CSCI-GA.2565-001 Machine Learning: Homework 2

Due 5pm EST, March 10, 2022 on Gradescope

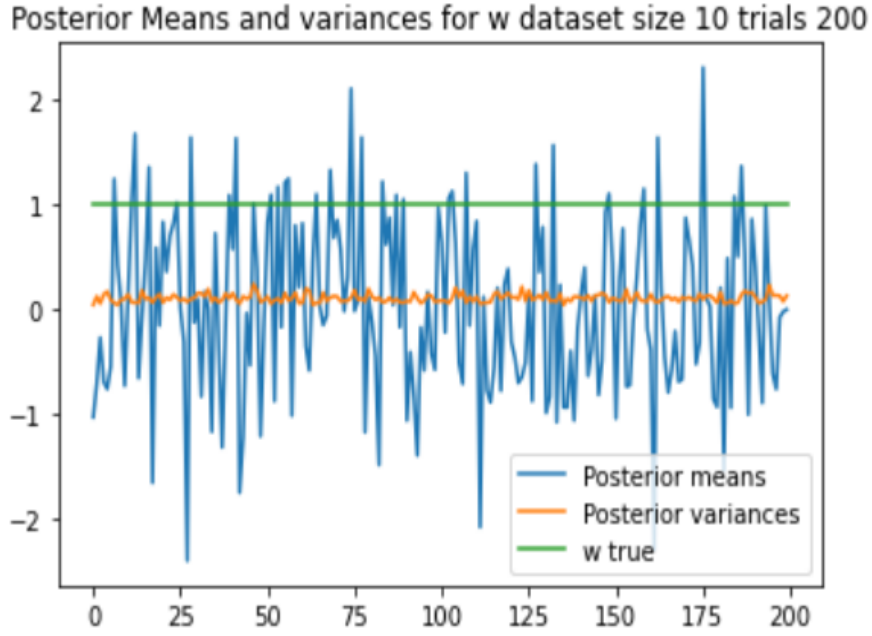
We encourage LaTeX-typeset submissions but will accept quality scans of hand-written pages.

1 Bayesian Linear Regression

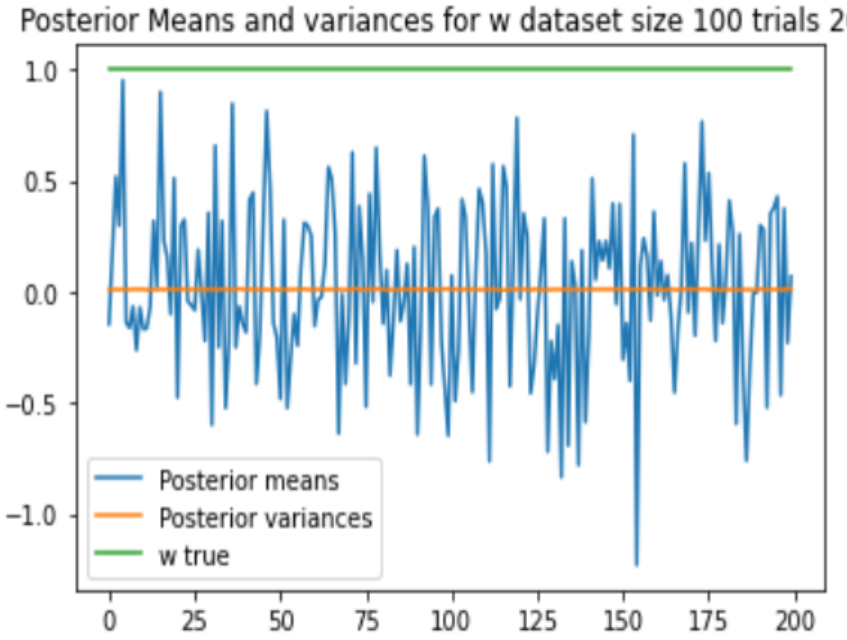
Consider a data distribution with $x \sim \mathcal{N}(0,1), \epsilon \sim \mathcal{N}(0,\sigma^2)$, and $y = w_{true}x^2 + \epsilon$ for $w_{true} = 1.0$ and $\sigma^2 = 1.0$. Not knowing this, we approach this (x, y) data with Bayesian linear regression.

(a) Assume prior $w \sim N(0,1)$ and let the likelihood be $y \sim N(wx, \sigma^2)$ for $\sigma^2 = 1.0$. Compute the posterior mean and variance of w after $N = 10, 100, 1000$ and $10,000$ points. Does the posterior concentrate on w_{true}? Why or why not?

