DDA3020 Homework 2

Due date: Oct 14, 2024

Instructions

- The **deadline** is 23:59, Oct 14, 2024.
- The weight of this assignment in the final grade is 20%.
- **Electronic submission**: Turn in solutions electronically via Blackboard. Be sure to submit your homework as one pdf file plus two python scripts. Please name your solution files as "DDA3020HW1_studentID_name.pdf", "HW1_yourID_Q1.ipynb" and "HW1_yourID_Q2.ipynb". (.py files also acceptable)
- Note that **late submissions** will result in discounted scores: 0-24 hours $\rightarrow$ 80%, 24-120 hours $\rightarrow$ 50%, 120 or more hours $\rightarrow$ 0%.
- Answer the questions in English. Otherwise, you'll lose half of the points.
- Collaboration policy: You need to solve all questions independently and collaboration between students is **NOT** allowed.

1 Written Problems (50 points)

1.1. (MLP & CNN, 25 points)

(Backpropagation for MLP, 15 points) Consider the following classification MLP with one hidden layer:

$$\begin{aligned}
\boldsymbol{x} &= \text{input} \in \mathbb{R}^D \\
\boldsymbol{z} &= \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}_1 \in \mathbb{R}^K \\
\boldsymbol{h} &= \text{ReLU}(\boldsymbol{z}) \in \mathbb{R}^K \\
\boldsymbol{a} &= \boldsymbol{V}\boldsymbol{h} + \boldsymbol{b}_2 \in \mathbb{R}^C \\
\mathcal{L} &= \text{CrossEntropy}(\boldsymbol{y}, \text{Softmax}(\boldsymbol{a})) \in \mathbb{R}
\end{aligned}$$

Where: $\mathbf{W} \in \mathbb{R}^{K\times D}$ and $\mathbf{b}_1 \in \mathbb{R}^K$ are the weights and biases of the hidden layer. $\mathbf{V} \in \mathbb{R}^{C\times K}$ and $\mathbf{b}_2 \in \mathbb{R}^C$ are the weights and biases of the output layer. D is the input dimension, K is the number of hidden units, and C is the number of classes. $\mathbf{y} \in \mathbb{R}^C$ is the one-hot encoded true label.

The activation functions and loss function are defined as:

- ReLU activation function: $\text{ReLU}(a) = \max(0, a)$
- Softmax function: $\text{Softmax}(\mathbf{a}) = \dfrac{\exp(\mathbf{a})}{\sum_{i=1}^{C} \exp(a_i)}$
- Cross-Entropy loss function: $\text{CrossEntropy}(\boldsymbol{y}, \boldsymbol{p}) = -\sum_{i=1}^{C} y_i \log p_i$

Task: Draw the computational graph of forward pass and derive the specific form $\boldsymbol{\psi}_i, i = 1, \ldots, 5$ of gradients w.r.t. the parameters and input.

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= \boldsymbol{\psi}_1(\mathbf{u}_2, \mathbf{h}) \in \mathbb{R}^{C\times K} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}_2} &= \boldsymbol{\psi}_2(\mathbf{u}_2) \in \mathbb{R}^{C} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{W}} &= \boldsymbol{\psi}_3(\mathbf{u}_1, \mathbf{x}) \in \mathbb{R}^{K\times D} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{b}_1} &= \boldsymbol{\psi}_4(\mathbf{u}_1) \in \mathbb{R}^{K} \\
\frac{\partial \mathcal{L}}{\partial \mathbf{x}} &= \boldsymbol{\psi}_5(\mathbf{W}, \mathbf{u}_1) \in \mathbb{R}^{D}
\end{aligned}
$$

Where:

$$
\begin{aligned}
\mathbf{u}_2 &= \frac{\partial \mathcal{L}}{\partial \mathbf{a}} = \mathbf{p} - \mathbf{y} \in \mathbb{R}^C \\
\mathbf{u}_1 &= \frac{\partial \mathcal{L}}{\partial \mathbf{z}} = \mathbf{V}^\top \mathbf{u}_2 \odot H(\mathbf{z}) \in \mathbb{R}^K \\
H(\mathbf{z}) &= \mathbb{I}(\mathbf{z} > 0) \quad \text{(derivative of ReLU function)}
\end{aligned}
$$

where $\odot$ represents the element-wise (Hadamard) product, $H(a) = \mathbb{I}(a > 0)$ is the heaviside function.

