DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

Exam 1

submitted to

Professor Joseph Picone
ECE 8527: Introduction to Pattern Recognition and Machine Learning
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

February 23, 2024

prepared by:

Leo Berman
Email: leo.berman@temple.edu

A. P[E] VS ALPHA (PRIORS EQUAL)

This is the visualization of the data and the visualization of QDA decision surfaces where alpha dictates the position of the bottome left corner of a data set that is 1x1: This is the plot of the error rates as a function

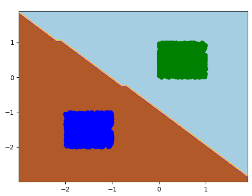


Figure 1: Alpha = -2

;

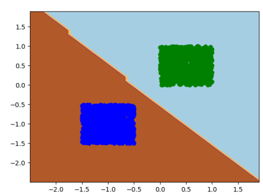


Figure 2: Alpha = -1.42

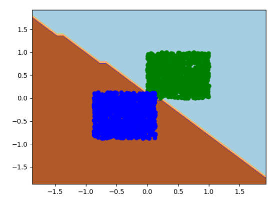


Figure 3: Alpha = -0.85

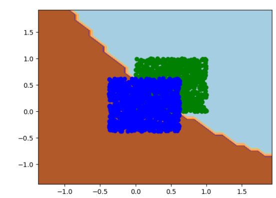


Figure 4: Alpha = -0.28

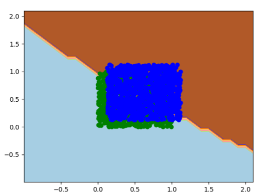


Figure 5: Alpha = 0.28

;

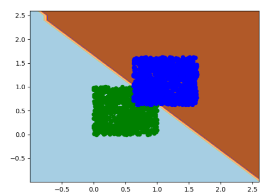


Figure 6: Alpha = 0.85

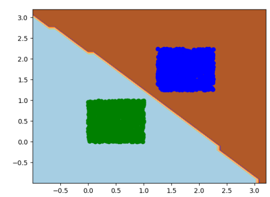


Figure 7: Alpha = 1.42

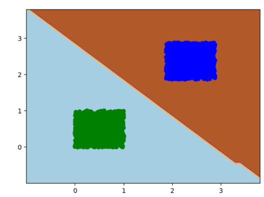


Figure 8: Alpha = 2

of alpha.

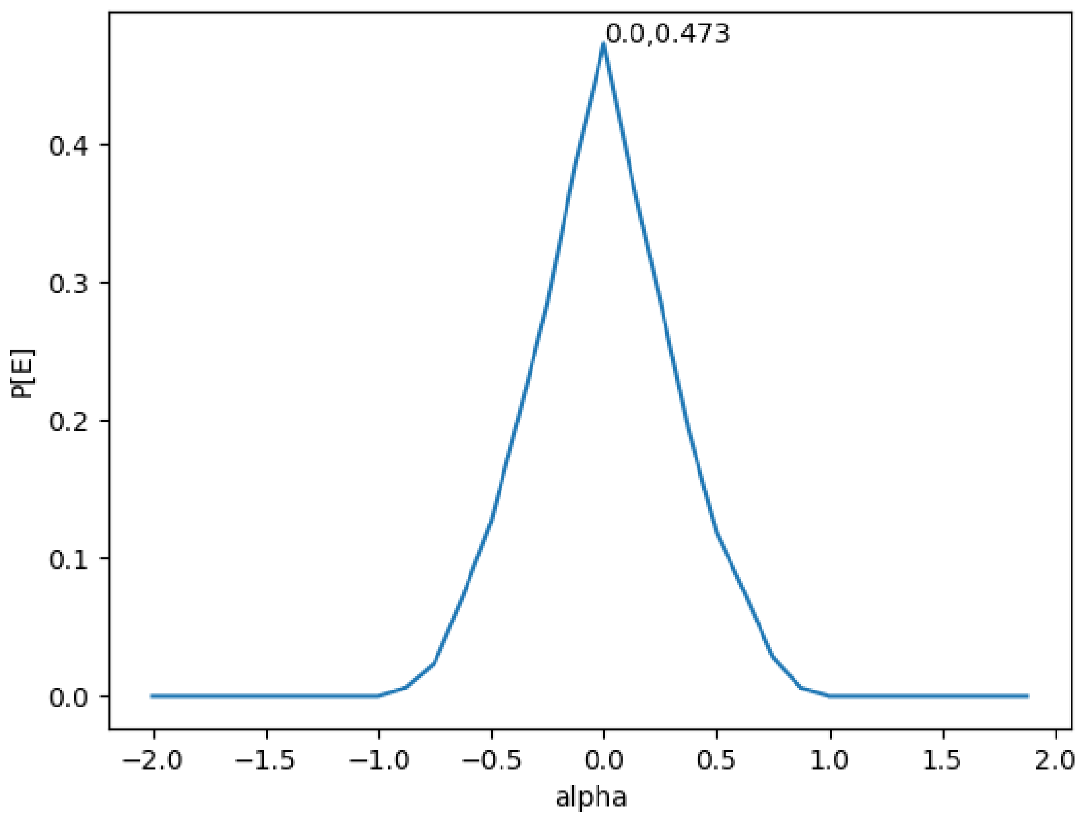
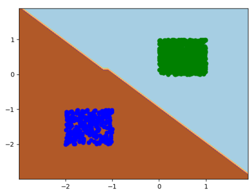


