Foundations of Machine Learning: Week 2: Logistic Regression

Professor Christopher Yau
christopher.yau@manchester.ac.uk

October 23, 2020

Foundations of Machine Learning: Week 2: Logistic Regression

Professor Christopher Yau
christopher.yau@manchester.ac.uk
October 23, 2020

Overview

In this lecture:

1. Introduce the logistic regression model
2. Generative model
3. Loss function
4. Optimisation with SGD
5. Properties

Overview

In this lecture:
1. Introduce the logistic regression model
2. Generative model
3. Loss function
4. Optimisation with SGD
5. Properties

In this lecture, we will use the theory we have established to construct a **logistic regression** algorithm.
This is a very simple probabilistic binary classification algorithm but whose construction has features that still underpin modern machine learning techniques.
I will talk about the generative models that underlie the logistic regression and the loss function we need to optimise using stochastic gradient descent to train the classifier.
I will then talk about some of the properties of logistic regression.

Probabilistic classifier

Linear classifiers use a weighted (**w**) linear function of the inputs and a bias term w_0:

$$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = w_1x_1 + w_2x_2 + \cdots + w_px_p + w_0$$

We could construct a probabilistic binary classifier by using the following model:

$$\begin{aligned} z &= \mathbf{w}^t\mathbf{x} + w_0, \\ h &= \text{Logistic}(z), \\ p(y=1) = h,\ p(y=0) &= 1-h, \end{aligned}$$

so that the probability that the class y equals 1 is given by h.

Probabilistic classifier
Linear classifiers use a weighted (**w**) linear function of the inputs and a bias term w_0:
$f(\mathbf{x}) = \mathbf{w}^t\mathbf{x} + w_0 = w_1x_1 + w_2x_2 + \cdots + w_px_p + w_0$
We could construct a probabilistic binary classifier by using the following model:
$z = \mathbf{w}^t\mathbf{x} + w_0,$
$h = \text{Logistic}(z),$
$p(y=1) = h,\ p(y=0) = 1-h,$
so that the probability that the class y equals 1 is given by h.

In the previous lecture we discussed how to construct a binary classifier using the perceptron algorithm. In this lecture, we will consider an alternative probabilistic approach known as logistic regression.
As before, we start with a linear construction where the inputs **x** are additively weighted by parameters **w**.
We could construct a probabilistic binary classifier by saying that a latent variable z corresponds to this weighted input.
We can transform z into a probability h between 0 and 1 using some function, in this case, the logistic function which we will examine shortly.
And then to use that probability to say that the output $y = 1$ with probability h and conversely $y = 0$ with probability $1 - h$.

Generative Process

Logistic regression assumes the output is generated from the input via the following algorithm.

Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$:

1. Compute $z = \mathbf{w}^t\mathbf{x} + w_0$,
2. Apply the logistic function to z to get $h = \text{Logistic}(z)$,
3. Generate a random number between 0 and 1 uniformly, $r \sim \text{Uniform}(0, 1)$
4. If $r < h$, set $y = 1$, else set $y = 0$.

This is a probabilistic algorithm because the output depends on the random number generated in step 3.

Generative Process

Logistic regression assumes the output is generated from the input via the following algorithm.

Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$:

1. Compute $z = \mathbf{w}^t\mathbf{x} + w_0$,
2. Apply the logistic function to z to get $h = \text{Logistic}(z)$,
3. Generate a random number between 0 and 1 uniformly, $r \sim \text{Uniform}(0, 1)$
4. If $r < h$, set $y = 1$, else set $y = 0$.

This is a probabilistic algorithm because the output depends on the random number generated in step 3.

What I have described is something known as a *generative process*.
In probabilistic modelling, it describes the mathematical process by which we model how our data is generated.
We can encode this in an algorithm.
Given input $\mathbf{x}$ and parameters $(\mathbf{w}, w_0)$:

1. Compute $z = \mathbf{w}^t\mathbf{x} + w_0$,
2. Apply the logistic function to z to get $\theta = \text{Logistic}(z)$,
3. Generate a random number between 0 and 1 uniformly
4. And if $r < h$, set $y = 1$, else set $y = 0$.

This is a probabilistic algorithm because the output depends on the random number generated in step 3.

