**Data Mining & Machine Learning – Assignment 4**

(20% of total grade)

**\*\*\* Important Notice \*\*\***

Handwriting is allowed ONLY in pictures and formulas (NEVER in texts).

Texts should be typeset.

You will get a 15% deduction when not sticking to this rule.

\*\*\*

\* If two homework submissions are found to be similar to each other, both submissions will receive 0.

\* Homework solutions must be submitted through Canvas. If you have multiple files, please include all files as one zip file.

\* For coding assignments, it is strongly recommended to use **Jupytor notebook** and submit **.ipynb** file.

\* Answers with **math expressions** and **graphs** can be handwritten and scanned.

\* If you find any **typo/error** in the assignment, let me know.

1. [5 pts] (Refer to p. 54 in deep learning slide) Suppose we use MSE(mean squared error) error function in this RNN. Compute the error of first step (input=’h’)

2. [3 pts] In p. 55 in deep learning slide, suppose the output of RNN is not forwarded as an input to the next step. In this case, does the RNN always generates the same output in each step? Explain in detail “**USING”** the formula of RNN node.

Yes, because every output would now only be influenced by the current input. In the formula, h\_t-1 would equate to 0, making the formula become tanh(W\_xh \* x\_t).

3. [3 pts] In semi-supervised learning, we use both labeled and unlabeled data. During the training process we repeatedly predict (and also update) the label of unlabeled data and use them in supervised learning. In this case using cross entropy error function and using KL divergence have the same result? Explain it in detail “**USING”** the formula of cross-entropy and KL divergence.

If the value of y\_i is 1, then cross-entropy and KL divergence would return the same results. The first term in the KL divergence formula would equate to 0, and the rest of the equation would be the formula for cross-entropy

4. [3 pts/ea] In sigmoid function, explain the following “**USING”** the graphs of activation functions.

1) When the initial weight values are very high, sigmoid may cause vanishing gradient. Explain it in detail

Because the denominator of the sigmoid function will continue to increase, which will continue to create a smaller and smaller number that gets closer to 0

2) When all weight values are normalized around N(0,1), entire network may become a linear model. Explain in detail.

The sigmoid graph becomes less linear the higher the positive or negative values of the inputs are (x <= -1, x >= 1). So by normalizing the weights between 0 and 1, the outputs of the sigmoid function stray from that range and are closer to 0, therefore lying on the part of the sigmoid graph that is more linearly shaped

3) In gradient descent, using tanh usually makes it converge faster than using sigmoid. Explain the reason.

The output range of tanh is [-1,1], so the presence of positive and negative outputs allows for better adjustment of weights as compared to sigmoid outputs which are only positive. The midrange of tanh has “stronger gradients”, therefore gradient descent converges faster

5. [3 pts/ea] Neural network with many hidden layers and each hidden node is using ReLU.

1) Explain why ReLU sometimes has the effect of Dropout.

Initial values of weights can be unsuitable and cause the output to be 0, and once a neuron is dead its weight has a chance of not updating anymore. A bad learning rate can also cause outputs of 0 leading to the same result

2) Explain why output values may explode (grows exponentially)

Outputs can grow exponentially if the numbers being input into ReLU are very big numbers. They are passed and continue to grow because of the multiplication of these big numbers

3) ReLU function sometime has the effect of regularization (making the model compact) especially when many input values are negative. Explain this in detail using the formula of ReLU.

The ReLU activation function works by outputting the max between 0 and the input. If many neurons are taking in negative inputs then they will be deactivated, therefore making the network more compact

6. [3 pts] If we use batch gradient with a large dataset and error function is mean squared error (not sum squared error), we may have a vanishing gradient. Explain the reason **USING** the formula of batch update gradient.

7. [4 pts] In p. 23 in “gradient-descent” slide, explain the rightmost picture.

8. [4 pts] In p. 29 in “gradient-descent” slide, explain why the method uses ‘n\_in’ parameter.

9. [4 pts] Explain why we can NOT use the gradient method in hyperparameter optimization. Explain this **USING** MSE error function with L2 regularization.

10. For the following confusion matrix

|  |  |  |
| --- | --- | --- |
|  | Pred Pos | Pred Neg |
| Actual Pos | 50 | 10 |
| Actual Neg | 5 | 100 |

1) [4 pts] compute precision, recall, specificity, f1-measure

2) [6 pts] draw a ROC graph using the following values.

