

Module 4 - Linear Regression and Probability Estimation

- 4.1 An Introduction to Linear Regression
- 4.2 Linear Regression
- 4.3 Regularized Linear Regression
- 4.4 Linear Models for Conditional Probability Estimation
- 4.5 Logistic Regression
- 4.6 Logistic Regression in Use

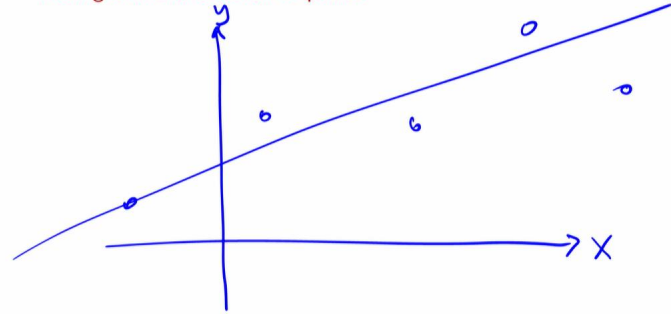
4.1 An Introduction to Linear Regression

Topics we'll cover

- 1 The regression problem in one dimension
- 2 Predictor and response variables
- 3 A loss function formulation
- 4 Deriving the optimal solution

Linear regression

Fitting a line to a bunch of points.

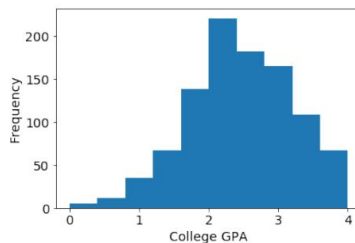


For the past few weeks we've been focusing on classification problems. What we'll do today is to switch gears a little bit and look at regression. Today, we'll look at a one-dimensional version of the problem as an introduction, and this will allow us to define some basic concepts like predictor and response variables. We'll see how this problem can be formulated as an optimization task, and how it can be solved using elementary calculus.

So in one dimension, regression is simply the business of fitting a line to a bunch of points. This is something we've all seen before. We have a x-axis and a y-axis. We have a set of points. And we wanna fit a line to them. There's no line that goes exactly through the points, and so we just want something that kind of goes through the middle of them somehow. How exactly do we do this? And why would we want to do this?

Example: college GPAs

Distribution of GPAs of students at a certain Ivy League university.

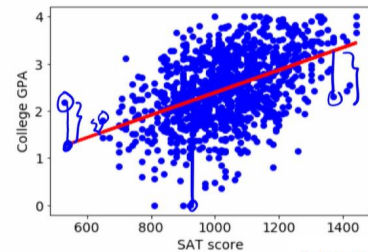


What GPA to predict for a random student from this group?

- Without further information, predict the **mean**, 2.47.
- What is the average squared error of this prediction?
That is, $\mathbb{E}[(\text{student's GPA}) - (\text{predicted GPA})]^2$?
The **variance** of the distribution, 0.55.

Better predictions with more information

We also have SAT scores of all students.



Mean squared error (MSE) drops to 0.43.

This is a **regression** problem with:

- Predictor variable:** SAT score x
- Response variable:** College GPA y

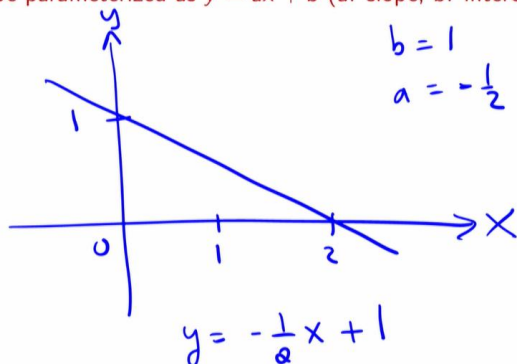
So let's look at a little example over here. A certain Ivy League university collected data on its entering freshman class. So at the end of freshman year, they noted down everybody's GPA. That's the histogram you see over here. So the GPA, the grade point average, is a number between zero and four. That's what's shown on the horizontal axis. And on the vertical axis is the frequency, the number of people who got that grade. In this case, it looks like the most frequent grade was somewhere over here. So something like 2.25, which is a C plus. So here's a question. Suppose a random student shows up and we want to predict his or her GPA. What score would we predict? Given this information, a reasonable thing to do would be to simply predict the mean of this distribution. That's easy enough to compute. If we do that, it turns out that in this case it is 2.47. So how good is this prediction? One way to sort of assess its quality is to say, "What is its average squared error?" So a student has shown up at random, okay? So this is their actual GPA. And we predicted this particular value, 2.47. If we look at the difference between these two and square it, what's the expected value of that? Well, if we look at this closely, this is exactly the formula for variance. It is the variance of this distribution. And in this case, the variance turns out to be .55.

Now, it turns out that at this university, they had also collected some other information. In addition to having everybody's freshman year GPA, they also had the SAT score from high school. This is a scatter plot of the two against each other. So on the horizontal axis are the SAT scores, and on the vertical axis are the college GPAs. You can see that the data is tilted slightly upwards, which indicates a positive correlation. So what we can do in this case is to see whether the SAT score helps us to predict the college GPA. So we go ahead and we fit a line like this. Now, we'll get to the business of exactly how we fit this line, but for the time being, let's just see how we would use it. Let's say a random student shows up, and we wanna predict his or her GPA. Well, now we find out what the SAT score is. And let's say it's 1200. The SAT score is 1200. We just go up to this line, we go across, and our prediction is three. We predict a GPA of 3. Now, can we use this additional piece of information to make predictions? It turns out that the mean squared error drops. It drops to about .43. So that additional piece of information was actually quite helpful. This is a classic regression problem. There's something we want to predict, which is called the **response variable**, that's the college GPA, and then there's information that helps us make this prediction. Those are the **predictor variables**, and in this case, there's just one predictor variable, the SAT score. So this is the x, the thing that helps us make predictions. And then there's the value we actually want to guess, which is the y, the college GPA. So what is the mean squared error though? Well, let's look at this data, okay? So look at this person over here. Using our line, the prediction we would make for that person would actually be down here. So there's a certain amount of error. There's this much error, and we can square that. Let's look at that squared error, okay? Let's look at this person over here. The squared error on that person is this distance squared. Let's look at this one over here. The squared error on that person is this distance squared, it's much smaller. Let's look at this person. The squared error on their squared error is actually pretty large. So if we take the average of all these squared errors, that is the **mean squared error**.

总结:

Parametrizing a line

A line can be parameterized as $y = ax + b$ (a : slope, b : intercept).



So, how do we go about fitting a line of this form? Well, the first thing is to figure out how you're going to parametrize a line. And one convenient way of doing this is to write down a line as y equals ax plus b , where a is the slope of the line and b is the y -intercept. For example, let's say that we have our x -axis and our y -axis, and we have a line like this. Maybe this is two and this is one. So the y -intercept is where the line crosses the y -axis, that's one. And the slope in this case, well it's sloped downwards, so it's gonna be negative. And the slope in this case is negative $1/2$. And so the equation of this line is y equals negative $1/2 x$ plus one. So we would parametrize a line using these two numbers, a and b . Okay.

The line fitting problem

Pick a line (parameters a, b) suited to the data, $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R} \times \mathbb{R}$

- $x^{(i)}, y^{(i)}$ are predictor and response variables, e.g. SAT score, GPA of i th student.
- Minimize the mean squared error,

$$\text{MSE}(a, b) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - (ax^{(i)} + b))^2.$$

This is the **loss function**.

So this lets us define the problem precisely. We're given a set of data points. These are x, y pairs. x and y are both real numbers. The x are the predictor variables, in our case, the SAT scores. The Y s are the things we wanna guess, the response variables, in our case, the GPAs. So we have n data points of this form. We wanna find a line. In other words, we wanna find parameters a and b that define a line. And what we want is to find the line that incurs the least mean squared error. So what is that? That's what's given by this formula over here. If we look at the i th data point, x sub i , then using this line a, b , our prediction on x sub i would be this. That would be the value we would guess. The correct value is y sub i . So the squared error incurred on the i th data point is this term over here. Now we take that and we average over all the data points. This is the **mean squared error**. And this is the **loss function** that we are trying to minimize. There's something interesting going on over here. We are taking a learning task and formulating it as an optimization problem, as the problem of finding parameters that minimize some kind of loss function. This optimization approach has proved to be extremely fruitful in machine learning, and we'll be seeing a whole lot more of it.

Minimizing the loss function

Given $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})$, minimize

$$\frac{d}{db} u^2 = 2u \frac{du}{db}$$

$$L(a, b) = \sum_{i=1}^n \underbrace{(y^{(i)} - (ax^{(i)} + b))}_u^2.$$

To minimize, set $\frac{dL}{da} = \frac{dL}{db} = 0$

$$\frac{dL}{db} = \sum_{i=1}^n 2(y^{(i)} - (ax^{(i)} + b)) \cdot (-1) = 0$$

$$\Rightarrow \sum_i y^{(i)} = a \sum_i x^{(i)} + nb$$

$$\Rightarrow b = \left[\frac{1}{n} \sum_i y^{(i)} \right] - a \cdot \left[\frac{1}{n} \sum_i x^{(i)} \right]$$

$$\bar{x} = \frac{1}{n} \sum_{i=1}^n x^{(i)}, \quad \bar{y} = \frac{1}{n} \sum_{i=1}^n y^{(i)}$$

$$b = \bar{y} - a\bar{x}$$

$$\frac{dL}{da} = 0 \Rightarrow a = \frac{\sum_{i=1}^n (y^{(i)} - \bar{y})(x^{(i)} - \bar{x})}{\sum_{i=1}^n (x^{(i)} - \bar{x})^2}$$

So let's go ahead and see how we can minimize this loss function. So we wanna minimize this function L which has two parameters, a and b . Well, we can just take the derivative and set it to zero. Usual calculus way. So since the two parameters, we need to take the derivative with respect to each of them. So to minimize, what we will do is we'll simply set the derivative with respect to a and the derivative with respect to b to be zero. Okay, so let's go ahead and compute these derivatives.