Hint: Use the chain rule and matrix calculus rules, and pay attention to the dimensions of matrices and vectors to ensure correct computations.

Note: in our notation, the gradient (which has the same shape as the variable with respect to which we differentiate) is equal to the Jacobian's transpose when the variable is a vector and to the first slice of the Jacobian when the variable is a mtrix.

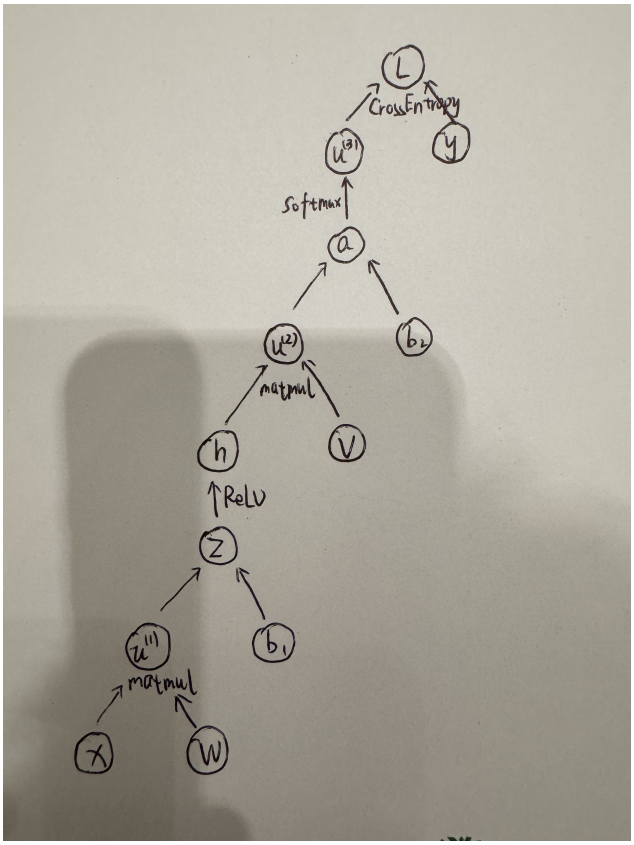


Figure 1: Answer of 1.1, computational graph (5 pts)

$$\nabla_{\boldsymbol{V}}\mathcal{L} = [\frac{\partial \mathcal{L}}{\partial \boldsymbol{V}}]_{1,:} = \boldsymbol{\psi}_1(\boldsymbol{u_2}, \boldsymbol{h}) = \boldsymbol{u_2}\boldsymbol{h}^T \in \mathbb{R}^{C\times K} \quad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{b}_2}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_2})^T = \boldsymbol{\psi}_2(\boldsymbol{u}_2) = \boldsymbol{u}_2 \in \mathbb{R}^C \quad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{W}}\mathcal{L} = [\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}]_{1,:} = \boldsymbol{\psi}_3(\boldsymbol{u_1}, \boldsymbol{x}) = \boldsymbol{u}_1\boldsymbol{x}^T \in \mathbb{R}^{K\times D} \quad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{b}_1}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{b}_1})^T = \boldsymbol{\psi}_4(\boldsymbol{u}_1) = \boldsymbol{u}_1 \in \mathbb{R}^K \quad \text{(2 pts)}$$

$$\nabla_{\boldsymbol{x}}\mathcal{L} = (\frac{\partial \mathcal{L}}{\partial \boldsymbol{x}})^T = \boldsymbol{\psi}_5(\boldsymbol{W}, \boldsymbol{u}_1) = \boldsymbol{W}^T\boldsymbol{u}_1 \in \mathbb{R}^D \quad \text{(2 pts)}$$

