

[CS 376] Machine Learning

Assignment #1: Regression and Clustering

Joyce Jiyoung Whang, KAIST School of Computing

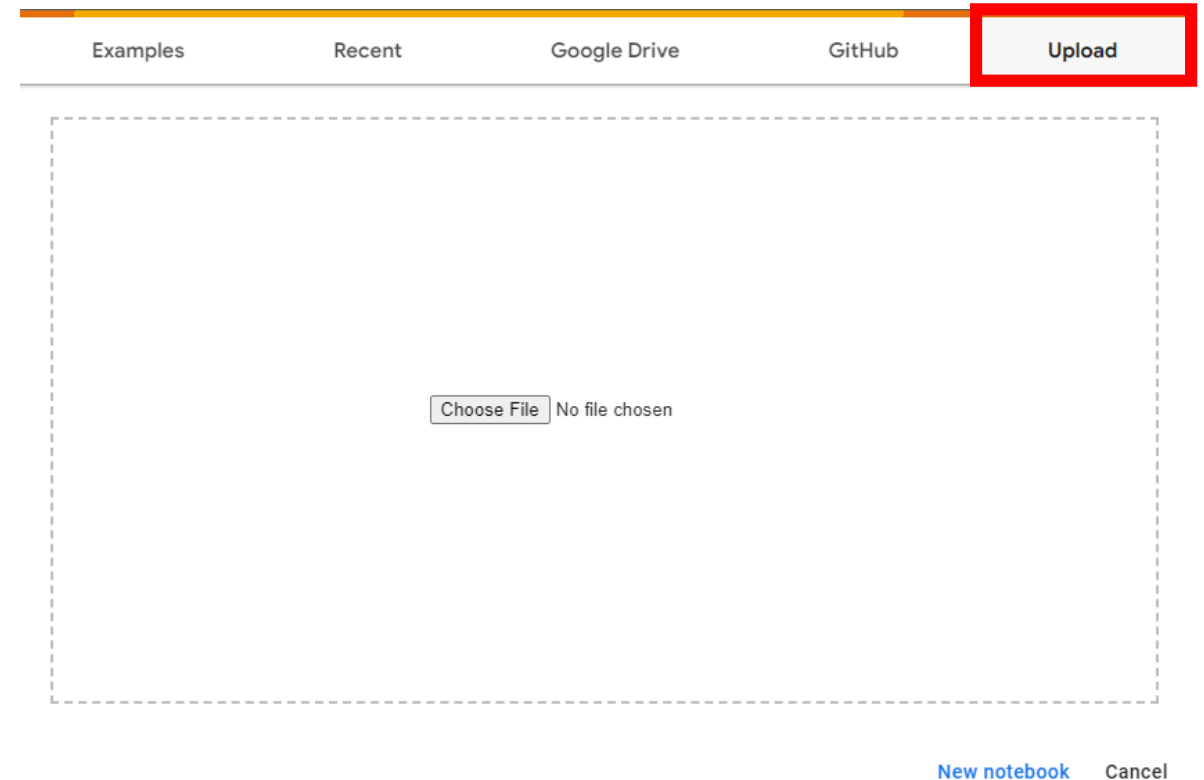
Preliminaries

Google Colab

In this assignment, we will provide you files in ipynb (Jupyter Notebook) format. You can open this file with Google Colab, which you can use freely with a Google account.

First, go to <https://colab.research.google.com>. Then, press the “Upload” tab and upload the provided ipynb file. Now, you can read and make changes to it.

A folder named Colab Notebooks will be created in your Google Drive, so you can access uploaded notebooks here.



