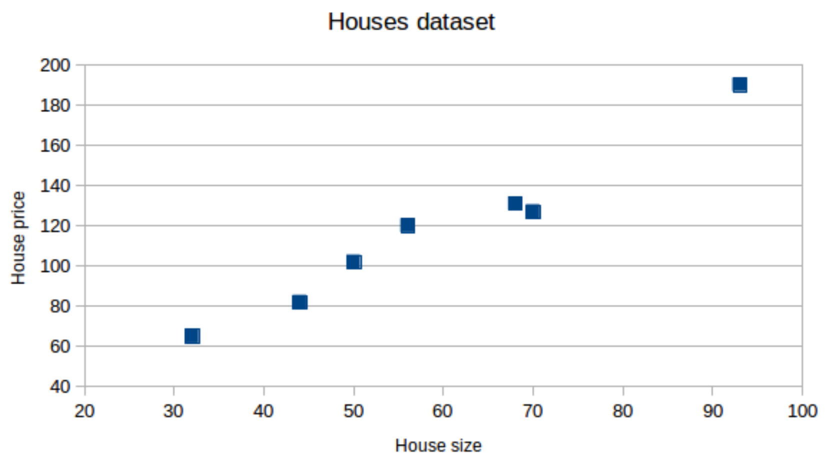
before studying more complex learning methods. Many extensions have been developed for linear regression which I will introduce later in this post.

The most basic form of linear regression deals with dataset of a single feature per data point (think of it as the house size). Because we are dealing with supervised learning, each row (house) in the dataset should include the price of the house (which is the value we wish the predict).

An example of our dataset:

House size (X)	House price (Y)
50	102
70	127
32	65
68	131
93	190
44	82
56	120

In a visual representation:



In linear regression we wish to fit a function (model) in this form:

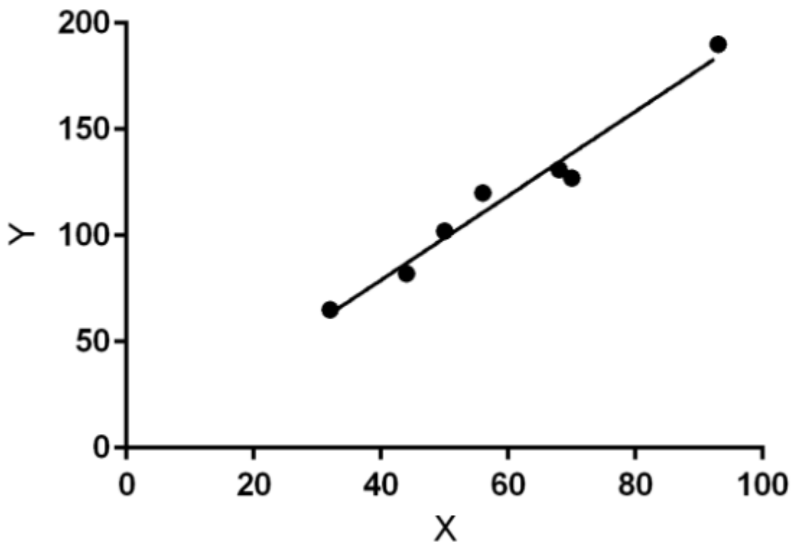
> $\hat{Y} = \beta0+\beta1X$

Where X is the vector of features (the first column in the table below), and β0, β1 are the coefficients we wish to **learn**.

By learning the parameters I mean executing an iterative process that updates β at every step by reducing the loss function as much as possible. Once we reach the minimum point of the loss function we can say that we completed the iterative process and learned the parameters.

Just to make it even more clear, the combination of the β coefficients are our trained model—which means that we have a solution to the problem!

After executing the iterative process, we can visualize the solution on the same graph:



Where the trained model is:

> $\hat{Y} = -0.5243+1.987X$

Now let's assume we want to predict based on our trained model, what will be the price of a house of size 85. In order to predict the price, we will substitute the β values we found into the model function, including the house size, and get the predicted house price:

> $\hat{Y} = 168.37$

To recap what we’ve covered so far:

> Every machine learning problem is basically an optimization problem. That is, we want to minimize (or maximize) some function.
>
> Our dataset is consist of features (X) and a label (Y). In our case—house size is the single feature, house price is the label.
>
> In linear regression problems, we want to minimize the quadratic loss which is the sum of distances between the predictions and the actual value (ground truth).
>
> In order to minimize the loss function and find the optimal β coefficients, we will execute an iterative process.
>
> To predict the label (house price) of a new house based on its size, we will use the trained model.

The iterative process for minimizing the loss function (a.k.a learning the coefficients β), will be discussed in another post. Although it can be done with one line of code, I highly recommend reading more about iterative algorithms for minimizing loss functions like **Gradient Descent**.

. . .

3. Linear regression with multiple features

In real world problems, you usually have more than one feature per row (house). Let’s see how linear regression can help us with multi-feature problems.

Considering this dataset:

House size (X_1)	rooms (X_2)	floor (X_3)	House price (Y)
50	2	5	123
70	2	3	118
32	1	3	62
68	3	7	148
93	4	10	250
44	2	6	100
56	3	1	110

So currently we have 3 features:

house size

number of rooms

floor

Therefore, we need to adapt our basic linear model to an extended one that can take into account the additional features for each house:

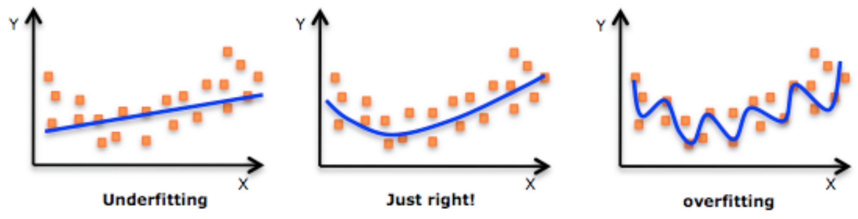
> $\hat{Y} = \beta 0+\beta 1X1+\beta 2X2+\beta 3X3$

In order to solve the multi-feature linear regression problem, we will the same iterative algorithm and minimize the loss function. The main difference will be that we will end up with four β coefficients instead of only two.

