Homework 1. Machine Learning (MIIS)

Marc Juvillà Garcia

October 9, 2016

(a) Generate a dataset of size 20. Plot the examples $\{(x_n , y_n)\}$ as well as the target function f on a plane.

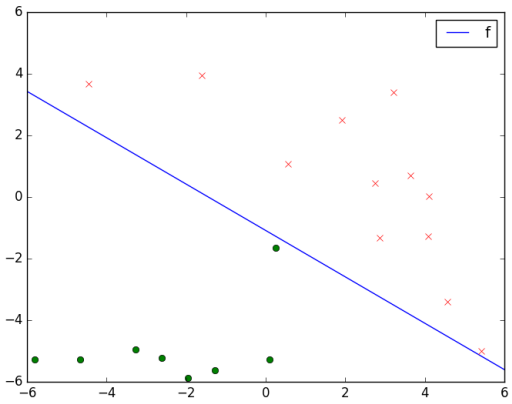

Figure 1: Dataset with 20 points with random f.

(b) Run the perceptron algorithm on the dataset. Report the number of updates that the algorithm takes before converging. Plot the examples $\{(x_n , y_n)\}$, the target function f, and the final hypothesis g in the same figure.

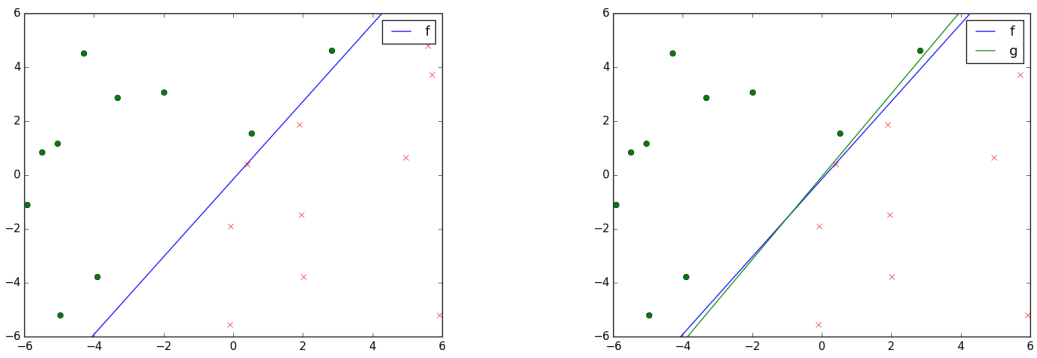

Figure 2: Dataset with 20 points with computed g.

For the model shown in Figure 2, it took 12 updates of the model (that is, correcting 12

mistakes) and 0.292 ms before converging.

(c) Repeat everything in b) with another randomly generated dataset of size 20, and compare the result to b).

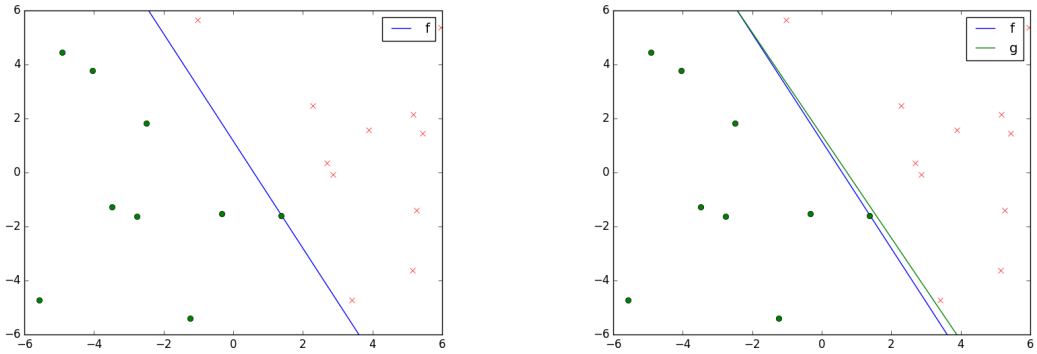

Figure 3: Another instance of the model of the previous question.

It took 16 updates of the model and 0.414 ms before converging, for the shown instance. However, I noticed that these numbers can vary a lot so I computed the average time and update steps for 50 instances of the problem (different dataset and different f, but same number of data points and dimension of the space), which resulted in an average of 12.2 updates and 0.218 ms.

(d) Repeat everything in b) with another randomly generated dataset of size 100, and compare the result to b).

