PY-599 (Fall 2018): Applied Artificial Intelligence

Logistic Regression

& Homework Assignment

Deadline to Submit: Tuesday 3:00 PM, Oct 9th (right before the Tuesday session)

Submission Method: Please share your Colab notebook

Group Submission is allowed

We talked about regression problems and regression models in the class. A regression problem is a problem where the goal is to predict continuous (real-valued) output from the inputs. A regression problem is a supervised learning problem in the sense that we are given a set of examples, or training data, (x_1,y_1), (x_2,y_2), (x_3,y_3),…, (x_m,y_m), where x is an example input, and y is the actual continuous output. Our goal in regression modeling is to come up with a model that fits this training data, and second, generalizes beyond this specific data set and predicts the outputs of new unseen input data as well.

We introduced linear regression models to map the input features to the continuous output:

$$\hat{y} = x_p.W \tag{1}$$

x_p is a (1,k+1) row vector that contains the inputs, W is a (k+1,1) column vector that contains the model parameters, and $\hat{y}$ is a continuous output that our model predicts. k is the number of features, or in other words, the number of inputs to the model. In linear regression model we have a bias parameter as well. For a more compact notation, we included an extra feature x_0 that is always 1, and we consider the bias parameter as another multiplicative parameter that is multiplied to this additional feature. That is how we ended up with a (1,k+1) row vector for the inputs, k of them are the inputs, and one is the constant feature 1. In the house-price example discussed in the class, the x input was:

$x_p=[1,x_1]$

As you see the first feature is 1, and the second feature is the size of the house.

Note that in Eq. (1) we used $\hat{y}$ notation instead of y itself to emphasize the fact that our model predicts or estimates the outputs, and theses estimates can be different from the actual outputs y. Our aim in training the model is to come up with a parameter vector W that minimizes the error; the difference, between the estimated outputs $\hat{y}$ and the actual outputs y that we get from the labeled training data. In the class we introduced mean squared error as a measure of error, and then we found a parameter set W that minimized this cost function. We analytically solved this optimization problem using the Normal equation, and then we followed a gradient descent method to solve the same optimization problem.

How about the classification problems? What if the problem is to estimate which class a given input belongs to, instead of producing a continuous output. We can modify the

linear regression models to solve the classification problems as well. We call the resulting model the Logistic Regression Model.
In Logistic regression, similar to linear regression, we are given a set of training data, *(x_1,y_1), (x_2,y_2), (x_3,y_3),…, (x_m,y_m)*. But this time *y* values are the class labels. They are not continuous, they are discrete-valued outputs that say to which class the corresponding inputs belong. Here for simplicity we study two-class problems, where the examples belong to class 0 or class 1. This is similar to two of our previous homeworks, where we designed Naïve Bayes classifiers to determine whether a review is positive or negative, and we designed a perceptron to determine which class the data points belong to (The Genetic Algorithm plus perceptron problem).

Logistic Regression for Classification

We already designed a model for linear regression:
$\hat{y} = x_p.W$
A simple strategy to adapt this model to classification problems can be adding a step function, a threshold function, to the output of this linear model. This step function would map the continuous outputs of the linear models to two levels, 0 and 1, representing and symbolizing two classes. We used this trick in homework 2.

$$output = \begin{cases} 1 & if\ \ x_p.W > 0 \\ 0 & otherwise \end{cases} \tag{2}$$

