University of BRISTOL

Engineering Mathematics

Machine Learning

Coursework Assignment

Daniel Zyss (dz15536), Jack Morgan (jm15357), Chester Robinson (cr15169)

Sources

- SPS Notes and Black board page
- Machine Learning Notes and Github page
- Pattern Recognition and Machine Learning Book by Christopher Bishop
- https://github.com/Salma-El-Alaoui

November 23, 2017

Question 1

1)i) The likelihood is the expression of the unknown function f and the variable x given the data y. The Gaussian distribution is a distribution appropriate for independent and identically distributed random variables of unknown distribution, as stated by the central limit theorem. The central limit theorem is appropriate in circumstances when almost nothing is known about the variables except for the fact that the variables may come from some identical distribution. We can think of it as some sort of average, and as it turns out the average of i.i.d. random variables we do not know, is Gaussian. Additionally, In order to predict the function f for regression we often try to maximize the likelihood through Maximum-Likelihood (ML).

Choosing a Gaussian distribution for the likelihood is a sensible thing to do, in that case, because the Gaussian distribution is easily to maximize using log-likelihood. Moreover, as we get more data the likelihood would be centered at $f(x_i)$ such that doing Maximum Likelihood using log-likelihood would not move that center, because the logarithm function is monotonically increasing.

(ii) Choosing a spherical covariance (or isotropic covariance) means that the covariance matrix is a product of the identity matrix, in our case the covariance matrix is $\Sigma = \sigma^2 I$. It is convenient to use such a covariance matrix when dealing with i.i.d variables (especially as we set the covariance in both axes to 0) and Multivariate Gaussian distribution. Using a spherical covariance matrix means that the components of y do not covary conditionally with f and x, such that our data points are uncorrelated. Using a spherical covariance with value σ makes the condition on our data constant and equal. Moreover, If we were to have a classical covariance matrix, non-diagonally defined, we would have N(N+3)/2 parameters in total for the whole model, which is therefore quadratically increasing in size with the quantity of data. This makes it extremely computationally expensive. Using a spherical covariance matrix; however, enables the number of parameters associated with the variance in the model to be reduced to N independent parameters. By doing this, the number of parameters of the model are greatly reduced; however, unfortunately the form of the probability density is restricted and so too is its ability to capture intriguing correlations in the data.

Question 2

If the data points are not independent, then the product rule applies:

$$p(\mathbf{Y}|\mathbf{X}, f) = p(y_N|\mathbf{X}, f) = \prod_{n=1}^{N} p(y_n|y_{i+1}, ..., y_{N-1}, y_N, \mathbf{X}, f)$$

In that case, then the covariance will not equal 0, thus indicating that they covary (because they are dependent on each other).

$$\begin{bmatrix} f(x_0) \\ f(x_1) \end{bmatrix} \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} \longrightarrow \begin{bmatrix} f(x_0) \\ f(x_1) \end{bmatrix} \begin{bmatrix} \sigma & \beta \\ \beta & \sigma \end{bmatrix}$$

We can also note that the prior distribution will be different at points x_0 and x_1.

Question 3

3) In order to compute the likelihood $p(\mathbf{Y}|\mathbf{X}, \mathbf{W})$ with $\mathbf{W}$ the weight vector of the linear regression we have to assume that $\mathbf{W}$ is comprised of the weights β. We also have to assume that the variable X is i.i.d and that the noise ϵ is i.i.d as well, and that it follows a Gaussian distribution $\epsilon - N(0, \sigma^2)$. Thus with these assumptions we can write y_i as:

$$y_i = \mathbf{W}x_i + \epsilon = \beta_0 + \beta_1 x + \epsilon$$

Using the assumptions, we can now say that the variable Y is also independent, and therefore the likelihood can be estimated using the product rule, such that:

$$p(\mathbf{Y}|\mathbf{X},\mathbf{W}) = \prod_{i=1}^{N} p(y_i|x_i, w, \sigma^2) = \prod_{i=1}^{N} p(y_i|x_i, \beta_0, \beta_1, \sigma^2) = \prod_{i=1}^{N} N(y_i|\beta_0 + \beta_1 x_i, \sigma^2)$$

$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i-(\beta_0+\beta_1 x_i))^2}{2\sigma^2}}$

Question 4

If the posterior distribution is in the same functional form (same family) as the prior, then the prior and the posterior are referred to as conjugate distributions. The prior in this scenario is known as the conjugate prior. Using a conjugate distribution for the prior, means that we can establish the following relation between the prior and the likelihood:

$posterior \propto prior \times likelihood$

As the Gaussian distribution is a family distribution which is its own conjugate, which makes the Bayesian posterior method easier to compute.

Question 5

The preference determines how accurate the prediction on the prior of the vector W (the linear weight vector) is according to the W1 and W2(the weights of the linear regression) data points. The p norm (generalized version of the L1 and L2 norm) is defined as:

$$||W||_p = (\sum_{i=1}^{N} |w_{1i} - w_{2i}|^p)^{1/p}$$

Therefore the L1 norm is defined as

$$||W||_1 = \sum_{i=1}^{N} |w_{1i} - w_{2i}|$$

and the L2 norm is defined as:

$$||W||_2 = (\sum_{i=1}^{N} |w_{1i} - w_{2i}|^2)^{1/2}$$

The L2 norm penalizes the errors to a greater extent and takes into account W2, which offers a more precise fit. On the other hand, using an L1 norm optimizes the parameters W according to W1 without taking into account W2 (setting it at zero). The L1 norm therefore offers a simpler solution with a sparser weight matrix (defined sparsely as : $\tau^2 I$) which easier for further computations but is less precise and offers less penalization for wrong “guesses”.

The L1 norm will therefore have larger residuals but will also offer a solution with more zeros. It would be best here to use the L1 norm as the covariance matrix of the prior is defined sparsely as a spherical covariance matrix and therefore would be less computationally intensive even though less accurate.

From changing from L2 norm to L1 norm on the preference optimization, we reduce the intensity of the computation thanks to the sparse matrix, but we also reduce the accuracy on the prediction error.

Question 6

Using the expressions from previous answers and Bayes rule for updating our assumptions (the prior) using our data (the likelihood) we can evaluate the posterior over the parameter vector W as follows:

$$posterior = 1/Z \times (prior) \times (likelihood)$$

such that:

$$p(\mathbf{W}|\mathbf{X},\mathbf{Y}) = 1/Zp(\mathbf{Y}|\mathbf{X},\mathbf{W})p(\mathbf{W}) = 1/Zp(\mathbf{Y}|\mathbf{X},\mathbf{W})N(W_0,\tau^2 I)$$

$$1/Z \times [\prod_{i=1}^{N} 1/\sqrt{(2\pi\sigma^2)}e^{(y_i-(wx_i)^2/2\sigma^2)}] \times 1/\sqrt{(2\pi\tau^2)}e^{(-(w-w_0)^2/2\tau^2)}$$

$$1/Z \times [\prod_{i=1}^{N} 1/\sqrt{(2\pi\sigma^2)}e^{(y_i-(\beta_0+\beta_1 x_i)^2/2\sigma^2)}] \times 1/\sqrt{(2\pi\tau^2)}e^{(-(w-w_0)^2/2\tau^2)}$$

The steps for solving that posterior for **W**. is to take the natural logarithm of the expression, which because of its monotonically increasing properties, reduces the expression in complexity while conserving the value of the optimal **W**. Then, find the partial derivative of that expression with respect to W and then find the stationary point of that partial derivative, yielding the optimal **W**.

