Victor Motogna
Advanced Methods in Data Analysis

Logistic Regression
Experimental results on 2 different implementations

Predictive data analysis

Predictive data analysis plays an important part in every industry. Interpreting data to predict behaviors in the future, based on the information from the records, can be a very powerful tool for multiple use cases. In order to make predictive analysis applications a little bit clearer, we will cover a couple of examples, that will be relevant and applicable considering the approaches presented in the paper.

Detecting fraud/spam in the online environment is a very appreciated featured, as cybersecurity is becoming a greater concern every year. By having data on email or message patterns, applying predictive analysis methods can result in anticipating a fraud/spam message. A pretty different approach can be adopted in the retail industry. Big companies, with lots of information on previous years sales, deposit statistics and client data, can use this information to forecast inventory and manage resources. Having these details can be a huge factor in saving up money and working more efficiently. Both examples are already applied in multiple industries. Depending on the human resource, data and needs, companies apply different predictive analysis algorithms.

Regression in Data Analysis

Regression techniques are widely used in data analysis problems, as a form of predictive modeling, investigating the relationship between a dependent (target) and independent variable(s) (predictor). The main use cases include forecasting, predictions or finding the casual effect

relationship between variables - for example, finding the relationship between driving under the influence and the number of accidents, or average income and education level.

There are two main benefits of regression analysis: indicating the significant relationship between dependent and independent variables, and indicating the strength of impact of multiple independent variables on one dependent variable. This will allow the data scientist/analyst to compare the effects of the variables, measured on different scales. These results can produce data for market researchers, data analysts or data scientists in their field of research.

There are multiple kinds of regression techniques, separated by three main differences: number of independent variables, type of dependent variables and the shape of the regression line/graph). Two of the most popular and used regression techniques are the Linear Regression and the Logistic Regression.

Logistic Regression

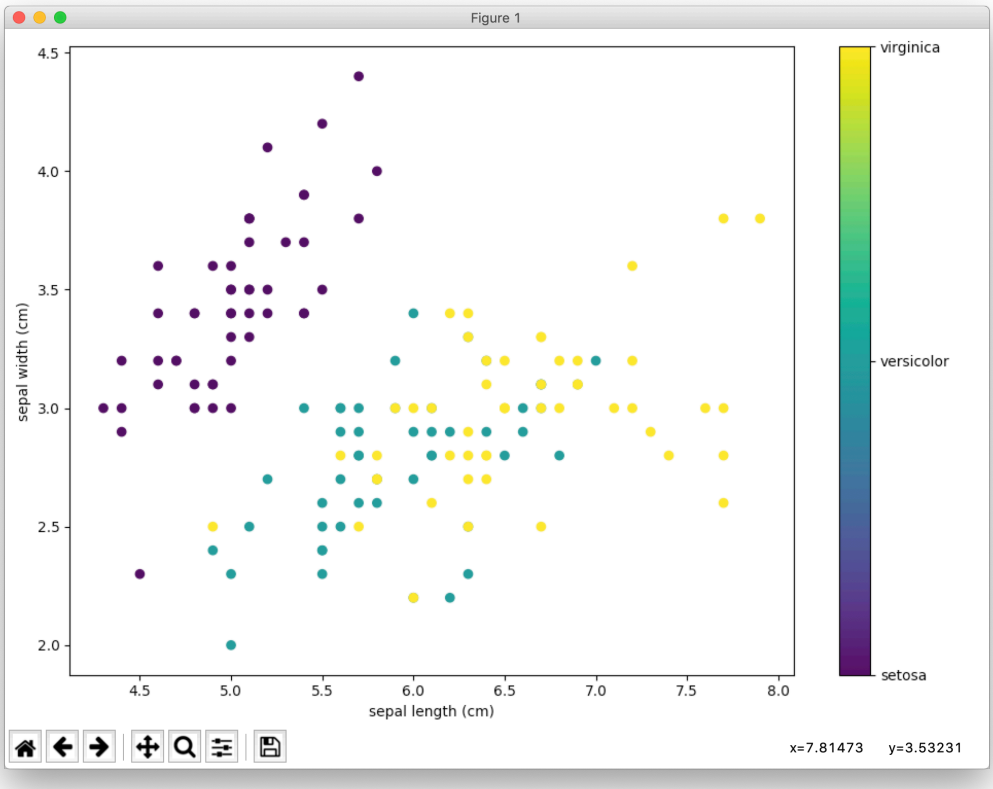
Logistic regression is a widely used predictive analysis technique, that gained a lot of popularity in the early years of the 20th century, with most of its applications in social sciences and biology. One of the most known applications nowadays would be categorizing a mail as being spam or not.

