CS5691: Pattern Recognition and Machine Learning
Assignment #1

Topics: K-Nearest Neighbours, Naive Bayes, Regression **Deadline:** 28 Feb 2023, 11:55 PM

Teammate 1: (Siddhagavali Shital Bhiku) **Roll number:** ME20B166
Teammate 2: (Srinivas chowdary Ramineni) **Roll number:** ME20B174

- Please refer to the **Additional Resources** tab on the Course webpage for basic programming instructions.
- This assignment has to be completed in teams of 2. Collaborations outside the team are strictly prohibited.
- Any kind of plagiarism will be dealt with severely. These include copying text or code from any online sources. These will lead to disciplinary actions according to institute guidelines. Acknowledge any and every resource used.
- Be precise with your explanations. Unnecessary verbosity will be penalized.
- Check the Moodle discussion forums regularly for updates regarding the assignment.
- You should submit a zip file titled **'rollnumber1_rollnumber2.zip'** on Moodle where rollnumber1 and rollnumber2 are your institute roll numbers. Your assignment will **NOT** be graded if it does not contain all of the following:
  1. Type your solutions in the provided LaTeX template file and title this file as **'Report.pdf'**. **State your respective contributions at the beginning of the report clearly.** Also, embed the result figures in your LaTeX solutions.
  2. Clearly name your source code for all the programs in **individual Google Colab files**. Please submit your code only as Google Colab file (.ipynb format). Also, embed the result figures in your Colab code files.
- We highly recommend using `Python 3.6+` and standard libraries like `NumPy, Matplotlib, Pandas, Seaborn`. Please use `Python 3.6+` as the only standard programming language to code your assignments. Please note: the TAs will only be able to assist you with doubts related to Python.
- You are expected to code all algorithms from scratch. **You cannot use standard inbuilt libraries for algorithms**. Using them will result in a straight zero on coding questions, `import` wisely!
- We have provided different training and testing sets for each team. f.e. train_1 and test_1 denotes training and testing set assigned to team id 1. Use sets assigned to your team only for all questions, reporting results using sets assigned to different team will result in straight zero marks.
- Any graph that you plot is unacceptable for grading unless it labels the x-axis and y-axis clearly.

- **Please start early and clear all doubts ASAP.**
- Please note that the TAs will **only** clarify doubts regarding problem statements. The TAs won't discuss any prospective solution or verify your solution or give hints.
- Please refer to the CS5691 PRML course handout for the late penalty instruction guidelines.
- Post your doubt only on Moodle so everyone is on the same page.

---

1 Contributions

1.1 Question 1

1.1.1 Part A

- Main code: Srinivas
- Verification and changes: Shital

1.1.2 Part B

- Main code: Shital
- Verification and changes: Srinivas

1.2 Question 2

- Main code: Srinivas
- Verification and changes: Shital

1.3 Question 3

- Main code: Shital
- Confusion matrix plotting is the same as question 2
- Verification and changes: Srinivas

1.4 Latex Report

- Observations: Srinivas
- Embedding result figures and code: Shital

1. **[Regression]** You will implement linear regression as part of this question for the dataset1 provided here.

   Note that you can only regress over the points in the train dataset and you are not supposed to fit a curve on the test dataset. Whatever solution you get for the train data, you have to use that to make predictions on the test data and report results.

   (a) (2 marks) Use standard linear regression to get the best-fit curve. Split the data into train and validation sets and try to fit the model using a degree 1 polynomial then vary the degree term of the polynomial to arrive at an optimal solution.

   For this, you are expected to report the following -

   - Plot different figures for train and validation data and for each figure plot curve of obtained function on data points for various degree term of the polynomial.(refer to fig. 1.4, Pattern Recognition and Machine Learning, by Christopher M. Bishop).
   - Plot the curve for Mean Square Error(MSE) Vs degree of the polynomial for train and validation data.(refer to fig. 1.5, Pattern Recognition and Machine Learning, by Christopher M. Bishop)
   - Report the error for the best model using Mean Square Error(MSE) for train and test data provided(Use closed-form solution).
   - Scatter plot of best model output vs expected output for both train and test data provided to you.
   - Report the observations from the obtained plots.

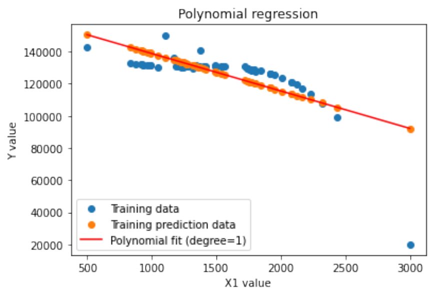
Solution:

1.4.1 Plots of train and validation data

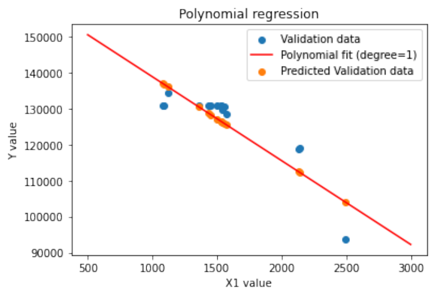
Listing 1: Code for plot train and validation data with different Degree

```
import numpy as np
import matplotlib.pyplot as plt

def polyregression(degree, n_folds=5):
    #loading data
    train_data=np.genfromtxt('train35_dataset1.csv',
        delimiter=',')
    test_data=np.genfromtxt('test35_dataset1.csv', delimiter=
        ',')
    #split into feature and target,
    #note: dropping 2nd column as discussed in class
    X, y=train_data[:,0],train_data[:,2]
    #perform k-fold cross validation
    fold_size =len(X)//n_folds
    mse_val=[]
```

```
for i in range(n_folds):
    #split data into train and validation sets
    X_val=X[i*fold_size:(i+1)*fold_size]
    X_train=np.concatenate((X[:i*fold_size],X[(i+1)*
       fold_size:]))
    y_val=y[i*fold_size:(i+1)*fold_size]
    y_train=np.concatenate((y[:i*fold_size], y[(i+1)*
       fold_size:]))
    #create polynomial features empty one
    X_train_poly=np.column_stack((np.ones(X_train.shape
       [0]), X_train))
    X_val_poly=np.column_stack((np.ones(X_val.shape[0]),
       X_val))
    for d in range(2, degree+1):
        X_train_poly=np.column_stack((X_train_poly,
           X_train**d))
        X_val_poly=np.column_stack((X_val_poly, X_val**d)
           )
    #fit linear regression model
    beta=np.linalg.inv(X_train_poly.T@X_train_poly)
    #print(beta)
    beta=beta@X_train_poly.T@y_train
    #Compute predictions on validation set
    y_val_pred=X_val_poly@beta
    #compute validation error
    mse_val.append(np.mean((y_val_pred -y_val)**2))
    #beta_values.append(list(beta))

#choose the split corresponding toh lowest validation
   error
best_fold = np.argmin(mse_val)
X_val=X[best_fold*fold_size:(best_fold+1)*fold_size]
X_train=np.concatenate((X[:best_fold*fold_size],X[(
   best_fold+1)*fold_size:]))
y_val=y[best_fold*fold_size:(best_fold+1)*fold_size]
y_train=np.concatenate((y[:best_fold*fold_size], y[(
   best_fold+1)*fold_size:]))
#create polynomial features using the best fold split
X_val_poly=np.column_stack((np.ones(X_val.shape[0]),
   X_val))
X_train_poly=np.column_stack((np.ones(X_train.shape[0]),
   X_train))
for d in range(2,degree+1):
    X_train_poly=np.column_stack((X_train_poly,X_train**d
            ))
         X_val_poly=np.column_stack((X_val_poly, X_val**d))

    #fit linear regression  closed form
    beta=np.linalg.inv(X_train_poly.T@X_train_poly)
       @X_train_poly.T@y_train
    #compute predictions on validation and test sets
    y_val_pred=X_val_poly@beta
    y_train_pred=X_train_poly@beta
    #Define range of x values
    x_range=np.linspace(min(train_data[:, 0]),max(train_data
       [:, 0]),100)
    y_pred= np.zeros_like(x_range)
    for d in range(len(beta)):
      y_pred+=beta[d]*x_range**d
    #compute MSE on validation set
    mse_val= np.mean((y_val_pred-y_val)**2)
    print(f'Validation MSE:{mse_val:.4f}')
    #pplot training data which was given
    plt.scatter(X_train,y_train,label='Training data')
    plt.scatter(X_train,y_train_pred,label='Training 
       prediction data')
    plt.xlabel("X1 value")
    plt.ylabel("Y value")
    plt.plot(x_range,y_pred,color='red',label=f'Polynomial 
       fit (degree={degree})')
    plt.legend()
    plt.title('Polynomial regression')
    plt.show()
    #plot validation sample
    plt.scatter(X_val,y_val,label='Validation data')
    plt.plot(x_range,y_pred,color='red',label=f'Polynomial 
       fit (degree={degree})')
    plt.scatter(X_val,y_val_pred,label='Predicted Validation 
       data')
    plt.xlabel("X1 value")
    plt.ylabel("Y value")
    plt.legend()
    plt.title('Polynomial regression')
    plt.show()

for degree in (1,3,9,10,20,40):
  polyregression(degree)
```

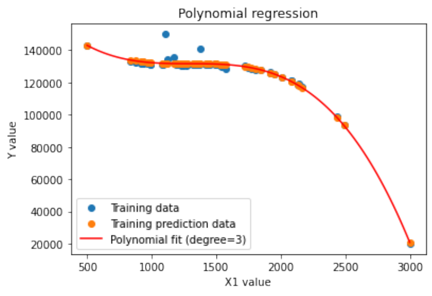


(a) Training data polynomial fit.