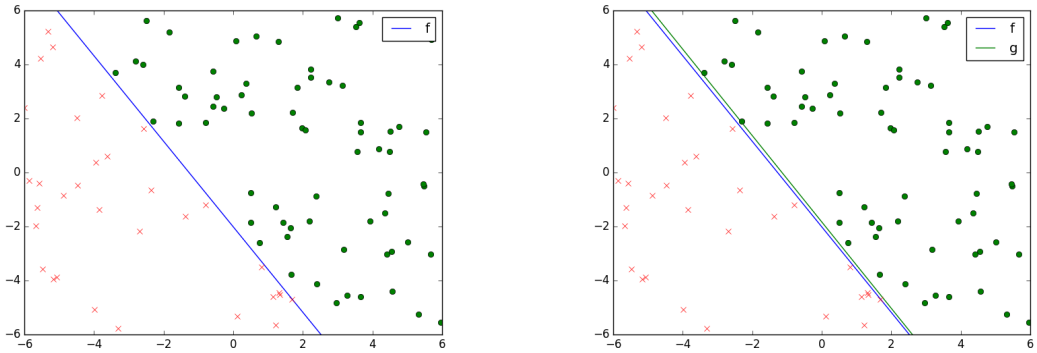

Figure 4: Dataset with 100 random points.

It took 78 updates of the model and 4.221 ms before converging.
Averaging for 50 instances of the problem, it resulted on 87.48 update steps and 2.841 ms of training time.

Comparing these results to the ones in section b, we see that the update steps have increased by a factor of ∼7, and the training time by ∼13.

(e) Repeat everything in b) with another randomly generated dataset of size 1000, and compare the result to b).

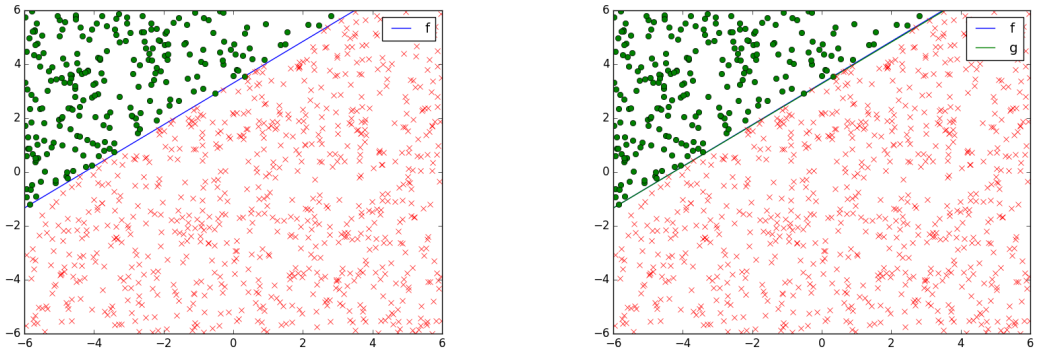

Figure 5: Dataset with 1000 random points.

It took 2176 updates of the model and 195.669 ms before converging.
Averaging for 50 instances of the problem, it resulted on 452.7 update steps and 50.815 ms of training time. Comparing these results to the ones in section b, we see that the update steps have increased by a factor of ∼37, and the training time by ∼233.
Compared to the results of section d, multiplying by 10 the number of samples results in an increase by ∼5 of the number of update steps and by ∼18 of the processing time.

(f) Modify the experiment such that $x_n \in R^{10}$ instead of R^2 . Run the algorithm on a randomly generated dataset of size 1000. How many updates does the algorithm take to converge?

It took 6529 updates of the model and 979.125 ms before converging.
Averaging for 50 instances of the problem, it resulted on 5908.7 update steps and 737.511 ms of training time.

(g) Summarize your conclusions regarding the accuracy and running time of the algorithm as a function of N (the number of data points) and d (the number of dimensions).

The following table summarizes the results of the previous sections.

	N = 20; d = 2	N = 100; d = 2	N = 1000; d = 2	N = 1000; d = 10
Steps	12.2	87.48	452.7	5908.7
Time (ms)	0.218	2.841	50.815	737.511

Table 1: Summarized results.

In terms of accuracy, given that the two classes are linearly separable (we made it that way when generating the datasets), the perceptron training algorithm always finds the perfect

boundary, so the accuracy is 1.
In terms of speed, we can see an increase of time steps when the number of samples is increased, and a stronger increase when we increment the dimension of the space. The following figures try to illustrate this effect.

