quiz3_st121411_Rom

October 19, 2020

1 QUIZ 3 | st121411 - Rom Parnichkun

The quiz is designed to be completed in 120 minutes and they are all about Classification. The quiz will start at 1 PM 19th October 2020 till 3 PM 19th October 2020.

- You may use puffer/jupyter.
- You may use online resources.
- You will not be graded if submit late.
- You are not allowed to have a conversation with a human being. (A conversation with plants and pets is fine.)
- *** Submit in PDF and jupyter notebook format ***
- *** Write your student ID and student Name on the head of this file and save the file in this naming format. quiz3_stid_name***

Given 2 datasets, perform a classification on both datasets and answer the followings question.

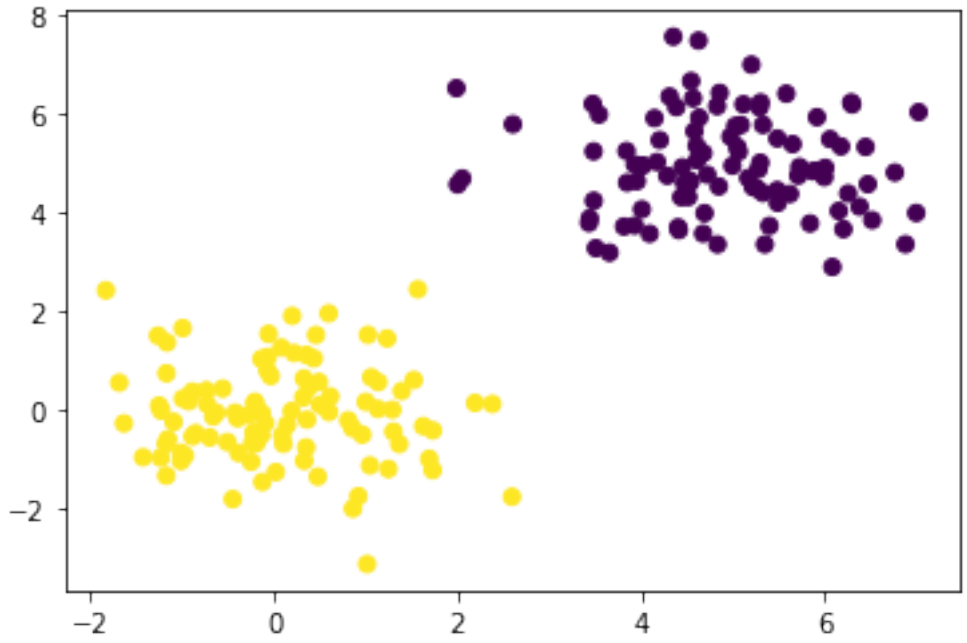
1. plot the data using a scatter plot. (2 pt)
2. Base on the data you saw, which method will you choose wo perform and why? (2 pts)
3. Perform the classification from scratch. (4 pts)
4. Is the choosen method yeild a good result? If not, why? (2 pts)

2 Dataset 1 (10 pts)

[1]:

[2]:
```
# 1. plot the data using a scatter plot. (2 pt)
import matplotlib.pyplot as plt
import numpy as np


dataset1_np = np.array(dataset1)

plt.scatter(dataset1_np[:,0],dataset1_np[:,1],c=dataset1_np[:,2])
```

[2]: <matplotlib.collections.PathCollection at 0x7faa788c3940>



[3]:
```
# 2. Base on the data you saw, which method will you choose wo perform and why? (2 pts)
print("The two blobs seem to be normally distributed therefore I will use the gaussian discrimant analysis to model this data")
```

The two blobs seem to be normally distributed therefore I will use the gaussian discrimant analysis to model this data

