Assignment 3: Linear Regression, Ridge Regression, Binary Classification

Machine Learning

Fall 2021

Learning Objectives

- Explore the limitations of linear regression.
- Derive and implement ridge regression.
- Learn about the framing of the classification problem in machine learning.

Prior Knowledge Utilized

- Linear regression including the sum of squares error loss function
- Supervised learning problem framing.
- Training / testing splits.

1 More Linear Regression and Ridge Regression

So far, you have experimented two types of supervised learning: classification and regression. You worked through the derivation of linear regression. We are going to start this assignment by applying linear regression to our bike share dataset. You will find and plot residuals, practice splitting your data into a training and testing set, and apply a fancy twist on linear regression called ridge regression.

External Resource(s) (120 minutes)

Assignment 3 Companion Notebook.

1.1 Ridge Regression Math

In the Companion Notebook, you manipulated the value of lambda (λ) to change the penalty for having large weights. One way to mitigate the problem of having two little data or having features that are linear combinations of each other is to modify the linear regression problem to prefer solutions that have small weights. We do this by penalizing the sum of the squares of the weights themselves. This is called ridge regression (or Tikhonov regularization). Below, we show the original version of ordinary least squares along with ridge regression.

Ordinary least squares:

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x_i} - y_i\right)^2 \tag{1}$$

$$= \arg\min_{\mathbf{w}} \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^\top \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) \tag{2}$$

Formula for the optimal weights in linear regression:

$$\mathbf{w}^\star = \left(\mathbf{X}^\top \mathbf{X}\right)^{-1} \mathbf{X}^\top \mathbf{y} \tag{3}$$

Ridge regression (note that λ is a non-negative parameter that controls how much the algorithm cares about fitting the data and how much it cares about having small weights):

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(\mathbf{w}^\top \mathbf{x_i} - y_i\right)^2 + \lambda \sum_{i=1}^{d} w_i^2 \tag{4}$$

$$= \arg\min_{\mathbf{w}} \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right)^\top \left(\mathbf{X}\mathbf{w} - \mathbf{y}\right) + \lambda \mathbf{w}^\top \mathbf{w} \tag{5}$$

The penalty term may seem a little arbitrary, but it can be motivated on a conceptual level pretty easily. The basic idea is that in the absence of sufficient training data to suggest otherwise, we should try to make the weights small. Small weights have the property that changes to the input result in minor changes to our predictions, which is a good default behavior.

Exercise 1 (60 minutes)

Derive an expression to compute the optimal weights, $\mathbf{w}^\star$, to the ridge regression problem.
Hint 1: This is very, very similar to Assignment 2, Exercise 5.
Hint 2: If you follow the same steps as you did in exercise 5, you'll arrive at an expression that looks like this (note: $\mathbf{I}_{d\times d}$ is the d by d identity matrix).

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} + \lambda \mathbf{w}^\top \mathbf{I}_{d\times d}\mathbf{w}$$

Hint 3: to get $\mathbf{w}^\star$, take the gradient, set it to 0 and solve for $\mathbf{w}$.

☆ Solution

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^\top \mathbf{w} \tag{6}$$

$$= \arg\min_{\mathbf{w}} \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} + \lambda \mathbf{w}^\top \mathbf{I}_{d\times d}\mathbf{w} \tag{7}$$

$$= \arg\min_{\mathbf{w}} \mathbf{w}^\top \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_{\mathbf{d}\times\mathbf{d}}\right) \mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} \tag{8}$$

$$2\left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_{\mathbf{d}\times\mathbf{d}}\right) \mathbf{w}^\star - 2\mathbf{X}^\top \mathbf{y} = 0 \quad \text{take the gradient and set to 0} \tag{9}$$

$$\mathbf{w}^\star = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_{\mathbf{d}\times\mathbf{d}}\right)^{-1} \mathbf{X}^\top \mathbf{y} \tag{10}$$

2 The Classification Problem

So far in this class we've looked at supervised learning problems where the responses y_i are continuous-valued and the loss function is quadratic ($\ell(y, \hat{y}) = (y - \hat{y})^2$). This is an example of a regression problem. There are many times, however, where it is unnatural to frame a problem as a regression. For instance, it may be the case that y_i does not come from a continuous range but instead can only take on a few different values. This sort of problem is known as a classification problem. For instance, you might want to have a system that takes in an image of a person and predicts their identity. The identity could be thought of as the output, y_i, and it would only make sense for y_i to be one of several values (e.g., each value might represent a particular person the system was trained to recognize). In this assignment you'll learn about a special case of the classification problem known as binary classification (where y_i is either 0 or 1, e.g., a Paul versus Sam recognizer).