- I think the one with respect to b is gonna be a little bit simpler, so let's start with that. What is the derivative of L with respect to b ? Okay, so L is a big summation. The derivative of a sum is the sum of the derivatives. So we'll start by writing that down. Now, how do we take a derivative of that? We have the squared term over there. Okay, so there's this general rule which says we're taking the derivative with respect to b , "If you're taking the derivative of something squared, u squared, that turns out to be $2u \cdot du$." So it's $2u$ times the derivative of u with respect to b . And for us, u is this thing over here. That is the y_i , we wanna take the derivative of that. So let's write down $2u$ first of all. So $2u$, we just copy down u . Okay. And now we take the derivative of u with respect to b , which is actually just negative one, okay? So times negative one. Good, so that's the entire derivative, and we wanna set that to zero. Now, since we're setting it to zero, we can just divide out the negative one and we can divide out the two. And let's see what we get. So we get the summation of the y_i s equals a times the summation of the x_i s plus n times b . Because we're summing b from i equals one to n , so that's n times b . And let's go ahead and solve for b . So that tells us that b is equal to, we can divide both sides by n , one over n times the summation of the y_i s minus a times one over n times the summation of the x_i s. That's the optimal setting for b . So what are these things over here? What is this term over here? Well, that's just the average y value, the average response value. And what is this thing over there? Well, that's just the average x value. In our case case, that would average SAT score. Okay, so let's write that down in a slightly more compact form. Okay, so we'll just remember this equation. And what we said is let's look at the average x value. I'm just gonna call that \bar{x} , okay? So we take the average of all the x s. And let's look at the average of all the y s. And it turned out that the optimal setting for b is the average y value minus a times the average x value. And this makes perfect sense. We want y to be equal to a times x plus b . Now, of course, we aren't gonna be able to get that exactly, but it makes perfect sense that b would be the average y value minus a times the average x value.
- Okay, so we still have to solve for a . And the way we do that is we set the derivative of L with respect to a to zero. And then we go through the calculation. And this time the algebra is a little bit more involved, but I can just tell you what the final answer is gonna be. If you like you can try out the steps at home. It's just a little bit messier. And it turns out that what the optimal setting for a is is simply the covariance of x and y divided by the variance of x . So it's something of this form. We take all the y s, we subtract off the average y , and you look at the covariance with x . And then you divide by the x minus the average x squared. So that's the optimal setting for a . Very simple, a nice closed-form solution.

Well, that concludes our little introduction to regression. Today, we just talked about the one-dimensional setting where there's a single predictor variable. What we'll do next time is to generalize this methodology to multiple predictor variables. See you then.

POLL

Given the line $y = -2x + 8$, and the points $a = (1,2)$ and $b = (3,4)$, which point has the smallest squared error from the line?

$x=1, y=6$
 $\Delta y = 6-2$

$x=3, y=2$
 $\Delta y = 2-4$

RESULTS

- | | |
|--|-----|
| <input type="radio"/> point a | 50% |
| <input checked="" type="radio"/> point b | 50% |
| <input type="radio"/> both have the same squared error | 0% |

FEEDBACK

point b

4.2 Linear Regression

Topics we'll cover

- 1 Regression with multiple predictor variables
- 2 Least-squares regression
- 3 The least-squares solution

Last time, we got a little taste of linear regression, by looking at the case of one-dimensional data. Today, we'll move on to full-fledged least-squares regression. This is a very simple and powerful method, and one of the cornerstones of statistics.

So we'll begin by formulating what it means to do regression with multiple predictor variables. We will phrase the learning problem as an optimization task, with an explicit loss function. And then we'll see that this optimization task, is actually quite easy to solve.

Diabetes study

Data from $n = 442$ diabetes patients.

For each patient:

- 10 features $x = (x_1, \dots, x_{10})$
age, sex, body mass index, average blood pressure, and six blood serum measurements.
- A real value y : the progression of the disease a year later.

Regression problem:

- **response** $y \in \mathbb{R}$
- **predictor variables** $x \in \mathbb{R}^{10}$

So, as our running example, we will use data from a diabetes study. So the goal of this study was to assess what features might impact the progression of the disease, how much worse it gets over the course of a fixed period of time, for example. So data was collected from 442 diabetes patients. For each patient, 10 features were measured, features that could plausibly impact the progression of the disease. They were age, sex, body mass index, which is a measure of how overweight somebody is, average blood pressure, and various blood serum measurements. So these 10 features were measured for each of the patients, and then, the patient came back a year later, and a value was measured which indicated how much the disease had progressed over the course of the year. And that value was called y . This is a classic regression problem. We have a response value y , we want to be able to predict this response in terms of a bunch of predictor variables, and we have 10 of these, x_1 through x_{10} , which we can package together into a 10-dimensional vector. We'd like to model y as a linear function of the x 's. Let's see what this means.

Least-squares regression

Linear function of 10 variables: for $x \in \mathbb{R}^{10}$,

$$f(x) = w_1x_1 + w_2x_2 + \dots + w_{10}x_{10} + b = w \cdot x + b$$

where $w = (w_1, w_2, \dots, w_{10})$.

Penalize error using squared loss $(y - (w \cdot x + b))^2$.

Least-squares regression:

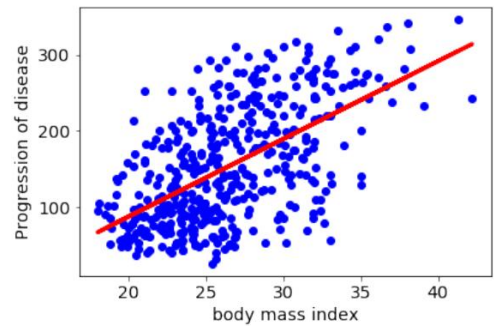
- *Given:* data $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \mathbb{R}$
- *Return:* linear function given by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$
- *Goal:* minimize the **loss function**

$$L(w, b) = \sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2$$

So when we have just one variable, x , let's say, a linear function of the x s, a linear function of x would be something like this. y equals $w x$ plus b , where w is the slope, and b is the intercept. When we have 10 variables, a linear function looks like this. We have $w_1 x_1$ plus $w_2 x_2$, all the way to $w_{10} x_{10}$, and then again, we have the intercept term at the end. Now we can write this a little bit more simply, as $w \cdot x$ plus b , where what we've done is to package together the coefficients w_1 through w_{10} , into a single vector w . So this is a nice compact form of the linear function. So that's the sort of model that we're gonna fit to the data. Now, on any given point x , our prediction is gonna be $w \cdot x$ plus b , and the correct value is y . So the error, we have to figure out some way to penalize it, and we're gonna use squared loss, so the squared difference between the value we predict, and the actual value. And this leads immediately to the least-squares regression problem. So we have end data points, in our case there are 442 patients, so n equals 442. The points x_1 through x_{10} are d -dimensional vectors, for us there are 10 features, so d equals 10, and then we also have the response values y_1 through y_n . Now based on this data, we wanna learn a linear function. In other words, we want to learn parameters w and b . And we wanna pick the linear function that minimizes the total squared error, so that minimizes the sum over all the data points, of the squared error on the i th data point. So a very simple loss function. We wanna find the vector w , and the value b that minimize this loss function. Now it will turn out that there is a nice closed form for the optimal w and b , and we'll see that very soon. But first, let's just see what happens on the diabetes data.

Back to the diabetes data

- No predictor variables: mean squared error (MSE) = 5930
- One predictor ('bmi'): MSE = 3890
- Two predictors ('bmi', 'serum5'): MSE = 3205
- All ten predictors: MSE = 2860



So suppose we had no predictor variables at all, we just had to predict y , what value of y would we predict? Well we'd predict the average value of y , the mean value. And in this case, the mean squared error would simply be the variance in y . So in the diabetes example, this variance turns out to be 5930. Now this might seem a little large, but actually it turns out these y values, the progression of the disease, are numbers like 100, 200, 300. So when y is 200, y squared is 40,000. So relative to that, a variance of about 6,000 seems plausible. Now let's say we throw in one predictor feature, let's say we use body mass index for example. So what I've shown over here is a scatter plot with body mass index on the x axis, and the y value, the progression of the disease on the y axis. So, this is a scatter plot of the 442 data points, and I've also shown the line that is obtained by least-squares regression. By using this line, the mean squared error drops significantly, down to 3,890. Then we can throw in a second predictor variable, let's say we use one of the serum measurements. The mean squared error drops further. And then we can go ahead and use all 10 variables, to make the mean squared error the smallest of all, 2860. Now computationally, all of these different calculations are very simple, because there's a nice closed formula for w . And in fact, this is one of the few cases in this entire course where there's such a simple solution for the optimization tasks that we are dealing with. So, so let's see how it works out.

Least-squares solution 1

Linear function of d variables given by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$:

$$f(x) = w_1 x_1 + w_2 x_2 + \dots + w_d x_d + b = w \cdot x + b$$

Assimilate the intercept b into w :

- Add a new feature that is identically 1: let $\tilde{x} = (1, x) \in \mathbb{R}^{d+1}$

$$\begin{pmatrix} 4 & 0 & 2 & \dots & 3 \end{pmatrix} \Rightarrow \begin{pmatrix} 1 & 4 & 0 & 2 & \dots & 3 \end{pmatrix}$$

- Set $\tilde{w} = (b, w) \in \mathbb{R}^{d+1}$
- Then $f(x) = w \cdot x + b = \tilde{w} \cdot \tilde{x} = (b, w) \cdot (1, x) = b + w \cdot x$

Goal: find $\tilde{w} \in \mathbb{R}^{d+1}$ that minimizes

$$L(\tilde{w}) = \sum_{i=1}^n (y^{(i)} - \tilde{w} \cdot \tilde{x}^{(i)})^2$$



So once again, we are fitting a linear function to the data, the linear function is something of this form, if we have d features. Now, this term over here, the intercept term, is important, but it sticks out a little bit, it's a little bit inconvenient because it looks a little different from everything else. And when we're doing the optimization, for example, we have to treat it separately as a result. Luckily it turns out that there's a way to make it disappear, basically by assimilating it into w . So let's see how we do this. So here's the trick for assimilating b into the w vector. The first thing we do, is to add an extra feature to the data x , by just sticking a one in front of all the data points. So if, for example, we have a data point that looks like this, we write down the same thing, but with a one in front of it. So each point x now becomes a new point, which we'll call x twiddle, which is just one followed by whatever x was before. So these new points, x twiddle, are now d plus one dimensional. The next thing we do, is to stick b and w together, and we call that vector w twiddle. So now let's see what happens to our linear function, so our linear function was $w \cdot x$ plus b , but this is exactly the same as w twiddle dot x twiddle. Why, because w twiddle is become a w , and x twiddle is one comma x so it's exactly b plus $w \cdot x$. So we've managed to rewrite the linear function without b , by adding an extra feature to the data points, and also to the w vector. So now, we need only optimize for w twiddle, and here's the new loss function. We wanna find the w twiddle that leads, once again, to the least-squared error on the data set. And as we'll see, there's a nice formula for w twiddle. So the first step in deriving the formula will actually be to re-write this loss function, purely as a matrix vector product. Why would we do this? Well it turns out that it leads to a nice, simple expression for w twiddle.

