Modern Scientific Methods in Physics
Modelling Perceptrons using the Iris Data Set

Domonkos Haffner (S24Q2W)

March 19th 2021

Contents

1 Introduction

The research I chose for the semester was to create a simple perceptron model in $C++$ and find an appropriate data set to test it with. I chose the Iris data set, since it's perfectly adequate for the simplest perceptron model.

2 Iris data set

The Iris data set is is a multivariate data set, consisting of three different types of irises' petal and sepal lengths. The three species are the following:

1. Iris-setosa,
2. Iris-virginica,
3. Iris-versicolor.

The data set is a 150×5 matrix, containing the sepal lengths, sepal widths, petal lengths, petal widths and the names of the irises. [1] [2] I chose to separate the Iris-setosas and Iris-versicolors from each other with the perceptron model, however any combination of these three flowers can be easily chosen, one simply needs to load in the different data into the $C++$ code.

Let's take a look at figure 1, where the different properties of the irises are represented as a scatter plot. As it's seen, the properties are very distinguishable for Iris-Setosa and Iris-Versicolor. This means that the simple perceptron algorithm should be very effective and only after a couple iterations, the results should be accurate. Note, that figure 1 was created by Python.

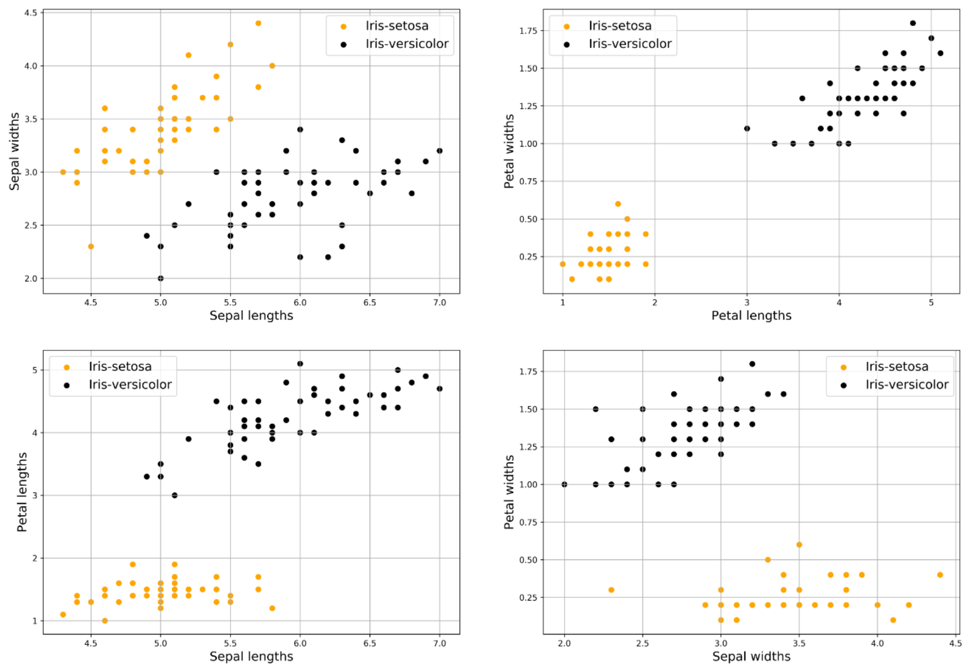

Figure 1: Scatter plot of the different properties of the irises.

That data set can be easily found and downloaded from the internet. [3]

3 The model

After loading in the data set, I had to choose which irises to use. As mentioned above, I used the Iris-setosas and Iris-versicolors. I separated the values for the different properties and loaded it into the $C++$ code with a function.

Since this is a very basic Machine Learning project, the data set must be separated into training and testing data sets. I came up with an algorithm, which uses every second element in the properties' array as training- and every fist as testing data, or vice-versa. This is split by a function, which has two arguments; the data array and an integer which can either be zero or one. If it's zero, the return of the function is an array with the even-, if it's one, then the odd number indexed elements. All these lists of values are saved in $std :: vector < double >$ objects. Since the property vectors are consisting of double values, and the label vector is consisting of string values, the *train_test_split* function must be written as a template function.

After splitting the data set, the perceptron model must be created. This is described by the following equation:

$$\text{classification} = Act(\underline{X} \cdot \underline{w} + b), \tag{1}$$

where X is the property vector for each iris, w is the weight vector, b is the bias and Act is some kind of activation function. These weights and bias are initialised as zero. The activation function I'm using is the following:

$$Act(x) = 1, \text{ if } x > 0, \qquad Act(x) = -1, \text{ if } x \leq 0, \tag{2}$$

where 1 represents the Iris-versicolors and -1 the Iris-setosas. After the classification, these results are used to update the weights in the following way:

$$\text{update} = \alpha \cdot (\text{y\_real[j]} - \text{y\_predicted[j]}), \tag{3}$$

$$\text{weights} += \text{update} * \text{x\_train[j]}. \tag{4}$$

where α is the learning rate, j is the row index of X and *x_train* a 4×1 vector containing the four properties of the irises.

After updating the weights, the classification function is called again, so a new prediction for the *y_train* data set is created. This iteration is repeated several times, which is set by the *epochs* variable. After the set epochs, the program finishes running and the weights are returned. These weights can be later used to predict the *y_test* classification, which can be compared with the actual testing labels.

4 The results

As it's seen from the equations, since the first bias and weights are all zero, every single classification will be -1 after the first iteration, which causes a 50% accuracy. This is due to the fact that half of the data set is classified as Iris-setosa and half is as Iris-versicolor. This is, of course not a very accurate model yet, so it's highly advised to set the epoch number greater than 1.

After the first iteration, the weights and bias are already non-zero, which can be used to recalculate them. As mentioned before, these values are quite well separated, thus the classification is already perfect after the second iteration.

5 Conclusion

In this research, a simple perceptron model was created to classify the different iris types into two groups; Iris-versicolor and Iris-setosa. This task was successfully done with mainly using $C++$ and using Python for plotting.

As assumed in the beginning of the research, the different iris types can be easily predicted from the four properties. Two is already a sufficient epoch number, for which the model predicts 100% of the iris types accurately.

Multiple different learning rates were tried out during the computations, however, this did not influence the results whatsoever. The reason is, that when classifying the different iris types, everything above zero belongs to the Iris-versicolor group, while everything below zero belongs to the Iris-setosa group. This means, that when setting the weights and bias from zero, the smallest change in the correct direction already puts all the values in the correct classification. This results in having a 100% accuracy for the iris type prediction after the second iteration.

References

[1] Wikipedia - Iris data set:
https://en.wikipedia.org/wiki/Iris_flower_data_set

[2] Scikit-learng documentation - Iris data set:
https://scikit-learn.org/stable/auto_examples/datasets/plot_iris_dataset.html

[3] Github - Downloading Iris data set:
https://gist.github.com/curran/a08a1080b88344b0c8a7