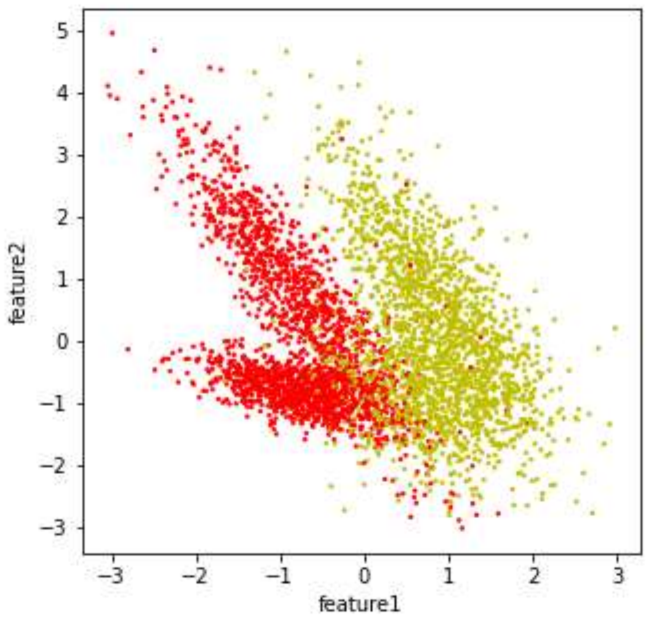
Machine learning

Programming assignment: perceptron classification

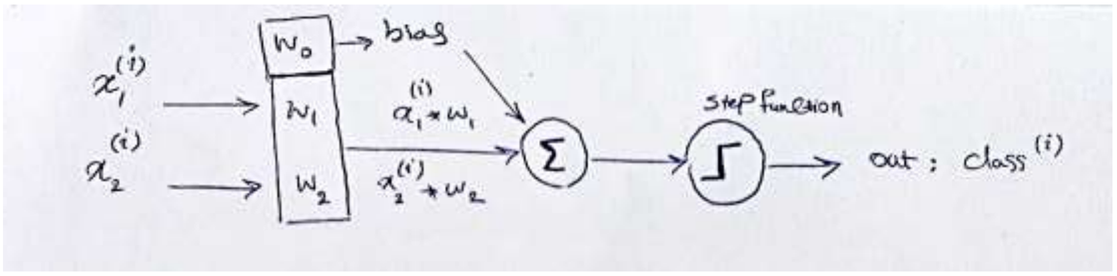
Attached file: 02.ipynb

I read the training data from the train.csv file as a pandas library dataframe. For a small part of the calculation, the numpy library was also added and I used matplotlib.pyplot to draw graphs. The problem data matrix contains three columns, the first two columns are $x^{(i)}$ and the nominal value is the data label as the third column. 4000 training data are available. The problem is two-class and the contents of the label column, like any data, have a value equal to 0 or 1. To make the process easier, I equated these values to 1 and -1.

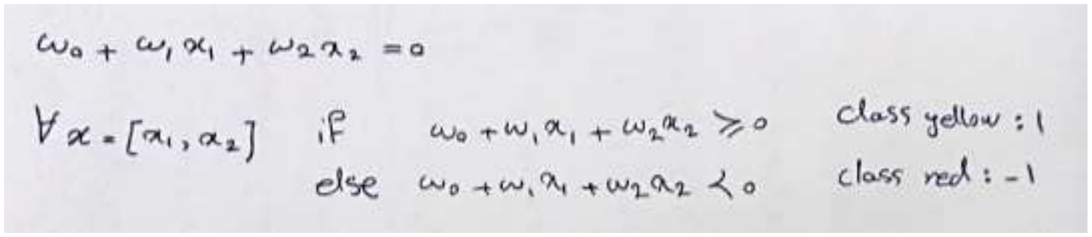
I draw a dot plot of the training data. The group labeled -1 is the red points and the yellow points are the same as the data labeled 1. The vertical axis is like the value of the second column of data or feature2 or $x2^{(i)}$ on calculations, and the horizontal axis is like the value of the second column of data or feature1 or $x1^{(i)}$ during calculations.



