Logistic Regression

Introduction

Logistic regression is a **supervised learning** technique, which uses correlation between variables to classify data into groups (for example classifying data into airlines). Much like linear regression, the model trains itself by varying values of theta which are coefficients for input variables. The values of theta are tuned such that as many data in the training set as possible are classified into the correct group. For the model to work, the groups must be represented numerically, so 0 represents a group and 1 the other (classification into >2 groups will be explained later).

How Data are Classified

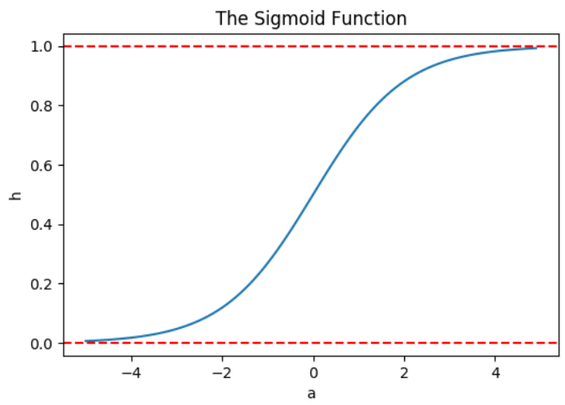
The process for classifying a datum starts with an equation of the input variables (which can be anything that might correlate to your grouping and don't have to be numerical*):

$$a(x_1, x_2) = \theta_2 x_2^2 + \theta_1 x_1 + \theta_0$$

The output a is then put through the sigmoid function

$$h(a) = \frac{1}{1 + e^{-a}}$$

which normalises the value to between 0 and 1.



The output of the sigmoid function is then rounded to 0 or 1, and this represents the models prediction. The difference between the output before rounding and its rounded value actually represents the confidence of the model in its answer.

*Numerically representing data is key to logistic regression, for example an input column of which manufacturer produced an aircraft could be useful. This can be done by creating a column for each manufacturer, and the row value being a 1 if the aircraft were created by that manufacturer and 0 if not. Each column (manufacturer) would then be its own input variable, with its own theta value.

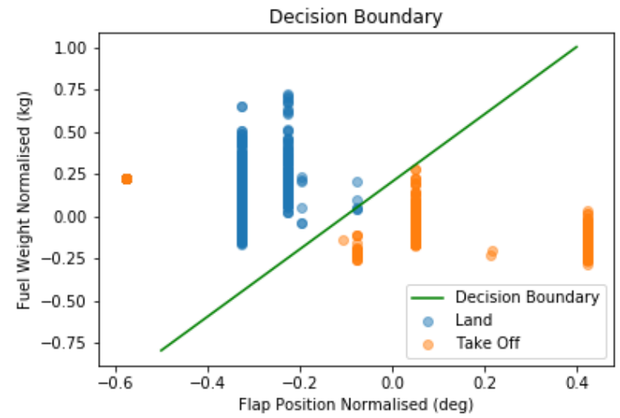
The Decision Boundary

The decision boundary for a logistic regression model represents the line where the model is right on the edge of choosing either group. Since

$$\frac{1}{1 + e^{-0}} = 0.5$$

and 0.5 is the boundary for rounding to 0 or 1, the decision boundary lies where $a = 0$. This boundary's position is changed by varying $\theta_0, \theta_1, \theta_2$.

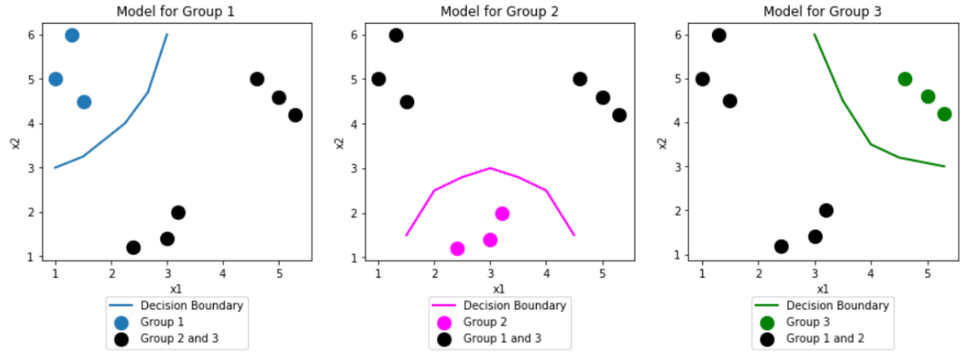
Let's say we wanted to classify whether an aircraft was taking off (0) or landing (1). Thinking about this, good input variables might be fuel weight (likely to have more fuel taking off) and flap position (further extended for landing).



In this trained model, the theta values have been modified such that the decision boundary separates the groups as well as possible.

More than 2 Groups

When there are more than 2 groups, a model is created for each group, where that group is considered a positive classification (1) and all the others a negative classification (0):



For predicting new data, all models are run for each datum and it's classified into the group with highest value of h for the corresponding model.

