HW6

ALESSANDRO MANZOTTI

1. Problem 1, Perceptron

Code scripts:

Kernel online: `kernel_perceptron_online.py`
Kernel batch: `kernel_perceptron_batch.py`
Linear online: `linear_perceptron_online.py`
Linear batch: `linear_perceptron_batch.py`
Kernel test sigma: `kernel_perceptron_online.py`
Linear test run number: `kernel_perceptron_online.py`
Kernel test run number: `kernel_perceptron_run_batch.py`
Kernel Labels: `label_batch_kernel_gaussian.txt`
Linear Labels: `label_batch_linear.txt`

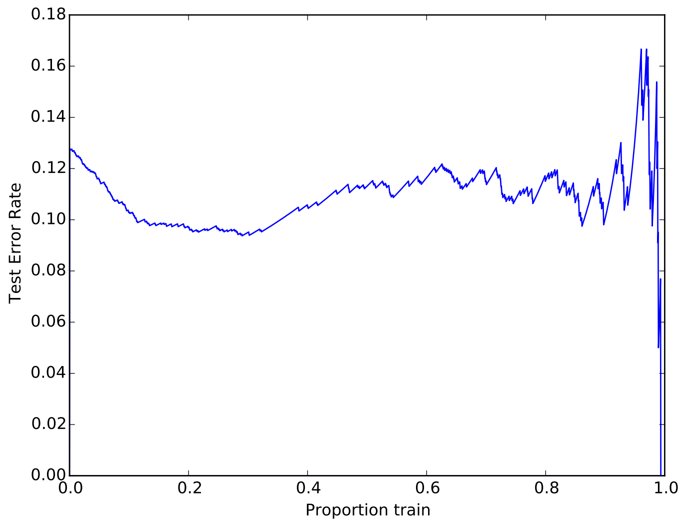

Figure 1. Linear perceptron. Number of mistakes (rate) as a function of the example seeen. The percetron is initialized so that $\mathbf{w} = x[0]$ where $x[0]$ is the first training point. This was not done holding a test set from the data. I trained the perceptron online on the data recording the errors made. I did not run different initialized run so this curves are noisy and in particular the tail where we are just looking at the number of errors in the last few points of the data.

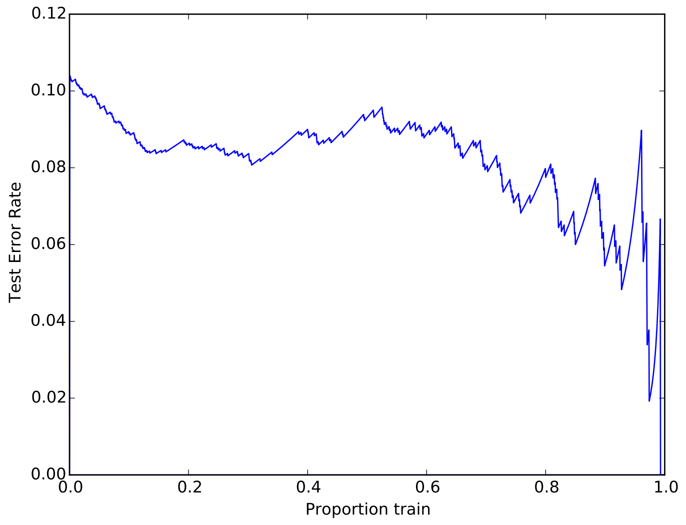

FIGURE 2. RBF Gaussian with $\sigma = 1$. Number of mistakes (rate) as a function of the example seeen. The percetron is initialized so that $\mathbf{w} = x[0]$ where $x[0]$ is the first training point. This was not done holding a test set from the data. I trained the perceptron online on the data recording the errors made. I did not run different initialized run so this curves are noisy and in particular the tail where we are just looking at the number of errors in the last few points of the data.