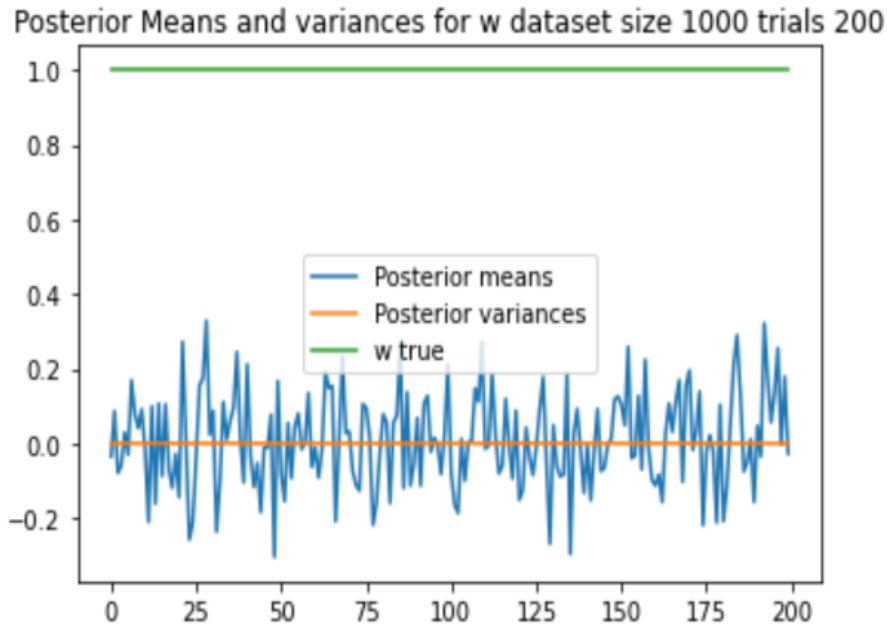
For the given values of N, we conduct T trials. In each trial, we sample a data set of size N. We then find the posterior mean for w and the posterior variance. We plot the posterior means of w for various trials for that value of N. The posterior distribution for w does not concentrate around w_{true}. As N increases, the empirical variance of the posterior mean of w tends to decrease. We also plot for the variance of w as obtained analytically.



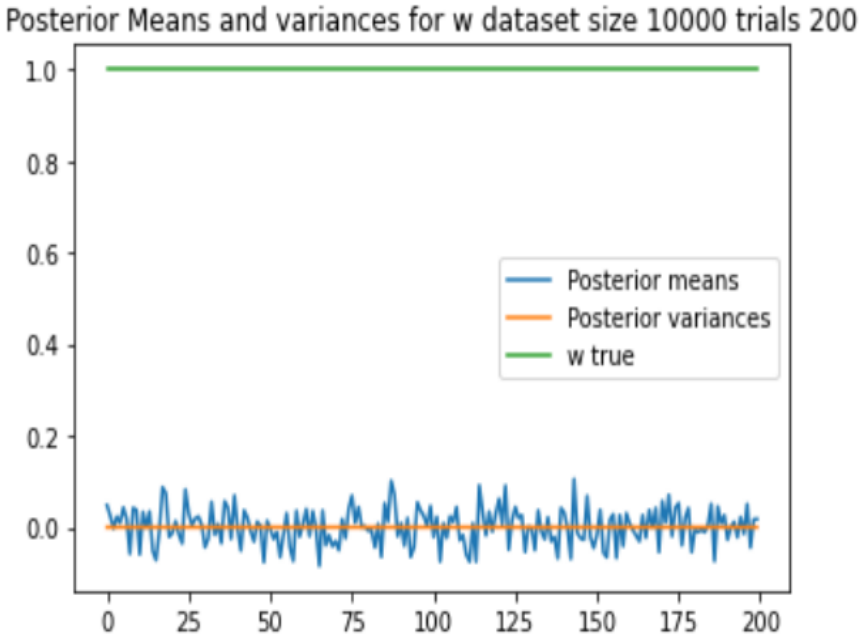

```
mean of posterior mean is tensor([0.0166])
```




```
mean of posterior mean is tensor([0.0248])
```




```
mean of posterior mean is tensor([0.0100])
```




```
mean of posterior mean is tensor([0.0017])
```

We take a look at the analytical expressions for the maximum likelihood estimate of w and the posterior mean of w. The posterior mean for w is given by,

$$w_{posterior} = \frac{x^T y}{1 + x^T x}$$

And the variance for w,

$$variance_{posterior} = \frac{1}{1 + x^T x}$$

Consider the posterior mean, $x^T x$ in the denominator can be considered as $N * \frac{x^T x}{N}$. Here, the second term is an empirical approximation of $E[X^2]$. We know that the concerned expectation is $\mu^2 + \sigma^2$ which is $1.x^T x$ can be considered as close to N. Even empirically, we can see that the $x^T x$ can be close to the N value, especially for higher values of N. For the numerator, we consider the following expectation where X, Z are independent standard normal distributions.

$$E_{p(X,Z)}[(X^2 + Z)X]$$

Following from the factorization, expanding, knowing that $E[X^3]$ is zero, we can see that this expression is 0.Notice that $\frac{x^T y}{N}$ acts as an empirical approximation to this expectation. Even empirically, we can see the value of $x^T y$ to be close to 0.