Implementing Logistic Regression

Source code can be found at P:\ENGINEERING\FPO\Digitalisation\1 FPO Projects\IIX Data Analytics\02-General presentations\Knowledge folder - preparation\Data Science Learning Framework\Machine Learning Posters\3 - Logistic Regression

Introduction

We're going to use the example above, classifying whether an aircraft is taking off or landing at a given moment in flight based on its fuel weight and flap position, to implement a simple version of logistic regression. Using training data, we'll tune θ_0, θ_1, θ_2 such that the equations

$$a(x_1, x_2) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$
$$h(a) = \frac{1}{1 + e^{-a}}$$

x_1- Flap Position
x_2- Fuel Weight

output $h < 0.5$ if the aircraft is taking off and $h > 0.5$ if the aircraft is landing, for as many data as possible. This can then be used to predict if an aircraft is taking off or landing for unlabeled data!

Much of the data preprocessing is similar/identical to linear regression.

Formatting the Data

Input and output data must be put into the correct format, and you can apply powers if you want a non-linear decision boundary. A column of ones should be added to train θ_0, and variables must be separated into features (X) and labels (y) that you're training for. y values should be represented numerically (0 for take-off, 1 for landing).

	constant	flap_pos	total_fuel_weight
0	1	13.974609	8708.964844
1	1	13.974609	8745.251953
2	1	13.974609	8799.683594
3	1	13.974609	8926.689453
4	1	24.960938	4154.901855

Table X

	flight_segment
0	0
1	0
2	0
3	0
4	1

Table y

The feature columns should then be scaled to allow faster convergence, using the equation

$$\mathrm{X}_{i,j} = \frac{\mathrm{X}_{i,j} - \overline{\mathrm{X}_j}}{\max \mathrm{X}_j - \min \mathrm{X}_j}$$

which will return all values scaled to $-0.5 < x < 0.5$.

Finally, data should be separated (randomly to avoid bias) into training and testing data, essentially taking a number of rows that you won't show the model while it's training, such that it can be tested against them without having "seen the answers". The % split of training and testing data is arbitrary, but 70/30 is a good ballpark figure.

Cost and Gradient Descent

Cost (J) measures the accuracy of the model, it assesses the difference between predicted and actual values in the training data. Since the difference will always be equal to or less than 1 (the most wrong it can be is 1-0 for classification), a natural log is applied to amplify how much the model is penalized.

$$J(\theta_0, \theta_1, \theta_2) = -\frac{1}{m}\sum_{i=1}^{m} \underbrace{y^{(i)} \log(h(\theta_0 + \theta_1 x_1 + \theta_2 x_2))}_{\text{Term 1}} + \underbrace{(1 - y^{(i)}) \log(1 - h(\theta_0 + \theta_1 x_1 + \theta_2 x_2))}_{\text{Term 2}}$$

m – Number of rows in training data
$y^{(i)}$ – Label/Target for the i^{th} row
$x_j^{(i)}$- Value for the j^{th} feature of the i^{th} row

There's a lot going on here, it may be easier to understand with some examples.

Example 1 – Target is Take-Off (0)

$y^{(i)} = 0 \therefore \text{Term } 1 = 0$

$1 - y^{(i)} = 1 \therefore \lim_{h \to 0} \text{Term } 2 = 0 \text{ and } \lim_{h \to 1} \text{Term } 2 = -\infty$

So we can see the error tends from 0 to ∞ as h gets further from the right answer (0)

Example 2 – Target is Land (1)

$y^{(i)} = 1 \therefore \text{ and } \lim_{h \to 1} \text{Term } 1 = 0 \text{ and } \lim_{h \to 0} \text{Term } 1 = -\infty$

$1 - y^{(i)} = 0 \therefore \text{ Term } 2 = 0$

So we can see the error tends from 0 to ∞ as h gets further from the right answer (1)

$$\theta_j = \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m} (h(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)}) - y^{(i)}) x_j^{(i)}$$

New value of θ_j ← θ_j → Old value of θ_j

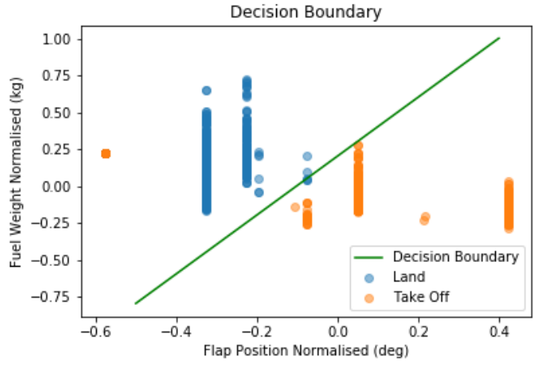
This gradient descent equation above is performed iteratively, updating the values of $\theta_0, \theta_1, \theta_2$ before using these updated values to repeat itself, this is how the model trains itself. The derivative term is subtracted from the old θ values, reducing the cost by bringing θ closer to the desired values. The learning rate is a coefficient which dictates how much the model will change its θ values. If it's too small excessive iterations will be needed, too large and the model will diverge as it hops over the correct value with each iteration.