Overfit in machine learning algorithms

Having more features may seem like a perfect way for improving the accuracy of our trained model (reducing the loss) — because the model that will be trained will be more flexible and will take into account more parameters. On the other hand, we need to be extremely careful about **overfitting** the data. As we know, every dataset has noisy samples. For example, the house size wasn't measured accurately or the price is not up to date. The inaccuracies can lead to a low-quality model if not trained carefully. The model might end up memorizing the noise instead of learning the trend of the data.

A visual example of a nonlinear overfitted model:



Overfit can happen in linear models as well when dealing with multiple features. If not filtered and explored up front, some features can be more destructive than helpful, repeat information that already expressed by other features and add high noise to the dataset.

Overcoming overfit using regularization

Because overfit is an extremely common issue in many machine learning problems, there are different approaches to solving it. The main concept behind avoiding overfit is simplifying the models as much as possible. Simple models do not (usually) overfit. On the other hand, we need to pay attention the to gentle trade-off between overfitting and underfitting a model.

One of the most common mechanisms for avoiding overfit is called regularization. Regularized machine learning model, is a model that its loss function contains another element that should be minimized as well. Let's see an example:

> $L = \sum(\hat{Y}i - Yi)2 + \lambda\sum \beta 2$

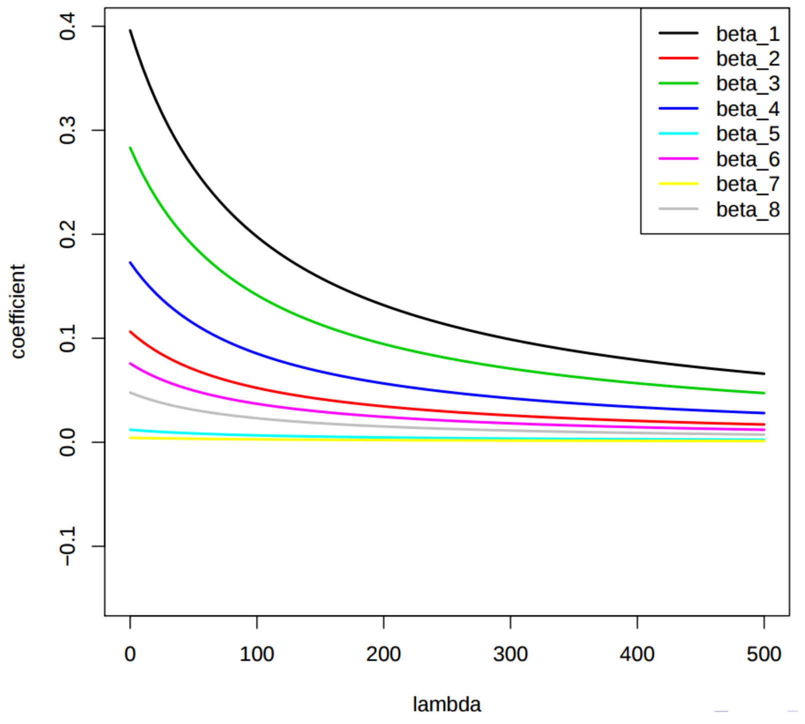
This loss function includes two elements. The first one is the one you've seen before—the sum of distances between each prediction and its ground truth. The second element though, a.k.a the regularization term, might seem a bit bizarre. It sums over squared β values and multiplies it by another parameter λ. The reason for doing that is to "punish" the loss function for high values of the coefficients β. As aforesaid, simple models are better than complex models and usually do not overfit. Therefore, we need to try and simplify the model as much as possible. Remember that our goal of the iterative process is to minimize the loss function. By punishing the β values we add a constraint to minimize them as much as possible.

There is a gentle trade-off between fitting the model, but not overfitting it. This approach is called **Ridge regression**.

Ridge regression

Ridge regression is an extension for linear regression. It's basically a regularized linear regression model. The λ parameter is a scalar that should be learned as well, using a method called **cross validation** that will be discussed in another post.

A super important fact we need to notice about ridge regression is that it enforces the β coefficients to be lower, but it **does not** enforce them to be zero. That is, it will not get rid of irrelevant features but rather **minimize their impact on the trained model**.



Lasso method

Lasso is another extension built on regularized linear regression, but with a small twist. The loss function of Lasso is in the form:

> L = ∑(Ŷi- Yi)2 + λ∑ |β|

The only difference from Ridge regression is that the regularization term is in absolute value. But this difference has a huge impact on the trade-off we've discussed before. Lasso method overcomes the disadvantage of Ridge regression by not only punishing high values of the coefficients β but actually setting them to zero if they are not relevant. Therefore, you might end up with fewer features included in the model than you started with, which is a huge advantage.

Conclusions

Machine learning is getting more and more practical and powerful. With zero knowledge in programming, you can train a model to predict house prices in no time.

We've covered the basics of machine learning, loss function, linear regression, ridge and lasso extensions.

There is more Math involved from what I've covered in this post, I tried to keep it as practical and, on the other hand, high-level as possible (Someone said trade-off?).

I encourage you to take a deep dive into this amazing world. Check out my blog for more posts to come—https://codingStartups.com—coders with entrepreneurial mindset.