Logistic regression to determine titanic survival

Alan Antonio Macías López

Introduction

The Titanic accident occurred on April 15, 1912. The iceberg collision made the Titanic sink in approximately two and a half hours which was relatively fast for a ship of such size.

The ship was carrying 2224 people of all ages, genders and classes that night, 1514 passengers died in the icy waters and only 710 escaped in lifeboats. It is said that the chances of surviving of a passenger where influenced by several factors such as age, gender, class, etc…

The purpose of this report is to use a statistical model called logistic regression to determine if a passenger survived given some information about him or her.

Logistic Regression

Logistic regression is a statistical model used for classification problems. The most common type of logistic regression is binary, which is used to solve problems where the dependent variable (label) can land in two different categories, like checking if a tumor is malignant or not, determining if an email is spam or not, or in this particular case, predicting the survival of a titanic passenger.

Predictions

For each sample (passenger) we will make a prediction to determine if he or she survives or not. In order to do this, we first make a hypothesis using the linear function $Z = m1x1 + m2x2 + ..+b$. Where each x is an attribute of the passenger, each m is the parameter that affects that attribute and b is the bias.

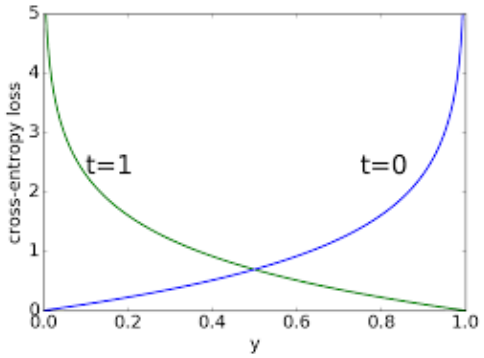
Once the hypothesis is calculated we apply the sigmoid function $\frac{1}{1+e^{-z}}$, which will allow us to get a value that strictly ranges from 0 to 1, so for example if we get 0.65 as a result of the sigmoid function for one passenger, we can say that that passenger has a 65% probability of surviving.

Cost Function

The cost function allows us to know how badly the model is performing, and therefore allowing it to improve over time. We will use cross-entropy as the cost function for this model. This function can be divided in two parts depending on the value of y.

$$\begin{aligned} \text{Cost}(h_\theta(x), y) &= -\log(h_\theta(x)) && \text{if y} = 1 \\ \text{Cost}(h_\theta(x), y) &= -\log(1 - h_\theta(x)) && \text{if y} = 0 \end{aligned}$$

The advantage of doing this is clearer when you look at both functions graphed.



This combined function makes it easier to calculate the gradient and minimize the cost. Since there is only one minimum, it also solves the problem of landing at a local minimum.

Gradient descent

It is the process of following the gradient of the cost function to land at its minimum, in order to have the smallest possible error. For this implementation we will use stochastic gradient descent, which updates the parameters every iteration to minimize the error. To update the parameters we will use the following formula. $m = m+ \propto +\big(y - h(x)\big) * h(x) * \big(1 - h(x)\big) * x[i,j]$

Where: m = parameter to update

∝ = learning rate, which limits how much the model can learn per iteration.

x = sample of the dataset

h(x) = Predicted class of the sample x

x[i, j] = attribute that is affected by the parameter

Titanic dataset

Available at: https://www.kaggle.com/c/titanic/data

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr.	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings, Mr	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, M	female	26	0	0	STON/O2. 31	7.925		S
4	1	1	Futrelle, Mrs	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr. W	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mr. J	male		0	0	330877	8.4583		Q
7	0	1	McCarthy, M	male	54	0	0	17463	51.8625	E46	S

These are the first samples of the titanic “train.csv” dataset. We will preprocess the whole dataset to solve some problems so that our model works properly.

Data preprocessing

Problem 1: Unnecessary fields

Before anything else, we must ask ourselves: ¿What passenger attributes may influence the chances of surviving? There are many ways of answering this question, but for this implementation these are the fields that will be considered as relevant, the rest will be removed.

Survived	Pclass	Sex	Age	SibSp	Parch

It is important to mention at this point that the "Survived" column does not contain an attribute, but the real label of each sample, so 1 = survived and 0 = did not survive. We will use this value to get the error of each prediction and help the model learn when doing gradient descent.