This expression, described above, has this particular form because it follows Bayes rule for Gaussian conjugate prior and likelihood (yielding a Gaussian probability that can be solved using maximization). The expression also includes the term Z, which when under the form 1/Z is the evidence, a normalizing term for the quantity of data taken in account in the model such that $Z = p(\mathbf{X},\mathbf{Y})$ for the maximization of **W**.

Question 7

In the parametric model for linear regression described above, we model the function f (that we try to regress) using a parameter vector W, we therefore base our approach in the parameter space viewpoint. We then try to find the probability distribution for that parameter and try to optimise our parameter on the data (using both the data with the likelihood, and our assumption with the prior) to then generate predictions on unseen data using the parameter W. In the non-parametric model, we base our viewpoint on the function space, and encode the complexity of our model on the data itself rather than on the parameter vector. Instead of using the model we define a prior probability distribution over the functions directly (even though the space of function is in theory infinite, in practice, with data, that space ends up being finite and thus computations can be performed).

Whereas, parametric models have a finite set of parameters, which will be determined from the data set (where the complexity of these models are bounded, regardless whether the the data set is bounded or unbounded), nonparametric models are considered to have infinitely many parameters. They, thus, can describe the data in greater detail than parametric models. In a parametric model, quantities define the model (parameters) whereas in non-parametric models, both the training points and the parameters define the model.

In nonparametric models, we aim to find a probability distribution over f(**X**) which becomes itself a random variable. The model is based on the joint distribution of the different f(**X**) that are encoded in our data with the use of kernel functions (which is basically a representation of our parameters **W** , but inherent to the data). We are really interested in the Gaussian Process, or the probability distribution $f(\mathbf{x})$ at f evaluated over the data points, that we know to be Gaussian. As the data is centered at zero (the mean of the data points are zero), we are only interested in the covariance. This is because of isotropism, defined by the hyper-parameter such that the covariance is actually $\theta^{-1}I$ (with I the identity matrix). The problem is then modeled through the joint distribution for y and f using the isotropic Gaussian, and the marginal distributions p(f) and p($\mathbf{Y}$) (which is an integration over f).

In the non-parametric model we build a model of joint distributions over the set of data point whose complexity and covariance is encoded into the Gram matrix (which is really a matrix of inner products between the data points, which depends on the distance between data points). The model is then interpretable very easily into a regression of adaptive complexity (linear, polynomial, . depending on the kernel function). By looking at the joint distribution over the new data (which is Gaussian as well), which is then used to predict the function value on the new data point. In the end, the non-parametric representation permits to perform highly complex model of data that can be used for regression without any apparent parameter, at the exception of the hyperparameter θ which denotes the precision of the random noise variable (which follows a Gaussian as well).

Question 8

The prior is here represented as a Gaussian Process under the expression: $p(f|\mathbf{X}, \theta)$

However, the definition of f is in itself inherent to the data X such that f is a random variable defined over $\mathbf{X}$ as $f_i = f(x_i)$. The expression is therefore equivalent to the simple prior $p(f)$ conditional to the hyper-parameters θ. The prior represents a gaussian process which follows a Gaussian centered at zero (such that we can eliminate a parameter by centering the datapoint at zero, and because f depends on the data points by definition, so does its Gaussian probability distribution). The covariance of the Gaussian probability distribution of the prior is defined on the distance of the data points between themselves, represented by their inner product under the kernel function k, reducing the only parameter of the prior to the hyper-parameters θ(which is the isotropic variance of the noise assumed in the model).

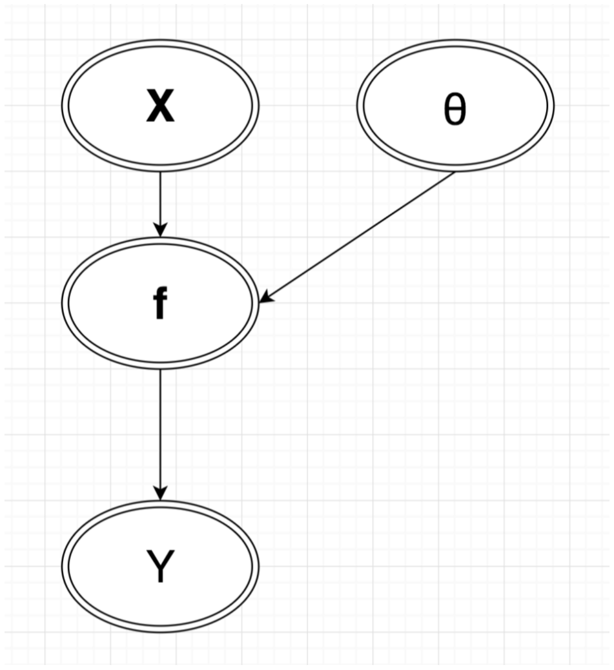
This prior reduces the infinite function space to a finite function space using the relationship of data points to each other in the kernel function as covariance in the Gaussian Process. The function space is then constrained under the covariance matrix. From this prior, we can then evaluate the marginal distribution of the y value which is performed through integration. Through this marginal distribution we can then use the model to generate predictions over unseen data points. This prior therefore aims at reducing the functions space over the data points for the computation of the marginal distribution over $\mathbf{Y}$.

Question 9

It encodes all possible functions. Due the Gaussian nature there is no instance where the probability mass is 0, however there is only a subset of functions that provide relevant information and the vast majority of the function space is effectively 0 (but not actually 0).

Question 10

Because $\mathbf{Y}$ is conditionally dependent on X and θ (that are themselves independent together), the model can be graphed as follows:
and the joint probability is expressed as:



$$p(\mathbf{Y}, \mathbf{X}, \boldsymbol{\theta}, f) = p(\mathbf{Y}|f)p(f|x, \boldsymbol{\theta})p(\mathbf{X})p(\boldsymbol{\theta})$$

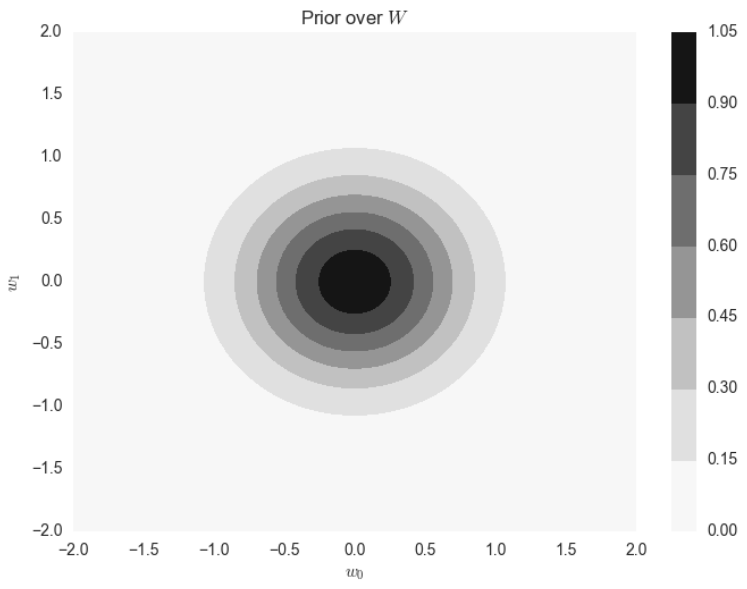
Question 11

The marginalization 'sums-up' the likelihood of our data points over all the function space given by the prior. That way using the prior and the joint distribution, following a Gaussian distribution: our marginalization follows a Gaussian distribution too.