Figure 9: P[E] vs Alpha (Priors Equal)

B. P[E] VS ALPHA (PRIORS UNEQUAL)

This is the visualization of the data and the visualization of QDA decision surfaces where alpha dictates the position of the bottome left corner of a data set that is 1x1:



;

Figure 10: Alpha = -2

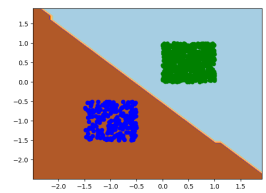


Figure 11: Alpha = -1.42

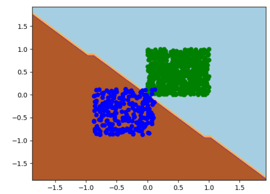


Figure 12: Alpha = -0.85

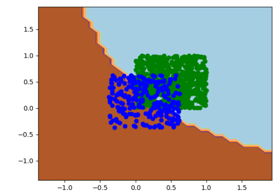
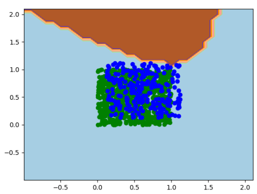


Figure 13: Alpha = -0.28



;

Figure 14: Alpha = 0.28

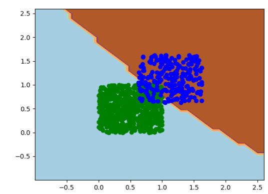


Figure 15: Alpha = 0.85

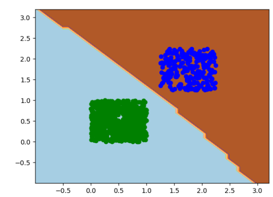


Figure 16: Alpha = 1.42

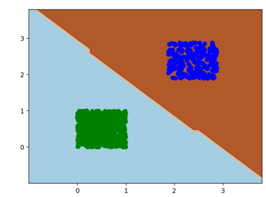


Figure 17: Alpha = 2

This is the plot of the error rates as a function of alpha.