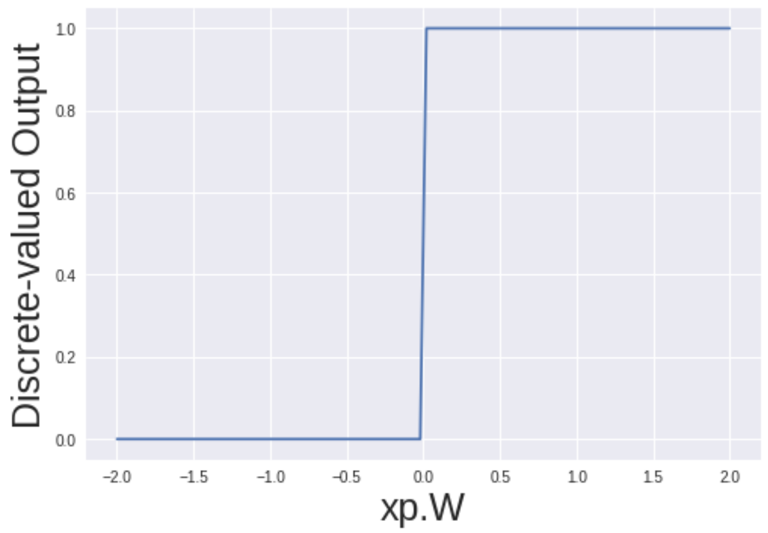


Fig. 1: We use a step function to produce a binary label

We used this simple trick in homework 2, and indeed it enabled us to classify those two classes of datapoints from each other. But there are challenges that come with this output function. For example, this output function is not differentiable, and therefore we cannot use gradient descent-based optimization methods to train such a model. To address this issue, people started to suggest different smooth versions of step function. Logistic function, also known as sigmoid function, is a famous function that is smooth and differentiable, and fits the needs of our problem. The sigmoid function is defined as:

$$S(z) = \frac{1}{1+e^{-z}} \tag{3}$$

Figure 2 plots the outputs of sigmoid function *S(z)* for different input values *z*:

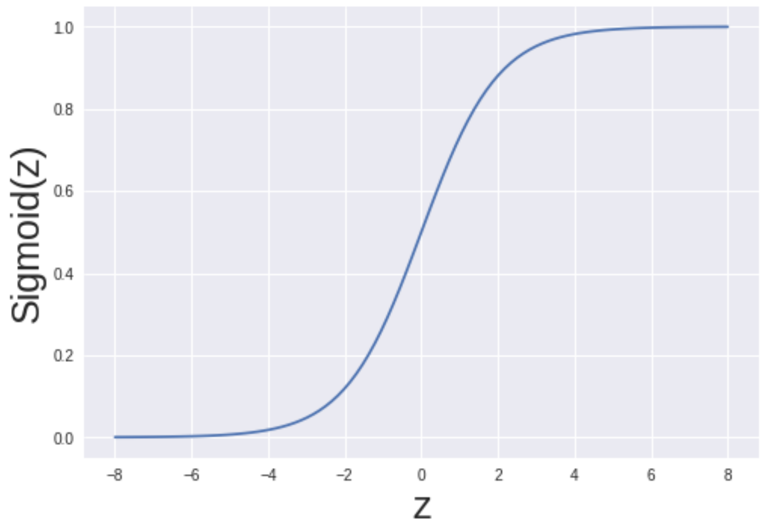


Fig. 2: The sigmoid function

You may or may not find an implementation of a Sigmoid function in a Python module. But it is extremely easy to write your own Sigmoid function to implement it. You just need to write a code for Eq. (3) (and define it as a function in Python).

It is important to notice that the sharp step function in Fig. 1 gives a definite binary label of 0 or 1 to each input, whereas the sigmoid function gives a number between 0 and 1. There are multiple implications for this observation:

- One can interpret the output of the sigmoid function when $(x_p.W)$ is given as the input, as the probability that the input x_p belongs to class 1. In other words, the sigmoid function gives $P(y=1|x_p,W)$. $P(y=1|x_p,W)$ is the conditional probability of output label being 1 given the input x_p and model parameters W. We are given x_p, and we like to estimate the probability of output being 1 for the given input. Note that the conditional probability depends on the model parameters W as well. Different model parameters create different conditional probabilities for the output label. That is why the conditional probability of output label being 1 depends on W as well.
- Since we have just two possible output labels, 1 and 0, and the output must be 0 or 1, then we can write $P(y=0|x_p,W) = 1 - P(y=1|x_p,W)$.
- So now we have two conditional probabilities for the output labels, $P(y=1|x_p,W)$, which model directly gives us, and $P(y=0|x_p,W)$ that we obtain by subtracting $P(y=1|x_p,W)$ from 1 (the total probability). As a result, the process of picking a label has come down to looking at these two probabilities, and picking the label that has a higher probability given the input and the model parameters.

$$output = \begin{cases} 1 & if\ \ P(y=1|x_p,W) \geq (y=0|x_p,W) \\ 0 & otherwise \end{cases} \tag{4}$$