Logistic Function

The **logistic function** maps the real line to $(0, 1]$ via the transformation:

$$h = \frac{1}{1 + \exp(-z)}$$

Note that as:

- $z \to \infty$: $h \to 1$,
- $z \to -\infty$: $h \to 0$,
- $z = 0 : h = 0.5$

So our previous statements are equivalent to saying:

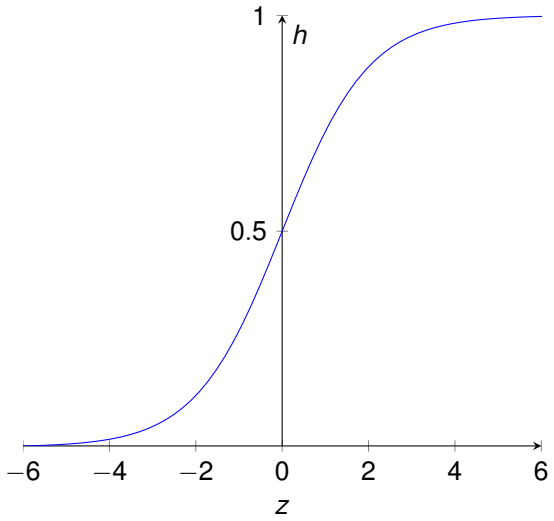
$$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\underbrace{[\mathbf{w}^t\mathbf{x} + w_0]}_{z})}$$

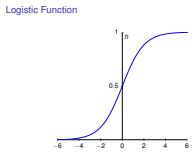
Logistic Function
The **logistic function** maps the real line to (0, 1] via the transformation:
$h = \frac{1}{1 + \exp(-z)}$
Note that as:
- $z \to \infty$: $h \to 1$,
- $z \to -\infty$: $h \to 0$,
- $z = 0 : h = 0.5$

So our previous statements are equivalent to saying:
$p(y = 1|\mathbf{x}, \mathbf{w}) = \frac{1}{1 + \exp(-\underbrace{[\mathbf{w}^t\mathbf{x} + w_0]}_{z})}$

The logistic function has the following form one over one plus exponential of the negative of the input argument.
This transformation maps the real number line from minus infinity to plus infinity to the range between 0 and 1. Hence why it is useful for converting a range of real numbers into a range that is suitable for describing probabilities.
Note that the as the input z tends to infinity, h tends to 1. And as the input z tends to minus infinity, h tends to 0. When $z = 0$, h equals 0.5.
So our previous statements are equivalent to saying that the probability that $y = 1$ is equal to the logistic function applied directly to the inputs $\mathbf{x}$ weighted by $\mathbf{w}$.
It isn't necessary to have the intermediate variable z but it is mathematically and computationally convenient for presentation and implementation.

Logistic Function





To illustrate here is a plot of the logistic function with its characteristic S shaped. It tends to one or zero in the extremes and has the value 0.5 when the input is 0. There are other related transformations, such as the probit function, which have their own properties but we will consider the logistic function only in this course.

Training

To train a logistic regression classifier, we require:

1. A data set, $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, comprising of n pairs of matched inputs $\mathbf{x}$ and outputs y,
2. A method for estimating the regression parameters $\mathbf{w}$ from the data D.

If we think of logistic regression as a probabilistic model then the optimal way of tackling (2) is by **maximum likelihood estimation**.

Training

To train a logistic regression classifier, we require:

1. A data set, $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, comprising of n pairs of matched inputs $\mathbf{x}$ and outputs y,
2. A method for estimating the regression parameters $\mathbf{w}$ from the data D.

If we think of logistic regression as a probabilistic model then the optimal way of tackling (2) is by **maximum likelihood estimation**.

So to train a logistic regression model, we require as usual two things:

1. A data set comprising of pairs of matched inputs and outputs,
2. and a method for estimating the regression parameters from the data.

Because logistic regression is a probabilistic model we will be adopting an approach known as maximum likelihood estimation for training.

Likelihood

Definition

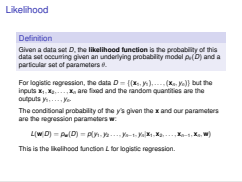
Given a data set D, the **likelihood function** is the probability of this data set occurring given an underlying probability model $p_\theta(D)$ and a particular set of parameters θ.

For logistic regression, the data $D = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ but the inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are fixed and the random quantities are the outputs $y_1, \ldots, y_n$.

The conditional probability of the y's given the $\mathbf{x}$ and our parameters are the regression parameters $\mathbf{w}$:

$$L(\mathbf{w}|D) = p_{\mathbf{w}}(D) = p(y_1, y_2 \ldots, y_{n-1}, y_n|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{w})$$

This is the likelihood function L for logistic regression.



Likelihood refers to the probability of observing some data given an underlying probability model which may have some parameters associated with it.
In logistic regression, our training data is pairs of matched inputs and outputs but the inputs are fixed and something we control in the experiment, so the only random outcome is encoded in the outputs.
We are therefore interested in the conditional probability of the outputs given the inputs and our regression parameters which are the **w**'s.
This conditional probability represents the likelihood function for our logistic regression model.
Note the likelihood is a function only of the parameters **w** because the inputs are fixed because we control them and in the training data we have already observed the outputs, therefore the only unknown with respect to the training data is what **w** generated that data.