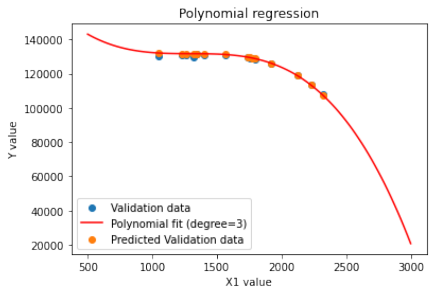


(b) Validation data polynomial fit.

Figure 1: Polynomial regression with degree 1.

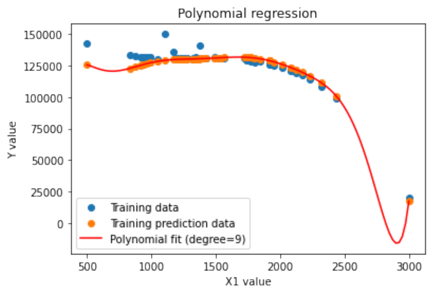


(a) Training data polynomial fit.

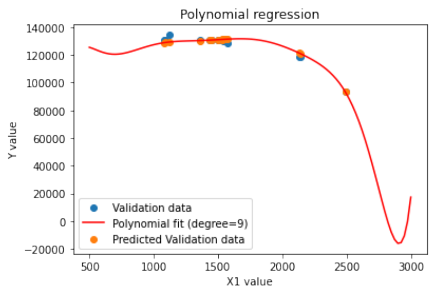


(b) Validation data polynomial fit.

Figure 2: Polynomial regression with degree 3.

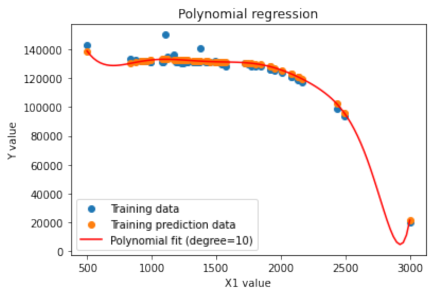


(a) Training data polynomial fit.

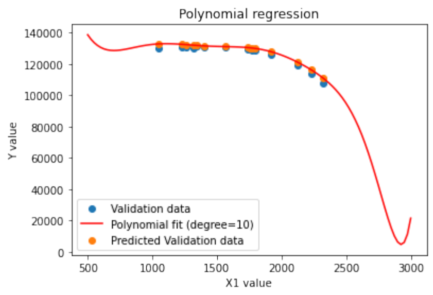


(b) Validation data polynomial fit.

Figure 3: Polynomial regression with degree 9.

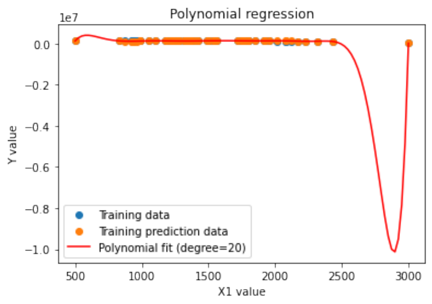


(a) Training data polynomial fit.

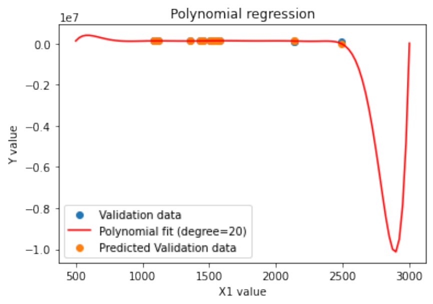


(b) Validation data polynomial fit.

Figure 4: Polynomial regression with degree 10.

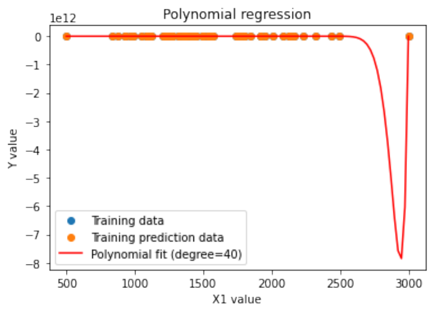


(a) Training data polynomial fit.

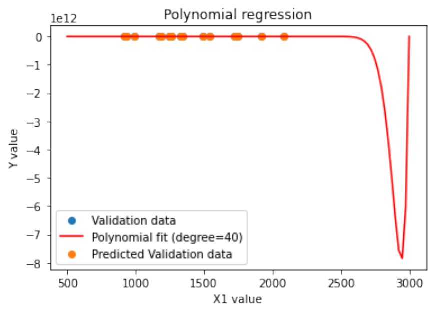


(b) Validation data polynomial fit.

Figure 5: Polynomial regression with degree 20.



(a) Training data polynomial fit.



(b) Validation data polynomial fit.

Figure 6: Polynomial regression with degree 40.

1.4.2 Mean Square Error (MSE) vs Degree

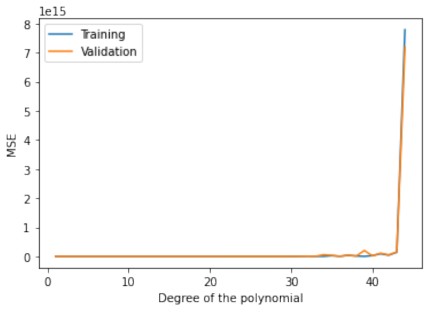
Listing 2: Code for plot MSE values of train and validation data Vs Degree

```
import numpy as np
import matplotlib.pyplot as plt
def mse_calc(degree, n_folds=5):
  #Load data
  train_data=np.genfromtxt('train35_dataset1.csv',delimiter=',')
  test_data=np.genfromtxt('test35_dataset1.csv',delimiter=',')
  X,y=train_data[:,0],train_data[:,2]
  #Perform k-fold cross validation
  fold_size=len(X)//n_folds
  mse_val=[]
  for i in range(n_folds):
      #Split data into train andvalidation sets
      X_val= X[i*fold_size:(i+1)*fold_size]
      y_val= y[i*fold_size:(i+1)*fold_size]
      X_train =np.concatenate((X[:i*fold_size],X[(i+1)*fold_size
         :]))
      y_train =np.concatenate((y[:i*fold_size],y[(i+1)*fold_size
         :]))
      #Create polynomial features
      X_train_poly =np.column_stack((np.ones(X_train.shape[0]),
         X_train))
      X_val_poly =np.column_stack((np.ones(X_val.shape[0]),X_val
         ))
      for d in range(2, degree+1):
          X_train_poly =np.column_stack((X_train_poly,X_train**d
             ))
          X_val_poly =np.column_stack((X_val_poly,X_val**d))
      #fit linear regression model closed form
      beta=np.linalg.inv(X_train_poly.T@X_train_poly)
         @X_train_poly.T@y_train
      #Compute predictions on validation set
      y_val_pred=X_val_poly@beta
      #Compute validation error
      mse_val.append(np.mean((y_val_pred - y_val)**2))
      #beta_values.append(list(beta))

  #choose beta with lowest validation error
  best_fold = np.argmin(mse_val)
  X_val = X[best_fold*fold_size:(best_fold+1)*fold_size]
  y_val = y[best_fold*fold_size:(best_fold+1)*fold_size]
  X_train = np.concatenate((X[:best_fold*fold_size], X[(
     best_fold+1)*fold_size:]))
  y_train = np.concatenate((y[:best_fold*fold_size], y[(
     best_fold+1)*fold_size:]))
  #create polynomial features using all training data and fit
      linear regression model using all training data
  X_train_poly = np.column_stack((np.ones(X_train.shape[0]),
      X_train))
  X_val_poly = np.column_stack((np.ones(X_val.shape[0]), X_val))
  for d in range(2, degree+1):
      X_train_poly = np.column_stack((X_train_poly, X_train**d))
      X_val_poly = np.column_stack((X_val_poly, X_val**d))
  beta=np.linalg.inv(X_train_poly.T@X_train_poly)@X_train_poly.
      T@y_train

  #compute predictions on validation and test sets and compute
      validation error
  y_val_pred=X_val_poly @ beta
  y_train_pred=X_train_poly @ beta
  mse_values=(np.mean((y_val_pred-y_val)**2))
  mse_train=(np.mean((y_train_pred-y_train)**2))
  return mse_train,mse_values
mse_train=[]
mse_val=[]
degrees=[]
min_mse_Degree=0
min_mse_train,min_mse_val=mse_calc(1)
for degree in range(1,45):
  a,b=mse_calc(degree)
  if b<min_mse_val:
    min_mse_val=b
    min_mse_Degree=degree
  mse_train.append(a)
  mse_val.append(b)
  degrees.append(degree)
print(mse_train)
print(mse_val)
print('mse for training data:',mse_train[min_mse_Degree-1],"
    corresponding degree:",min_mse_Degree)
print('min mse for validation data:',min_mse_val,"corresponding
    degree:",min_mse_Degree)
plt.plot(degrees,mse_train,label='Training')
plt.plot(degrees,mse_val,label='Validation')
plt.legend()
plt.xlabel('Degree of the polynomial')
plt.ylabel("MSE")
plt.show()
plt.plot(degrees,np.subtract(mse_val,mse_train),label='VAL - 
    TRAIN')
plt.legend()
plt.xlabel('Degree of the polynomial')
plt.ylabel("difference in MSE values of val and train")
plt.show()
```

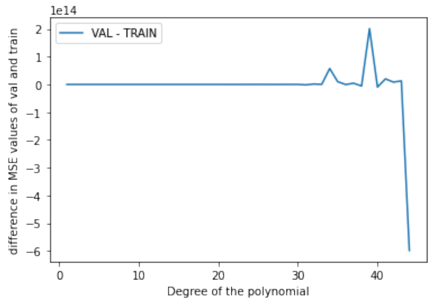
Best Degree for polynomial : 3

Value of MSE for train data corresponding to min val mse : 7479342.843100697

Minumum value of MSE for validation data : 640129.7743926719



(a) MSE of train and validation data vs Degree



(b) Difference in MSE values of val and train vs Degree

Figure 7: MSE plot for train and validation data.