[4]:
```
# 3. Perform the classification from scratch. (4 pts)
class GDAModel:

    def parameters(self,X,y,naive = False,same_sigma = True):
        unique_y = np.unique(y)
        prior = np.zeros(len(unique_y))
        mean = np.zeros((len(unique_y),X.shape[1]))
        sigma = np.zeros((len(unique_y),X.shape[1],X.shape[1]))
        for j, yi in enumerate(unique_y):
            prior[j] = np.sum(y==yi)/y.size
            mean[j] = np.mean(X[y==yi],axis=0)
            sigma[j] = (1/np.sum(y==yi))*(X[y==yi]-mean[j]).T@(X[y==yi]-mean[j])

        # sets 0 to covariance terms if using the naive assumption
        if(naive == True):
            sigma[:,~np.eye(X.shape[1],dtype=bool)] = 0
```

```
        # if we're using the same sigma for all classes, we take the average of␣
↪them
        if(same_sigma == True):
            sigma[:] = np.mean(sigma,axis=0)

        return prior,mean,sigma,unique_y

    def joint_probability(self,x,prior,mean,sigma):
        joint_prob = np.zeros(prior.size)
        for j,mu in enumerate(mean):
            if(sigma.ndim == 2):
                sig = sigma
            else:
                sig = sigma[j]
            gaussian = (1/(((2*np.pi)**(x.size/2))*np.sqrt(np.linalg.
↪det(sig))))*np.exp(-1/2*(x-mu)@np.linalg.inv(sig)@((x-mu)))
            joint_prob[j] = prior[j]*gaussian
        return joint_prob

    def predict(self,X,prior,mean,sigma,unique_y):
        if(X.ndim == 2):
            y_pred = []
            for i in range(X.shape[0]):
                joint_prob = self.joint_probability(X[i],prior,mean,sigma)
                y_pred.append(unique_y[np.argmax(joint_prob)])
            y_pred = np.array(y_pred)
        else:
            joint_prob = self.joint_probability(X,prior,mean,sigma)
            y_pred = unique_y[np.argmax(joint_prob)]
        return y_pred

#6. score/error calculation
    def accuracy(self,y,y_pred):
        acc = np.sum(y == y_pred)/y.size
        return acc
```

[5]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(dataset1_np[:,[0,1]],␣
↪dataset1_np[:,2], test_size=0.20, random_state=42)
```

[6]:
```
X_test.shape
```

[6]: (40, 2)

[7]:
```
GDA = GDAModel()

prior,mean,sigma,unique_y = GDA.
 ↪parameters(X_train,y_train,naive=False,same_sigma=True)
```

[8]:
```
y_pred = GDA.predict(X_test,prior,mean,sigma,unique_y)
```

[9]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        16
         1.0       1.00      1.00      1.00        24

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

[10]:
```
# 4. Is the choosen method yeild a good result? If not, why? (2 pts)
print('''The chosen method was able to yield a good result and was able to␣
 ↪train very quickly because the assumption that both the blobs were normal is␣
 ↪sound.''')
```

```
The chosen method was able to yield a good result and was able to train very
quickly because the assumption that both the blobs were normal is sound.
```

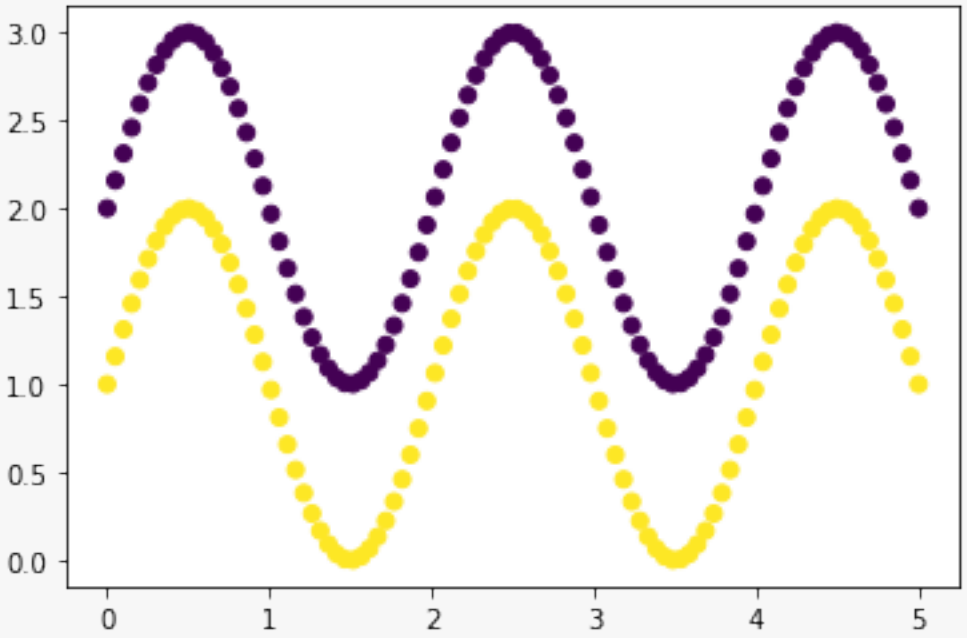
3 Dataset 2 (10 pts)

[11]:
```

```

[12]:
```
# 1. plot the data using a scatter plot. (2 pt)
dataset2_np = np.array(dataset2)

