DS-GA 1003 Machine Learning: Homework 1

Due 11.59 p.m. EST, February 27, 2024 on Gradescope

Jiasheng Ni

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

1 Linear Regression Model

Consider the data generating process as such: $\mathbf{x} \in \mathbb{R}^D$ is drawn from some unknown $p(\mathbf{x})$ and $y = w_1^{true} x_1 + \epsilon_y$, where $w_1^{true} \in \mathbb{R}$ and $\epsilon_y \sim \mathcal{N}(0, 1)$. This is unknown to us, as a result, we construct a linear model for y using all D features of $\mathbf{x}$, instead of just using x_1.

(A) Explain what the terms **model class** and **model misspecification** mean. Is our model correctly *specified* here? Why or why not?

Solution. The model class is linear regression model and is correctly chosen since it captures the real data generation process(which is also a linear regression model).
However the model with w_1^{all} contains some model misspecification. Also it is a linear model, we choose all D features of the data to fit the model, which is different from data generation process, which only uses the first dimension of the feature. This could cause multicollinearity problem as we will explore below. □

Let $\widehat{w_1}$ be the estimate of w_1^{true} using only x_1 and let $\widehat{w_1^{all}}$ be the estimate of w_1^{true} when using all of $\mathbf{x}$. We will study the effects of our model by analyzing the relationships between $\mathbb{E}[\widehat{w_1^{all}}]$ and $\mathbb{E}[\widehat{w_1}]$, as well as between $\text{Var}[\widehat{w_1^{all}}]$ and $\text{Var}[\widehat{w_1}]$. We do so empirically by running PyTorch simulations as follows:

1. Pick any value of w_1^{true} you like as ground truth, e.g. with `torch.randn(1)`.
2. Write a function, taking D and c as input, that does the following: **(1)** Generate $N = 50$ samples of $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \Sigma)$, where Σ is the $D \times D$ covariance matrix with all diagonal entries equal to $\sigma^2 = 1$ and all off-diagonal entries equal to c. **(2)** Compute y using the relationship above (note that y only depends on the first feature). This involves drawing N samples of noise $\epsilon_y \sim \mathcal{N}(0, 1)$. **(3)** Using our dataset of N samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, compute the least-squares solutions for $\widehat{w_1^{all}}$ (i.e. using all features and taking the first coefficient) and $\widehat{w_1}$ (i.e. using only one feature).
3. Write a function that performs Step 2 for $T = 100$ trials, i.e. each trial generates a new dataset to compute $\widehat{w_1^{all}}$ and $\widehat{w_1}$. (Note that w_1^{true} is constant throughout.) For each estimator, compute the mean and standard deviation of the T trials.
4. Perform Step 3 for each $c \in \{0.1, 0.2, 0.3, \ldots, 0.9\}$ and each $D \in \{2, 4, 8, 16, 32\}$. Separately plot the means and standard deviations as a function of c, using the same plot for both estimators. This means you should have 10 plots altogether: two plots (means and standard deviations) for each of the five choices of D. Each plot will contain two curves (the two estimators).

(B) What do you observe with respect to c and D? How do you explain your results? Show a few (not all) of your 10 generated plots to support your answer.

Solution. The observations are as follows:

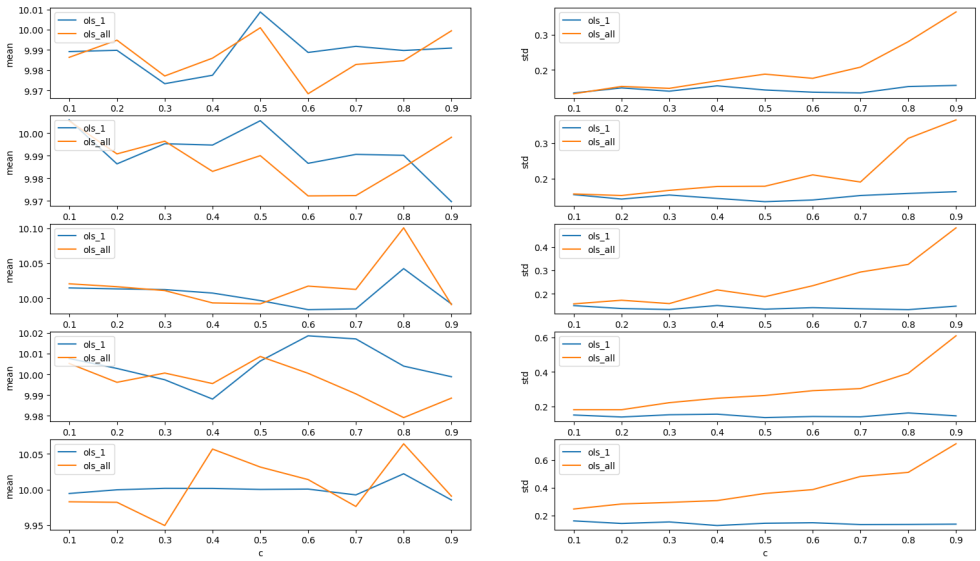


Figure 1: Grid Look

(a) Observation 1: As c increases(feature correlation), we have higher variance in $\hat{w}_1^{all}$ and stable variance in $\hat{w}_1$.

(b) Observation 2: As c increases(feature correlation), we have both $\hat{w}_1^{all}$ and $\hat{w}_1$ around w_{true}.

(c) Observation 3: As D increases, we don't observe any strange behaviors across c.

Then we give a detailed proof for each of these observations.

Claim 1: The $\hat{w}_1$ is both unbiased and consistent estimator for w_{true}

Proof. For $\widehat{\omega}_1$, our linear model is defined as:

$$y_i = \omega_1 x_i + \varepsilon_y$$

where $\varepsilon_y \sim N(0,1)$ The OLS estimator given dataset $\{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, is:

$$\hat{\omega}_1 = \frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2}$$

We first show that $\hat{\omega}_1 \xrightarrow{p} \omega_1^{\text{true}}$ Since $Y = \omega_1^{\text{true}} X + \varepsilon_y$, we have:

$$\begin{aligned}
&\text{cov}(X,Y) = E[XY] - E[X]E[Y] \\
&= E\left[W_1^{\text{true}} X^2 + X\varepsilon_y\right] - 0 \\
&= \omega_1^{\text{true}} E\left[X^2\right] + E\left[X\varepsilon_y\right]. \\
&= \omega_1^{\text{true}} \\
&\text{Var}[X] = 1 \\
&\frac{1}{N-1}\sum_{i=1}^{N} x_i y_i \xrightarrow{P} \text{Cov}(X,Y) \\
&\frac{1}{N-1}\sum_{i=1}^{N} x_i^2 \xrightarrow{P} Var(X)
\end{aligned}$$

by WLLN.
$\therefore$ By continuous mapping theorem.

$$\frac{\sum_{i=1}^{N} x_i y_i}{\sum_{i=1}^{N} x_i^2} \xrightarrow{P} \frac{\text{Cov}(X,Y)}{\text{Var}(x)} = \omega_1^{\text{true}}$$

We then show $E[\hat{w}_1] = \omega_1^{\text{true.}}$

$$\begin{aligned}
\because E[\hat{\omega}_1] &= \frac{E\left[\sum_{i=1}^{N} x_i \left(\omega_1^{\text{true}} x_i + \varepsilon_y\right)\right]}{\sum_{i=1}^{N} x_i^2} \\
&= \omega_1^{\text{true}} + \frac{E\left[\sum_{i=1}^{N} x_i \varepsilon_y\right]}{\sum_{i=1}^{N} x_i^2} \\
&= \omega_1^{\text{true}}
\end{aligned}$$

□

Claim 2: The $\widehat{w_1^{all}}$ is unbiased but not a consistent estimator for w_1^{true}