1.4.3 Error for the best model and Degree

Note: By using above code you can get minimum value of MSE for train and validation data. Below you can see solution.

Best Degree for polynomial : 3

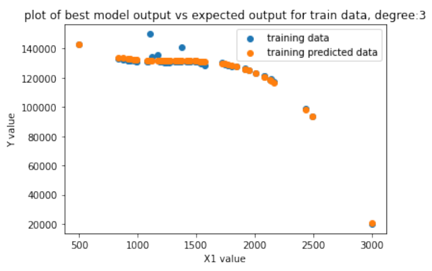
Error for the best model using Mean Square Error(MSE) for test data : 1052853.2890679438

Error for the best model using Mean Square Error(MSE) for train data : 7479342.843100697

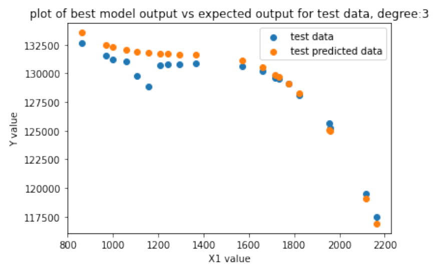
1.4.4 Scatter Plot of Best Model

Listing 3: Plot of best model output vs expected output for both train and test data

```
import numpy as np
import matplotlib.pyplot as plt
n_folds = 5
degree = 3
#Load data
train_data = np.genfromtxt('train35_dataset1.csv', delimiter=','
    )
test_data = np.genfromtxt('test35_dataset1.csv', delimiter=',')

#split into features and target
X, y = train_data[:, 0], train_data[:, 2]
X_test, y_test = test_data[:, 0], test_data[:, 2]

#perform k-fold cross validation
fold_size = len(X) // n_folds
mse_val = []
for i in range(n_folds):
    #split data into train and validation sets
    X_val = X[i*fold_size:(i+1)*fold_size]
    y_val = y[i*fold_size:(i+1)*fold_size]
    X_train = np.concatenate((X[:i*fold_size], X[(i+1)*fold_size
        :]))
    y_train = np.concatenate((y[:i*fold_size], y[(i+1)*fold_size
        :]))

    #create polynomial features
    X_train_poly = np.column_stack((np.ones(X_train.shape[0]),
        X_train))
    X_val_poly = np.column_stack((np.ones(X_val.shape[0]), X_val
        ))
    for d in range(2, degree+1):
        X_train_poly = np.column_stack((X_train_poly, X_train**d
            ))
        X_val_poly = np.column_stack((X_val_poly, X_val**d))

    #Fit linear regression model
    beta=np.linalg.inv(X_train_poly.T@X_train_poly)@X_train_poly
        .T@y_train
    #Compute predictions on validation set
    y_val_pred=X_val_poly@beta
    mse_val.append(np.mean((y_val_pred-y_val)**2))
    #beta_values.append(list(beta))

#slect beta with lowest validation error
bf=np.argmin(mse_val)
X_train=np.concatenate((X[:bf*fold_size],X[(bf+1)*fold_size:]))
y_train=np.concatenate((y[:bf*fold_size],y[(bf+1)*fold_size:]))
X_train_poly=np.column_stack((np.ones(X_train.shape[0]),X_train)
    )
X_test_poly=np.column_stack((np.ones(X_test.shape[0]),X_test))
for d in range(2,degree+1):
    X_train_poly=np.column_stack((X_train_poly, X_train**d))
    X_test_poly=np.column_stack((X_test_poly, X_test**d))
#Fit linear regression model using all training data
beta=np.linalg.inv(X_train_poly.T@X_train_poly)@X_train_poly.
    T@y_train
#Compute predictions on validation and test sets
y_train_pred =X_train_poly@beta
y_test_pred =X_test_poly@beta
#Compute validation error
plt.scatter(X_train,y_train,label='training data')
plt.scatter(X_train,y_train_pred,label='training predicted data'
    )
plt.legend()
plt.title(f'plot of best model output vs expected output for 
    train data, degree:{degree}')
plt.xlabel("X1 value")
plt.ylabel("Y value")
plt.show()
plt.scatter(X_test,y_test,label='test data')
plt.scatter(X_test,y_test_pred,label='test predicted data')
plt.title(f'plot of best model output vs expected output for 
    test data, degree:{degree}')
plt.legend()
plt.xlabel("X1 value")
plt.ylabel("Y value")
plt.show()
mse_test=(np.mean((y_test_pred-y_test)**2))
mse_train=(np.mean((y_train_pred-y_train)**2))
print("error for the best model using Mean Square Error(MSE) for
     test data :",mse_test)
print("error for the best model using Mean Square Error(MSE) for
     train data :",mse_train)
```



(a) Plot of train data and train predicted data.



(b) Plot of test data and test predicted data.

Figure 8: Best Model fit for train and test data.

Error for the best model using Mean Square Error(MSE) for test data : 1052853.2890679438

Error for the best model using Mean Square Error(MSE) for train data : 7479342.843100697

1.4.5 Observations

From the graphs for degrees : (1,3,9,10,20,40) we can see that as the degree increase the polynomial function fits the training data better The mse value for both training and validation sets first decrease reach a minimum and then shoot up degree corresponding to minimum test error:8, degree corresponding to minimum test error:3 From the plot of best model output vs expected output for train data the model fits the training data really well except for a few outliers. From the plot of best model output vs expected output for test data we observe that the function predicts the test values close to the actual values

(b) (3 marks) Split the data into train and validation sets and use ridge regression, then report for which value of lambda (λ) you obtain the best fit. For this, you are expected to report the following -

- Choose the degree from part (a), where the model overfits and try to control it using the regularization technique (Ridge regression).
- Use various choices of lambda(λ) and plot MSE test Vs lambda(λ).
- Report the error for the best model using Mean Square Error(MSE) for train and test data provided (Use closed-form solution).
- Scatter plot of best model output vs expected output for both train and test data provided to you.
- Report the observations from the obtained plots.

Solution:

1.5

1.5.1 model overfits and trying to control it using the regularization technique

In this code we have used degree = 9. which was over fitting model in part "a".

Listing 4: Code resolve overfit model using Ridge Regression Technique

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Load the data
train_data=pd.read_csv('train35_dataset1.csv')
test_data=pd.read_csv('test35_dataset1.csv')
#Split the data into features and labels
X_train,y_train=train_data.iloc[:,0].values,train_data.iloc
    [:,2].values
X_test,y_test=test_data.iloc[:,0].values,test_data.iloc[:,2].
    values
#Set the degree of the polynomial
degree=13


#Create polynomial features for input variables
X_train_poly=np.column_stack((np.ones(X_train.shape[0]),X_train)
    )
X_test_poly=np.column_stack((np.ones(X_test.shape[0]),X_test))
for d in range(2,degree+1):
    X_train_poly=np.column_stack((X_train_poly,X_train**d))
    X_test_poly=np.column_stack((X_test_poly,X_test**d))

#Define the range of lambda values to try here
lamda_values=np.linspace(-40,5,1000)
#Initialize arrays to store MSE values
mse_train_values=np.zeros((len(lamda_values),5))
mse_val_values=np.zeros((len(lamda_values),5))
least_error_fold=0
least_mse=float('inf')
#Perform 5-fold cross-validation to select the best value of
    lambda
```

```
for i in range(5):
    #Split the training data into training and validation sets
    train_indices=np.arange(len(X_train_poly))
    val_indices=np.arange(i,len(X_train_poly),5)
    train_indices=np.delete(train_indices,val_indices)
    X_train_fold,y_train_fold=X_train_poly[train_indices],
       y_train[train_indices]
    X_val_fold,y_val_fold=X_train_poly[val_indices],y_train[
       val_indices]

    #Compute MSE for each value of lambda on the training and
       validation sets
    for j,lamda in enumerate(np.exp(lamda_values)):
        I=np.identity(X_train_fold.shape[1])
        #closed form
        beta=np.linalg.inv(X_train_fold.T@X_train_fold+lamda*I)
           @ X_train_fold.T@y_train_fold
        #print(lamda)
        y_train_pred=X_train_fold@beta
        y_val_pred=X_val_fold@beta
        mse_train_values[j,i]=np.mean((y_train_pred-y_train_fold
           )**2)
        mse_val_values[j,i]=np.mean((y_val_pred-y_val_fold)**2)
        if least_mse>np.mean((y_val_pred-y_val_fold)**2):
          least_error_fold=i
#Compute the average MSE across all folds for each value of
   lambda
mse_val_mean=mse_val_values.mean(axis=1)
mse_train_mean= mse_train_values.mean(axis= 1)
#Find the best value of lambda that minimizes the validation
   error
lmda_val=np.argmin(mse_val_mean)
best_lambda= (lamda_values[lmda_val])
print("Best Lambda value: ",np.exp(best_lambda))
train_indices= np.arange(len(X_train_poly))
val_indices= np.arange(least_error_fold,len(X_train_poly),5)
train_indices=np.delete(train_indices,val_indices)
#print(train_indices)
X_train_poly,y_train=X_train_poly[train_indices],y_train[
   train_indices]
#Train the model with the best value of lambda on the full
   training set
I=np.identity(X_train_poly.shape[1])
```