(CNN, 10 points) Consider the convolutional network defined by the layers below. The input shape is $32 \times 32 \times 3$ and the output is 10 neurons. Consider layers:

$$\text{Conv5(10)} + \text{Maxpool}_2 + \text{Conv3(20)} + \text{Maxpool}_2 + \text{FC10}$$

where

- Conv5(10): 10 filters with each size $5 \times 5 \times D$, where D is the depth of the activation volume at the previous layer, stride $= 1$, padding $= 2$;

- Conv3(20): 20 filters with each size $3 \times 3 \times D$, where D is the depth of the activation volume at the previous layer, stride = 1, padding = 1;
- Maxpool_2: 2×2 filter, stride = 2, padding = 0;
- FC10: A fully-connected layer with 10 output neurons.

1. (5 pts) Compute the shape of activation map of each layer.
2. (5 pts) Compute the total number of parameters of each layer.

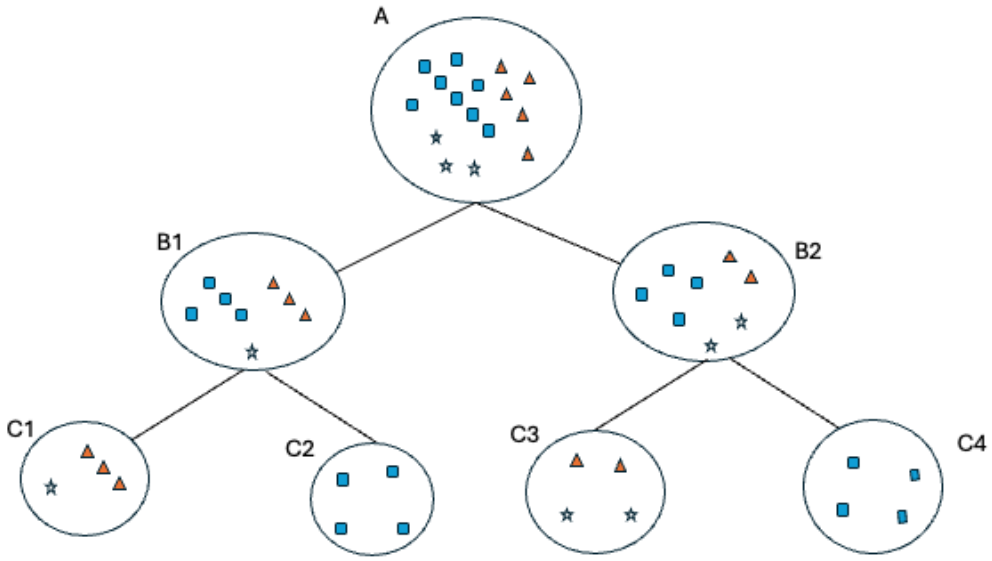
The answer is as in Table 1.

Table 1: Understanding and calculating the number of parameters in CNN

NO.	**Layer**	**Activation Shape**	**# Parameters**	Mark
1	Input Layer	(32,32,3)	0	-
2	Conv5(10)	(32,32,10)	(5*5*3 + 1)*10 = 760	2 pts
3	Maxpool2	(16,16,10)	0	2 pts
4	Conv3(20)	(16,16,20)	(3*3*10 + 1)*20 = 1820	2 pts
5	Maxpool2	(8,8,20)	0	2 pts
6	FC10	(10,1)	8*8*20*10 + 10 = 12810	2 pts

1.2. (CART & Performance Evaluation, 25 points)

i) Compute the Gini index, the entropy and the classification error for each node of the tree in the figure below, where different color (also shape) represents different class.