Proof. For $\widehat{\omega_1^{all}}$, we defined our linear model as:

$$\vec{y} = X\vec{w} + \vec{\varepsilon}$$

where. $\widehat{w_1^{all}} = \vec{w_1^*}, \vec{\varepsilon} \sim N(0, I_N)$
The OLS estimator is given by:

$$\vec{w^*} = \left(X^\top X\right)^{-1} X^\top \vec{y}$$

We show that $E[\vec{w^*}] = \vec{w}^{\text{true}}$

$$\begin{aligned}
&\because E[\vec{w^*}] = E\left(\left(X^\top X\right)^{-1} X^\top \left(X\vec{w}^{\text{true}} + \vec{\varepsilon}\right)\right] \\
&= \left(X^\top X\right)^{-1}\left(X^\top X\right)\vec{w}^{\text{true}} + E\left[\left(X^\top X\right)^{-1} X^\top \vec{\varepsilon}\right] \\
&= \vec{\omega}^{\text{true}} + \vec{0} \\
&= \vec{\omega}^{\text{true}} \\
&\therefore E[\vec{w_i^*}] = \omega_i^{\text{true}} \; \forall i = 1, 2, \cdots, D
\end{aligned}$$

We then show $\text{Var}[\vec{w^*}] = \left(X^\top X\right)^{-1}$ Since $\text{Var}(\vec{w^*}) = E\left[\left(\vec{w^*} - \vec{\omega}^{\text{true}}\right)\left(\vec{w^*} - \vec{\omega}^{\text{true}}\right)^\top\right]$

$$\begin{aligned}
&= E\left[\left(X^\top X\right)^{-1} X^\top \vec{\varepsilon}\vec{\varepsilon}^\top X \left(x^\top X\right)^{-1}\right] \\
&= \left(X^\top X\right)^{-1} X^\top E\left[\vec{\varepsilon}\vec{\varepsilon}^\top\right] X \left(X^\top X\right)^{-1} \\
&= \left(X^\top X\right)^{-1} X^\top I_N X \left(X^\top x\right)^{-1} \\
&= \left(X^\top X\right)^{-1}
\end{aligned}$$

□

$\therefore$ When c is large, there's strong or perfect multicollineanity in data matrix X, which means if $\vec{v}$ is an eigenvector of $X^\top X$ then $X^\top X\vec{v} = X^\top(\overrightarrow{0}) = \overrightarrow{0}$. since some columns can be linear combination of other columns.

$\therefore$ We could have small eigenvalues, close to zero. Since $\lambda_i\left(X^\top X\right) = \lambda_i\left(\left(X^\top X\right)^{-1}\right)^{-1}$

$\therefore$ We expect large eigenvalues from $\left(X^\top X\right)^{-1}$, which corresponds to the variance of the entries of $\hat{\vec{w}}^{\text{all}}$, thus we observe large variance of w_1^{all} at larger C values. $\square$

2 Bayesian Linear Regression Model

Consider the data generating process as such: $x \sim \mathcal{N}(0,1)$ and $y = w_{true}x^2 + \epsilon$, where $w_{true} = 1.0$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, and $\sigma^2 = 1.0$. Again, this is unknown to us. We will model the data using Bayesian linear regression.

(A) Using PyTorch, simulate a dataset $\mathcal{D}_N = \{(x_i, y_i)\}_{i=1}^N$ for each $N \in \{10, 100, 1000, 10000\}$ according to the true data generating process above. For our Bayesian linear regression model, let us choose our prior as $w \sim N(0,1)$ and our likelihood as $y|w, x \sim N(wx, \sigma^2)$ for $\sigma^2 = 1.0$. Compute the mean and variance of the posterior $w|\mathcal{D}_N$ for each dataset. Does the posterior concentrate on w_{true}? Why or why not?

Solution. Suppose we have dataset $D_N = \{(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)\}$, where we assume a $1-d$ model $y_i = \omega x_i + \varepsilon_i$ where $\varepsilon_i \sim N\left(0, \sigma^2\right), \omega \sim N\left(\mu_0, \sigma_0^2\right)$

Then the posterior distribution:

$$
\begin{aligned}
f\left(\omega \mid D_N\right) &= f\left(\omega \mid x_1, x_2, \cdots, x_N, y_1, y_2, \cdots, y_n, \sigma^2\right) \\
&= \frac{f\left(y_1, y_2, \cdots, y_N \mid \omega, x_1, x_2, \cdots, x_N, \sigma^2\right) f(\omega)}{f\left(y_1, y_2, \cdots y_N \mid x_1, x_2, \cdots x_N\right)} \\
&\propto \prod_{i=1}^{N} \exp\left\{-\frac{1}{2\sigma^2}\left(y_i - \omega x_i\right)^2\right\} \exp\left\{-\frac{1}{2\sigma_0^2}\left(\omega - \mu_0\right)^2\right\} \\
&\propto -\frac{\sigma_0^2 \sum_{i=1}^N \left(y_i - \omega x_i\right)^2 + \sigma^2\left(\omega - \mu_0\right)^2}{2\sigma^2\sigma_0^2} \\
&= -\frac{\left(\sum_{i=1}^N x_i^2 \cdot \omega^2 - 2\sum_{i=1}^N x_i y_i \cdot \omega + \sum_{i=1}^n y_i^2\right)\sigma_0^2 + \left(\omega^2 - 2\mu_0 \cdot \omega + \mu_0^2\right) \cdot \sigma^2}{2\sigma^2\sigma_0^2} \\
&= -\frac{\left(\sigma_0^2 \sum_{i=1}^N x_i^2 + \sigma^2\right)\omega^2 - 2\left(\sigma_0^2 \sum_{i=1}^N x_i y_i + \mu_0 \cdot \sigma^2\right)\omega + C}{2\sigma^2\sigma_0^2} \\
\therefore \mu_{\text{post}} &= \frac{\sigma_0^2 \sum_{i=1}^N x_i y_i + \mu_0\sigma^2}{\sigma_0^2 \sum_{i=1}^N x_i^2 + \sigma^2}, \sigma^2_{\text{post}} = \frac{\sigma^2\sigma_0^2}{\sigma_0^2 \sum_{i=1}^N x_i^2 + \sigma^2}
\end{aligned}
$$

, where in this question we have $\sigma_0^2 = \sigma_1^2 = 1, \mu_0 = 0$, and that as we notice from the graph $\mu_{\text{post}} = \frac{\sum_{i=1}^N x_i y_i}{\sum_{i=1}^N x_i^2 + 1}$, $\sigma^2_{\text{post}} = \frac{1}{\sum_{i=1}^N x_i^2 + 1}$. As shown in the graph:

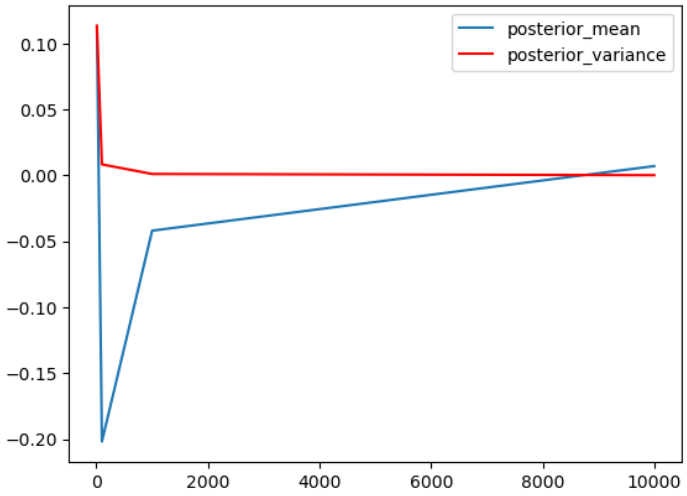


Figure 2: Main name 2

the posterior variance is decreasing as we increasing the number of data points, this is the same to increase the denominator of posterior variance. The posterior mean will not concentrate around $w_t rue$ since $x \sim \mathcal{N}(0,1)$, which makes the $\sum_{i=1}^{n} x_i y_i$ term centered around 0 when $n \to \infty$. □

(B) What would be challenging about our analysis in part (A) if we had picked a different prior, for example, Laplace or Gamma?