```
beta=np.linalg.inv(X_train_poly.T @ X_train_poly+best_lambda*I)
   @ X_train_poly.T @ y_train
#Compute the mean squared error on the training and test sets
y_train_pred=X_train_poly@beta
mse_train=np.mean((y_train_pred - y_train)**2)
y_test_pred=X_test_poly @ beta
mse_test=np.mean((y_test_pred - y_test)**2)
X_test_poly=np.column_stack((np.ones(X_test.shape[0]),X_test))
for d in range(2,degree+1):
    X_test_poly=np.column_stack((X_test_poly,X_test**d))
mse_test_range=[]
for l in lamda_values:
  beta=np.linalg.inv(X_train_poly.T @ X_train_poly+l*I) @
     X_train_poly.T @ y_train
  y_test_pred=X_test_poly @ beta
  mse_test_range.append(np.mean((y_test_pred - y_test)**2))
#print(mse_test_range)
#Print the mean squared error on the training and test sets
print(f"Mean squared error on train set: {mse_train:.3f}")
print(f"Mean squared error on test set: {mse_test:.3f}")

#Plot the mean squared error as a function of lambda
#plt.plot((lamda_values),mse_train_mean,label='Train')
plt.plot((lamda_values),mse_test_range,label='Test data')
plt.axvline(x=best_lambda,color='r',linestyle='--',label='Best 
   lambda')
plt.xlabel('ln(Lambda)')
plt.ylabel('Mean squared error of test data')
#plt.xscale('log')
plt.title(f'Mean squared error vs lambda (degree={degree})')
plt.legend()
plt.show()
#Plot the predicted values against the true values for the
   training and test sets
plt.scatter(y_train,y_train_pred,label='Train')
plt.scatter(y_test,y_test_pred,label='Test')
plt.plot([np.min(y_train),np.max(y_train)],[np.min(y_train),np.
   max(y_train)],label='Y=X line')
plt.xlabel('True values')
plt.ylabel('Predicted values')
plt.title(f'Predicted vs true values (degree={degree},lambda={np
   .exp(best_lambda)})')
plt.legend()
plt.show()
```

1.5.2 MSE test Vs lambda

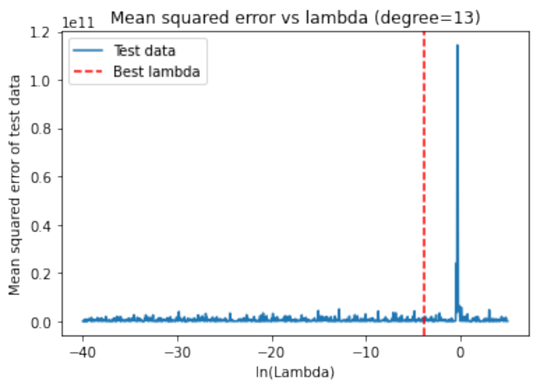


Figure 9: MSE Vs Lambda for best fit.

Best Lambda value: 0.020777723040662114

Mean squared error on train set: 73113216.119

Mean squared error on test set: 44612900.461

1.5.3 Scatter Plot of Best Model

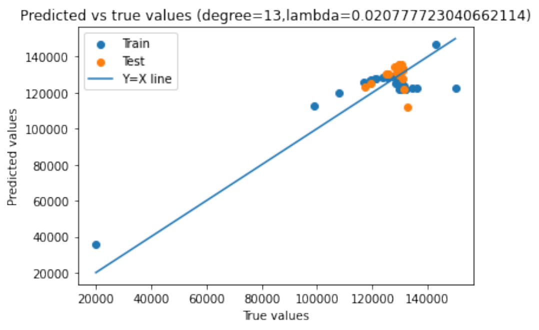


Figure 10: Scatter plot for train and test data with best fit.

1.5.4 Observations

For this part of the question, we chose degree = 13 because, from this degree onwards, the mse of the validation set is higher than that of mse of training data.

From the plot of MSE vs ln(lambda), we can see that the mse decreases and then increase.

From the plot for train and test data with best fit we can see that the predicted values and true values lie close to the Y=X line

Observations on train and test data with best fit model

For Lambda = 0.020777723040662114

Mean squared error on the train set with ridge regression: 73113216.119

Mean squared error on the test set with ridge regression: 44612900.461

Mean Square Error(MSE) for train data without ridge: 67459928.21291

Mean Square Error(MSE) for test data without ridge: 72619267.29729

The mse test decreases

The mse train increases.

shows the new model generalizes well on unseen data

2. **[Naive Bayes Classifier]** In this Question, you are supposed to build Naive Bayes classifiers for the datasets assigned to your team. Train and test datasets for each team can be found here. For each sub-question below, the report should include the following:

   - Accuracy on both train and test data.
   - Plot of the test data along with your classification boundary.
   - confusion matrices on both train and test data.

   You can refer to sample plots here and can refer Section 2.6 of "Pattern classification" book by [Duda et al. 2001] for theory.

   (a) (1 mark) Implement Naive Bayes classifier with covariance = I on dataset2. where, I denotes the identity matrix.

Solution:

1.6 Code

Listing 5: Naive Bayes classifiers Code

```
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import csv
def load_data(file_path):
    data = []
    with open(file_path,'r') as csvfile:
        csvreader = csv.reader(csvfile)
        next(csvreader)  #Skip the first row it is text
        for row in csvreader:
            data.append(row)
    return np.array(data)
def mean_and_covariance(x,y,custom_covariance=None):
    n_classes= len(np.unique(y))
    no_of_features= x.shape[1]
    means= np.zeros((n_classes,no_of_features))
    covariances =np.zeros((1, no_of_features, no_of_features))
    for c in range(1):
        xyc =x[y ==c]
        means[c] = np.mean(xyc, axis=0)
        if custom_covariance is None and c==0:
            ct =xyc-means[c]
            covariances[c] =np.dot(ct.T, ct)/(xyc.shape[0] - 1)
               +np.eye(no_of_features)*0.01
    return means,covariances
def Question2(train_x,train_y,test_x,test_y,cov_matrix):
  #function for calculating and ploting the confusion matrix
     using sns library
  def plt_confusion_matrix(y_true, y_pred, num_classes):
    k = np.zeros((num_classes, num_classes))
    #Plot the confusion matrix using sns heatmap
    for i in range(num_classes):
        for j in range(num_classes):
            k[i, j] = np.sum((y_true == i) & (y_pred == j))
    print(k)
    total = np.sum(k)
    percentages = (k / total) * 100
    new_column=[]
    for i in range(len(k)):
      new_column.append(100*k[i][i]/np.sum(k[i]))
    print(new_column)
    new_row=[]
    k=k.T
    for i in range(len(k)):
      new_row.append(100*k[i][i]/np.sum(k[i]))
    l=np.sum(new_row)/(len(new_row))
    new_row.append(np.sum(new_row)/(len(new_row)))
    k=k.T
    print(new_row)
    percentage=np.hstack((k,np.array([new_column]).T))
    percentage=np.vstack((percentage,np.array([new_row])))
    k=percentage

    #Create heatmap with annotations
    fig,ax = plt.subplots(figsize=(8,8))  #Set figsize to
        increase the size of cells
    im = ax.imshow(k,cmap='Dark2')

    #Loop over data dimensions and create text
    for i in range(len(k)):
        for j in range(len(k[0])):
            #Add confusion matrix text
            if i!=len(k)-1 and j!=len(k)-1:
              ax.text(j,i+0.15,f'{percentages[i,j]:.2f}%',ha="
                  center",va="center",color="black",fontsize=9)
              ax.text(j,i,f'{k[i,j]:}',ha="center",va="center",
                  color="black")
            else:
              ax.text(j,i,f'{k[i,j]:.1f}'+'%',ha="center",va="
                  center",color="black")
              ax.text(j,i+0.15,f'{100-k[i,j]:.1f}'+'%', ha="
                  center",va="center",color="black",fontsize=9)
    #Set ticks and axis labels
    ax.set_xticks(np.arange(len(k)-1))
    ax.set_yticks(np.arange(len(k)-1))
    ly=[]
    for a in range(len(k)-1):
      ly.append('class '+str(a))
    ax.set_xticklabels(ly)
    ax.set_yticklabels(ly)
    ax.set_xlabel('Target Class')
    ax.set_ylabel('Output Class')
    ax.xaxis.set_label_coords(0.5,-0.08)
    ax.yaxis.set_label_coords(-0.2,0.5)
    ax.grid(False)
    ax.set_title(f"Confusion Matrix with accuracy {l:1f}%")
    fig.colorbar(im)
    plt.show()

def mean_and_covariance(x,y,custom_covariance):
    num_classes= len(np.unique(y))
    num_of_features= x.shape[1]
    means= np.zeros((num_classes,num_of_features))
    covariances =np.zeros((num_classes, num_of_features,
       num_of_features))
    for c in range(num_classes):
        xyc =x[y ==c]
        means[c] = np.mean(xyc, axis=0)
        if custom_covariance is None:
            ct =xyc-means[c]
            covariances[c] =np.dot(ct.T, ct)/(xyc.shape[0] -
               1) +np.eye(num_of_features)*0.01
        else:
            covariances[c]=custom_covariance
    return means,covariances
def pdf(x,mean,covariance):
    n_features =x.shape[0]
    det =np.linalg.det(covariance)
    inv =np.linalg.inv(covariance)
    diff= x-mean
    exp= -0.5*np.dot(np.dot(diff.T, inv),diff)
    factor =1/((2 * np.pi)**(n_features/2)*np.sqrt(det))
    return factor*np.exp(exp)
def predict(x,means,covariances,priors):
    n_classes =len(priors)
    posteriors= np.zeros(n_classes)
    for c in range(n_classes):
        likelihood=pdf(x,means[c],covariances[c])
        posterior =likelihood*priors[c]
        posteriors[c]=posterior
    return np.argmax(posteriors)
def accuracy(y_true,y_pred):
    return np.mean(y_true==y_pred)