Preliminaries

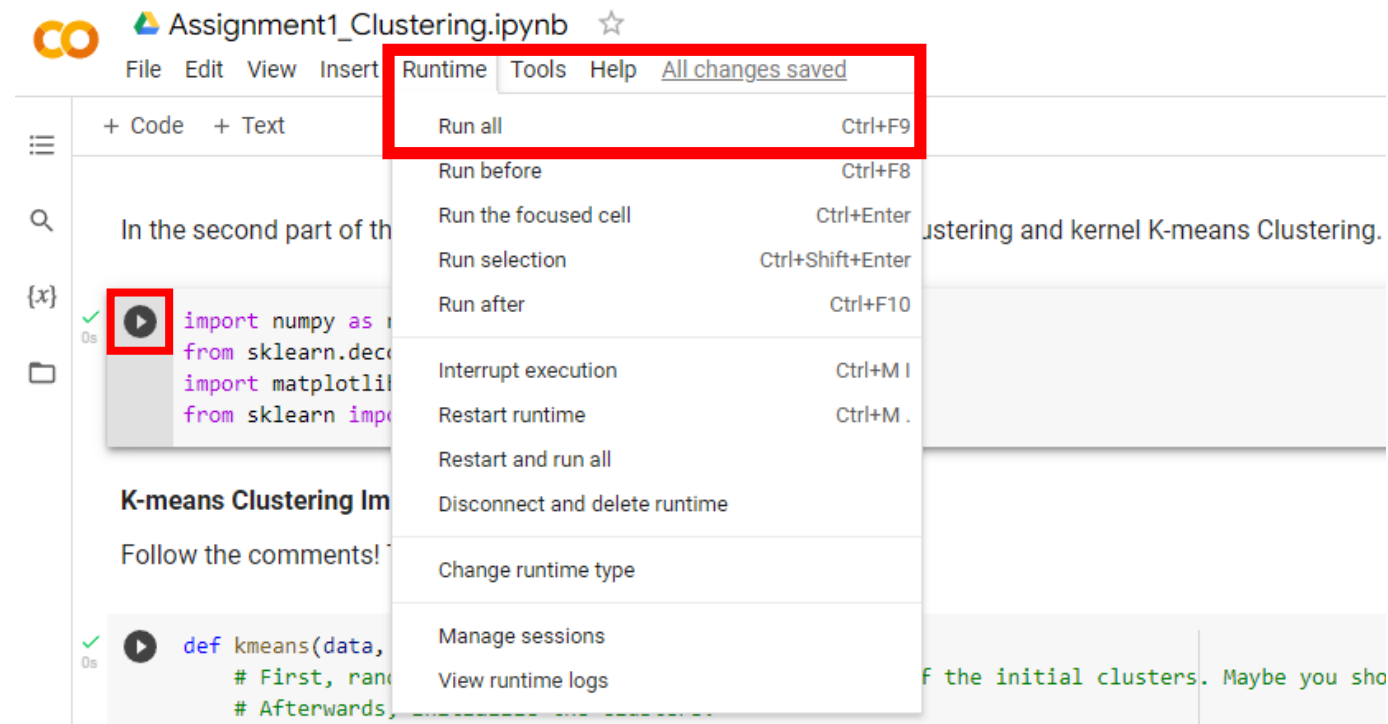
Google Colab

Here are some basic tips for running notebooks in Google Colab.

If you want to run the entire code in the notebook, you can either press Ctrl+F9, or go through the tabs “Runtime” -> “Run all”.

Blocks in notebooks are called cells. If you want to run a single cell, press Ctrl+Enter, or press the Play button, which is located at the top-left corner of each cell.

Variables in previous cells are preserved once you run them, so you do not have to rerun all cells every time you change and run a single cell.



Preliminaries

NumPy

In this assignment, you will need to handle multi-dimensional data. NumPy is a Python library for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. Here are some basic functions in NumPy that you may find useful for this assignment. For more details on these functions, check the official [NumPy documentation](#).

Notations

a, b: NumPy arrays. m, n, k: integers. np: numpy (after running the line “import numpy as np”).

np.abs(a): it returns the absolute value of an array a, element-wise.

np.arange(start=m, stop=n, step=k): it returns evenly spaced values within a given interval.

np.argmin(a, axis = None): it returns an index of the minimum element in a given array a.

np.array_equal(a, b): it checks whether given two arrays a and, b are arrays of the same values.

np.dot(a, b): it returns the dot product of the given two arrays, a and b.

np.exp(a): it performs an element-wise exponential to a given array a.

np.expand_dims(a, axis = k): it expands the shape of a given array a. Insert a new axis that will appear at position k in the expanded shape.

np.inf: floating point representation of (positive) infinity.