(14 pts)

ii) Suppose we randomly sample a training set D from some un-known distribution. For each training set D we sample, we train a regression tree h_D to predict y from x (one dimensional). We repeat this process 10 times resulting in 10 trained models. For a new test sample $(x_1, x_2, x_3, x_4) = (3.1, 1.5, 0.0, 4.3)$, with true output $y_{\text{truth}} = 8.0$.
Suppose the predictions of the new test sample based on the 10 trained models are 7.0, 8.0, 9.0, 6.0, 9.0, 3.0, 9.0, 8.0, 9.0, 8.0. Based on this 10 trials, please compute the empirical mean squared error (MSE), $Bias^2$ and Variance on this test sample. (11 pts)
Hint:

– For a new test sample (x, y), we define its mean squared error (MSE) by different models as:

$$\text{MSE}(x, y) = E_D\left[(h_{D_i}(x) - y)^2\right]$$

– It can be observed that: $E_{(x,y),D}\left[(h_{D_i}(x) - y)^2\right] = E_{(x,y)}[MSE(x, y)]$

– The empirical estimation of MSE based on n trained models is:

$$\widehat{MSE}(x, y) = \frac{1}{n}\sum_{i=1}^{n}(h_D(x) - y)^2$$

i) Calculation for the Gini index, the Entropy and the Classification error:
For each node, 2 points for all calculations correct, 1 point for partly correct, 0 points

for all wrong answers

Detailed Calculation of Node A:

$$\begin{aligned} \text{Entropy} &= -\left(\frac{8}{16}\log_2\frac{8}{16} + \frac{5}{16}\log_2\frac{5}{16} + \frac{3}{16}\log_2\frac{3}{16}\right) \\ &= -\left(0.5 \cdot \log_2(0.5) + 0.3125 \cdot \log_2(0.3125) + 0.1875 \cdot \log_2(0.1875)\right) \\ &= 1.477 \end{aligned} \tag{1}$$

$$\begin{aligned} \text{Gini} &= 1 - \left(\left(\frac{8}{16}\right)^2 + \left(\frac{5}{16}\right)^2 + \left(\frac{3}{16}\right)^2\right) \\ &= 1 - \left(0.5^2 + 0.3125^2 + 0.1875^2\right) \\ &= 0.617 \end{aligned} \tag{2}$$

$$\begin{aligned} \text{Classification Error} &= 1 - \max\left(\frac{8}{16}, \frac{5}{16}, \frac{3}{16}\right) \\ &= 1 - 0.5 = 0.5 \end{aligned} \tag{3}$$

Table 2: Table showing node statistics with entropy, Gini index, and classification error

Node	Counts	Entropy	Gini Index	Classification Error
A	[8, 5, 3]	1.477217	0.617188	0.5
B1	[4, 3, 1]	1.405639	0.59375	0.5
B2	[4, 2, 2]	1.5	0.625	0.5
C1	[0, 3, 1]	0.811278	0.375	0.25
C2	[4, 0, 0]	-0.0	0.0	0.0
C3	[0, 2, 2]	1.0	0.5	0.5
C4	[4, 0, 0]	-0.0	0.0	0.0

ii) 2 points for correct average prediction value, 3 points for correct answers for each correct output(MSE, Variance and Bias)

$$\begin{aligned} \mathbb{E}[\hat{y}] &= \frac{1}{10}\sum_{i=1}^{10}\hat{y}_i \\ \mathbb{E}[\hat{y}] &= \frac{7.0 + 8.0 + 9.0 + 6.0 + 9.0 + 3.0 + 9.0 + 8.0 + 9.0 + 8.0}{10} = \frac{76.0}{10} = 7.6 \end{aligned} \tag{4}$$

$$\begin{aligned} \text{MSE}(x, y) = \frac{1}{10}\big((7.0 - 8.0)^2 + (8.0 - 8.0)^2 + (9.0 - 8.0)^2 + (6.0 - 8.0)^2 + (9.0 - 8.0)^2 \\ +(3.0 - 8.0)^2 + (9.0 - 8.0)^2 + (8.0 - 8.0)^2 + (9.0 - 8.0)^2 + (8.0 - 8.0)^2\big) = 3.4 \end{aligned} \tag{5}$$