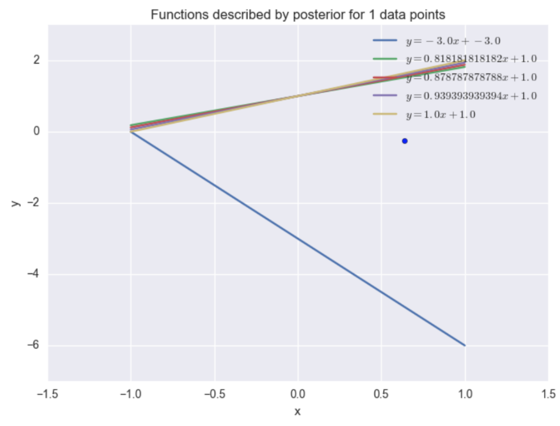
The uncertainty is transcribed into our model by merging the likelihood without our uncertainty in the joint distribution of the random variable f.

Because theta is a constant it remains in the model past the marginalization (the result of the marginalization is a function of theta): such that it is the only 'parameters' that needs to be learnt in the model, although its impact on the model impacts only accuracy of predictions by defining the variance of the error on each data points. It is the only thing that has to be learnt for the model to perform optimally.

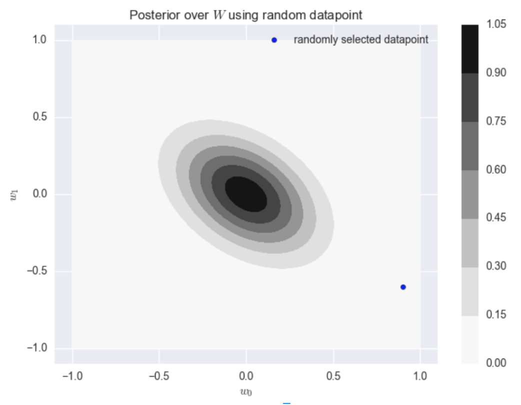
Question 12



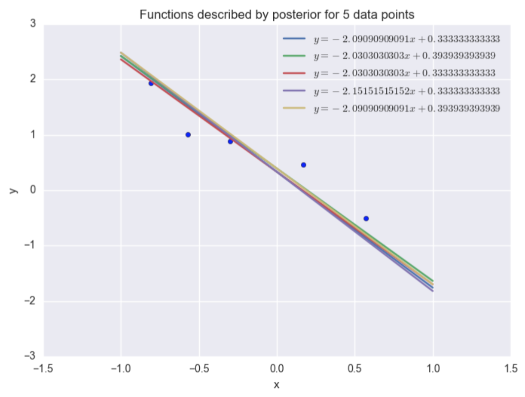
Prior over W
w_0
w_1
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00



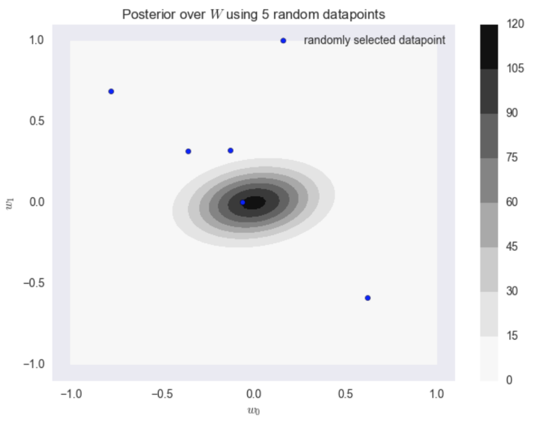
Functions described by posterior for 1 data points
$y = -3.0x + -3.0$
$y = 0.818181818182x + 1.0$
$y = 0.878787878788x + 1.0$
$y = 0.939393939394x + 1.0$
$y = 1.0x + 1.0$
x
y



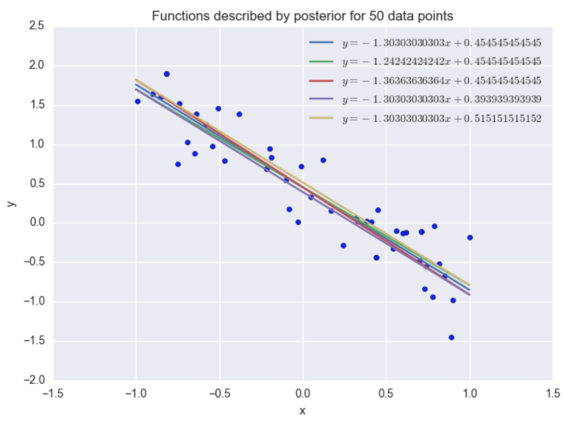
Posterior over W using random datapoint
randomly selected datapoint
w_0
w_1
1.05
0.90
0.75
0.60
0.45
0.30
0.15
0.00

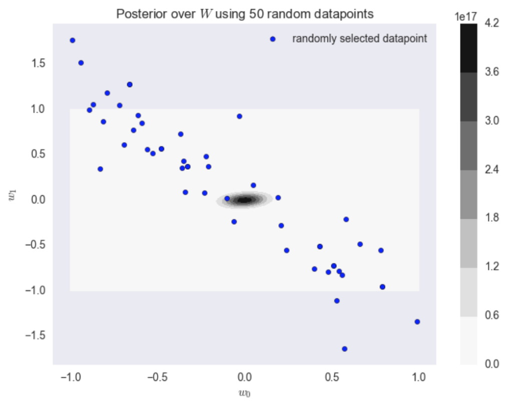


Functions described by posterior for 5 data points
$y = -2.09090909091x + 0.333333333333$
$y = -2.0303030303x + 0.393939393939$
$y = -2.0303030303x + 0.333333333333$
$y = -2.15151515152x + 0.333333333333$
$y = -2.09090909091x + 0.393939393939$
x
y