Solution. If we pick different prior, it may be difficult to compute the posterior. For example, for Gamma prior, it is not a conjugate prior for gaussian likelihood(but is for possion likelihood), so the posterior is not Gamma. The same happens for Laplacian prior, the posterior is not gaussian either. But if we use MAP estimate, then even if we don't know the exact posterior distribution, we may still get our estimation $\hat{w}$. □

(C) Repeat part (A), except we use the basis set $\boldsymbol{\phi}(x) = [x, x^2]$ (instead of x itself) and perform 2D Bayesian linear regression. We choose our prior to be $\mathbf{w} \sim N(\mathbf{0}, \mathbf{I})$, where $\mathbf{I}$ is the 2×2 identity matrix, and our likelihood to be $y|\mathbf{w}, x \sim N(\mathbf{w}^\top \boldsymbol{\phi}(x), \sigma^2)$ for $\sigma^2 = 1.0$. Compute the mean and variance of the posterior $\mathbf{w}|\mathcal{D}_N$ for each dataset. What do you observe about the posterior as N changes? Why?

Solution. The statistics are summarized in the following table:

N	mean	covariance matrix
10	[0.2046, 0.3908]	$\begin{bmatrix} 0.0749 & -0.0020 \\ -0.0020 & 0.0330 \end{bmatrix}$
100	[-0.0693, 0.2350]	$\begin{bmatrix} 0.0079 & -0.0004 \\ -0.0004 & 0.0021 \end{bmatrix}$
1000	[0.0149, 0.3488]	$\begin{bmatrix} 1.0616 \times 10^{-3} & 5.5125 \times 10^{-6} \\ 5.5125 \times 10^{-6} & 4.1348 \times 10^{-4} \end{bmatrix}$
10000	[0.0149, 0.3488]	$\begin{bmatrix} 9.9178 \times 10^{-5} & -3.6099 \times 10^{-7} \\ -3.6099 \times 10^{-7} & 3.2670 \times 10^{-5} \end{bmatrix}$

We notice that as $N \to \infty$, the variance of the posterior distribution converges to zero and the mean of the posterior distribution could be anywhere, but generally, it is the weighted average between the likelihood mean and the prior mean. □

(D) Reflecting on your answers in parts (A) and (C), name one challenge that **cannot be solved** by using a Bayesian model (instead of a frequentist approach like standard linear regression).

Solution. One challenge is the choice of prior distribution. If our prior variance is too small and the prior mean is too far away from the true parameter, then small prior variance causes the posterior distribution to put more weight on the prior during each bayesian updating iteration. Moreover, if we choose a bad prior that has no conjugacy with its likelihood, the posterior distribution may be hard to compute. □

(E) Reflecting on your answers in part (C) and in Question 1, name one challenge that **can be improved** by using a Bayesian model.

Solution. One improvement with bayesian model is that our posterior variance for our estimator will always be smaller for each iteration of bayesian updating. In the case of multicollinearity features, unregularized linear regression model like in problem 1 will generally overfit to the dataset and have larger variance in the estimator while for bayesian model, adding prior is the same to adding the regularization terms to the parameter, making the variance of the estimator within bounds.

For example, OLS(MLE) delivers $(X^\top X)^{-1}X^\top \vec{y}$ while MAP delivers $\vec{\mu}_W + (X^\top X + \Sigma_W^{-1})^{-1}X^\top(\vec{y} - X\vec{\mu}_W)$ where $\vec{\omega} \sim \mathcal{N}(\mu_W, \Sigma_W)$(prior distribution). We could easily see that the MAP results gurantee the invertibility of $X^\top X$ while fall victims to the choice of prior distribution where $\vec{\mu}_W$ and Σ_W could greatly impact the location of MAP estimate. □

Hint: Both part (C) above and Question 1 involve doing linear regression with many correlated features. How do these two sets of findings relate? Does using a Bayesian approach affect the way the model treats correlated features?

3 Class-Conditional Gaussian Generative Model

Consider a classification task where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \{1, \ldots, K\}$. We observe the dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Let us construct a model for the joint distribution as

$$p_\theta(\mathbf{x}, y) = p_\theta(\mathbf{x}|y)p_\theta(y)$$

where θ denotes the set of all parameters of the model.

(A) Our model is known as a **class-conditional generative model**. What about the model makes it generative? What makes it class-conditional?

Solution. We break each term as follows:

(a) Class-Conditional: This is because during the learning process, if we are giving the dataset $\mathcal{D}$, we can use MLE to fit a gaussian distribution with

$$\Sigma_k = \frac{1}{n_k} \sum_{i=1}^{n_k} (\vec{x}_i^{[k]} - \vec{\mu}_k)(\vec{x}_i^{[k]} - \vec{\mu}_k)^\top$$

$$\vec{\mu}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \vec{x}_i^{[k]}$$

for each class k such that we have a class-conditional probability distribution for the features $f(\vec{x}|y = k) \sim \mathcal{N}(\vec{\mu}_k, \Sigma_k)$

(b) Generative: This is because given a class, we can generate(sample) new data points according to this class-conditional distribution.

□

(B) For a given value of θ, how would you predict the label for a new test point $\mathbf{x}_\star$ using your model $p_\theta(\mathbf{x}, y)$?

Solution. Basically we will use the idea of MAP as follows:

$$\begin{aligned} y_* &= \text{argmax}_{\text{y}\in\{1,2,\cdots,\text{k}\}}\, P\left(y \mid \vec{x}_*\right) \\ &= \text{argmax}_{y\in\{1,2,\cdots k\}}\, \frac{f\left(\vec{x}_* \mid y\right) P_\theta(y).}{\sum_{y\in\{1,\cdots,k\}} f\left(\vec{x}_* \mid y\right) P_\theta(y)} \end{aligned}$$

where $f\left(\vec{x}_* \mid y\right) \sim N\left(\vec{\mu}_y, \Sigma_y\right)$. □

Let us model y as a Categorical distribution Cat($\boldsymbol{\pi}$). Here $p_\theta(y = k) \triangleq \pi_k$, where $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]$ such that $\forall k$, $\pi_k \geq 0$ and $\sum_k \pi_k = 1$. You may leave $p_\theta(\mathbf{x}|y)$ unspecified for now.

(C) Write down an expression for the log-likelihood of the observed dataset $\mathcal{D}$.

Solution.

$$\mathcal{L}(\vec{\pi}) = \prod_{i=1}^{N} \prod_{j=1}^{K} \pi_j^{\mathbf{1}\{y_i = j\}}$$

$$l(\vec{\pi}) = \sum_{i=1}^{N} \sum_{j=1}^{K} \mathbf{1}\left\{y_i = j\right\} \log \pi_j$$

where $\sum_{j=1}^{K} \pi_j = 1$ □

(D) Derive an expression for the maximum likelihood estimator (MLE) for $\boldsymbol{\pi}$, which we will denote as $\widehat{\boldsymbol{\pi}}$. Make sure to account for the constraints on $\boldsymbol{\pi}$ using Lagrange multipliers.