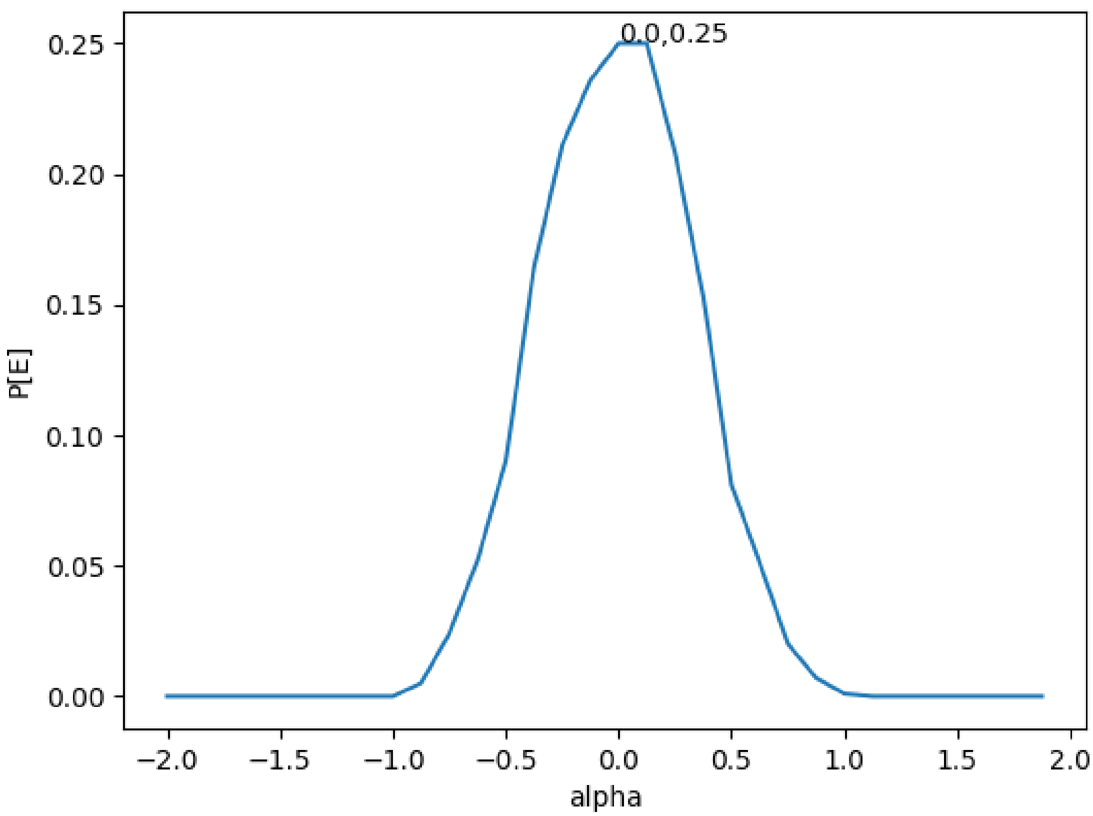


Figure 18: P[E] vs. Alpha (Unequal Priors)

C. SUMMARY

When we compare the P[E] as a result of the priors we can see that when scaled, their maximum P[E] correlates to the lowest prior.

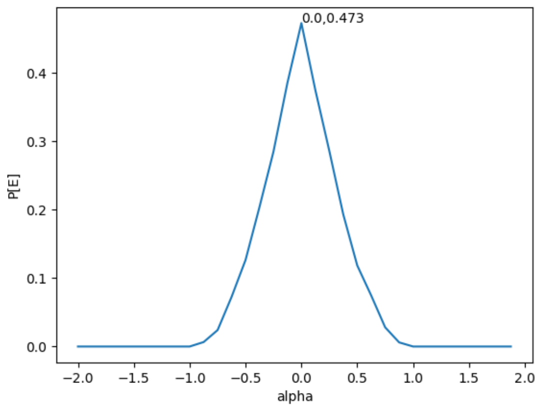


Figure 19: Equal Priors

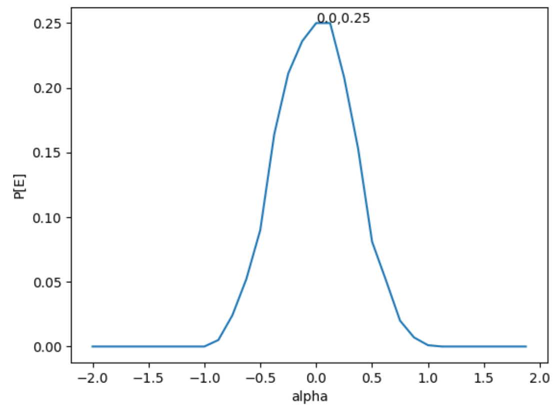
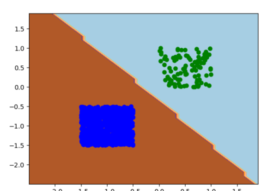


