CSE/ECE-471: Statistical Methods in AI **Spring 2020**

Logistic Regression and Introduction to Artificial Neural Networks

Prepared by: Aditya(2019201047) Indranil(2019202008) Garima(2019701029)

1 What is Wrong with Linear Regression for Classification ?

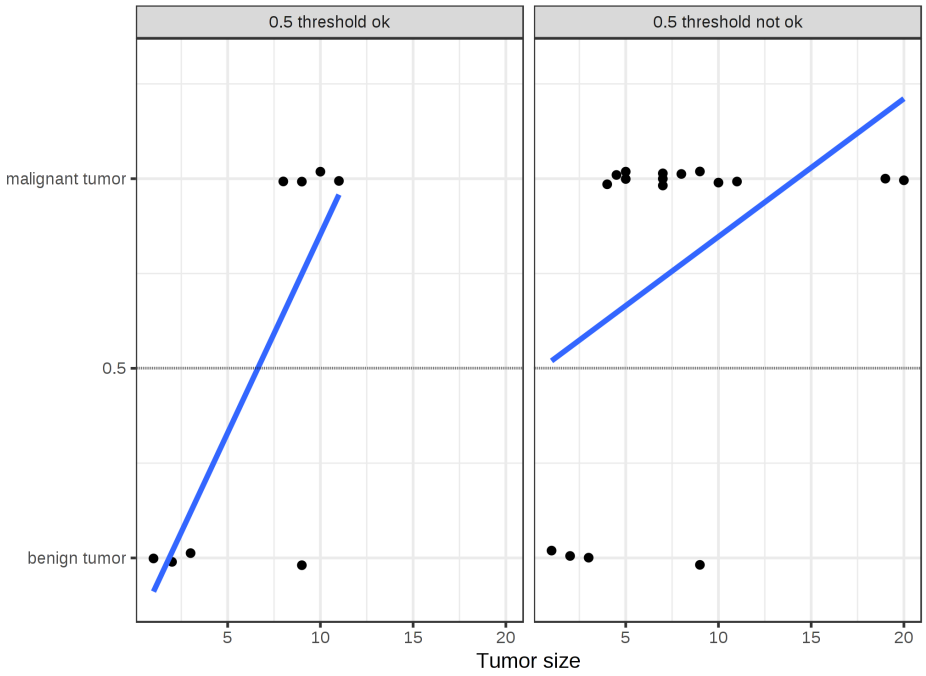
The linear regression model can work well for regression, but fails for classification. Why is that? In case of two classes, you could label one of the classes with 0 and the other with 1 and use linear regression. Technically it works and most linear model programs will spit out weights for you. But there are a few problems with this approach:

A linear model does not output probabilities, but it treats the classes as numbers (0 and 1) and fits the best hyper-plane (for a single feature, it is a line) that minimizes the distances between the points and the hyper-plane. So it simply interpolates between the points, and you cannot interpret it as probabilities.

A linear model also extrapolates and gives you values below zero and above one. This is a good sign that there might be a smarter approach to classification.

Since the predicted outcome is not a probability, but a linear interpolation between points, there is no meaningful threshold at which you can distinguish one class from the other.

The higher the value of a feature with a positive weight, the more it contributes to the prediction of a class with a higher number, even if classes that happen to get a similar number are not closer than other classes.