#mean and covariance matrix for each class in the training
   data
means, covariances =mean_and_covariance(train_x, train_y,
   cov_matrix)
print(covariances)
n_samples = train_x.shape[0]
#prior probability for each class
priors = np.bincount(train_y) / n_samples
#make predictionon the test data
train_predict=np.zeros(train_y.shape[0])
```

```
for i in range(train_x.shape[0]):
    x=train_x[i]
    train_predict[i]=predict(x, means,covariances,priors)
train_accuracy =accuracy(train_y,train_predict)
train_predict= np.zeros(train_y.shape[0])
for i in range(train_x.shape[0]):
    x=train_x[i]
    train_predict[i]=predict(x,means,covariances,priors)

#calculate the accuracy of the prediction
train_accuracy=accuracy(train_y,train_predict)
print(f'train accuracy:{train_accuracy:.4f}')
num_classes=len(np.unique(train_y))
print('training data confusion')
plt_confusion_matrix(train_y,train_predict,num_classes)

test_predictions =np.zeros(test_y.shape[0])
for i in range(test_x.shape[0]):
    x=test_x[i]
    test_predictions[i] =predict(x, means, covariances, priors
       )


#calculate the accuracy of the predictions
test_accuracy=accuracy(test_y,test_predictions)
test_predictions=np.zeros(test_y.shape[0])
for i in range(test_x.shape[0]):
    x=test_x[i]
    test_predictions[i]= predict(x,means,covariances,priors)

#Calculate the accuracy of the predictions
test_accuracy=accuracy(test_y,test_predictions)
print(f'test accuracy:  {test_accuracy:.4f}')
num_classes=len(np.unique(test_y))
print('test data confusion')
plt_confusion_matrix(test_y,test_predictions,num_classes)
#a meshgrid that covers the region of interest
y_min,y_max = test_x[:,1].min()-1,test_x[:,1].max()+1
x_min,x_max = test_x[:,0].min()-1,test_x[:,0].max()+1
xx,yy = np.meshgrid(np.arange(x_min,x_max,0.01),np.arange(
   y_min,y_max,0.1))
#predictions for each point in the meshgrid
Z=np.zeros(xx.shape)
for i in range(xx.shape[0]):
        for j in range(xx.shape[1]):
            x = np.array([xx[i,j], yy[i,j]])
            Z[i,j] = predict(x,means,covariances,priors)


    #Ploting the decision boundary
    plt.xlabel('x coordinate')
    plt.ylabel('y coordinate')
    plt.scatter(test_x[:,0],test_x[:,1])
    plt.title('Plot of the test data along with your
        classification boundary')
    plt.contourf(xx,yy,Z,alpha=0.1)
```

1.7 code explanation

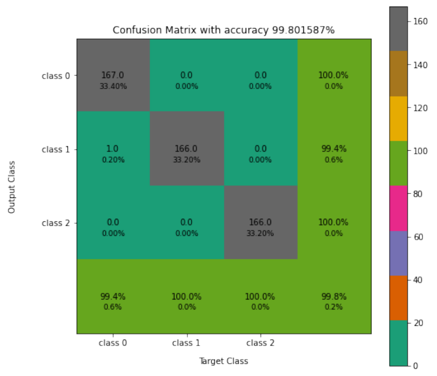
so the differentiating factor in this classification is the usage of pdf the predict function uses priors .the pdf is dependent on mean and covariance/custom covariance of classes for each part we are using different covariance matrices for same covariance we are calculating covariance matrix for one class and applying the same to all classes for different covariance we are calculating individual covariance matrix specific to each class in the end we create a mesh grid and the run the model of all points in the mesh to get the decision boundary

Listing 6: part "a" plots and values

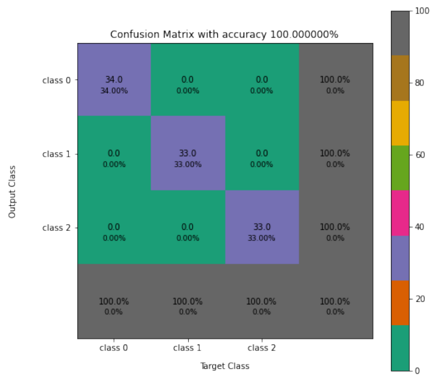
```
import csv
#Load train data
train_data=load_data('train_35_dataset2.csv')
train_x =train_data[:,:2].astype(np.float32)  #Extract x values
train_y =train_data[:, 2].astype(np.int32)  #Extract labels
#Load test data
test_data=load_data('test_35_dataset2.csv')
test_x =test_data[:,:2].astype(np.float32)  #Extract x values
test_y =test_data[:, 2].astype(np.int32)  #Extract labels
print("Part a:")
Question2(train_x,train_y,test_x,test_y,[[1,0],[0,1]])
```

Train Accuracy : 0.9980

Test Accuracy : 1.000



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 11: confusion matrix plot

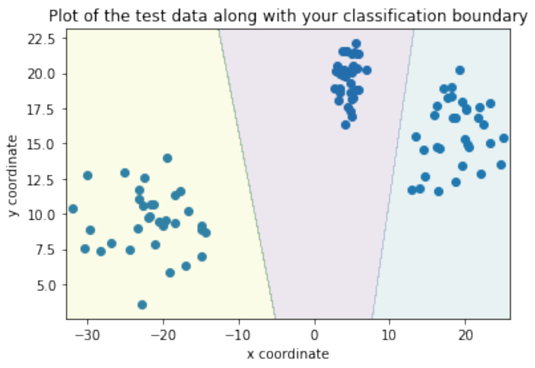


Figure 12: Classification boundary

(b) (1 mark) Implement Naive Bayes classifier with covariance = I on dataset3. where, I denotes the identity matrix.

Solution:

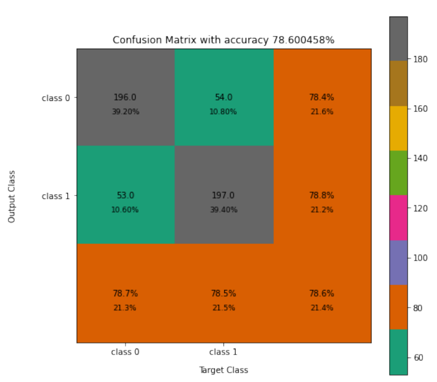
Listing 7: part "b" plots and values

