Report: Assignment 1:
Big Data Analytics Programming

Andreas Hinderyckx

December 2021

1 Learning Curves

All the learning curves discussed in this section are plotted with the default parameters (PERIOD $= 10^5, \eta = 10^{-9}, n_{min} = 200, \tau = 0.05$ and $\delta = 10^{-7}$). The weights of the Perceptron are initialised to zeroes (see section 2).

The learning curves for the Perceptron (PC) and Very Fast Decision Tree (VFDT) run on the `clean` datasets are shown in figures 1 and 2 respectively. The accuracy of the PC shows much more volatility whereas the VFDT's accuracy follows a more stable course which, however, only reaches about 77%. After around 100.000.000 examples have been trained with, we can clearly see the change of model used to generate the data in both graphs: both the PC's and VFDT's accuracy take a steep drop. The PC quickly recovers back up to its original state, whereas the VFDT doesn't reach the same accuracy it had before: this could be due to the VFDT being overfit to the previously used model to generate the data, whereas the PC's weights can be changed entirely as needed.

Tested on the `noisy` data set, we acquire the learning curves shown in figures 3 and 4. Now, the volatility of the PC's accuracy becomes clearly visible: it struggles to learn the model due to the added noise. The VFDT's graph is highly similar to its graph for the `clean` data, with a reduction of achieved accuracy of 10% and some extra volatility.

2 Experiments

To study the effect of different parameters present in the programs, we pose the following questions:

1. What's the effect of non-zero initialisation of the weights of the PC?
2. What's the effect of varying η (eta: learning rate) in the PC implementation?
3. What's the effect of varying δ (delta) in the VFDT implementation?
4. What's the effect of varying τ (tau) in the VFDT implementation?
5. What's the effect of varying n_{min} (how often split function is recalculated) in the VFDT implementation?