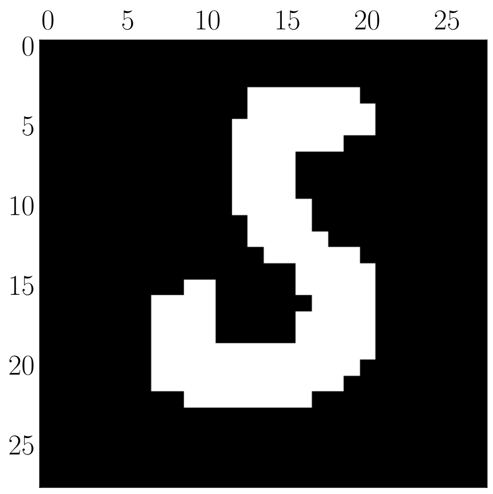

FIGURE 3. A representative of the few examples in tha data set hard to learn. In this case a 5 was very often mistaken for a 3 in several run of both the linear and the RBF Gaussian kernel batch mode algorithm.

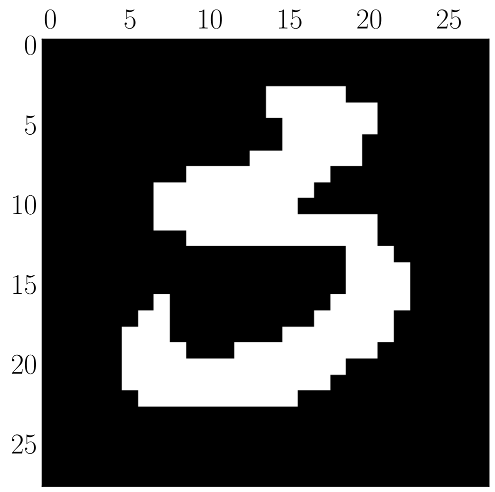

FIGURE 4. A representative of the few examples in tha data set hard to learn. In this case a 3 was very often mistaken for a 5 in several run of both the linear and the RBF Gaussian kernel batch mode algorithm.

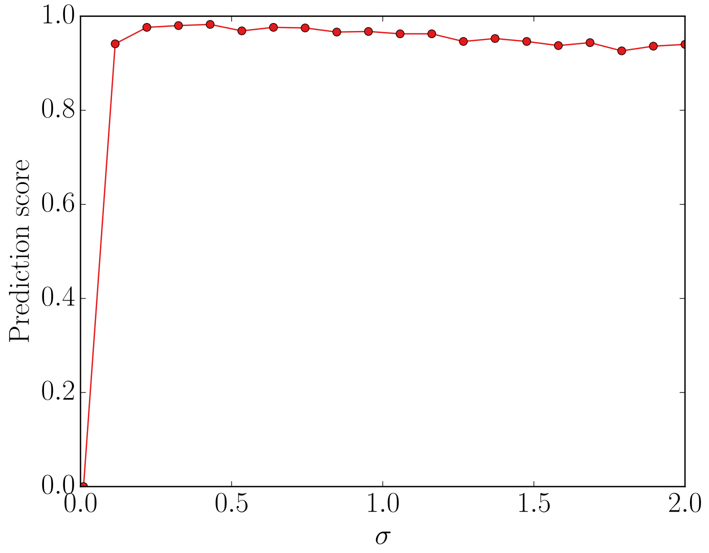

FIGURE 5. First of 3 different scales example of my training of the sigma parameter in the RBF kernel. I held 40% of the dataset off from the training one and then tested the accuracy on that. I did not perform more advanced cross validation trying to held different part of the dataset etc. The performance seems to be quite flat in σ for a significant range. Slightly decreasing from 0.4 to 2. This was tested with 200 runs of the batch version of the algorithm.