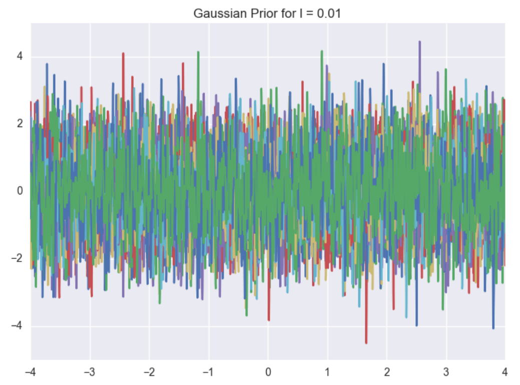
Posterior over W using 5 random datapoints
randomly selected datapoint
w_0
w_1
120
105
90
75
60
45
30
15
0

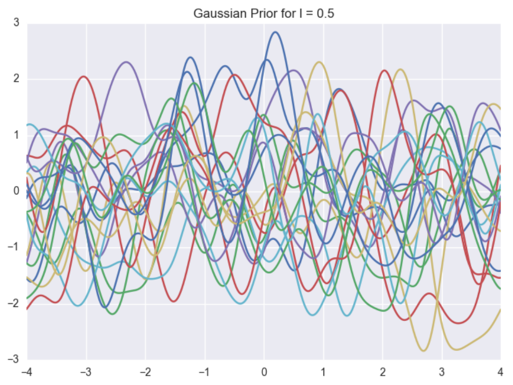


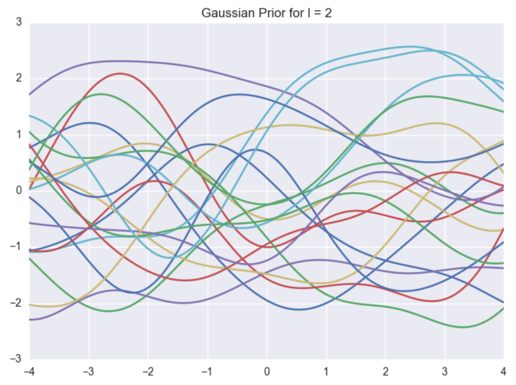


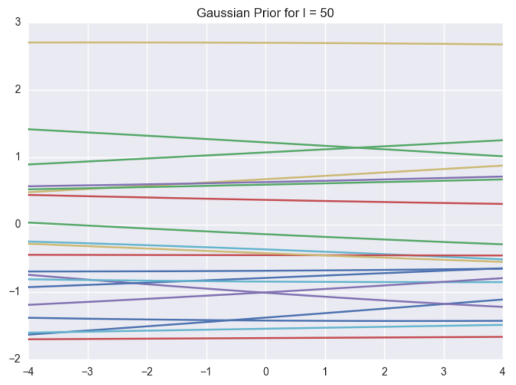
The prior is a simple expression of the known precision of the data points we have, that acts as the encoding of our assumptions on the data. From a single data points, we can see the posterior getting constrained by the likelihood of that single data point. As we add more data points, the spread of the posterior gets smaller and smaller, such that when we sample parameter vector from the posterior, the function prediction get more and more accurate and uniform: proving that the model converges. After 50 points, the model converges toward the correct solution.

Question 13



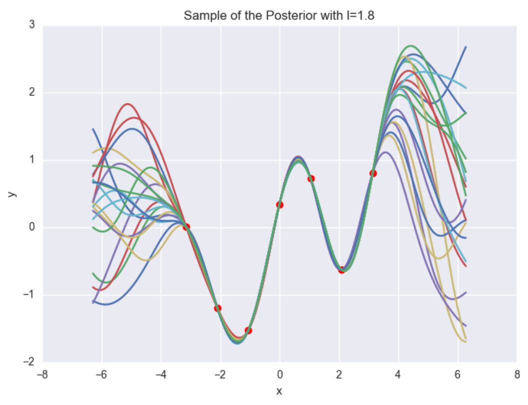


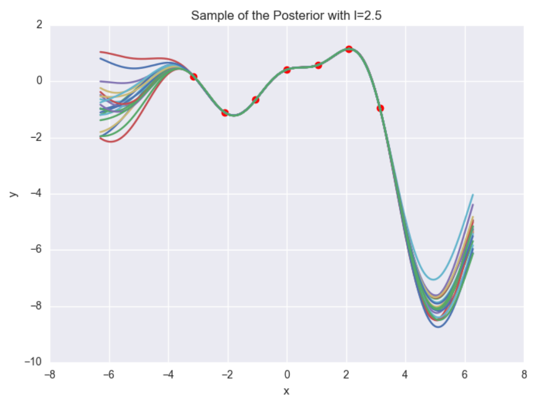


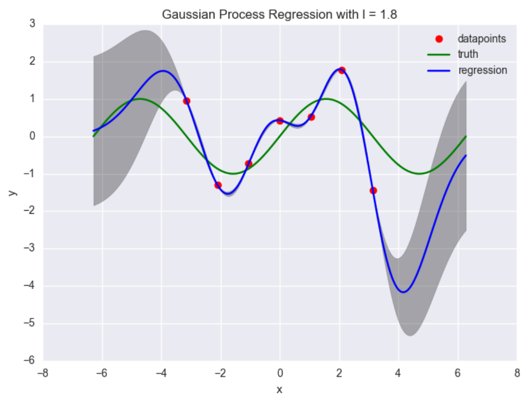


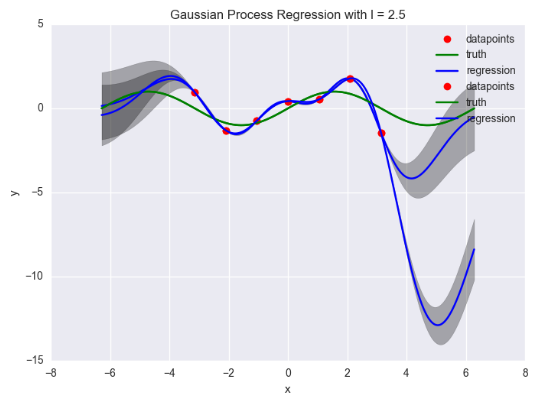
As we can see from the herein-above graphs, the larger the length-scale gets, the smoother the resulting prior is. As the length-scale gets bigger, the resulting $k(x_i, x_j)$ gets smaller and thus more confidence is attributed to the particular f_i and f_j . For a small l however, we attribute a low confidence to them and consider them to be uncorrelated. The length-scale encodes our belief in the correlation of the data.

Question 14









On the graphs we can observe how our uncertainty increases as we get away from the data points. We can as well see how our regression model does not follow the 'truth' sin wave outside of the data point range. We have shown the behavior of the regression at different range scales: namely $l = 1.8$ and $l = 2.5$. In zones of the plot where do not have data to update our assumptions, the regression tends to converge toward the prior observable in question 13, the variance shown in the shaded region increases also towards the corresponding prior variance. We can therefore establish a clear relation between the posterior and the prior in zones where do not have sufficient data to converge accurately toward a solution.

Adding a spherical covariance matrix to the kernel function would would be equivalent to adding independent Gaussian noise to previously obtained predictions through the simple kernel. It could prevent over-fitting by the model.