print(dataset2_np.shape)
plt.scatter(dataset2_np[:,0],dataset2_np[:,1],c=dataset2_np[:,2])
```

```
(200, 3)
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7faa578f7588>
```



[13]:
```
# 2. Base on the data you saw, which method will you choose wo perform and why? (2 pts)
print('from the data shown here, we can use SVM with a gaussian kernel because it will be able to accuracy classify the data. An alternative could be to use decision trees with boosting, but since decision trees ends up dividing the nodes in only 1 axis at a time, it wouldnt be as smooth as svm')
```

```
from the data shown here, we can use SVM with a gaussian kernel because it will
be able to accuracy classify the data. An alternative could be to use decision
trees with boosting, but since decision trees ends up dividing the nodes in only
```

1 axis at a time, it wouldnt be as smooth as svm

[14]:
```
# 3. Perform the classification from scratch. (4 pts)
from numpy import linalg
import cvxopt
import cvxopt.solvers
import pylab as pl

class SVMModel:

    def linear_kernel(self,x1, x2):
        return np.dot(x1, x2)

    def polynomial_kernel(self,x, y, p=2):
        return (1 + np.dot(x, y)) ** p

    def gaussian_kernel(self,x, y, sigma=0.9999):
        return np.exp(-linalg.norm(x-y)**2 / (2 * (sigma ** 2)))

    def fit(self, X, y, kernel, C):
        n_samples, n_features = X.shape

        # Gram matrix
        # initialize kernel matrix
        K = np.zeros((n_samples, n_samples))
        # Kernel matrix

        for i in range(n_samples):
            for j in range(n_samples):
                if kernel == 'linear_kernel':
                    K[i,j] = self.linear_kernel(X[i], X[j])
                elif kernel == 'polynomial_kernel':
                    K[i,j] = self.polynomial_kernel(X[i], X[j])
                else:
                    K[i,j] = self.gaussian_kernel(X[i], X[j])


        P = cvxopt.matrix(np.outer(y,y) * K)
        q = cvxopt.matrix(np.ones(n_samples) * -1)
        A = cvxopt.matrix(y, (1,n_samples))
        b = cvxopt.matrix(0.0)

        if C is None:
            G = cvxopt.matrix(np.diag(np.ones(n_samples) * -1))
            h = cvxopt.matrix(np.zeros(n_samples))
        else:
            tmp1 = np.diag(np.ones(n_samples) * -1)
```

```
            tmp2 = np.identity(n_samples)
            G = cvxopt.matrix(np.vstack((tmp1, tmp2)))
            tmp1 = np.zeros(n_samples)
            tmp2 = np.ones(n_samples) * C
            h = cvxopt.matrix(np.hstack((tmp1, tmp2)))

        # solve QP problem
        solution = cvxopt.solvers.qp(P, q, G, h, A, b)

        # Lagrange multipliers
        a = np.ravel(solution['x'])

        # Support vectors have non zero lagrange multipliers
        sv_idx = a > 1e-5
        ind = np.arange(len(a))[sv_idx]
        a = a[sv_idx]
        sv = X[sv_idx]
        sv_y = y[sv_idx]
        print("%d support vectors out of %d points" % (len(a), n_samples))