Least-squares solution 2

Write

$$X = \begin{pmatrix} \leftarrow \tilde{x}^{(1)} \rightarrow \\ \leftarrow \tilde{x}^{(2)} \rightarrow \\ \vdots \\ \leftarrow \tilde{x}^{(n)} \rightarrow \end{pmatrix}_{n \times (d+1)}, \quad y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{pmatrix}_{n \times 1}$$

$$X\tilde{w} = \begin{pmatrix} \tilde{w} \cdot \tilde{x}^{(1)} \\ \tilde{w} \cdot \tilde{x}^{(2)} \\ \vdots \\ \tilde{w} \cdot \tilde{x}^{(n)} \end{pmatrix}$$

$$y - X\tilde{w} = \begin{pmatrix} y^{(1)} - \tilde{w} \cdot \tilde{x}^{(1)} \\ \vdots \\ y^{(n)} - \tilde{w} \cdot \tilde{x}^{(n)} \end{pmatrix}$$

Then the loss function is

$$L(\tilde{w}) = \sum_{i=1}^n (y^{(i)} - \tilde{w} \cdot \tilde{x}^{(i)})^2 = \|y - X\tilde{w}\|^2$$

and it minimized at $\tilde{w} = (X^T X)^{-1} (X^T y)$.

$\xleftarrow{(d+1) \times 1}$



And so let's see what happens. This is our loss function. We're gonna rewrite it in terms of matrices and vectors, and actually this is something that is often very useful to do. Part of the reason for this is that standard platforms like Python have highly optimized routines for matrix math, for matrix and vector multiplies. At the same time, they can be extremely slow on things like for loops. So if you can somehow take your for loop, and rewrite it as a matrix vector product, or as a matrix matrix product, your code could potentially speed up dramatically. But how do we rewrite things in this way, well, let's take a look at this loss function, so we have a loss function that contains a summation, now normally if we have a summation, that would make us think of a for loop. But we can rewrite it, and let's see how.

- So, first, we create a matrix with one data point per row, so we have n rows, and, of course we have augmented our data, now we've added this extra feature, we've stuck ones in the front of each vector. So, we have d plus one columns. So that's the matrix x . And let's create a vector with all the response values, so y is n by one. Now, let's see what is x times w twiddle. Well, it's the first row of x dot product, it would w twiddle, the second row of x dot product, it would w twiddle and so on, it's exactly the predicted values. So this is exactly w twiddle dot x_1 , w twiddle dot x_2 , and so on, all the way to w twiddle dot x_n . And likewise, if we look at y minus xw twiddle, what does that work out to? It's simply the vector of errors. It's the error on the first point, y_1 minus w twiddle dot x_1 , all the way to the error on the last point, y_n minus w twiddle dot x_n . So in particular, the squared norm of this vector, which is this quantity over here, is exactly the sum of all the squared errors, it's exactly our loss function. So we've rewritten our loss function, without any summation at all. Purely in terms of vectors and matrices.
- At this point, we can take the derivative, and set it to zero, and it turns out that the solution is the following. We said w twiddle to x transpose x inverse, that's a d plus one by d plus one matrix, times x transpose y , which is a d plus one times y vector. And, the result is d plus one times one. And of course the very first term in that answer is the intercept term that we were previously calling b .

So it's a very simple solution, and that concludes our treatment of least-squares regression. It's a very easy method, and it's a very powerful method as well, that's useful in a wide range of applications. This is the bread and butter of much of statistics. Okay, see you next time.

总结:

POLL

In order to assimilate b into w , what must we do to X ?

RESULTS

- | | | |
|----------------------------------|---|------|
| <input type="radio"/> | X becomes X^T | 0% |
| <input checked="" type="radio"/> | Add a feature to X which has a value of 1 | 100% |
| <input type="radio"/> | Add a feature to X which has a value of b | 0% |
| <input type="radio"/> | No changes must be made to X | 0% |

FEEDBACK

Add a feature to X which has a value of 1

4.3 Regularized Linear Regression

Topics we'll cover

- 1 Generalization
- 2 Regularization
- 3 Ridge regression
- 4 Lasso

Today we will look at some popular variant of least squares regression. So we'll begin by considering questions of generalization. Does doing well on the training set mean that you will necessarily do well on future test data? And if not, are there loss functions that aren't exclusively geared to one's performance on the training set?

This will lead to the popular notion of regularization. And we'll see two regularized forms of least squares regression that are called ridge regression and the lasso.

Least-squares regression

Given a **training set** $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \mathbb{R}$, find a linear function, given by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$, that minimizes the squared loss

$$L(w, b) = \sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2.$$

Is training loss a good estimate of **future** performance?

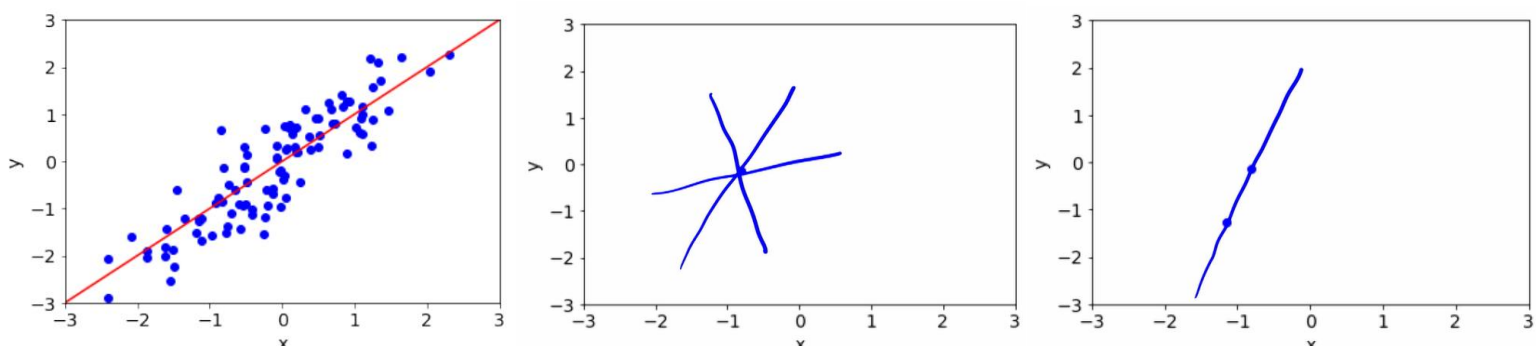
- If n is large enough: maybe.
- Otherwise: probably an underestimate.

Okay, so least squares regression involves finding a linear function that minimizes the squared loss on the training set. Does this mean good performance in terms of low squares loss on future data as well? Well, not necessarily. So, there are two regimes that are useful to consider.

- If n is large, that is if you have a lot of training data, then good performance on the training set **might reasonably** indicate fairly good performance in future as well.
- But if n is small, then performance on the training set is **probably a gross underestimate of future error**.

And let's see this by means of concrete example.

Example



总结:

So, here's a data set in which there's just one feature x and a response value y on the y axis. If we were to fit a line to this data, it would look something like this. And in this case, the line is just given by two parameters. It's just a one dimensional setting. So we have two parameters, and we have something like 100 data points. So this is a generous amount of data given how few parameters we need to fit. And in this case it's quite plausible that the training error is a reasonable estimate of the kinds of error rates that will appear on future data as well.

But suppose we didn't have this much training data. Suppose that instead of these 100 points, we just have, say, one of them. What line would we fit to this one data point? Well, there are lots of possibilities. We could fit this line or this line or this line. And they all have zero training error. Does this mean zero error on future test points? No, certainly not. So that's an extreme case.

Let's say that instead of one training point, we had two training points. In this case, the line we would fit would be something like this. Again, this is zero training error. How would this line do on future test points? Well, let's go back and see what the distribution looked like. This is the overall distribution of the data. This line would perform very poorly on it.

Better error estimates

Recall: k -fold cross-validation

- Divide the data set into k equal-sized groups S_1, \dots, S_k
- For $i = 1$ to k :
 - Train a regressor on all data except S_i
 - Let E_i be its error on S_i
- Error estimate: average of E_1, \dots, E_k

A nagging question:

When n is small, should we be minimizing the squared loss?

$$L(w, b) = \sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2$$

So all of this raises some big questions.

- The first big question is, **if training error is not a good measure of future test error, how do we get a good estimate of future error? And the usual way to do this, one simple way of doing this is by cross validation**. Now, we talked about this when we were doing nearest neighbor, but let me just quickly go over it again since it's really a very important technique. So in k -fold cross validation, what we would do is to take our training set and divide it into k chunks of equal size. So if we have n points, we divide them into k groups of n over k points. Then we take the first chunk and think of it as a test set. We would train on the remaining k minus one chunks. That would yield some linear function. And then we would evaluate the function on chunk number one. This would give us some error rate. Then we think of chunk two as the test set. And we train on the remaining k minus one chunks. We get some linear function and evaluate it on chunk two. And so on, we then think of chunk three as the test set. **So in this way, we get k different error estimates, E_1 through E_k . And we just return their average. And this is a much more reasonable estimate of future error**. Okay, so that resolves that question.
- The second question is, **if it is the case that training error is really not a very good measure of future error, then why are we using an optimization criterion that's based exclusively on training error? Is there something else we could be optimizing? Something that would likely lead to better future performance? And indeed there are other optimization criteria of this type**. So let's look at an example.

Ridge regression

Minimize squared loss **plus** a term that penalizes “complex” w :

$$L(w, b) = \underbrace{\sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2}_{\text{squared loss}} + \underbrace{\lambda \|w\|^2}_{\text{penalty term}}$$

Adding a penalty term like this is called **regularization**.

$$\lambda = 0$$

$w = \text{least-squares solution}$

$$\lambda \rightarrow \infty$$

$$w = 0$$

Put predictor vectors in matrix X and responses in vector y :

$$w = (X^T X + \lambda I)^{-1} (X^T y)$$

In ridge regression, the goal is, as before, to learn a linear function. So given by w and b . But the loss function is a little bit different. So there is the least squares loss as before. But there's also an additional term called a **regularizer**. In this case, it's just **the squared L2 norm of w times the constant λ** . So what is the effect of this? What is this regularizer going to do? Let's look at two extremes.