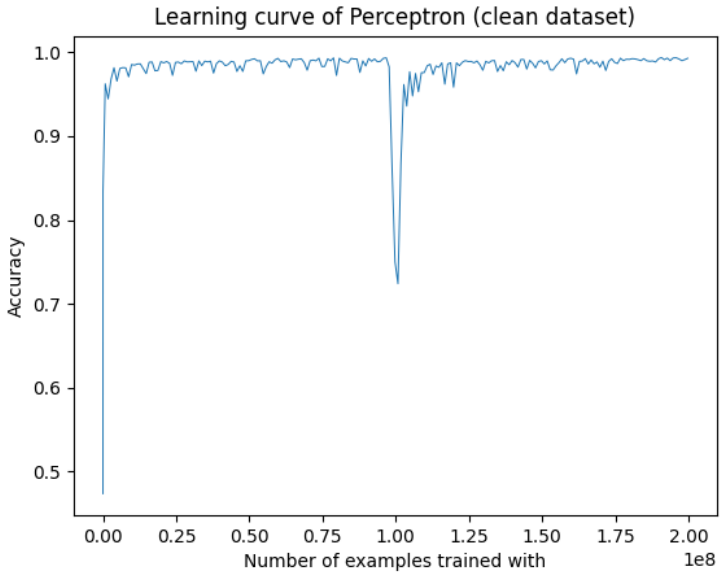

Figure 1: Learning curve of PC on the `clean` dataset

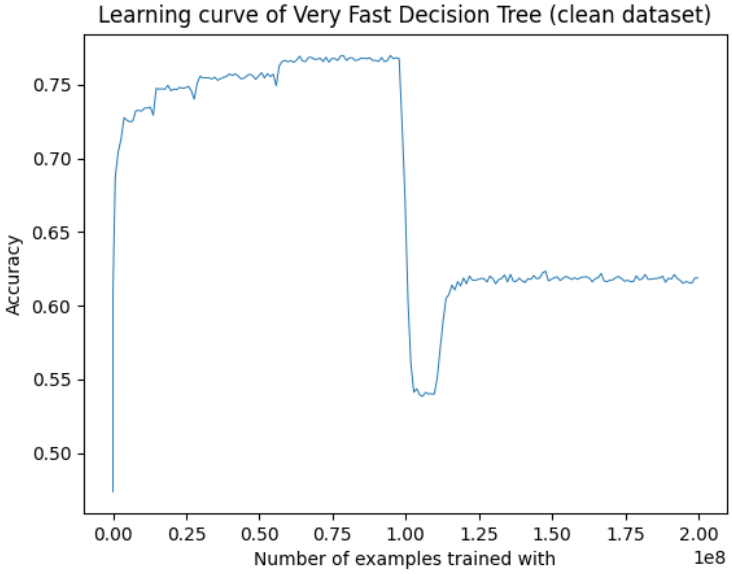

Figure 2: Learning curve of VFDT on the `clean` dataset

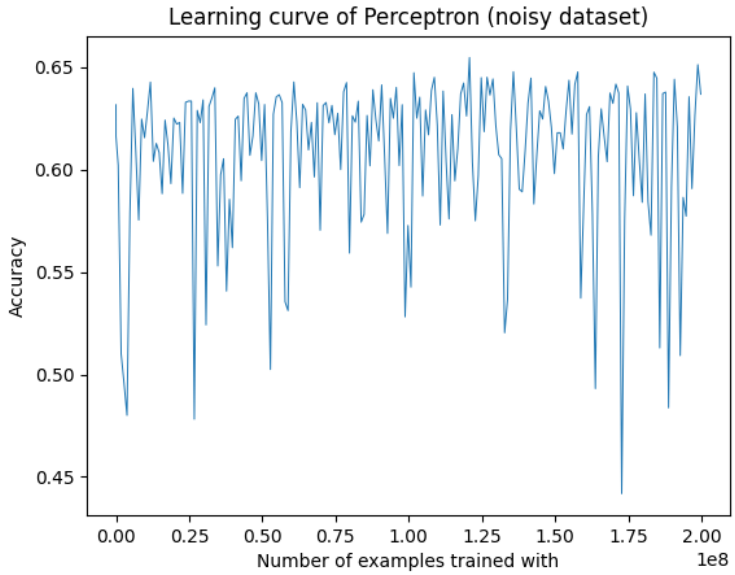

Figure 3: Learning curve of PC on the `noisy` dataset

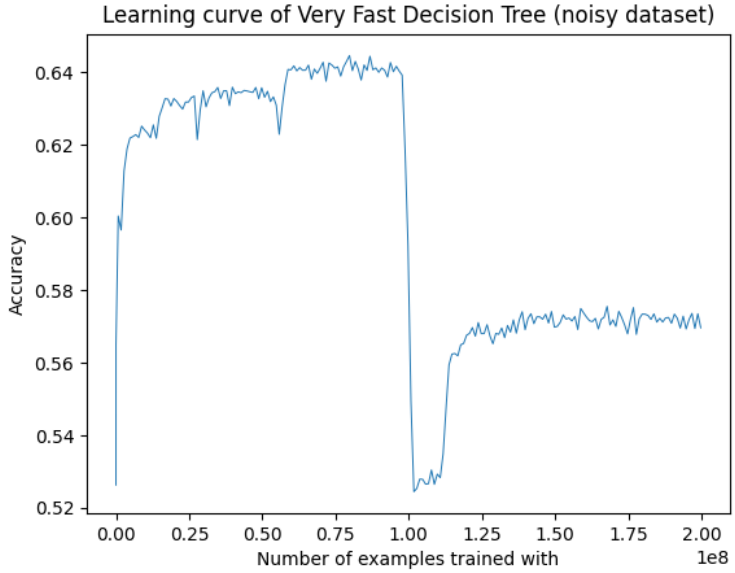

Figure 4: Learning curve of VFDT on the `noisy` dataset

We set up experiments in which we vary the parameters specified in the mentioned questions above and become the following results.