Solution. The optimization problem becomes.

$$L(\vec{\pi}, \mu) = -\sum_{i=1}^{N}\sum_{j=1}^{K} \mathbf{1}\{y_i = j\} \log \pi_j + \mu\left(\sum_{j=1}^{K} \pi_j - 1\right)$$

And we want:

$$\min_{\vec{\pi}} L(\vec{\pi}, \mu)$$

Now we prove that $\mathcal{L}(\vec{\pi}, \mu)$ is convex in $\vec{\pi}$.

$$\because \log(\pi_j) = \log\left(\vec{a}_j^\top \vec{\pi}\right) \text{ where } \vec{a}_j = [0, 0, \cdots 1, 0, 0]$$

$-\log\left(\vec{a}_j^\top \vec{\pi}\right)$ is convex in $\vec{\pi}$ since $-\log x$ is convex in x and $-\log\left(\vec{a}_j^T \vec{\pi}\right)$ is a convex composition of a linear mapping.
$\because -\sum_{i=1}^{N}\sum_{j=1}^{k} \mathbf{1}\{y_i = j\} \log \pi_j$ is nonnegative sum of convex functions and thus is convex.
$\because \mu\left(\sum_{j=1}^{K} \pi_j - 1\right)$ is convex in $\vec{\pi}$ for similarly reason.
$\therefore L(\vec{\pi}, \mu)$ is convex in $\vec{\pi}$
Thus we enforce KKT conditions on $L(\vec{\pi}, \mu)$,which guarantees optimality under convexity:

$$\frac{\partial \mathcal{L}(\vec{\pi}, \mu)}{\partial \pi_i} = -\sum_{i=1}^{N} \frac{\mathbf{1}\{y_i = j\}}{\pi_j} + \mu = 0 \qquad \forall j = 1, 2, \cdots, K.$$

$$\sum_{i=1}^{K} \pi_i = 1$$

Solving for the KKT we get:

$$\therefore \widehat{\pi_j} = \frac{\sum_{i=1}^{N} \mathbf{1}\{y_i = j\}}{\mu} \quad \forall j = 1, 2, \cdots, K.$$

We now solve for μ:

$$\because \sum_{j=1}^{K} \hat{\pi}_j = 1$$

$$\therefore \sum_{j=1}^{K}\sum_{i=1}^{N} \frac{\mathbf{1}\{y_i = j\}}{\mu} = 1$$

$$\therefore \mu = \sum_{j=1}^{K}\sum_{i=1}^{N} \mathbf{1}\{y_i = j\} = N \text{ which is a constant}$$

□

Let us further model $\mathbf{x}|y$ as (multivariate) Gaussian distributions $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \Sigma_k)$ for all K classes, where $\boldsymbol{\mu}_k \in \mathbb{R}^D$ and Σ_k is a $D \times D$ (positive semi-definite) covariance matrix. Assume that there are only $K = 2$ classes. This means that the total set of parameters are $\theta = \{\boldsymbol{\pi}, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \Sigma_1, \Sigma_2\}$.

Now, consider the case where the data comes from this model. That is, $y \sim \text{Cat}(\boldsymbol{\pi}^{true})$ and $\mathbf{x}|y = k \sim \mathcal{N}(\boldsymbol{\mu}_k^{true}, \Sigma_k^{true})$ for all k. After we observe this data, we can then construct a *discriminative model* to predict y from $\mathbf{x}$, that is, we will learn a model for $p_{true}(y = k|\mathbf{x})$. Let us do so using **logistic regression**:

$$y|\mathbf{x} \sim \text{Bernoulli}(\sigma(\mathbf{w}^\top \mathbf{x}))$$

where $\mathbf{w}$ are the parameters of the model and $\sigma(z)$ is the logistic sigmoid:

$$\sigma(z) = \frac{1}{1 + \exp[-z]}$$

(E) Will logistic regression always be able to model the true data conditional $p_{true}(y = k|\mathbf{x})$? If so, why? If sometimes, when? And if there are any cases where logistic regression will not be able to model $p_{true}(y = k|\mathbf{x})$, are there any ways to fix it?

Solution. The answer is not always. Let's say if logistic regression perfectly model the true data conditional $p_{\text{true}}\ (y = k \mid \mathbf{x})$, then since $K = 2$, we will have $\frac{p_{\text{true}}\ (y=1|\mathbf{x})}{p_{\text{true}}\ (y=0|\mathbf{x})} = \frac{1}{e^{-\mathbf{w}^\top \mathbf{x}}}$ and the decision boundary requires $\frac{1}{e^{-\mathbf{w}^\top \mathbf{x}}} = 1$ where $e^{\mathbf{w}^\top \mathbf{x}} = 1$ and $\mathbf{w}^\top \mathbf{x} = 0$, which is a linear boundary. But when $\Sigma_1 \neq \Sigma_2$, namely the covariance matrix from different class is different, the true decision boundary is quadratic instead of linear. The proof is as follows, we know in GDA, given test data point $\mathbf{x}_*$, our prediction for $\hat{\mathbf{y}}_*$ is calculated by MAP rules:

$$\begin{aligned}
\text{MAP}(\mathbf{x}_*) &:= \arg \max_{y \in \{1,2,\ldots,K\}} p_{\tilde{y}|\tilde{x}}(y \mid \mathbf{x}_*) \\
&= \arg \max_{y \in \{1,2,\ldots,K\}} \frac{p_{\tilde{y}}(y) f_{\tilde{x}|\tilde{y}}(\mathbf{x}_* \mid y)}{\sum_{k \in \{1,2,\ldots,k\}} p_{\tilde{y}}(k) p_{\tilde{x}|\tilde{y}}(\mathbf{x}_* \mid k)} \\
&= \arg \max_{y \in \{1,2,\ldots,K\}} \frac{\frac{\alpha_y}{\sqrt{(2\pi)^d |\Sigma_y|}} \exp\left(-\frac{1}{2}(\mathbf{x}_* - \mu_y)^T \Sigma_y^{-1} (\mathbf{x}_* - \mu_y)\right)}{\sum_{k \in \{1,2,\ldots,K\}} \frac{\alpha_k}{\sqrt{(2\pi)^d |\Sigma_k|}} \exp\left(-\frac{1}{2}(\mathbf{x}_* - \mu_k)^T \Sigma_k^{-1} (\mathbf{x}_* - \mu_k)\right)}
\end{aligned} \tag{1}$$

and the decision boundary is determined by:

$$\frac{p_{\tilde{y}|\tilde{x}}(y = 1 \mid \mathbf{x}_*)}{p_{\tilde{y}|\tilde{x}}(y = 2 \mid \mathbf{x}_*)} = 1$$

using (1) we get:

$$\frac{\alpha_1 \sqrt{|\Sigma_2|}}{\alpha_2 \sqrt{|\Sigma_1|}} \exp\left(\frac{1}{2}(\mathbf{x}_* - \mu_2)^T \Sigma_2^{-1} (\mathbf{x}_* - \mu_2) - \frac{1}{2}(\mathbf{x}_* - \mu_1)^T \Sigma_1^{-1} (\mathbf{x}_* - \mu_1)\right) = 1$$

Then we take logarithms to get the decision boundary,

$$\frac{1}{2}(\mathbf{x}_* - \mu_2)^T \Sigma_2^{-1} (\mathbf{x}_* - \mu_2) - \frac{1}{2}(\mathbf{x}_* - \mu_1)^T \Sigma_1^{-1} (\mathbf{x}_* - \mu_1) + \log\left(\frac{\alpha_1 \sqrt{|\Sigma_2|}}{\alpha_2 \sqrt{|\Sigma_1|}}\right) = 0.$$

where if $\Sigma_1 = \Sigma_2$, we can cancel out the quadratic terms and obtain a linear boundary. Otherwise, the decision bounary is quadratic and the logistic regression fails to approximate the true distribution well. □

4 Poisson Generalized Linear Model

Consider a classification task where $\mathbf{x} \in \mathbb{R}^D$ and $y \in \mathbb{N} = \{0, 1, 2, 3, \dots\}$, noting that the support of y is the unbounded set of natural numbers. We have an observed dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Let us also assume that the number of features, D, is larger than the number of examples, N. We will model this data using a Poisson Generalized Linear Model (GLM). Let $\boldsymbol{\theta}$ denote the linear coefficients of the model.