Figure 20: Unequal Priors

D. FOR THE DILIGENT STUDENT



;

Figure 21: P[w1] = .1, P[w2] = .9

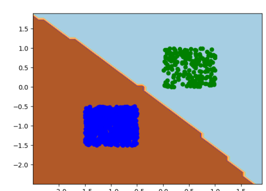


Figure 22: P[w1] = .0.21, P[w2] = .79

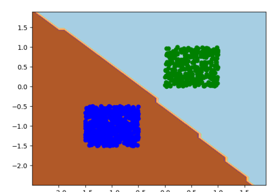


Figure 23: P[w1] = .35, P[w2] = .65

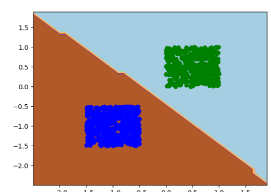
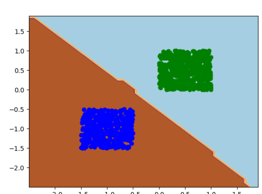


Figure 24: P[w1] = .47, P[w2] = .53



;

Figure 25: P[w1] = .58, P[w2] = .42

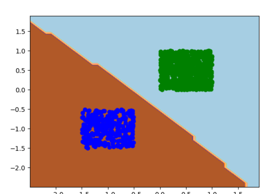


Figure 26: P[w1] = .69, P[w2] = .31

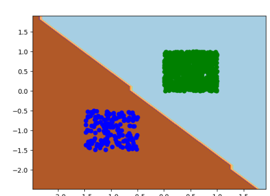


Figure 27: P[w1] = .83, P[w2] = .17

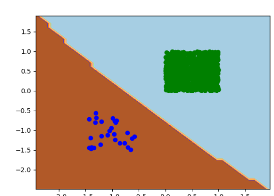


Figure 28: P[w1] = .97, P[w2] = .03

This is the plot of the error rates as a function of alpha.