For different values of η to take effect on the model, we must initialise the PC's weights to non-zero values[1]. We initialise the weights with normally distributed values with $\mu = 0$ and $\sigma = 0.001$. The learning curves for the PC with varying values for η on the `clean` data set are shown in figure 5. We notice that η must be sufficiently large in order for the model to gain accuracy: $\eta = 10^{-9}$ is sufficient to increase accuracy, albeit at a very slow rate due to the small updates. Taking $\eta \geq 10^{-7}$ yields much quicker accuracy gains at the cost of more volatility, and doesn't quite reach the same final accuracy as with $\eta = 10^{-9}$, due to the learning rate being too large to make precise adjustments to the weights to try to increase the accuracy further beyond $\sim 97\%$.

For studying the parameters of the VFDT, we use the `clean` dataset. In figure 6 we can see that varying δ affects the speed at which accuracy is gained: in the first part of the learning curve, larger values for δ lead to faster gains in accuracy. This is what we expect, as increasing delta implies a less strict Hoeffding bound, which will cause the VFDT to make splits quicklier (see fig. 7). After the concept drift, $\delta = 10^{-7}$ increases the accuracy the quickliest: this value possibly hits the right middleground between splitting as quick as possible (larger δ) and not overfitting too much too quickly on the data (smaller δ); cfr. figure 8.

From figure 9 we can deduce that smaller values of τ increase the learning efficiency: this makes sense as the algorithm spends less time on deciding between minor differences in split evaluations and will be able to split sooner on these cases. We also note that for larger values of τ, the experiments didn't finish due to excessive memory usage, as the algorithm will then split more often and thus consumes more memory as τ increases.

Finally, increasing n_{min} implies the VFDT will re-compute the split evaluation function less frequently, which decreases computation times (as calculating the split evaluation is the most costly part of the algorithm), but should also decrease the rate at which is learned, as eligible splits are only discovered after a delay of at most n_{min} examples. The latter is confirmed by figure 10, of which an enlarged view can be seen in figure 11.

[1] `https://datascience.stackexchange.com/a/27305`

3 Special points about implementation

In the implementation of the Perceptron I added a variable `randomiseWeights` to toggle between the weights being either initialised to all zeroes or to initialise the weights with randomly normally distributed values with $\mu = 0$ and $\sigma = 0.001$, as mentioned above. This was set to true to verify the effect of changing the value of η. For the default provided value of the learning rate ($\eta = 10^{-7}$), however, the implementation with weights initialised to all zeroes obtained the best results. Therefore, the weights will by default be initialised to all zeroes (i.e. `randomiseWeights = false`).

In the method `addChildren` in `VfdtNode.java`, the `nijk` counts of the current node that executes `addChildren` are set to `null` after the children are added. These counts are no longer needed from this point on and the memory occupied by them can therefore be made available for garbage collection.