In this assignment we will formalize the binary classification problem and see a very useful algorithm for solving it called *logistic regression*. You will also see that the logistic regression algorithm is a very natural extension of linear regression. Our plan for getting there is going to be pretty similar to what we did for linear regression.

- Build some mathematical foundations
- Introduce logistic regression from a top-down perspective
- Learn about logistic regression from a bottom-up perspective

3 Formalizing the Classification Problem

Let's start by making the binary classification problem more formal. Suppose, we are given a training set, $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)$, where each $\mathbf{x_i}$ is an element of the input space (e.g., a vector) and each y_i is a binary number (either 1 or 0). In this setting we will attempt to use the training data to determine a function, $\hat{f}^\star$, that predicts the corresponding output, y, for any possible input, $\mathbf{x}$. For example, $\mathbf{x}$ could be an image and y_i could be 1 when the picture contains a puppy and 0 otherwise.

Exercise 2 (10 minutes)

(a) Given this partial setup of the binary classification problem, we still need to specify the loss function, ℓ. Recall that ℓ takes as input the actual output y, and the predicted output $\hat{y}$. What function could you use for ℓ that would result in the learning algorithm choosing a good model? If the choice of ℓ depends on the application, how so?

☆ Solution

An easy choice is to output a 1 if the values don't match and a 0 otherwise (essentially counting the number of mistakes the model makes). Alternatively, you could have different penalties for a false positive (the model says $\hat{y} = 1$, but the actual value is $y = 0$) or false negatives (the model says $\hat{y} = 0$, but the actual value is $y = 1$).

(b) One natural choice for ℓ, which you may have already come up with, is to define our loss function as $\ell(y, \hat{y}) = \mathbb{I}[y \neq \hat{y}]$ (the funny looking $\mathbb{I}$ is the indicator function that takes on value 1 when the condition inside is true and 0 otherwise. Given this choice the supervised learning problem becomes:

$$\hat{f}^\star = \arg\min_{\hat{f}} \sum_{i=1}^{n} \mathbb{I}\left[\hat{f}(\mathbf{x_i}) \neq y_i\right] \quad . \tag{11}$$

Convert Equation 11 to English to make sure you understand it.

☆ Solution

The equation says that $\hat{f}^\star$ is the function that minimizes the number of mistakes it makes on the training set.

While the loss function given in Exercise 1(b) (minimizing mistakes on the training set) is a totally reasonable choice for the loss function, it turns out that it has a number of drawbacks.

- It is all or nothing. Either we are completely right or completely wrong.

- It is not a particularly easy function to work with mathematically. In fact, for many common classes of models, it will be difficult for the learning algorithm to find the best possible model[1].

It turns out that we can create a more natural loss function by thinking about predictions in terms of probabilities.

[1] One of the key challenges that must be met in machine learning, and modeling in general, is balancing computational considerations (e.g., how long does it take to find the best possible model) with the realism of the model (e.g., how directly does the task you pose to the learning algorithm match the problem you are solving). Sometimes these things are in conflict and you must make tradeoffs.

4 Probability and the log loss

Imagine that instead of our model, $\hat{f}$, spitting out either 0 or 1, it outputs a confidence that the input $\mathbf{x}$ has an output $y = 1$. In other words, rather than giving us its best guess (0 or 1), the classifier would indicate to us its degree of certainty regarding its prediction. This notion of "certainty" can be formalized using the concept of a probability.

We haven't formally defined probability in this class, and we won't do so here (we'll be working with probabilities extensively in module 2). Here are a few things to keep in mind about probabilities.

- A probability, p, specifies the chance that some event occurs. $p = 0$ means that the even will definitely not occur and $p = 1$ means that it will definitely occur.
- A probability, p, must be between 0 and 1 ($0 \leq p \leq 1$).
- If the probability an event occurs is p, then the probability that the event doesn't occur is $1 - p$.

✔ Understanding Check

Note: these sorts of boxes are here to help you test your understanding of a concept you have just read. We are trying to use these to breakup long blocks of reading. These need not be submitted and we won't ask about them on the Canvas quiz.

For these questions, assume that for a given input the classifier outputs a probability that the output will be 1.

(a) If a classifier has no clear idea of whether the output for a particular input is 1 or 0, what probability should the classifier output?

☆ Solution

The output would be about 0.5.

(b) If a classifier is relatively certain that the output for a particular input is 1, what probability should the classifier output?

☆ Solution

The output would be close to 1 (e.g., 0.99). The degree of closeness to 1 would depend on how certain the classifier was.

(c) If a classifier is relatively certain that the output for a particular input is 0, what probability should the classifier output?

☆ Solution

The output would be close to 0 (e.g., 0.01). The degree of closeness to 0 would depend on how certain the classifier was.