Results

Having used gradient descent on our scaled data with an α of 10 (found through trial and error) and 100 iterations, θ values have converged:

$$\theta_0 = 1.28, \theta_1 = 13.44, \theta_2 = -7.1$$

By plugging flap position and weight values for the test data into the original equation with these theta values, applying the sigmoid and rounding, prediction of whether the aircraft is taking off or landing is 99% accurate.



Appendix for Code **Only Read if Following the Code**

The line for gradient descent is vectorised so it's more efficient, but makes it a little less readable. Below is an explanation step by step of how the operations work:

```
thetas[i+1]=thetas[i]-(alpha/len(X))*np.dot(sigmoid(np.sum(thetas[i]*X_train,axis = 1))-y_train,X_train)
```

Working from the inside out:

`thetas[i]*X_train` – This takes the feature values for the training data, and multiplies them all by the most recent theta value for their column.

$$\begin{matrix}\theta_0 & \theta_1 & \theta_2\end{matrix} \quad \begin{matrix}X_{10} & X_{11} & X_{12}\\ X_{20} & X_{21} & X_{22}\\ \vdots & \vdots & \vdots\end{matrix}$$

$$\downarrow$$

$$\begin{matrix}\theta_0X_{10} & \theta_1X_{11} & \theta_2X_{12}\\ \theta_0X_{20} & \theta_1X_{21} & \theta_2X_{22}\\ \vdots & \vdots & \vdots\end{matrix}$$

`np.sum(``thetas[i]*X_train`**`,axis = 1)`** – All the columns in each row are summed:

$$\begin{matrix}\theta_0X_{10} & \theta_1X_{11} & \theta_2X_{12}\\ \theta_0X_{20} & \theta_1X_{21} & \theta_2X_{22}\\ \vdots & \vdots & \vdots\end{matrix}$$

$$\downarrow$$

$$\begin{matrix}\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}\\ \theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}\\ \vdots\end{matrix}$$

`sigmoid(``np.sum(thetas[i]*X_train,axis = 1)`**`)`** – the sigmoid function is then applied to each rows:

$$\begin{matrix}\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}\\ \theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}\\ \vdots\end{matrix}$$

$$\downarrow$$

$$\begin{matrix}\frac{1}{1-e^{\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}}}\\ \frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}\\ \vdots\end{matrix}$$

`sigmoid(np.sum(thetas[i]*X_train,axis = 1))`**`-y_train`** – the target value is then subtracted from the predicted value:

$$\begin{matrix}\frac{1}{1-e^{\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}}}\\ \frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}\\ \vdots\end{matrix}$$

$$\downarrow$$

$$\begin{matrix}\frac{1}{1-e^{\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}}}-y_1\\ \frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}-y_2\\ \vdots\end{matrix}$$

`np.dot(``sigmoid(np.sum(thetas[i]*X_train,axis = 1))-y_train`**`,X_train)`** – Matrix multiplication is used to create 3 columns where each is the transposed output of the previous step multiplied by the features 1, 2 and 3, then summed:

$$\left(\frac{1}{1-e^{\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}}}-y_1 \quad \frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}-y_2 \quad \ldots\right)\cdot\begin{matrix}X_{10} & X_{11} & X_{12}\\ X_{20} & X_{21} & X_{22}\\ \vdots & \vdots & \vdots\end{matrix}$$

$$\downarrow$$

$$\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{10}+\theta_1X_{11}+\theta_2X_{12}}}-y_i\right)X_{i0} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}-y_i\right)X_{i1} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{20}+\theta_1X_{21}+\theta_2X_{22}}}-y_i\right)X_{i2}$$

`(`**`alpha/len(X))*`**`np.dot(sigmoid(np.sum(thetas[i]*X_train,axis = 1))-y_train,X_train)` – Scalar multiplication of all 3 values in the vector by $\frac{\alpha}{m}$ (learning rate over number of training examples):

$$\frac{\alpha}{m}\left(\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{11}+\theta_1X_{12}+\theta_2X_{13}}}-y_i\right)X_{i0} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i1} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i2}\right)$$

`thetas[i]-(``alpha/len(X))*np.dot(sigmoid(np.sum(thetas[i]*X_train,axis = 1))-y_train,X_train)` – Finally, subtracting the three values from the old values of theta:

$$\theta_0 \quad \theta_1 \quad \theta_2-\frac{\alpha}{m}\left(\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{11}+\theta_1X_{12}+\theta_2X_{13}}}-y_i\right)X_{i0} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i1} \quad \sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i2}\right)$$

$$\downarrow$$

$$\theta_0-\frac{\alpha}{m}\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{11}+\theta_1X_{12}+\theta_2X_{13}}}-y_i\right)X_{i0} \quad \theta_1-\frac{\alpha}{m}\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i1} \quad \theta_2-\frac{\alpha}{m}\sum_{i=1}^{m}\left(\frac{1}{1-e^{\theta_0X_{21}+\theta_1X_{22}+\theta_2X_{23}}}-y_i\right)X_{i2}$$