Based on our plots, the posterior mean of w is closer to zero as compared to the true value of w which is 1. We can see that the posterior does not concentrate around the true value of w.

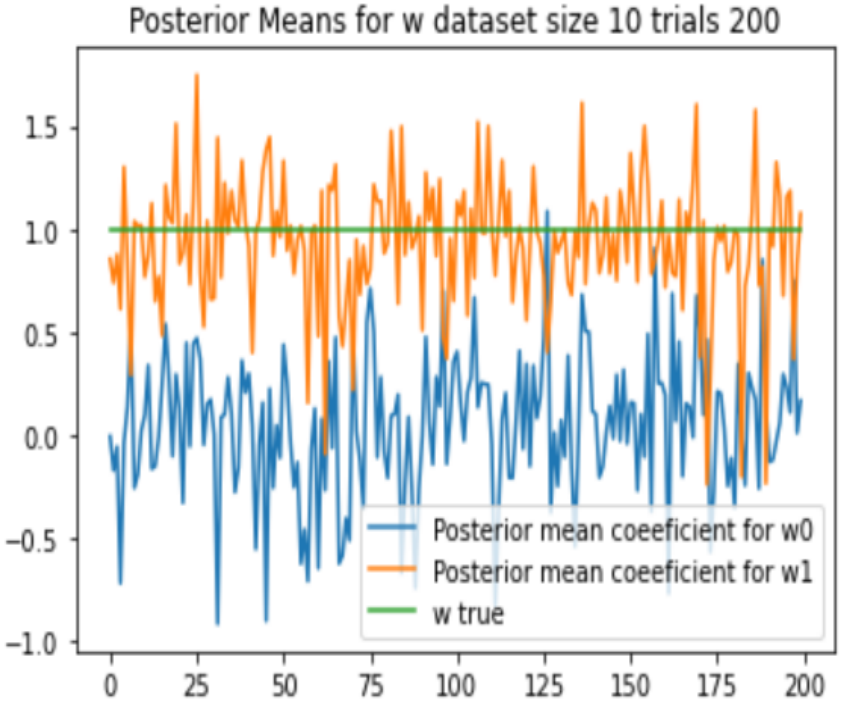
If we consider the hypothesis space for this question, the set of solutions which our model will produce does not contain the correct solution representing the input to output mapping for our problem. Introducing priors, we get a result for w consisting of an aggregation of the prior beliefs for w and the likelihood from the data. The purpose of a prior can be for regularization, but here, the model setting itself is not sufficient enough to capture the true distribution.

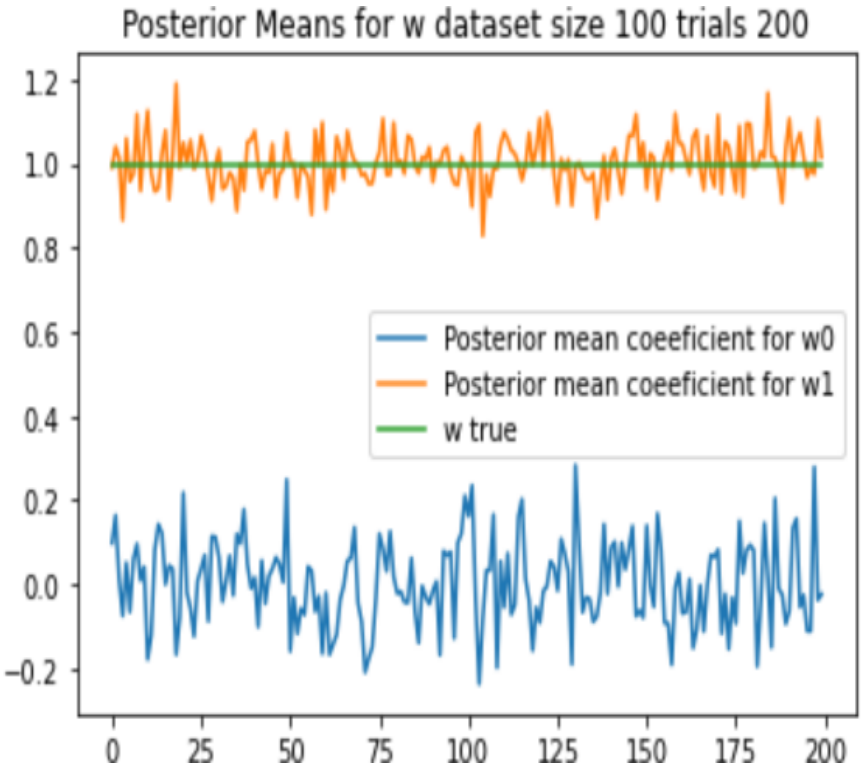
(b) What would be challenging about this if we picked a different prior, for example Laplace or Gamma? The Gaussian prior is the conjugate prior for the Gaussian likelihood, we have a neat analytical form for