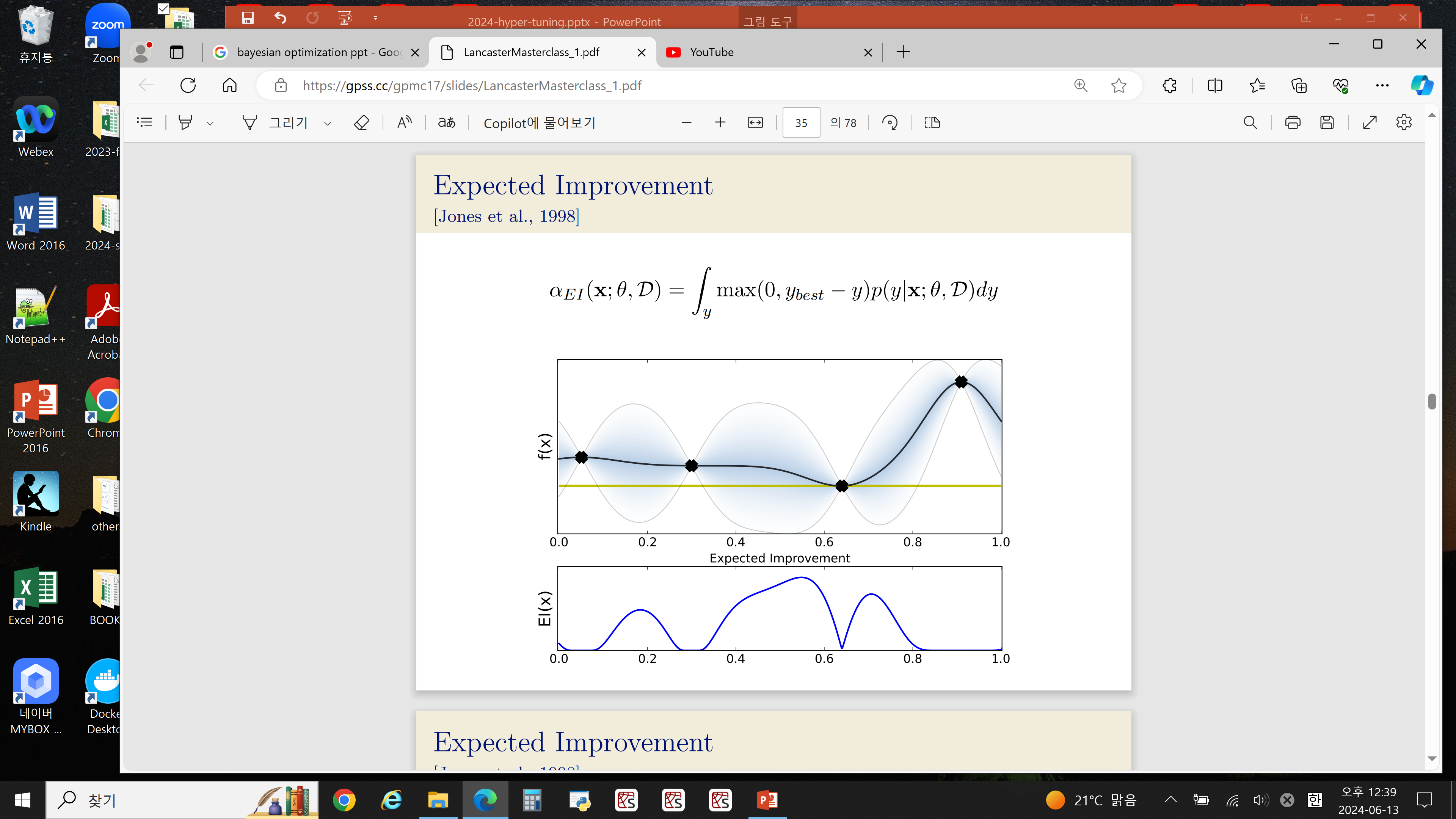
True Labels: [1, 0, 1, 0]

Predicted Probabilities: [0.8, 0.3, 0.6]

11. [8 pts] Using the following table, verify whether algo 1 and algo 2 are statistically similar or different using paired t-test.

|  |  |  |
| --- | --- | --- |
| experiments | Algo 1 | Algo 2 |
| 1 | 70 | 75 |
| 2 | 65 | 68 |
| 3 | 80 | 82 |
| 4 | 90 | 94 |
| 5 | 60 | 66 |

12. [5 pts] In the following acquisition function in Bayesian optimization, explain why the X values near X=5.5 is the best choice for the next parameter value.