The desired model of a perceptron is as follows: (the step function used is the same as the sign function)



That is, we are supposed to find an equation of the line as shown below, which if drawn on this graph, will completely separate the two classes of data as shown below.

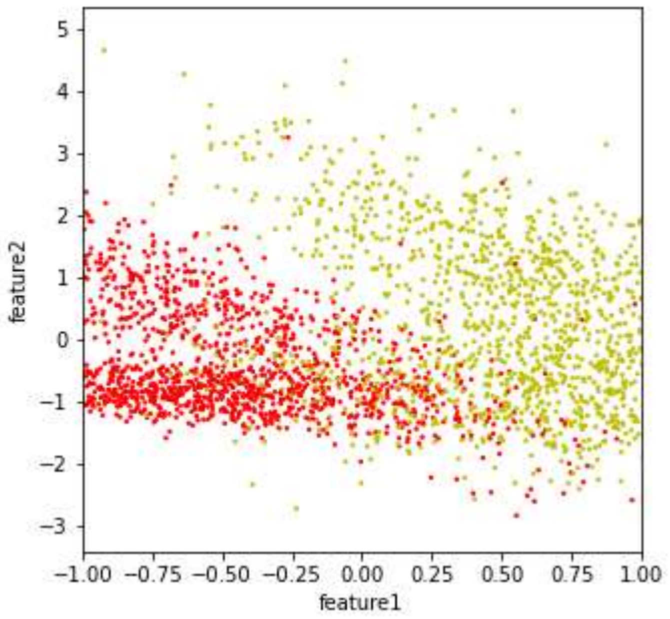


$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

$\forall x = [x_1, x_2]$ if $w_0 + w_1 x_1 + w_2 x_2 \geqslant 0$ class yellow : 1

else $w_0 + w_1 x_1 + w_2 x_2 < 0$ class red : -1

If I look at the diagram again, and especially carefully at the cut below from the domain of feature1, it is easy to see that there is no such line. This means that these data are not linearly interpolated.



So the linear model will never reach 100 accuracy on the training data (it will classify everything correctly), so if I use the perceptron theory algorithm, which stops until there is no data in the wrong category, the algorithm will never stop. In order to find a close solution for the problem with the linear model, I conventionally execute the weight update algorithm up to a certain amount of repetition. Here I chose 100.

I added a column 1 to the data matrix to make the model bias calculations more convenient. which takes the value of x0$^{(i)}$ on calculation.

Now, using the data that was prepared, I run the perceptron learning algorithm. The result is a three-component vector w. This is the line that we expected to separate the data on both sides.

Remember that when I wanted to measure the performance of the model on the test data, the same changes should be applied to that data matrix.

The main training loop is implemented in the train_perceptron function, which takes the prepared matrix and updates the perceptron weights vector from the following relationship until all the data are correctly classified or up to 100 times. where M is the misclassified data set and y is the label value.