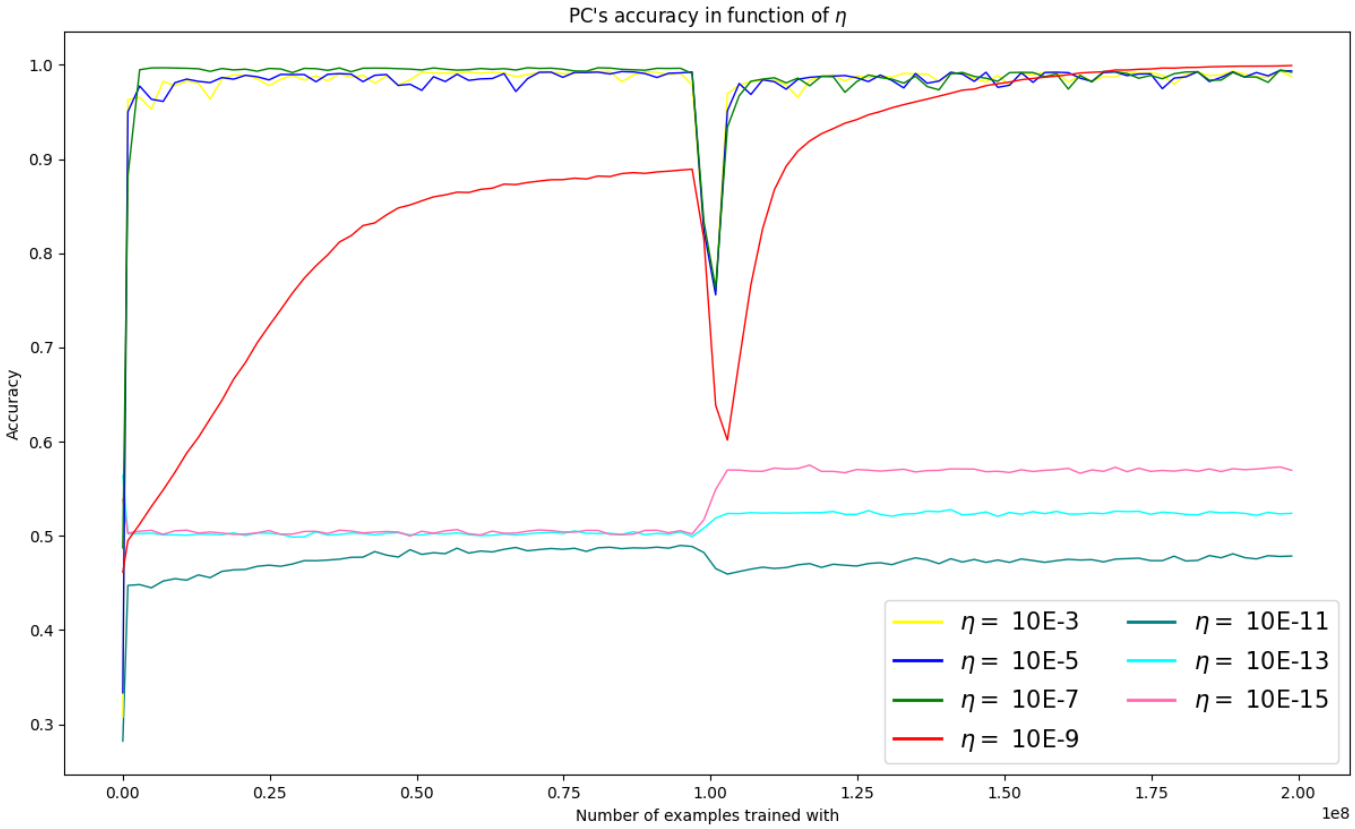

Figure 5: Effect of varying the learning rate η (with weights initialised to non-zero values)