Equation 4 can be simplified to Eq. (5), because in binary classifications with just two possible labels we do not need to calculate $P(y=0|x_p,W)$ as well, we just need to check whether $P(y=1|x_p,W)$ is greater than 0.5 or not. If it is, the output is 1, otherwise $P(y=0|x_p,W)$ is going to be greater than 0.5 and we should choose 0 as the output label.

$$output = \begin{cases} 1 & if \;\; P(y=1|x_p,W) \geq 0.5 \\ 0 & otherwise \end{cases} \tag{5}$$

- And yes, this is a Discriminative model in the sense that it directly tries to labels the inputs without estimating any intermediate probability distribution such as joint probability distribution between inputs and outputs, or conditional probability of inputs given the output labels. Those models that try to model the data and the distribution between the inputs and outputs are called Generative models.
- Roughly speaking, a model that produces a probability (a continuous value) as a measure of confidence in which label is the correct output label, is easier to train than a model that produces a definite 0/1 outputs. In former case, we have a continuous, smooth path between two cases that we can use in order to train the model and shift and adjust the outputs, whereas in the latter case, there outputs and system behavior is discrete and it is harder to gradually adjust and shift the system parameter.
-

Training Logistic Regression Model

So we now have a logistic regression model that receives inputs, and according to its model parameters produces a conditional probability function that gives a degree of belief in whether the inputs belongs to class 0 or 1. And we choose the class label that comes with a higher conditional probability.

But the remaining question is how to pick and choose the model parameter W that minimizes the error of the logistic regression model in classification. Therefore we have an optimization problem in hand, and we need two things to solve this optimization problem, 1) a suitable cost function to represent how well or poorly the model is performing the classification task, and 2) an optimization technique to minimize the cost function by adjusting the model parameters W.

In homework 2 we used the number of misclassifications as an error (cost) function. That cost function was adequate for Genetic algorithm technique that we used for optimization, but not for the gradient-based methods. The number of misclassifications, as the name suggests, gives a discrete-valued, integer numbers as a measure of error. We cannot apply gradient-based methods to such a discrete-valued function. In gradient-based methods we need a smooth, differentiable cost function.
A good solution can be to directly use the continuous-valued conditional probabilities, instead of 0 and 1 outputs obtained from Eq. (5), and calculate an cost function using our good old mean squared error function:

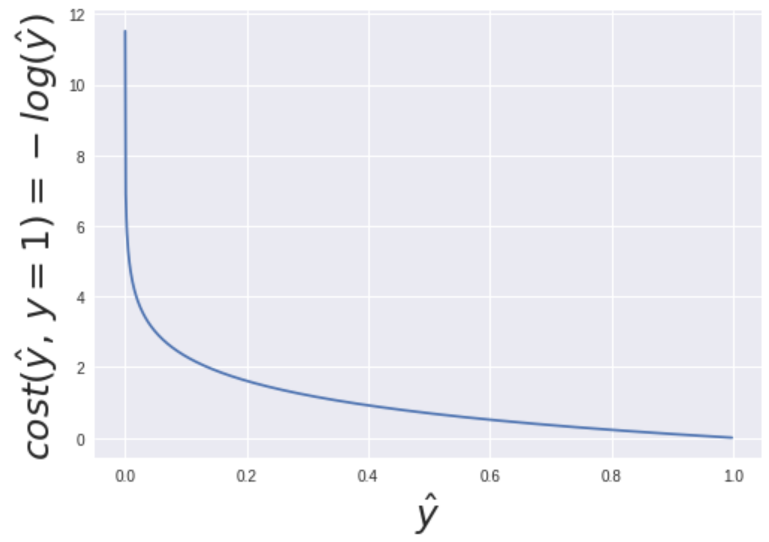
$$J(W) = \frac{1}{2m}\sum_i \left(S\left(x_p^i.W\right) - y^i\right)^2 \tag{6}$$

$$= \frac{1}{2m} \sum_i \left(\frac{1}{1 + e^{-(x_p^i . W)}} - y^i \right)^2$$

The good thing about this mean squared error function is that it is smooth and differentiable. However, Eq. (6) cost function is not convex any more because of the additional complex, exponential function inside the parenthesis, and as a result *J(W)* contains many local minimums. Therefore, the regular gradient descent technique is going to find and return back the local minimum points, not the global, optimal solution. One might suggest using the stochastic gradient descent in order to escape from these local minimums. Although this is doable, there is an easier, more practical approach to applying gradient descent methods in logistic regression. Instead of using mean squared error let us use a different error function, cross entropy error function. For simplicity of notation let us pick up the old notation for the output $\hat{y} = S(x_p^i . W)$. Cross entropy function says that the cost of producing $\hat{y}$ when the actual output is *y* is:

$$cost(\hat{y}, y) = \begin{cases} -\log(\hat{y}) & if\ y = 1 \\ -\log(1 - \hat{y}) & if\ y = 0 \end{cases} \tag{7}$$