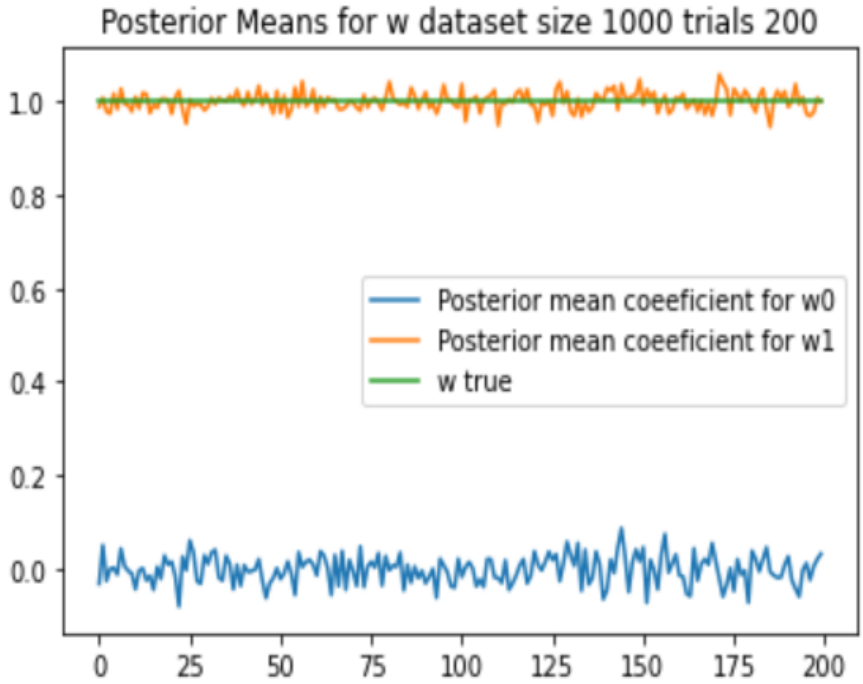
the expression of the posterior distribution. With a laplace or gamma prior, while we may find the MAP estimate for w based on the loss, we are unable to find easily, a posterior distribution for w.

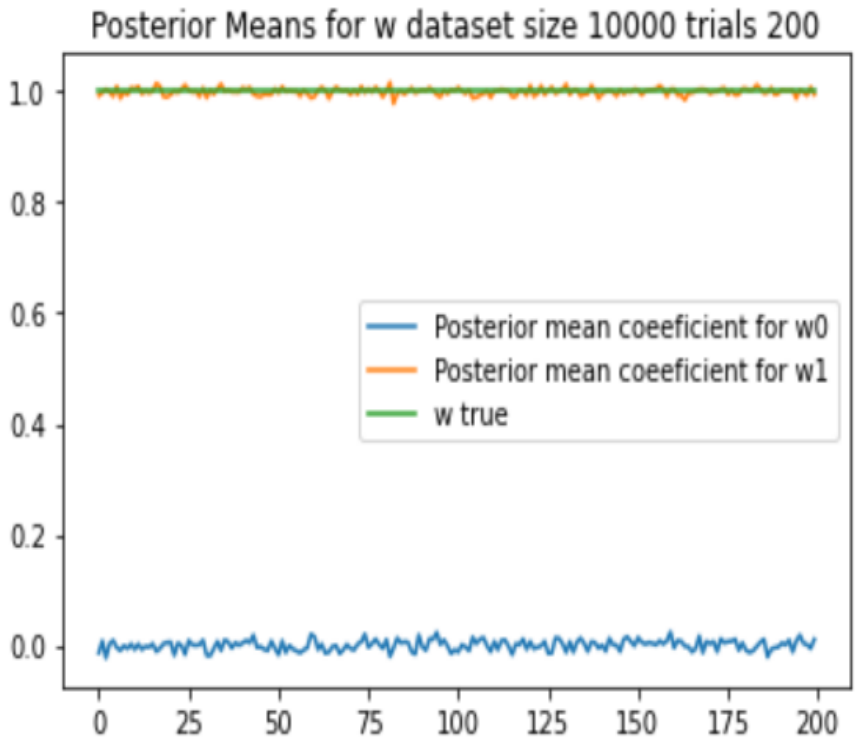
(c) Instead of x, we use the basis $\phi(x) = [x, x^2]$ and perform 2D Bayesian linear regression with $w \sim N(0, I)$ where I is the 2×2 identity matrix and likelihood $y \sim N(w^\top \phi(x), \sigma^2)$ for $\sigma^2 = 1.0$. Compute the posterior mean and variance of the two coordinates w_1, w_2 after $N = 10, 100, 1000$ and $10,000$ points. What do you observe? Why?