Preliminaries

NumPy

Notations

a, b : Numpy arrays. m, n, k : integers. np : numpy (after running the line “import numpy as np”).

np.insert(arr= a , obj= n , values= k , axis=None): it inserts values k before an index(obj) n for a given axis.

np.linalg.norm(a , axis = None): it returns matrix/vector norm of a . If axis is specified, norm will be performed along that axis.

np.min(a , axis=None): it returns the minimum of an given array a or minimum along an axis.

np.random.seed(seed= n): it reseeds the singleton RandomState instance.

np.random.uniform(low= m , high= n , size=None): it draws random samples from an uniform distribution.

np.random.normal(low= m , high= n , size=None): it draws random samples from a normal (Gaussian) distribution.

np.random.choice(a , size=None, replace=True): it generates a random sample from a given array a .

np.sign(a): it returns an element-wise indication of the sign of a number.

np.sum(a , axis = None): it adds all elements of a given array a . If axis is specified, sum is performed along that axis.

np.shape(a): it returns the shape (dimensions) of a given array a .

np.square(a): it performs element-wise square to a given array a .

np.sqrt(a): it returns the non-negative square-root of an array, element-wise.

np.tile(a , reps= m): it returns an array by repeating a given array a the number of times given reps m .

Part 1: Regression

In Part 1, you should implement the Ridge regression and the Lasso regression by following the steps below:

Step 1. Building the Regression Models.

Step 2. Generating Data.

Step 3. Changing the Hyperparameters in the Regression Models.


Step 4. Training and Testing the Regression Models.

Part 1: Regression

[Step 1] Building the Regression Models.

In this assignment, you should use only NumPy to build the Ridge and Lasso regression models.

Do not use other libraries to implement the regression model.

```
 import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.model_selection import train_test_split  
np.random.seed(123)
```

Part 1: Regression

[Step 1] Building the Regression Models.

The followings are the skeleton codes. Implement the functions inside the Regression class: the prediction function, the gradient descent, the loss function, the evaluation function using the root mean square error, and the training function.

```
# Function for training
def fit(self, X, Y, epochs, learning_rate):
    X_bar = np.insert(X, 0, 1, axis=1)
    num_examples, num_features = X_bar.shape
    # Init weights and biases
    self.W = np.zeros(num_features)

    # List for recording losses during training
    self.train_loss = []

    # Variable that supports training with Gradient Descent
    for ep in range( epochs ):
        # Get gradients
        raise NotImplementedError
        # Update weight
        raise NotImplementedError
        # Record the loss per epoch
        raise NotImplementedError
        self.train_loss.append(loss)
```

```
# Function for prediction
def predict(self, X):
    raise NotImplementedError
    return
```

```
# Function for calculating the gradient
# Objective function: PPT
def calculate_gradient(self, X, Y):
    num_examples = X.shape[0]
    # Get predictions
    raise NotImplementedError
    # Calculate gradients
    if self.ridge:
        raise NotImplementedError
    else:
        raise NotImplementedError
    raise NotImplementedError
    return
```

```
# Function for evaluation
# Metric: Root Mean Square Error (RMSE)
def eval(self, X, Y):
    X_bar = np.insert(X, 0, 1, axis=1)
    num_examples = X_bar.shape[0]
    # Get predictions
    raise NotImplementedError
    # Calculate RMSE
    raise NotImplementedError
    return
```

```
# Function for calculating the loss
# Objective function: PPT
def calculate_loss(self, X, Y):
    num_examples = X.shape[0]
    # Get predictions
    pred = self.predict(X)
    # Calculate loss
    if self.ridge:
        raise NotImplementedError
    else:
        raise NotImplementedError
    raise NotImplementedError
    return
```


Part 1: Regression

[Step 2] Generating Data.

Generate a dataset by following the provided code.

