CS5783: Machine Learning
Assignment 4

Prof. Christopher Crick

1 Gaussian process regression

Use the crash test dataset from assignment 3 again. In order to make numerical instability less of an issue, scale the x and t values of your dataset to values between 0 and 1, i.e. normalize each value by dividing it by the maximum value in that dimension.

We will be generating Gaussian processes using two different kernels:

- Squared exponential: $k(x, x') = \exp\{-\frac{(x-x')^2}{2\sigma^2}\}$
- Exponential: $k(x, x') = \exp\{-\frac{|x-x'|}{\sigma}\}$

For each of these kernel families, construct your Gram matrix K and add diagonal noise to form C. In the last assignment, we estimated the β precision parameter for the noise as 0.0025 (because we eyeballed the standard deviation $\sigma = 20$, and $\beta = \frac{1}{\sigma^2}$). If you scale σ by the same magnitude as you scaled all of the t values, you can compute the appropriate β for C.

You can now use C, t and the kernel function distances between x^* and each x to predict y^* values at x^*. First, figure out an appropriate order of magnitude for the σ parameter (this is the σ parameter for the kernels, not the standard deviation of the noise, as in the previous paragraph!). Look at the output of your Gaussian process (perhaps by plotting using evenly-spaced x values) and look for values that seem to be relatively well-behaved (poorly chosen ones might look nothing like the data, or might crash your evaluator).

Once you have found a reasonable value of σ, perform five-fold cross-validation on 100 values of σ of the same order of magnitude as your rough calculation found, computing average MSE and determining a best-fit hyperparameter value.

For each of the kernel functions, plot the training data and the output of the Gaussian process with the best-fit hyperparameter (by plotting 100 evenly spaced x values and their corresponding GP outputs).

2 K-means clustering

Use the MNIST test set rather than the training set, simply because 10000 examples will be a little easier to work with then 60000, and we're doing unsupervised learning anyhow. We wish to minimize the K-means objective function

$J(z, \mu) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} ||x_n - \mu_k||^2$,

where z_{nk} is 1 if example n is in cluster k and 0 otherwise.

Implement a K-means algorithm function that takes a value for the number of clusters to be found (K), a set of training examples and a K-dimensional vector μ_k^0 that serves as an initial mean vector. This function should return the n-dimensional cluster assignment (presumably as an $n \times k$ one-hot matrix, since that is most convenient), as well as the converged μ_k vector. At each iteration, print a dot as a progress indicator. Once J has converged, print out its value, as well as the number of iterations it took.

Run your algorithm with K=10 (the true number of clusters) on the following intializations μ_k^0:

1. Ten data points chosen uniformly at random
2. Ten data points found using the K-means++ assignment algorithm
3. A data point drawn from each labeled class (found by looking at the test set labels – and yes, this is cheating)

Visualize the 28×28-pixel images corresponding to each cluster mean found by your algorithm, for each of these initializations.

Cluster the data using K=3, initialized using K-means++. Plot the cluster mean images and a few randomly chosen representatives from the data for each class.

3 Hidden Markov Models

Construct a state machine that mimics the "occasionally dishonest casino" used as an example in lecture. This machine has two states, "Loaded" and "Fair". When in the "Fair" state, it outputs a value between 1 and 6, chosen uniformly at random. When in the "Loaded" state, it also outputs a value between 1 and 6, but this time the odds of emitting 1-5 are $\frac{1}{10}$ each, while the odds of emitting a 6 are $\frac{5}{10}$. This can be represented in a table:

$p(x_t|z_t) =$

x_t	$z_t = F$	$z_t = L$
1	0.16667	0.1
2	0.16667	0.1
3	0.16666	0.1
4	0.16667	0.1
5	0.16667	0.1
6	0.16666	0.5

Furthermore, the transition matrix A between hidden variables is the following:

$p(z_t|z_{t-1}) =$

z_t	$z_{t-1} = F$	$z_{t-1} = L$
F	0.95	0.10
L	0.05	0.90

The process should start in the "Fair" state. Capture the output of this process for 1000 steps in a vector x, and record the true state of the hidden variable z for each step, as well.