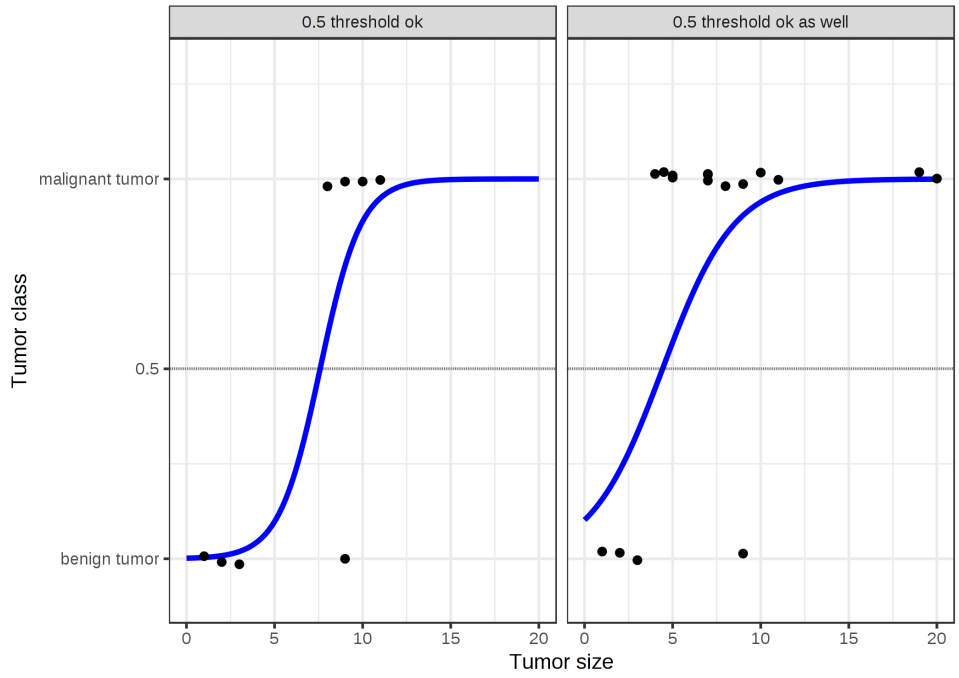


A linear model classifies tumors as malignant (1) or benign (0) given their size. The lines show the prediction of the linear model. For the data on the left, we can use 0.5 as classification threshold. After

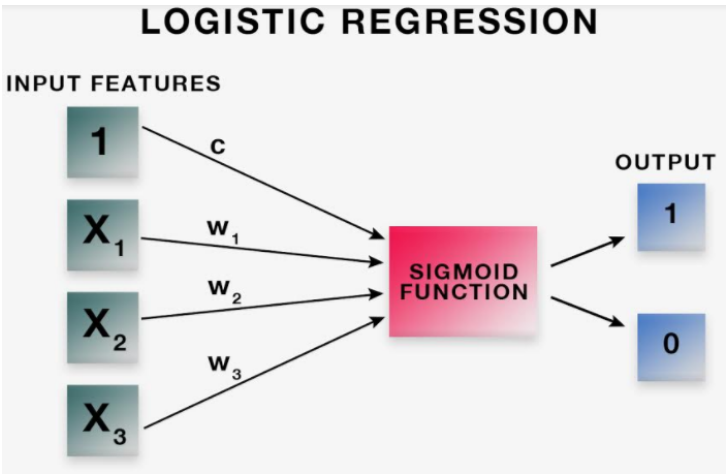
introducing a few more malignant tumor cases, the regression line shifts and a threshold of 0.5 no longer separates the classes. Points are slightly jittered to reduce over-plotting.

If we use linear regression for this problem, there is a need for setting up a threshold based on which classification can be done. Say if the actual class is malignant, predicted continuous value 0.4 and the threshold value is 0.5, the data point will be classified as not malignant which can lead to serious consequence in real time. From this example, it can be inferred that linear regression is not suitable for classification problem. Linear regression is unbounded, and this brings logistic regression into picture.

Let us revisit the tumor size example again. But instead of the linear regression model, we use the logistic regression model:



Logistic Regression is used when the dependent variable(target) is categorical.



We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn't make sense for $h_\theta(x)$ to take values larger than 1 or smaller than 0 when we know that y ϵ { 0,1 } .

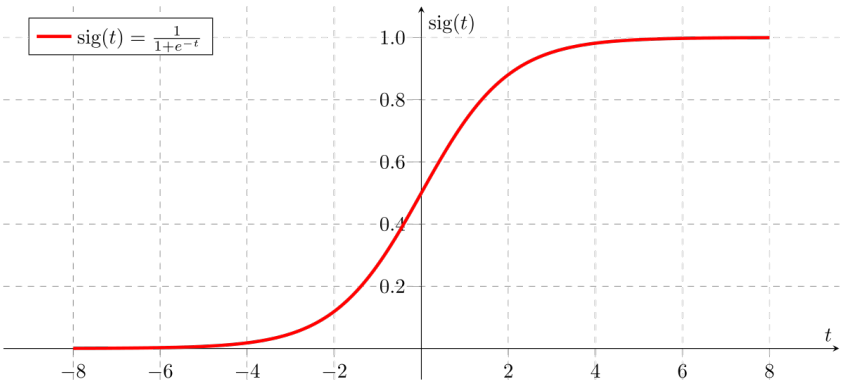
To fix this, let's change the form for our hypotheses $h_\theta(x)$. We will choose

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

where

$$g(z) = \frac{1}{1 + e^{-z}}$$

is called the logistic function or the sigmoid function. Here is a plot showing $g(z)$:



Notice that $g(z)$ tends towards 1 as $z \to +\infty$, and $g(z)$ tends towards 0 as $z \to -\infty$. Moreover, $g(z)$, and hence also $h(x)$, is always bounded between 0 and 1. Before moving on, here's a useful property of the derivative of the sigmoid function, which we write as g' :

$$\begin{aligned} g'(z) &= \frac{d}{dz}\frac{1}{1+e^{-z}} \\ &= \frac{1}{(1+e^{-z})^2}\left(e^{-z}\right) \\ &= \frac{1}{(1+e^{-z})} \cdot \left(1 - \frac{1}{(1+e^{-z})}\right) \\ &= g(z)(1-g(z)). \end{aligned}$$

So, given the logistic regression model, how do we fit θ for it? We need to maximize the probability. Let's endow our classification model with a set of probabilistic assumptions, and then fit the parameters via maximum likelihood.

