CHAPTER 7

Regression

Now we will turn to a slightly different form of machine-learning problem, called *regression*. It is still supervised learning, so our data will still have the form

$$S_n = \left\{ \left(x^{(1)}, y^{(1)}\right), \dots, \left(x^{(n)}, y^{(n)}\right) \right\} \quad .$$

"Regression," in common parlance, means moving backwards. But this is forward progress!

But now, instead of the y values being discrete, they will be real-valued, and so our hypotheses will have the form

$$h : \mathbb{R}^d \rightarrow \mathbb{R} \quad .$$

This is a good framework when we want to predict a numerical quantity, like height, stock value, etc., rather than to divide the inputs into categories.

The first step is to pick a loss function, to describe how to evaluate the quality of the predictions our hypothesis is making, when compared to the "target" y values in the data set. The choice of loss function is part of modeling your domain. In the absence of additional information about a regression problem, we typically use *squared error* (SE):

$$\text{Loss}(\text{guess}, \text{actual}) = (\text{guess} - \text{actual})^2 \quad .$$

It penalizes guesses that are too high the same amount as it penalizes guesses that are too low, and has a good mathematical justification in the case that your data are generated from an underlying linear hypothesis, but with Gaussian-distributed noise added to the y values.

We will consider the case of a linear hypothesis class,

$$h(x; \theta, \theta_0) = \theta^T x + \theta_0 \quad ,$$

remembering that we can get a rich class of hypotheses by performing a non-linear feature transformation before doing the regression. So, $\theta^T x + \theta_0$ is a linear function of x, but $\theta^T \varphi(x) + \theta_0$ is a non-linear function of x if φ is a non-linear function of x.

We will treat regression as an optimization problem, in which, given a data set $\mathcal{D}$, we wish to find a linear hypothesis that minimizes mean squared error. Our objective, often called *mean squared error*, is to find values for $\Theta = (\theta, \theta_0)$ that minimize

$$J(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)} \right)^2 \quad ,$$

resulting in the solution:

$$\theta^*, \theta_0^* = \arg\min_{\theta,\theta_0} J(\theta, \theta_0) \ . \tag{7.1}$$

1 Analytical solution: ordinary least squares

One very interesting aspect of the problem finding a linear hypothesis that minimizes mean squared error (this general problem is often called *ordinary least squares* (OLS)) is that we can find a closed-form formula for the answer!

What does "closed form" mean? Generally, that it involves direct evaluation of a mathematical expression using a fixed number of "typical" operations (like arithmetic operations, trig functions, powers, etc.). So equation 7.1 is not in closed form, because it's not at all clear what operations one needs to perform to find the solution.

Everything is easier to deal with if we assume that the $x^{(i)}$ have been augmented with an extra input dimension (feature) that always has value 1, so we may ignore θ_0. (See chapter 3, section 2 for a reminder about this strategy).

We will approach this just like a minimization problem from calculus homework: take the derivative of J with respect to θ, set it to zero, and solve for θ. There is an additional step required, to check that the resulting θ is a minimum (rather than a maximum or an inflection point) but we won't work through that here. It is possible to approach this problem by:

- Finding $\partial J/\partial\theta_k$ for k in $1, \ldots, d$,
- Constructing a set of k equations of the form $\partial J/\partial\theta_k = 0$, and
- Solving the system for values of θ_k.

We will use d here for the total number of features in each $x^{(i)}$, including the added 1.

That works just fine. To get practice for applying techniques like this to more complex problems, we will work through a more compact (and cool!) matrix view.

Study Question: Work through this and check your answer against ours below.

We can think of our training data in terms of matrices X and Y, where each column of X is an example, and each "column" of Y is the corresponding target output value:

$$X = \begin{bmatrix} x_1^{(1)} & \ldots & x_1^{(n)} \\ \vdots & \ddots & \vdots \\ x_d^{(1)} & \ldots & x_d^{(n)} \end{bmatrix} \quad Y = \begin{bmatrix} y^{(1)} & \ldots & y^{(n)} \end{bmatrix} \ .$$

Study Question: What are the dimensions of X and Y?

In most textbooks, they think of an individual example $x^{(i)}$ as a row, rather than a column. So that we get an answer that will be recognizable to you, we are going to define a new matrix and vector, W and T, which are just transposes of our X and Y, and then work with them:

$$W = X^T = \begin{bmatrix} x_1^{(1)} & \ldots & x_d^{(1)} \\ \vdots & \ddots & \vdots \\ x_1^{(n)} & \ldots & x_d^{(n)} \end{bmatrix} \quad T = Y^T = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \ .$$

Study Question: What are the dimensions of W and T?

Now we can write

$$J(\theta) = \frac{1}{n}\underbrace{(W\theta - T)^T}_{1\times n}\underbrace{(W\theta - T)}_{n\times 1} = \frac{1}{n}\sum_{i=1}^{n}\left(\left(\sum_{j=1}^{d} W_{ij}\theta_j\right) - T_i\right)^2$$

and using facts about matrix/vector calculus, we get

$$\nabla_\theta J = \frac{2}{n} \underbrace{W^T}_{d \times n} \underbrace{(W\theta - T)}_{n \times 1} \quad .$$

Setting to 0 and solving, we get:

$$\begin{aligned}
\frac{2}{n} W^T (W\theta - T) &= 0 \\
W^T W\theta - W^T T &= 0 \\
W^T W\theta &= W^T T \\
\theta &= (W^T W)^{-1} W^T T
\end{aligned}$$

And the dimensions work out!

$$\theta = \underbrace{\left(W^T W\right)^{-1}}_{d \times d} \underbrace{W^T}_{d \times n} \underbrace{T}_{n \times 1}$$

So, given our data, we can directly compute the linear regression that minimizes mean squared error. That's pretty awesome!