(A) Write down the log-likelihood function of the Poisson GLM.

Solution. Suppose we have dataset $D_N = \{(\vec{x}_1, y_1), (\vec{x}_2, y_2), \cdots, (\vec{x}_N, y_N)\}$ where $\vec{x}_i \in R^D$ the poisson model assumes, $f(y_i; \vec{x}_i) \overset{\text{ind}}{\sim} \text{poisson}(\lambda_i)$

$$\begin{aligned}
\therefore f(y_i; \vec{x}_i) &= \frac{1}{y_i!} e^{-\lambda} \lambda^{y_i} \\
&= e^{y_i \cdot \log \lambda - \lambda} \cdot \frac{1}{y_i!} \\
&= e^{y_i \cdot \vec{x}_i^\top \vec{\theta} - e^{\vec{x}_i^\top \vec{\theta}}} f_0(y_i)
\end{aligned}$$

where the link function is $g(\lambda) = \log \lambda$
The log-likelihood function is:

$$\begin{aligned}
l(\vec{\theta}) &= \log \prod_{i=1}^N f(y_i; \vec{x}_i) \\
&= \log \exp \left\{ \sum_{i=1}^N y_i \cdot \left(\vec{x}_i^\top \vec{\theta} \right) - e^{\vec{x}_i^\top \vec{\theta}} \right\} + \log f_0(y_i) \\
&= \sum_{i=1}^N y_i \left(\vec{x}_i^\top \vec{\theta} \right) - e^{\vec{x}_i^\top \vec{\theta}}
\end{aligned}$$

□

(B) Given a test point $\mathbf{x}_\star$ and some estimate $\hat{\boldsymbol{\theta}}$ of the parameter, how do you make a prediction $\hat{y}_\star$?

Solution. Given the link function $g(\lambda) = \log \lambda$, we know the prediction model:

$$\begin{aligned}
\log \hat{y}_* &= \mathbf{x}_\star^\top \hat{\boldsymbol{\theta}} \\
\hat{y}_* &= e^{\mathbf{x}_\star^\top \hat{\boldsymbol{\theta}}}
\end{aligned}$$

since we are basically predicting $\mathbb{E}[Y|X]$, the conditional mean of the distribution. □

(C) Now suppose that the parameter $\hat{\boldsymbol{\theta}}$ of the Poisson GLM is estimated using ℓ_2-regularized maximum likelihood estimation. If the test point $\mathbf{x}_\star$ is *orthogonal* to the subspace generated by the training data, what is the distribution $\hat{y}_\star | \mathbf{x}_\star$ predicted by the Poisson GLM model? Prove your answer.

Solution. Using l_2 is the same as assuming the prior of $\vec{\theta}$ to be $N\left(\overrightarrow{0}, \vec{v}^2 I_0\right)$

$$\begin{aligned}
MLE(\vec{\theta}) &\propto \sum_{i=1}^N y_i \left(\vec{x}_i^\top \vec{\theta} \right) - e^{\vec{x}_i^\top \vec{\theta}} \\
Prior(\vec{\theta}) &= \frac{1}{\sqrt{2\pi}\sigma} \exp \left\{ -\frac{1}{2\sigma^2} \vec{\theta}^\top \vec{\theta} \right\} \\
log f_{\vec{\theta}|X,\vec{y}}(\vec{\theta}) &\propto MLE(\vec{\theta}) \cdot Prior(\vec{\theta})
\end{aligned}$$

Thus we have l2-regularized MLE as:

$$l(\vec{\theta}) = \left\{ \sum_{i=1}^{N} y_i \left(\vec{x}_i^\top \vec{\theta} \right) - e^{\vec{x}_i^\top \vec{\theta}} \right\} - \frac{1}{2\sigma^2} \vec{\theta}^\top \vec{\theta}$$

and that:

$$\widehat{\vec{\theta}} = \underset{\vec{\theta}}{\text{argmax}} l(\vec{\theta})$$

We notice that the l_2-regularized max-likelihuod

$$\begin{aligned} \nabla^2_{\vec{\theta}} l(\vec{\theta}) &= -\sum_{i=1}^{N} \vec{x}_i \vec{x}_i^\top e^{\vec{x}_i^\top \vec{\theta}} - \frac{1}{\sigma^2} \\ &= -\left(e^{\vec{x}_i^\top \vec{\theta}} X^\top X + \frac{1}{\sigma^2} I_D \right) \leq 0 \end{aligned}$$

$\therefore l(\vec{\theta})$ is concave in $\vec{\theta}$, thus by FOC we have:

$$\therefore \nabla l(\vec{\theta}) = X^\top \vec{y} - \sum_{i=1}^{N} \vec{x}_i e^{\vec{x}_i^\top \widehat{\vec{\theta}}} - \frac{1}{\sigma^2} \widehat{\vec{\theta}} = 0$$

Solving for the $\widehat{\vec{\theta}}$ we get:

$$\widehat{\vec{\theta}} = \sigma^2 (X^\top \vec{y} - \sum_{i=1}^{N} \vec{x}_i e^{\vec{x}_i^\top \widehat{\vec{\theta}}})$$

Since by orthogonality assumption we have:

$$\begin{aligned} &\because \vec{x}_* \perp \vec{x}_i \quad \forall i = 1, 2, \cdots, N \\ &\therefore \vec{x}_*^\top \vec{x}_i = 0 \quad \forall i = 1, 2, \cdots, N \\ &\therefore \hat{y}_* = e^{\vec{x}_*^\top \cdot \sigma^2 \left(X^\top \vec{y} - \sum_{i=1}^{N} \vec{x}_i e^{\vec{x}_i^\top \hat{\theta}} \right)} \\ &= e^{\sigma^2 (0-0)} \\ &= 1 \end{aligned}$$

$\therefore$ The parameter λ we are estimating is 1.
$\therefore \widehat{y}_* \mid \vec{x}_*$ by our Poisson GLM is just a passion distribution with $\lambda = 1$ and the distribution is:

$$f(\hat{y}_*, \vec{x}_*) = e^{-1} = \frac{1^{\hat{y}_*} e^{-1}}{\hat{y}_*!}$$

□

(D) From your answer to part (C), motivate ℓ_1-regularization when the number of features, D, is larger than the number of examples, N.

Solution. When the number of feature $D > N$, there will be multi-colinearity problem in our dataset, even if it doesn't affect our solution in l2-regularization, the interpretability of our coefficients $\vec{\theta}_i$ will be ruined since Poisson GLM in essense is still a linear model. The $\vec{\theta}_i$ may absorb some weights from $\vec{\theta}_j$ due to the multi-colinearity in between. In contrast, for l1-regularization, some coefficients $\vec{\theta}_i$ will be set to 0(robustness), and we only have to interpret those features with non-zero coefficients, which typically won't absorb other coefficients. Due to time limit, we don't provide a formal proof, which may involve advanced techniques like subgradient method. □

5 Distances and Optimization Directions

Consider two pairs of distributions with mean and variance parameterization:

Pair 1: Normal(0, 0.0001), Normal(0.1, 0.0001)

Pair 2: Normal(0, 1000), Normal(0.1, 1000)

(A) Make two plots where each plot shows the pdfs for the distributions in the pair.