Use the forward-backward algorithm on your vector of outputs, as well as the true probabilities contained in the transition and emission matrices, to construct the MAP estimate of the state distribution at each time point. Produce two plots of the estimate of $\hat{z}$ of the probability of a loaded die at time t, compared to the actual state which you saved when you generated the process in the first place. In other words, one line on the graph will be a probability somewhere between 0 and 1, while the other will be a step function that transitions between exactly 0 and exactly 1. One

of your plots should be your estimate after performing your forward pass but before computing the backward pass, and the other should be your estimate of $\hat{z}$ when the entire inference process is complete.

4 Turning in

Your code must run on Prof. Crick's Python3 interpreter. He has the numpy, matplotlib and scipy libraries installed, as well as the standard Python libraries such as random and math. You should not need any others to do this assignment, and if you use any others, he will not be able to execute it.

You must have a file named 'assn4.py', and within it, three functions named 'problem1()', 'problem2()', and 'problem3()'.

You may have any number of .py files in your submission, which your assn4.py will import as necessary. You do not have to include 'crash.txt' or 't10k-images-idx3-ubyte' with your submission, but you should assume that I will put files with those names into the working directory along with your code.

If I execute the Python commands below, I am expecting to see something like the following. Note that your program's output should be qualitatively similar, but will not likely be identical, since both you and the random number generator will make different choices than I did.

```
>>> import assn4
>>> assn4.problem1()
Squared Exponential
Best sigma = 0.11
See plot.
Exponential
Best sigma = 0.15
See plot.
>>> assn4.problem2()
Random initialization
.............................................................
64 iterations, J = 25647803615.36019
See plot.
k-means++ initialization
...................................................
53 iterations, J = 25491276527.472775
See plot.
Cheating initialization
.........................
29 iterations, J = 25409428225.92401
See plot.
.....................................
40 iterations, J = 30394791469.14684
See plot.
See plot.
```

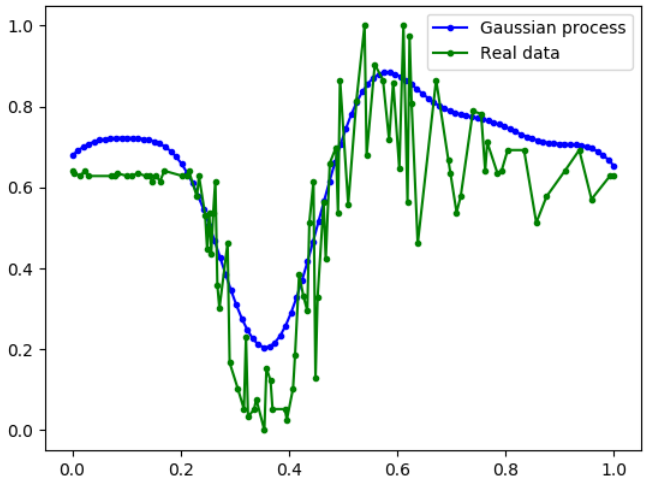

Figure 1: Output of assn4.problem1(), part 1

```
>>> assn4.problem3()
Best alpha: 0.003126
See plot.
```