- Suppose we set λ equal to zero. In this case, the second term just falls away. It doesn't matter, and we just get regular least squares regression. So in this case w would be the least squares solution. **This is a reasonable solution if we have a lot of data.**
- Now let's say we set λ to be something large. In fact let's let λ go to infinity. What happens then? Well, if λ goes to infinity, the only thing that matters is the second term. And so we will at all costs make w as small as possible. We will try to shrink its norm as much as we can. And in fact we'll just set w to zero. **This is a reasonable solution if we have no data at all.**

Now, in practice what we'll be doing is to choose some value of λ that's between these two extremes. And so what it will do is to do something like least squares but to move that solution closer to zero, to shrink it towards zero, in a sense. And for this reason, it is sometimes called a **shrinkage estimator**.

In fact, for ridge regression, we have a nice closed form equation for the optimal w . Let me show you what it is. This is the formula for w . Now, if we set λ to zero, the λ times identity term drops out. And we just get the least squares solution that we saw last time. But as we grow λ , this first term over here becomes larger. And so the inverse of it puts it in the denominator. It means that we have a larger and larger denominator. And so we shrink the resulting w .

Toy example

Training, test sets of 100 points

- $x \in \mathbb{R}^{100}$, each feature x_i is Gaussian $N(0, 1)$
- $y = x_1 + \dots + x_{10} + N(0, 1)$

λ	training MSE	test MSE
0.00001	0.00	585.81
0.0001	0.00	564.28
0.001	0.00	404.08
0.01	0.01	83.48
0.1	0.03	19.26
1.0	0.07	7.02
10.0	0.35	2.84
100.0	2.40	5.79
1000.0	8.19	10.97
10000.0	10.83	12.63

Handwritten annotations: A blue arrow points down the first column. A blue arrow points to the first row with the text "least-squares". A blue bracket groups the test MSE values from 10.0 to 1000.0, with an arrow pointing to the value 2.84. A blue arrow points to the last row with the text " $w \approx 0$ ".

Let's see how this works out in practice. So over here, I've created a toy data set in which there are 100 features, x_1 to x_{100} . Each of which has a Gaussian distribution. And there's also a response value y . **And the interesting thing here is that the response depends on only 10 of the features. It only depends on features x_1 through x_{10} . And the remaining features, the remaining 90 features are pure noise.** Now, the training set and test set each consist of 100 points. 100 training points in 100 dimensions. Is that a lot of data? No, it certainly is not. In fact you can pretty much always fit 100 points in 100 dimensions. It's like having 100 linear equations in 100 unknowns. So there isn't much data. And let's see what ridge regression would do in this case.

So lambda is something that we need to set. So let's look at a wide range of different value of lambda.

- When lambda is very small, like in this line over here, as we saw, we are essentially getting the least squares solution. The regularization term doesn't matter in this case. So it's the least squares solution. **It has zero training error, why? Because, as I said, you can pretty much always fit 100 points in 100 dimensions. But the test error is very high. So least squares is not doing very well in this case.**
- Now let's look at the other extreme, when lambda is very large. In this case, as we saw, what's happening is that we are gonna get a minuscule w , something that's pretty close to zero. So this is interesting. When w is essentially zero, the training error is 12.6. This emphasizes just how bad the least squares estimator is. It's getting a training error of 585 (at lambda is very small). So it's really doing very, very poorly in this case.
- Now, as we grow lambda, what's happening is that we are gradually reducing the emphasis on the training set. And we're gradually increasing the emphasis on having a small w . So since we're reducing the emphasis on the training set, the training error is gradually creeping up, as you can see. But the test error goes down for quite a while. **And in fact it bottoms out over here.** At that point it starts to increase again because what's happening then is essentially we are not paying enough attention to the training set. So the optimal value of lambda in this case would be somewhere around here.

How would we choose the optimal lambda in practice? Quite simply **by cross validation.**

The lasso

Popular “shrinkage” estimators:

- **Ridge regression**

$$L(w, b) = \sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2 + \lambda \|w\|_2^2$$

- **Lasso:** tends to produce sparse w

$$L(w, b) = \sum_{i=1}^n (y^{(i)} - (w \cdot x^{(i)} + b))^2 + \lambda \|w\|_1$$

Toy example:

Lasso recovers 10 relevant features plus a few more.

So we talked about one shrinkage estimator, ridge regression, in which the regularization term was simply the squared **L2 form of w** .

But what if we use a different norm? What would happen in that case? Well, one very popular choice is to use the **L1 norm**. And in that case the estimator is called the **lasso**. It has roughly the same good generalization behavior as a ridge regression. But **it has one additional big added benefit, which is that it tends to produce sparse solutions w** . That is to say it produces vectors w that tends to have a lot of zeros in it, just a few nonzero entries. So, for example, if you were to go back to our little toy data set over here and run the lasso on it, the w it would find would only have a few nonzero features. It would identify the 10 relevant features and actually also find a few more. This is pretty impressive.

Now, in general, why would we want sparse solutions? Why is it attractive to have a w that is sparse?

- Well, there's certainly some **computational benefits**. If we're only using a few features, then that means there's much **less data that we need to store and the computation is also more quick**.
- But most of all, the benefit of having a sparse w is that the resulting linear model is **easier to understand and to interpret**. And humans are much more likely to appreciate and use a model that they actually understand.

So that concludes our treatment of regression. One thing we haven't talked about is going beyond linear regression to regression. To regression that, for example, uses polynomial functions. But a little bit later in the class, we will develop some methods that show how this can be managed quite easily. See you next time.

POLL

Increasing λ has what effect on the training error?

RESULTS

- | | |
|---|-----|
| <input type="radio"/> Increasing λ has no effect on training error | 0% |
| <input checked="" type="radio"/> Increasing λ causes a larger training error | 62% |
| <input type="radio"/> Increasing λ causes a smaller training error | 12% |
| <input type="radio"/> Increasing λ causes the training error and test error to converge | 25% |

FEEDBACK

Increasing λ causes a larger training error

总结:

4.4 Linear Models for Conditional Probability Estimation

Topics we'll cover

- 1 Sources of uncertainty in prediction
- 2 Linear functions for conditional probability estimation
- 3 The logistic regression model

In the beginning of this class, we spent a lot of time in classification problems. And recently, we've been talking quite a bit about regression. What we gonna do today is to move on to another type of prediction problem. **Conditional probability estimation**.

So we'll start by looking at sources of uncertainty in prediction problems and then we'll see how this uncertainty can be captured by linear functions and eventually this will lead to the logistic regression formulation.

Uncertainty in prediction

Can we usually expect to get a perfect classifier, if we have enough training data?

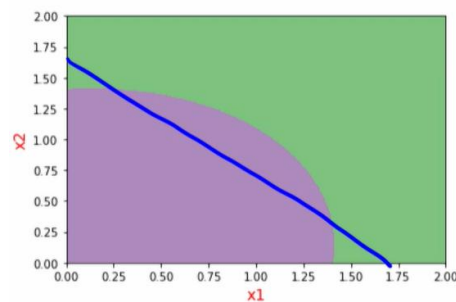
Problem 1: Inherent uncertainty

The available features x do not contain enough information to perfectly predict y , e.g.,

- x = complete medical record for a patient at risk for a disease
- y = will he/she contract the disease in the next 5 years?

Problem 2: Limitations of the model class

The type of classifier being used does not capture the decision boundary, e.g. using linear classifiers with:



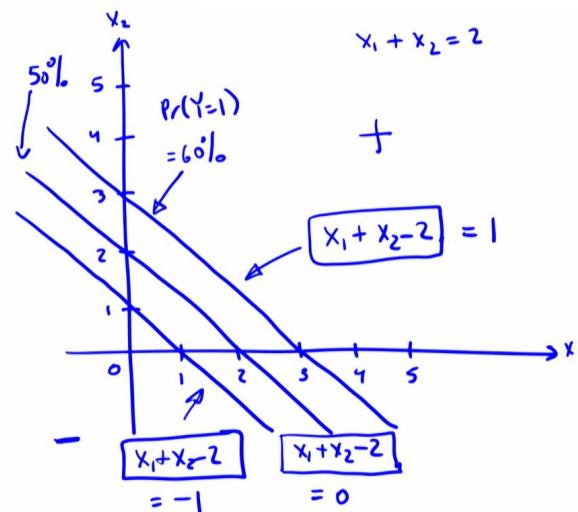
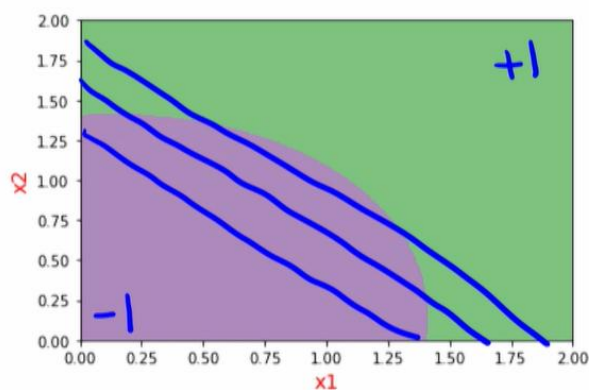
So here's a basic question. Suppose you wanna build a classifier and let's say you have all the training data you could possibly want, as much data as you like. Can you, in general, hope to get a perfect classifier. The answer, sadly, is in general, no. And there are two big reasons for this.

- The first reason is that the features you have, the x , may simply not contain enough information to perfectly predict y . So for example, let's say that the data, the x , are medical records for patients who are at risk for a certain disease. And let's say y is whether the patient actually does get that disease over the, say next five years. So even with all these information, even with the complete medical record, it's still not possible to predict with complete certainty, whether or not somebody is gonna fall sick. There's so much that depends on chance. So this is a situation which we would like to be able to make a prediction, but it would also be useful if we could accompany that prediction with some sort of quantification of the uncertainty. If we could say something like, there is an 80% chance that this person will fall sick.
- Now the second big reason, who are in general, we can't hope for a perfect classifier, is that very often the kind of classifier that we are using is simply not capable of perfectly capturing the true decision boundary. So let's look at this picture over here. We have a two-dimensional situation. There are two classes, purple and green and there is some true decision boundary between the two classes. So, let's say we are using linear classifiers. There's no way to perfectly capture this boundary. The best we could do would probably be something like this. So, in this case, we would still be able to make predictions but it would be nice if we could also quantify the uncertainty. If we could say something like, there is a 70% chance that the label is green.

