Luca Defilippi, Alessio Bollea

Data analysis and data mining

Report 3

PERCEPTRON

A perceptron is a computing element that generates an output as the weighted sum of its inputs. It's considered a very simplified version of an artificial neuron. We can also insert a bias, that is directly added to the weighted sum.

To make the perceptron a binary classifier we can't use this output: we need the perceptron to generate only two values, so we can consider as its prediction the result of a particular function that takes the output of the perceptron as argument. This function can be, for example, the *sign* function (to generate only values in the interval $\{-1;+1\}$) or the step function (to generate only values in the interval $\{0;+1\}$).

So, using the *sign* function, considering a generic number of inputs and writing the weighted sum as a scalar product, the perceptron prediction equation can be written in this way:

$$f(\underline{x}) = sign(\underline{w}^T \underline{x} + b)$$

Where $\underline{w}$ is the vector of weights and $\underline{x}$ is the vector of inputs.

PERCEPTRON LEARNING ALGORITHM

To tune the weights and the bias, an idea can be starting with random values. Then you iterate through all the entries of the dataset and you try to predict the label of that entry (with index k) using the current weights and bias. You update the weights and the bias in the way such that:

$$\underline{w}_{k+1} = \begin{cases} \underline{w}_k & \text{if the prediction is correct} \\ \underline{w}_k + y_k \underline{x}_k & \text{otherwise} \end{cases}$$

<u>Note:</u> y is the expected label of that entry.

So, the weights and bias are the same if the prediction is correct, otherwise they're updated.

You terminate when you no longer have prediction errors.

Here the bias is considered as part of the weights, where the input it multiplies is always 1.

It can be proved that, if the two classes are linearly separable, this algorithm converges to the optimal solution in a finite number of steps, if the solution exists.

IMPLEMENTATION

The only dataset which has feature values not between 0 and 1 is the *iris* dataset, so the first thing we do is normalizing the data, if needed. To do it, we iterate through all the features; for each feature we search its minimum and maximum values. Since we want to normalize between 0 and 1, we find the equation of the line that passes through the points (min feature value, 0) and (max feature value, 1), to make a linear mapping between 0 and 1.

The *iris* dataset is also the only one which has 3 classes. Since we can only distinguish linearly separable classes and we know that class1 (labelled as 1) is linearly separable respect to class2 + class3 (labelled as 2 and 3), we transform this particular classification problem into a binary classification one by setting the labels of classes 2 and 3 as –1.

Then we randomly sample the dataset to create a learning set and a validation set, given a percentage which represents the dimension of the learning set proportionally to the dimension of the dataset. To avoid this sampling, it is sufficient to set the percentage as 100% to create a learning set which is exactly the whole dataset.

For the Perceptron Learning Algorithm, we start by assigning random values between –1 and 1 to the weights and the bias. Then we first compute the number of errors before even starting the learning phase: we make a prediction for every entry of the learning set and we store all these predictions in a vector; then we create a vector of boolean values, where every value is 0 if the sign of the prediction label agrees with the sign of the expected label (right), 1 otherwise (wrong). The error is computed as the sum of these logic values of the array, so we count the number of prediction errors.

Then we enter in the learning loop, where we consider every entry of the learning set and we see if the perceptron is able to correctly predict the label of that entry; if not, we tune the weights and bias values as said before and we update the number of errors. We keep on iterating until we get 0 prediction errors.

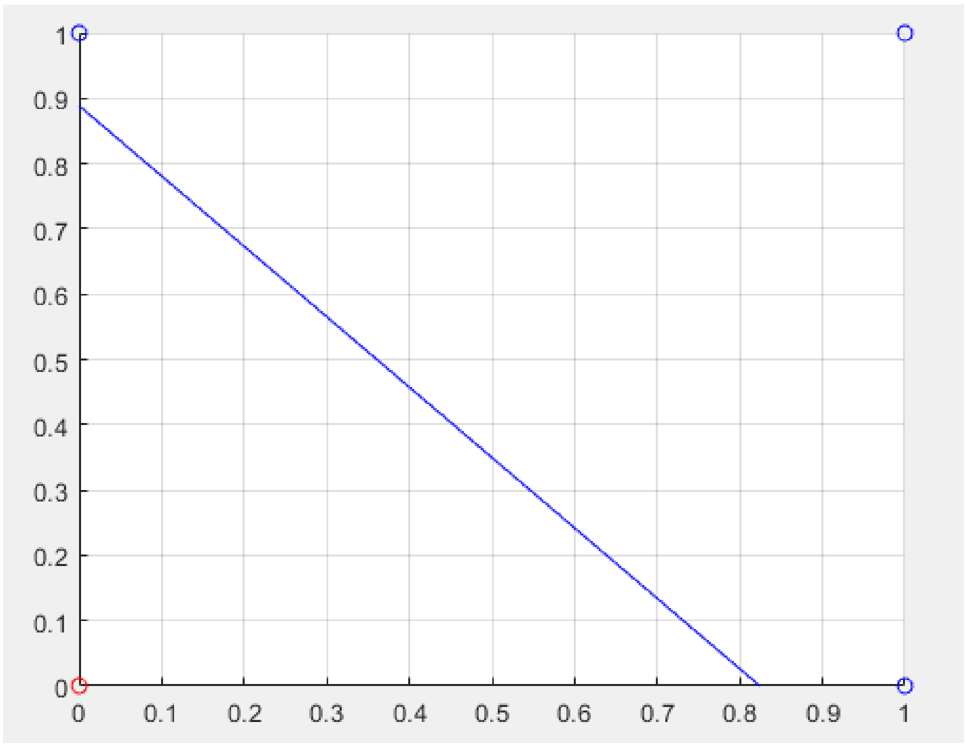
At the end of the learning loop we have the best values for the weights and the bias, so we can use them to make the validation step on the validation set, by making the same computations for the predictions and the error as in the case of the learning set.