In this model, we get the the posterior mean for the coefficient of x^2 tends to 1 and the posterior mean of the coefficient of x tends to zero. This is in accordance with our true model. The empirical variance observed in the posterior means of the weights decreases as N increases.




Posterior Means for w dataset size 100 trials 200
Posterior mean coeeficient for w0
Posterior mean coeeficient for w1
w true


Posterior Means for w dataset size 1000 trials 200
Posterior mean coeeficient for w0
Posterior mean coeeficient for w1
w true



For this basis, the hypothesis space contains our desired solution which is $y = w_{true}x^2 + 0.x + \epsilon$. This is as hypothesis space consists of solutions of the form $y = w_1x + w_2x^2 + \epsilon$. For the maximum likelihood estimate, the range of possible values which the parameters can take is unconstrained so we tend to learn values of the parameters such that there is variance in the model with respect to the set of functions learned as there is a tendency for over-fitting. The introduction of priors can act as a regularizer, priors of a gaussian standard normal will add belief to the claim that the weights are coming from a standard normal distribution.

(d) The previous answer seems to suggest that we should use large feature sets or bases in Bayesian linear regression. How does this relate to your findings in HW1 Q1 about regression with many correlated features?

If we use large feature sets or bases, it is likely that the desired solution likes in our hypothesis space. However, as our model complexity increases, our variance increases, the tendency for over-fitting increases and we would want to incorporate a prior which can act as a regularization. In Bayesian linear regression, as we are incorporating priors, we can plausibly use a large set of bases in our model while not risking heavy over-fitting.

(e) The *model class* is the set of possible predictive models based on your assumptions and learning algorithm, e.g. *the mean of y is all linear functions of x* or *mean of y is all quadratic functions of x*. Which of the prior and/or the likelihood specify the model class? The likelihood is of the form.

$$p(y/x; w) = \mathcal{N}(w_1x + w_2x^2; \sigma^2)$$

$$E[y/x] = w_1x + w_2x^2$$

This represents our model class, since our solution for the mean of y belongs to the class of quadratic functions in x.

2 Very Random Forest

Consider building a random forest by both subsampling the data and choosing a single feature per tree randomly. For example, consider a dataset $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i \in \mathbb{R}^D$ and $y_i \in \mathbb{R}$ for $i = 1, .., N$. A tree would be constructed as follows:

1. Randomly sample one feature index $j \in \{1, \ldots, D\}$
2. Draw a sample of the data $\mathcal{D}_{\mathbf{k}}$ of size $M \leq N$ with replacement. These datapoints will have indices $\mathbf{k} = k_1, \ldots, k_M$.
3. Keep only the j^{th} feature of the M samples: i.e. letting $x_{i,j}$ be the i^{th} datapoint and j^{th} feature, we use data
$$\mathcal{D}^j_{\mathbf{k}} = \{(x_{(k_1,j)}, y_{(k_1)}), ..., (x_{(k_M,j)}, y_{(k_M)})\}$$
4. Build a decision tree on $\mathcal{D}^j_{\mathbf{k}}$.
5. Repeat the above process R times so that r^{th} tree T_r uses feature $j^{(r)}$ and data $\mathbf{k}^{(r)}$ for $r \in \{1, \ldots, R\}$. Using this notation, the prediction of the r^{th} tree on new input $\mathbf{x}^\star$ is $T_r(\mathbf{x}^\star; \mathcal{D}^{j^{(r)}}_{\mathbf{k}^{(r)}})$.
6. Sum R of these random trees to construct the very random forest. That is, for input $\mathbf{x}^\star$, the very random forest predicts $\hat{y} = \sum_r T_r(\mathbf{x}^\star; \mathcal{D}^{j^{(r)}}_{\mathbf{k}^{(r)}})$

(a) For which class of conditional distributions for $y \mid \mathbf{x}$ are very random forests unbiased? Here unbiased is in the sense of bias-variance. You may make and clearly state any assumptions about N, M, R, or any details about the trees when forming your answer.

For our prediction, $\hat{y} = f(x) = \sum_r T_r(\mathbf{x}^\star; \mathcal{D}^{j^{(r)}}_{\mathbf{k}^{(r)}})$ If our tree minimized the squared error norm, we can choose the conditional expectation as our prediction. If we have our Gaussian distribution for prediction of y given $\hat{y}$, the conditional expectation is our mean which is $\hat{y}$.

$$Bias = E_{P(D)}[f(x)] - E[y/x]$$

For our model to be unbiased,

$$E_{P(D)}[f(x)] = E[y/x]$$

$$E[y/x] = E[\sum_r T_r(\mathbf{x}^\star; \mathcal{D}^{j^{(r)}}_{\mathbf{k}^{(r)}})]$$

$$E[y/x] = \sum_r E[T_r(\mathbf{x}^\star; \mathcal{D}^{j^{(r)}}_{\mathbf{k}^{(r)}})]$$

$$E[y/x] = \sum_r E[y/x^{j^{(r)}}]$$

Let c_j denote the number of trees in R using feature j. We can average over these trees.

$$E[y/x] = \sum_r \sum_j \frac{E[y/x^j] I(x^r = x^j)}{c_j}$$

$$E[y/x] = \sum_j \sum_r \frac{E[y/x^j] I(x^r = x^j)}{c_j}$$

Therefore,

$$E[y/x] = \sum_j E[y/x^j]$$

$$E[y/x] = \sum_{j=1}^{n} E[y/x^j]$$

This is the form of E[y/x] which we get from our one model. If this condition holds in our data, in the sense that such a formulation is part of the data generating process, then expectation of predicted values over the data set can be equal to our predicted value given by the conditional expectation based on our model.