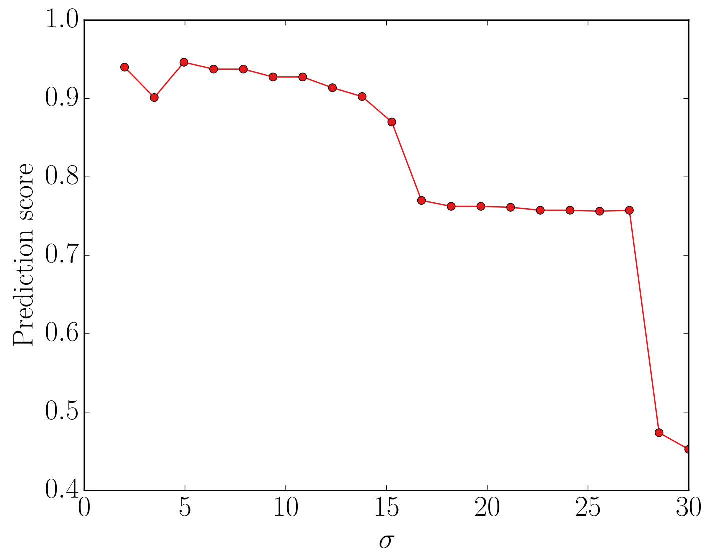

FIGURE 6. Training of the sigma parameter in the RBF kernel. I held 40% of the dataset off from the training one and then tested the accuracy on that. This is an example of the secrease in performance once σ became too big (> 10).This was tested with 200 runs of the batch version of the algorithm.

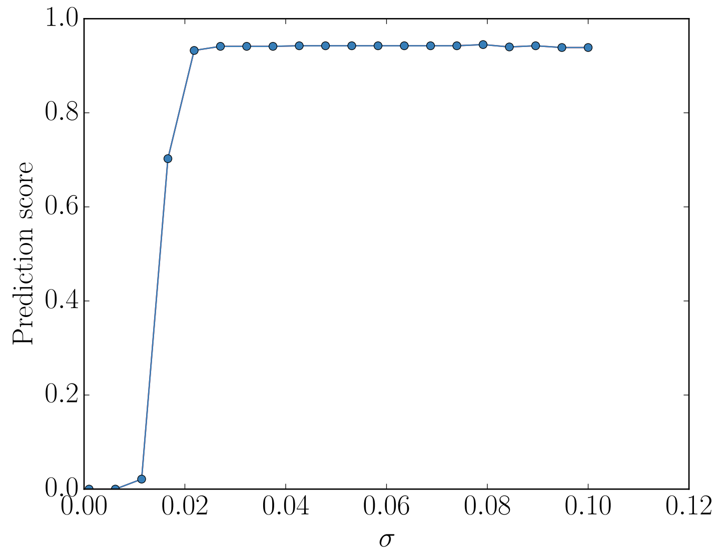

FIGURE 7. Training of the sigma parameter in the RBF kernel. I held 40% of the dataset off from the training one and then tested the accuracy on that. A zoom in the lowest end of σ. Smaller than 0.02 seems to be too small. This was tested with 200 runs of the batch version of the algorithm.

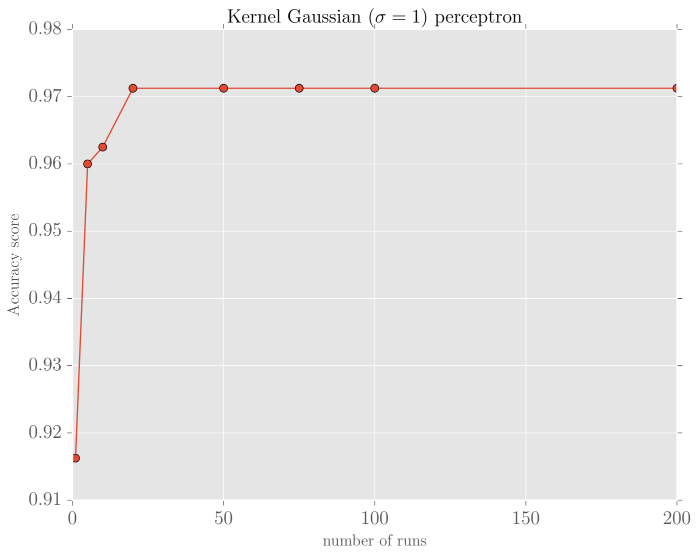

Figure 8. Gaussian. Quick test of the performances as a function of number of times the algorithm is run in batch mode.