Conditional probability estimation for binary labels

- Given: a data set of pairs (x, y) , where $x \in \mathbb{R}^d$ and $y \in \{-1, 1\}$
- Return a classifier that also gives probabilities $\Pr(y = 1|x)$

Simplest case: using a linear function of x .



So, we're talking about situations in which we want to classify, but in addition, we also want to output a probability. We wanna output the probabilities of the different labels. Now, for simplicity, we're gonna be looking at the case where there are just two possible labels plus and minus, plus one and minus one. This is the binary case. The methods that we developed, the **logistic regression method**, applies also in the multi-class setting and we'll see how this generalization works, it's very easy, but we won't see it for another week or two. So, for the time being, we're looking at **binary problems**. And we have points x and we wanna be able to predict the probability that y equals one at x . Now, of course, the probability that y equals minus one is just one minus this. Now, let's see what happens in the picture we looked at before. And let's say we that we wanna use a linear function. So we'd have a boundary like this. **And along the boundary, there is a 50% chance that y is one.** Let's say in this case that green is plus one and purple is minus one. So, along that line, there's a 50% chance, say, that y equals one and then we'll look at parallel lines. So, along this line the chance would be slightly higher. Let's say 60%, and along this line, the chance would be slightly lower. The probability of y equals one might be something like 40% along that line. So this is the kind of model that we'll be coming up with. We'll have a linear decision boundary. On one side of the boundary, we'll predict plus. On the other side, we'll predict minus. **And in terms of the probabilities, it really depends on the distance to the boundary. The closer to the boundary, the closer the probability is to a half.**

(以下是蓝色坐标轴的解说)

Now, let's see how this would work out mathematically. So, let me go ahead and draw a 2d grid. So, let's say we have two dimensions like this. This is zero, one, two, three, four, five. And along the x_1 , so this is the x_1 one direction and then we have the x_2 two direction and along this direction, we have one, two, three, four and five. So, we have two-dimensional data.

- Let me draw a linear boundary. What about this boundary over here. What is the equation of this line? Well, let's see, the equation of the line is basically $x_1 + x_2 = 2$. That's the equation of that line over there. **Let's write it in a slightly different way. As $x_1 + x_2 - 2 = 0$. And in general, we're gonna be writing our linear boundaries like this, as some linear function equals zero.** So, this is the linear function in our case. The boundary is when this linear function equals zero. On one side of the boundary, we'll predict plus while on the other side we'll predict minus.
- What is the set of points along which we are gonna predict, say, a 60% chance of plus. It might be something like this. What is the equation of that line? We'll, let's see. That line is $x_1 + x_2 = 3$. And another way to write it is that **$x_1 + x_2 - 2 = 1$** .
- Now, let's look at a line on the other side. Maybe something like this. If these are the 50% points, where there is a 50-50 chance of the two labels, and these are the 60% points, where the probability of y equals one is 60%, then the symmetric line on the other side will be the 40% points, where the probability that y equals one is 40% and conversely, the probability that y equals minus one is 60%. What's the equation of this line? Well, **this is $x_1 + x_2 - 2 = -1$** .

So, do you see the general pattern over here. In order to classify a point and in order to determine the probability that y equals one, all we're gonna do is to compute this linear function. In this case, $x_1 + x_2 - 2$. If it works out to zero, there's a 50, 50 chance. If it's positive, then the chance is greater than half. If it's negative, the chance is less than a half. **And the larger this value, the larger the probability that y equals one.**

总结:

A linear model for conditional probability estimation

For data $x \in \mathbb{R}^d$, classify and return probabilities using a linear function

$$\underbrace{w_1 x_1 + w_2 x_2 + \dots + w_d x_d + b}_{w \cdot x + b}$$

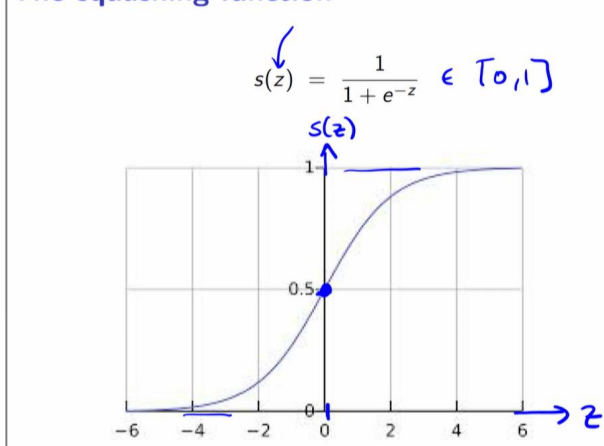
where $w = (w_1, \dots, w_d)$.

The probability of $y = 1$:

- Increases as the linear function grows.
- Is 50% when this linear function is zero.

How can we convert $w \cdot x + b$ into a probability?

The squashing function



Now let's move to a more general setting, where we have d dimensional data. And again, we're gonna be computing some linear function of this data. So, we have data points x in \mathbb{R}^d . What is a general linear function in d dimensions? Well, it's something of this form. $w_1 x_1 + w_2 x_2 + \dots + w_d x_d + b$. So, this is a general linear function. And we can rewrite it a little bit more compactly as $w \cdot x + b$, where w is just a vector of the coefficients, w_1 through w_d . So, that's our linear function. So, we get a point x , we compute this linear function. If it turns out to be zero, that means that this point is right on the boundary between the two classes. There's a 50, 50 chance that y equals one. And the larger this function is, the larger the probability that y equals one. But how do we get an actual probability out of this? After all, this function, $w \cdot x + b$, this returns an arbitrary real value, it could be anywhere from minus infinity to plus infinity. How do we convert this into a probability? How do we take a number from minus infinity to plus infinity and get a number in the range zero to one? The way we do that, is by using the **squashing function**.

So, the squashing function takes a real value, z and it outputs one over one plus e to the minus z . And this is something that's always in the range zero to one. So, in this graph, this is the z over here and on this axis, you had s at z . So, when you plug in z equals zero, for instance, into that formula, you get back a half. So, you get that point over there. And as z gets larger, the numbers get larger as well. And as z goes to infinity, they asymptotically reach one. That's what you're seeing up here. And when z goes to negative infinity, they asymptotically reach zero. So, this takes a real number, turns it into a probability and when you plug in the value zero, you get a half. So, this is really exactly what we need.

The logistic regression model

Binary labels $y \in \{-1, 1\}$. Model:

$$\Pr(y = 1|x) = \frac{1}{1 + e^{-(w \cdot x + b)}}$$

What is $\Pr(y = -1|x)$?

$$1 - \frac{1}{1 + e^{-(w \cdot x + b)}} = \frac{e^{-(w \cdot x + b)}}{1 + e^{-(w \cdot x + b)}} \times \frac{e^{(w \cdot x + b)}}{e^{(w \cdot x + b)}}$$

$$\Pr(y = -1|x) = \frac{1}{1 + e^{(w \cdot x + b)}} \leftarrow \leftarrow$$

$$\Pr(y|x) = \frac{1}{1 + e^{-y(w \cdot x + b)}}$$

So, when we have a point x , the probability that y equals one is gonna be the following. First, we compute the linear function, $w \cdot x + b$. And then, we put it through the squashing function. And this gives us a probability in the range zero to one. Now, this is the probability that y equals one, for that specific point x . What is the probability that y equals minus one? Well, let's just one minus this. So, let's write that down. The probability that y equals minus one is one minus whatever we have up there, which at one minus one over one plus e to the negative $w \cdot x + b$. Let's simplify it a little bit. So, we get e to the negative $w \cdot x + b$ over one plus e to the negative $w \cdot x + b$. And you know what, let's multiply the numerator and denominator by exactly the same quantity, which is e to the $w \cdot x + b$. So, let's multiply top and bottom by the same thing. And what do we get? Well, the top just becomes one and the bottom becomes one plus e to the $w \cdot x + b$. So, we've simplified it a little bit.

But now, if you look at the two formulas, so the one at the bottom is the probability that y equals minus one, given x . If you look at this formula, and you look at the one up there, you see that they're actually very similar. And in fact, we can put them together and just have a single, unified formula. What is that? It's the following. The probability of any y given x , where y 's either plus one or minus one, is simply one over one plus e to the negative y times $w \cdot x + b$. So, if you plug in y equals one, you get the formula on the top. If you plug in y equals negative one, you get this formula over here and so now, we have a single formula that covers both cases. This is the logistic regression model.

Summary: logistic regression for binary labels

- Data $x \in \mathbb{R}^d$
- Binary labels $y \in \{-1, 1\}$

Model parametrized by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$:

$$\Pr_{w,b}(y|x) = \frac{1}{1 + e^{-y(w \cdot x + b)}}$$

Learn parameters w, b from data

So, here's a summary of what we have. We have data in d dimensional space. There are two possible labels and in addition to classifying, we're gonna up with probabilities and this is the form of the probability function. Now, it's based on two parameters, w and b , that define a linear function. What are these parameters? How do we set them? Well, this is what we have to learn from data and we'll see how to do that next time.

So, that's it for today. We've talked a little bit about quantifying the uncertainty in prediction and we have seen that this can be done quite easily using linear functions.

Next time, we'll see how to learn this linear function. So see you then.

POLL

If we have a two dimensional linear decision boundary, $x_1 + 2x_2 - 2 = 0$, how would we classify the point $p = (2, 0)$?

RESULTS

- | | |
|--|------------|
| <input type="radio"/> p is classified as 1, with <50% probability | 0% |
| <input checked="" type="radio"/> p is classified as 1, with 50% probability | 88% |
| <input type="radio"/> p is classified as 1, with >50% probability | 0% |
| <input type="radio"/> p is classified as -1, with >50% probability | 12% |

FEEDBACK

p is classified as 1, with 50% probability

总结:

4.5 Logistic Regression

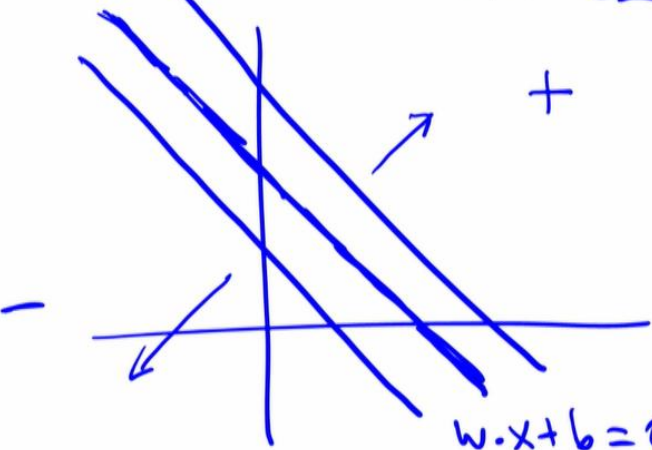
Topics we'll cover

- 1 The logistic regression model
- 2 Loss function: properties
- 3 Solution by gradient descent

Today, we'll look at how to learn a logistic regression model from data. So we'll begin by reviewing the form of the logistic regression model and then we'll phrase the learning problem as an **optimization task**, in which the goal is to minimize a suitable loss function. It'll turn out that this is a very nice loss function. In particular, it is **convex**. We'll then see how this function can be minimized using **gradient descent**.

