## Problem 1 - Concepts and Fundamental Knowledge

**(a)** The statement [VC dimension will be finite for any mode] is **False**

by definition, the VC dimension of a hypothesis space $\mathcal{H}$, denoted as $d_{VC}(\mathcal{H})$, is the cardinality of the largest set that can be shattered by $\mathcal{H}$

if the growth function $\mathcal{G}_{\mathcal{H}}(n) = 2^n$ for all $n$, (i.e., no breakpoint exists, and $\mathcal{H}$ can shatter any set of size $n$), then $d_{VC}(\mathcal{H}) = \infty$

put in words, there are models, such as those with an infinite number of parameters, or those that can fit any possible input-output mapping, that can shatter any set of points, and thus have an infinite VC dimension (regardless how points are labeled)

**(b)** the test error is the error rate of a predictive model on a test set that was not used during training, estimating how well the model will perform on new, unseen data

(i.e., the test error can be expressed as $Er_{\text{test}} = \frac{1}{N}\sum_{i=1}^{N} e(f(\mathbf{x}_i), g(\mathbf{x}_i))$, where $N$ is the number of test samples, $f$ is the predictive model, $g$ is the true function model, and $e$ is the error function)

the out-of-sample error is a measure of how well a model will perform on data that was not used in any part of the training process, and is often used to estimate the generalization error of a model

(i.e., suppose data $\mathbf{x}$ follows a certain distribution $\mathbf{D}$ with **i.i.d.**, then $Er_{\text{out}} = \mathbb{E}_{\mathbf{x}\sim\mathbf{D}}[e(f(\mathbf{x}), g(\mathbf{x}))]$ )

the out-of-sample error is more general than the test error, and we often use the test error to approximate the out-of-sample error when the test set is representative (sufficiently large) of the out-of-sample data

in practice, test error and out-of-sample error are often used interchangeably (especially when the test set is the only set of unseen data available), however, when considering more complex validation schemes, the distinguish out-of-sample error might refer to errors on validation procedures

**(c)** The statement [to make out-of-sample error small, we only need to make in-sample error as small as possible] is **False**

there is a fundamental trade-off in learning between the in-sample error and the out-of-sample error:

$$Er_{\text{out}} = \underbrace{Er_{\text{out}} - Er_{\text{in}}}_{\text{generalization error}} + \underbrace{Er_{\text{in}}}_{\text{training error}} \tag{1}$$

we need to simultaneously make generalization error and training error small in order to achieve a small out-of-sample error

**(d)** The statement [LR is a linear classifier] is **True**

for statement 1), Logistic Regression (LR) is a classification technique that tailored to model the probability of a binary outcome (i.e. $y \in \{-1, +1\}$) (while LR uses a linear combination of features to make predictions, it is not designed for linear regression of continuous outcomes)

for statement 2), the learning problem of Logistic Regression (LR) can be formulated as $\hat{\theta} = \arg\min_{\theta \in \mathbb{R}^d} \mathcal{L}(\theta)$, thus no closed-form solution exists, and iterative optimization methods are required (it is only under certain conditions, LR has a closed-form solution obtained by Maximum Likelihood Estimation (MLE))

for statement 4), the Logistic Regression (LR) can be extended to multi-class classification by using the softmax function using the reasoning of MLE:

$$\widehat{\Theta} = \arg \min_{\Theta \in \mathbb{R}^{d \times K}} -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} 1_{\{y_i = k\}} \log \left( \frac{\exp(\theta_k^{\mathrm{T}} \mathbf{x}_i)}{\sum_{j=1}^{K} \exp(\theta_j^{\mathrm{T}} \mathbf{x}_i)} \right) \tag{2}$$

where $\Theta$ is the parameter matrix, $K$ is the number of classes, and $\theta_k$ is the $k$-th components of $\Theta$

