# Week 2 – Learning Parameters of Logistic Regression

**Goal: Learning parameters of logistic regression**

**Intuition behind maximum likelihood estimation**

The quality metric is called the Maximum Likelihood estimation. Larger likelihood values indicate a better set of coefficients. We are looking to maximize the likelihood estimate.

We want to find a set of coefficients, w, that make P(y = +1|x) go to 1.0 for training data with positive sentiment and make P(y = +1|x) go to 0 for training data with negative sentiment. Of course, we won’t get perfect 1’s or 0’s for each row in the data.

**Data Likelihood**

Given this data;



We want to find w that will cause us to match the output values y by maximizing the associated probability.



The final set of coefficients will based on combination of each row’s probabilities. Probabilities are combined as a product (since they are independent events).

Explicitly for the data set in the table above;

Or more succinctly

Or more succinctly

Or most succinctly

**Finding best linear classifier with gradient ascent**

The goal is to pick w that maximizes *l(****w****)* over all possible w.

**Review of gradient ascent**

Basically, we want to iterate on the gradient and move in the direction of the gradient until the gradient goes to within some tolerance of zero. This works because our goal is a concave curve, so the slope is zero at the maximum of the curve.

While not converged

estimated coefficient in step t

estimated coefficient in step t+1

means “computed at ”, so specifying that the derivative of the likelihood function with respect to w is calculated with w(t).

is the step size, which scales the derivative before it is added to the coefficient to get the estimated coefficient for the next step.

The maximum is when , but in practice we stop when we get within some tolerance of zero;

converged =

In higher dimensional spaces we use the gradient, which is a vector of partial derivatives, one for each coefficient;

So the gradient of the likelihood function is a D+1 dimensional vector where D is the number of features (not counting the constant feature). In this form, the gradient ascent algorithm is;

While not converged

And we converge when all partial derivatives are within a tolerance of zero.

converged =

In fact, rather than the likelihood function, we use log-likelihood function. From [Wikipedia](https://en.wikipedia.org/wiki/Likelihood_function#Log-likelihood):

For many applications, the [natural logarithm](https://en.wikipedia.org/wiki/Natural_logarithm) of the likelihood function, called the **log-likelihood**, is more convenient to work with. Because the logarithm is a [monotonically increasing](https://en.wikipedia.org/wiki/Monotonically_increasing) function, the logarithm of a function achieves its [maximum](https://en.wikipedia.org/wiki/Maximum) value at the same points as the function itself, and hence the log-likelihood can be used in place of the likelihood in [maximum likelihood](https://en.wikipedia.org/wiki/Maximum_likelihood)estimation and related techniques. Finding the maximum of a function often involves taking the [derivative](https://en.wikipedia.org/wiki/Derivative) of a function and solving for the parameter being maximized, and this is often easier when the function being maximized is a log-likelihood rather than the original likelihood function.

For example, some likelihood functions are for the parameters that explain a collection of statistically independent observations. In such a situation, the likelihood function factors into a product of individual likelihood functions. The logarithm of this product is a sum of individual logarithms, and the [derivative](https://en.wikipedia.org/wiki/Derivative) of a sum of terms is often easier to compute than the derivative of a product. In addition, several common distributions have likelihood functions that contain products of factors involving [exponentiation](https://en.wikipedia.org/wiki/Exponentiation). The logarithm of such a function is a sum of products, again easier to differentiate than the original function.

The partial derivative of the log-likelihood with respect for feature j is given by;

N is the number of data points (rows of feature matrix H)

i is the index of the data (the ith data input, ith row of feature matrix H)

is the jth feature of the ith data input

is the prediction that xi is positive for the given set of coefficients w.

is the indicator function. This outputs 1 if the known output value yi is labeled as +1 and 0 if yi is labeled as -1.

This part of the derivative is the difference between the truth (known value of y) and the prediction that this is a positive sample.

This is then weighted by the feature value (which is the word count for the jth word in the bag of words if we are doing sentiment analysis). So if the count is zero, the does not ending up counting at all. If the word count is high, then any difference is heavily weighted.

So when the weighted difference for each feature prediction falls below the tolerance (when it gets close to zero), then the algorithm converges.

**Example of computing derivative for logistic regression**

Remembering the indicator function (see above) and the score function;

and that

This part of the derivative;

is then

We apply each feature value (each word count) to this expression and sum these to get the gradient of the log-likelihood function for that input. To show this, we can build a table that calculates the contribution of each term (the w0 term does not contribute because w0 is zero, so we only look at w1 and w2 terms in the table).

****

**Interpreting derivative for logistic regression**

The partial derivative of the log-likelihood with respect for feature j is given by;

We can call the term is inside the summation the delta for the coefficient – it is how much the coefficient is changed in each iteration of the gradient ascent;

If we say for the sake of illustration that , so that we can concentrate on the other term within the summation, we can build a table of how that term affects the gradient and so how the coefficient is changed;



We can see in the two cases where the prediction matches the truth that we will not make any change in the coefficient. That is good because our coefficients already predict the truth – we don’t want them to change. But look at the case in the upper-right where the truth is positive, but our prediction is negative. In this case the delta is positive; it will cause the coefficient to increase, which is what we want. Also look at the case in the lower-left where the truth is negative, but we have predicted positive; in that case the negative delta would have us decrease the coefficient, which is again what we want.

**Summary of gradient ascent for logistic regression**

Gradient Ascent for Logistic Regression

At t=1, initialize , or some other smart choice.

while do

for j = 0..D

partial[j]=

t = t + 1

N is the number of data points (rows of feature matrix H)

i is the index of the data (the ith data input, ith row of feature matrix H)

is the jth feature of the ith data input

is the prediction that xi is positive for the given set of coefficients w at iteration t.

is the indicator function. This outputs 1 if the known output value yi is labeled as +1 and 0 if yi is labeled as -1.

*𝜂* is the step size, which scales the derivative before it is added to the coefficient to get the estimated coefficient for the next step.

More plainly;

partial[j]=

and this term can be computed once for the ith input row and then applied to each feature in the row;

**Choosing step size**

It turns out that the algorithm is very sensitive to this, so it is something that has to be chosen carefully. If the step size is too small, then the gradient ascent algorithm will require a lot of iterations to converge. If the step size is large, the convergence will not be smooth – it will look early on a like a sawtooth oscillation, but eventually smooth out and converge.

**Careful with step sizes that are too large**

If the step size is too large, early oscillations become very large and curve will continue to oscillate and so may not converge. In effect, if the step size is too large, then we may jump past, back and forth around the goal but never converge on it. Another aspect of step sizes that are too large is that their log-likelihood does not maximize, so it stays below those step sizes that converge.

**Rule of thumb for choosing step size**

* Try several sizes that are exponentially spaced. For instance, try 10-4 and 10-6.
* Goal
  + Find a step size that is too small (converges smoothly but slowly)
  + Find a step size that is too large (oscillates and does not converge)
  + Now look within this range for one that leads to the best training data likelihood.
* Advanced
  + Use the above procedure to find a starting step size , but decrease the stepsize as the algorithm progresses.
  + For example, scale step size based on iteration;