13. [5 pts] In Bayesian optimization, the functions generated in Gaussian process are smooth (e.g., the functions in p. 24) Explain the reason **USING** the following kernel function.

**\* For coding assignments:**

1) Don’t change the basic program code (unless you have my permission).

2) Never use any ready-made library to implement algorithms (except for bonus questions)

3) Don’t put entire program in one cell in Jupyter. Instead, for each sub questions, show the corresponding program code in .ipynb and explain it.

4) Don’t copy from other sources

5) HWs violating these guidelines will get zero.

14. (coding) **Transfer learning VGG16 + cifar 10**

You have to install Tensorflow library first. (Tensorflow includes keras)

1) Complete the following code

from keras import datasets, layers, models

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Flatten

import matplotlib.pyplot as plt

import numpy as np

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.cifar10.load\_data()

# Creating a list of all the class labels

class\_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Converting the pixels data to float type

train\_images = train\_images.astype('float32')

test\_images = test\_images.astype('float32')

# Normalize input

train\_images = train\_images / 255

test\_images = test\_images / 255

# change target class to one hot encoding

# use ‘to\_categorical’ function

num\_classes = 10

train\_labels = OOO

test\_labels = OOO

‘’’

14. 1)-1 (2 pts)

YOUR WORK HERE

Change above train\_labels and test\_labels to one hot encoding, respectively

‘’’

X\_train=train\_images

X\_test=test\_images

Y\_train=train\_labels

Y\_test=test\_labels

‘’’

14. 1)-2 (2 pts)

YOUR WORK HERE

1) Show the shape of (train\_images, train\_labels), (test\_images, test\_labels)

2) Show 1 sample image

‘’’

# Create network model using transfer learning

# fine\_tune: # of layers to be trainable

def create\_model(input\_shape, fine\_tune=0):

# Pretrained VGG16 networks are loaded.

# include\_top is set to False, in order to exclude the model's fully-connected layers.

conv\_base = VGG16(include\_top=False,

weights='imagenet',

input\_shape=input\_shape)

# Defines how many layers trainable(updatable) during training.

# Layers in the convolutional base are set to non-trainable

# depending on the size of the fine-tuning parameter.

if fine\_tune > 0:

for layer in conv\_base.layers[:-fine\_tune]:

layer.trainable = False

else:

for layer in conv\_base.layers:

layer.trainable = False

# Create a few dense layers of the model (i.e. fully-connected layers).

# This is a new top\_model onto the pretrained layers.

flatten\_layer = Flatten()

‘’’

14. 1)-3 (4 pts)

YOUR WORK HERE

(How to define layers in Keras? Refer to https://keras.io/api/layers/)

1) Add 2 dense (fully-connected) layers: each with 20 and 10 nodes, respectively.

2) Add one output layer with 10 nodes)

‘’’

model = models.Sequential([

conv\_base,

flatten\_layer,

OOO (from 14.1)-3

OOO

OOO

])

return model

input\_shape=X\_train[0].shape

model=create\_model(input\_shape, fine\_tune=0)

print(model.summary())

‘’’

14. 1)-4 (4 pts)

YOUR WORK HERE

train the model

1) use compile and fit

2) Show accuracy of test\_images

‘’’

2) [4 pts] change the number of layers you want to freeze. Plot a graph accuracy vs number of trainable layers. What is the optimal number of layers?

3) [bonus 8 pts] Use mnist dataset as an input and repeat above process.

15. (coding) **Gradient descent (RMSProp)**

1) Implement the following RMSProp

import numpy as np

# Define the function and its gradients

def f(x, y):

return x\*\*2 + 2\*y +3\*x

# Compute the derivative with respect to x

def f\_der\_x(x):

‘’’

15. 1)-1 (2 pts)

YOUR WORK HERE

return derivative value wrt x

‘’’

# Compute the derivative with respect to y

def f\_der\_y(y):

‘’’

15. 1)-2 (2 pts)

YOUR WORK HERE

return derivative value wrt y

‘’’

# Implement RMSProp

# RMSProp parameters

# : gamma

# : lrate

# : epsilon

gamma = 0.8

lrate = 0.01

epsilon = 1e-8

# Accumulated gradient squares for x and y, respectively

# : gt\_x and gt\_y

gt\_x = 0

gt\_y = 0

# Initial values of x and y

x, y = 10.0, -10.

no\_iterations = 100

# RMSProp optimization loop

for i in range(no\_iterations):

‘’’

15. 1)-3 (8 pts)

YOUR WORK HERE

For variable x, compute f\_der\_x ( ) and gt\_x ( )

Repeat above for variable y.