Question 15

An assumption is the formulation of prior knowledge on the data before actually looking at the data. However the preference is inherent to the data, it is a way of expressing a prior (in a Bayesian probabilistic sense) using some natural characteristics from samples of the data. For a given variable, the assumption would be encoded in the prior probability distribution, but reversely: the preference is encoded in the data and yields another form of probabilistic prior. By setting a prior over the variable $\mathbf{X}$, we are able to quantify our preference over the variable $\mathbf{X}$ to constraint our model (namely the variable W to be predicted) using our assumptions.

Question 16

Using a prior over **X** centered at 0 with standard deviation of 1 means that we consider our latent variable **X** to have independent dimension.

Question 17

Assuming our data points are noisy with noise: $\epsilon \approx N(0, \sigma^2 I)$ then assuming that $y_i = \mathbf{W}x_i + \epsilon$ we can express the likelihood as a Gaussian:

$$p(y_i|\mathbf{W}, x_i) = N(y_i|\mathbf{X}, \sigma^2\mathbf{I})$$

because the Gaussian distribution is it self conjugate, that the likelihood is Gaussian and that the prior is $p(\mathbf{X})N(0, \mathbf{I})$, then the marginalization is a Gaussian too. We just have to integrate the product of the two Gaussian or express the marginalization through the mean and the covariance of its distribution (which is more simple). The marginal distribution can be expressed as:

$$p(\mathbf{Y}|\mathbf{W}) = \int p(\mathbf{Y}|\mathbf{X}, \mathbf{W})p(\mathbf{X})d\mathbf{X}$$

such that,

$$p(y_i|\mathbf{W})N(E[y_i|\mathbf{W}], cov[y_i, y_i\mathbf{W}])$$

.
We can derive the mean and the covariance of those two values as follows:

$$E[y_i|\mathbf{W}] = E[\mathbf{W}x_i + \epsilon] = \mathbf{W}E[X_i] + E[\epsilon] = 0$$

and

$$cov[y_i, y_i|\mathbf{W}] = E[(y_i - E[y_i])(y_i - E[y_i])^T] = E[(\mathbf{W}x_i + \epsilon)(\mathbf{W}x_i + \epsilon)^T] = \mathbf{W}\mathbf{W}^T + \sigma^2 I$$

such that,

$$p(y_i|\mathbf{W})N(y_i|0, \mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})$$

We can then obtain the full marginalization using the product rule such that:

$$P(\mathbf{Y}|\mathbf{W}, \mathbf{X}) = \prod_{i=1}^{N} p(y_i|\mathbf{W})$$

.

Question 18

The Maximum Likelihood Estimation is a variable optimization on the likelihood probability distribution with no inclusion of the prior, where as the Maximum A Posterior Estimation is a variable on the Bayesian posterior which takes in account the prior probability distribution.

As we observe more data, the the likelihood has more and more influence on the posterior such that the prior is undermined. Therefore the MLE and MAP methods would converge toward the same solution. We can also say that the MAP method is more efficient with less data then the MLE would be with the same amount.

If we assume the equation to be an equivalence to Baye's rule, then the posterior is not actually dependent on the parameter W, such that the integral can be ignored.

Question 19

If we move the posterior in the log space then we can represent

$$p(\mathbf{Y}|\mathbf{W}) = \prod_{i=1}^{N} p(y_i|\mathbf{W})$$

as a sum such that:

$$ln(P(\mathbf{Y}|\mathbf{W})) = \sum_{i=1}^{N} ln(p(y_i|\mathbf{W})) = \sum_{i=1}^{N} ln(\frac{1}{2\pi|\mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}|} e^{-\frac{1}{2}(y_i^T(\mathbf{W}\mathbf{W}^T+\sigma^2\mathbf{I})^{-1})y_i})$$

$$= -\frac{ND}{2} ln(2\pi) + Nln(|\mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I}|) - \frac{1}{2}\sum_{i=1}^{N} y_i^T(\mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})^{-1}y_i$$

thus,

$$-log(P(\mathbf{Y}|\mathbf{W})) = L(\mathbf{W}) = \frac{1}{2}(NDln(2\pi) + Nln(|\mathbf{W}\mathbf{W}^T + \sigma^2 I|) + Trace((\mathbf{W}\mathbf{W}^T + \sigma^2\mathbf{I})^{-1}\mathbf{Y}\mathbf{Y}^T)$$

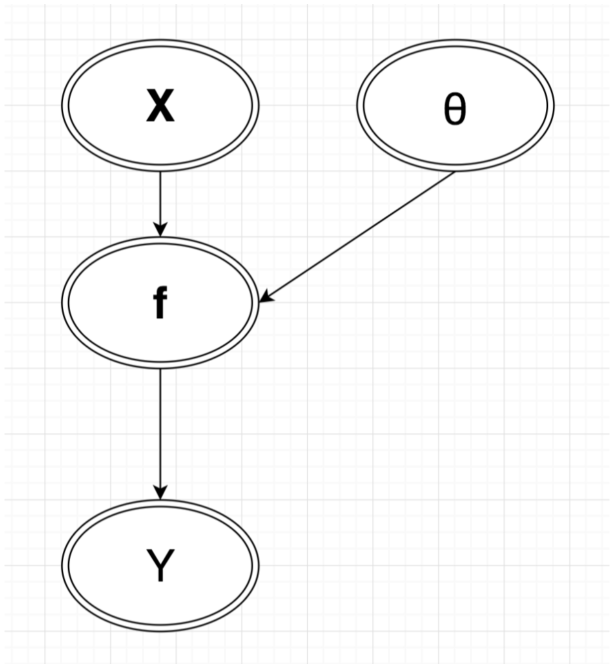
We now try to find the partial derivative of that expression with respect to $\mathbf{W}$ as:

$$(\frac{\partial L(W)}{\partial \mathbf{W}})_{ij} = \frac{1}{2}Trace(\mathbf{Y}\mathbf{Y}^T[-(\mathbf{W}\mathbf{W}^T+\sigma^2\mathbf{I})^{-1}\times\frac{\partial \mathbf{W}\mathbf{W}^T}{\partial \mathbf{W}_{ij}}\times(\mathbf{W}\mathbf{W}^T+\sigma^2\mathbf{I})^{-1}]) + \frac{N}{2}Trace[(\mathbf{W}\mathbf{W}^T+\sigma^2\mathbf{I})^{-1}\times\frac{\partial \mathbf{W}\mathbf{W}^T}{\partial \mathbf{W}_{ij}}]$$

with,

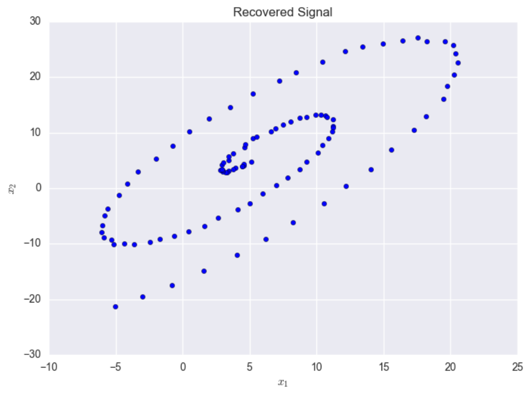
$$\frac{\partial \mathbf{W}\mathbf{W}^T}{\partial \mathbf{W}_{ij}} = J_{ij}\mathbf{W}^T + \mathbf{W}J_{ij}^T$$