        # Intercept
        b = 0
        for n in range(len(a)):
            b += sv_y[n]
            b -= np.sum(a * sv_y * K[ind[n],sv_idx])
        b /= len(a)

        # Weight vector
        if kernel == 'linear_kernel':
            w = np.zeros(n_features)
            for n in range(len(a)):
                w += a[n] * sv_y[n] * sv[n]
        else:
            w = None
        return sv, sv_y, a, w, b

    def project(self,X, kernel,sv, sv_y, a, w, b):
        if w is not None:
            return np.dot(X, w) + b
        else:
            y_predict = np.zeros(len(X))
            for i in range(len(X)):
                s = 0
                for a_val, sv_y_val, sv_val in zip(a, sv_y, sv):
                    if kernel == 'polynomial_kernel':
                        s += a_val * sv_y_val * self.polynomial_kernel(X[i],␣
↪sv_val)
```

```
                    else:
                        s += a_val * sv_y_val * self.gaussian_kernel(X[i],␣
↪sv_val)
                y_predict[i] = s
            return y_predict + b

    def predict(self,X, kernel, sv, sv_y, a, w, b):
        return np.sign(self.project(X, kernel,sv, sv_y, a, w, b))

    def plot_contour(self,X1_train, X2_train, kernel, sv, sv_y, a, w, b):
        pl.plot(X1_train[:,0], X1_train[:,1], "ro")
        pl.plot(X2_train[:,0], X2_train[:,1], "bo")
        pl.scatter(sv[:,0], sv[:,1], s=100, c="g")
        # here we choose the range between -7 and 7 as we have choosen
        # the mean to be between -4 and 4 while generating data with the␣
↪variance of 0.8
        X1, X2 = np.meshgrid(np.linspace(-7,7,50), np.linspace(-7,7,50))
        X = np.array([[x1, x2] for x1, x2 in zip(np.ravel(X1), np.ravel(X2))])
        Z = self.project(X, kernel,sv, sv_y, a, w, b).reshape(X1.shape)
        pl.contour(X1, X2, Z, [0.0], colors='k', linewidths=1, origin='lower')
        pl.contour(X1, X2, Z + 1, [0.0], colors='grey', linewidths=1,␣
↪origin='lower')
        pl.contour(X1, X2, Z - 1, [0.0], colors='grey', linewidths=1,␣
↪origin='lower')

        pl.axis("tight")
        pl.show()
```

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(dataset2_np[:,[0,1]],␣
↪dataset2_np[:,2], test_size=0.20, random_state=42)

y_train[y_train==0] = -1
y_test[y_test==0] = -1
```

[16]:
```
SVM = SVMModel()

kernel = 'gaussian_kernel'

sv, sv_y, a, w, b = SVM.fit(X_train,y_train,kernel=kernel,C=None)
```

```
    pcost       dcost       gap    pres   dres
 0: -6.9496e+01 -1.9281e+02  5e+02  1e+01  2e+00
 1: -1.7109e+02 -2.9223e+02  2e+02  6e+00  9e-01
 2: -2.7725e+02 -3.8134e+02  1e+02  2e+00  4e-01
 3: -3.1719e+02 -3.9262e+02  8e+01  5e-01  9e-02
 4: -3.3379e+02 -3.5527e+02  2e+01  1e-01  2e-02
 5: -3.4160e+02 -3.5062e+02  9e+00  4e-03  6e-04
```