Note that y is the class label; therefore it is either 0 or 1. Let us examine this cost function to see what it does. Figure 3 (top panel) plots $cost(\hat{y}, y)$ for different $\hat{y}$ values when *y*=1, and the bottom panel shows the cost function for different $\hat{y}$ values when *y*=0.



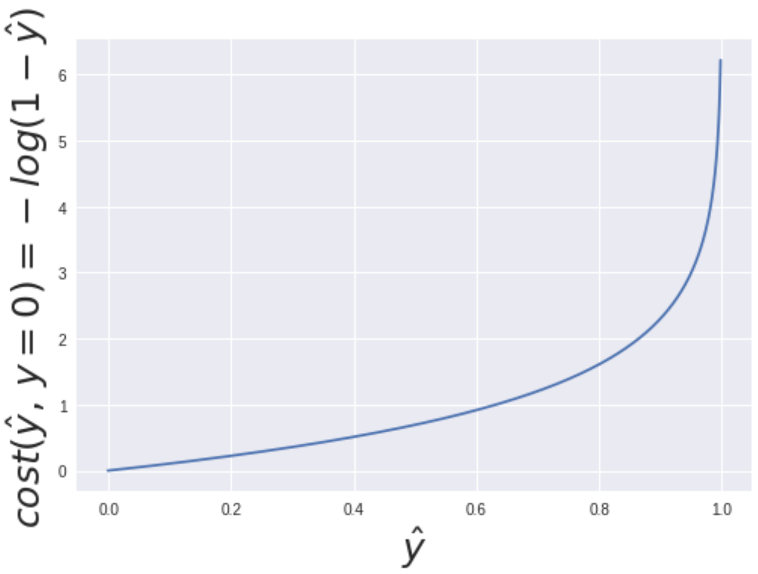


Fig. 3: Top Panel: $cost(\hat{y}, y)$ when y=1 evaluated for different $\hat{y}$ values. Bottom Panel: $cost(\hat{y}, y)$ when y=0 evaluated for different $\hat{y}$ values

Couple of important notes:

- Remember that $\hat{y}$, as a probability function, takes values between 0 and 1 because $\hat{y} = S(x_p^i.W)$. As a result, we evaluate the cost function for these values of $\hat{y}$.
- The top panel shows the cost function when the correct label is 1, y=1. In this case the model is better to predict 1 with a high probability, otherwise it is going to be penalize based on how off $P(y = 1|x_p, W)$ is. When the logistic regression model's output is almost 1, $\hat{y} \rightarrow 1$, meaning that the model is very confident that the label is 1, then the cost of this prediction is almost 0, $cost(\hat{y}, y) \rightarrow 0$. This makes sense because the hypothesis is correct and there should be no cost for this prediction. On the other extreme case, if the model is very confident that the label is 0 by producing a very small $P(y = 1|x_p, W)$, meaning that $P(y = 0|x_p, W)$ is very high, there is an extremely high cost for this very wrong and confident prediction.
- The cost function for y=0 is the same as the cost function for y=1 when it is flipped horizontally. So both costs functions penalize the model in the same way depending on how off the predicted probabilities (predictions) are from the correct label. When y=0 the model is better to be confident that the label is 0 by predicting $\hat{y}$ as close to 0 as possible, $\hat{y} \rightarrow 0$, otherwise it is going to be penalized based on Fig. 3 bottom.

Equation 7 cost function can be rewritten in a simpler, more compact way:

$$cost(\hat{y}, y) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}) \qquad (8)$$

Note that y can always take just two values, 0 and 1. When y=0, the Eq. 8 cost function will become:

$$cost(\hat{y}, y) = \cancel{-y\log(\hat{y})} - (1-y)\log(1-\hat{y})$$

which is the same as Eq. (7), and when y=1, then:

$$cost(\hat{y}, y) = -y\log(\hat{y}) \cancel{- (1-y)\log(1-\hat{y})}$$

and this is again what the Eq. (6) says.