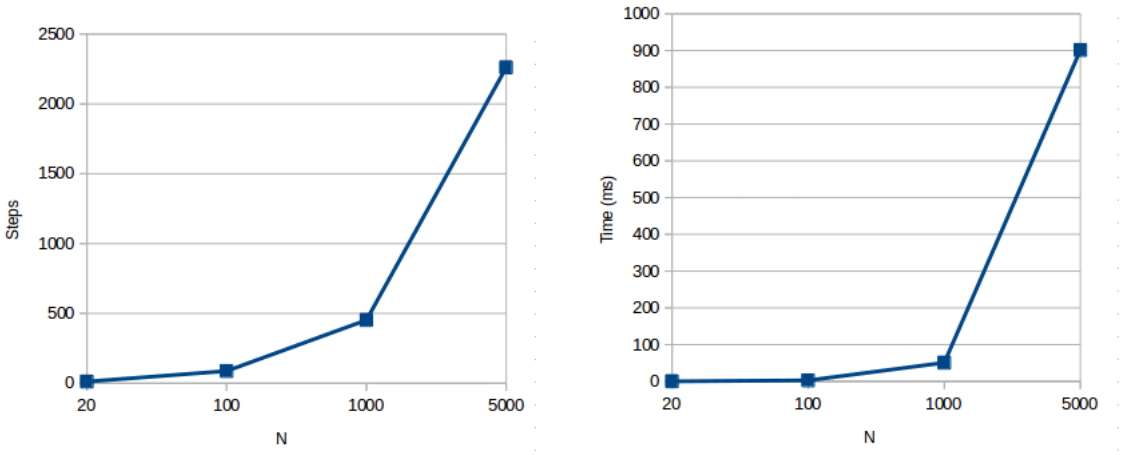

Figure 6: Exponential increase of number of steps and processing time (d=2)

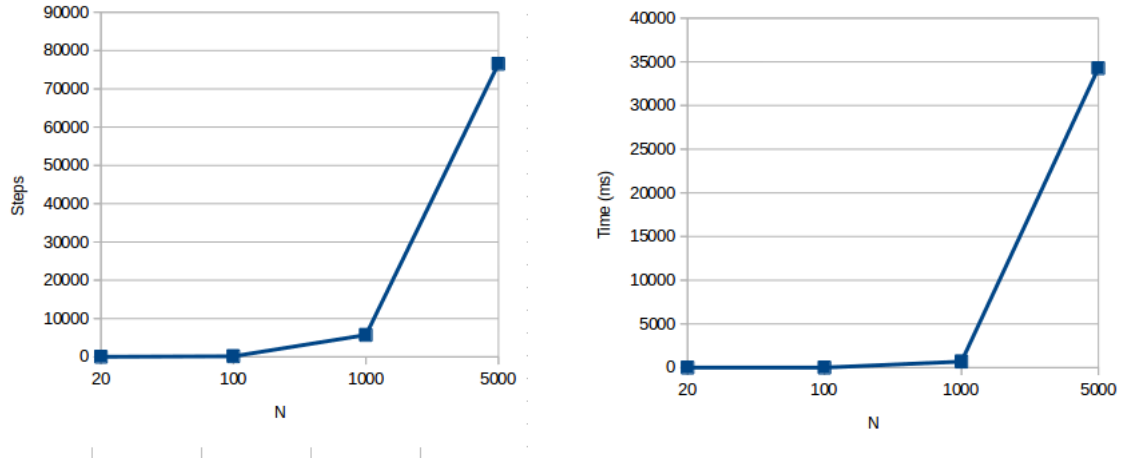

Figure 7: Exponential increase of number of steps and processing time (d=10)

As can be seen in the attached figures, both models present an exponential increase in its number of steps amb processing time when incrementing the data points, but this effect is stronger with more dimensions.

In summary, it was foreseeable that the processing time was going to increment when adding data points to the dataset or adding dimensions to the space - the more data points, the more

(statistically) errors to correct; the more dimensions, the more processing time when computing the perceptron parameters.

Speaking about accuracy, in general terms adding training samples to the dataset helps the training algorithm to generalize better and by extension to avoid overfitting, while adding dimensions to the training samples tends to make the dataset more sparse and so easier to separate in classes. For this reason, we need to find a trade-off between the accuracy and the processing time, adjusting the parameters of the model.