Update variable x and y, respectively.

‘’’

# Show the progress

print(f"Iteration = {i + 1}: x = {x}, y = {y}, f(x, y) = {f(x, y)}")

‘’’

15. 1)-4 (2 pts)

YOUR WORK HERE

Show the final x value and f value.

‘’’

2) [bonus 4 pts] Add L2 regularization. You have to modify f, f\_der\_x, f-der\_y functions.

3) [bonus 8 pts] Implement AdaDelta. You have to define the following and modify the update rule.

16. (coding) **Bayesian optimization**

Suppose you want to optimize a hyperparameter X(e.g., learning rate). We are going to achieve it using Bayesian optimization.

1) Implement the following program code.

import numpy as np

import matplotlib.pyplot as plt

# Define the true function

# Suppose the true error function is cosine function

def true\_function(X):

return np.cos(X)

# Define the RBF (Radial Basis Function) kernel (Gaussian Kernel)

# Computes the RBF (Gaussian) kernel between two vectors X1 and X2.

# X1 : numpy array 1D (size=n)

# X2 : numpy array 1D (size=m)

# lambda() : Kernel coefficient for RBF

# returns:

# K : numpy array of shape (size of X1=n, size of X2=m).

def rbf\_kernel(X1, X2, lambda=1.0):

‘’’

16. 1)-1 (6 pts)

YOUR WORK HERE

1) Implement the following RBF kernel K

where

2) return K

‘’’

# Define the Gaussian Process Regression function

def gaussian\_regression(X\_train, y\_train, X\_test, lambda=1.0):

‘’’

16. 1)-2 (16 pts)

YOUR WORK HERE

Perform Gaussian Process Regression

Given test data X\_test, we compute and using the following formula.

: X\_train

: y\_train

: X\_test

: mean of X\_test

: Covariance of X\_test

Suppose the size of vector and vector , respectively. The shapes of matrices in above formula are as follows.

You have to the following.

1) Calculate the inverse of covariance matrix of the training data

2) Calculate the covariance between training data and test data

3) Calculate the covariance matrix of the test points

4) Compute the mean of the posterior predictive distribution

5) Compute the covariance of the posterior predictive distribution

7) return and

‘’’

# Suppose you did the following 6 experiments changing the values of ‘X\_train’.

# ‘y\_train’ is its corresponding error function value

X\_train = np.array([[1], [3], [5], [6], [7], [8]])

y\_train = true\_function(X\_train).ravel()

# Let’s estimate the error function value when x=2.2

X\_test=np.array([[2.2]])

‘’’

16. 1)-3 (6 pts)

YOUR WORK HERE

1) By using Gaussian\_regression function, estimate mean and covariance of ‘X\_test’

2) Compute standard deviation(s.d.) of X\_test values (diagonal values of covariance is variance value)

‘’’

2) [4 pts] Now change X\_test to X\_test=np.array([[3.4]]). Compute mean and s.d. of X\_test=3.4.

3) [6 pts] Now X\_test=np.array([[2.2]]) is changed as follows.

X\_test = np.linspace(0, 10, 10).reshape(-1, 1)

By using Gaussian regression, show the mean value vector and s.d. vector for X\_test, respectively.

4) [4 pts] After Q. 3) is finished, at the end of the program, run the following and show the Gaussian distribution of error function.

# Plot the results

# mean: mean value vector of X\_test

# sd: standard deviation vector X\_test

plt.figure()

plt.plot(X\_test, true\_function(X\_test), 'r:', label="True function")

plt.plot(X\_train, y\_train, 'r.', markersize=10, label="Training data")

plt.plot(X\_test, mean, 'b-', label="Prediction")

plt.fill\_between(X\_test.ravel(), mean - 1.96\*sd, mean + 1.96\*sd, alpha=0.2, color='b', label="Confidence interval")

plt.xlabel('X')

plt.ylabel('y')

plt.title('Gaussian Process Regression')

plt.legend()

plt.show()

5) [6 pts] (Lower Confidence Bound) For each value in X\_test of Q. 3), compute lower confidence bound value.

‘’’

YOUR WORK HERE

1) For each X\_test value, compute the following UCB value.

: mean

: standard deviation

2) Choose your next hyperparameter value X based on above results

‘’’