Equation (8) gives the cost function (or the error function) for just one instance of the training data. To obtain the cost function for the entire training data, we need to find the average cost function across the entire training data set:

$$J(W) = \frac{1}{m}\sum_i cost(\hat{y}^i, y^i)$$
$$= \frac{1}{m}\sum_i (-y^i \log(\hat{y}^i) - (1-y^i)\log(1-\hat{y}^i)) \quad (9)$$
$$= \frac{1}{m}\sum_i (-y^i \log\left(\frac{1}{1+e^{-(x_p^i . W)}}\right) - (1-y^i)\log\left(1-\frac{1}{1+e^{-(x_p^i . W)}}\right))$$

We have the cost function J(W), we need to find optimal model parameters W to minimize this cost function. You might be wondering what the reason or logic was behind choosing sigmoid (Logistic) function as the output function (also called an activation function) and cross entropy as the cost function. The answer is that the resulting Eq (9) function is a convex function, therefore we can safely apply gradient descent methods to find the optimal W, and equally interesting, when we calculate the partial derivates of Eq. (9) with respect to model parameters w_j, we will obtain a very familiar equation:

$$\frac{\partial}{\partial w_j} J(W) = \frac{1}{m}\sum_i x_{pj}^i(\hat{y}^i - y^i)$$

which is the same formula for linear regression models! Of course $\hat{y}^i$ has different definitions in linear regression and in Logistic regression; in linear regression $\hat{y}^i = x_p^i . W$ whereas in Logistic regression $\hat{y}^i = \frac{1}{1+e^{-(x_p^i . W)}}$. However, the equations for gradient descents are exactly the same in terms of $\hat{y}^i$, y^i, and x_{pj}^i.

Homework:

In homework 2 we were given a two-class data set:

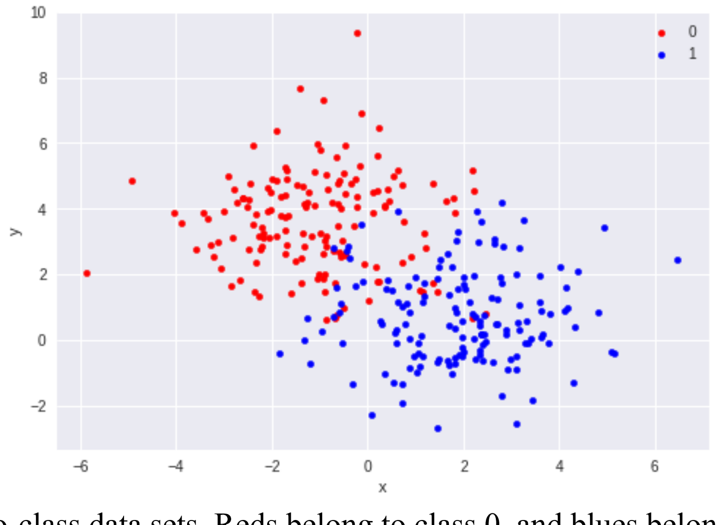


Fig. 4: A two-class data sets. Reds belong to class 0, and blues belongs to class 1.

Our aim was to design a classifier to classify class 0 from class 1. We used a simple perceptron with a step function as the activation function (output function), which produced 0 or 1. And we used GA to train this perceptron, and find the optimal parameters that enabled the perceptron to do the classification with minimal error.

In this homework our mission is to design and train a logistic regression classifier to perform the same task, but this time we like to train the model using gradient descent methods. As we discussed in this note, we need to choose a proper activation function and cost function in order to apply the gradient dissent method. We introduced sigmoid function and cross-entropy cost functions for this purpose.

So here is the plan:

- Go back to homework 2 and take the same training data set.
- Define a function for Logistic regression classifier ($\hat{y} = \frac{1}{1+e^{-(x_p.W)}}$).
- In session 11, we solved a linear regression problem and found the optimal model parameter W, using three different gradient descent methods: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent. Use these three methods to train your logistic regression classifier.

Feel free to use the codes I gave during session 11, or write your own codes.