$$\begin{aligned} \text{Variance} = \frac{1}{10}\big((7.0 - 7.6)^2 + (8.0 - 7.6)^2 + (9.0 - 7.6)^2 + (6.0 - 7.6)^2 + (9.0 - 7.6)^2 \\ +(3.0 - 7.6)^2 + (9.0 - 7.6)^2 + (8.0 - 7.6)^2 + (9.0 - 7.6)^2 + (8.0 - 7.6)^2\big) = 3.24 \end{aligned} \tag{6}$$

$$\begin{aligned} \text{Bias}^2 &= (\mathbb{E}[\hat{y}] - y_{\text{true}})^2 \\ \text{Bias}^2 &= (7.6 - 8.0)^2 = (-0.4)^2 = 0.16 \end{aligned} \tag{7}$$

2 Programming (50 points)

2.1 Decision Tree (25 points)

Task Description You are asked to classify the variable species in the **Penguins** dataset, using decision tree, bagging, and random forests. All these algorithms should be implemented by calling **sklearn** in Python, including "DecisionTreeClassifier", "BaggingClassifier" and "RandomForestClassifier".

Datasets **Penguins** contains 345 data points (saved in 345 rows) and 3 classes. For each data, the first column is the species of penguins that we want to classify; the remaining 6 columns indicate 6 features (or attributes). You can use "penguins=seaborn.load_dataset('penguins')" to load the dataset.

What you should do

1. **(1.5 points) Data Preprocessing**: Firstly, check the data in the dataset and delete incomplete data points with any feature as 'NaN' or 'Null'. Then, you should convert features of string type into numerical values. Lastly, randomly split the whole dataset into train/test with the ratio 70% for training set and 30% for test set. You should report data size before and after preprocessing. The output format is as follows:

```
Q2.1.1 Data Preprocessing:
Number of data points in the whole dataset originally:  xx
Number of data points in the whole dataset after deleting incomplete data:  xx
Number of data points in the training set after splitting:  xx
Number of data points in the test set after splitting:  xx
```

You should fill up xx in the template.

Number of data points in the whole dataset originally: 344
Number of data points in the whole dataset after deleting incomplete data: 333
Number of data points in the training set after splitting: 233
Number of data points in the test set after splitting: 100

Scoring scheme:

- 0.5 points for "delete incomplete data points with any feature as 'NaN' or 'Null"'. The number of data points in the whole dataset before and after deleting incomplete data should be correct.
- 0.5 points for "convert features of string type into numerical values" (check the code).
- 0.5 points for "split the whole dataset into train/test with the ratio 70% for training set and 30% for test set". The number of data points in the training set and test set should be correct.

2. **(2.5 points) Data Statistics**: Analyze the statistics of each feature across three classes. You visualize the statistics by plotting a histogram for each feature per class. You should report 6 figures (corresponding to 6 features), each figure contains 3 subfigures (corresponding to 3 classes). For example, in the i-th figure, there should be 3 subfigures, where the first subfigure is the histogram for the i-th feature in the first class, and the second subfigure is the histogram for the i-th feature in the second class, etc. Remember to set the title for each subfigure and each figure, indicating the class name and feature name respectively. The report format is as follows:

```
Q2.1.2 Data Statistics:
{figure 1}
{figure 2}
{figure 3}
{figure 4}
{figure 5}
{figure 6}
```

You should fill up `{figure x}` in the template. You can design the order of subfigures in each figure by yourself.

Scoring scheme:

- 1 point for plotting six figures corresponding to six features correctly.
- 1 point for plotting three subfigures corresponding to three classes for each figure (i.e. feature) correctly.
- 0.5 points for setting figures' and subfigures' titles correctly.

You can see the example in the provided "HW2_decision_tree_code.ipynb".