4.1 Log loss

If our model outputs a probability p when supplied with an input $\mathbf{x}$ (i.e., $\hat{f}(\mathbf{x}) = p$), we might then ask ourselves what loss function we should choose in order to select the best possible model? This loss function will be used to quantify how bad a prediction p is given the actual output y (recall that for binary classification the output is either 0 or 1). To make this more intuitive, consider the task of quantifying the quality of a weatherperson's predictions. Let's assume that on the ith day the weather is either sunny ($y_i = 1$) or rainy ($y_i = 0$). Suppose that each night the weatherperson gives the probability of it being sunny the next day. Here are two potential choices for quantifying the loss of each prediction compared to the outcome (the actual weather).

1. **0-1 loss:** we will extract from the weatherperson's prediction the most likely output (e.g., if $p = 0.75$, that would be sunny, if $p = 0.4$, that would be rainy). If the most likely output matches the actual output we give a loss of 0, otherwise we give a loss of 1 (this is similar to Equation 11).

2. **squared loss:** one downside of *0-1 loss* is that it doesn't take into account the certainty expressed by the weatherperson. The weatherperson gets the same loss if it is rainy and they predicted $p = 0.51$ or $p = 1$. For squared loss we compute the difference between the outcome and p and square it to arrive at the loss. For example if the weatherperson predicts $p = 0.51$ and it is sunny the loss is $(1 - 0.51)^2$. If it was rainy in this same example, the loss is $(0 - 0.51)^2$.

As an example, here are hypothetical predictions from two forecasters, the actual weather, and the resulting loss with either *0-1* loss or *squared loss*.

actual weather	forecast 1	0-1 loss	squared loss	forecast 2	0-1 loss	squared loss
sunny (y = 1)	$p = 0.2$	1	$(1-0.2)^2 = 0.64$	$p = 0.9$	0	$(1-0.9)^2 = 0.01$
rainy (y = 0)	$p = 0.6$	1	$(0-0.6)^2 = 0.36$	$p = 0.999$	1	$(0-0.999)^2 = 0.998$
sunny (y = 1)	$p = 0.8$	0	$(1-0.8)^2 = 0.16$	$p = 0.99$	0	$(1-0.99)^2 = 0.0001$
sum		2	1.16		1	1.01

✔ Understanding Check

According to the table above, which forecaster is better with regards to *0-1 loss*? Which forecaster is better with regards to *squared loss*?

☆ Solution

Forecaster 2 is better with respect to both loss functions (the losses are, on average, smaller).

One entry in the table above is particularly interesting. In the third row the second forecaster assigned a probability of 0.999 to it being sunny. It turned out to rain (boo!!!). The forecaster was almost certain it would be sunny and it wasn't. The 0-1 loss of course doesn't capture this at all. The squared loss seems to assign a fairly large loss. One might argue, though, that this loss does not fully capture how bad the prediction was (for one thing the loss can never be above 1). This last observation motivates a third loss function that we can use to evaluate probabilistic predictions: the log loss.

External Resource(s) (30 minutes)

`wiki.fast.ai` has some nice resources on a number of topics. They have a nice concise writeup that explains the concept of log loss. We ask that you read about log loss on NB so you can ask questions. If you want the original page (e.g., to click on the links), you can access the log loss page on wiki.fast.ai.

Exercise 3

Revisit the example from before with the two weather forecasters. Compute the log loss for each forecaster. Who makes better predictions according to the log loss?

☆ Solution

actual weather	forecast 1	log loss	forecast 2	log loss
sunny (y = 1)	$p = 0.2$	$-\ln 0.2$	$p = 0.9$	$-\ln 0.9$
rainy (y = 0)	$p = 0.6$	$-\ln 0.4$	$p = 0.999$	$-\ln 0.001$
sunny (y = 1)	$p = 0.8$	$-\ln 0.8$	$p = 0.99$	$-\ln 0.99$
sum		2.75		7.02

The first forecaster is better according to log loss.

5 A little more on confounding variables

We discussed confounding variables in class. Please watch this video on Simpson's Paradox.

Exercise 4 (15 minutes)

(a) Did you watch the video?

☆ Solution

Hopefully yes!

(b) Please come up with an example of Simpson's paradox that was not mentioned in the video. You don't have to know for sure that your example falls in to Simpson's paradox– a reasonable suspicion of a confounding variable is fine. Please describe the paradox and the confounding variable that you suspect. This is a simple question, but I realize it may be challenging to answer. If you are stuck, you might consider studies related to diet or health, the pandemic, or statistics related to elections. You may look up recent studies or articles to spark your imagination. I would discourage you from Googling "Simpson's paradox examples"– if you do, please adjust your response in the quiz accordingly.

☆ Solution

Simpson's paradox has to do with a "lurking" or "confounding" variable that isn't accounted for.