2 Regularization

Well, actually, there are some kinds of trouble we can get into. What if $\left(W^T W\right)$ is not invertible?

Study Question: Consider, for example, a situation where the data-set is just the same point repeated twice: $x^{(1)} = x^{(2)} = (1, 2)^T$. What is W in this case? What is $W^T W$? What is $(W^T W)^{-1}$?

Another kind of problem is *overfitting*: we have formulated an objective that is just about fitting the data as well as possible, but as we discussed in the context of margin maximization, we might also want to *regularize* to keep the hypothesis from getting *too* attached to the data.

We address both the problem of not being able to invert $(W^T W)^{-1}$ and the problem of overfitting using a mechanism called *ridge regression*. We add a regularization term $\|\theta\|^2$ to the OLS objective, with trade-off parameter λ.

Study Question: When we add a regularizer of the form $\|\theta\|^2$, what is our most "preferred" value of θ, in the absence of any data?

Here is the ridge regression objective function:

$$J_{\text{ridge}}(\theta, \theta_0) = \frac{1}{n} \sum_{i=1}^{n} \left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)^2 + \lambda \|\theta\|^2$$

Larger λ values pressure θ values to be near zero. Note that we don't penalize θ_0; intuitively, θ_0 is what "floats" the regression surface to the right level for the data you have, and so you shouldn't make it harder to fit a data set where the y values tend to be around one million than one where they tend to be around one. The other parameters control the orientation of the regression surface, and we prefer it to have a not-too-crazy orientation.

There is an analytical expression for the θ, θ_0 values that minimize J_{ridge}, but it's a little bit more complicated to derive than the solution for OLS because θ_0 needs special treatment.

If we decide not to treat θ_0 specially (so we add a 1 feature to our input vectors), then we get:

$$\nabla_\theta J_{\text{ridge}} = \frac{2}{n} W^T(W\theta - T) + 2\lambda\theta \quad .$$

Setting to 0 and solving, we get:

$$\begin{aligned}
\frac{2}{n} W^T(W\theta - T) + 2\lambda\theta &= 0 \\
\frac{1}{n} W^T W\theta - \frac{1}{n} W^T T + \lambda\theta &= 0 \\
\frac{1}{n} W^T W\theta + \lambda\theta &= \frac{1}{n} W^T T \\
W^T W\theta + n\lambda\theta &= W^T T \\
(W^T W + n\lambda I)\theta &= W^T T \\
\theta &= (W^T W + n\lambda I)^{-1} W^T T
\end{aligned}$$

Whew! So,

$$\theta_{\text{ridge}} = \left(W^T W + n\lambda I\right)^{-1} W^T T$$

which becomes invertible when $\lambda > 0$.

This is called "ridge" regression because we are adding a "ridge" of λ values along the diagonal of the matrix before inverting it.

Study Question: Derive this version of the ridge regression solution.

Talking about regularization In machine learning in general, not just regression, it is useful to distinguish two ways in which a hypothesis $h \in \mathcal{H}$ might contribute to errors on test data. We have

Structural error: This is error that arises because there is no hypothesis $h \in \mathcal{H}$ that will perform well on the data, for example because the data was really generated by a sin wave but we are trying to fit it with a line.

Estimation error: This is error that arises because we do not have enough data (or the data are in some way unhelpful) to allow us to choose a good $h \in \mathcal{H}$.

When we increase λ, we tend to increase structural error but decrease estimation error, and vice versa.

There are technical definitions of these concepts that are studied in more advanced treatments of machine learning. Structural error is referred to as *bias* and estimation error is referred to as *variance*.

Study Question: Consider using a polynomial basis of order k as a feature transformation ϕ on your data. Would increasing k tend to increase or decrease structural error? What about estimation error?

3 Optimization via gradient descent

Inverting the $d \times d$ matrix $\left(W^T W\right)$ takes $O(d^3)$ time, which makes the analytic solution impractical for large d. If we have high-dimensional data, we can fall back on gradient descent.

Well, actually, Gauss-Jordan elimination, a popular algorithm, takes $O(d^3)$ arithmetic operations, but the bit complexity of the intermediate results can grow exponentially! There are other algorithms with polynomial bit complexity. (If this just made no sense to you, don't worry.)

Study Question: Why is having large n not as much of a computational problem as having large d?

Recall the ridge objective

$$J_{\text{ridge}}(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)^2 + \lambda\|\theta\|^2$$

and its gradient with respect to θ

$$\nabla_\theta J = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right)x^{(i)} + 2\lambda\theta$$

and partial derivative with respect to θ_0

$$\frac{\partial J}{\partial \theta_0} = \frac{2}{n}\sum_{i=1}^{n}\left(\theta^T x^{(i)} + \theta_0 - y^{(i)}\right) \quad .$$

Armed with these derivatives, we can do gradient descent, using the regular or stochastic gradient methods from chapter 6.

Even better, the objective functions for OLS and ridge regression are *convex*, which means they have only one minimum, which means, with a small enough step size, gradient descent is *guaranteed* to find the optimum.