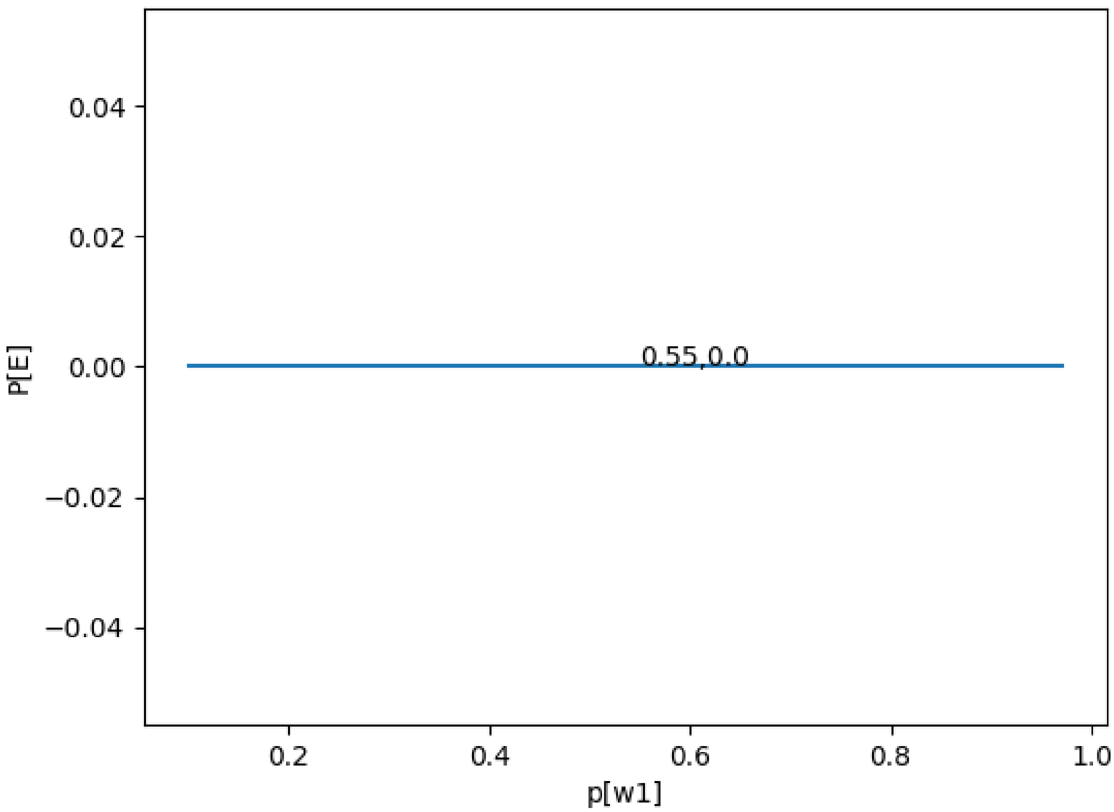


Figure 29: P[E] vs. Alpha (Unequal Priors)

E. APPENDIX

Code for Unequal priors:

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.decomposition import PCA
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.svm import SVC
from sklearn.inspection import DecisionBoundaryDisplay
import os

# generate data points
def generate_data(x1=0,x2=1,y1=0,y2=1):

  # create lists to store values
  xpoints1 = []
  ypoints1 = []
  xpoints2 = []
  ypoints2 = []
  for i in range(1000):
    xpoints1.append(random.uniform(x1,x2))
    ypoints1.append(random.uniform(x1,x2))
    xpoints2.append(random.uniform(y1,y2))
    ypoints2.append(random.uniform(y1,y2))

  # return lists of lists
  return list(map(list, zip(xpoints1, ypoints1))),list(map(list, zip(xpoints2, ypoints2)))

# generate weighted data for second part
def generate_weighted_data(x1=0,x2=1,y1=0,y2=1):

  # create lists to store values
  xpoints1 = []
  ypoints1 = []
  xpoints2 = []
  ypoints2 = []

  # w1
  for i in range(750):
    xpoints1.append(random.uniform(x1,x2))
    ypoints1.append(random.uniform(x1,x2))

  #w2
  for i in range(250):
    xpoints2.append(random.uniform(y1,y2))
    ypoints2.append(random.uniform(y1,y2))

  # return lists of lists
  return list(map(list, zip(xpoints1, ypoints1))),list(map(list, zip(xpoints2, ypoints2)))

# plot the decision surfaces
def plot_qda_decision_surfaces(data1,qda):

  # calculate the bounds of the data set
  min1, max1 = data1[:, 0].min()-1, data1[:,  0].max()+1
  min2, max2 = data1[:, 1].min()-1, data1[:, 1].max()+1

  # arrange the data sets so they are evenly spaced
  x1grid = np.arange(min1, max1, 0.1)
  x2grid = np.arange(min2, max2, 0.1)
```

```
  # create a meshgrid
  xx, yy = np.meshgrid(x1grid, x2grid)