The logistic regression technique is somewhat similar to linear regression, in the matter that it involves a dependent variable - the value that will represent the prediction. The main characteristic of logistic regression would be that the dependent variable represents a binary value (0 or 1, true or false, yes or no, male or female, etc.). The formula for applying this can be started from the linear regression formula: if we take a linear regression formula as $Y=b_0+b_1*X_1+..b_k*X_k$, then consider a sigmoid function $P=1/1+e^{-y}$ and then put them together, we will get the formula we can use for logistic regression.

Experimental results

Dataset

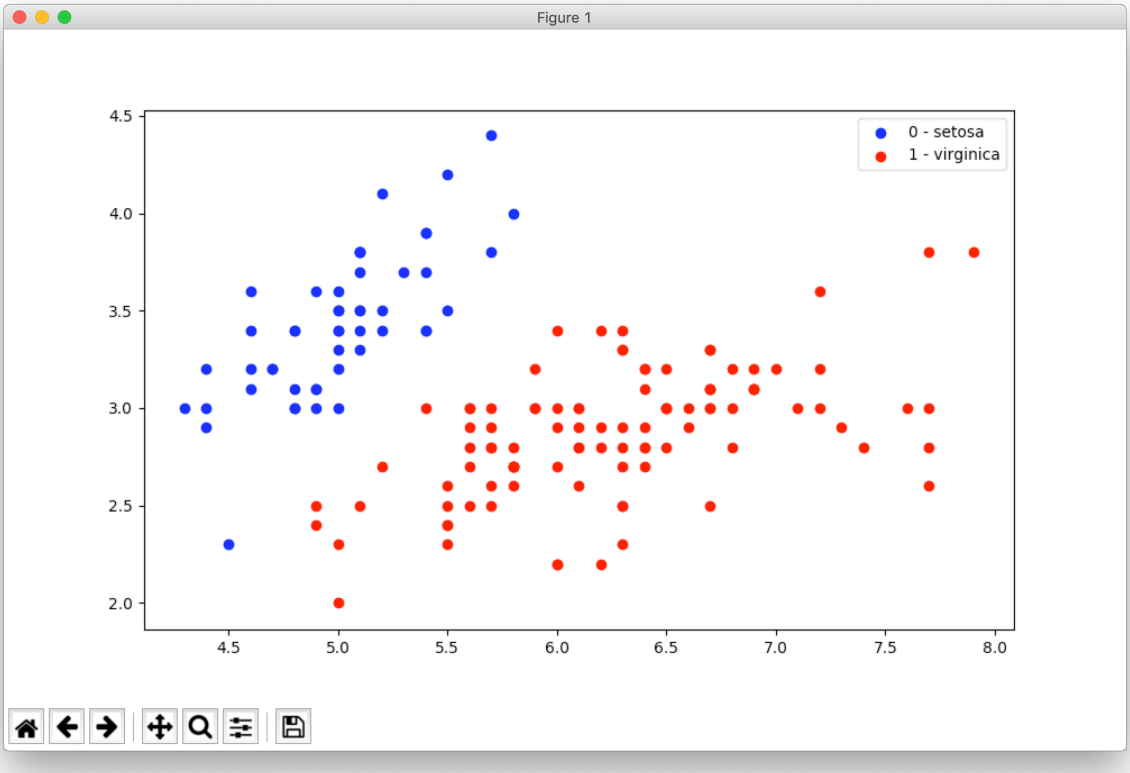
To cover the complete process of getting the results in this paper, we will also mention the dataset, code and put into comparison two different implementations: from scratch and using an existing implementation from a well known library: scikit learn.



[Figure 1 - Iris Dataset representation in regards with sepal length & width]

First, we will cover the dataset. The experiments we have conducted are using the well-known Iris dataset [1]. It contains 150 records, containing 3 variations of Iris flower species - Iris Setosa, Iris Versicolour, Iris Virginica. Each data entry is complete (the dataset has no missing

values) and has 4 attributes: sepal length , sepal width, petal length and petal width (all measured in centimeters). As mentioned above, the dependent variable for logistic regression is always binary, which led us to considering only two species (in our case, Setosa & Virginica [2]). Also, because of the reduced dataset (only 50 entries for each species) and the need to prove performance and accuracy differences between the 2 approaches on a powerful machine, we initially took into consideration all 4 attributes when computing the predictor value, but also only the sepal length and width. This led us to having less actual valuable data to get to the dependent variable, and influenced the accuracy in some test cases.



[Figure 2 - Iris Dataset representation for Iris Setosa & Iris Virginica in regards with sepal length & width]

Implementations

This paper covers two different implementations of the logistic regression technique, in order to compare the accuracy and performance results. The implementations are very different: first one is a self implementation from scratch of the algorithm, the second one is an already existing, optimized approached implementation from the Python library scikit learn.