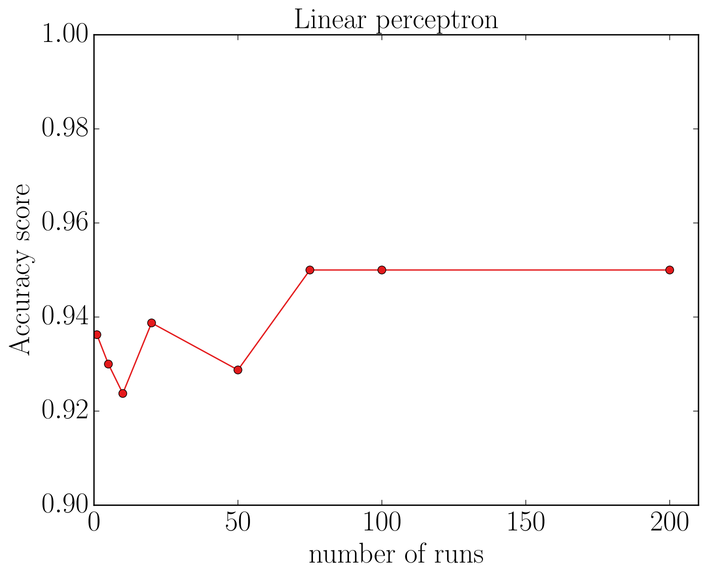

Figure 9. Linear. Quick test of the performances as a function of number of times the algorithm is run in batch mode.

2. Problem 2

Code: All is done in `gaussian_processes.py`

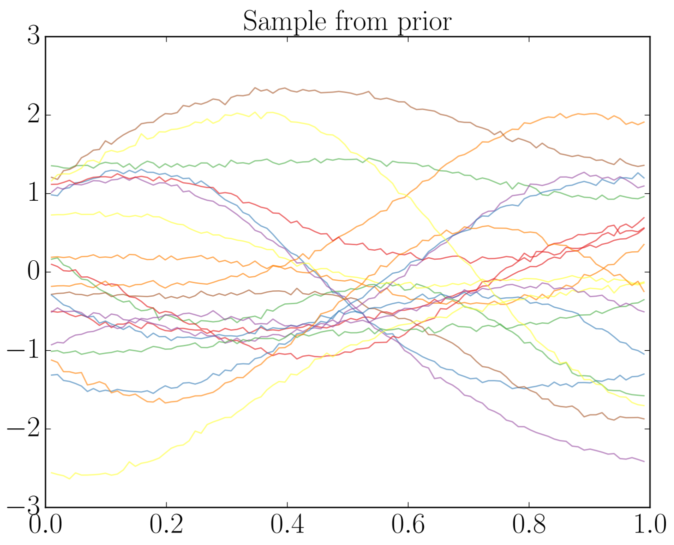

Figure 10. 2a. Example of 20 fucntions draw from the $G(\mu, k)$ prior. This was done using the iterative procedure suggested in the HW text. I needed to add a small $1e5$ regularization factor to K and avoid inverting the K matrix directly to avoid numerical instabilities while using a lot of (very close) points.

