MACHINE LEARNING
Programming Exercise 1

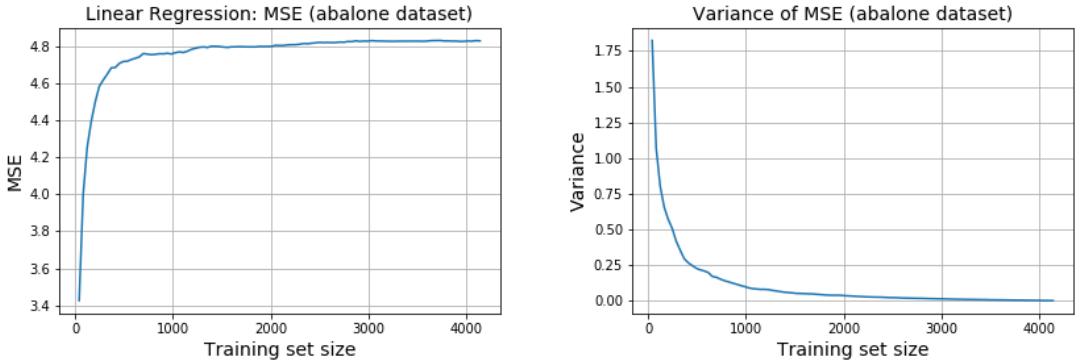
Emma Fraxenet, Josep Ferrer

October 24th 2020

1 REGRESSION TASK

To start with this first task, we picked the Abalone regression dataset (# of features: 8 and # of data: 4,177). All the programming has been done with python, using the Jupyter notebook environment. The linear regression algorithm has been imported from the scikit-learn library. For the figures presented in this report, we applied a statistical approach, averaging over 250 iterations. Each of these iterations started by doing a random shuffle of the data.

- Plot the approximation error (square loss) on the training set as a function of the number of samples N (i.e. data points in the training set).



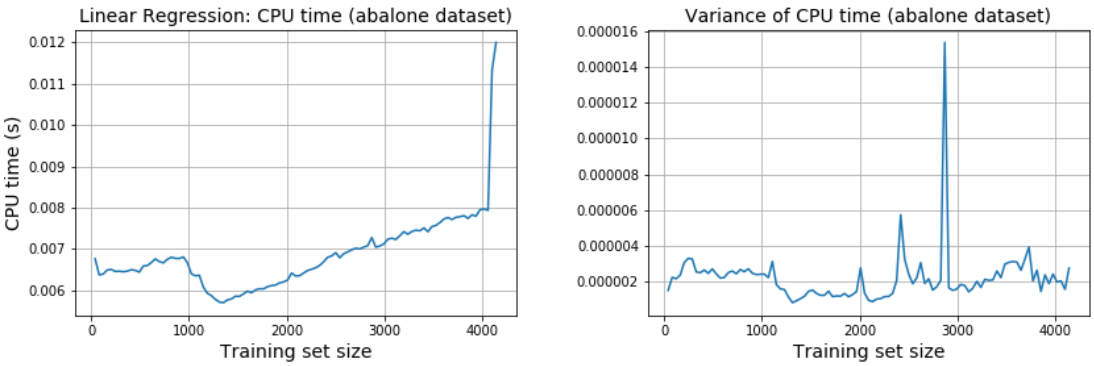
Here we present the Mean Squared Error(MSE) on the training set as a function of its size. As expected, we obtain a saturation curve that presents a logistic-like behaviour. As we already reviewed on class, the Training Loss is expected to increase with bigger values of the training data (N) up to a saturation value (the bias), whereas the True Loss drops until that same value. Therefore, the bigger the training set is, the better the algorithm performs on a given data set. However, it performs worse on the training set itself.

To review why the learning curve is like that we just have to remember that it is easy to fit the linear regression for a few points (lower training loss for small training sets), but it is a worse generalization on new data (higher true loss for small training sets). For more data points, the model will not be able to fit the training data so accurately, so the error will increase. However, the true loss will decrease and the model will give better predictions on new data points.