## Problem 2 - Multi-Class Logistic Regression

**(a)** the formulated learning problem:

$$\min_{\Theta,\mathbf{b}} \mathcal{L}(\Theta,\mathbf{b}) \coloneqq -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{K}\mathbf{1}_{\{y_i=l\}}\log\left(\frac{\exp((\theta^l)^{\mathrm{T}}\mathbf{x}_i+\mathbf{b}^l)}{\sum_{j=1}^{K}\exp((\theta^j)^{\mathrm{T}}\mathbf{x}_i+\mathbf{b}^j)}\right) \tag{3}$$

where $\mathbf{x}\in\mathbb{R}^d$ is the input sample, $\mathbf{y}\in\mathbb{R}^K$ is the one-hot vector whose value is 1 for the corresponding class and 0 otherwise, $\Theta\in\mathbb{R}^{K\times d}$ is the parameter matrix, $\mathbf{b}\in\mathbb{R}^K$ is the bias term

denote $\sigma(\mathbf{z}) \coloneqq \frac{\exp(\mathbf{z})}{1^{\mathrm{T}}\exp(\mathbf{z})}\in\mathbb{R}^K$ as the softmax function, where $\exp(\mathbf{z})$ is the element-wise exponential function, hence the optimization problem can be rewritten as:

$$\min_{\Theta,\mathbf{b}} \mathcal{L}(\Theta,\mathbf{b}) \coloneqq \frac{1}{n}\left(-\mathbf{y}^{\mathrm{T}}\log\sigma(\Theta\mathbf{x}+\mathbf{b})\right) \tag{4}$$

let $\mathbf{1}$ denotes all-one vector:

$$\begin{aligned}\mathcal{L}(\Theta,\mathbf{b}) &= \frac{1}{n}\left(-\mathbf{y}^{\mathrm{T}}\left(\log\exp(\Theta\mathbf{x}+\mathbf{b})-\mathbf{1}\log(\mathbf{1}^{\mathrm{T}}\exp(\Theta\mathbf{x}+\mathbf{b}))\right)\right)\\ &= \frac{1}{n}\left(-\mathbf{y}^{\mathrm{T}}(\Theta\mathbf{x}+\mathbf{b})+\log(\mathbf{1}^{\mathrm{T}}\exp(\Theta\mathbf{x}+\mathbf{b}))\right)\end{aligned} \tag{5}$$

differentiating $\mathcal{L}(\Theta,\mathbf{b})$:

$$d\mathcal{L}(\Theta,\mathbf{b}) = \frac{1}{n}\left(-\mathbf{y}^{\mathrm{T}}(d\Theta)\mathbf{x}+\frac{\exp(\Theta\mathbf{x}+\mathbf{b})^{\mathrm{T}}(d\Theta)\mathbf{x}}{\mathbf{1}^{\mathrm{T}}\exp(\Theta\mathbf{x}+\mathbf{b})}\right) \tag{6}$$

take trace operator and rearrange using exchange property:

$$\mathrm{Tr}(d\mathcal{L}(\Theta,\mathbf{b})) = \frac{1}{n}\mathrm{Tr}\left(\mathbf{x}+\mathbf{b}\left(-\mathbf{y}^{\mathrm{T}}+\frac{\exp(\Theta\mathbf{x}+\mathbf{b})^{\mathrm{T}}}{\mathbf{1}^{\mathrm{T}}\exp(\Theta\mathbf{x}+\mathbf{b})}\right)d\Theta\right) \tag{7}$$

hence the gradient of $\mathcal{L}(\Theta,\mathbf{b})$:

$$\begin{aligned}\frac{\partial\mathcal{L}(\Theta,\mathbf{b})}{\partial\Theta} &= \frac{1}{n}\left(-\mathbf{y}+\sigma(\Theta\mathbf{x}+\mathbf{b}\mathbf{1}^{\mathrm{T}})\right)\mathbf{x}^{\mathrm{T}}\\ \frac{\partial\mathcal{L}(\Theta,\mathbf{b})}{\partial\mathbf{b}} &= \frac{1}{n}\left(-\mathbf{y}+\sigma(\Theta\mathbf{x}+\mathbf{b}\mathbf{1}^{\mathrm{T}})\right)\mathbf{1}\end{aligned} \tag{8}$$

**(b)** please refer to "**p2_coffee.py**" and "**p2_weather.py**" for the implementation of the (accelerated)/ gradient descent algorithms (more details on "**p2_test.ipynb**"), figure 1 and figure 2 show the losses and accuracies of the Coffee and Weather datasets respectively