(b) Compare the bias and variance of this very random forest with the traditional random forest that selects a random subset of the data and a random subset of features to build each tree. *Hint: Look at the generalization bound from the lecture on random forests. You only need to look at the final result, not the derivation.*

Recall that for the generalization error,

$$ge = E_{X,Y}[Margin(x, y) < 0]$$

We have the strength s for the random forest,

$$s = E_{X,Y}[Margin(X, Y)]$$

We have,

$$ge \leq \frac{\mathbb{V}ar[margin]}{\mathbb{E}[margin]^2} = \frac{\bar{\rho}(1 - s^2)}{s^2}$$

For our task, we can think of s as the ability to minimize the squared error. Strength is more generally the goodness of prediction or lack of bias.

In case of bias, the very random forest will have a higher bias as compared to the random forest. For our very random forest to have a low bias, based on the kinds of conditional expectations it can compute, we would have some properties to follow in the true distribution. Such a constraint is not necessarily present in the random forest model. A good model should have a low correlation between trees and a high strength of trees.

Since in our very random forest, based on the way we are creating one tree of only one feature, we may not have any dependence/correlation between the trees. A traditonal RF will have that correlation as compared to a VRF since it can use multiple features for one tree. The variance of the estimator is lower when the dependence of the trees is smaller. Thus the variance of a very random forest will be lower than a random forest.

3 Conditional Modelling with Gaussians

For any finite set of points $\mathcal{D} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, assume the following conditional model:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \Bigg| \mathbf{x}_1, ..., \mathbf{x}_N \sim \mathcal{N}(\vec{0}, K)$$

where K is the covariance matrix and $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2)$. You can assume that the covariance matrix K is positive-definite for any dataset $\mathcal{D}$.

(a) Write the down the distribution of $y_1, \cdots y_N, y_{N+1}, \mid \mathbf{x}_1, \cdots \mathbf{x}_N, \mathbf{x}_{N+1}$ under the assumed model.

Here, K_{N+1} is the (N+1)*(N+1) covariance matrix following from the above definition. .

$$\begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} \Bigg| \mathbf{x}_1, ..., \mathbf{x}_N \sim \mathcal{N}(\vec{0}, K_{N+1})$$

(b) Suppose we know $\mathbf{x}_{N+1}$ and are given a dataset $\{\mathbf{x}_1, y_1, \cdots, \mathbf{x}_N, y_N\}$ but not $\mathbf{y}_{N+1}$. How do you make a prediction using the assumed model?

From bayes rule,

$$p(y_{N+1}, ..., y_1/x_1, ..., x_{N+1}) = p(y_{N+1}/y_N, .., y_1, x_{N+1}, ..., x_1)p(y_N, ..., y_1/x_{N+1}, .., x_1) \ \cdots (1)$$

The second term on the right is given by the Gaussian in the question. The term on the left is given by part(a).

We can make a comparison with the following template form for the Gaussian,

$$p(x_a, x_b) = p(x_a/x_b)p(x_b) \ \cdots (2)$$

Where $p(x_a, x_b)$ and $p(x_b)$ are given Gaussians. We need to find $p(x_a/x_b)$.

For the joint $p(x_a, x_b)$ the covariance is written as,

$$\begin{bmatrix} K_{aa} & K_{ab} \\ K_{ba} & K_{bb} \end{bmatrix}$$

mean written as,

$$\begin{bmatrix} \mu_a \\ \mu_b \end{bmatrix}$$

We have,

$$p(x_a/x_b) = \mathcal{N}(x/u_{a|b}, K_{a|b})$$

Where,

$$\mu_{a|b} = \mu_a + K_{ab}{K_{bb}}^{-1}(x_b - \mu_b)$$

In our case, for equation (2), the covariance matrix of size (N+1)*(N+1) for term on the left has the following form,

$$\begin{bmatrix} 1 & c^T \\ c & K_N \end{bmatrix}$$

Here c is a size N vector such that,

$$c[i] = k(x_{N+1}, x_i)$$

The mean here is a N+1 sized zero vector as given.

For equation (2), the second term on the left has the covariance matrix K_N and N size zero vector as mean.