We add some Gaussian noise to data points.

```
# Generate data with gaussian noise
X = np.random.uniform(-10.0, 10.0, (10000,2))
W = np.random.uniform(-1.0, 1.0, 2)
b = np.random.uniform(-1.0,1.0, 1)
Y = np.dot(X, W) + b + np.random.normal(0,1,10000)
```

```
X_train, X_test, Y_train, Y_test = train_test_split( X, Y, test_size=0.3, random_state=35 )
X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=0.2, random_state=35)
print(X_train.shape, X_val.shape, X_test.shape, Y_train.shape, Y_val.shape, Y_test.shape)
```

```
(5600, 2) (1400, 2) (3000, 2) (5600,) (1400,) (3000,)
```

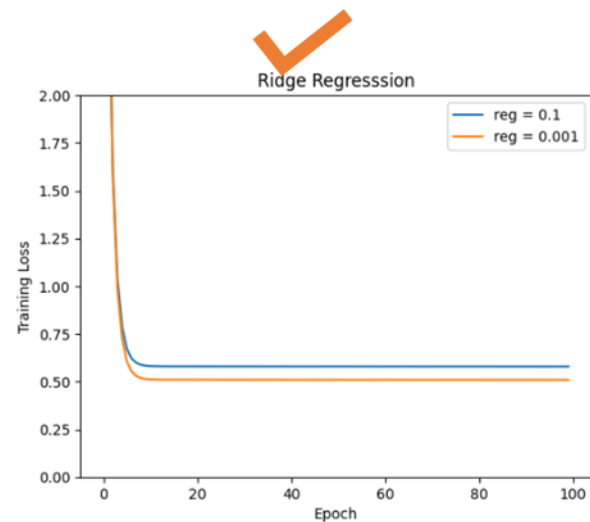
Part 1: Regression

[Step 3] Changing the Hyperparameters in the Regression Models.

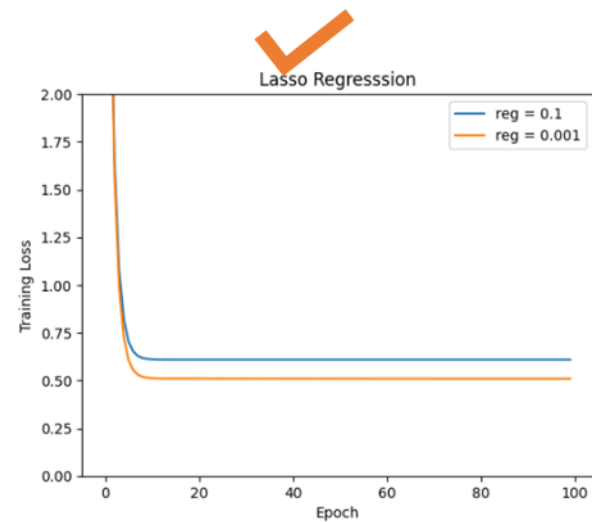
Test different regularization rates and different learning rates using the validation set.

We see that those hyperparameters significantly affect the results.

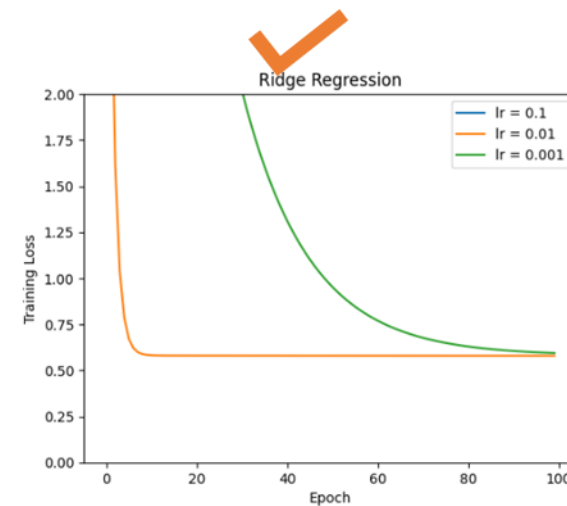
- The ridge and lasso regression models with a high learning rate fail to converge.