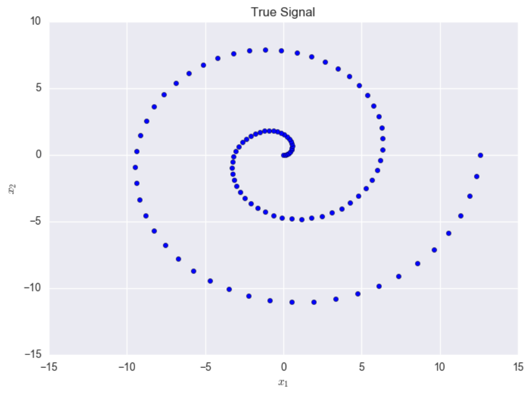
Question 20



The reason that the marginalization is much simpler in the non parametric model is because the latent variable f carries the uncertainty of both $\mathbf{X}$ and θ and does not require the integration over $\mathbf{X}$ that is present in out current situation.

Question 21





The representation we have found through Type-II Maximum Likelihood is rotated compared to the true signal $[x \times sin(x), x \times cos(x)]$ because the likelihood in the log-space is rotation invariant, therefore any rotated solution parameter is considered as correct by the optimizer.

Question 22

This model is the simplest model because instead of attributing a probability distribution, it simply gives a uniform probability of $\frac{1}{512}$ which is the parameter M_0 or the mean of out data.

This model could also arguably be considered as the most complex model because the model's prediction will be consistent even with data way outside of the range of the initial datasets.

Question 23

M3 is definitely the most complex model of all because it has the most parameters, and because it can set of some of its parameter to zero, we can expect it to set most of its probability distribution over a wide range of data points (larger than the set of data points it learnt from). M2 on the over hand will attribute more probability to data-sets with decision boundaries near the origin, but M3 can offset its decision boundary thanks to the presence of the bias. But the flexibility of the model could actually be a bad thing for M3 as it spends a fair share of its probability distribution over non-observed data. Therefore we can say that M3 is more complex than M2, that M2 is more restrictive and M3 more flexible .

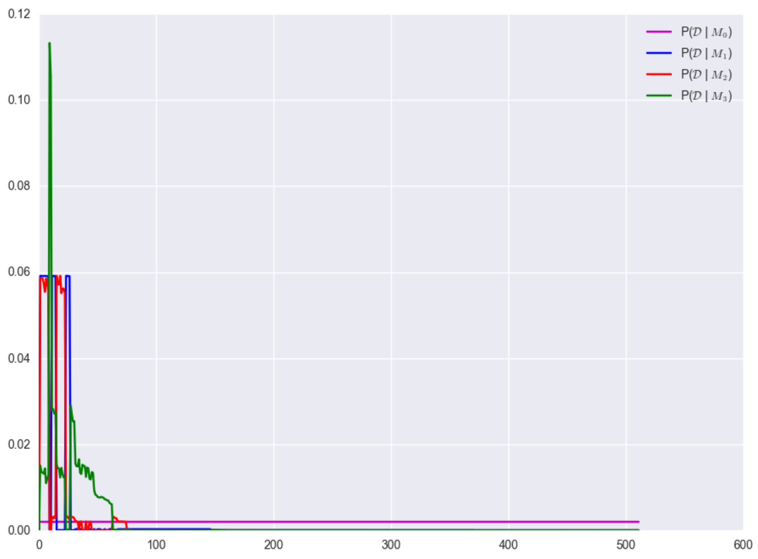
Question 24

We choose a spherical covariance for the prior, such that the elements of θ are independent and we also choose a high variance which means that we have a preference for sharp decision-boundary.

Question 25

The evidence for each dataset sums up to one. The evidence over M_0 is constant, the evidence over M_1 oscillates over a narrow range of values, the evidence over M_2 oscillates as well but over a wider range of values and finally the evidence of M_3 oscillates in higher values and over an even wider ranges of values.

Question 26



The main differences in the models shown by the evidence plot is in the span of the probability mass associated over the instances of the data presented by the x-axis. The evidence of M_3 is maximal and ranges over the wider x-axis value, for a complex model. Although M_0 is model that is the most consistent over the range of data values. We can see that as we increase the number of parameters (thus the complexity) the spread over the instances of D is larger and the evidence peaks higher. The more complex the model, the more boundaries can be computed, especially as with M_3 we have a bias term.

Question 27

M_0 : This model has uniform probability mass over D, thus there is no minimum or maximum.
M_1 : This model only takes in account decision boundaries that are function of x_1 and not x_2. It associates most probability to simple boundaries function x_1:

Table 1: M_1 Most Probable

0	0	x
0	0	x
0	x	x

and it associates lowest probability to a non-linearly separable dataset:

Table 2: M_1 Least Probable

0	x	0
0	x	x
x	x	0

M_2 : This model is a standard logic regression, however its decision boundary crosses the origin, such that its highest probability is non-linear (but not very complex) and crosses the origin:

Table 3: M_2 Most Probable

0	0	0
x	0	0
x	x	0

and its least probable is as well highly nonlinear and does not cross the origin:

Table 4: M_2 Least Probable

x	0	0
x	0	x
0	0	0

M_3 : This model behaves like the prior but has an offset from the origin with the bias term. It therefore favors sharp decision boundary. Its most probable configuration would thus be of the form :

Table 5: M_3 Most Probable

x	x	x
x	x	x
x	x	x

and its least probable configuration would be a highly nonlinear not sharp decision boundary:

Table 6: M_3 Least Probable

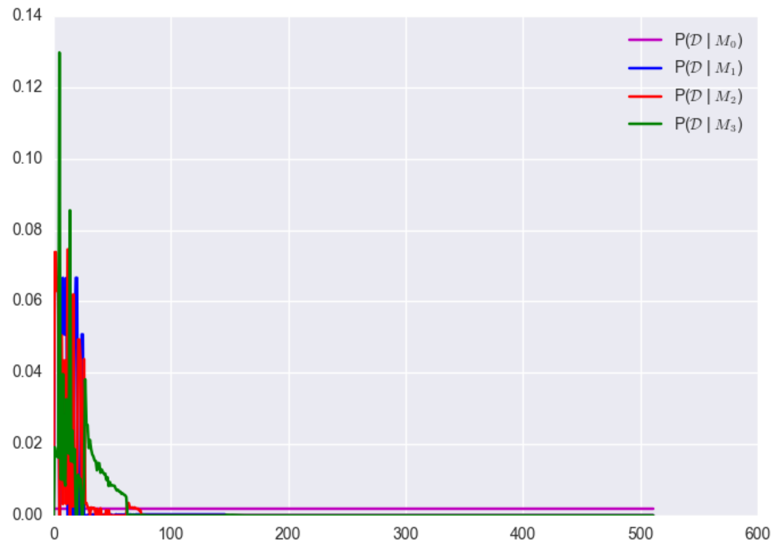
0	x	0
0	0	x
x	0	0

Question 28

We have seen from the models that greater variances, the sharper the decision boundary. This would imply that if the variance was smaller we would obtain a more uniform decision boundary similar to M_0 .