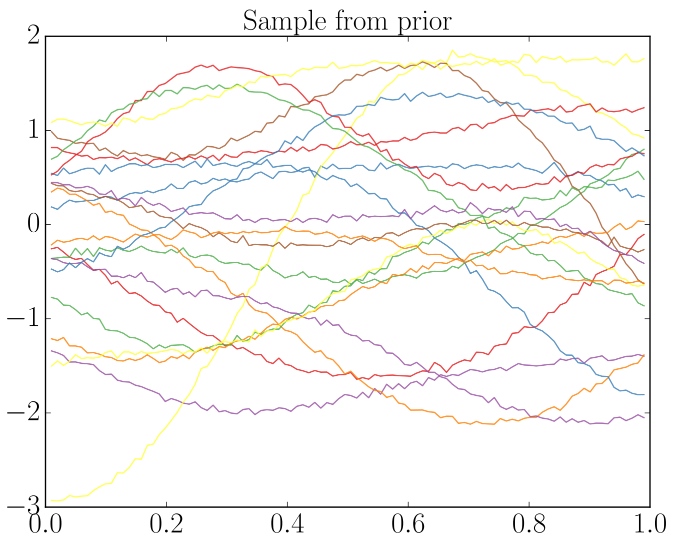

Figure 11. 2a. Example of 20 fucntions draw from the $G(\mu, k)$ prior. This was done sampling from a multivariate gaussian with mean zero and covariance k using native `numpy` functions

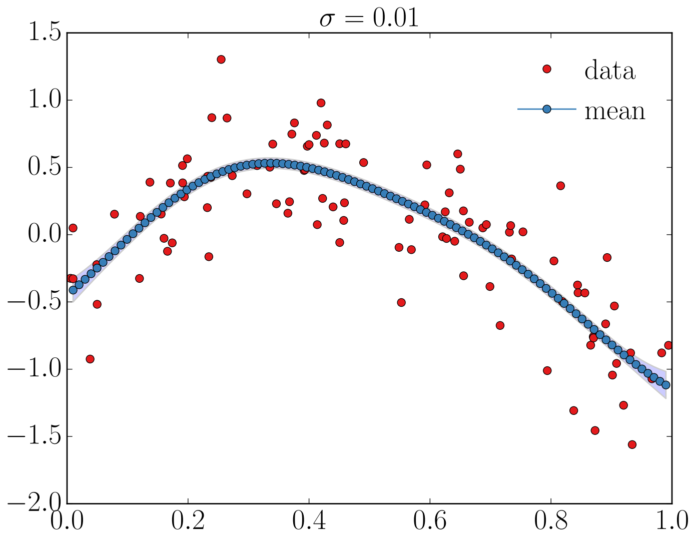

FIGURE 12. 2b. Posterior mean and 2 standard deviation bound for different choices of the parameter σ. The data are also plotted.

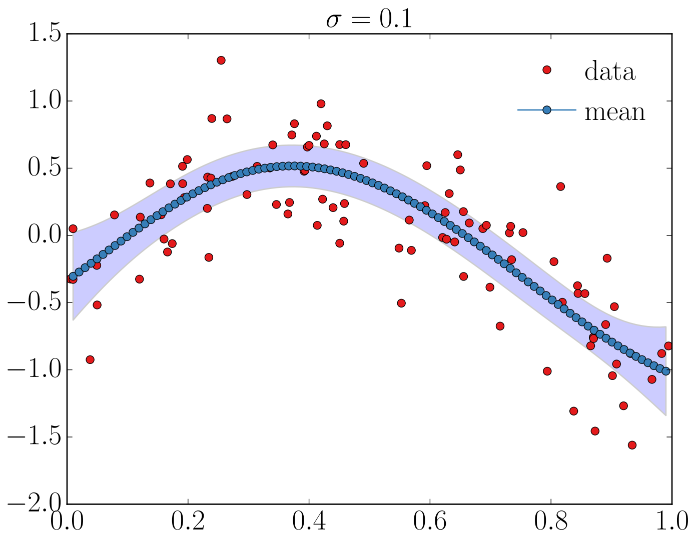

FIGURE 13. 2b. Posterior mean and 2 standard deviation bound for different choices of the parameter σ. The data are also plotted.

