Assignment 3: Classification, Logistic Regression, and Gradient Descent

Machine Learning

Fall 2019

Learning Objectives

- Learn about the framing of the classification problem in machine learning.
- Learn about the logistic regression algorithm.
- Learn about gradient descent for optimization.

Prior Knowledge Utilized

- Supervised learning problem framing.
- Training / testing splits.

1 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

2 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 1 (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?

(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:

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

Convert Equation **??** to English to make sure you understand it.

While the loss function given in Exercise 1**??** (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.

3 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.

[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.

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?

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

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

3.1 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 **??**).

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*?

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 questionsl. 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 2

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?

4 Logistic Regression (top-down)

Now that we have built up some understanding of how probabilities can be used as a way of quantifying confidence in predictions, you are ready to learn about the logistic regression algorithm.

As always, we assume we are given a training set of inputs and outputs. As in linear regression we will assume that each of our inputs is a d-dimensional vector $\mathbf{x_i}$ and since we are dealing with binary classification, the outputs, y_i, will be binary numbers (indicating whether the input belongs to class 0 or 1). Our hypothesis functions, $\hat{f}$, output the probability that a given input has an output of 1. What's cool is that we can borrow a lot of what we did in the last couple of assignments when we learned about linear regression. In fact, all we're going to do in order to make sure that the output of $\hat{f}$ is between 0 and 1 is pass $\mathbf{w}^\top \mathbf{x}$ through a function that "squashes" its input so that it outputs a value between 0 and 1. This idea is shown graphically in Figure **??**.

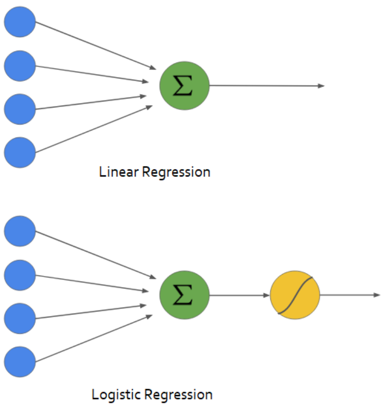

Figure 1: Graphical representation of both linear and logistic regression. The key difference is the application of the squashing function shown in yellow. Original source.

To make this intuition concrete, we define each $\hat{f}$ as having the following form (note: this equation looks daunting. We have some tips for interpreting it below).

$$\begin{aligned} \hat{f}(\mathbf{x}) &= \text{probability that output, } y\text{, is } 1 \\ &= \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}} \end{aligned} \tag{2}$$

Here are a few things to notice about this equation:

1. The weight vector that we saw in linear regression, $\mathbf{w}$, has made a comeback. We are using the dot product between $\mathbf{x}$ and $\mathbf{w}$ (which creates a weighted sum of the x_i's), just as we did in linear regression!

2. As indicated in Figure **??**, the dot product $\mathbf{w}^\top \mathbf{x}$ has been passed through a squashing function known as the sigmoid function. The graph of $\sigma(u) = \frac{1}{1+e^{-u}}$ is shown in Figure **??**. $\sigma(\mathbf{w}^\top \mathbf{x})$ is exactly what we have in Equation **??**.

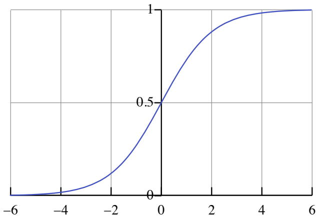

Figure 2: a graph of the sigmoid function $\frac{1}{1+e^{-x}}$.

4.1 Motivating Example: Sensor Networks

⚠ Notice

Pretty much every writeup of logistic regression contains a simple two-independent variable example. Usually these examples involve things like predicting who would be approved for a credit card or who will be admitted to a college. We are going to do something a little messier and bit different. If you want the admission to college example, you can read about it in Building a Logistic Regression in Python.

External Resource(s) (60 minutes)

Switch on over to the Assignment 3 Companion Notebook for a presentation of the motivating example.

5 *Putting it All Together: Deriving the Logistic Regression Learning Rule*

Let's summarize what we've done thus far in this assignment.

- We motivated the binary classification problem.
- We presented a particular useful loss function (log loss).
- We met the logistic regression model and tried it out on a real dataset.

Next, we're going to build on these pieces and formalize the logistic regression problem and derive a learning rule to solve it (i.e., compute the optimal weights). The formalization of logistic regression will combine Equation **??** with the selection of ℓ to be log loss. This choice of ℓ results in the following objective function.

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(-y_i \ln \sigma(\mathbf{w}^\top \mathbf{x_i}) - (1 - y_i) \ln(1 - \sigma(\mathbf{w}^\top \mathbf{x_i}))\right) \tag{3}$$

$$= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \left(-y_i \ln \left(\frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x_i}}}\right) - (1 - y_i) \ln \left(1 - \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x_i}}}\right)\right) \quad \text{expanded out if you prefer this form} \tag{4}$$

While this looks a bit crazy, since y_i is either 0 or 1, the multiplication of the expressions in the summation by either y_i or $1 - y_i$ are essentially acting like a switch—depending on the value of y_i we either get one term or the other. Our typical recipe for finding $\mathbf{w}^\star$ has been to take the gradient of the expression inside the $\arg\min$, set it to 0, and solve for $\mathbf{w}^\star$ (which will be a critical point and hopefully a minimum). The last two steps will be a bit different for reasons that will become clear soon, but we will need to find the gradient. We will focus on finding the gradient in the next couple of parts.