We also have plotted the variance obtained from the statistical treatment. In the right plot one can observe the variance of the error, we obtain the expected behaviour. We can observe that for lower values of N the variance is quite high, whereas for higher values of N this variance rapidly decreases and saturates. This can be easily explain as a lower N means a smaller amount of initial data. As the algorithm now works with a small amount of data, the results will be highly influenced by the selected inputs. As we are repeating the experiment different times, each time the result

will be different as the initial data will be completely random. When the value of N increases, this dependency to the selected data decreases as the algorithm works with a bigger amount of data. Thus, no matter how many times we repeat the simulation, all results will be more similar and will present less differences as the predictions will most likely tend to be similar.

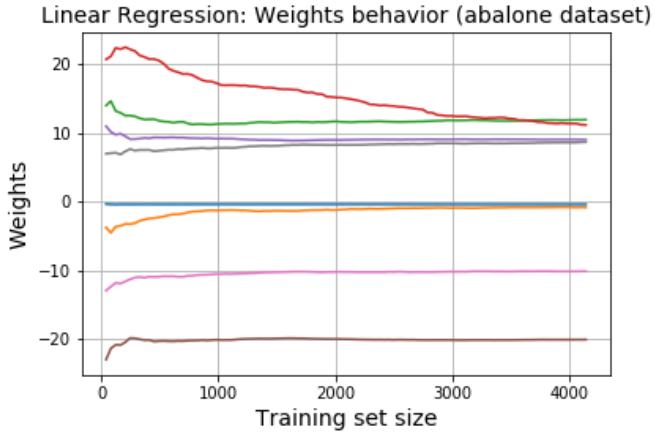
- Plot the cpu-time as a function of N.



The plot on the left corresponds to the CPU time required to perform the calculations given a training set size. This is expected to grow as function of N, since the computational cost is larger for larger amounts of data, and thus, the time required to perform the simulation increases as well. We do not understand the sudden growth at the end of the figure, although we need to consider it is still a fairly small time scale. We can conclude that CPU time is quite low for any given training set size, and usually it increases linearly as a function of N.

The plot on the right corresponds to the variance of the CPU time. As one can easily observe, all variance values are quite low, indicating there is no much difference between the different simulations. This makes sense, as all simulations use the same amount of input data, thus having the same computational cost. The time required to do the corresponding operations is the same, ending up in similar times.

- Explore how the learned weights change as a function of N. Can you find an interpretation for the learned weights?



Here we can see how the plots evolve for an increasing training data set size N. In this first Abalone data set, as there are 8 features, the linear regression algorithm obtains 8 corresponding weight coefficients. For an increasing number N (Training set size), we can observe how the value of these weights tends to saturate to a constant. The weights that stabilize in zero translate into the feature not having any relevance in the model, while the other ones stabilize to the value of their relevance.

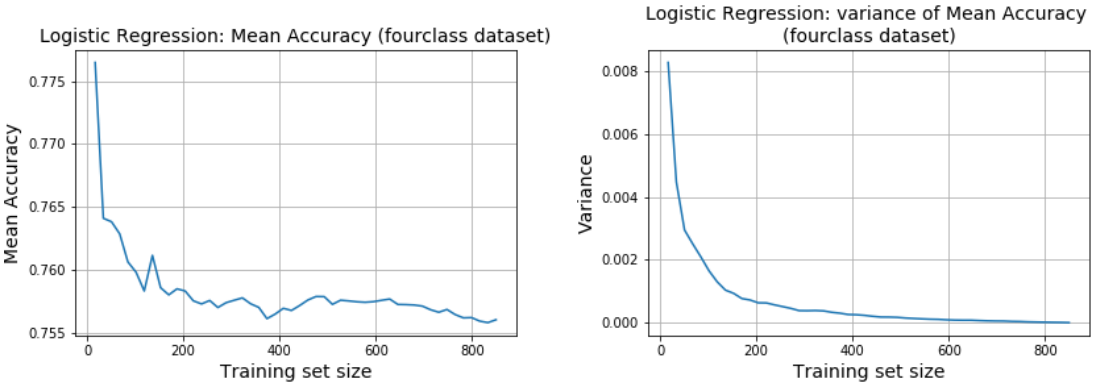
We can observe how for low values of N, the final weight coefficients tend to change but when