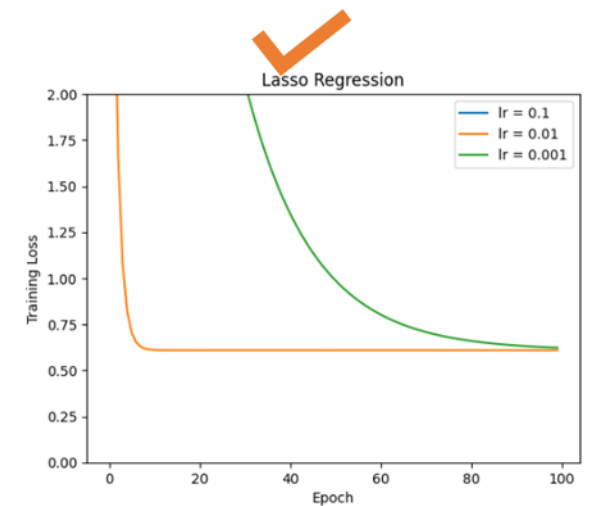
Root Mean Square Error (Validation)
reg = 0.1: 1.022682002116763
reg = 0.001: 1.020572637756531



Root Mean Square Error (Validation)
reg = 0.1: 1.0244019730663128
reg = 0.001: 1.0206026788912874



Root Mean Square Error (Validation)
lr = 0.1: 7.4648658424549265e+37
lr = 0.01: 1.022682002116763
lr = 0.001: 1.0509413104411203



Root Mean Square Error (Validation)
lr = 0.1: 3.240265283505742e+37
lr = 0.01: 1.0244019730663128
lr = 0.001: 1.0502434503799667

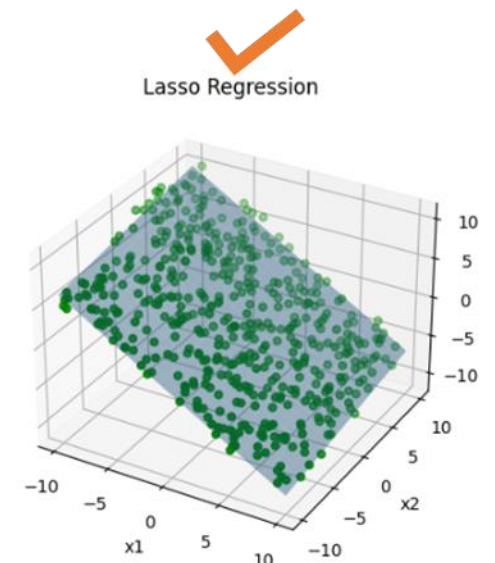
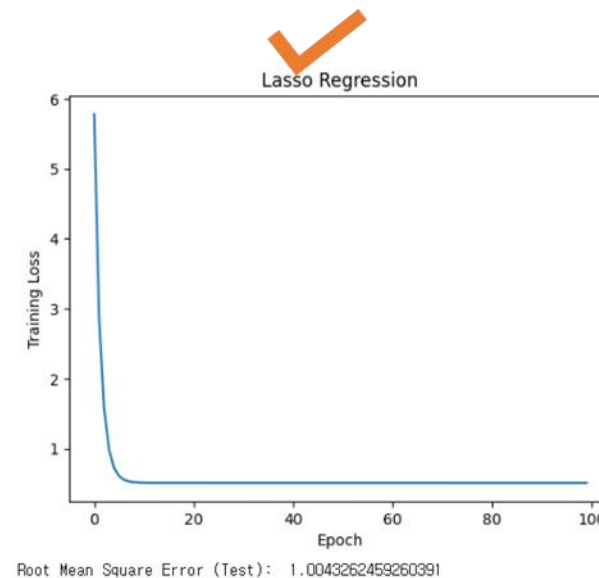
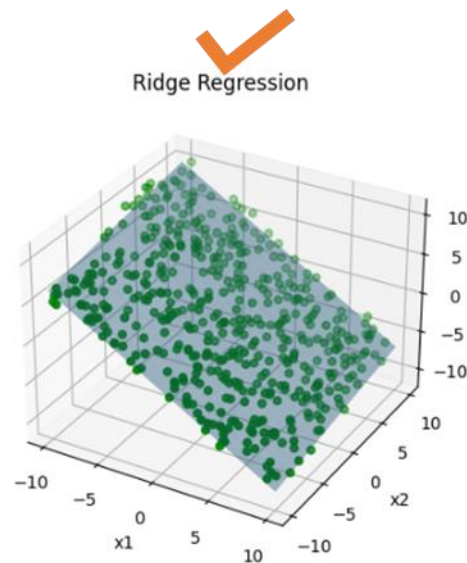
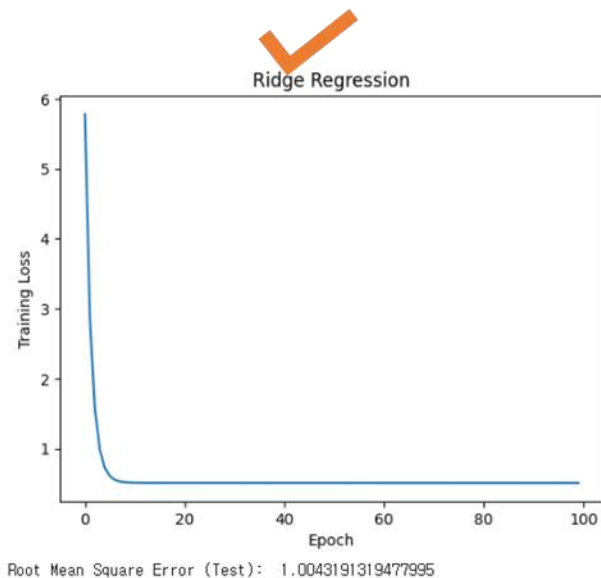
Part 1: Regression