Solution. The distributions are shown below:

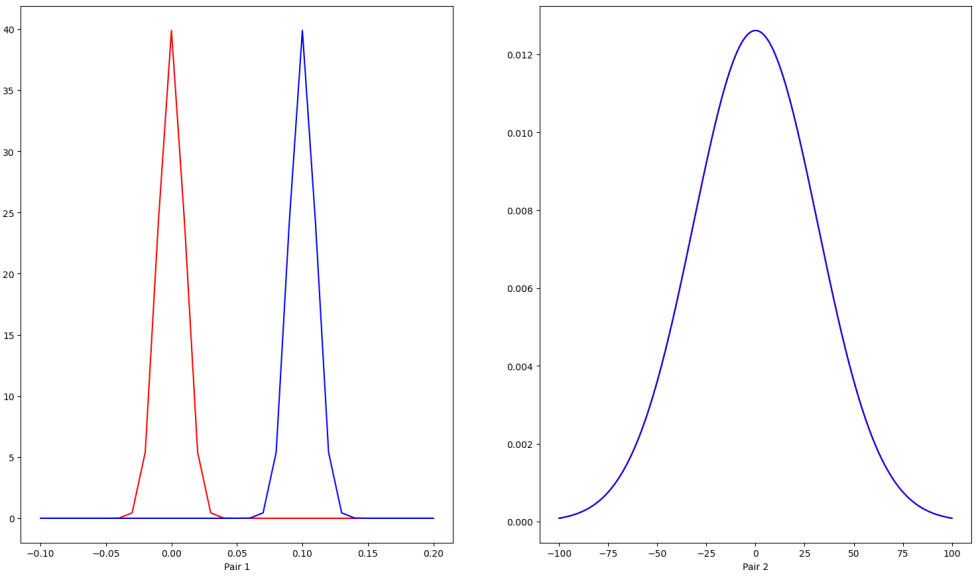


Figure 3: Two pairs of distribution

□

(B) Compute the Euclidean distance between the parameter vector (mean, variance) for both pairs of distributions. For the same pairs of distributions compute the KL-divergence. Which distance fits intuition better and why?

Solution. (a) The euclidean distance is given by:

$$\sqrt{(\mu_1 - \mu_2)^2 + \left(\sigma_1^2 - \sigma_2^2\right)^2}$$

using python we get:

i. d(pair1) = 0.1

ii. d(pair2) = 0.1

(b) The KL-Divergence is given by

$$KL(p, q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

using python we get:

i. d(pair1) = 50.0

ii. d(pair2) = 5×10^{-6}

The KL divergence fits intuition better since it considers the actual probability distribution outcomes(spread) instead of just numerical values of mean and variance. In other words, for pair 2, due to larger variance, the outcomes from two distributions may overlap a lot, much like a low power test where the null distribution and alternative distribution overlaps a lot. □

(C) Assume $\boldsymbol{\theta}_t$ is a parameter for a probability distribution and ρ_t is a scalar. What is the solution to the following optimization algorithm?

$$\max_{\boldsymbol{\theta}_{t+1}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}_t}(y_i|\mathbf{x}_i) + (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)^\top \left[\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i|\mathbf{x}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \right] - \frac{1}{2\rho_t} ||\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t||_2^2$$

Solution. We immediately notice that it is a convex optimization problem w.r.t $\vec{\theta}_{t+1}$. Denote:

$$\mathcal{L}(\vec{\theta}_{t+1}) = \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}_t}(y_i \mid \mathbf{x}_i) + (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)^\top \left[\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \right] - \frac{1}{2\rho_t} \|\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t\|_2^2$$

, we have:

$$\nabla_{\vec{\theta}_{t+1}} \mathcal{L}(\vec{\theta}_{t+1}) = -\frac{1}{\rho_t}(\vec{\theta}_{t+1} - \vec{\theta}_t) + \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} = 0 \tag{1}$$

$$\nabla^2_{\vec{\theta}_{t+1}} \mathcal{L}(\vec{\theta}_{t+1}) = 2 \geq 0$$

thus the function is concave and we solve for (1) and could get: $\vec{\theta}_{t+1} = \vec{\theta}_t + \rho_t \nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i)|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$ □

(D) What algorithm does the previous solution correspond to? Does part (B) say anything about why this algorithm might be suboptimal? How would you fix it?

Solution. The solution is just the gradient descent where the function we are descending is $\sum_{i=1}^{n} \log p_{\boldsymbol{\theta}_t}(y_i \mid \mathbf{x}_i)$. Suboptimality results from the choice of distance metric between the parameters of two consecutive probability distributions $\vec{\theta}_{t+1}$ and $\vec{\theta}_t$. Here we are choosing euclidean distance as our distance metric where it will over-penalize the distance between two parameters. For example in problem (B) for if $\vec{\theta}_t$ is Normal$(0, 1000)$ and $\vec{\theta}_{t+1}$ is Normal$(0.1, 1000)$, then euclidean distance gives 0.1, which is huge compared with KL-divergence metric, which gives 5×10^{-6}. So a quick fix is to rewrite the optimization problem as follows:

$$\max_{\boldsymbol{\theta}_{t+1}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}_t}(y_i \mid \mathbf{x}_i) + (\boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t)^\top \left[\nabla_{\boldsymbol{\theta}} \sum_{i=1}^{n} \log p_{\boldsymbol{\theta}}(y_i \mid \mathbf{x}_i) \Big|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t} \right] - \frac{1}{2\rho_t} D_{KL}(\boldsymbol{\theta}_{t+1}, \boldsymbol{\theta}_t)$$

which gives more accurate measurement of the distance. □

A Codes

In [1]:
```
from torch.utils.data import Dataset
import torch
import matplotlib.pyplot as plt
```

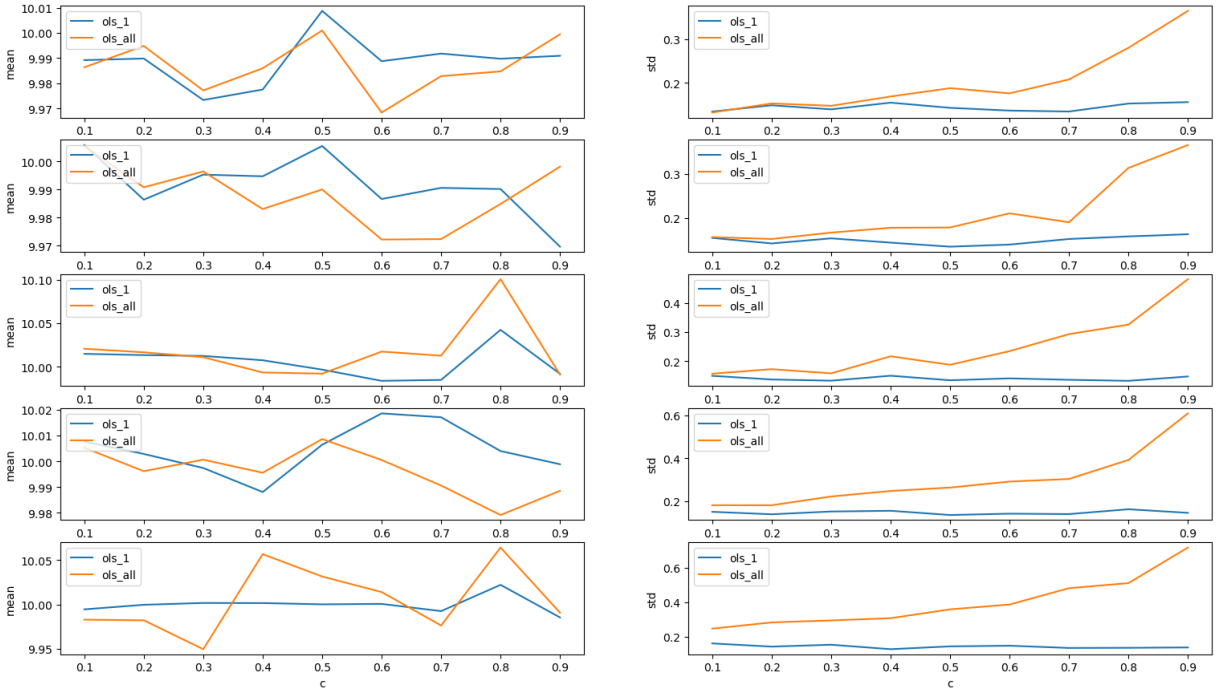
In [11]:
```
import scipy
```

Problem 1

(A)