**Brief Summary**:

- **convergence speed**:
  - on the coffee dataset, GD and AGD show a similar convergence speed, in terms of both training and testing, with AGD slightly faster than GD (considering a possible slight edge of AGD)
  - on the weather dataset, AGD shows a significantly faster convergence speed than GD, in terms of both training and testing (steeper initial decrease in loss-curves)
- **accuracy**:
  - for the coffee dataset, both GD and AGD achieve similar accuracies, with AGD slightly better than GD, in terms of training and testing (marginally higher accuracies)
  - for the weather dataset, GD performs similar accuracies during training and testing, and AGD achieves a significantly higher accuracy than GD in training, however, the testing accuracy of AGD is slightly lower than GD (possible overfitting)
- **overfitting**:
  - on the coffee dataset, there is no clear indication of overfitting for either algorithms, as the training and testing accuracies for both GD and AGD are relatively close (good generalizations of both algorithms to the test sets)
  - on the weather dataset, discrepancy occurs between the training and testing progresses, suggesting the existences of overfitting for both GD and AGD, moreover, GD seems to perform better in generalizing to the test set compared to AGD (a significant overfitting of AGD observed)

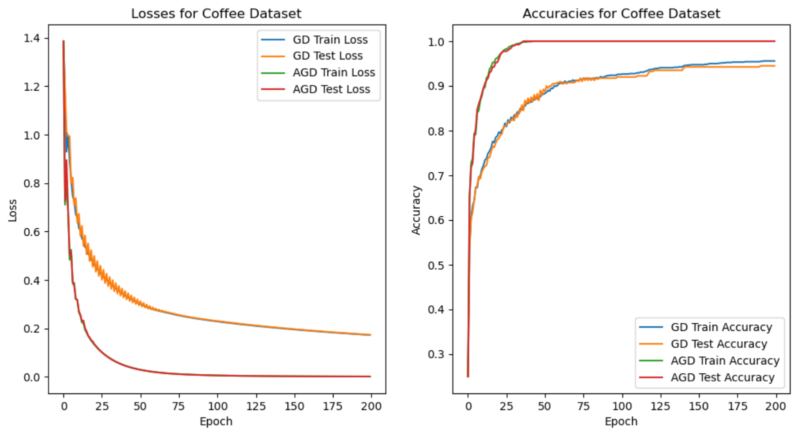


Figure 1: Losses and Accuracies of Coffee Dataset

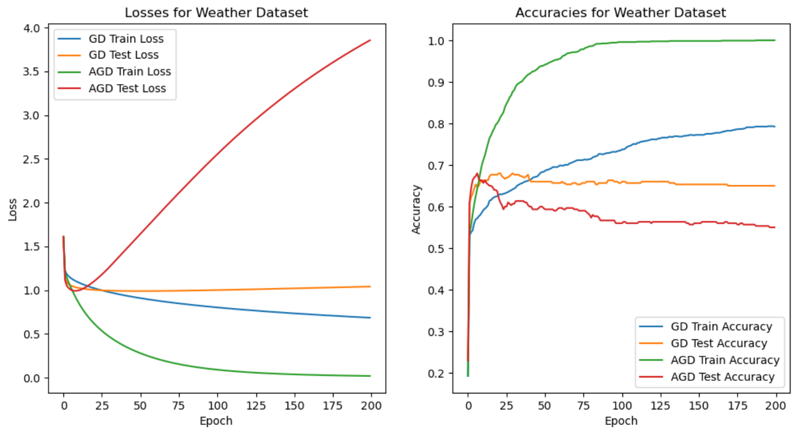


Figure 2: Losses and Accuracies of Weather Dataset

**Problem 3 - Subgradient Method for Optimizing Non-Smooth Function**

**(a)** the plotting results are shown as in figure 3

- **convergence to optimal solution**:
  - for the constant learning rate, the plot shows that it converges to the optimal solution, but with a lower efficiency, compared to the diminishing learning rates (the optimal gap decreases, but still the largest among the three)
  - for the polynomial diminishing learning rate, it converges faster than the constant learning rate, however, this allows a larger initial step size, which fluctuate navigating the optimization landscape (even the learning rate is gradually reducing)
  - for the geometrically diminishing learning rate, it appears to converge at the fastest rate, comparing to the other two, with a smaller number of iterations to reach a smallest optimal gap (helpful for fine-tuning the optimization process)
- **speed of each learning rate schedule**:
  - the speed of constant learning rate is steady but relatively slow, which is typical for constant learning rates where the step size does not adapt to the optimization progress
  - the polynomial diminishing learning rate demonstrates a faster initial decrease, but the fluctuation of the step size might slow down the convergence process, and thus slower than the geometrically diminishing learning rate
  - the geometrically diminishing learning rate shows a steady and fastest convergence rate, suggesting a quicker response to the gradient information at each optimization margin
- **difference from the gradient descent method**:
  - the subgradient method is a generalization of the gradient descent method, which can be applied to non-smooth (i.e., non-differentiable) optimization problems, while the gradient descent method requires differentiability in general