Logistic regression for binary labels

- Data $x \in \mathbb{R}^d$ and binary labels $y \in \{-1, 1\}$
- Model parametrized by $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$:

$$\Pr_{w,b}(y|x) = \frac{1}{1 + e^{-y(w \cdot x + b)}} \quad \left. \vphantom{\Pr_{w,b}(y|x)} \right\}$$


So here again is the logistic regression model. We were looking at the case of binary labels, where Y can only take on two possible values, minus one and plus one. The data points X lie in say D dimensional space and for any given point X the probability of Y where Y is either minus one and plus one, is given by this formula over here. We first compute a linear function of X $W \cdot X$ plus B , and then we take one over one plus E to the minus Y times that linear function. This is the logistic regression model and it has two parameters, W and B , that we need to set.

So, what does this look like geometrically? If we look, for example, at the boundary okay so we have the plus side, we have the minus side, **what is the decision boundary? Well, it's the set of points for which this probability is exactly a half. And those are the points for which $W \cdot X$ plus B equals zero.** Okay so, we have some set of points for which $W \cdot X$ plus B equals zero, and this is the decision boundary. Plus on this side and minus on this side. Now in two dimensions, this boundary is a line. In three dimensions, it's a plane. And in higher dimensions, it's a hyperplane. Now put at the points for which the probability of Y being one is 60%. Well, it's a parallel hyperplane on this side. It's something like this. And what are the points for which the probability of Y being one is 40%? Well, it's another parallel hyperplane, on the other side. And so as one moves away so **this central hyperplane over here, $W \cdot X$ plus B equals zero, is the decision boundary and as one moves further in this direction, the probability of Y being one increases, and in this direction it decreases.** So this is nice and intuitive. And the remaining question now is how do we learn this linear function, this W and B , from data? So let's see how we do that.

The learning problem

Maximum-likelihood principle: given data $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \{-1, 1\}$, pick $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$ that maximize

$$\prod_{i=1}^n \Pr_{w,b}(y^{(i)} \mid x^{(i)})$$

Take log to get **loss function**

$$L(w, b) = - \sum_{i=1}^n \ln \Pr_{w,b}(y^{(i)} \mid x^{(i)}) = \sum_{i=1}^n \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)} + b)})$$

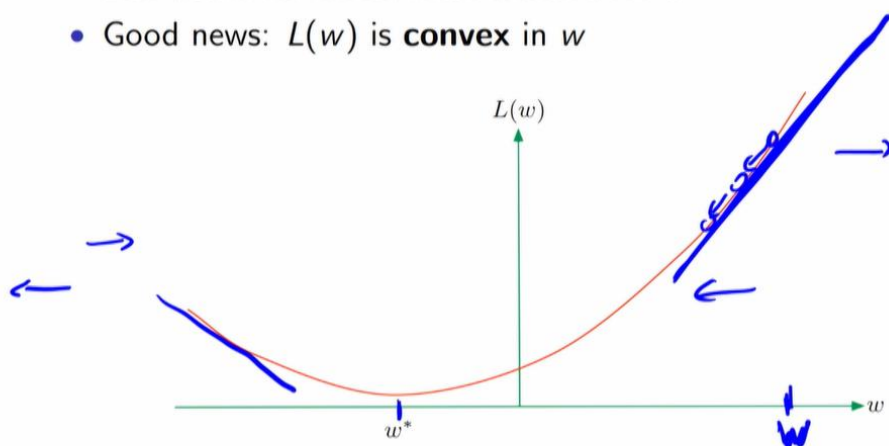
Goal: minimize $L(w, b)$.

As with linear regression, can absorb b into w .
Yields simplified loss function $L(w)$.

So we have a set of data points, X_1 through X_n , along with their labels, Y_1 through Y_n . What we're gonna do is to use the maximum likelihood principle and we're gonna pick the linear function, the W and B , that maximizes the probability of the data. Okay, so **what we wanna maximize is the probability of Y_1 given X_1 , times the probability of Y_2 given X_2 , times the probability of Y_3 given X_3 , all the way to the probability of Y_n given X_n . Okay, so we have a concrete optimization criteria**. So we're in good shape. Now, one little problem over here is that this function that we're trying to optimize has got this big product in the front and this can be a little bit inconvenient. It would be simpler if we were instead dealing with a sum. So how do we go from a product to a sum? Well we just take the logarithm. Okay so if you remember, the logarithm of A plus B , of A times B , is the log of A plus the log of B . So instead of maximizing this function we can just go ahead and maximize its log which leads to the same answer and then what we get is this sum over here. Now **we're also gonna stick a minus in front so that instead of maximizing, we actually wanna minimize**. So this then is our final loss function. This function over here. We want to find the W and B that minimize this function. How do we do that? Well if you remember, when we were talking about least squares regression, we were in a little bit of a similar situation. We also wanted to find some W and B , and we also had a loss function that we were trying to minimize. So what was it that we did back then? Well the first thing we did was that we tried to get rid of B , because **it's a little bit of a nuisance**. So how do we do that? Oh, well what we said is that if you add a feature to X , if you take your data vector as X and you add an extra feature that's identically equal to one, then you can just assimilate B into W . **B just becomes another coordinate of W .** Okay so let's just start by doing that. And then we can forget about B and we just have a loss function that's in terms of the W , okay? So that's a help, **but still what is the optimal W ? What is the solution?**

Convexity

- Bad news: no closed-form solution for w
- Good news: $L(w)$ is **convex** in w



How to find the minimum of a convex function? By **local search**.

And there's good news and bad news.

- The bad news is that there's **no closed-form equation for W** . When we were doing least squares regression, we had this nice formula, W equals something nice and simple. **But that's rare**. That almost never happens and that does not happen in this case. Okay, so that's the bad news.
- The good news is that **the loss function is very well-behaved. It is convex**. What does that mean? Well it's a very important concept. It's something that plays a central role in machine learning and we'll be spending more time on it later on. But we won't get into it right now, **for the time being, what it means is that the loss function basically looks like a bowl. And we wanna get to the bottom of the bowl.** but what's so great about that? Well what's great about it is that the loss function does not look like this. Now loss functions like that also arise very frequently in machine learning and they are much more troublesome because they have all **these local optima** in which we can **get stuck**. The convex situation is much nicer. There's just one local optimum over here, and it's also the **global optimum**.

So how do we find that point? How do we find that optimal value W ? Well the way we're gonna do it is **by local search**. Okay, what that means is you start off at any old point, let's say you start off by guessing that as being the right W . And then you just tweak it to make it a little bit better and you tweak it a little bit more to make it better and then you keep doing these little tweaks until finally, you end up somewhere around here and you find that none of the tweaks are really helping anymore and so you decide that you've converged and that's the answer. Okay, so that's **local search**.

In a sense, you're starting over here and gradually rolling down the hill. So when you're at a particular value of W , let's say this is what your current guess at W is, how do you know which way to go? Well there are many ways to decide this. But one rather simple way is **by looking at the slope of the function**. At this point for example, the function's slope is positive. What does that mean? It means that if you go this way the loss function will increase, and if you go that way, it'll decrease. What do we want? Do we want to increase it or decrease it? We wanna decrease it because we're trying to minimize it. So we know we have to move to the left. Likewise over here, the slope is negative. We know that if you go that way, it goes down, the function goes down. And if you go that way, the function goes up. So we have to move to the right. So in this way, **the gradient or slope of the function can really serve as our guide. We just have to move in the opposite direction to the gradient.** That is called **gradient descent**.

Gradient descent procedure for logistic regression

Given $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \{-1, 1\}$, find

$$\arg \min_{w \in \mathbb{R}^d} L(w) = \sum_{i=1}^n \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)})})$$

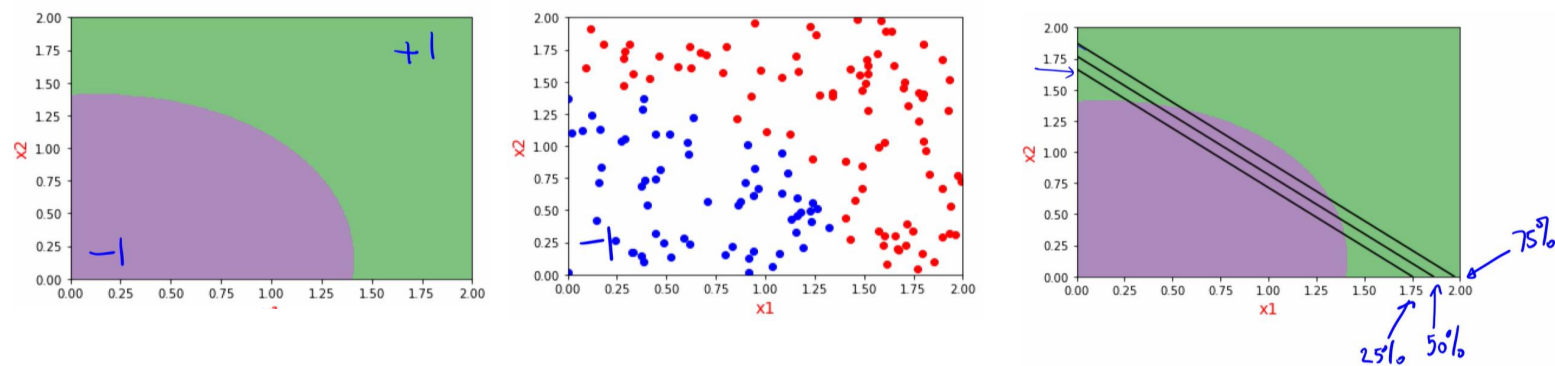
- Set $w_0 = 0$
- For $t = 0, 1, 2, \dots$, until convergence:

$$w_{t+1} = w_t + \eta_t \sum_{i=1}^n y^{(i)} x^{(i)} \underbrace{\Pr_{w_t}(-y^{(i)} | x^{(i)})}_{\text{doubt}_t(x^{(i)}, y^{(i)})},$$

where η_t is a "step size"