Maximum Likelihood Estimation

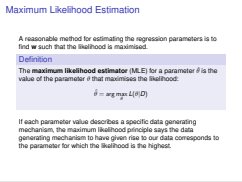
A reasonable method for estimating the regression parameters is to find **w** such that the likelihood is maximised.

Definition

The **maximum likelihood estimator** (MLE) for a parameter $\hat{\theta}$ is the value of the parameter θ that maximises the likelihood:

$$\hat{\theta} = \arg\max_{\theta} L(\theta|D)$$

If each parameter value describes a specific data generating mechanism, the maximum likelihood principle says the data generating mechanism to have given rise to our data corresponds to the parameter for which the likelihood is the highest.



It is therefore clear that we might want to choose **w** such that the probability of the data, or the likelihood, is maximised. This is the principle of maximum likelihood estimation. The maximum likelihood estimator for a parameter is the value of the parameters that has the largest likelihood over all possible parameters. What this means is that if we think of each possible parameter value as indexing a particular version of our logistic regression model. We want to choose the version of the logistic regression model under which our observed data is the most probable over all alternatives.

MLE for Logistic Regression

We will make the assumption that our data are **independent and identically distributed** (iid).

This means that each output y_i depends only on its matching input $\mathbf{x}_i$ and the probability distribution by which we get outputs from inputs does not change from data point to data point.

MLE for Logistic Regression

We will make the assumption that our data are **independent and identically distributed** (iid).

This means that each output y_i depends only on its matching input $\mathbf{x}_i$ and the probability distribution by which we get outputs from inputs does not change from data point to data point.

A key and often made assumption is that the data samples are independent and identically distributed or iid.
This means that each pair of inputs and outputs is independent of one another. So y_1 only depends on x_1, y_2 on x_2 etc. The outputs do not depend on multiple inputs nor on other outputs. This assumption is consistent with reality when each sample corresponds to an effective repeat of an experiment or process under the same experimental conditions so the probability of outcomes given the same inputs remains unchanged.
iid assumptions are therefore not valid when there are dependencies in the data, such as if the data is collected over time for example and there might be a time effect, or if the data collection device or system changes as you collect data.

MLE for Logistic Regression

Under this assumption, we can factorise the likelihood:

$$
\begin{aligned}
L(\mathbf{w}|D) &= p(y_1, y_2 \ldots, y_{n-1}, y_n|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{w}), \\
&= p(y_1|\mathbf{x}_1, \mathbf{w}) \times p(y_2|\mathbf{x}_2, \mathbf{w}) \times \cdots \times p(y_n|\mathbf{x}_n, \mathbf{w}), \\
&= \underbrace{\prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w})}_{\text{Shorthand for the product}}
\end{aligned}
$$

MLE for Logistic Regression
Under this assumption, we can factorise the likelihood:
$L(\mathbf{w}|D) = p(y_1, y_2 \ldots, y_{n-1}, y_n|\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{n-1}, \mathbf{x}_n, \mathbf{w})$,
$= p(y_1|\mathbf{x}_1, \mathbf{w}) \times p(y_2|\mathbf{x}_2, \mathbf{w}) \times \cdots \times p(y_n|\mathbf{x}_n, \mathbf{w})$,
$= \underbrace{\prod_{i=1}^{n} p(y_i|\mathbf{x}_i, \mathbf{w})}_{\text{Shorthand for the product}}$

Using iid assumptions, we can therefore simplify our likelihood.
Remember the likelihood is simply a conditional joint distribution, and under iid assumptions, we are saying that we can factorise this joint distribution into a number of components corresponding to each pair of data.
Each component is the same probability distribution because they are independent but identically distributed.

MLE for Logistic Regression

It is often more mathematically convenient to use the logarithm of the likelihood (log-likelihood):

$$\log L(\mathbf{w}|D) = \sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i, \mathbf{w})$$

Note - the log changes the product to a sum and is taken inside the summation.

MLE for Logistic Regression
It is often more mathematically convenient to use the logarithm of the likelihood (log-likelihood):
$\log L(\mathbf{w}|D) = \sum_{i=1}^{n} \log p(y_i|\mathbf{x}_i, \mathbf{w})$
Note - the log changes the product to a sum and is taken inside the summation.

It is often more mathematically convenient to work with the logarithm of the likelihood or the log-likelihood.
The logarithm is a monotonic transformation so it doesn't change the point at which the likelihood is maximised so finding theparameters which lead to the maximum likelihood or the maximum log-likehood will give you the same answer.
The logarithmic transformation changes our product of probability terms into a sum and the logarithm is brought inside the summation and we log each probability.