Problem 2: Not all values are numeric

Sex
male
female
female
female
male
male
male

If we leave the column "Sex" as it is, our code will crash when trying to make operations with the values 'male' and 'female' to update the parameters. So a very simple and straightforward solution is to replace ´male´ with 0 and ´female´ with 1.

Problem 3: Some values are empty

Age
22
38
26
35
35
54

As the previous problem, our code will crash when trying to make operations, but this time because the value is empty. There are several solutions for this problem; one of them is to manually assign the value to be equal to the average of the rest of the samples, but to keep things even more simple, we will just leave the value as an empty string, and when we want to update a parameter, we will first check if x[i, j] is not an empty string so that the program does not crash.

Training the model

We will update our parameters using gradient descent for a specific amount of epochs. For each iteration we will store the error based on the decimal prediction obtained using the sigmoid function so we can graph all the errors and see how our model improves over time. We also want to graph how well our model predicts if a passenger will survive or not, so each time a prediction is made, we will round it up to 0 or 1 and compare it with the real label, so that way each iteration we can see how accurate our model is and also see the improvement over time.

At the end we will save the final parameters, because those values are the ones that will be used to test our model and see how well it works with an unknown dataset.

This approach makes sense, but there´s a single problem: we will not know how good our model will perform with unseen data until we test it with the testing dataset. That´s why it is recommended to use something called validation dataset before jumping straight into testing.

Cross-validation

Cross-validation is a very useful technique used to split the training dataset in two, so that one part trains the model, and the other one validates it. This validation step is pretty much testing the model and measuring the error and accuracy (in our case), but the very important difference is that this testing is done without using the testing dataset. This process is repeated for any given folds, so at the end when we finish validating, we will have tested our model several times against unseen data to have a better perspective on how it will behave on average. We will graph the results and compare the results of the different folds to come to a conclusion. If we see that the accuracy has a small amount of variation between folds, then we know that our model is stable and that it behaves in a similar way against different cases of training and unseen data.

In the other hand if we see a lot of variation between the fold accuracies, we know that our model is unstable and some things can be changed (like the hyperparameters) to help our model stabilize.

Once we are happy with the cross-validation results, we will stop validating and now we are ready to start testing with the testing dataset. To do this we will train our model one more time but now with the whole dataset (no splitting) and use the final parameters obtained to do the testing.

Testing the model

Once we have the final parameters, we will use them to make one prediction for each sample of the "test.csv" file and upload the results to kaggle to compare the predictions with the real labels.

Submission and Description	Public Score
results.csv 11 days ago by Alan Macías	0.77033

We got 77% accuracy which is not the best, but it is still a great problem to solve using logistic regression and get familiar with machine learning basics.

Conclusions

Logistic regression is probably the simplest model used to predict the class that a sample will have, but in some cases the simplicity of the algorithm limits its performance when facing more complex problems.

There are some things that could have been done differently, like tuning the hyper parameters and trying cross-validation for each change to pick a more stable model, but the purpose of this report was to help understand important topics to get started with machine learning and successfully implement a logistic regression algorithm.

Titanic problem available at:

https://www.kaggle.com/c/titanic

Logistic regression ode available at:

https://github.com/maci2233/machine_learning/tree/master/Logistic_regression/titanic

REFERENCES

TITANIC: 'ICEBERG RIGHT AHEAD'. (n.d.). Retrieved March 20, 2019, from https://www.ultimatetitanic.com/the-sinking

Swaminathan, S. (2018, March 15). Logistic Regression — Detailed Overview. Retrieved March 20, 2019, from https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc

Logistic Regression. (n.d.). Retrieved March 20, 2019, from https://ml-cheatsheet.readthedocs.io/en/latest/logistic_regression.html

Brownlee, J. (2016, October 31). How To Implement Logistic Regression With Stochastic Gradient Descent From Scratch With Python. Retrieved March 20, 2019, from https://machinelearningmastery.com/implement-logistic-regression-stochastic-gradient-descent-scratch-python/

Brownlee, J. (2018, May 23). A Gentle Introduction to k-fold Cross-Validation. Retrieved March 24, 2019, from https://machinelearningmastery.com/k-fold-cross-validation/

Titanic: Machine Learning from Disaster. (n.d.). Retrieved March 20, 2019, from https://www.kaggle.com/c/titanic/data