We're gonna be talking about optimization in more detail later on. And we'll also be spending quite a bit of time on gradient descent. So I'm not gonna explain exactly how the gradient descent procedure for logistic regression comes about right now, but I just wanna show you what the procedure looks like, 'cause it's extremely simple. So we begin by setting W to some value say the all zeroes vector. That's our initial guess for W , W sub zero. At time T , our guess for W is W_T , and then we're gonna tweak it a little bit and we get our updated guess, W_T plus one. So **what is this tweak that we do? Well we take W and we look at the negative gradient and we move in that direction times a small step size.** So what are these two things? Well let's look at the summation first. The Y s, well the Y s are just plus one or minus one. This is some probability, some number, and these are the data vectors. **So it looks like we're doing some sort of weighted average of the data points.** Where the weight for a particular X_i depends on how well we classified it. So suppose that the current guess W sub T assigns high probability to the wrong label for X_i . Let's say it assigns high probability to the opposite label, negative Y_i . That means we're doing pretty badly on that point and in this case we'll increase the weight on it. So we emphasize that point a little bit more. We know that it's a point we really need to work on. So **we compute a weighted average of this kind, which is quite intuitive, and then so that gives us a direction in which to move, but we also have to decide how far we wanna move in that direction. And for that we use the step size over here.** Now this is again something we'll be talking about more later on. For the time being, you can just pretend that it's a constant, something like point one. Although, as we'll see later, it often makes sense to reduce it as time goes on.

总结:



Okay, so this is a nice simple algorithm for a logistic regression. So let's see what it does on our toy example.

So again we have these two classes, let's say this is minus one and let's say this class is minus one and this is plus one. We go ahead and sample some points and assign them their correct labels. And then we apply that gradient descent procedure to find the right linear function. And this is what it ends up finding. Okay, so what are these three lines? This line over here, is the decision boundary, the 50% line. This line over here is the 25% line. And this line on this side is the 75% line. Okay so what it's saying is that if you look at this line on the top if you look at the Xs that lie along that line, there is a 75% probability that the label is green. And if you look at this line over here, it says that for the Xs along that line, there is a 25% probability that the label is green. In other words, a 75% probability that the label is purple. Is that reasonable? Well, maybe. In general, when you're using the conditional probabilities from a logistic regression model, you do have to take them with a little grain of salt.

Okay, well that's it for today. We've see how a logistic regression model can be learned from data, and we've tried it out on a toy data set. Next time, we'll try it out on a more substantial learning problem, a text classification task. See you then.

POLL

What is the benefit of using gradient descent on a convex function?

RESULTS

- | | |
|---|-----|
| <input type="radio"/> It takes less steps to find the optimal solution | 20% |
| <input type="radio"/> There can exist multiple solutions | 0% |
| <input checked="" type="radio"/> There exists only one optimal solution | 80% |
| <input type="radio"/> Convexity doesn't benefit gradient descent | 0% |

FEEDBACK

There exists only one optimal solution

4.6 Logistic Regression in Use

Topics we'll cover

- 1 A text classification problem
- 2 Bag-of-words representation for text
- 3 Solution by logistic regression
- 4 Margin versus test error
- 5 Interpreting the model

Today we will look at a small case study of logistic regression in use. The problem we'll be solving is a text classification problem.

So first we will see how a text document can be converted into a vector of fixed length so that methods like logistic regression can be run on it. Then we'll start the learning process and interpret the results that we get along the way.

Sentiment data

Data set: sentences from reviews on Amazon, Yelp, IMDB.
Each labeled as positive or negative.

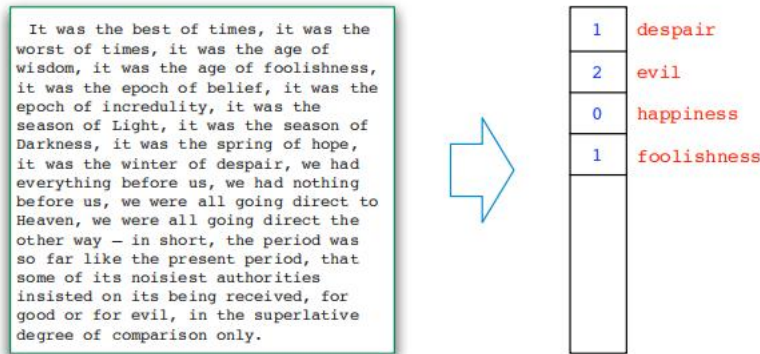
- Needless to say, I wasted my money. —
- He was very impressed when going from the original battery to the extended battery. +
- I have to jiggle the plug to get it to line up right to get decent volume. —
- Will order from them again! +

2500 training sentences, 500 test sentences

So here's the problem we wanna solve. We get a review of a product or movie or restaurant and we wanna figure out if it's a positive review or a negative review. The data we have is collected from various websites like Amazon and Yelp, and from each review, just a single sentence has been extracted and it's been labeled as plus or minus, positive or negative. Okay, so let's look at some of these examples. So the first one. Needless to say, I wasted my money. Okay, so clearly that's a negative review and the word that really tips you off over there is wasted. Okay, so that's negative. The second one. He was very impressed when going from the original battery to the extended battery. So that's positive. And what is the keyword over there? Impressed. The third one. I had to jiggle the plug in order to get it to lineup right to get decent volume. So that's negative. And in terms of words that really stick out, there isn't really anything. There's no single word in there that's completely negative. So that one's a little bit tricky. And let's look at the last one. Will order from them again. Okay, so that's positive. And again, there's no single word that really indicates that. Again, not really. There are probably lots of reviews that say things like will never order from them again. So that's also a tricky one. So this is going to be a nontrivial task. And because it's a little bit difficult, logistic regression will help us quantify the uncertainty in some of the predictions. Since it outputs not just the prediction, plus or minus, but also gives an accompanying probability. Now how much data do we have? About 2,500 training sentences, which is not very much at all, and then there's a separate test set of 500 sentences.

Handling text data

Bag-of-words: vectorial representation of text sentences (or documents).



- Fix $V =$ some vocabulary.
- Treat each sentence (or document) as a vector of length $|V|$:

$$x = (x_1, x_2, \dots, x_{|V|}),$$

where $x_i = \#$ of times the i th word appears in the sentence.

Now in order to run logistic regression we need the data to be vectors of some fixed length. How do we convert sentences into vectors? Now there's actually a very standard way of making documents into vectors, and this is called the **bag-of-words representation**. In our case you can think of each sentence as being a mini document of some kind. So here's how it works. You start by fixing some vocabulary. For instance, all the words in the documents, or maybe the 5,000 most common words, or in some other way you choose a list of words. For concreteness, let's say that we choose the 5,000 most common words. **Then each document is represented as a 5,000 dimensional vector. We have 5,000 dimensions with one entry for each of the words in the vocabulary. And that entry is literally how many times that word occurs in the document.** So for example, let's say that the first word in the vocabulary is despair. So we have a document and what we do is we go through it and we count how many times despair occurs in the document. Oh, it occurs once? So the entry for that is one. Let's say the next word is evil. So we go through the document and see how many times evil occurs in the document. Oh, two times, so the entry is two. The third word is happiness. That doesn't occur, so zero. And so on. And in this way, we convert this document into a 5,000 dimensional vector.

Now one thing that's worth pointing out is that this representation is extremely sparse. If we're applying this to sentences and we have a sentence of just 10 words, then out of those 5,000 inquiries, at most 10 of them are gonna be nonzero. **So it's a peculiarity of this particular way of representing documents.**

A logistic regression approach

Code positive as +1 and negative as -1.

$$\Pr_{w,b}(y | x) = \frac{1}{1 + e^{-y(w \cdot x + b)}}$$

Given training data $(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^d \times \{-1, 1\}$, find w, b minimizing

$$L(w, b) = \sum_{i=1}^n \ln(1 + e^{-y^{(i)}(w \cdot x^{(i)} + b)})$$

Convex problem with many solution methods, e.g.

- gradient descent, stochastic gradient descent
- Newton-Raphson, quasi-Newton

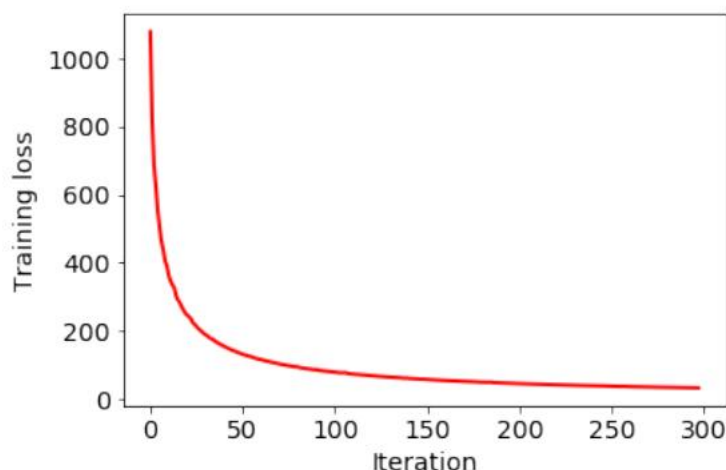
All converge to the optimal solution.

Okay so now we're ready to apply logistic regression. We have two classes, positive and negative. Let's code positive as plus one and negative as minus one. We have our 2,500 data points. Each of which is now a 5,000 dimensional vector. And we need to find a linear function, given by w and b , that minimizes the logistic regression loss function.

As we saw last time, this is a nice loss function. It is convex and it can be optimized quite easily using local search methods. There are many choices. Gradient descents, stochastic gradient descent, Newton-Raphson, and so on and so forth. They all return the right answer. So what we'll do today is use one of these, stochastic gradient descent. This is a very important method and this is something that we'll cover in more detail next week.

Local search in progress

Look at how loss function $L(w, b)$ changes over iterations of stochastic gradient descent.