  # flatten the grid
  r1, r2 = xx.flatten(), yy.flatten()

  # reshape them into vectors of the right size
  r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))

  # concatenate the vectors
  grid = np.hstack((r1,r2))

  # create a prediction for all the values
  yhat = qda.predict(grid)

  # reshape the x value to be an axis
  zz = yhat.reshape(xx.shape)

  # contour the plots with paried colors
  plt.contourf(xx, yy, zz, cmap='Paired')

# plot lda decision surface
def plot_lda_decision_surfaces(data1,lda):

  # calculate intercept and coefficients
  b, w1, w2 = lda.intercept_[0], lda.coef_[0][0], lda.coef_[0][1]

  # calculate the line
  x1 = np.array([np.min(data1[:,0], axis=0), np.max(data1[:,0], axis=0)])
  y1 = -(b+x1*w1)/w2

  # plot the line
  plt.plot(x1,y1,c="r")


def main():

    # lists for probability of error
    probability_error = []

    # list for xaxis
    xaxis = []

    # resolution
    iterations = 40

    # remove frames from previous
    os.system("rm this*.png")

    # iterate through the resolutions
    for i in range(iterations):

      # generate the data
      data1,data2 = generate_weighted_data(0,1,-2+(i/(iterations/4)),-1+(i/(iterations/4)))

      # generate the labels
      labels = [0]*len(data1) + [1]*len(data2)

      # copy the lists for later testing
      test1 = data1.copy()
      test2 = data2.copy()

      # concatenate data
```

```
    data1.extend(data2)
    data1 = np.array(data1)

    # create a QDA model
    model = QDA()

    # fit model to the data
    model.fit(data1,labels)

    # set the priors
    model.priors=[.75,.25]

    # calculate the probability of error
    probability_error.append(1-model.score(data1,labels))

    # append to the axis
    xaxis.append(-2 + i/(iterations/4))

    # create lists for each x and y for each data set
    # for plotting the scatter
    xes1 = []
    yes1 = []
    xes2 = []
    yes2 = []
    for x in range(len(test1)):
      xes1.append(test1[x][0])
      yes1.append(test1[x][1])
    for x in range(len(test2)):
      xes2.append(test2[x][0])
      yes2.append(test2[x][1])

    # plot the lda decision surfaces
    # plot_lda_decision_surfaces(data1,model)

    #plot the qda decision surfaces
    plot_qda_decision_surfaces(data1,model)

    # plot the two datasets
    plt.scatter(xes1,yes1,color = "green")
    plt.scatter(xes2,yes2,color = "blue")

    # save the plots as images
    plt.savefig("this"+str(i)+".png")

    # clear plots
    plt.cla()

  # plot the probability of errors
  plt.plot(xaxis,probability_error)

  # label axis
  plt.xlabel("alpha")
  plt.ylabel("P[E]")

  # label the middle point
  plt.text(xaxis[iterations//2],probability_error[iterations//2],str(xaxis[iterations//2])
                                                  + "," + str(probability_error[iterations//2])
                                                 )

  # save this image
  plt.savefig("results.png")


main()
```

Code for the equal priors

```
import numpy as np
import matplotlib.pyplot as plt
import random
from sklearn.decomposition import PCA
from sklearn.multiclass import OneVsRestClassifier
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis as QDA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.svm import SVC
from sklearn.inspection import DecisionBoundaryDisplay
import os

# generate data points
def generate_data(x1=0,x2=1,y1=0,y2=1):

  # create lists to store values
  xpoints1 = []
  ypoints1 = []
  xpoints2 = []
  ypoints2 = []
  for i in range(1000):
    xpoints1.append(random.uniform(x1,x2))
    ypoints1.append(random.uniform(x1,x2))
    xpoints2.append(random.uniform(y1,y2))
    ypoints2.append(random.uniform(y1,y2))

  # return lists of lists
  return list(map(list, zip(xpoints1, ypoints1))),list(map(list, zip(xpoints2, ypoints2)))

# generate weighted data for second part
def generate_weighted_data(x1=0,x2=1,y1=0,y2=1):

  # create lists to store values
  xpoints1 = []
  ypoints1 = []
  xpoints2 = []
  ypoints2 = []

  # w1
  for i in range(750):
    xpoints1.append(random.uniform(x1,x2))
    ypoints1.append(random.uniform(x1,x2))

  #w2
  for i in range(250):
    xpoints2.append(random.uniform(y1,y2))
    ypoints2.append(random.uniform(y1,y2))