MLE for Logistic Regression

We now need to specify the form of the term $p(y_i|\mathbf{x}_i, \mathbf{w})$ which is given by:

$$p(y_i|\mathbf{x}_i, \mathbf{w}) = h(\mathbf{x}_i, \mathbf{w})^{y_i}(1 - h(\mathbf{x}_i, \mathbf{w}))^{(1-y_i)}$$

where

$$h(\mathbf{x}_i, \mathbf{w}) = \text{Logistic}(\mathbf{w}^t\mathbf{x}_i + w_0)$$

(we will refer to the latter as h_i for shorthand but do not forget that it is actually a function of $\mathbf{w}$ and $\mathbf{x}_i$!)

This type of probability distribution is formally known as a **Bernoulli** probability distribution.

MLE for Logistic Regression
We now need to specify the form of the term $p(y_i|\mathbf{x}, \mathbf{w})$ which is given by:
$p(y_i|\mathbf{x}_i, \mathbf{w}) = h(\mathbf{x}_i, \mathbf{w})^{y_i}(1 - h(\mathbf{x}_i, \mathbf{w}))^{(1-y_i)}$
where
$h(\mathbf{x}_i, \mathbf{w}) = \text{Logistic}(\mathbf{w}^t\mathbf{x}_i + w_0)$
(we will refer to the latter as h_i for shorthand but do not forget that it is actually a function of $\mathbf{w}$ and $\mathbf{x}_i$!)
This type of probability distribution is formally known as a **Bernoulli** probability distribution.

Finally, we have so far ignored exactly what probability model we are using for the probability of each output given the input and parameters.
The form we will use is the Bernoulli distribution and the reason we choose it is that it encodes exactly what we specified in our generative process earlier.

MLE for Logistic Regression

It mathematically corresponds to saying $y_i = 1$ with probability h_i and $y_i = 0$ with probability $1 - h_i$.

You can check this by setting $y_i = 1$ or $y_i = 0$:

$$\begin{cases} y_i = 1: & p(y_i = 1|\mathbf{x}_i, \mathbf{w}) = h_i^1 \cancel{(1 - h_i)^{(1-1)}}, \\ y_i = 0:: & p(y_i = 0|\mathbf{x}_i, \mathbf{w}) = \cancel{h_i^0} (1 - h_i)^{(1-0)}, \end{cases}$$

MLE for Logistic Regression
It mathematically corresponds to saying $y_i = 1$ with probability h_i and $y_i = 0$ with probability $1 - h_i$.
You can check this by setting $y_i = 1$ or $y_i = 0$:
$\begin{cases} y_i = 1: & p(y_i = 1|\mathbf{x}, \mathbf{w}) = h_i^1 \cancel{(1 - h_i)^{(1-1)}}, \\ y_i = 0:: & p(y_i = 0|\mathbf{x}, \mathbf{w}) = \cancel{h_i^0} (1 - h_i)^{(1-0)}, \end{cases}$

This is because it mathematically corresponds to saying that $y = 1$ with probability h and $y = 0$ with probability $1 - h$.
You can try this as an exercise.
Set $y = 1$ and the term involving $(1 - h)$ is removed because the exponent is zero.
Set $y = 0$ and the term involving h is removed.

MLE for Logistic Regression

This simplifies the log-likelihood to the following form:

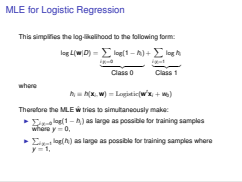
$$\log L(\mathbf{w}|D) = \underbrace{\sum_{i:y_i=0} \log(1-h_i)}_{\text{Class 0}} + \underbrace{\sum_{i:y_i=1} \log h_i}_{\text{Class 1}}$$

where

$$h_i \equiv h(\mathbf{x}_i, \mathbf{w}) = \text{Logistic}(\mathbf{w}^t \mathbf{x}_i + w_0)$$

Therefore the MLE $\hat{\mathbf{w}}$ tries to simultaneously make:

- $\sum_{i:y_i=0} \log(1-h_i)$ as large as possible for training samples where $y=0$,
- $\sum_{i:y_i=1} \log(h_i)$ as large as possible for training samples where $y=1$,



The result is that this simplifies the log-likelihood to the following form which involves summing terms $1-h_i$ for all data where the output $y=0$ and summing h_i for all data where $y=1$.
So when we try to find the MLE for $\mathbf{w}$ what we are trying to do is to set the h's which depend on both $\mathbf{x}$ and $\mathbf{w}$ in such a way that we maximise the probability of seeing the $y=0$ and $y=1$ training samples.

Implementing maximum likelihood estimation

There is no *closed form* solution for the MLE $\hat{\mathbf{w}}$ for a logistic regression model.

Numerical methods are required such as (stochastic) gradient descent.

The loss function is the log-likehood function.

In machine learning, the loss function for a logistic regression model can sometimes be known as the **cross-entropy loss**.

Implementing maximum likelihood estimation

There is no *closed form* solution for the MLE $\hat{\mathbf{w}}$ for a logistic regression model.