```
import csv
#Load train data
train_data=load_data('train_35_dataset3.csv')
train_x =train_data[:,:2].astype(np.float32)  #Extract x values
train_y =train_data[:, 2].astype(np.int32)  #Extract labels
#Load test data
test_data=load_data('test_35_dataset3.csv')
```

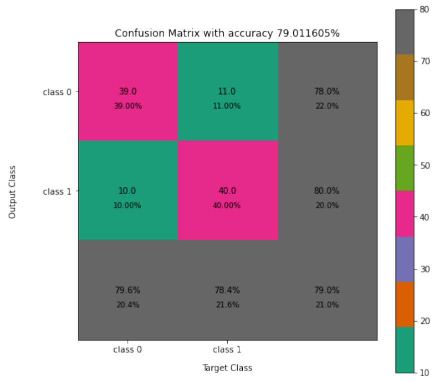
```
test_x =test_data[:,:2].astype(np.float32)  #Extract x values
test_y =test_data[:, 2].astype(np.int32)  #Extract labels
print("Part_b:")
Question2(train_x,train_y,test_x,test_y,[[1,0],[0,1]])
```

Train Accuracy : 0.7860

Test Accuracy : 0.7900



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 13: confusion matrix plot

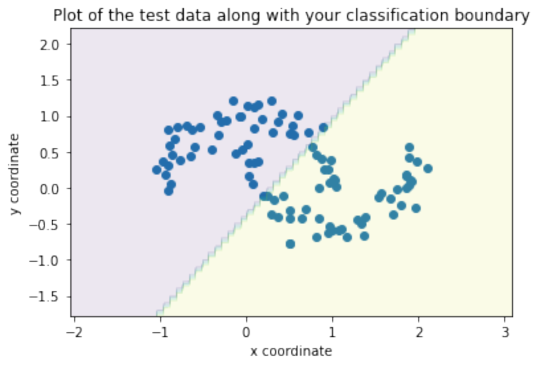


Figure 14: Classification boundary

(c) (1 mark) Implement Naive Bayes classifier with covariance same for all classes on dataset2.

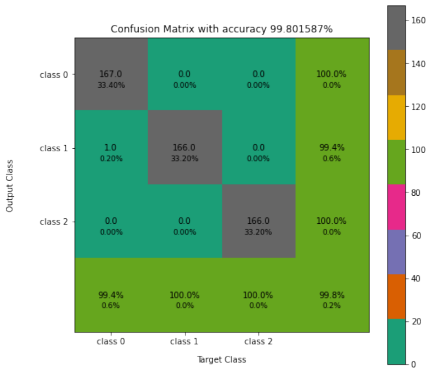
Solution:

Listing 8: part "c" plots and values

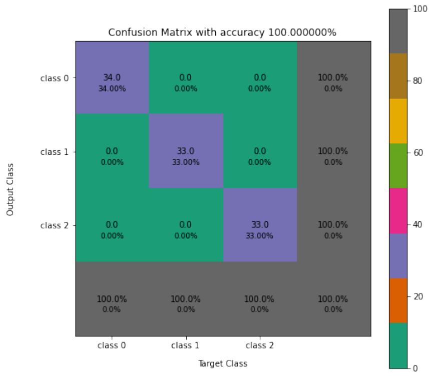
```
print("Part c:")
mean,covariance=mean_and_covariance(train_x,train_y)
#print(covariance)
Question2(train_x,train_y,test_x,test_y,covariance)
```

Train Accuracy : 0.9980

Test Accuracy : 1.0000



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 15: confusion matrix plot

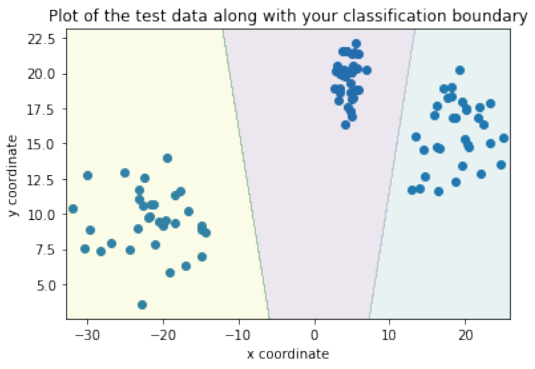


Figure 16: Classification boundary

(d) (1 mark) Implement Naive Bayes classifier with covariance same for all classes on dataset3.

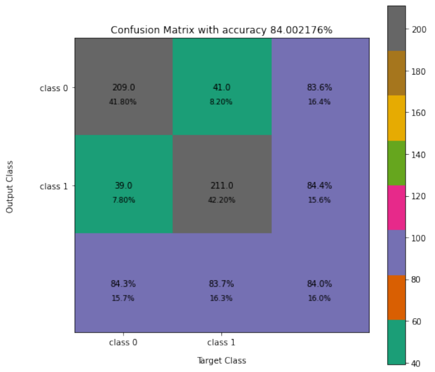
Solution:

Listing 9: part "d" plots and values

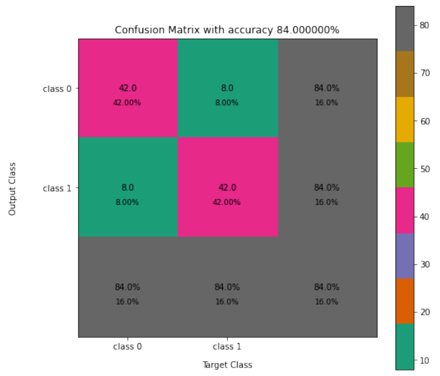
```
print("Part d:")
mean,covariance=mean_and_covariance(train_x,train_y)
#print(covariance)
Question2(train_x,train_y,test_x,test_y,covariance)
```

Train Accuracy :0.8400

Test Accuracy : 0.8400



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 17: confusion matrix plot

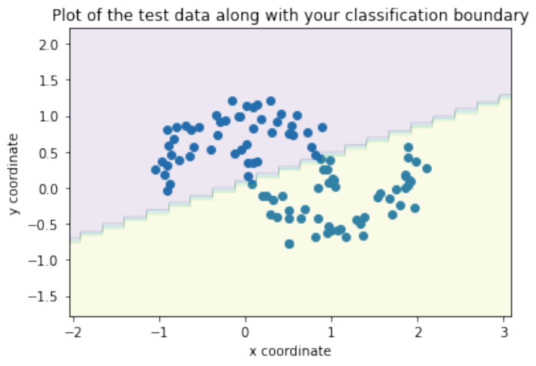


Figure 18: Classification boundary

(e) (1 mark) Implement Naive Bayes classifier with covariance different for all classes on dataset2.

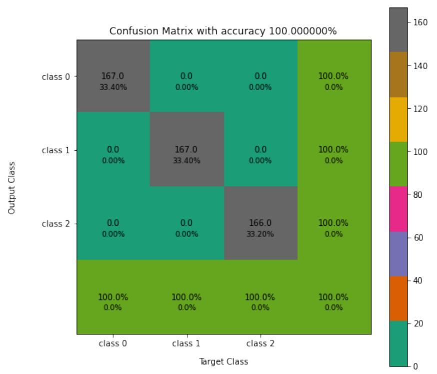
Solution:

Listing 10: part "e" plots and values

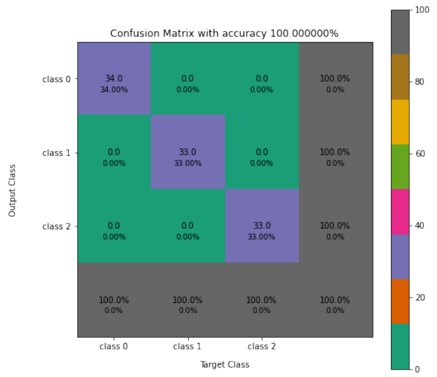
```
print("Part␣e:")
Question2(train_x, train_y, test_x, test_y, None)
```

```
Train Accuracy : 1.0000
```

```
Test Accuracy : 1.0000
```



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 19: confusion matrix plot

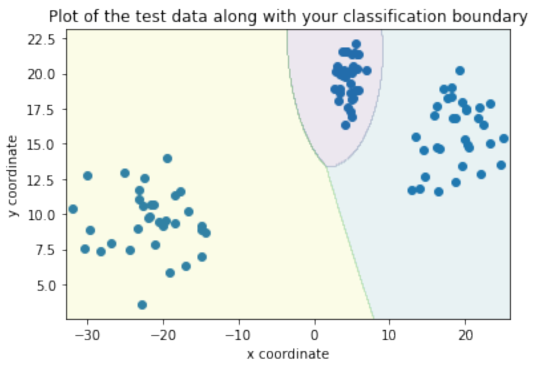


Figure 20: Classification boundary

(f) (1 mark) Implement Naive Bayes classifier with covariance different for all classes on dataset3.

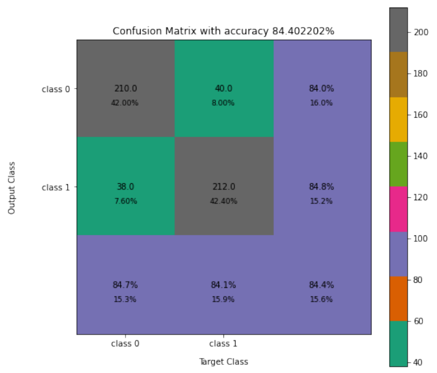
Solution:

Listing 11: part "f" plots and values

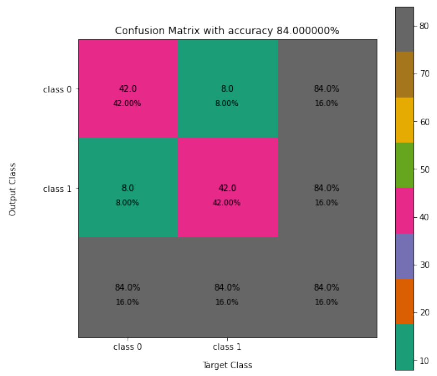
```
print("Part_f:")
Question2(train_x, train_y, test_x, test_y, None)
```