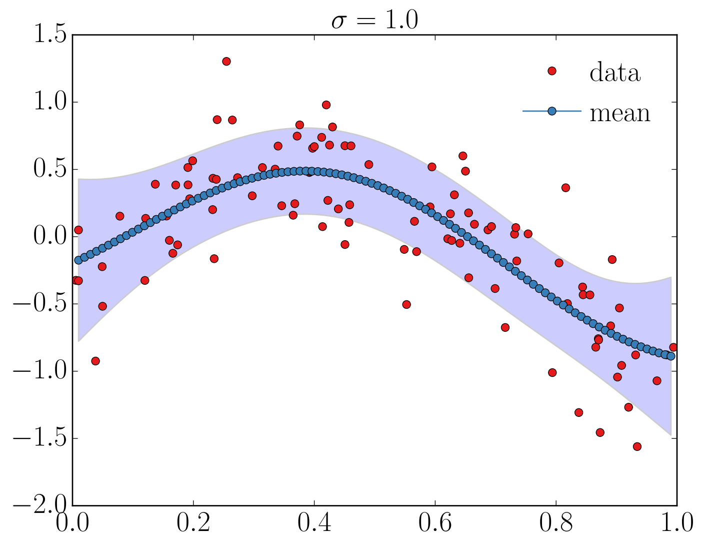

FIGURE 14. 2b. Posterior mean and 2 standard deviation bound for different choices of the parameter σ. The data are also plotted.

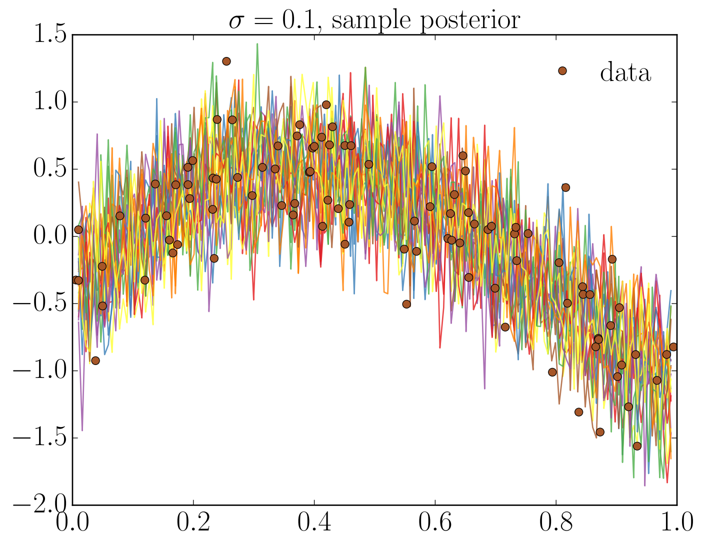

FIGURE 15. 2c. 20 functions sampled from the posterior. This example was for $\sigma = 0.1$

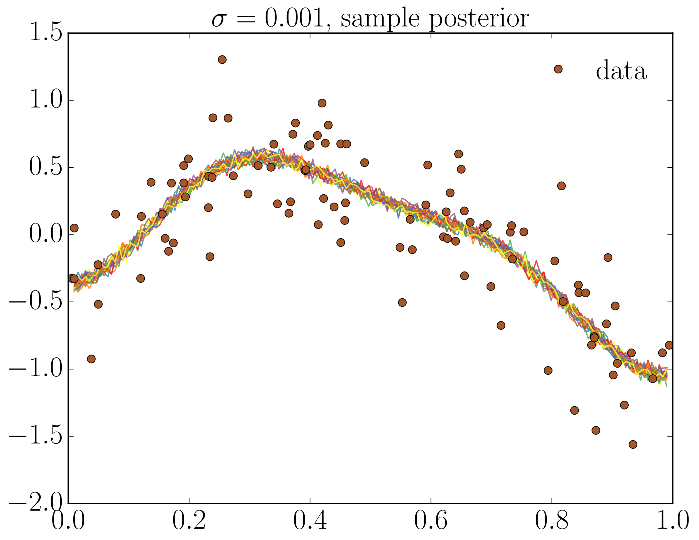

FIGURE 16. 2c. 20 functions sampled from the posterior. This example was for $\sigma = 0.001$. Assuming a small noise in the data force the sampled curves to stay close to the mean. Not sure if there is an optimal value for σ. By eye the sigma seems to be close to $0.1 - 1$. However given the value of τ in the kernel the functions are pretty rapidly oscillating and basically picking up noise if $\sigma > 0.01$