Numerical methods are required such as (stochastic) gradient descent.

The loss function is the log-likehood function.

In machine learning, the loss function for a logistic regression model can sometimes be known as the **cross-entropy loss**.

It turns out that we cannot algebraically solve to find the MLE of $\mathbf{w}$ in closed form.
It is necessary to use numerical methods and in this lecture we will turn to stochastic gradient descent as we considered previously.
Other methods can also be applied in the case of logistic regression such as iteratively reweighted least squares.
However, stochastic gradient descent is universally applicable across many types of probabilistic models and classifiers, even if for any particular model, it may not be the most efficient approach.
To apply stochastic gradient descent we need a loss function and in this case our loss is the likelihood function (or log likelihood).
Note that in machine learning and some programming libraries, the loss function for a logistic regression model maybe referred to as a cross entropy loss. This is the same thing but the name arises due to historical reasons where people from different disciplines have applied different techniques to end up with the same result.

SGD for Logistic Regression

Loss function (minimisation $\Leftrightarrow$ negative log-likelihood):

$$l(\mathbf{w}) = -\frac{1}{n}\log L(\mathbf{w}|D) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h(\mathbf{w}, \mathbf{x}_i) + (1 - y_i)\log(1 - h(\mathbf{w}, \mathbf{x}_i)]$$

For SGD, we sample one of the data samples $(\mathbf{x}_i, y_i)$ to estimate the gradient:

$$\frac{\partial l(\mathbf{w})}{\partial w_j} = -\left[y_i \frac{1}{h(\mathbf{w}, \mathbf{x}_i)} - (1 - y_i)\frac{1}{1 - h(\mathbf{w}, \mathbf{x}_i)}\right]\frac{\partial h(\mathbf{w}, \mathbf{x}_i)}{\partial w_j}$$

SGD for Logistic Regression
Loss function (minimisation ⇔ negative log-likelihood):
$l(\mathbf{w}) = -\frac{1}{n}\log L(\mathbf{w}|D) = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log h(\mathbf{w}, \mathbf{x}_i) + (1 - y_i)\log(1 - h(\mathbf{w}, \mathbf{x}_i)]$
For SGD, we sample one of the data samples $(\mathbf{x}_i, y_i)$ to estimate the gradient:
$\frac{\partial l(\mathbf{w})}{\partial w_j} = -\left[y_i \frac{1}{h(\mathbf{w}, \mathbf{x}_i)} - (1 - y_i)\frac{1}{1 - h(\mathbf{w}, \mathbf{x}_i)}\right]\frac{\partial h(\mathbf{w}, \mathbf{x}_i)}{\partial w_j}$

If we are doing loss minimisation, we will need to use the negative of the log-likelihood as our loss function. It is also conventional to divide by n to standardise the loss so that its magnitude does not depend on the size of the data.

For stochastic gradient descent, we need to compute the derivative of the loss function with respect to each of the parameters w_j.

Remember that with stochastic gradient descent, we only use a subset of the data at any time, possibly as little as one data sample and we will adopt that here for this derivation.

SGD for Logistic Regression

Applying the chain rule for differentiation:

$$\begin{aligned}\frac{\partial h(\mathbf{w}, \mathbf{x}_i)}{\partial w_j} &= \frac{\partial}{\partial w_j}\left[\frac{1}{1+\exp(-[\mathbf{w}^t\mathbf{x}_i + w_0])}\right], \\ &= \exp(-[\mathbf{w}^t\mathbf{x}_i + w_0])[1+\exp(-[\mathbf{w}^t\mathbf{x}_i + w_0])]^{-2}x_i^j, \\ &= h(\mathbf{w}, \mathbf{x}_i)(1-h(\mathbf{w}, \mathbf{x}_i))x_i^j\end{aligned}$$

So,

$$\begin{aligned}\frac{\partial l(\mathbf{w})}{\partial w_j} &= -\left[y_i\frac{1}{h(\mathbf{w}, \mathbf{x}_i)} - (1-y_i)\frac{1}{1-h(\mathbf{w}, \mathbf{x}_i)}\right]h(\mathbf{w}, \mathbf{x}_i)(1-h(\mathbf{w}, \mathbf{x}_i))x_i^j, \\ &= -[y_i(1-h(\mathbf{w}, \mathbf{x}_i)) + (1-y_i)h(\mathbf{w}, \mathbf{x}_i)]x_i^j, \\ &= -[y_i - h(\mathbf{w}, \mathbf{x}_i)]x_i^j\end{aligned}$$

SGD for Logistic Regression

We can apply the chain rule for differentiation to the derivative of h with respect to w_j to derive the various quantities required.
In this lecture, I will not go through the detailed derivations, but please read through these notes and the supporting materials online.
You will need to be comfortable with some aspects of these algebraic manipulations for your assessment and examples of the types of things required will be given in the supporting exercises.