Let us assume that

$$\begin{aligned} P(y = 1 \mid x; \theta) &= h_\theta(x) \\ P(y = 0 \mid x; \theta) &= 1 - h_\theta(x) \end{aligned}$$

Note that this can be written more compactly as

$$p(y \mid x; \theta) = (h_\theta(x))^y \left(1 - h_\theta(x)\right)^{1-y}$$

Assuming that the m training examples were generated independently(hence, taking product of each sample of data), we can then write down the likelihood of the parameters as

$$\begin{aligned} L(\theta) &= p(\vec{y} \mid X; \theta) \\ &= \prod_{i=1}^{m} p(y^{(i)} \mid x^{(i)}; \theta) \\ &= \prod_{i=1}^{m} \left(h_\theta(x^{(i)})\right)^{y^{(i)}} \left(1 - h_\theta(x^{(i)})\right)^{1-y^{(i)}} \end{aligned}$$

Taking log, for mathematical convenience,

$$\begin{aligned} \ell(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)})) \end{aligned}$$

Written in vectorial notation, our updates will therefore be given by :

$$\theta := \theta + \alpha \nabla_\theta \ell(\theta)$$

(Note the positive rather than negative sign in the update formula, since we're maximizing, rather than minimizing, a function now.) Let's start by working with just one training example (x, y), and take derivatives to derive the stochastic gradient ascent rule

$$\begin{aligned} \frac{\partial}{\partial \theta_j} \ell(\theta) &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) \frac{\partial}{\partial \theta_j} g(\theta^T x) \\ &= \left(y \frac{1}{g(\theta^T x)} - (1 - y) \frac{1}{1 - g(\theta^T x)} \right) g(\theta^T x)(1 - g(\theta^T x)) \frac{\partial}{\partial \theta_j} \theta^T x \\ &= \left(y(1 - g(\theta^T x)) - (1 - y) g(\theta^T x) \right) x_j \\ &= \left(y - h_\theta(x) \right) x_j \end{aligned}$$

Above, we used the fact that $g'(z) = g(z)g(1 - z)$. This therefore gives us the stochastic gradient ascent rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

2 Advantages and Disadvantages of Logistic Regression

- Many of the pros and cons of the linear regression model also apply to the logistic regression model. Logistic regression has been widely used by many different people, but it struggles with its restrictive expressiveness (e.g. interactions must be added manually) and other models may have better predictive performance.

- Another disadvantage of the logistic regression model is that the interpretation is more difficult because the interpretation of the weights is multiplicative and not additive.
- Logistic regression can suffer from complete separation. If there is a feature that would perfectly separate the two classes, the logistic regression model can no longer be trained. This is because the weight for that feature would not converge, because the optimal weight would be infinite. This is really a bit unfortunate, because such a feature is really useful. But you do not need machine learning if you have a simple rule that separates both classes. The problem of complete separation can be solved by introducing penalization of the weights or defining a prior probability distribution of weights.
- On the good side, the logistic regression model is not only a classification model, but also gives you probabilities. This is a big advantage over models that can only provide the final classification. Knowing that an instance has a 99 % probability for a class compared to 51% makes a big difference.
- Logistic regression can also be extended from binary classification to multi-class classification. Then it is called Multinomial Regression.

3 Multinomial Logistic Regression

- Multinomial logistic regression (often just called 'multinomial regression') is used to predict a nominal dependent variable given one or more independent variables.
- It is sometimes considered an extension of binomial logistic regression to allow for a dependent variable with more than two categories.
- As with other types of regression, multinomial logistic regression can have nominal and/or continuous independent variables and can have interactions between independent variables to predict the dependent variable.
- It is a modification of logistic regression using the softmax function instead of the sigmoid function the cross entropy loss function. The softmax function squashes all values to the range [0,1] and the sum of the elements is 1.

$$\mathbf{softmax}(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

- Cross entropy is a measure of how different 2 probability distributions are to each other. If p and q are discrete we have :