Additionally, it is acceptable to merge subfigures across classes into one, i.e. six figures corresponding to six features with each figure containing all three classes rather than splitting into three subfigures.

3. **(6 points) Decision Tree**: Solve the above problem using the decision tree method; report the train/test accuracy with respect to three different maximum depths, and three different least node sizes; plot each learned tree. The report format is as follows:

```
Q2.1.3 Decision Tree:
1. Results with maximum depth as d1 and least node size as n1:
training accuracy: xx, test accuracy: xx
{learned tree figure}

2. Results with maximum depth as d1 and least node size as n2:
training accuracy: xx, test accuracy: xx
{learned tree figure}

3. Results with maximum depth as d1 and least node size as n3:
training accuracy: xx, test accuracy: xx
{learned tree figure}

4. Results with maximum depth as d2 and least node size as n1:
training accuracy: xx, test accuracy: xx
{learned tree figure}

5. Results with maximum depth as d2 and least node size as n2:
training accuracy: xx, test accuracy: xx
{learned tree figure}

6. Results with maximum depth as d2 and least node size as n3:
training accuracy: xx, test accuracy: xx
{learned tree figure}

7. Results with maximum depth as d3 and least node size as n1:
training accuracy: xx, test accuracy: xx
{learned tree figure}

8. Results with maximum depth as d3 and least node size as n2:
training accuracy: xx, test accuracy: xx
{learned tree figure}

9. Results with maximum depth as d3 and least node size as n3:
training accuracy: xx, test accuracy: xx
{learned tree figure}
```

You should first select your settings and fill up `d1`, `d2`, `d3`, `n1`, `n2`, `n3`. Accuracy are calculated by $\frac{\text{correct prediction}}{\text{number of data}}$.

Scoring scheme:

- 1 point for training and prediction with decision tree classifier correctly.
- 1 point for setting maximum depth correctly.
- 1 point for setting the least node size correctly.
- 1 point for calculating training accuracy correctly; 1 point for test accuracy.
- 1 point for plotting learned trees.

4. **(5 points) Bagging of Trees**: Solve the above problem using the bagging method, with the decision tree as the base learner; report the train/test accuracy with respect to three different maximum depths, and three different numbers of trees. The report format is as follows:

```
Q2.1.4 Bagging of Trees:
1.  Results with maximum depth as d1 and number of trees as n1:
training accuracy:  xx, test accuracy:  xx

2.  Results with maximum depth as d1 and number of trees as n2:
training accuracy:  xx, test accuracy:  xx

3.  Results with maximum depth as d1 and number of trees as n3:
training accuracy:  xx, test accuracy:  xx

4.  Results with maximum depth as d2 and number of trees as n1:
training accuracy:  xx, test accuracy:  xx

5.  Results with maximum depth as d2 and number of trees as n2:
training accuracy:  xx, test accuracy:  xx

6.  Results with maximum depth as d2 and number of trees as n3:
training accuracy:  xx, test accuracy:  xx

7.  Results with maximum depth as d3 and number of trees as n1:
training accuracy:  xx, test accuracy:  xx

8.  Results with maximum depth as d3 and number of trees as n2:
training accuracy:  xx, test accuracy:  xx

9.  Results with maximum depth as d3 and number of trees as n3:
training accuracy:  xx, test accuracy:  xx
```

Scoring scheme:

- 1 point for training and prediction with bagging of trees correctly.
- 1 point for setting maximum depth correctly.
- 1 point for setting the number of trees correctly.
- 1 point for calculating training accuracy correctly; 1 point for test accuracy.