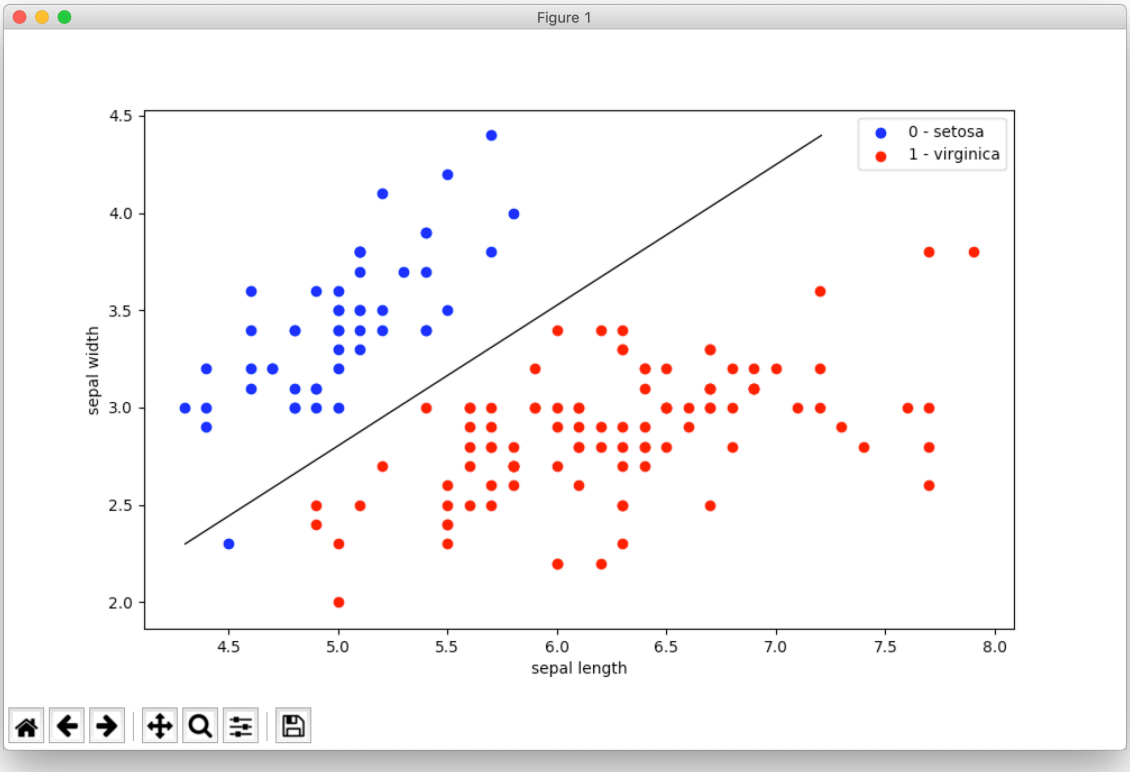
The library implementation was pretty straight forward. We have used scikit learn, a well documented data science library, that contains many performant implementations of widely used algorithms. For this approach, we have created an instance of the LogisticRegression class from the library, then a set of methods called fit and predict, in order to get to the result. The accuracy of the instance was computed using the score attribute from the class.

For our implementation, we wanted to take all things from scratch. The general idea was to take a set of inputs X from the dataset and assign them a value: 0 or 1. In order to do this, we have created a class by our own for the logistic regression algorithm, that contains a sigmoid method: the function that outputs values between 0 and 1, assigning the dependent value. Besides this vital method, we also needed a loss function (that will assign weights, starting from random values), a function to measure the gradient descent (that will compute the derivative of the loss function with respect to each weight, in order to minimize the loss function) and two prediction functions that will assign a value between 0 and 1 given by the sigmoid to the value needed (0 or 1 - this will differ based on the problem and the binary values chosen).

Run environment

As mentioned above, we used only two Iris species and split the experiments in two different approaches: using all the attributes in the dataset and using only the sepal length and width. To get relevant accuracy and performance results, we also needed to consider the iterations and the machine that would run the program. The machine was using a 2.7 GHz Intel i5 dual core processor, paired with 8 GB of DDR3 RAM memory. All experiments were run in the same environment: macOS and Python 3.6. In the implementation we have measured everything by

the implementation (from scratch vs. from a library), prediction accuracy and time needed to get to the result. Both methods used the same number of iterations: 1.000, 50.000 and 300.000.