SGD for Logistic Regression

The general SGD update is given by:

$$w_j \leftarrow w_j - \lambda \frac{\partial l(\mathbf{w})}{\partial w_j}$$

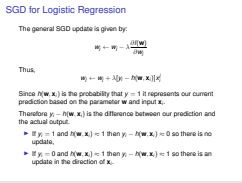
Thus,

$$w_j \leftarrow w_j + \lambda[y_i - h(\mathbf{w}, \mathbf{x}_i)]x_i^j$$

Since $h(\mathbf{w}, \mathbf{x}_i)$ is the probability that $y = 1$ it represents our current prediction based on the parameter $\mathbf{w}$ and input $\mathbf{x}_i$.

Therefore $y_i - h(\mathbf{w}, \mathbf{x}_i)$ is the difference between our prediction and the actual output.

- If $y_i = 1$ and $h(\mathbf{w}, \mathbf{x}_i) \approx 1$ then $y_i - h(\mathbf{w}, \mathbf{x}_i) \approx 0$ so there is no update,
- If $y_i = 0$ and $h(\mathbf{w}, \mathbf{x}_i) \approx 1$ then $y_i - h(\mathbf{w}, \mathbf{x}_i) \approx 1$ so there is an update in the direction of $\mathbf{x}_i$.



Remember that the general form of a SGD update is to take the current value of the parameter and then to subtract off the gradient scaled by a learning rate in this case denoted by λ.
The derivation here shows that the gradient is given by the difference between the output y and h which is the predicted probability that $y = 1$ scaled by the input.
This means we update each parameter by moving slightly in the direction of the input scaled by both the learning rate and also how close our current predictions are to the actual output.
So, for example, if the true output $y = 1$ and we predict $h = 1$ then the difference is zero and there is rightly no update. We have got it right.
If though, the true output is $y = 0$ and we predict $h = 1$ which is the same as predicting that $y = 1$, there is a big difference so we update that parameter in the direction of the input.

Linear decision boundaries

Like the perceptron algorithm, logistic regression leads to **linear decision boundaries**.

Recall that:

$$p(y = 1|\mathbf{x}, \mathbf{w}) = h(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(-[\mathbf{w}^t\mathbf{x} + w_0])}$$

Our decision rule could be that:

- If $p(y = 1|\mathbf{x}, \mathbf{w}) < d$ then choose $y = 1$,
- else if $p(y = 1|\mathbf{x}, \mathbf{w}) > d$ then choose $y = 0$,

where d denotes the probability threshold at which we would choose one class over another (note - d does not have to be 0.5!).

Linear decision boundaries
Like the perceptron algorithm, logistic regression leads to **linear decision boundaries**.
Recall that:
$$p(y = 1|\mathbf{x}, \mathbf{w}) = h(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + \exp(-[\mathbf{w}^t\mathbf{x} + w_0])}$$
Our decision rule could be that:
- If $p(y = 1|\mathbf{x}, \mathbf{w}) < d$ then choose $y = 1$,
- else if $p(y = 1|\mathbf{x}, \mathbf{w}) > d$ then choose $y = 0$,

where d denotes the probability threshold at which we would choose one class over another (note - d does not have to be 0.5!).

Once we have fitted a logistic regression model, in the same way that the perceptron algorithm gives us linear decision boundaries so does logistic regression. Recall that once we have estimated parameters $\mathbf{w}$ we can use h to estimate the probability that $y = 1$ for any input $\mathbf{x}$.
A decision rule can be constructed such that whenever this probability is within a range, we choose $y = 1$ and otherwise we choose $y = 0$.
The threshold does not necessarily have to be a probability of 0.5.

Linear decision boundaries

Rearranging:

$$d = \frac{1}{1 + \exp(-[\mathbf{w}^t\mathbf{x} + w_0])},$$
$$\Rightarrow \mathbf{w}^t\mathbf{x} + w_0 = \log\left(\frac{d}{1-d}\right)$$

Given d, the relationship of the decision boundary with $\mathbf{x}$ is linear.

Linear decision boundaries
Rearranging:
$d = \frac{1}{1 + \exp(-[\mathbf{w}^t\mathbf{x} + w_0])}$
$\Rightarrow \mathbf{w}^t\mathbf{x} + w_0 = \log\left(\frac{d}{1-d}\right)$
Given d, the relationship of the decision boundary with $\mathbf{x}$ is linear.

We can rearrange this expression to show that given a decision threshold d, the shape of the decision boundary is given by a linear expression in the inputs $\mathbf{x}$.

