Chapter 6
Logistic Regression

Slides (pdf) Video (Panopto)

6.1 MSE / Convexity

In previous chapters we have discussed how mean squared error (MSE) can be used to create machine learning regression models which can perform quite well. However, in classification MSE is not possible because the task is not convex, continuous or differentiable. In addition, there is not a closed form solution, like in regression. This means that algorithms like gradient descent won't be able to find an optimal set of weights for the discrete results.

Definition 6.1: Convexity

A function is considered **convex** if a line segment between any two points of the function does not lie below the graph.

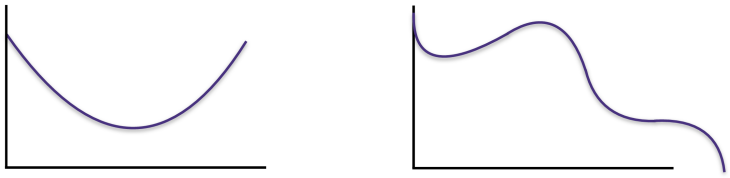


Figure 6.1: Convex (right) and non convex (left) functions.

Without a convex function, gradient descent might only be able to find a local minimum for the function or be unable to find any minimum whatsoever. Imagine taking gradient descent on the left non convex function shown in Figure 6.1. In this scenario, we would never reach a global minimum.

In classification with MSE, the three requirements of a convex, continuous and differentiable function are not met. The solutions are discrete values without any ordinal nature so we need to transform our problem into something where we can measure how close a prediction is to some category, rather than directly outputting that category from our model. One way we can do this is by measuring error using probabilities. Probabilities are continuous and can help measure the level of certainty we might have for a prediction.

Example(s)

For example, let us imagine we are creating a model to predict the sentiment of a restaurant review where a positive review maps to +1 and a negative review maps to -1. Ideally the trained model would be able to predict a review like "The sushi everything else were awesome!" positively.

$$P(y = +1 \mid x) = \text{“The sushi and everything else were awesome!”}) = 0.99$$

Then, using probabilities for reviews that have a higher degree of uncertainty regarding the sentiment, our model could recognize that.

$$P(y = -1 \mid x) = \text{“The sushi was alright and service was okay.”}) = 0.50$$

With probabilities the certainty of an event is now known but the results must be mapped back to the classifier to make a choice. We can assign values based on if the resulting probability is greater than 0.50 to be the output. With probabilities it is easy to recognize when a model has a high degree of certainty (high probabilities). In addition, we can compare what types of inputs lead to uncertainty for the model (mid-range probabilities).

6.2 Logistic Regression Model

The **sigmoid function** takes arbitrarily large and small numbers then maps them between 0 and 1. We can input a score to this function and receive a probability so that we will be able to take gradient descent to train the model. The use of the sigmoid function in this way is called the logistic regression model.

Definition 6.2: Logistic Regression Classifier

$$P(y = +1 \mid x_i, w) = sigmoid(Score(x_i)) = \frac{1}{1 + e^{-w^T h(x_i)}}$$

This model allows its objective function to be trained such that $P(y = +1 \mid x_i, w)$ can be maximized for some input x and weights w. The goal is to maximize correct behavior / classification and minimize incorrect behavior of the model. Also, it is important to remember that in a binary classification, or for any number of classes, the probabilities must sum to 1. For binary classification $P(y = -1 \mid x_i, w) = 1 - P(y = +1 \mid x_i, w)$. We can combine these terms and use the logistic regression definition to show this:

$$\begin{aligned} P(y = -1 \mid x_i, w) &= 1 - P(y = +1 \mid x_i, w) && (6.1)\\ &= 1 - \frac{1}{1 + e^{-w^T h(x_i)}} && (6.2)\\ &= \frac{e^{-w^T h(x_i)}}{1 + e^{-w^T h(x_i)}} && (6.3) \end{aligned}$$

In order to train our model we want to maximize the likelihood function. The likelihood function in this case is:

Definition 6.3: Likelihood Function for Logistic Regression

$$\ell(w) = \prod_{i}^{n} P(y^{(i)} | x^{(i)}, w)$$

This seems different from regression where previously we wanted to minimize our error from the objective function by taking the argmin respective to the weights. However, in this case we are maximizing (argmax respective to weights) the likelihood function (maximizing the likelihood of a correct decision) is essentially the same as minimizing error.

$$\hat{w} = \underset{w}{\text{argmax}}\, \ell(w) \tag{6.4}$$

$$= \underset{w}{\text{argmax}}\, \frac{1}{n} \prod_{i=1}^{n} P(y^{(i)}|x^{(i)}, w) \tag{6.5}$$

$$= \underset{w}{\text{argmax}}\, \frac{1}{n} \sum_{i=1}^{n} \log(P(y^{(i)}|x^{(i)}, w)) \quad \text{(From product to sum of logs)} \tag{6.6}$$

$$= \underset{w}{\text{argmin}}\, \frac{1}{n} \sum_{i=1}^{n} -\log(P(y^{(i)}|x^{(i)}, w)) \quad \text{(Negative Log-Likelihood Loss)} \tag{6.7}$$

When we adjust our weights to find this point it is called the Maximum Likelihood Estimate (MLE). Remember that maximizing our likelihood is the same as minimizing our error for our model so that we have a high probability for a correct prediction and a low probability for incorrect (Definition 6.3).

Definition 6.4: Negative Log-Likelihood Loss Function

$$L(w) = \sum_{i=1}^{n} -\log(P(y^{(i)}|x^{(i)}, w))$$

Thus far, everything discussed for classification has been a binary +1 or -1 classification but we can also represent multi-class classification problems. Previously we used a sigmoid function to turn the score of some input into a probability, which then uses a Negative Log-Likelihood to measure the loss for the binary classification. For multi-class classification we will use the **softmax** function instead of the sigmoid function. We will still use the Negative Log-Likelihood to measure the loss. The combination of these two is called **cross-entropy loss** which we will cover in more depth later in deep learning.

Just like in regression, the logistic regression model can predict simple problems quite well if there is an easy linear separation between points. But like regression again we can model more complicated problems by including more features and/or more complex features.

Example(s)

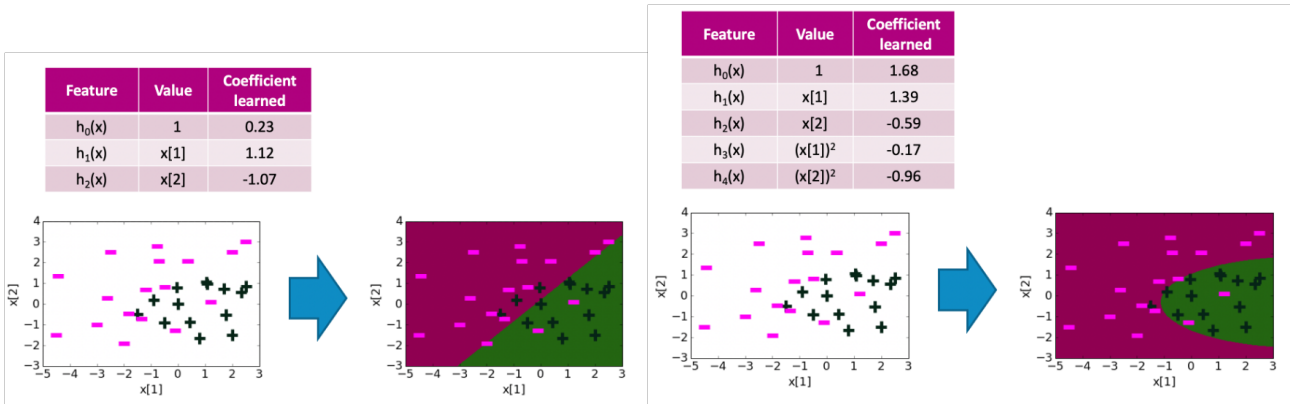
For instance, instead of only using two words as features for our sentiment review classifier:

$$h_1(x) = \text{number of "awesome" words}$$
$$h_2(x) = \text{number of "awful" words}$$