```
Train Accuracy : 0.8440
```

```
Test Accuracy : 0.8400
```



(a) confusion matrix for train data



(b) confusion matrix for test data.

Figure 21: confusion matrix plot

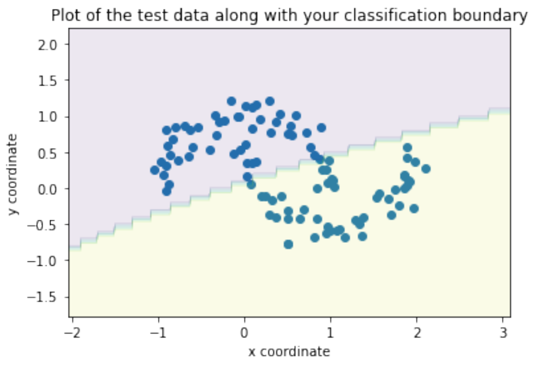


Figure 22: Classification boundary

1.8 Observations

For both dataset as the covariance matrix for each class gets closer to its actually value the classification accuracy increases

3. **[KNN Classifier]** In this Question, you are supposed to build the k-nearest neighbors classifiers on the datasets assigned to your team. Dataset for each team can be found here. For each sub-question below, the report should include the following:

- Analysis of classifier with different values of k (number of neighbors).
- Accuracy on both train and test data for the best model.
- Plot of the test data along with your classification boundary for the best model.
- confusion matrices on both train and test data for the best model.

(a) (2 marks) Implement k-nearest neighbors classifier on dataset2.

Solution:

1.9 For Dataset 2

1.9.1 code

Listing 12: K-NN classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#Load train and test data
train_df = pd.read_csv('train_35_dataset2.csv')
test_df = pd.read_csv('test_35_dataset2.csv')
#Separate the features and target variable
X_train=train_df.iloc[:,:-1].values
y_train=train_df.iloc[:,-1].values
X_test=test_df.iloc[:,:-1].values
y_test=test_df.iloc[:,-1].values
def knnclassifier(X_train,y_train,X_test,k):
    #distance between particular test point and all training
        points
    dist=np.sqrt(((X_test[:,np.newaxis,:]-X_train[np.newaxis
        ,:,:])**2).sum(axis=2))
    #indices of the k nearest neighbors for each test point
    nn_indices=np.argpartition(dist,k,axis=1)[:,:k]
    #labels of the k nearest neighbors
    nn_labels=y_train[nn_indices]
    #Predict the labels by majority
    y_pred=np.apply_along_axis(lambda x:np.bincount(x).argmax(),
       axis=1,arr=nn_labels)
    return y_pred

def calc_accuracy(y_true,y_pred):
    num_correct =np.sum(y_true == y_pred)
    num_total=y_true.shape[0]
    accuracy=num_correct / num_total
    return accuracy

def plt_confusion_matrix(y_true, y_pred, num_classes, c):
    k = np.zeros((num_classes, num_classes))
    #Plot the confusion matrix using sns heatmap
    for i in range(num_classes):
        for j in range(num_classes):
            k[i, j] = np.sum((y_true == i) & (y_pred == j))
    print(k)
    total = np.sum(k)
    percentages = (k / total) * 100
    new_column=[]
    for i in range(len(k)):
      new_column.append(100*k[i][i]/np.sum(k[i]))
    print(new_column)
    new_row=[]
    k=k.T
    for i in range(len(k)):
      new_row.append(100*k[i][i]/np.sum(k[i]))
    l=np.sum(new_row)/(len(new_row))
    new_row.append(np.sum(new_row)/(len(new_row)))
    k=k.T
    print(new_row)
    percentage=np.hstack((k,np.array([new_column]).T))
    percentage=np.vstack((percentage,np.array([new_row])))
    k=percentage

    #Create heatmap with annotations
    fig,ax = plt.subplots(figsize=(8,8))  #Set figsize to
       increase the size of cells
    im = ax.imshow(k,cmap='Dark2')
    #Loop over data dimensions and create text
    for i in range(len(k)):
        for j in range(len(k[0])):
            #Add confusion matrix text
            if i!=len(k)-1 and j!=len(k)-1:
              ax.text(j,i+0.15,f'{percentages[i,j]:.2f}%',ha="
                 center",va="center",color="black",fontsize=9)
              ax.text(j,i,f'{k[i,j]:}',ha="center",va="center",
                 color="black")
            else:
              ax.text(j,i,f'{k[i,j]:.1f}'+'%',ha="center",va="
                 center",color="black")
              ax.text(j,i+0.15,f'{100-k[i,j]:.1f}'+'%', ha="
                 center",va="center",color="black",fontsize=9)
    #Set ticks and axis labels
    ax.set_xticks(np.arange(len(k)-1))
    ax.set_yticks(np.arange(len(k)-1))
    ax.set_xticklabels(['class1','class2','class3'])
    ax.set_yticklabels(['class1','class2','class3'])
    ax.set_xlabel('Target Class')
    ax.set_ylabel('Output Class')
    ax.xaxis.set_label_coords(0.5,-0.08)
    ax.yaxis.set_label_coords(-0.2,0.5)
    ax.grid(False)
    ax.set_title(f"Confusion Matrix with accuracy {l:1f}%")
    fig.colorbar(im)
    plt.show()



#Define the K values to be tested
k_values=list(range(1, 400))
#Define empty lists to store train and test accuracy values for
   each k value
train_acc=[]
test_acc=[]

#Loop through each k value and calculate train and test accuracy
for k in k_values:
    #Train the KNN classifier on the training data
    ypredtrain=knnclassifier(X_train,y_train,X_train,k)
    #Calculate train accuracy
    acc_train=calc_accuracy(y_train,ypredtrain)
    train_acc.append(acc_train)
    #Predict on the test data using the trained classifier
    y_pred_test=knnclassifier(X_train,y_train,X_test,k)
    #Calculate test accuracy
    acc_test=calc_accuracy(y_test, y_pred_test)
    test_acc.append(acc_test)

#Plot the train and test accuracy versus k values
plt.plot(k_values, train_acc, label='Train Accuracy')
plt.plot(k_values, test_acc, label='Test Accuracy')
plt.title('Accuracy vs K Values')
plt.xlabel('K Values')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
#Print the best K value based on the test accuracy
best_k = k_values[np.argmax(train_acc)]
print('Best K value: ', best_k)
k = best_k

#number of unique classes
num_classes = len(np.unique(y_train))

#Initialize lists to store the training and test accuracies for
   each k value
train_acc_1 =[]
test_acc_1 =[]
#predict on the training data
y_train_pred=knnclassifier(X_train, y_train, X_train, k)
#predict on the test data
y_test_pred=knnclassifier(X_train, y_train, X_test, k)
#Compute the training and test accuracies
train_acc=calc_accuracy(y_train, y_train_pred)
test_acc=calc_accuracy(y_test, y_test_pred)
#Store the accuracies in the lists
train_acc_1.append(train_acc)
test_acc_1.append(test_acc)
#Plot the test data with the decision boundary for the current k
    value
plt.figure()
sns.scatterplot(x=X_test[:, 0],y=X_test[:, 1],hue=y_test)
sns.despine()
plt.title(f'k ={k}')
#grid of points to plot the decision boundary
x_min,x_max=X_test[:,0].min()-1, X_test[:,0].max() +1
```

```
y_min,y_max=X_test[:,1].min()-1, X_test[:,1].max() +1
xx,yy=np.meshgrid(np.arange(x_min,x_max,0.1),np.arange(y_min,
    y_max,0.1))
Z=knnclassifier(X_train,y_train,np.c_[xx.ravel(),yy.ravel()],k)
Z = Z.reshape(xx.shape)
plt.xlabel('X coordinate')
plt.ylabel('Y coordinate')
#Plot the decision boundary
plt.contourf(xx,yy,Z,alpha=0.5)
plt.show()
#Calculate the confusion matrices for the current k value
plt_confusion_matrix(y_train,y_train_pred,num_classes,'training 
    data')
plt_confusion_matrix(y_test,y_test_pred,num_classes,'test data')
#Print the accuracies for the current k value
print("accuracy on test data:", test_acc)
print("accuracy on train data: ", train_acc)
```

1.9.2 Analysis of classifier with different values of k

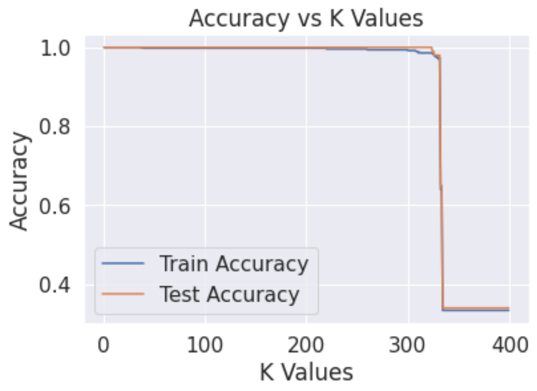


Figure 23: Analaysis of k (accuracy Vs K-values)

1.9.3 Accuracy on both train and test data for the best model

Train Accuracy : 1.0

Test Accuracy: 1.0

1.9.4 Plot of the test data along with our classification boundary for the best model

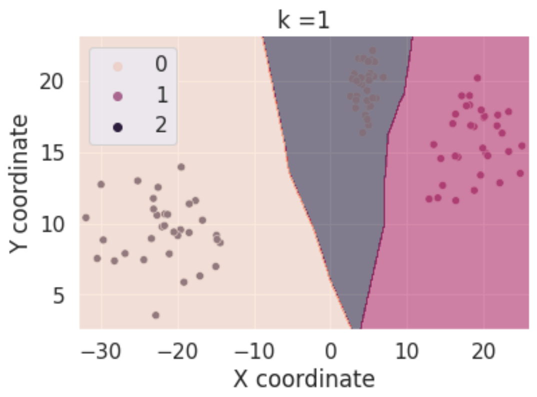
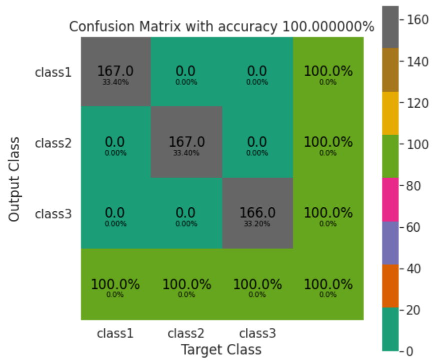
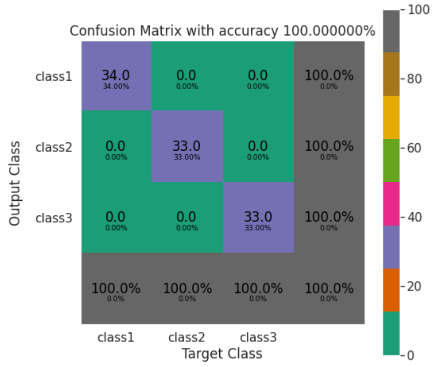


Figure 24: Classification boundry for dataset 2.

1.9.5 confusion matrices on both train and test data for the best model



(a) Confusion matrix for train data



(b) Confusion matrix for test data

Figure 25: Confusion matrices for dataset 2

(b) (2 marks) Implement k-nearest neighbors classifier on dataset3.