Linear decision boundaries

To illustrate, now suppose we have two inputs x_1 and x_2:

$$w_1 x_1 + w_2 x_2 + w_0 = \log\left(\frac{d}{1-d}\right),$$
$$x_2 = \frac{1}{w_2}\left[\log\left(\frac{d}{1-d}\right) - w_0 - w_1 x_1\right]$$

Thus, the relationship between x_1 and x_2 along the decision boundary is given by a straight line!

We can look at a more explicit example. Suppose there are two inputs x_1 and x_2, rearranging this expression to give x_2 in terms of x_1 gives us an equation which describes a straight line!

Linear decision boundaries

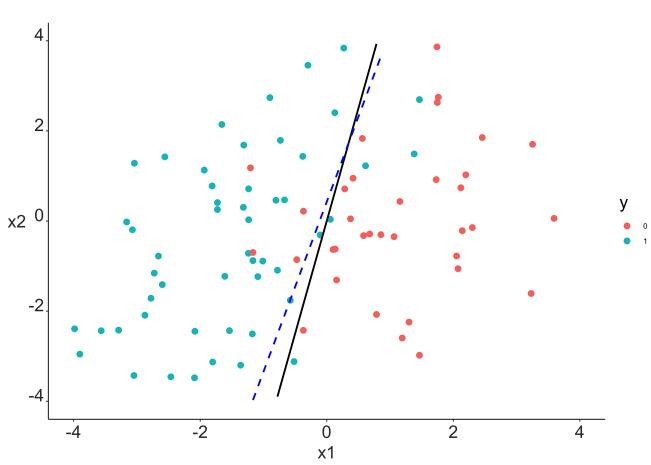


Figure: (Black) True decision boundary ($d = 0.5$). (Blue) Estimated decision boundary.

Here is an illustrated example showing the true and estimated decision boundaries for this data set.
Everything to the left of the decision boundary will be classified as Class 1 whilst everything to its right will be noted as Class 0.
In this case the data were simulated to be linearly separable and so logistic regression does well to find a linear separating plane.
If the data are not separable then nonlinear techniques are required and these will be explored later in the course.

Linear decision boundaries

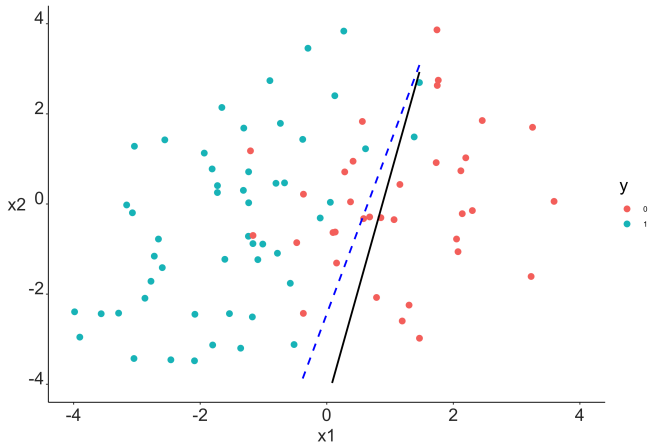


Figure: (Black) True decision boundary ($d = 0.1$). (Blue) Estimated decision boundary.

Now if I change the decision threshold d from 0.5 to 0.1, the decision boundary shifts dramatically to the right.
This is because we have changed the decision rule to allow low probability predictions for $y = 1$ to be counted as class 1.
Therefore the decision boundary shifts to ensure as many of the possible class 1s are covered at the expense of more errors for class 0s.

Practical Challenges

Logistic regression (and many other classification algorithms) can have problems dealing with a range of scenarios:

1. Imbalanced data
2. Multicollinearity
3. Completely separated training data

Practical Challenges

Logistic regression (and many other classification algorithms) can have problems dealing with a range of scenarios:

1. Imbalanced data
2. Multicollinearity
3. Completely separated training data

Logistic regression and many other classification algorithms can have problems dealing with a range of scenarios including imbalanced data, multicollinearity and completely separated data.
The latter is a peculiar one which happens because, as a probabilistic algorithm, if logistic regression does not have training data which contains classification ambiguity, it cannot calibrate it probabilistic output correctly.
Lets examine the other problems in turn.

Imbalanced data

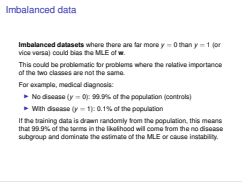
Imbalanced datasets where there are far more $y = 0$ than $y = 1$ (or vice versa) could bias the MLE of $\mathbf{w}$.

This could be problematic for problems where the relative importance of the two classes are not the same.

For example, medical diagnosis:

- No disease ($y = 0$): 99.9% of the population (controls)
- With disease ($y = 1$): 0.1% of the population

If the training data is drawn randomly from the population, this means that 99.9% of the terms in the likelihood will come from the no disease subgroup and dominate the estimate of the MLE or cause instability.