5. **(5 points) Random Forests**: Solve the above problem using the random forest method, with the decision tree as the base learner; report the train/test accuracy with respect to three different maximum depths, and three different numbers of trees. The report format is as follows:

```
Q2.1.5 Random Forests:
1.  Results with maximum depth as d1 and number of trees as n1:
```

```
training accuracy: xx, test accuracy: xx

2. Results with maximum depth as d1 and number of trees as n2:
training accuracy: xx, test accuracy: xx

3. Results with maximum depth as d1 and number of trees as n3:
training accuracy: xx, test accuracy: xx

4. Results with maximum depth as d2 and number of trees as n1:
training accuracy: xx, test accuracy: xx

5. Results with maximum depth as d2 and number of trees as n2:
training accuracy: xx, test accuracy: xx

6. Results with maximum depth as d2 and number of trees as n3:
training accuracy: xx, test accuracy: xx

7. Results with maximum depth as d3 and number of trees as n1:
training accuracy: xx, test accuracy: xx

8. Results with maximum depth as d3 and number of trees as n2:
training accuracy: xx, test accuracy: xx

9. Results with maximum depth as d3 and number of trees as n3:
training accuracy: xx, test accuracy: xx
```

Scoring scheme:

- 1 point for training and prediction with random forests correctly.
- 1 point for setting maximum depth correctly.
- 1 point for setting the number of trees correctly.
- 1 point for calculating training accuracy correctly; 1 point for test accuracy.

6. **(5 points) Relationship of Bias/Variance with respect to Number of Trees in Random Forests**: Plot the curve of bias^2 with respect to different numbers of trees in random forests, e.g., $tree = 10, 20, \ldots, 100$. Then, describe the relationship between bias^2 and different numbers of trees; repeat the procedure for variance. You can use the function from https://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/ to get the results. The report format is as follows:

```
Q2.1.6 Relationship of Bias/Variance with respect to Number of Trees in Random Forests:
bias^2 - trees:
{figure}
{description of relationship}
```

```
variance - trees:
{figure}
{description of relationship}
```

Scoring scheme:

- 0.5 points for calling random forests correctly.
- 0.5 points for using "bias_variance_decomp" function.
- 1 point for plotting the curve of bias^2 with respect to different numbers of trees in random forests; 1 point for plotting the curve of variance.
- 1 point for the description of the relationship of the first figure; 1 point for the second.

2.2 Performance Evaluation (25 points)

Machine learning performance evaluation is crucial for assessing how well a model performs on a given task, which is essential for model selection, hyper-parameter tuning, and ensuring that the model can generalize to new, unseen data. Generally, for supervised learning tasks, we can use K-fold Cross-validation to tune the hyper-parameters on validation set and then evaluate the model performance on test set.

We want to compare the classification performance of logistic regression and SVM (Support Vector Machine) on tabular mnist dataset. The training, validation and test set are provided in the attachment (HW2_Q2.zip), in which training and validation data are included in "train_validation_data.npy" that contains 1000 samples (two classes (0, 1), half-and-half) with 100 attributes, and test data is in "test_data.npy" that contains 400 samples (two classes (0, 1), half-and-half) with 100 attributes. The corresponding "XX_label.npy" files contain true label information.

An unfinished code project is provided in "HW2_Q2.ipynb", you need to complete the blank code cells marked by **TODO** to finish your evaluation. (**Note**: You need to include the completed code project when submitting your answer report.)

For the binary classification task, you are asked to conduct **5-fold cross-validation** on provided dataset. Now, you need to finish the following tasks step by step.

(**Note**: For Question 1 and Question 2, you do not need to put your implementations in the answer report, just save them in the code project.)

This question is designed to test the hyper-parameters fine-tuning and model selection for supervised learning task. The main content concerns the correct implementation and execution of K-folds cross validation and several evaluation metrics (Precision, Recall, F1, Accuracy, AUROC).

See an example code in HW2_Q2_Completed.ipynb for Question 1 and Question 2.

Scoring of Rubric

1. (10 pts) Note that performing a data shuffle before data splitting is not necessary for K-Folds Cross Validation. No marks will be deducted for answers adding randomness before data splitting.

   - (10 pts): The completed code implements K-folds cross validation and runs correctly in subsequent code calls.
   - (-5 pts): The data splitting logic is correct but the code does not run correctly.
   - (-5 pts): Uses existing functions (from third-party package) that implement most of K-Fold's functionality.