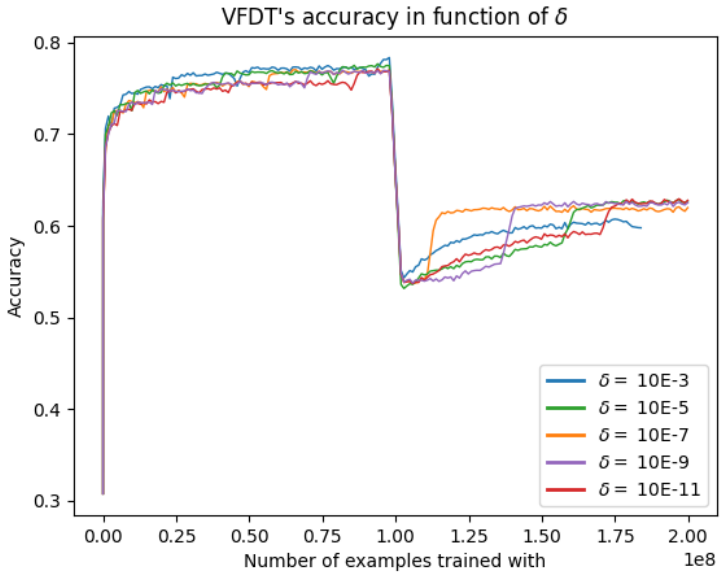

Figure 6: Effect of varying δ

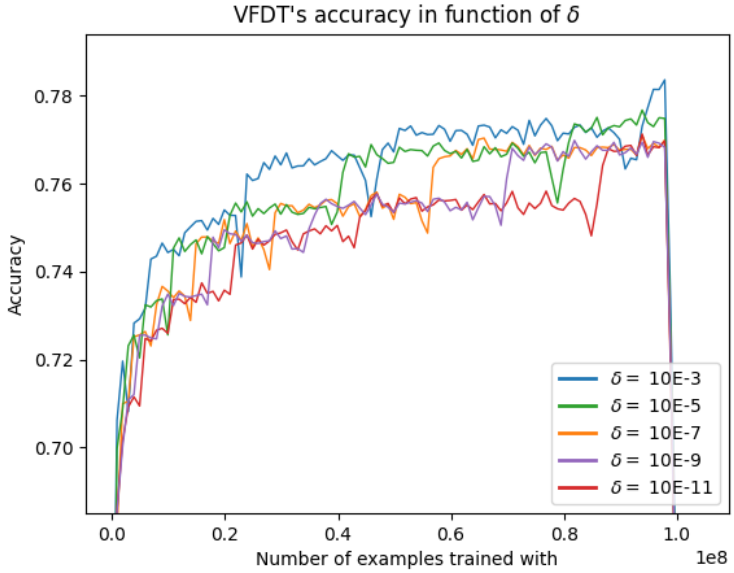

Figure 7: Enlarged view of figure 6 (before concept drift)

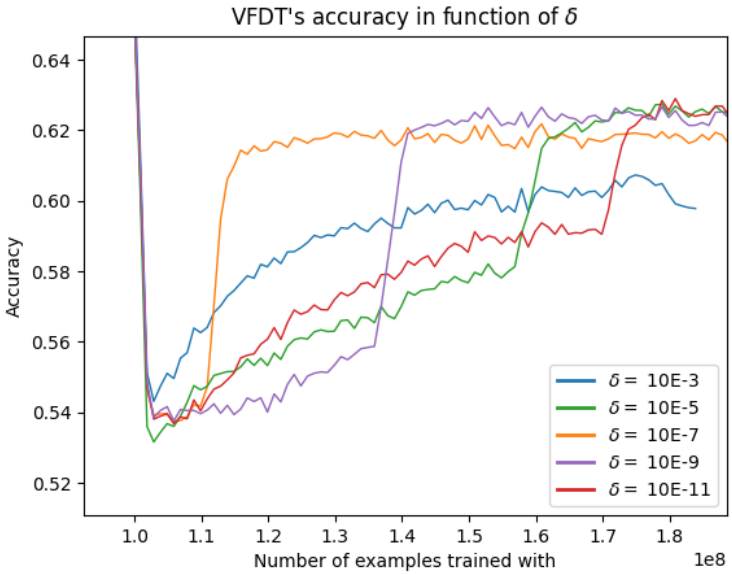

Figure 8: Enlarged view of figure 6 (after concept drift)