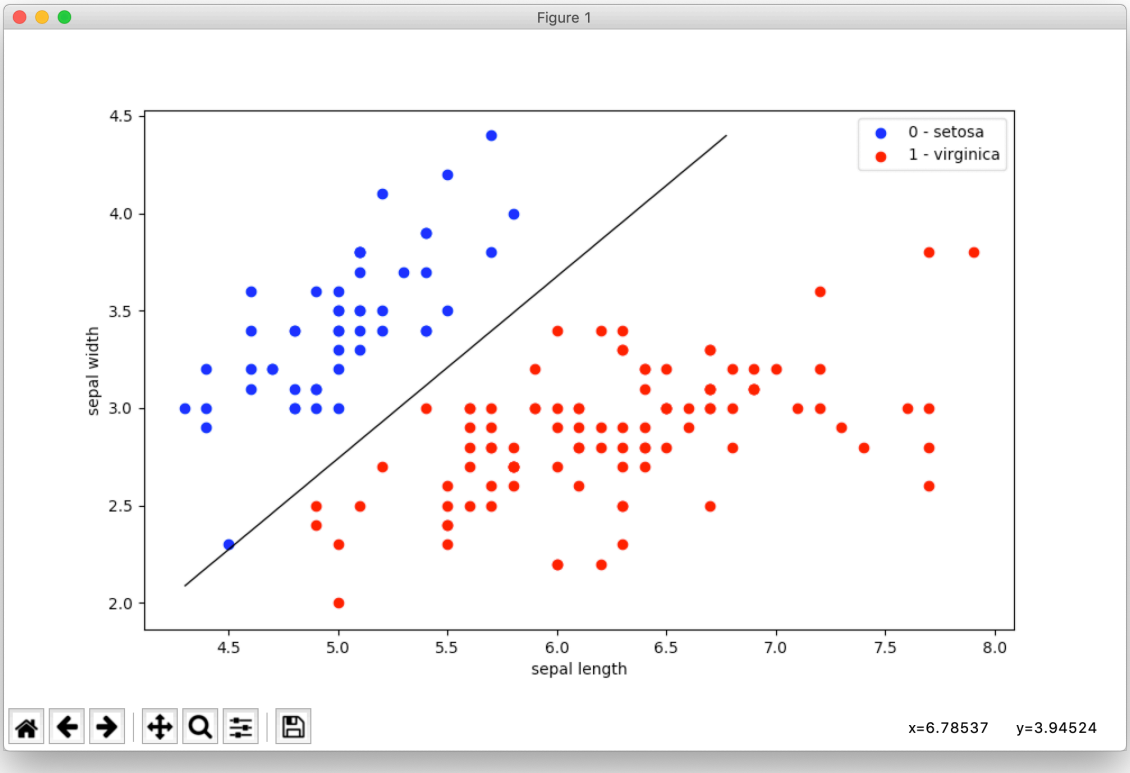


[Figure 3 - Representation of the results using 50k iterations, only taking into consideration sepal length & width]

Results

The results noted that the library implementation was not only far more performant, bar also more accurate (in most cases). Besides when using all 4 attributes and when going over 200.000 iterations, our implementation didn't get to 100% accuracy and was also considerably slower. Because running with only 1000 iterations and taking into consideration all 4 attributes got 100% accuracy on both implementation, it was the only experiment conducted using this

approach, and was used only to compare run time. All the results shown below will present the accuracy as a numeric value between 0 and 1 and the run time will be shown in seconds.



[Figure 3 - Representation of the results using 50k iterations, only taking into consideration sepal length & width]

On 1000 iterations, all 4 attributes, both implementations had an accuracy of 1, while our approach needed 00,052889 seconds, the library one needed 00,009124. This pattern of our approach being much slower can also be observed in the next results, that will use only 2 attributes (sepal length and width), instead of all four. This also had an influence on the accuracy for our algorithm. Using only 1000 iterations, the first implementation had an accuracy of 0,9(3) and a duration of 00,061029, while the second one had an accuracy of 1 and a duration of 00,010585. On 50.000 iterations, we have gotten the exact same accuracy values from both

approaches - 0,9(3) and 1 - but different run times: 2,214487 and 0,014063 seconds. Our algorithm took over 2 seconds to run so many iterations, while other one only took 0,01 second. A greater difference can be seen when using 300.000 iterations: our algorithm needed 15.376322 seconds to get an accuracy of 1, while the other one only needed 00,019008 (and of course, keeping the accuracy of 1).

In Figure [3] we can observe how on 1.000 and 50.000 iterations the accuracy for our implementation was not 100%. We have plotted a straight line (similar to when considering linear regression) that separates the two results and creating a binary response: what's above the line vs. what's under the line.

Figure [4] shows us how using 300.000 iterations creates a different graph, including that exact result and having 100% accuracy.

Conclusion

In conclusion, the two implementations proved to be very different in both accuracy and compile time. A very important thing we have learnt was that we needed to take into consideration also the approach that used only part of the attributes, in order to see differences in the accuracy value.