Final model: **test error 0.21.**

But for the time being, it's a local search method. It starts with some solutions, some guess at w and b , and then it tweaks it a little bit and it keeps tweaking it and eventually it converges to the right answer. So let's see what happens over training iterations. Now it turns out that for this particular dataset, 300 iterations are needed before the linear function, w , b , converges. Each iteration involves a pass through the entire dataset. And as these iterations proceed, the loss function on the training set, the training loss, keeps going down. And in fact, during the first 50 or so iterations, the loss function just plunges. And after that it just tapers off gradually until it finally grinds to a halt. So that's how training goes. And at the end we're left with our final classifier, w and b . How does it do? What is its performance on the test set? So it turns out that the test error is 21%. That's not great but it's also not bad considering the ambiguity and difficulty of these sentences.

Some of the mistakes

Not much dialogue, not much music, the whole film was shot as elaborately and aesthetically like a sculpture. 1

This film highlights the fundamental flaws of the legal process, that it's not about discovering guilt or innocence, but rather, is about who presents better in court. 1

You need two hands to operate the screen. This software interface is decade old and cannot compete with new software designs. -1

The last 15 minutes of movie are also not bad as well. 1

If you plan to use this in a car forget about it. -1

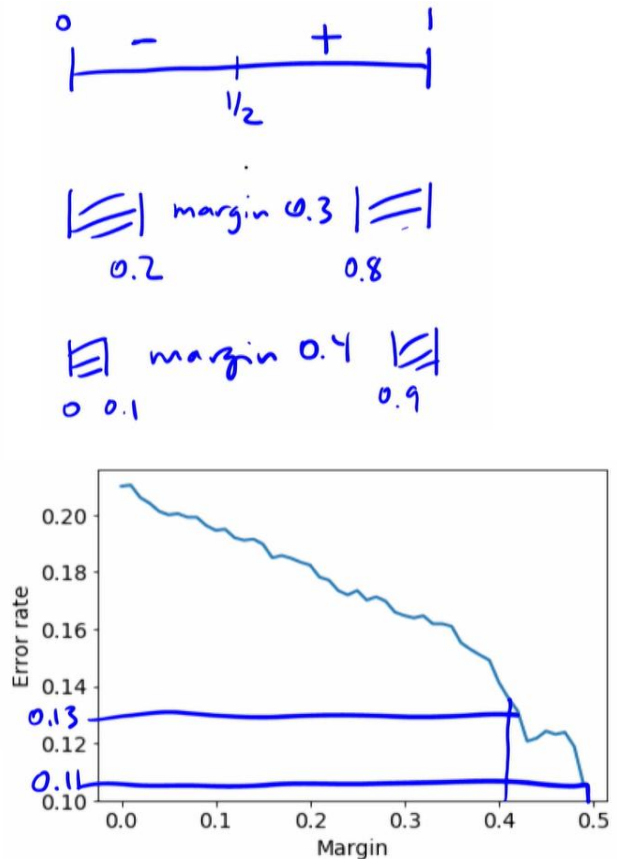
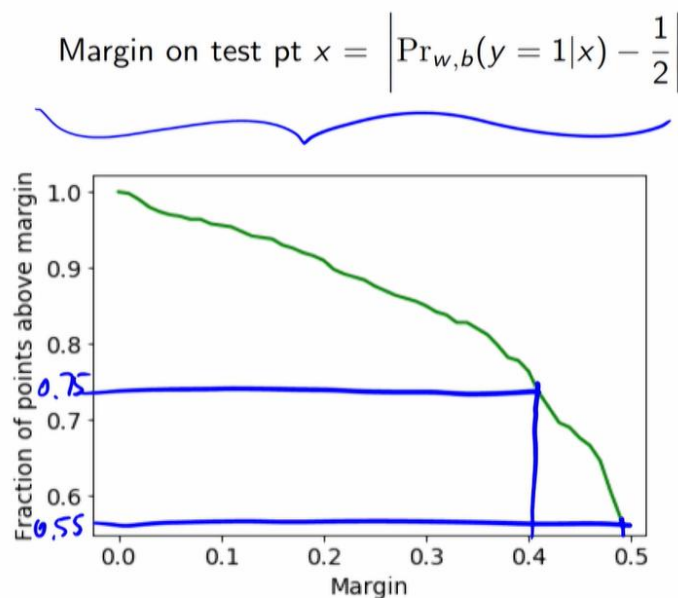
If you look for authentic Thai food, go else where. -1

Waste your money on this game. 1

So let's go ahead and look at some of the mistakes.

- So let's see the first one. Not much dialogue, not much music, the whole film was shot as elaborately and aesthetically like a sculpture. And that's positive. It starts off sounding a little bit negative and there's really no word in there which indicates that it would be a positive review. It's not an easy one. So that's one of the mistakes it made.
- Let's look at this one. The last 15 minutes of movie are also not bad as well. So it's got the double negative, not bad, which is a little bit confusing. And so the review is actually positive, but the logistic regression model thought it was negative.
- How about this one? If you look for authentic Thai food, go elsewhere. Okay, that's a difficult one. It's got authentic Thai food in there which sounds good, but then it says go elsewhere. So that's difficult.
- And the last one, waste your money on this game. Okay, how are we supposed to get that right? That's a very difficult one.

Margin and test error



Okay, so we've been talking about using logistic regression for classification, but actually it also outputs a probability value, which we can think of as a kind of level of confidence. What do these probabilities, what do these confidence levels look like in this case for this dataset? On any given sentence x , we get a probability between zero and one. And if the probability's above a half, then we end up predicting plus, and if the probability is less than a half, we end up predicting minus. Now when the probability is close to a half, it means that we're really very uncertain. Predicting one way, we say plus or minus 'cause we're forced to, but actually there's a lot of uncertainty in there. The more confident predictions are the ones that are closer to zero or one. For example, suppose we look at the cases where it is above 0.8 or below 0.2. **You can think of this as a sort of 80% confidence. So we will call this, we'll describe this as having a margin of 0.3. What that means is that it's at least 0.3 away from a half.** And this is the formula for margin over here. It's how far the probability is from a half. Or we can look at 90% confidence where the probabilities are between 0 and 0.1, or between 0.9 and 1. So this is a margin of 0.4. Or we can look at even larger margins, like a margin of 0.49, which would be 99% confidence.

So for what fraction of points was the margin, say, 0.4? For what fraction of these test points was there 90% confidence? Well that's what this graph shows over here. If we want a margin of 0.4 we just go up over here and we see that oh, roughly 75% of the points had a margin of 0.4. So three quarters of the test points had 90% confidence or more. Let's look at 99% confidence. That's somewhere here. It looks like roughly 55% of the data had 99% confidence. That is a lot of confidence. So this logistic regression model is making very confident predictions even though it did not get a whole lot of training data. **That's a little suspicious in and of itself.** And so it would be interesting to see whether this confidence is at all warranted. We know that the overall error rate was 21%, but what if we just look at the points on which it was confident.

Let's say, the points on which it was 90% confident. What was the error rate on those points? And that's what this graph over here shows. So 90% confidence, that's when the probability's either greater than 0.9 or less than 0.1 and that's a margin of 0.4. That's this thing over here. And if we look at those points, the error rate was something like 13%. Much less than 21%. And if we look at the 99% confident points, the error rate there is just a little above 10%. So on the test set as a whole, the error rate was 21%. But more than half the points had a 99% confidence, and on those points, the error rate was much smaller. It was just a little over 10%. This shows two things. (后续)

总结:

- The first thing is that these confidence levels have to **be taken with a grain of salt**. When it's 99% confident, that doesn't mean that it's going to be correct 99% of the time.
- But the second thing is that **these confidence values are really informative**. By focusing on high confidence predictions, we really can decrease the error rate.

Okay, now let's try and get some other information about the model that we learned. What is the basis for its predictions? What words is it placing the greatest emphasis on? **How do we determine this? Well, at the end we have this vector w , which is 5,000 dimensional, and there's an entry for each word. So the most significant words, the ones that have the greatest role in prediction, are the ones with the largest coefficients. Either the most positive coefficients or the most negative coefficients.**

What are these words? Let's take a look. So here they are. On top are the words with the largest positive coefficients, and the bottom are the ones with the largest negative coefficients.

So what are the positive words? Beautiful, fantastic, excellent, wonderful, nice, awesome, perfect, and so on. And the negative words, disappointing, stupid, lazy, dirty, bad, fails, unfortunately, and so on. Do these seem like words that are reasonable indicators of positive or negative reviews? They do. It all seems quite reasonable.

Okay, well that's it for today. And that's also it for logistic regression. One of the discussions that we've been putting off a little bit is how you actually optimize all of these loss functions. What are the techniques involved, and the technology involved in this? Things like convexity and stochastic gradient descent, these really lie at the heart of modern machine learning. So what we're gonna do next is to take a little bit of time and to study these slowly and carefully. See you next time.

Interpreting the model

Words with the most positive coefficients

'sturdy', 'able', 'happy', 'disappoint', 'perfectly', 'remarkable', 'animation',
'recommendation', 'best', 'funny', 'restaurant', 'job', 'overly', 'cute', 'good', 'rocks',
'believable', 'brilliant', 'prompt', 'interesting', 'skimp', 'definitely', 'comfortable',
'amazing', 'tasty', 'wonderful', 'excellent', 'pleased', 'beautiful', 'fantastic',
'delicious', 'watch', 'soundtrack', 'predictable', 'nice', 'awesome', 'perfect', 'works',
'loved', 'enjoyed', 'love', 'great', 'happier', 'properly', 'liked', 'fun', 'screamy',
'masculine'

Words with the most negative coefficients

'disappointment', 'sucked', 'poor', 'aren', 'not', 'doesn', 'worst', 'average',
'garbage', 'bit', 'looking', 'avoid', 'roasted', 'broke', 'starter', 'disappointing', 'dont',
'waste', 'figure', 'why', 'sucks', 'slow', 'none', 'directing', 'stupid', 'lazy',
'unrecommended', 'unreliable', 'missing', 'awful', 'mad', 'hours', 'dirty', 'didn',
'probably', 'lame', 'sorry', 'horrible', 'fails', 'unfortunately', 'barking', 'bad', 'return',
'issues', 'rating', 'started', 'then', 'nothing', 'fair', 'pay'

RESULTS

- ☒ A larger coefficient means more consideration given to that word **90%**
- ☐ A larger coefficient means less consideration given to that word **0%**
- ☐ The size of the coefficient has no impact on how the model classifies the word **10%**

FEEDBACK

A larger coefficient means more consideration given to that word