In [94]:
```
import numpy as np
# Pick w_true
w_true = torch.Tensor([10])

# Parameters
N = 50
T = 100
c_list = np.arange(1,10) * 0.1 # off-diagonal
D_list = [2,4,8,16,32]

# Step 2
def compute_mean_std_iter(D, c):

    mean = [0]*D
    cov = c * np.ones((D, D)) - np.diag([c-1] * D)

    # Generating dataset
    x = torch.Tensor(np.random.multivariate_normal(mean, cov, size = N))
    epsilon = torch.randn(size = (N,))
    y = torch.Tensor(np.diag([w_true.item()] * N)).matmul(x[:,0]) + epsilon


    # Compute OLS
    # OLS_1
    x_1 = x[:,0]
    w_OLS_1 = (x_1.dot(y) / x_1.dot(x_1))

    # OLS_all
    w_OLS_all_1 = (torch.inverse(x.t().matmul(x)) @ (x.t().matmul(y)))[0]

    return w_OLS_1, w_OLS_all_1


# Step 3
def run_T_iters(D, c, T = 100):
    w_OLS_1_list = []
    w_OLS_all_1_list = []
    for i in range(T):
        w_OLS_1,w_OLS_all = compute_mean_std_iter(D, c)
        w_OLS_1_list.append(w_OLS_1)
```

```
        w_OLS_all_1_list.append(w_OLS_all)

    w_OLS_1_mean = torch.stack(w_OLS_1_list, dim = 0).mean(dim = 0)
    w_OLS_1_std = torch.stack(w_OLS_1_list, dim = 0).std(dim = 0)

    w_OLS_all_1_mean = torch.stack(w_OLS_all_1_list, dim = 0).mean(dim = 0)
    w_OLS_all_1_std = torch.stack(w_OLS_all_1_list, dim = 0).std(dim = 0)

    return (w_OLS_1_mean, w_OLS_1_std, w_OLS_all_1_mean, w_OLS_all_1_std)


# Step 4
def run_D_c(c_list, D_list):



    fig, axs = plt.subplots(5, 2)
    fig.set_size_inches(18.5, 10.5)

    for i, D in enumerate(D_list):

        res_list = [run_T_iters(D, c) for c in c_list]

        w_OLS_1_mean = list(map(lambda group: group[0], res_list))
        w_OLS_1_std = list(map(lambda group: group[1], res_list))
        w_OLS_all_1_mean = list(map(lambda group: group[2], res_list))
        w_OLS_all_1_std = list(map(lambda group: group[3], res_list))

        axs[i, 0].plot(c_list, w_OLS_1_mean, label="ols_1")
        axs[i, 0].plot(c_list, w_OLS_all_1_mean, label = 'ols_all')
        axs[i, 0].set_xlabel("c")
        axs[i, 0].set_ylabel("mean")

        axs[i, 1].plot(c_list, w_OLS_1_std, label="ols_1")
        axs[i, 1].plot(c_list, w_OLS_all_1_std, label = 'ols_all')
        axs[i, 1].set_xlabel("c")
        axs[i, 1].set_ylabel("std")

        axs[i, 0].legend(loc='upper left')
        axs[i, 1].legend(loc='upper left')

    plt.show()

run_D_c(c_list, D_list)
```



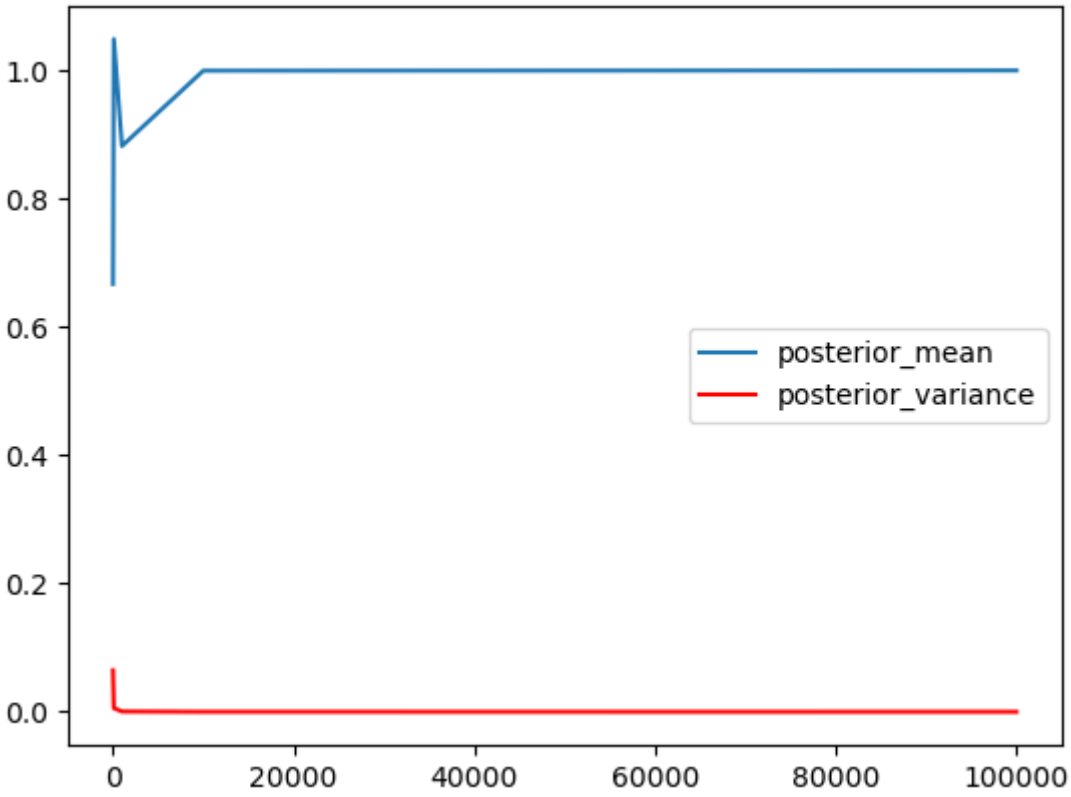
Problem 2

(A)

In [89]:
```
class BLRM_A(Dataset):
    def __init__(self, size = 100, data = None, labels = None):

        self.data = self.generate_data(size)[0] if data is None else data
        self.labels = self.generate_data(size)[1] if labels is None else lab


    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        # Retrieve the sample and label at the specified index

        sample = self.data[index]
        label = self.labels[index]

        return sample, label

    def compute_X_mean(self):
        return self.data

    def generate_data(self, size, w_true = 1):
        # Parameters for data generation
        sigma_epsilon = 1

        # Generate random inputs x ~ N(0, 1)
        x = torch.randn(size)
```

```
        # Generate outputs
        epsilon = sigma_epsilon * torch.randn(size)
        y = w_true * x.pow(2) + epsilon

        return (x,y)


def compute_posterior_A(x, y, prior_mean, prior_variance, likeli_variance):
    N = list(x.size())[0]

    posterior_mean = (x.dot(y) * prior_variance + prior_mean * likeli_variar
    posterior_variance =  (likeli_variance * prior_variance) / (prior_variar
    return posterior_mean, posterior_variance


N_list = [10, 100, 1000, 10000, 100000]

posterior_mean_list = []
posterior_variance_list = []
for N in N_list:
    dataset_N = BLRM_A(N)

    likeli_variance = 1
    prior_mean = 0
    prior_variance = 1
    posterior_mean, posterior_variance = compute_posterior_A(dataset_N.data,
    posterior_mean_list.append(posterior_mean)
    posterior_variance_list.append(posterior_variance)

plt.plot(N_list, posterior_mean_list, label="posterior_mean")
plt.plot(N_list, posterior_variance_list, c="red", label="posterior_variance
plt.legend()
```