Now, we can compare equation (1) and equation (2). As a side note, everything in equation 1 is conditioned on the x's. But the comparision between y's in equation (1) and x_a, x_b in equation (2).

We get,

$$K_{bb} = K_N$$
$$K_{ab} = c^T$$
$$x_b = y$$

i.e. column vector of y values from 1 to N,

$$\mu_b = 0 \; vector$$

We get, for the distribution of,

$$p(y_{N+1}/y_N, .., y_1, x_{N+1}, ..., x_1)$$

The distribution is a Gaussian with mean and covariance as,

$$\mu = c^T {K_N}^{-1} y$$
$$\Sigma = 1 - c^T {K_N}^{-1} c$$

We predict this μ for y_{N+1} as based on the conditional expectation.

(c) What if the point $\mathbf{x}_{N+1}$ is far away from the training data, for instance $\min_{1 \le i \le N} \|\mathbf{x}_{N+1} - \mathbf{x}_i\| > 1000$, what is the prediction for $\mathbf{x}_{N+1}$?

From the previous question, our prediction for y_{N+1} is,

$$\mu = c^T {C_N}^{-1} y$$

The c vector here contains.

$$c[i] = k(x_{N+1}, x_i)$$

where, $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2} \|\mathbf{x}_i - \mathbf{x}_j\|^2)$

We obtain our prediction after doing a dot product of ${K_N}^{-1} y$ and c. c contains terms scales with the distance between x_{N+1} and x_i. As the distance between x_{N+1} and x_i increases, the value of $c[i]$ decreases exponentially. So the c vector will have very small magnitudes. Our predicted answer for y_{N+1} will be close to zero.

(d) Now, imagine you fit a flexible neural network f_θ on the dataset $\mathcal{D} = \{\mathbf{x}_1, y_1, \cdots \mathbf{x}_N, y_N\}$ to predict y from $\mathbf{x}$. Suppose again that $\mathbf{x}_{N+1}$ is far away from the training data as above. What can you say about the prediction $f_\theta(\mathbf{x}_{N+1})$?

The neural network models the conditional distribution of y given x parameterized by some weights. Here, in our case, the new x value is very far from the given x values. We can think of p(X) as tending to zero. We know that the conditional of Y given X is given by the joint of Y and X divided by the marginal of X.

The conditional distribution is not defined where p(X) is zero. The constraint here is that $p(X_{N+1})$ is well defined and not zero which may not be the case here. Our predictions will likely be not very useful and structured.

Hint: think about the constraints a neural network may impose on the prediction for $\mathbf{x}_{N+1}$ like the conditional model assumption above we made.

(e) Compare the predictions in part (c) and part (d). Are they the same? If not, explain the difference. If one has a problem, suggest a way to solve it.

For different values of x_{N+1} far away from the x values, we may not necessarily get values close to 0 as in the gaussian conditional model where our y_{N+1} predictions will always tend to 0.

For our flexible neural network, we could normalize the input in order to make it lie within a certain range and introduce regularization based on consistent priors for weights. Regularization will improve our generalization ability.

4 Gradient Mechanics in Neural Networks

Consider predicting $y \in [0, 1]$ from input $x \in [0, 1]$. We will build a neural network to do so using squared error loss. Our neural network will consist of a sequence of L *blocks*, each with scalar input, scalar output, and H hidden units. The block at each layer has two parameter vectors $w_\ell^{(1)}, w_\ell^{(2)} \in \mathbb{R}^H$ initialized in $[-\frac{1}{H}, \frac{1}{H}]$. Let's use the sigmoid non-linearity, which is applied element-wise to any vector. The neural network block at layer ℓ is defined as:

$$\begin{aligned} h_\ell^{(0)} &= \text{ scalar input} \\ h_\ell^{(1)} &= w_\ell^{(1)} \cdot h_\ell^{(0)} \\ h_\ell^{(2)} &= w_\ell^{(2)\top} \sigma(h_\ell^{(1)}) \\ z_\ell &= \sigma(h_\ell^{(2)}) \\ \text{scalar output} &= z_\ell \end{aligned}$$

By definition of $h_\ell^{(0)}$ and the weights, we can conclude that $h_\ell^{(1)} \in \mathbb{R}^H$ and that $h_\ell^{(2)}, z_\ell \in \mathbb{R},$. We connect the layers and use them on the data as follows: for datapoint (x, y), x is our first layer's input: $h_{\ell=1}^{(0)} = x$. Each next layer's input $h_{\ell+1}^{(0)}$ is equal to the previous layer's output z_ℓ. We will use our last layer's output z_L to predict y. The loss $L = (y - z_L)^2$.