  # return lists of lists
  return list(map(list, zip(xpoints1, ypoints1))),list(map(list, zip(xpoints2, ypoints2)))

# plot the decision surfaces
def plot_qda_decision_surfaces(data1,qda):

  # calculate the bounds of the data set
  min1, max1 = data1[:, 0].min()-1, data1[:,  0].max()+1
  min2, max2 = data1[:, 1].min()-1, data1[:, 1].max()+1

  # arrange the data sets so they are evenly spaced
```

```
  x1grid = np.arange(min1, max1, 0.1)
  x2grid = np.arange(min2, max2, 0.1)

  # create a meshgrid
  xx, yy = np.meshgrid(x1grid, x2grid)

  # flatten the grid
  r1, r2 = xx.flatten(), yy.flatten()

  # reshape them into vectors of the right size
  r1, r2 = r1.reshape((len(r1), 1)), r2.reshape((len(r2), 1))

  # concatenate the vectors
  grid = np.hstack((r1,r2))

  # create a prediction for all the values
  yhat = qda.predict(grid)

  # reshape the x value to be an axis
  zz = yhat.reshape(xx.shape)

  # contour the plots with paried colors
  plt.contourf(xx, yy, zz, cmap='Paired')

# plot lda decision surface
def plot_lda_decision_surfaces(data1,lda):

  # calculate intercept and coefficients
  b, w1, w2 = lda.intercept_[0], lda.coef_[0][0], lda.coef_[0][1]

  # calculate the line
  x1 = np.array([np.min(data1[:,0], axis=0), np.max(data1[:,0], axis=0)])
  y1 = -(b+x1*w1)/w2

  # plot the line
  plt.plot(x1,y1,c="r")


def main():

  # lists for probability of error
  probability_error = []

  # list for xaxis
  xaxis = []

  # resolution
  iterations = 32

  # remove frames from previous
  os.system("rm this*.png")

  # iterate through the resolutions
  for i in range(iterations):

    # generate the data
    data1,data2 = generate_data(0,1,-2+(i/(iterations/4)),-1+(i/(iterations/4)))

    # generate the labels
    labels = [0]*len(data1) + [1]*len(data2)

    # copy the lists for later testing
    test1 = data1.copy()
    test2 = data2.copy()
```

```
    # concatenate data
    data1.extend(data2)
    data1 = np.array(data1)

    # create a QDA model
    model = QDA()

    # fit model to the data
    model.fit(data1,labels)

    # set the priors
    model.priors=[.5,.5]

    # calculate the probability of error
    probability_error.append(1-model.score(data1,labels))

    # append to the axis
    xaxis.append(-2 + i/(iterations/4))

    # create lists for each x and y for each data set
    # for plotting the scatter
    xes1 = []
    yes1 = []
    xes2 = []
    yes2 = []
    for x in range(len(test1)):
      xes1.append(test1[x][0])
      yes1.append(test1[x][1])
    for x in range(len(test2)):
      xes2.append(test2[x][0])
      yes2.append(test2[x][1])

    # plot the lda decision surfaces
    # plot_lda_decision_surfaces(data1,model)

    #plot the qda decision surfaces
    plot_qda_decision_surfaces(data1,model)

    # plot the two datasets
    plt.scatter(xes1,yes1,color = "green")
    plt.scatter(xes2,yes2,color = "blue")

    # save the plots as images
    plt.savefig("this"+str(i)+".png")

    # clear plots
    plt.cla()

  # plot the probability of errors
  plt.plot(xaxis,probability_error)

  # label axis
  plt.xlabel("alpha")
  plt.ylabel("P[E]")

  # label the middle point
  plt.text(xaxis[iterations//2],probability_error[iterations//2],str(xaxis[iterations//2]) +
                                              "," + str(probability_error[iterations//2]))

  # save this image
  plt.savefig("results.png")
```

```
main()
```