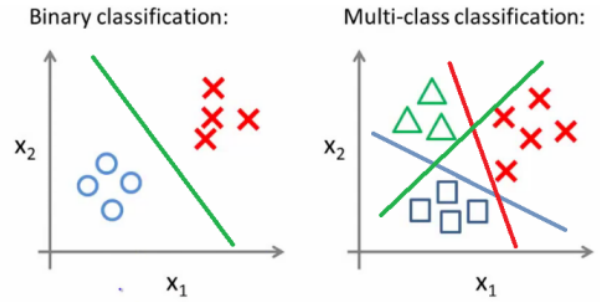
$$H(p, q) = -\sum_x p(x) \log q(x).$$

- This function has a range of $[0, \infty]$ and is equal to 0 when $p = q$ and infinity when p is very small compared to q or vice versa. For an example x, the class scores are given by vector $z = Wx + b$, where W is a $C \times M$ matrix and b is a length C vector of biases. We define the label y as a one-hot vector equal to 1 for the correct class c and 0 everywhere else. The loss for a training example x with predicted class distribution y and correct class c will be :

$$\hat{y} = \mathbf{softmax}(z)$$

$$\begin{aligned}\text{loss} &= H(y, \hat{y}) \\ &= -\sum_i y_i \log \hat{y}_i \\ &= -\log \hat{y}_c\end{aligned}$$

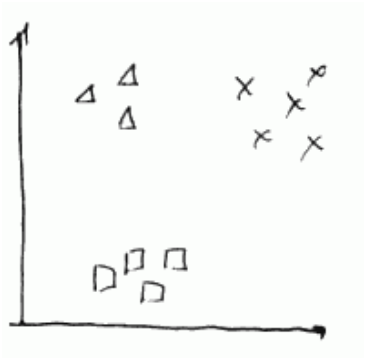
- As in the binary case, the loss value is exactly the negative log probability of a single example x having true class label c. Thus, minimizing the sum of the loss over our training examples is equivalent to maximizing the log likelihood. We can learn the model parameters W and b by performing gradient descent on the loss function with respect to these parameters.



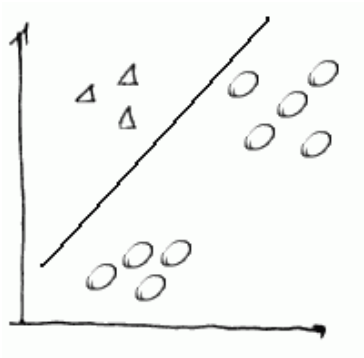
- Now, there are two common methods to perform multi-class classification using the binary classification logistic regression algorithm: one-vs-all and one-vs-one. We are going to discuss one-vs-all classification.

4 One-vs-All Classification

- Suppose we have a classifier for sorting out input data into 3 categories, 1.class 1 (△), 2. class 2 (□), 3. class 3(×).

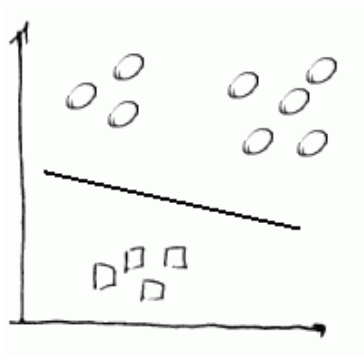


- We may turn this problem into 3 binary classification problems (i.e. where we predict only y ϵ {0,1}) to be able to use classifiers such as Logistic Regression.
- We take values of one class and turn them into positive examples, and the rest of classes - into negatives
- Step 1 - △ are positive, and the rest are negative - and we run a classifier on them.



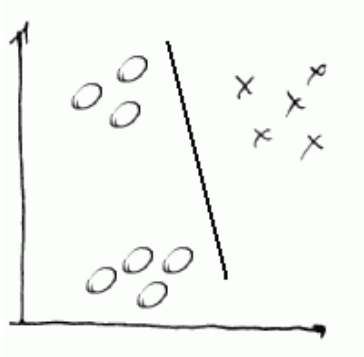
and we calculate $h_\theta^{(1)}(x)$ for it

- Step 2 - next we do same with □: make them positive, and the rest - negative



and we calculate $h_\theta^{(2)}(x)$

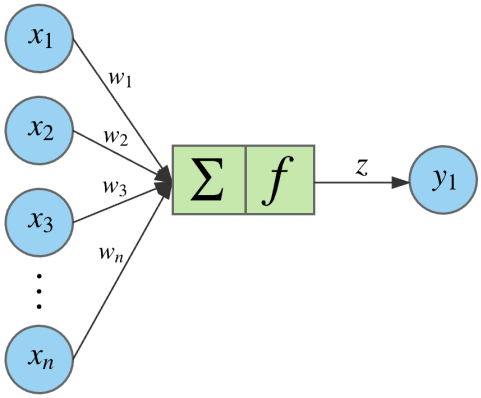
- Step 3 - finally, we make $\times$ as positive and the rest as negative and calculate $h_\theta^{(3)}(x)$