the N value increases, the corresponding coefficients rapidly stabilize in a constant function. When trying to understand the behaviour of these coefficients as function of N, one can easily determine that for a big amount of data, the algorithm predictions will tend to be more accurate, thus the value of the weight coefficient will be quite constant and will not not be likely to change. However, when this training set size is small, as the random initial data points will have a big influence on the algorithm performance, the predictions will not be accurate enough and the weight coefficients will tend to present more disperse values. Therefore, one can conclude that the bigger the training data set is, the better the algorithm will perform.

2 CLASSIFICATION TASK

For this second task, we picked the Fourclass classification dataset (# of classes: 2, # of data: 862 and # of features: 2). All the programming has been done with python, using the Jupyter notebook environment. The logistic regression algorithm has been imported from the scikit-learn library. In this case we also applied the same statistical treatment and the shuffle of the samples.

- Plot the approximation error (square loss) on the training set as a function of the number of samples N (i.e. data points in the training set).

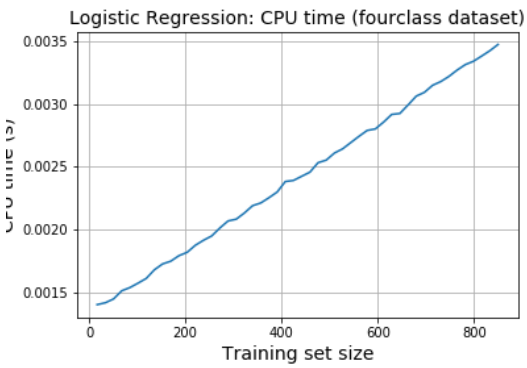


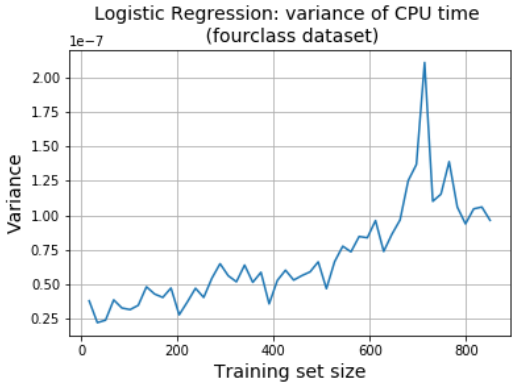
In this case our training loss is the mean accuracy ($\frac{\#correct\ predictions}{\#data\ points}$), and its provided by the function $sklearn.linear\_model.LogisticRegression.score()$. This definition is the opposite from the definition we learned in the lectures ($L_{S,lectures} = \frac{1}{m}\|\{i \in [m] : h(x_i) \neq y_i\}\|$, where m is the size of the training set, or $\frac{\#failed\ predictions}{\#data\ points}$). Therefore, its behaviour is also expected to be the opposite. If we look at the figure we see we obtained the expected results, as it behaves as $1 - L_{S,lectures}$.

For the figure on the right we obtain the same behaviour as in the previous section. The explanation from before holds for this case as well.

- Plot the cpu-time as a function of N..

The analysis for these figures is the same as the one from the previous section (linear regression). We get satisfactory results as well. In this second case, one can easily observe how the CPU time clearly increases linearly as a function of N (training set size).Therefore, the larger the data set is, the longer it will take to compute the Logistic regression. When talking about the variance, again, the obtained values are really low, thus indicating the all repetitions tend to obtain similar values.





- Explore how the learned weights change as a function of N. Can you find an interpretation for the learned weights?

In this second data set we only have two features, and therefore, the corresponding algorithm will present two weight coefficients. When monitoring the change that these two coefficients present, one can easily observe that we obtain the same behaviour as before: The weights vary for small training set sizes but stabilize in a constant value for larger training set sizes.