[Step 4] Training and Testing the Regression Models.

Train the regression models, check the root mean square error, and visualize the results.

We need to train a model using the Regression class defined in Step 1.

- Set the regularization rate=0.001 and the learning rate=0.01.



Part 2: Clustering

For part 2 of this assignment, you will implement the K-means clustering and the kernel K-means clustering.

Here are the steps you should implement for the assignment.

Step 1. Implement K-means clustering.

Step 2. Implement kernel K-means clustering.

Step 3. Run K-means clustering and the kernel K-means clustering on the given examples and print the NMI score.

Step 4. Visualize the results of the K-means clustering and the kernel K-means clustering.

Part 2: Clustering

[Step 1] Implement K-means clustering.

[Step 1-1] Randomly pick k data points for the centers of the initial clusters and allocate the clusters for each data point.

[Step 1-2] Calculate the mean of each cluster.

[Step 1-3] Assign each data point to the closest cluster.

[Step 1-4] Repeatedly update the mean of each cluster and assign data points to the closest clusters until the convergence.

[Step 2] Implement the kernel K-means clustering.

[Step 2-1] Randomly pick k data points for the centers of the initial clusters and allocate the clusters for each data point.

[Step 2-2] Calculate the kernelized distance using the gaussian kernel and save the results in a matrix.

[Step 2-3] Assign each data point to the closest cluster.

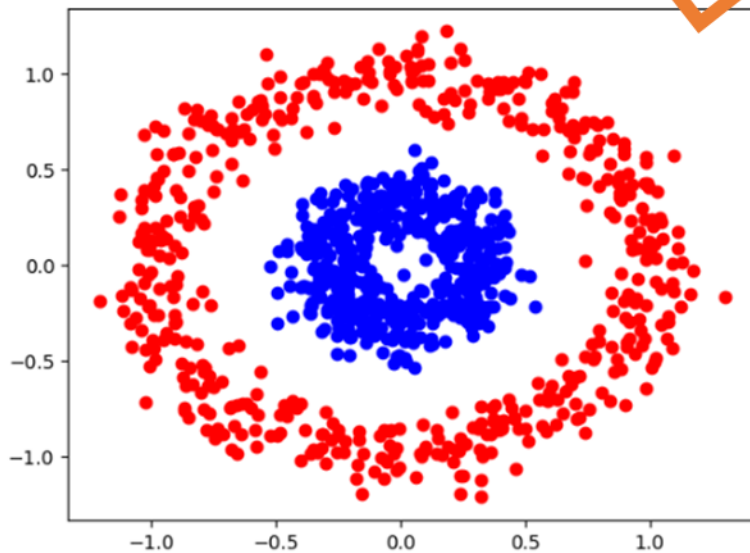
[Step 2-4] Repeatedly update the distances and the clustering assignments until the convergence.