We could use more features with 2 being more complex, like this:

$$h_1(x) = \text{number of "awesome" words}$$
$$h_2(x) = \text{number of "awful" words}$$
$$h_3(x) = (\text{number of "awesome" words})^2$$
$$h_4(x) = (\text{number of "awful" words})^2$$

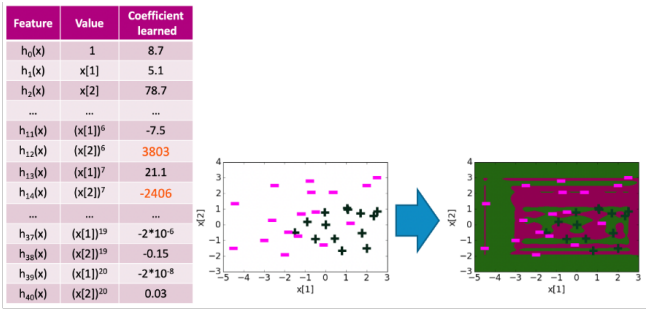
However, it is important to remember that just like regression a logistic regression model can overfit just like in regression. To avoid this, just as before, we can use regularization techniques including the L1 and L2 norms which can change our quality metric to avoid overfitting, and in the case of L1, aid in feature selection. The coefficient paths will follow similar trajectories so the same ideas of comparing models with different hyperparameters and regularization techniques using only MSE on validation data still applies.



Feature	Value	Coefficient learned
$h_0(x)$	1	0.23
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07

Feature	Value	Coefficient learned
$h_0(x)$	1	1.68
$h_1(x)$	x[1]	1.39
$h_2(x)$	x[2]	-0.59
$h_3(x)$	$(x[1])^2$	-0.17
$h_4(x)$	$(x[2])^2$	-0.96

Figure 6.2: Simple and quadratic classification boundaries



Feature	Value	Coefficient learned
$h_0(x)$	1	8.7
$h_1(x)$	x[1]	5.1
$h_2(x)$	x[2]	78.7
...	...	...
$h_{11}(x)$	$(x[1])^6$	-7.5
$h_{12}(x)$	$(x[2])^6$	3803
$h_{13}(x)$	$(x[1])^7$	21.1
$h_{14}(x)$	$(x[2])^7$	-2406
...	...	...
$h_{37}(x)$	$(x[1])^{19}$	$-2*10^{-6}$
$h_{38}(x)$	$(x[2])^{19}$	-0.15
$h_{39}(x)$	$(x[1])^{20}$	$-2*10^{-8}$
$h_{40}(x)$	$(x[2])^{20}$	0.03

Figure 6.3: Overfit classification boundaries

It is important to remember that while the plots from above show the decision boundaries, we are still representing our result as a probability / certainty of the model's output. When using regularization, smoothness can be introduced to better represent areas of certainty and uncertainty. Increasing λ corresponding to increasing the amount of regularization, for L2 especially, will increase the amount of this white / uncertain area in an attempt to resist overfitting.

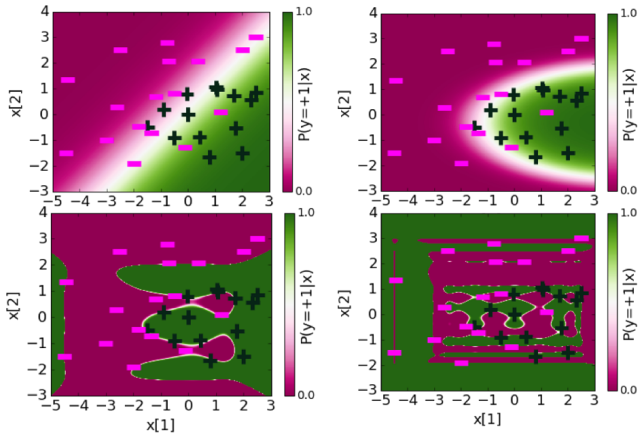


Figure 6.4: Regularized plotted probabilities

6.3 Gradient Descent

As previously stated, unlike a linear regression, we do not have a closed form solution for this problem. Instead, we must use an iterative method to find the global minimum for the function (equation 6.7) like gradient descent. In gradient descent, since we cannot compute the solution with a closed form, we must move towards the minimum step by step.