RESULTS

The *or* dataset is the simplest. It has only 4 points, which are all the possible combinations of values for two boolean variables. The dataset is in the form:

X1	**X2**	**X1 OR X2**	**LABEL**
0	0	0	-1
0	1	1	1
1	0	1	1
1	1	1	1

Since there are few points, the algorithm finds the linear separator really fast. Here's the plot of the result, drawing the points as circles in red for the –1 label and in blue for the +1 label.

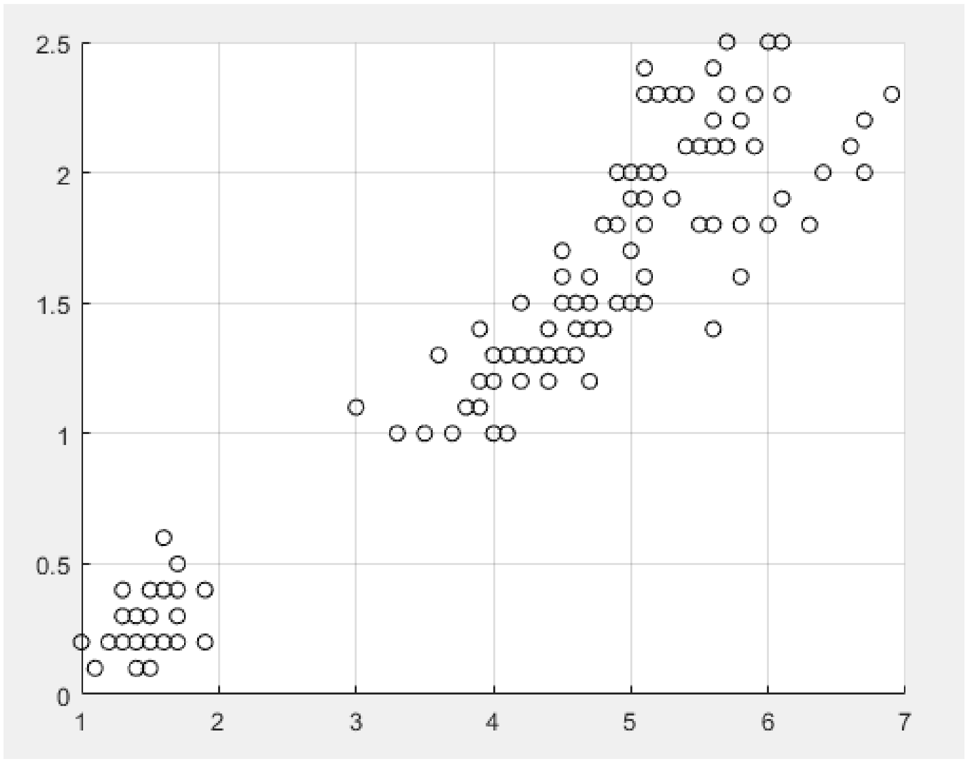


It needs, on average with 10 repetitions, only 9 iterations of the learning loop. The lowest number of iterations that we got is 5 and the highest is 17, considering that we start with random weights and bias.