$$W_{new} = W_{old} + \eta \sum_{m \in M} x^{(m)} y^{(m)}$$

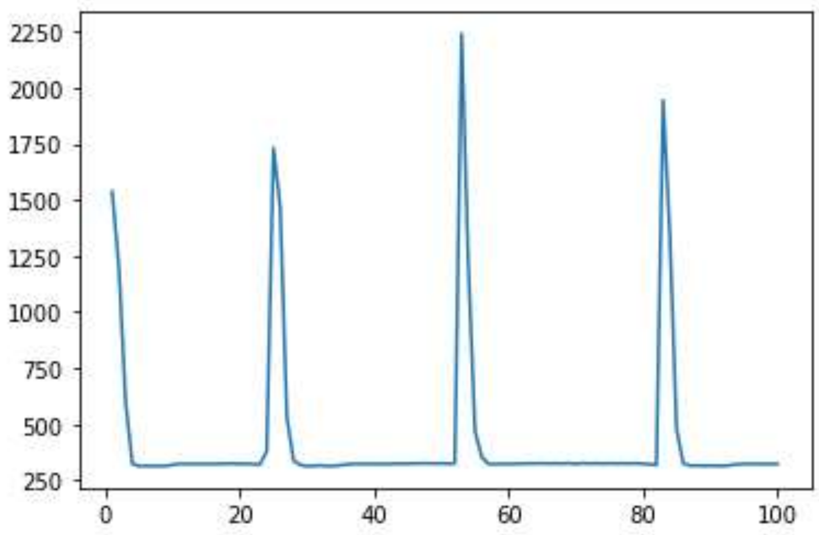
$$\begin{bmatrix} w0_{new} \\ w1_{new} \\ w2_{new} \end{bmatrix} = \begin{bmatrix} w0_{old} \\ w1_{old} \\ w2_{old} \end{bmatrix} + \eta \times \begin{bmatrix} \sum_{m \in M} y^{(m)} \\ \sum_{m \in M} x_1^{(m)} y^{(m)} \\ \sum_{m \in M} x_2^{(m)} y^{(m)} \end{bmatrix}$$

I chose the initial value of the weights randomly. Each time, with the weight vector obtained from the previous step, it performs classification on the entire training data with the classify function, and the result of the class obtained from the following relationship is stored on a column named assigned_label from the training data frame.

In the next step, it is necessary to compare this column with the label column and place the conflicting result on another column called missclass. This is done in the find_missed function. All data that has a true value on this column is used to update the weight in the current iteration of the loop. Actually, I am using batch mode for perceptron training algorithm. I use a temporary data frame to create the value that should be added and subtracted to the components of w, and finally, I update w with the learning rate set to 0.001. To control the training process, the number of misclassified data is kept in each iteration. The weights obtained:

W0	W1	W2
0.063877	0.0834247	0.226945

I draw the number of data that were wrongly classified on 100 iterations of the algorithm.



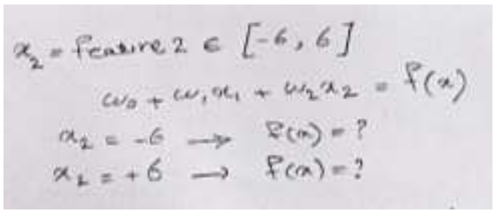
The chart has sudden jumps and descents. In certain iterations, if the training is stopped, the error of the training data is very high. This situation occurs because the data is not linearly separable. And in other parts of the graph, it remains with a very slight slope around the width of less than 500 to the error data. At best, the model has this number of errors on the training data. This error is caused by the low complexity of the model and cannot be fixed with this model.

I read the test data from the test.csv file as a data frame and made the necessary changes.

To evaluate the model on the training and test data matrix, I classify both matrices with the obtained model. After creating the assigned_label and missclass columns, I evaluate the classification result with two functions, measurements and plot_result.

The measurements function provides information about the performance of the model. This function reports the total number of misclassified data and their ratio to the total data and the model's accuracy on that data. Here, I took the accuracy equal to the ratios of the data that were correctly classified to the total data (same as correct_ratio).

The plot_result function is defined to display the data of two classes based on the color of their class on the input data matrix. Display the data that are misclassified in black. A green line is also drawn on the same graph, which indicates the final w value obtained. (I drew this line by applying w weights on the two points at the beginning and end of the vertical axis range)