1. compute the derivative of a single block's output with respect to its input $dz_\ell/dh_\ell^{(0)}$. Be sure to apply the chain rule at *each step*. Do not leave any symbolic expressions of the form $\frac{dy}{dx}$ (i.e. actually compute the derivatives of each function). **hint:** $\frac{d\sigma(x)}{dx} = \sigma(x) \cdot (1 - \sigma(x))$.
   Using the chain rule we get,

$$\frac{dz_\ell}{dh_\ell^{(0)}} = \sum_i \frac{dz_\ell}{dh_\ell^{(2)}} \frac{dh_\ell^{(2)}}{dh_\ell^{(1)}[i]} \frac{dh_\ell^{(1)}[i]}{dh_\ell^{(0)}}$$

   We will vectorize this appropriately,

$$\frac{dz_\ell}{dh_\ell^{(0)}} = \frac{dz_\ell}{dh_\ell^{(2)}} \cdot \frac{dh_\ell^{(2)}}{dh_\ell^{(1)}} \cdot \frac{dh_\ell^{(1)}}{dh_\ell^{(0)}}$$

   Therefore,

$$\frac{dz_\ell}{dh_\ell^{(0)}} = \sigma(h_\ell^{(2)})(1 - \sigma(h_\ell^{(2)})) \cdot w_\ell^{(2)T} \sigma(h_\ell^{(1)})(1 - \sigma(h_\ell^{(1)})) \cdot w_\ell^{(1)T} \sigma(h_\ell^{(0)})(1 - \sigma(h_\ell^{(0)}))$$

2. Compute the derivative $dL/dh_\ell^{(2)}$.
   note: your solution will necessarily involve a product over layers of some terms you have already derived. No need to re-derive within-layer gradients that you computed in the previous sub-problem.

$$\frac{dL}{dh_\ell^{(2)}} = \frac{dL}{dz_L} \cdot \frac{dz_L}{dh_L^{(0)}} \cdot \frac{dh_L^{(0)}}{dz_{L-1}} \cdot \frac{dz_{L-1}}{dh_{L-1}^{(0)}} \cdot \frac{dh_{L-1}^{(0)}}{dz_{L-2}} \cdots \frac{dz_\ell}{dh_\ell^{(2)}}$$

   This comes out to be,

$$\frac{dL}{dh_\ell^{(2)}} = \frac{dL}{dz_L} \cdot \frac{dz_L}{dh_L^{(0)}} \cdot \frac{dh_L^{(0)}}{dz_{L-1}} \cdot \frac{dz_{L-1}}{dh_{L-1}^{(0)}} \cdot \frac{dh_{L-1}^{(0)}}{dz_{L-2}} \cdots \frac{dz_\ell}{dh_\ell^{(2)}}$$

   Here for some layer k, $z_k = h^{(0)}{}_{k+1}$ The output of one layer is the input to the next layer.

3. compute derivative of $dL/dw_\ell^{(1)}$. Recall that the weights are initialized in $[-\frac{1}{H}, \frac{1}{H}]$. Is something bad happening here (especially at initialization time)?
note: same as previous note.

The derivative with respect to $w^(1)$ is given as,

$$\frac{dL}{dw_\ell^{(1)}} = \frac{dL}{dz_L} \cdot \frac{dz_L}{dh_L^{(0)}} \cdot \frac{dz_{L-1}}{dh_{L-1}^{(0)}} \cdots \cdot \frac{dh_\ell^{(2)}}{dh_\ell^{(1)}} \cdot \frac{dh_\ell^{(1)}}{dw_\ell^{(1)}}$$

$$\frac{dL}{dw_\ell^{(1)}} = \frac{dL}{dz_L} \cdot \frac{dz_L}{dh_L^{(0)}} \cdot \frac{dz_{L-1}}{dh_{L-1}^{(0)}} \cdots \cdot \frac{dh_\ell^{(2)}}{dh_\ell^{(1)}} \cdot h_\ell^{(0)}$$

The output of the sigmoid is between 0 and 1. In the forward pass, with each passing layer, the input will decease by a factor of H. For any input, if the magnitude of the layer output keeps shrinking with every layer, we will get small outputs for all inputs. In the backward pass, there may be a vanishing gradients problem.

4. What would happen if we added 100 to all of the $h_\ell^{(1)}$?

If we add 100 to all the hidden layers, the sigmoid output will be very close to 1. The corresponding gradient will be close to zero, there may be a vanishing gradients problem. If all intermediate outputs are close to 1, we are not modelling well.

5. Setup a network that doesn't have the issues above, if any. You can add terms or change non-linearities in the block as long as it is still scalar-in scalar-out.

In terms of non linearities, we could use ReLU which is empirically effective. The initialization of the weights should be the He initialization.

6. What are the takeaways for building neural nets on real data?

We should use best empirical practices while building neural networks on real data. We should try and ascertain that our model does not have a vanishing and an exploding gradients problem if possible. We should perform data pre processing and feature engineering. We should use regularization techniques as appropriate. For optimization, momentum based methods could be preferred over vanilla gradient descent. We could do a neural architecture search. We should follow the literature for best practices in a particular domain.