2. (6 pts)

   - (1 pts): The completed code implements Precision metric and runs correctly in subsequent code calls.
   - (1 pts): The completed code implements Recall metric and runs correctly in subsequent code calls.
   - (1 pts): The completed code implements F1 metric and runs correctly in subsequent code calls.
   - (1 pts): The completed code implements Accuracy metric and runs correctly in subsequent code calls.
   - (2 pts): The completed code implements AUROC metric and runs correctly in subsequent code calls.

3. (4 pts) Due to the random nature of the K-Fold process, there is no standard answer for the performance evaluation obtained.

   - (4 pts): The completed codes implement K-folds cross validation and all metrics required and runs correctly in subsequent code calls.
   - (0 pts): The completed codes do not runs correctly.

4. (4 pts) Due to the random nature of the K-Fold process, there is no standard answer for the performance evaluation obtained.

   - (4 pts): The completed codes implement K-folds cross validation and all metrics required and runs correctly in subsequent code calls.
   - (0 pts): The completed codes do not runs correctly.

5. (1 pts) The conclusion is consistent with the performance evaluation obtained.

1. Implementing the "train_validation_split" function module by yourself. (**Note**: You cannot change the inputs (arguments) and outputs (return values) of this function to make sure that the following code part can run correctly. You should first read all the code snippets and understand them, and then try to complete this function. **Hint**: You need to split training and validation set for the two classes evenly that means contains 800 samples (Class-0 / Class-1 = 1:1) for training set and 200 samples (Class-0 / Class-1 = 1:1) for validation set in each split.)

2. Implementing the evaluation metrics by yourself including "precision, recall, F1, accuracy and auroc (area under the ROC curve)" (Note: You can use the metrics provided by Sklearn.metrics but you will lose the points for coding part.).

$$\text{F1} = \frac{2PR}{P+R} \tag{8}$$

   where P and R denote the precision and recall, respectively.

3. Fine-tuning the hyper-parameter "penalty" of logistic regression to get better performance (higher average F1 score on two classes), you can choose ["l1", "l2"] for "penalty" and use different setting in each split. Fill in the following blanks in the Table 3,4,5,6 depending on optimal hyper-parameter selection.

Hyper-parameter	1	2	3	4	5
penalty					

Table 3: The optimal setting of "penalty" of logistic regression in each split.

Metric	1	2	3	4	5	Avg
Precision						
Recall						
F1						

Table 4: The performance evaluation of logistic regression for Class-0 by Precision, Recall and F1 scores on test set.

Metric	1	2	3	4	5	Avg
Precision						
Recall						
F1						

Table 5: The performance evaluation of logistic regression for Class-1 by Precision, Recall and F1 scores on test set.

Metric	1	2	3	4	5	Avg
Accuracy						
AUROC						

Table 6: The performance evaluation of logistic regression by Accuracy and AUROC on test set.

4. Fine-tuning the hyper-parameter "C" of SVM to get better performance (higher average F1 score on two classes). To simplify this tuning process, you can choose the value of "C" only from [1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1]. you can use different setting in each split. Fill in the following blanks in the Table 7,8,9,10 depending on optimal hyper-parameter selection.

Hyper-parameter	1	2	3	4	5
C					

Table 7: The optimal setting of "C" of SVM in each split.

Metric	1	2	3	4	5	Avg
Precision						
Recall						
F1						

Table 8: The performance evaluation of SVM for Class-0 by Precision, Recall and F1 scores on test set.

Metric	1	2	3	4	5	Avg
Precision						
Recall						
F1						

Table 9: The performance evaluation of SVM for Class-1 by Precision, Recall and F1 scores on test set.

Metric	1	2	3	4	5	Avg
Accuracy						
AUROC						

Table 10: The performance evaluation of SVM by Accuracy and AUROC on test set.

5. Drawing your conclusion which model is better on the given dataset and briefly stating your reasons.