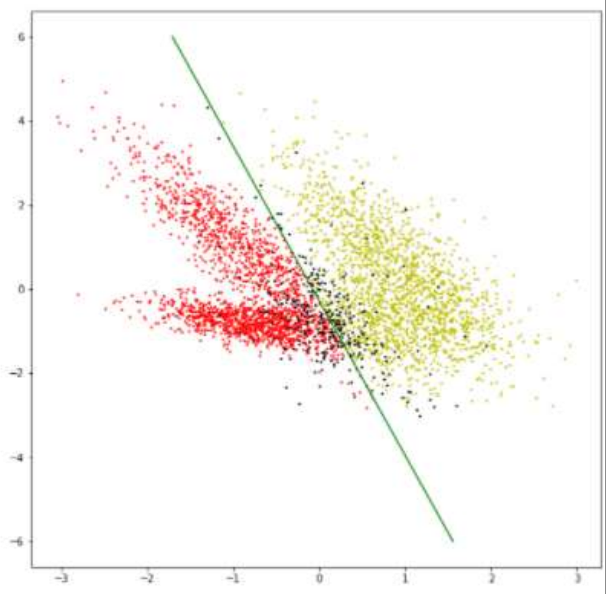
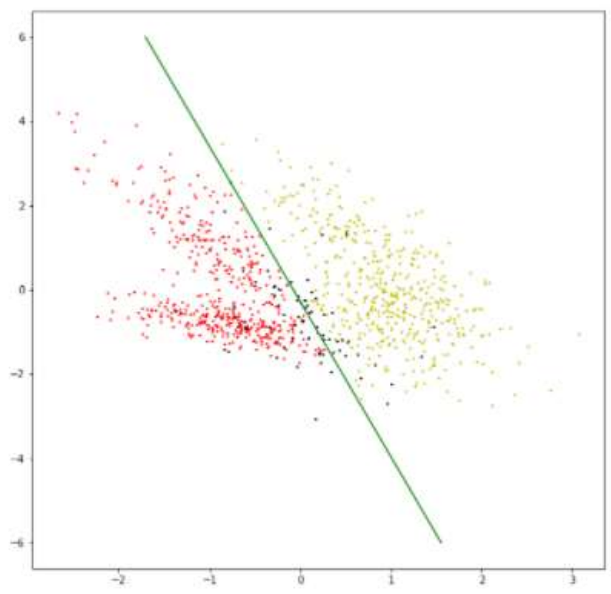
x_2 = feature 2 $\in [-6, 6]$

$w_0 + w_1 x_1 + w_2 x_2 = f(x)$

$x_2 = -6 \longrightarrow f(x) = ?$

$x_2 = +6 \longrightarrow f(x) = ?$

The results are below.

train	**test**
total = 4000 total_missclassed = 323 misclassed_ratio = 0.08075 correct_ratio = 0.91925	total = 1000 total_missclassed = 74 misclassed_ratio = 0.074 correct_ratio = 0.926
	

As expected from the figure of the first graph of the report, most of the misclassified data belong to the same part of the overlap of two classes, from which we concluded that the linear model cannot be a complete separator of the training data. If it wasn't for the inherent entanglement of these two classes on the screen, the model could do a better classification.

The total test and training data are 1000 and 4000, on each of which we still have 7 and 8 percent errors, respectively. which is also an acceptable stable number for Esen data. As expected from the blue graph of the number of wrong classifications in the training process, the total number of numerical training errors below 500 times 323 was obtained. If we consider the percentage of correctly classified totals on the test as the final value, this model can correctly classify 92% of the data of this problem, which is a valid number. But if the distribution of the test data was such that it mostly included the points of the two classes, the performance would drop, but we always choose the test and training data from the same distribution, and the reason for the closeness of the test and training results is actually that's it. Here, the final model numerically performed well, but if we look at the graphs, it is clear that we did not use the correct model, and I attribute the relatively favorable numerical result to the fact that although the two classes are confused, this confusion is on the distribution. The data is a small number (the first graphs of the report) which may be attributed even to the high noise of the data, but since we take the noise as a random factor, with the words about the similarity of the distribution (only in terms of appearance and not statistical analysis) I put these mixed data (black color on the diagram) on two categories of data, my opinion is that the main problem is the low complexity of the linear perceptron model for classifying and learning this data. This assumption can be checked by building higher order models.