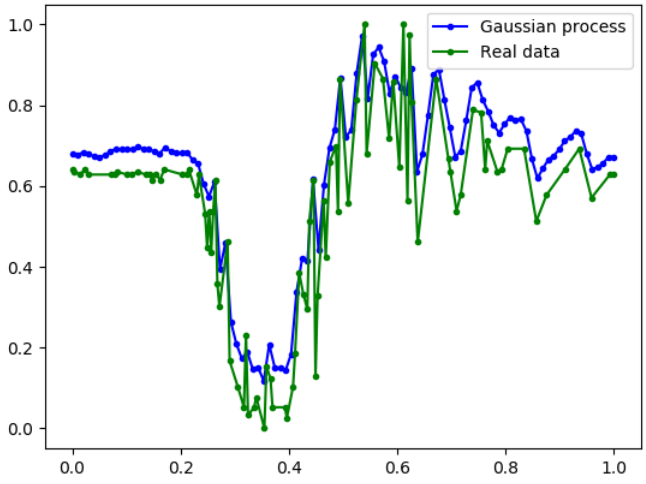

Figure 2: Output of assn4.problem1(), part 2

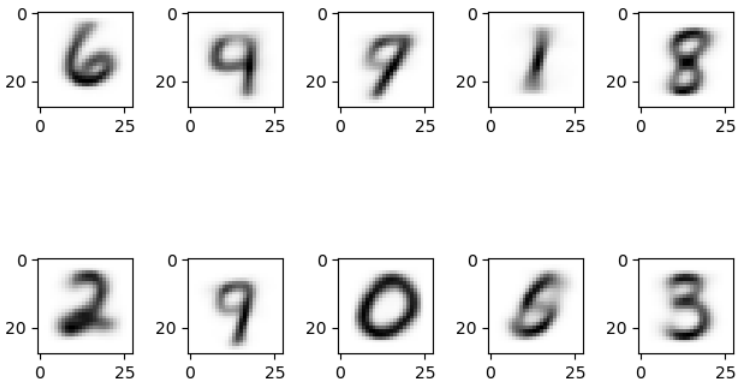

Figure 3: Output of assn4.problem2(), part 1

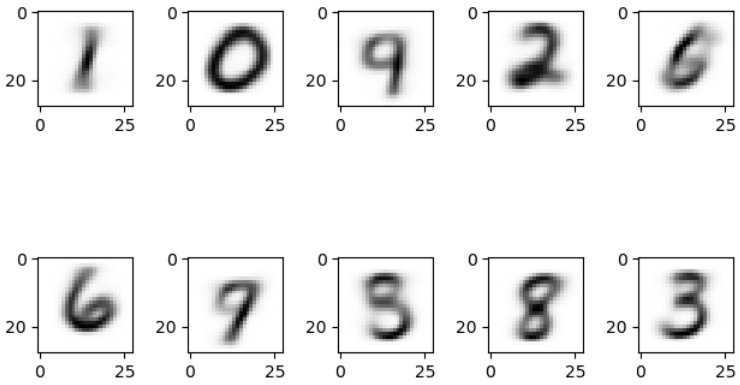

Figure 4: Output of assn4.problem2(), part 2

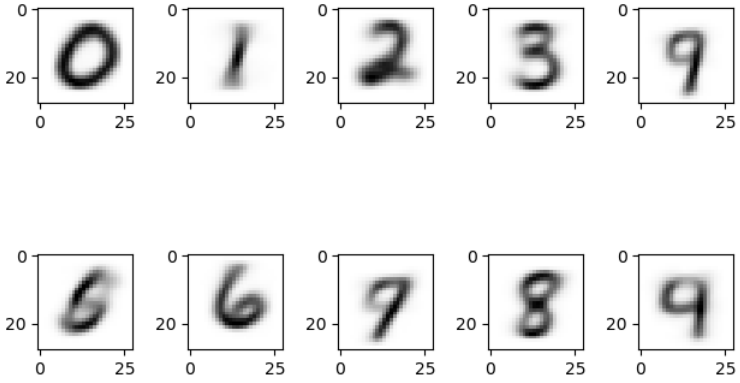