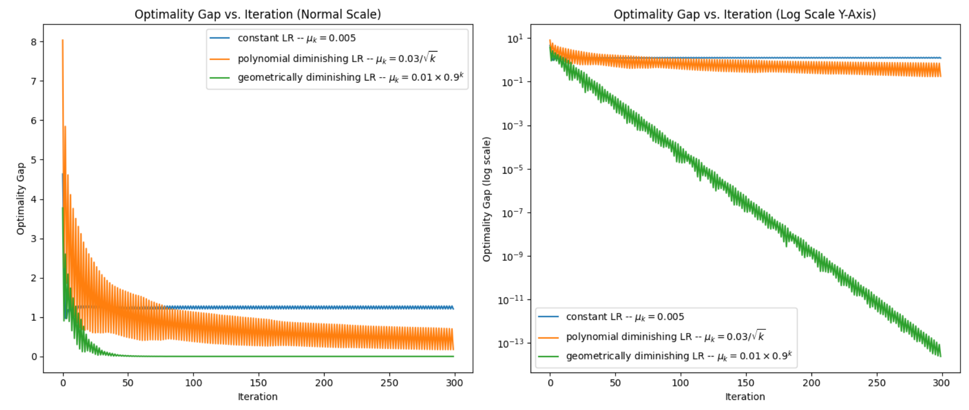


Figure 3: Optimality Gaps of Subgradient Method

- the gradient descent usually converges faster due to the smoothness and differentiability of the objective function, but the subgradient method is more robust in convergence (i.e., non-smooth) because of the non-smooth subgradient
- in gradient descent, the choice of learning rate can be crucial for convergence, while the subgradient method may be less sensitive to the learning rate, diminishing learning rates are often useful for convergence in both methods

in summary, the diminishing learning rates (both geometrically and polynomial) seem to outperform the constant learning rate in terms of convergence speed, suggesting that adapting the learning rate can be beneficial for optimization tuning

the subgradient method, while similar in concept to gradient descent, must handle the non-differentiability of the objective function, which can affect its convergence characteristics and the effectiveness of different learning rate schedules

**Problem 4 - Overfitting**

**(a)** the in-sample error and the defined model:

$$Er_{\text{in}}(f) = \frac{1}{n}\sum_{i=1}^{n}(f(x_i) - y_i))^2, \quad \text{where} \quad f_K(x) = \sum_{k=0}^{K} w_k L_k(x) \tag{9}$$

in matrix notation, the model can be represented as:

$$\mathbf{f}_K(\mathbf{x}) = \mathbf{L}(\mathbf{x})\mathbf{w} = \begin{pmatrix} w_0, w_1, \cdots, w_k \end{pmatrix} \begin{pmatrix} L_0(\mathbf{x}) \\ L_1(\mathbf{x}) \\ \vdots \\ L_k(\mathbf{x}) \end{pmatrix} \tag{10}$$

where $\mathbf{L}(\mathbf{x}) \in \mathbb{R}^{(k+1)\times n}$ is the Legendre polynomial, $\mathbf{w} \in \mathbb{R}^{k+1}$ is the weighted trainable parameter, $\mathbf{x}, \mathbf{y} \in \mathbb{R}^n$ are the input and output samples, hence the in-sample error can be expressed as:

$$Er_{\text{in}}(\mathbf{f}) = \frac{1}{n}\left\|\mathbf{L}(\mathbf{x})\mathbf{w} - \mathbf{y}\right\|_2^2 = \frac{1}{n}\left(\mathbf{w}^{\mathrm{T}}\mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{L}(\mathbf{x})\mathbf{w} - 2\mathbf{w}^{\mathrm{T}}\mathbf{L}(\mathbf{x}^{\mathrm{T}})\mathbf{Y} + \mathbf{y}^{\mathrm{T}}\mathbf{y}\right) \tag{11}$$

thus minimizing the corresponding $Er_{\text{in}}$:

$$\min_{\mathbf{w}} Er_{\text{in}}(\mathbf{f}) = \min_{\mathbf{w}} \frac{1}{n}\left\|\mathbf{L}(\mathbf{x})\mathbf{w} - \mathbf{y}\right\|_2^2 \tag{12}$$

and given that $n \geq K+1$, $\mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{L}(\mathbf{x})$ has full rank (i.e., invertible), the optimal solution by least square can be obtained by the first-order derivative of $Er_{\text{in}}(\mathbf{f})$ with respect to $\mathbf{w}$ such that:

$$\begin{gathered} \frac{\nabla Er_{\text{in}}(\mathbf{f})}{\nabla \mathbf{w}} = \frac{2}{n}\left(\mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{L}(\mathbf{x})\mathbf{w} - \mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{y}\right) = 0 \\ \widehat{\mathbf{w}} = (\mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{L}(\mathbf{x}))^{-1}\mathbf{L}(\mathbf{x})^{\mathrm{T}}\mathbf{y} \end{gathered} \tag{13}$$

hence, we can obtain $f_2$ and $f_{10}$ with $\widehat{\mathbf{w}}$:

$$f_2(\mathbf{x}) = \sum_{k=0}^{2} \hat{\mathbf{w}}_2\mathbf{L}_2(\mathbf{x}) \qquad f_{10}(\mathbf{x}) = \sum_{k=0}^{10} \hat{\mathbf{w}_{10}}\mathbf{L}_{10}(\mathbf{x}) \tag{14}$$

**(b)** the out-of-sample error is defined as:

$$\begin{aligned} Er_{\text{out}}(f) &= \mathbb{E}_x\left[(f(x)-g(x))^2\right] \\ &= \mathbb{E}_x\left[f(x)^2 - 2f(x)g(x) + g(x)^2\right] \end{aligned} \tag{15}$$

this gives the out-of-sample error as the expectation of:

$$\left(\sum_{k=0}^{K} w_k L_k(x)\right)^2 - \frac{2}{C_Q}\left(\sum_{k=0}^{K} w_k L_k(x)\right)\left(\sum_{q=0}^{Q_g} \alpha_q L_q(x)\right) + \frac{1}{C_Q^2}\left(\sum_{q=0}^{Q_g} \alpha_q L_q(x)\right)^2 \tag{16}$$

because the Legendre polynomials of different orders are orthogonal to each other, the cross term in the expectation is zero:

$$\sum_{k=0}^{K}(w_k L_k(x))^2 - \frac{2}{C_Q}\sum_{j=0}^{\min(K,Q_g)} w_j \alpha_j L_j^2(x) + \frac{1}{C_Q^2}\sum_{q=0}^{Q_g}(\alpha_q L_q(x))^2 \tag{17}$$

given that $\int_{-1}^{1} L_q(x) L_q(x) dx = \frac{2}{2q+1}$, this can be further simplified as:

$$\sum_{k=0}^{K} w_k^2 \frac{2}{2k+1} - \frac{2}{C_Q}\sum_{j=0}^{\min(K,Q_g)} w_j \alpha_j \frac{2}{2j+1} + \frac{1}{C_Q^2}\sum_{q=0}^{Q_g} \alpha_q^2 \frac{2}{2q+1} \tag{18}$$

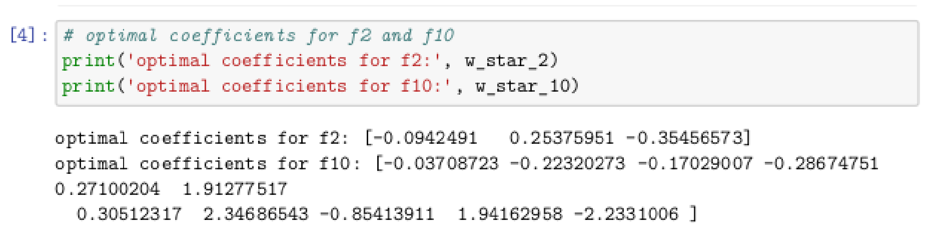
hence the $Er_{\text{out}}$ for the learned $f_2$ and $f_{10}$ can be calculated by substituting the optimal $\hat{\mathbf{w}}$ into the equation above