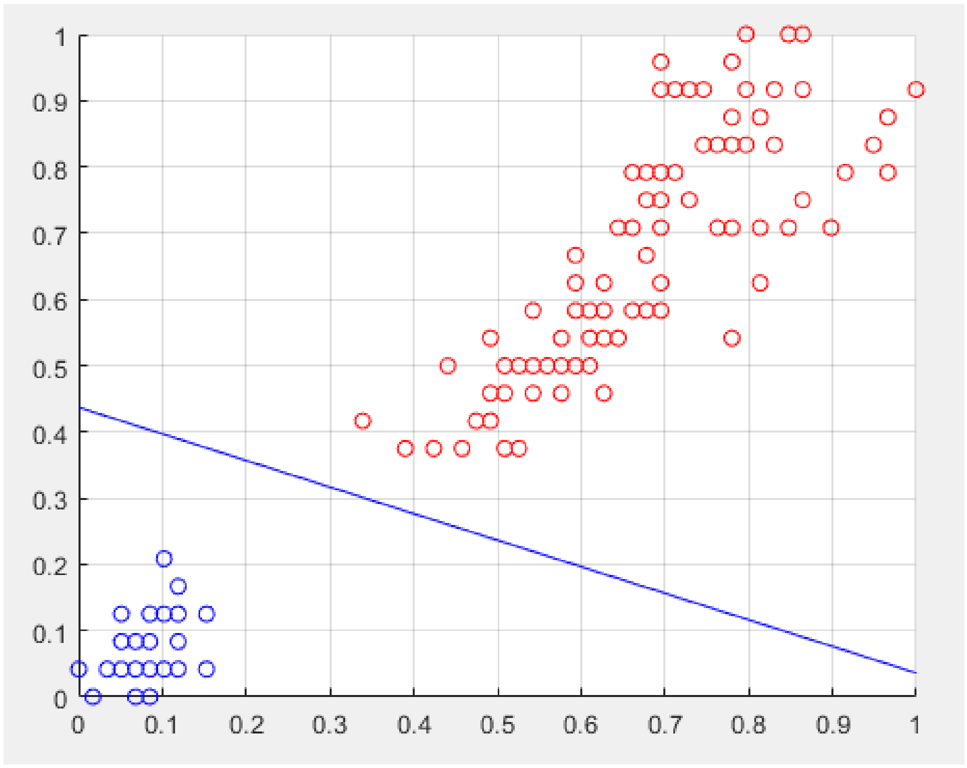
The *xor* dataset has the same points, but the last label (when $X1 = X2 = 1$) is –1, since $1\ XOR\ 1 = 0$.

This creates a problem, because in this case the two classes are no longer linearly separable, so the algorithm cannot find a line that separates them. In fact, it stays inside the learning loop for an infinite amount of time, since there is always at least one prediction error. This problem can be solved using a quadratic classifier, that can be created by adding a quadratic term (weighted sum of the products between every couple of inputs) to the perceptron, so it can learn quadratic separators. An example of solution can be a surface that includes the two points in the intervals $(0,1)$ and $(1,0)$ and doesn't include the points in the intervals $(0,0)$ and $(1,1)$.

The *iris* dataset is a little more complex. It's again in only two dimensions but it has more points and they're not normalized. Here's a plot of the dataset:

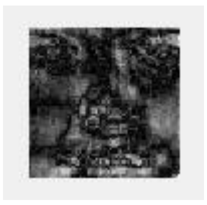


For this dataset the average number of iterations required to get 0 prediction errors is 315 on 10 repetitions. The highest number of iterations was 601, while the lowest was 0 (we've been lucky with the random generation of weights and bias). This is one of the linear separators we got (after normalization):



The *face* dataset contains 90 images of dimension 60x60 pixels of both male and female faces; every image is represented as a vector of 3600 values, where every value is a pixel of the image. Here's an example of male and female faces:

The first thing we can do is see how much time it takes to learn the entire dataset: it starts to take a few iterations to get 0 prediction errors: with 10 repetitions we got an average of 80400 iterations (~ 1.83 seconds). Another interesting thing is plotting the weights values (in absolute value) as an image, mapping the values to grayscale values. You can see the result below:



As we can see, it looks like a face: we can distinguish the traits of the eyes, nose and mouth. The darkest zones are the most relevant for the classification, since the corresponding weights are larger there. It's like this *image* is used as a filter, to be applied to the image we want to process.

Then we can try to randomly divide the dataset in two halves to create a learning set and a validation set and check the number of prediction errors made on the validation set: we made 10 attempts and we got 10, 11, 6, 9, 8, 6, 7, 6, 3, 6 errors, which corresponds to an average of 7 errors. Since the validation set has 45 entries, the average accuracy, which is pretty good, is:

$$\frac{45-7}{45} = \frac{38}{45} = 84.4\%$$

We can do the same thing with another multi-dimension dataset, the *sonar* dataset. It has 208 entries and 60 features. The higher number of entries has made the learning phase on the whole dataset a bit longer, in fact it took almost 1 minute to get 0 prediction errors. If we, instead, split the dataset in learning and validation set and we compute the errors, it takes 0.2 seconds to learn. We got 20, 29, 26, 29, 31, 32, 24, 31, 27, 31 errors, which corresponds to an average of 28 errors. The validation set has 104 entries, so the accuracy on average in this case is $\frac{104-28}{104} = \frac{76}{104} = 73.0\%$, which is lower than before. Probably the classes are closer to each other than in the previous cases. If we increase the dimension of the learning set to 70% of the number of entries (146 entries for learning, 62 for validation) we get an average error of 15, so an average accuracy of $\frac{62-15}{62} = \frac{47}{62} = 76\%$, which is a little better but, probably, to better separate the two classes, we'd need more data.

LINK TO THE CODE

https://github.com/lucad93/Data-mining/blob/master/Riconoscimento%20Sesso/main1.m