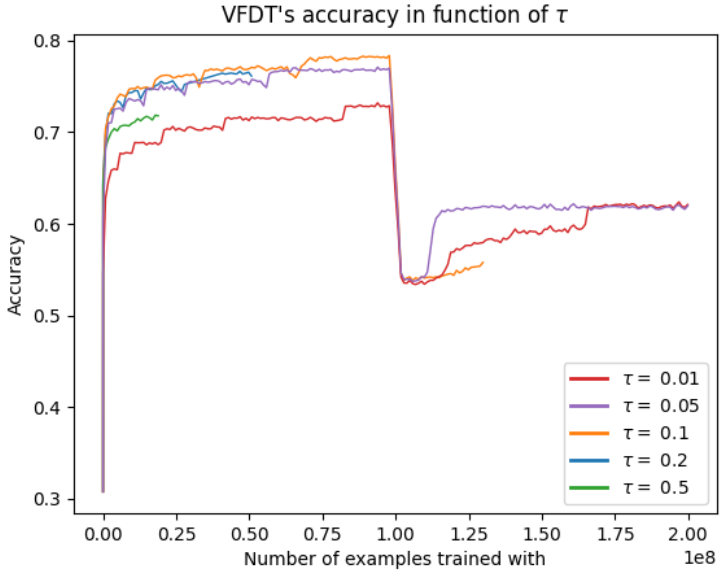

Figure 9: Effect of varying τ

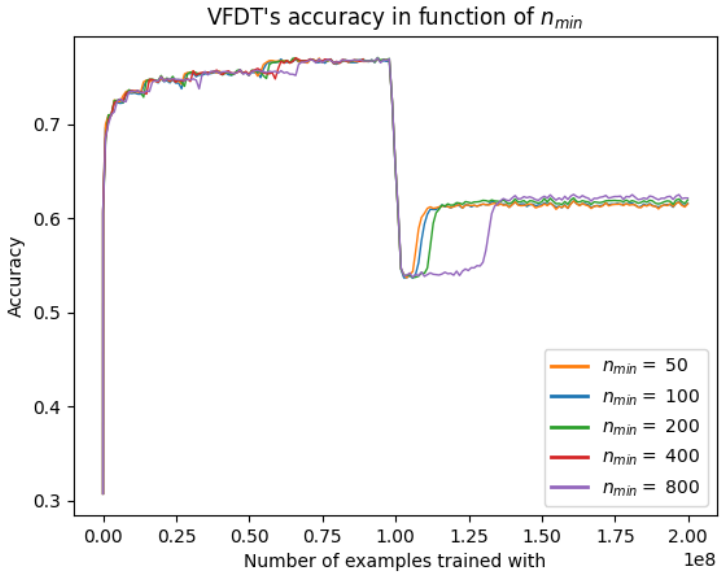

Figure 10: Effect of varying n_{min}

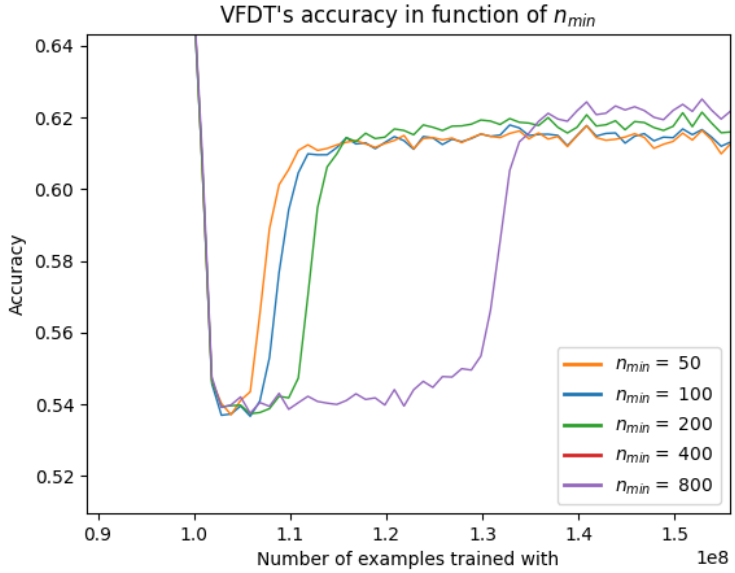

Figure 11: Enlarged view of figure 10