Algorithm 1 Gradient Descent

Start with random weights w
while w *has not converged* **do**
  $w = w - \alpha \Delta L(w)$
end
- α = Learning Rate
- $\Delta L(w)$ = Gradient of loss function on weights w

Remember that our learning rate is important as it controls how large of a step we take during gradient descent where a learning rate that is too small could take too long to converge but a large learning rate may result in gradient descent never reaching convergence. This process for choosing the learning rate as a hyperparameter often requires lots of trial and error but is necessary for a fast, yet reliable, gradient descent. Often we use values that are exponentially spaced out.

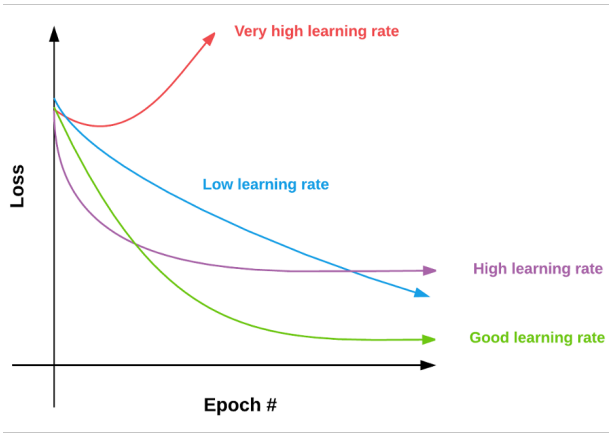


Figure 6.5: Effect of different learning rates.

Previously we discussed how different hyperparameters can affect our model's performance and what we should be looking for in hyperparameters. For fine-tuning hyperparameters, two methods are commonly used for finding the optimal values: **grid search** and **random search**

Definition 6.5: Grid Search

Start by defining a "grid" of hyperparameter values and evaluate the model at every position on the grid returning the argmin of the grid.

Definition 6.6: Random Search

Define a searching domain for possible hyperparameter values and randomly sample points in that domain evaluating the model each time.

Using these methods we can search for the hyperparameter or set of hyperparameters which results in the lowest error for the validation set. When we have multiple sets of hyperparameters to choose from in either grid or random search, the process is repeated so that every combination is tried together, usually involving nested for loops.

The learning rate in gradient descent does not need to be fixed, though. Ideally, gradient descent could descend quickly while still being reliable enough to reach some loss / weight value convergence. Note that we can either assess convergence by the lack of changing error or more ideally by the lack of changing weight values. We could start with large step sizes, then progressively lower the learning rate each epoch / iteration of gradient descent where t is the number of iterations.

$$\alpha_t = \frac{\alpha_0}{t} \tag{6.8}$$

To compute the gradient at each iteration for gradient descent (indicating which direction we should go) we have to calculate the loss from each data point to calculate the gradient of an update. This means that for large data sets especially, gradient descent can be very computationally expensive, even with optimal learning rates. One idea for avoiding this is to simply not calculate the loss for each data point at each iteration. Less work means our process can speed up! Instead, we can sample a small number of points from the training set, rather than the entire set, to calculate the gradient at each epoch to decrease computational time. This idea is called stochastic gradient descent. In practice, this is used widely as most often each step in gradient descent does not require the entire data set and by randomly sampling the model will still be able to learn well.

The number of points used at each step can vary widely from half of the data set to only 1 single data point. Clearly, 1 single data point is computationally faster, but might require more iterations to converge, or not converge at all if the data is particularly noisy. These batch sizes can change in size widely, largely depending on the noise and size of the data set being trained on and can be thought of as another hyperparameter similar to the learning rate.

In review of gradient descent and the logistic regression model, we have learned that

- Minimizing the error is the same as maximizing the likelihood in a logistic regression model.
- Predictions are made using probabilities.
- We can use the sigmoid function to turn a score into a probability.
- To prevent overfitting regularization should still be used.
- Gradient descent must be used as there is no closed form solution to our objective function.
- Learning rate in gradient descent are important for gradient descent convergence
- Stochastic gradient descent is a good option when working with large data sets.

In the following chapter other classification models will be discussed.