Out[89]: <matplotlib.legend.Legend at 0x217c679fc10>



(C)

In [98]:
```
class BLRM_C(Dataset):
    def __init__(self, size = 100, data = None, labels = None):

        self.data = self.generate_data(size)[0] if data is None else data
        self.labels = self.generate_data(size)[1] if labels is None else lab


    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        # Retrieve the sample and label at the specified index

        sample = self.data[index]
        label = self.labels[index]

        return sample, label

    def compute_X_mean(self):
        return self.data

    def generate_data(self, size, w_true = 1):
        # Parameters for data generation
        sigma_epsilon = 1

        # Generate random inputs x ~ N(0, 1)
        x = torch.randn(size)
```

```
        # Generate outputs
        epsilon = sigma_epsilon * torch.randn(size)
        y = w_true * x.pow(2) + epsilon

        return (x,y)

def compute_posterior_C(x, y, prior_mean, prior_variance, likeli_variance):
    # Construct data matrix
    x = x.reshape(-1,1)
    X = torch.cat([x, x.pow(2)],dim = 1)

    posterior_variance = torch.inverse(torch.inverse(prior_variance) + (1 /
    posterior_mean = posterior_variance.matmul(torch.inverse(prior_variance)

    return posterior_mean, posterior_variance


N_list = [10, 100, 1000, 10000]

posterior_mean_list = []
posterior_variance_list = []
for N in N_list:
    dataset_N = BLRM_C(N)

    likeli_variance = 1
    prior_mean = torch.Tensor(np.array([0,0]))
    prior_variance = torch.Tensor(np.array([[1,0],[0,1]]))
    posterior_mean, posterior_variance = compute_posterior_C(dataset_N.data,
    posterior_mean_list.append(posterior_mean)
    posterior_variance_list.append(posterior_variance)

print(posterior_mean_list)
print(posterior_variance_list)
```

```
[tensor([ 0.9784, -0.1960]), tensor([0.0695, 0.3556]), tensor([0.0383, 0.355
3]), tensor([0.0100, 0.3335])]
[tensor([[ 0.3384, -0.1367],
        [-0.1367,  0.0751]]), tensor([[0.0101, 0.0022],
        [0.0022, 0.0034]]), tensor([[1.0018e-03, 3.7392e-05],
        [3.7392e-05, 3.6821e-04]]), tensor([[1.0177e-04, 7.9674e-07],
        [7.9674e-07, 3.4257e-05]])]
```

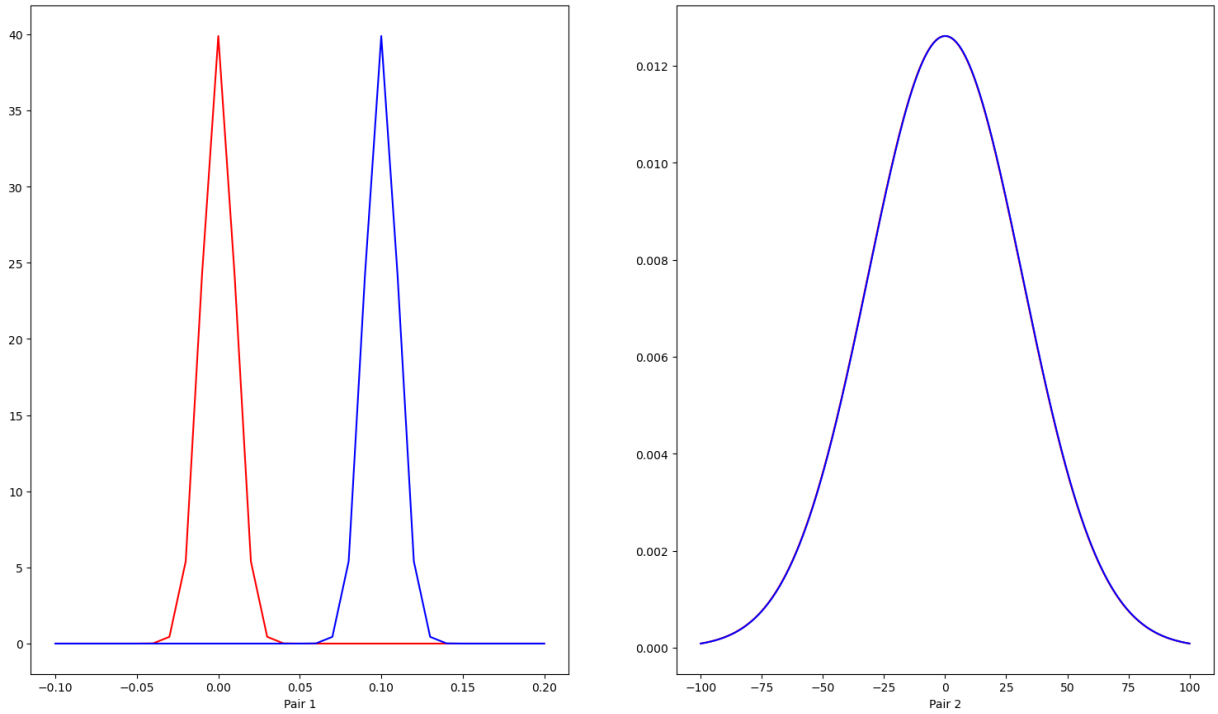
Problem 5

(A)

In [120…

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm
import statistics

# Plot between -10 and 10 with .001 steps.
```

```
x_axis_1 = np.arange(-0.1, 0.2, 0.01)
x_axis_2 = np.arange(-100, 100, 0.01)

# Plot
fig, axs = plt.subplots(1,2)

fig.set_size_inches(18.5, 10.5)
axs[0].plot(x_axis_1, norm.pdf(x_axis_1, 0, np.sqrt(0.0001)), c="red")
axs[0].plot(x_axis_1, norm.pdf(x_axis_1, 0.1, np.sqrt(0.0001)), c="blue")
axs[0].set_xlabel("Pair 1")

axs[1].plot(x_axis_2, norm.pdf(x_axis_2, 0, np.sqrt(1000)), c="red")
axs[1].plot(x_axis_2, norm.pdf(x_axis_2, 0.1, np.sqrt(1000)), c="blue")
axs[1].set_xlabel("Pair 2")
plt.show()
```



(B)

In [126…

```
mu_1_1 = 0
sigma_1_1 = np.sqrt(0.0001)

mu_1_2 = 0.1
sigma_1_2 = np.sqrt(0.0001)

mu_2_1 = 0
sigma_2_1 = np.sqrt(1000)

mu_2_2 = 0.1
sigma_2_2 = np.sqrt(1000)

def compute_euclidean(mu_1, mu_2, sigma_1, sigma_2):
    return np.sqrt((mu_1 - mu_2) ** 2 + (sigma_1 ** 2 - sigma_2 ** 2) ** 2)
```

```
def compute_KL(mu_1, mu_2, sigma_1, sigma_2):
    return np.log(sigma_2 / sigma_1) + (sigma_1 ** 2 + (mu_1 - mu_2) ** 2)/

d1 = compute_euclidean(mu_1_1, mu_1_2, sigma_1_1, sigma_1_2)
d2 = compute_euclidean(mu_2_1, mu_2_2, sigma_2_1, sigma_2_2)

d1_ = compute_KL(mu_1_1, mu_1_2, sigma_1_1, sigma_1_2)
d2_ = compute_KL(mu_2_1, mu_2_2, sigma_2_1, sigma_2_2)
```

In [127…
```
d1, d2
```

Out[127…
```
(0.1, 0.1)
```

In [128…
```
d1_, d2_
```

Out[128…
```
(50.00000000000001, 5.000000000032756e-06)
```

In [ ]:
```

```