$$\begin{aligned} Er_{\text{out}}(f_2) &= \sum_{k=0}^{2} \hat{w_k}^2 \frac{2}{2k+1} - \frac{2}{C_Q}\sum_{j=0}^{\min(2,Q_g)} \hat{w_j}\alpha_j \frac{2}{2j+1} + \frac{1}{C_Q^2}\sum_{q=0}^{Q_g}\alpha_q^2 \frac{2}{2q+1} \\ Er_{\text{out}}(f_{10}) &= \sum_{k=0}^{10} \hat{w_k}^2 \frac{2}{2k+1} - \frac{2}{C_Q}\sum_{j=0}^{\min(10,Q_g)} \hat{w_j}\alpha_j \frac{2}{2j+1} + \frac{1}{C_Q^2}\sum_{q=0}^{Q_g}\alpha_q^2 \frac{2}{2q+1} \end{aligned} \tag{19}$$

**(c)** please refer to "**p4_overfitting.ipynb**" for the experiments of the overfitting behaviours

**(c) - (1)** for the fixed $\sigma^2 = 0.1$, and $Q_g \in \{1, 2, \ldots, 50\}$, $n \in \{20, 25, \ldots, 120\}$, the optimal coefficients for $f_2$ and $f_{10}$ are shown in the following figure 4, more details on the results of $Er_{\text{out}}(f_2)$, $Er_{\text{out}}(f_{10})$ and the overfitting measure $\varrho_e$ can be found in codes

the heat-map for $\bar{\varrho}_e$ for each pair of $(n, Q_g)$ is shown in figure 5



```
[4]: # optimal coefficients for f2 and f10
     print('optimal coefficients for f2:', w_star_2)
     print('optimal coefficients for f10:', w_star_10)
```

```
optimal coefficients for f2: [-0.0942491   0.25375951 -0.35456573]
optimal coefficients for f10: [-0.03708723 -0.22320273 -0.17029007 -0.28674751
0.27100204  1.91277517
  0.30512317  2.34686543 -0.85413911  1.94162958 -2.2331006 ]
```

Figure 4: Optimal Coefficients for $f_2$ and $f_{10}$

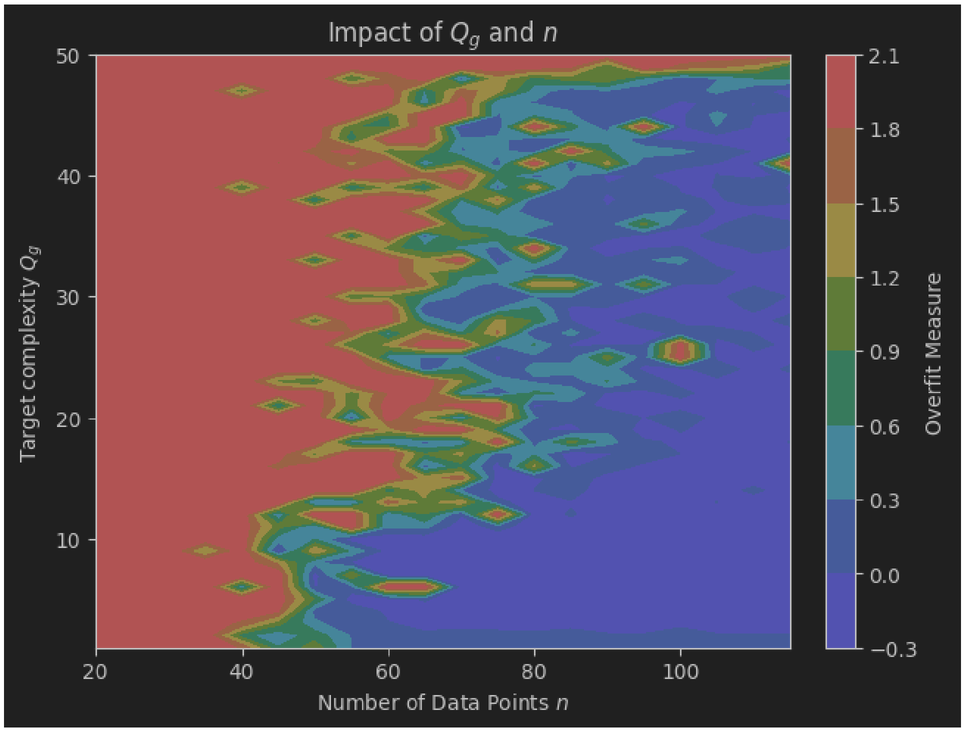


Figure 5: Heat-map of Overfitting Measure $\bar{\varrho}_e$ for Each Pair of $(n, Q_g)$