Imbalanced datasets occur where we have many more class 0s than class 1s or vice versa. As a consequence, an imbalance dataset might bias the MLE for $\mathbf{w}$ because the MLE will try to fit the dominant class rather than the rarer one.

Consider a medical prediction problem where the dominant group of patients do not have a disease condition. If we draw our training data randomly from the population, we will have lots of controls but few disease cases.

This means if we try to find an MLE for $\mathbf{w}$ from this data, it is will be skewed towards predicting the no disease status well whereas we might actually be more concerned about people with the disease!

In cases where there are very extreme data imbalances, the algorithms we might adopt to find an MLE might also fail due to instabilities.

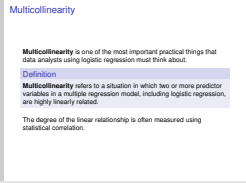
Multicollinearity

Multicollinearity is one of the most important practical things that data analysts using logistic regression must think about.

Definition

Multicollinearity refers to a situation in which two or more predictor variables in a multiple regression model, including logistic regression, are highly linearly related.

The degree of the linear relationship is often measured using statistical correlation.



Multicollinearity is an important practical consideration that data analysts using logistic regression must think about.
It refers to a situation in which two or more predictors variables in a multiple regression model, including logistic regression, are highly linearly related.
The degree of the linear relationship is often measured using statistical correlation.

Multicollinearity

Recall our specification of the decision boundary with two input variables:

$$w_1 x_1 + w_2 x_2 + w_0 = \log\left(\frac{d}{1-d}\right)$$

Now consider if $x_2 = \alpha x_1$:

$$\begin{aligned} w_1 x_1 + w_2(\alpha x_1) + w_0 &= (w_1 + \alpha w_2)x_1, \\ &= (w_1/\alpha + w_2)x_2, \end{aligned}$$

There is really only one predictor but if I use two predictors in my model fitting, my estimation algorithm will not be able to deduce this redundancy and will try to incorporate both.

Feature selection is an important step in ensuring we remove redundant predictors to mitigate against multicollinearity effects.

Multicollinearity
Recall our specification of the decision boundary with two input variables:
$$w_1 x_1 + w_2 x_2 + w_0 = \log\left(\frac{d}{1-d}\right)$$
Now consider if $x_2 = \alpha x_1$:
$$\begin{aligned} w_1 x_1 + w_2(\alpha x_1) + w_0 &= (w_1 + \alpha w_2)x_1, \\ &= (w_1/\alpha + w_2)x_2, \end{aligned}$$
There is really only one predictor but if I use two predictors in my model fitting, my estimation algorithm will not be able to deduce this redundancy and will try to incorporate both.
Feature selection is an important step in ensuring we remove redundant predictors to mitigate against multicollinearity effects.

Recall our specification of the decision boundary with two input variables.
If x_2 is a linear function of the other variable x_1 then there is effectively only one predictor.
However, if we include both variables in our model, a logistic regression model fit could use both, or either one of the parameter to define the same decision boundary.
It will not be able to deduce the redundancy and therefore different models could lead to the same outputs.
Feature selection is therefore important for ensuring we remove redundant predictors.

Utility of logistic regression

Logistic regression is a classic probabilistic approach for binary classification problems used in both Statistics and Machine Learning:

- Efficient and straightforward,
- Doesn't require large computation,
- Easy to implement, easily interpretable
- Used widely by data analyst and scientist.
- Provides a probability for predictions and observations.

Utility of logistic regression

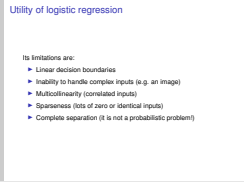
Logistic regression is a classic probabilistic approach for binary classification problems used in both Statistics and Machine Learning:
- Efficient and straightforward,
- Doesn't require large computation,
- Easy to implement, easily interpretable
- Used widely by data analyst and scientist.
- Provides a probability for predictions and observations.

So to end, logistic regression is a probabilistic approach for tackling binary classification problems.
It has roots in both statistics and machine learning and is a standard model for anyone learning about these topics and is part of the standard arsenal of modern data analysis.
It is a lightweight model that be efficiently implemented and is easy to interpret.

Utility of logistic regression

Its limitations are:

- Linear decision boundaries
- Inability to handle complex inputs (e.g. an image)
- Multicollinearity (correlated inputs)
- Sparseness (lots of zero or identical inputs)
- Complete separation (it is not a probabilistic problem!)



However, it is a linear classifier so it cannot handled complex classification problems.
It is best suited for low-dimensional problems with few inputs and so is not suitable for processing high-dimensional data like images.
If the inputs are correlated, this introduces an effect called multicollinearity which can make fitting the model unstable.
Sparse input data can also caused problems as well as unbalanced or completely separated data.

END LECTURE