If the covariance was not spherical, it would assign some dependence on the data, which would shift the probability mass distribution based on the correlation of the data.

The value of the mean simply defines the 'origin' of the decision boundary. For models that have to cross this origin (all models except M_3) this would carry a higher probability mass to data around that new mean, the same way it did around zero when we had a zero mean. This is shown by the following graph, as the evidence peaks more around those data:



Question 29

Same as Question 28.

Question 30

This coursework has supplied us with an example of both supervised and unsupervised learning as well as an increased theoretical understanding when approaching model selection. Incorporating a wide range of machine learning techniques, from linear regression to highly non-linear classification presents us with a greater knowledge bank to apply on a larger number of real world data sets. Something we observed to have reoccurring importance was the concept of encoding our assumptions with the evidence shown by the data in order to create powerful and complex models. We were particularly taken back by how Bayes rule can be applied to such a wide variety of models, frequent being useful by offering different results fitting the context of it's application.

A Python Code used for Coursework

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import euclidean
from scipy.stats import multivariate_normal as mv_norm
import seaborn
import scipy.optimize as opt
from itertools import product

def BayesianLinearRegression(s, plotprior=False):
    x = np.linspace(-1, 1, 201)
    W = np.array([-1.3, 0.5])
    epsilon = np.random.normal(0.0, 0.3, x.shape[0])
    y = W[0]*x + W[1] + epsilon
    precision = 1/0.3
    m_0 = [0, 0]
    s_0 = (1/precision)*np.identity(2)
    prior = mv_norm(m_0, s_0)

    if plotprior:
        plotPrior(prior)

    size = s
    datapoint, idx = samplerandomdatapoints(size, x, y)
    posterior, possiblew, W0, W1 = computeposteriorOverDataPoints(prior, datapoint)
    plotFunctions(size, possiblew, posterior, datapoint)
    plotPosterior(size, posterior, W0, W1, datapoint)

def samplerandomdatapoints(s, x, y):
    idx = np.random.randint(0, x.shape[0],s)
    datapoint = np.stack((x[idx], y[idx]), axis=-1)

    return datapoint, idx

def computeposteriorOverDataPoints(prior, datapoints):

    sigma = np.cov(datapoints[:,0], datapoints[:,1])[0,0]

    possiblew = np.linspace(-3, 3, 100)
    W0, W1 = np.meshgrid(possiblew, possiblew)
    possiblew = np.stack((W0.flatten(), W1.flatten()), axis=-1)
    priorpdf = prior.pdf(possiblew)

    likelihood = 1
    for i in range(0, datapoints.shape[0]):
        likelihood_i = (1/np.sqrt(2*np.pi*(sigma**2)))
        likelihood_i = np.multiply(likelihood_i,  np.exp(np.square(datapoints[i,1]-(datapo
        likelihood = np.multiply(likelihood, likelihood_i)

    likelihood /= datapoints.shape[0]

    posterior = priorpdf*likelihood

    return posterior, possiblew, W0, W1

def plotFunctions(s, possiblew, posterior, datapoint):

    indices = np.argsort(posterior)[::1]

    for i in range(0, 5):
        idx = indices[i]
        x_axis = np.linspace(-1, 1, 201)
        y_axis = x_axis*possiblew[idx, 0] + possiblew[idx, 1]
        plt.plot(x_axis, y_axis, label='$y=' + str(possiblew[idx, 0]) + 'x + ' + str(possib
    plt.scatter(datapoint[:,0], datapoint[:,1])
    plt.legend()
    plt.title('Functions described by posterior for '+str(s)+' data points')
    plt.xlabel('x')
    plt.ylabel('y')
    plt.savefig('function-'+str(s)+'datapoint.png')
    plt.close()

def plotPosterior(s, posterior, W0, W1, datapoint):

    posterior = np.reshape(posterior, (100, 100))
    cs = plt.contourf(W0,W1,posterior)
    plt.scatter(datapoint[:,0], datapoint[:,1], label='randomly selected datapoint')
    plt.legend()
    cbar = plt.colorbar(cs)
    plt.grid(False)
    plt.xlabel(r'$w_0$')
    plt.ylabel(r'$w_1$')
    plt.title('Posterior over $W$ using '+str(s)+' random datapoints')
    plt.savefig('posterior'+str(s)+'.png', facecolor='white', edgecolor='white')
    plt.close()

def plotPrior(prior):

    x = np.linspace(-2,2,100)
    xx, yy = np.meshgrid(x, x)
    values = np.stack((xx,yy), axis=-1)
    proba = prior.pdf(values)
    proba = (proba-proba.min())/(proba.max()-proba.min())
    cs = plt.contourf(xx, yy, proba)
    cbar = plt.colorbar(cs)
    plt.xlabel(r'$w_0$')
    plt.ylabel(r'$w_1$')
    plt.title('Prior over $W$')
    plt.savefig('prior.png')
    plt.close()


def plotPrior(l):
    X = np.linspace(-4.0,4.0,2000) [:,None]
    mu = np.zeros(shape=(2000))
    K = np.exp(-euclidean(X, X)/(l*l))
    Z = np.random.multivariate_normal(mu,K,20)
    for i in range(20):
        plt.plot(X[:],Z[i,:])
    plt.title('Gaussian Prior for l = '+str(l))
    plt.savefig('prior' + str(l) + '.png')

def kernelFunction(X, Y, l=1.):
    return np.exp(-euclidean(X, Y)/(l*l))

def PlotPosterior():
    X = np.linspace(-np.pi, np.pi, 7)
    epsilon = np.random.normal(0, 0.5, 7)
    Y = np.sin(X) + epsilon
    x = np.linspace(-2 * np.pi, 2 * np.pi, 800)[:,None]
    list = [1.8, 2.5]
    for l in list:
        X = X[:, None]
        k = kernelFunction(x, X, l)
        C = np.linalg.inv(kernelFunction(X, X, l))
        t = Y[:, None]
        mu = np.dot(np.dot(k, C), t)
        c = kernelFunction(x, x, l)
        sigma = c - np.dot(np.dot(k, C), np.transpose(k))