- So we have fit 3 classifiers: $h_\theta^{(i)}(x) = P(y = i|x; \theta)$, i = 1, 2, 3
- Now, having calculated the vector $h_\theta(x) = [h_\theta^{(1)}(x),\ h_\theta^{(2)}(x), h_\theta^{(3)}(x)]$ we just pick up the maximal value i.e. we choose $\max_i h_\theta^{(i)}(x)$

5 Introduction to Artificial Neural Networks

Artificial Neural Networks(ANN) is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem. Each unit implements simple function.



For the above general model of artificial neural network, the net input can be calculated as follows:

$y_{in} = x_1.w_1 + x_2.w_2 + x_3.w_3....x_m.w_m$

$y_{in} = \Sigma m_i x_i.w_i$

The output can be calculated by applying the activation function over the net input: $Y = F(y_{in})$

The perceptron model is a more general computational model . It takes an input, aggregates it (weighted sum) and returns 1 only if the aggregated sum is more than some threshold else returns 0. A single perceptron can only be used to implement linearly separable functions. It takes both real and Boolean inputs and associates a set of weights to them, along with a bias (the threshold thing I mentioned above). We learn the weights, we get the function.

Following are the major components of a perceptron:

- Input: All the features become the input for a perceptron. We denote the input of a perceptron by $[x_1, x_2, x_3, .., x_n]$, where x represents the feature value and n represents the total number of features. We also have special kind of input called the bias. In the image, we have described the value of the BIAS as w_0.
- Weights: The values that are computed over the time of training the model. Initially, we start the value of weights with some initial value and these values get updated for each training error. We represent the weights or perceptron by $[w_1, w_2, w_3, .., w_n]$.
- Step/activation function: The role of activation functions is to make neural networks nonlinear. For linear classification, for example, it becomes necessary to make the perceptron as linear as possible.
- Bias: A bias neuron allows a classifier to shift the decision boundary left or right. In algebraic terms, the bias neuron allows a classifier to translate its decision boundary. It aims to ”move every point a constant distance in a specified direction.” Bias helps to train the model faster and with better quality.
- Weighted summation: Weighted summation is the sum of the values that we get after the multiplication of each weight $[w_n]$ associated with the each feature value $[x_n]$.
- Output: The weighted summation is passed to the step/activation function and whatever value we get after computation is our predicted output.
- Description:
  - Firstly, the features for an example are given as input to the perceptron.
  - These input features get multiplied by corresponding weights (starting with initial value).

- The summation is computed for the value we get after multiplication of each feature with the corresponding weight.
- The value of the summation is added to the bias.
- The step/activation function is applied to the new value.

Perceptron Learning Rule : This rule is an error correcting the supervised learning algorithm of single layer feed-forward networks with linear activation function, introduced by Rosenblatt. As being supervised in nature, to calculate the error, there would be a comparison between the desired/target output and the actual output. If there is any difference found, then a change must be made to the weights of connection.

Mathematical Formulation : Suppose we have 'n' number of finite input vectors, x_n, along with its desired/target output vector t_n, where $n = 1$ to N.

Now the output 'y' can be calculated, as explained earlier on the basis of the net input, and activation function being applied over that net input can be expressed as follows:

$$y = f(y_{in}) \ = \ \begin{cases} 1, & y_{in} \ > \ \theta \\ 0, & y_{in} \ \theta \end{cases} \tag{1}$$

Where θ is threshold.

The updating of weight can be done in the following two cases

Case I when $t \neq y$, then

$$w(new) = w(old) \ + \ tx \tag{2}$$

Case II when $t = y$, then there is no change in weight.

References

[1] Lecture notes on Logistic Regression and Artificial Neural Networks, made by the authors of this document.

[2] Lecture notes by Andrew Ng,http://cs229.stanford.edu/notes/cs229-notes1.pdf

[3] https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

[4] https://statistics.laerd.com/spss-tutorials/multinomial-logistic-regression-using-spss-statistics.php

[5] https://medium.com/@jjw92abhi/is-logistic-regression-a-good-multi-class-classifier-ad20fecf1309

[6] http://mlwiki.org/index.php/One-vs-All_Classification