Solution:

1.10 For Dataset 3

1.10.1 code

Listing 13: K-NN classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

#train and test data loading
train_df = pd.read_csv('train_35_dataset3.csv')
test_df = pd.read_csv('test_35_dataset3.csv')
#Separate the features and target variable
X_train=train_df.iloc[:,:-1].values
y_train=train_df.iloc[:,-1].values
X_test=test_df.iloc[:,:-1].values
y_test=test_df.iloc[:,-1].values
def knnclassifier(X_train,y_train,X_test,k):
    #distance between particular test point and all training
        points
    dist=np.sqrt(((X_test[:,np.newaxis,:]-X_train[np.newaxis
        ,:,:])**2).sum(axis=2))
    #indices of the k nearest neighbors for each test point
    nn_indices=np.argpartition(dist,k,axis=1)[:,:k]
    #labels of the k nearest neighbors
    nn_labels=y_train[nn_indices]
    #predict the labels by majority
    y_pred=np.apply_along_axis(lambda x:np.bincount(x).argmax(),
        axis=1,arr=nn_labels)
    return y_pred

def calc_accuracy(y_true,y_pred):
    num_correct =np.sum(y_true == y_pred)
    num_total=y_true.shape[0]
    accuracy=num_correct / num_total
    return accuracy

def plt_confusion_matrix(y_true, y_pred, num_classes, c):
    k = np.zeros((num_classes, num_classes))
    #plot the confusion matrix using sns heatmap
    for i in range(num_classes):
        for j in range(num_classes):
            k[i, j] = np.sum((y_true == i) & (y_pred == j))
```

```
print(k)
total = np.sum(k)
percentages = (k / total) * 100
new_column=[]
for i in range(len(k)):
  new_column.append(100*k[i][i]/np.sum(k[i]))
print(new_column)
new_row=[]
k=k.T
for i in range(len(k)):
  new_row.append(100*k[i][i]/np.sum(k[i]))
l=np.sum(new_row)/(len(new_row))
new_row.append(np.sum(new_row)/(len(new_row)))
k=k.T
print(new_row)
percentage=np.hstack((k,np.array([new_column]).T))
percentage=np.vstack((percentage,np.array([new_row])))
k=percentage

#create heatmap with annotations
fig,ax = plt.subplots(figsize=(8,8))  #Set figsize to
    increase the size of cells
im = ax.imshow(k,cmap='Dark2')

#loop over data dimensions and create text
for i in range(len(k)):
    for j in range(len(k[0])):
        #Add confusion matrix text
        if i!=len(k)-1 and j!=len(k)-1:
          ax.text(j,i,f'{k[i,j]:}',ha="center",va="center",
              color="black")
          ax.text(j,i+0.15,f'{percentages[i,j]:.2f}%',ha="
              center",va="center",color="black",fontsize=9)
        else:
          ax.text(j,i,f'{k[i,j]:.1f}'+'%',ha="center",va="
              center",color="black")
          ax.text(j,i+0.15,f'{100-k[i,j]:.1f}'+'%', ha="
              center",va="center",color="black",fontsize=9)
ax.set_xticks(np.arange(len(k)-1))
ax.set_yticks(np.arange(len(k)-1))
ax.set_xticklabels(['class1','class2'])
ax.set_yticklabels(['class1','class2'])
ax.set_xlabel('Target Class')
ax.set_ylabel('Output Class')
```

```
    ax.xaxis.set_label_coords(0.5,-0.08)
    ax.yaxis.set_label_coords(-0.2,0.5)
    ax.grid(False)
    ax.set_title(f"Confusion Matrix with accuracy {l:1f}%")
    fig.colorbar(im)
    plt.show()



#define the K values to be tested
k_values=list(range(1, 400))
#define empty lists to store train and test accuracy values for
   each k value
train_acc=[]
test_acc=[]

#Loop through each k value, calculate train and test accuracy
for k in k_values:
    #Train the KNN classifier on the training data
    ypredtrain=knnclassifier(X_train,y_train,X_train,k)
    #Calculate train accuracy
    acc_train=calc_accuracy(y_train,ypredtrain)
    train_acc.append(acc_train)
    #Predict on the test data using the trained classifier
    y_pred_test=knnclassifier(X_train,y_train,X_test,k)
    #Calculate test accuracy
    acc_test=calc_accuracy(y_test, y_pred_test)
    test_acc.append(acc_test)

#plot the train and test accuracy versus k values
plt.plot(k_values, train_acc, label='Train Accuracy')
plt.plot(k_values, test_acc, label='Test Accuracy')
plt.title('Accuracy vs K Values')
plt.xlabel('K Values')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
#print the best K value based on the test accuracy
best_k = k_values[np.argmax(train_acc)]
print('Best K value: ', best_k)
k = best_k

#number of unique classes
num_classes = len(np.unique(y_train))
```

```
#Initialize lists to store the training and test accuracies for
    each k value
train_acc_1 =[]
test_acc_1 =[]
#predict on the training data
y_train_pred=knnclassifier(X_train, y_train, X_train, k)
#predict on the test data
y_test_pred=knnclassifier(X_train, y_train, X_test, k)
#compute the training and test accuracies
train_acc=calc_accuracy(y_train, y_train_pred)
test_acc=calc_accuracy(y_test, y_test_pred)
train_acc_1.append(train_acc)
test_acc_1.append(test_acc)
#plot the test data with the decision boundary for the current k
     value
plt.figure()
sns.scatterplot(x=X_test[:, 0],y=X_test[:, 1],hue=y_test)
sns.despine()
plt.title(f'k = {k}')
#grid of points to plot the decision boundary
x_min,x_max=X_test[:,0].min()-1, X_test[:,0].max() +1
y_min,y_max=X_test[:,1].min()-1, X_test[:,1].max() +1
xx,yy=np.meshgrid(np.arange(x_min,x_max,0.1),np.arange(y_min,
    y_max,0.1))
Z=knnclassifier(X_train,y_train,np.c_[xx.ravel(),yy.ravel()],k)
Z = Z.reshape(xx.shape)
plt.xlabel('X coordinate')
plt.ylabel('Y coordinate')
plt.contourf(xx,yy,Z,alpha=0.5)
plt.show()
#calculate the confusion matrices for the current k value
plt_confusion_matrix(y_train,y_train_pred,num_classes,'training 
    data')
plt_confusion_matrix(y_test,y_test_pred,num_classes,'test data')
#print the accuracies for the current k value
print("accuracy on test data:", test_acc)
print("accuracy on train data: ", train_acc)
```

1.10.2 Analysis of classifier with different values of k

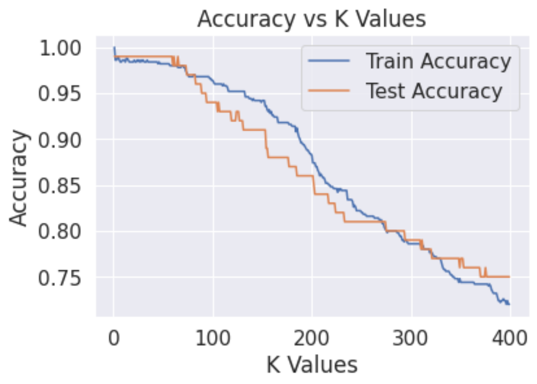


Figure 26: Analaysis of k (accuracy Vs K-values)

1.10.3 Accuracy on both train and test data for the best model

Train Accuracy : 1.0

Test Accuracy: 0.99

1.10.4 Plot of the test data along with our classification boundary for the best model

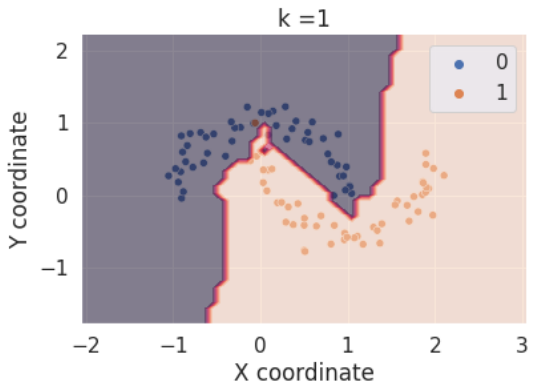
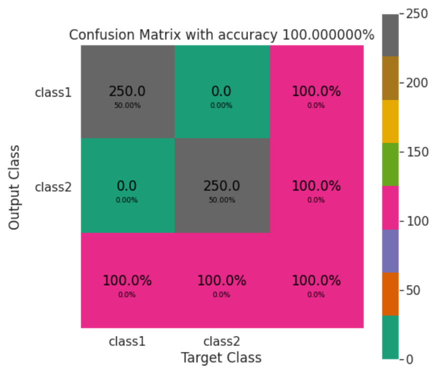
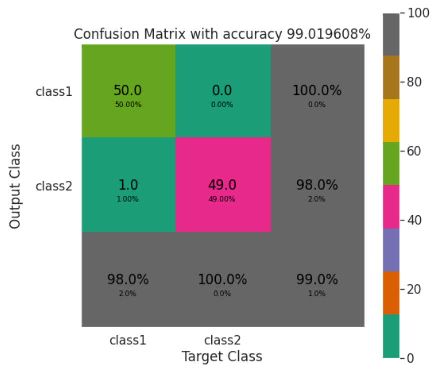


Figure 27: Classification boundry for dataset 3.

1.10.5 confusion matrices on both train and test data for the best model



(a) Confusion matrix for train data



(b) Confusion matrix for test data

Figure 28: Confusion matrices for dataset 3