5.1 *Useful Properties of the Sigmoid Function*

Looking at Equation **??** it looks really, really hairy! We see that in order to compute the gradient we will have to compute the gradient of $\mathbf{x}^\top \mathbf{w}$ with respect to $\mathbf{w}$ (we just wrapped our minds around this last assignment). Additionally, we will have to take into account how the application of the sigmoid function and the log function changes this gradient. In this section we'll learn some properties for manipulating the sigmoid function and computing its derivative.

Exercise 3 (60 minutes)

The sigmoid function, σ, is defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \ . \tag{5}$$

(a) Show that $\sigma(-x) = 1 - \sigma(x)$.

(b) Show that the derivative of the logistic function $\frac{d}{dx}\sigma(x) = \sigma(x)(1 - \sigma(x))$

5.2 Chain Rule for Gradients

We now know how to take derivatives of each of the major pieces of Equation **??**. What we need is a way to put these derivatives together. You probably remember that in the case of single variable calculus you have just such a tool. This tool is known as the chain rule. The chain rule tells us how to compute the derivative of the composition of two single variable functions f and g.

$$\begin{aligned} h(x) &= g(f(x)) && \text{h(x) is the composition of } f \text{ with } g \\ h'(x) &= g'(f(x))f'(x) && \text{this is the chain rule!} \end{aligned} \tag{6}$$

Suppose that instead of the input being a scalar x, the input is now a vector, $\mathbf{w}$. In this case h takes a vector input and returns a scalar, f takes a vector input and returns a scalar, and g takes a scalar input and returns a scalar.

$$\begin{aligned} h(\mathbf{w}) &= g(f(\mathbf{w})) && \text{h}(\mathbf{w}) \text{ is the composition of } f \text{ with } g \\ \nabla h(\mathbf{w}) &= g'(f(\mathbf{w}))\nabla f(\mathbf{w}) && \text{this is the multivariable chain rule} \end{aligned} \tag{7}$$

Exercise 4 (60 minutes)

(a) Suppose $h(x) = \sin(x^2)$, compute $h'(x)$ (x is a scalar so you can apply the single-variable chain rule).

(b) Define $h(\mathbf{v}) = (\mathbf{c}^\top \mathbf{v})^2$. Compute $\nabla_{\mathbf{v}} h(\mathbf{v})$ (the gradient of the function with respect to $\mathbf{v}$).

(c) Compute the gradient of the expression from Equation **??** (reproduced below for your convenience).

$$\sum_{i=1}^{n} -y_i \ln \sigma(\mathbf{w}^\top \mathbf{x_i}) - (1-y_i) \ln \left(1 - \sigma(\mathbf{w}^\top \mathbf{x_i})\right) \quad . \tag{8}$$

You can either use the chain rule and the identities you learned about sigmoid, or expand everything out and work from that.

5.3 Gradient Descent for Optimization

If we were to follow our derivation of linear regression we would set our expression for the gradient to 0 and solve for $\mathbf{w}$. It turns out this equation will be difficult to solve due to the σ function. Instead, we can use an iterative approach where we start with some initial value for $\mathbf{w}$ (we'll call the initial value $\mathbf{w^0}$, where the superscript corresponds to the iteration number) and iteratively adjust it by moving down the gradient (the gradient represents the direction of fastest increase for our function, therefore, moving along the negative gradient is the direction where the loss is decreasing the fastest).

External Resource(s) (45 minutes)

There are tons of great resources that explain gradient descent with both math and compelling visuals.

- Recommended: Gradient descent, how neural networks learn | Deep learning, chapter 2 (start at 5:20)
- An Introduction to Gradient Descent (on NB, original)
- The Wikipedia page on Gradient Descent (on NB, original)
- Ahmet Sacan's video on gradient descent (this one has some extra stuff, but it's pretty clearly explained).
- There are quite a few resources out there, do you have some suggestions? (suggest so on NB)

Exercise 5 (10 minutes)

To test your understanding of these resources, here are a few diagnostic questions.

(a) When minimizing a function with gradient descent, which direction should you step along in order to arrive at the next value for your parameters?

(b) What is the learning rate and what role does it serve in gradient descent?

(c) How do you know when an optimization performed using gradient descent has converged?

(d) True or false: provided you tune the learning rate properly, gradient descent guarantees that you will find the global minimum of a function.

If we take the logic of gradient descent and apply it to the logistic regression problem, we arrive at the following learning rule. Given some initial weights $\mathbf{w^0}$, and a learning rate η, we can iteratively update our weights using the formula below.

$$\begin{aligned}
\mathbf{w^{n+1}} &= \mathbf{w^n} - \eta \sum_{i=1}^{n} -(y_i - \sigma(\mathbf{w}^\top \mathbf{x_i}))\mathbf{x_i} \quad \text{applying the result from exercise 4} && (9) \\
&= \mathbf{w^n} + \eta \sum_{i=1}^{n} (y_i - \sigma(\mathbf{w}^\top \mathbf{x_i}))\mathbf{x_i} \quad \text{distribute the negative} && (10)
\end{aligned}$$

This beautiful equation turns out to be the recipe we need to implement logistic regression.

Exercise 6 (10 minutes)

Now that you've had a second round of working with an in-class partner, we want to hear more about how it's going.

(a) Did you and your partner work well together? To help us understand how this better, provide a concrete examples to illustrate what went well and what didn't. If you want to avoid working with you partner in the future, say so (make sure to include the person's name). Only use this option in extenuating circumstances.

(b) We are assigning partners as an experiment. Our rationale for this choice is that finding a partner is potentially stressful and it may be difficult to work with folks you don't normally socialize with. With this rationale in mind, do you think we should continue to assign class work partners? Should we tweak this somehow?