Figure 5: Output of assn4.problem2(), part 3

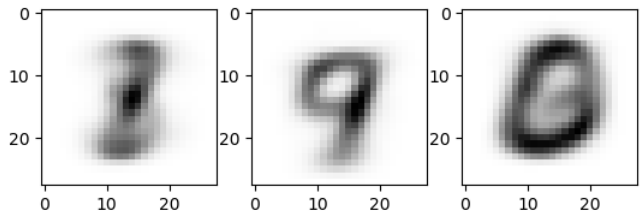

Figure 6: Output of assn4.problem2(), part 4

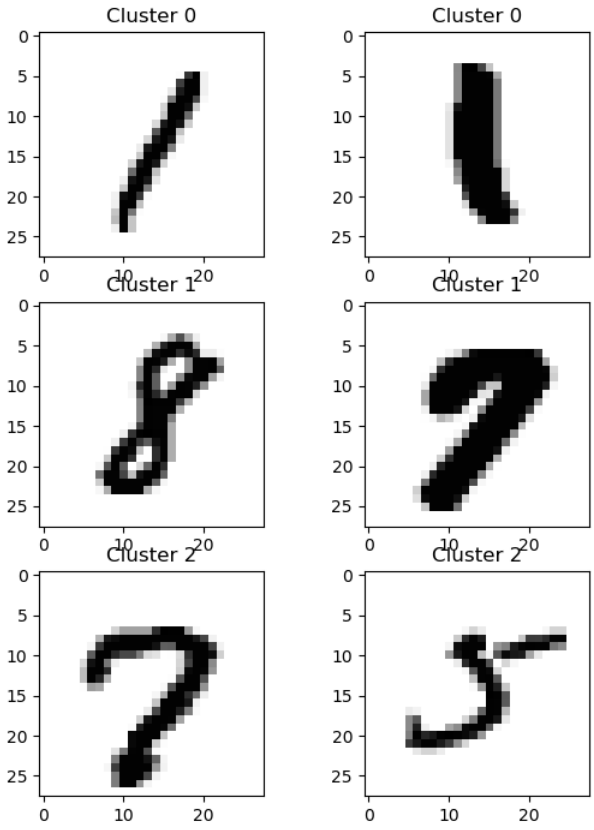

Figure 7: Output of assn4.problem2(), part 4

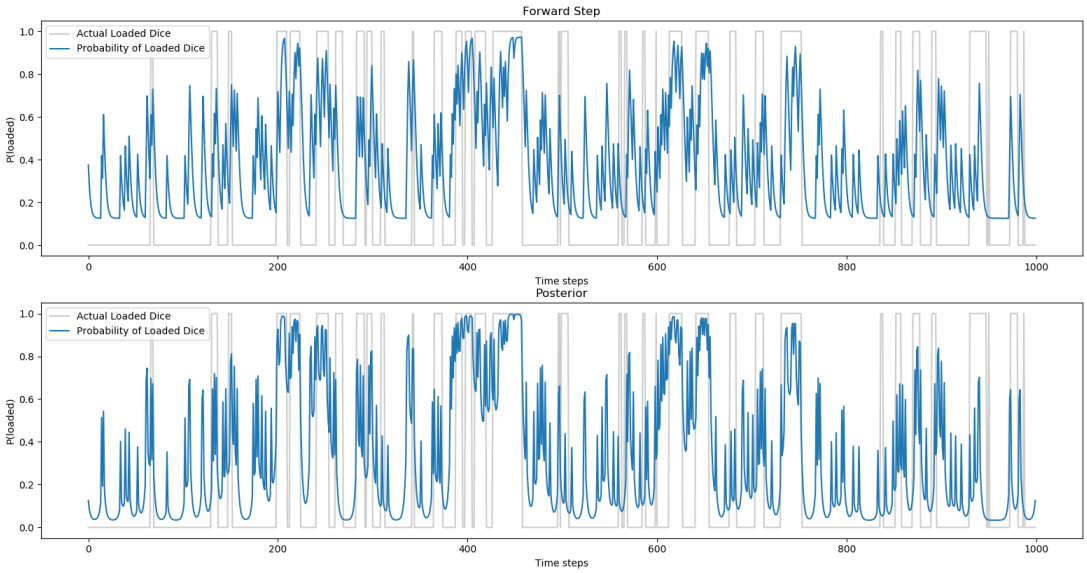

Figure 8: Output of assn4.problem3()