Part 2: Clustering

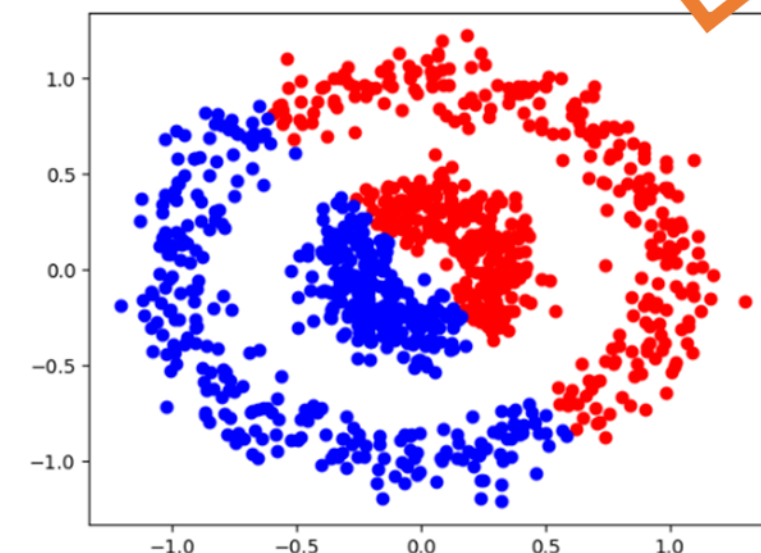
[Step 3] Run the K-means clustering and the kernel K-means clustering on the given examples.

[Step 4] Visualize the result of the K-means clustering and the kernel K-means clustering and print the NMI score.

Visualization of the given example

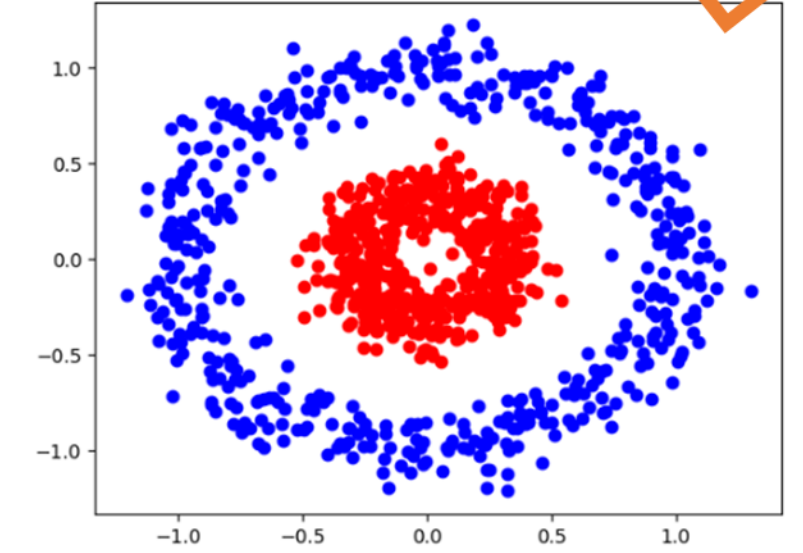


Visualization of the result of K-means clustering



NMI score of K-means clustering: 0.0004879010473852295

Visualization of the result of kernel K-means clustering



NMI score of kernel K-means clustering: 1.0

Part 2: Clustering

You are provided with skeleton code that will guide you through these steps.

For steps 3 and 4:

1. Fix the seed value of `np.random`.
 - You should use the same seed value for both steps.
2. Run each method 10 times and record the objective function values.
3. Pick the clustering result with the smallest objective function value.
4. Use that result for the final output (NMI score and visualization).

You can check that:

- The NMI score for K-means clustering is close to 0.
- The NMI score for kernel K-means clustering is close to 1.

Submission

- Take screenshots of all results for both part 1 and part 2. You should submit the results with the check mark: ✓
- You should reproduce the results by yourself. You should not submit the pictures provided in this material.
- **Make one PDF document containing all the screenshots.**
- **Deadline: Oct. 4th (Wednesday) PM 11:59. We do not accept late submissions.**
- If you have any questions, please post them on the KLMS Q&A Board.
- Good luck and have fun!