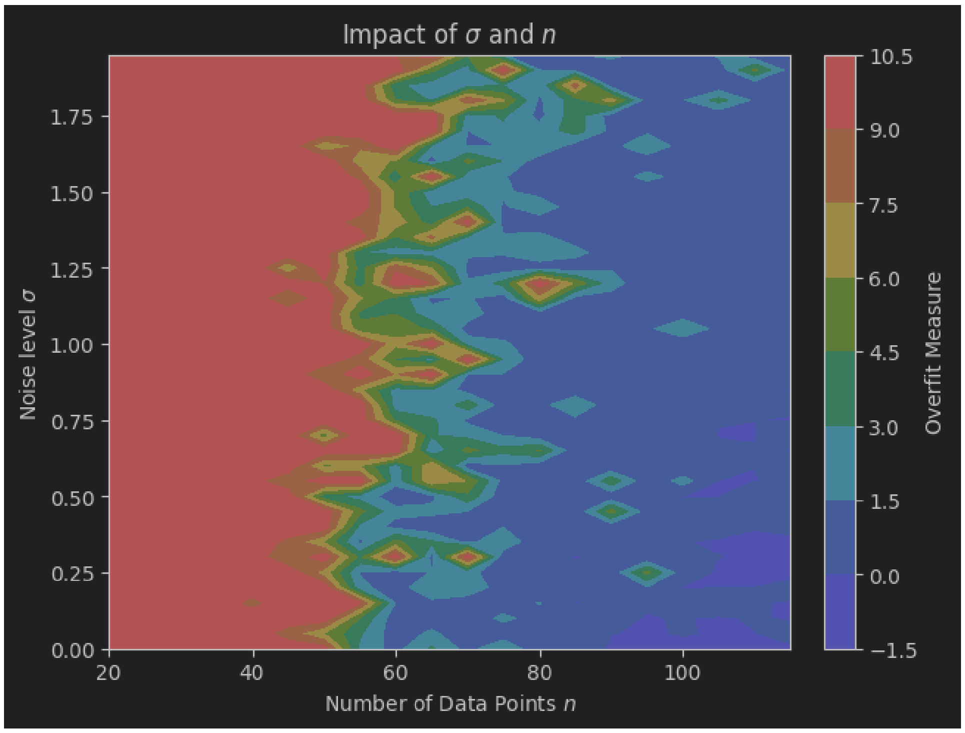


Figure 6: Heat-map of Overfitting Measure $\bar{\varrho}_e$ for Each Pair of $(n, \sigma^2)$

**(c) - (2)** for the fixed $Q_g = 20$, and $\sigma^2 \in \{0, 0.05, \ldots, 2\}$, $n \in \{20, 25, \ldots, 120\}$, the heat-map for $\bar{\varrho}_e$ for each pair of $(n, \sigma^2)$ is shown in figure 6

**(c) - (3)** in summary, these experiments examine the overfitting behaviours under different settings of the number of samples $n$, the model complexity $Q_g$, and the noise level $\sigma^2$

- both figure 5 and figure 6 illustrate that, as the number of samples $n$ increases, the overfitting measure $\bar{\varrho}_e$ decreases in general, suggesting that more samples can help learn a more generalizable model
- figure 5 also shows that, with the complexity $Q_g$ increases, the overfitting measure $\bar{\varrho}_e$ increases given a fixed number of samples $n$, suggesting that more complex models tend to overfit because it may capture the noise in the data rather than the underlying pattern

- while figure 6 additionally shows that, as the noise level $\sigma^2$ increases, the overfitting measure $\bar{\varrho}_e$ increases given a fixed number of samples $n$, suggesting that more noise in the training data can be responsible to overfitting

these introduce the causing factors of overfitting and how they might influence the model:

- **model complexity**: higher complexity in model ($Q_g$) can lead to overfitting as the model may capture noise
- **insufficient samples**: fewer samples ($n$) can result in overfitting because of less information to learn from
- **noise level**: higher noise level ($\sigma^2$) can cause overfitting as the model may adapt the noise rather than the pattern