        # plot observations
        plt.plot(X, Y, 'ro', label='datapoints')
        plt.plot(x, np.sin(x), color='green', label='truth')
        mu = np.reshape(mu, (800,))
        plt.plot(x, mu, color='blue', label='regression')
        upper = mu + 2 * np.sqrt(sigma.diagonal())
        lower = mu - 2 * np.sqrt(sigma.diagonal())
        ax = plt.gca()
        ax.fill_between(x, upper, lower, facecolor='k', interpolate=True, alpha=0.3)
        plt.title('Gaussian Process Regression with l = ' + str(l))
        plt.xlabel('x')
        plt.ylabel('y')
        plt.legend()
        plt.savefig('regl=' + str(l) + '.png')

def plotSamplePosterior():
    X = np.linspace(-np.pi, np.pi, 7)
    epsilon = np.random.normal(0, 0.5, 7)
    Y = np.sin(X) + epsilon
    l=1.8
    x = np.linspace(-2*np.pi, 2*np.pi, 800)[:, None]
    X = X[:, None]
    k = kernelFunction(x, X, l)
    C = np.linalg.inv(kernelFunction(X, X, l))
    t = Y[:, None]
    mu = np.dot(np.dot(k, C), t)
    c = kernelFunction(x, x, l)
    sigma = c - np.dot(np.dot(k, C), np.transpose(k))
    mu = np.reshape(mu,(800,))
    x = x[:,None]
    Z = np.random.multivariate_normal(mu,np.nan_to_num(sigma),20)
    plt.plot(X,Y,'ro')
    for i in range(20):
        plt.plot(x[:],Z[i,:])
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Sample of the Posterior with l=1.8')
    plt.savefig('samplePos1.png')


def Optimization():
    A = np.random.randn(20)
    A = A.reshape((10, 2))

    x = np.linspace(0, 4 * np.pi, 100)
```

```
X = np.zeros((100, 2))
X[:, 0] = np.multiply(x, np.cos(x))
X[:, 1] = np.multiply(x, np.sin(x))

Y = np.dot(X, np.transpose(A))
noise = np.random.multivariate_normal(np.zeros(10), 0.1 * np.eye(10), 100)

plt.scatter(X[:, 0], X[:, 1])
plt.title('True Signal')
plt.xlabel('$x_1$')
plt.ylabel('$x_2$')
plt.savefig('question20/truth.png')
plt.close()
Y = Y+noise

A = 20 * np.random.randn(20)
A = np.reshape(A, (20,))


def functionf(W):
    sigma = 2
    W = np.reshape(W, (10, 2))
    H = np.dot(W, np.transpose(W))
    I = sigma * np.eye(10)
    inv = np.linalg.inv(H + I)
    A = 50 * np.log(np.linalg.det(H + I))
    B = 0.5 * np.trace(np.dot(inv, np.dot(np.transpose(Y), Y)))
    return A + B + 0.5 * 10 * 100 * np.log(2 * np.pi)

def derivativef(W):
    sigma = 2
    W = np.reshape(W, (10, 2))
    inv = np.linalg.inv((np.dot(W, np.transpose(W)) + sigma * np.eye(10)))

    values = np.empty(W.shape)
    for i in range(values.shape[0]):
        for j in range(values.shape[1]):
            J = np.zeros(np.shape(W))
            J[i, j] = 1
            dWW = np.dot(J, np.transpose(W)) + np.dot(W, np.transpose(J))
            A = 100 * np.trace(np.dot(inv, dWW))
            B1 = np.dot(np.dot(-inv, dWW), inv)
            B = np.trace(np.dot(np.dot(np.transpose(Y), Y), B1))
            values[i, j] = 0.5 * A + 0.5 * B
    val = np.reshape(values, (20,))
    return val

Wstar = opt.fmin_cg(functionf, A, fprime=derivativef)
W = np.reshape(Wstar, (10, 2))
inv = np.linalg.pinv(np.dot(np.transpose(W), W))
X = np.dot(Y, np.dot(W, np.dot(np.transpose(W), W)))

plt.scatter(X[:, 0], X[:, 1])
plt.xlabel('$x_1$')
```

```
    plt.ylabel('$x_2$')
    plt.title('Recovered Signal')

    plt.savefig('recovered.png')


def SampleFromPrior(modelNumber, samples):
    sigma = 1000
    cov = sigma*np.eye(modelNumber)
    mean = np.ones(modelNumber) * 5
    return np.random.multivariate_normal(mean, cov, samples)

def CalculateEvidence(dataset, modelNumber, samples):
    p=0
    for i in range(len(samples)):
        if modelNumber == 0:
            return 1 / 512
        p1 = 1
        for i in range(3):
            for j in range(3):
                if modelNumber == 1:
                    p1 = p1 * 1 / (1 + np.exp(-dataset[i, j] * samples[i][0] * (i - 1)))
                if modelNumber == 2:
                    p1 = p1 * 1 / (1 + np.exp(-dataset[i, j] * (samples[i][0] * (i - 1) + s
                if modelNumber == 3:
                    p1 = p1 * 1 / (1 + np.exp(-dataset[i, j] * (samples[i][0] * (i - 1) + s
        p = p + p1

    return p/len(samples)

def Evidence():
    samples1 = SampleFromPrior(1, 10 ** 4)
    samples2 = SampleFromPrior(2, 10 ** 4)
    samples3 = SampleFromPrior(3, 10 ** 4)
    combinations = list(product([-1, 1], repeat=9))
    sets = []
    for l in combinations:
        arr = np.asarray(l)
        grid = np.reshape(arr, (3, 3))
        sets.append(grid)
    l = sets

    evidence = np.zeros([4, 512])

    for i in range(4):
        print(i)
        for j in range(512):
            if i == 0:
                evidence[i][j] = CalculateEvidence(l[j], i, samples1)
            if i == 1:
                evidence[i][j] = CalculateEvidence(l[j], i, samples1)
            if i == 2:
                evidence[i][j] = CalculateEvidence(l[j], i, samples2)
            if i == 3:
```

```
            evidence[i][j] = CalculateEvidence(l[j], i, samples3)

max = np.argmax(evidence, axis=1)
min = np.argmin(evidence, axis=1)
sum = np.sum(evidence, axis=1)
print(max, min, sum)

dist = np.zeros([evidence.shape[0], evidence.shape[0]])
for i in range(dist.shape[0]):
    for j in range(dist.shape[1]):
        dist[i, j] = evidence[i] - evidence[j]
        if i == j:
            dist[i, j] = pow(10, 4)

index = [];
D = np.arange(evidence.shape[0]).tolist()
ind = evidence.argmin()
index.append(ind)
D.remove(ind)

while D:
    N = []
    for i in range(len(D)):
        ind = dist[D[i], D].argmin()
        if D[ind] == index[-1]:
            N.append(D[ind])
    if not N:
        index.append(D[dist[index[-1], D].argmin()])
    else:
        index.append(N[dist[index[-1], N].argmin()])
    D.remove(index[-1])

plt.plot(evidence[0, index], label="P($\mathcal{D}$ | ${M}_0$)")
plt.plot(evidence[1, index], label="P($\mathcal{D}$ | ${M}_1$)")
plt.plot(evidence[2, index], label="P($\mathcal{D}$ | ${M}_2$)")
plt.plot(evidence[3, index], label="P($\mathcal{D}$ | ${M}_3$)")
plt.legend()
plt.show()
```