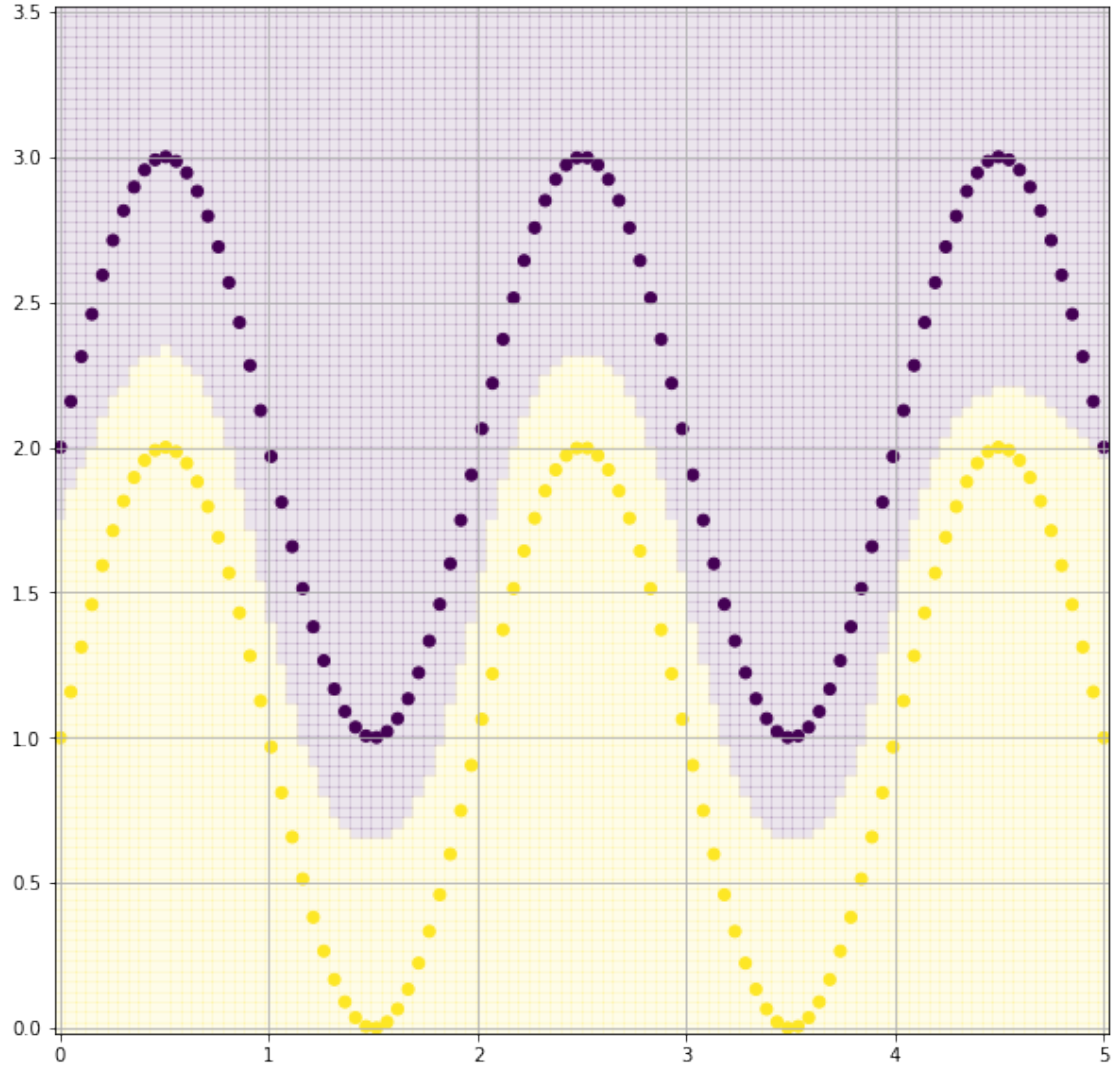
```
 6: -3.4564e+02 -3.4767e+02  2e+00  7e-04  1e-04
 7: -3.4659e+02 -3.4706e+02  5e-01  2e-05  3e-06
 8: -3.4689e+02 -3.4699e+02  1e-01  4e-06  6e-07
 9: -3.4692e+02 -3.4698e+02  6e-02  2e-06  2e-07
10: -3.4697e+02 -3.4697e+02  2e-03  3e-08  5e-09
11: -3.4697e+02 -3.4697e+02  2e-05  3e-10  5e-11
Optimal solution found.
25 support vectors out of 160 points
```

[17]:
```
y_pred = SVM.predict(X_test,kernel,sv,sv_y,a,w,b).astype(int)

print("y_test:",y_test.astype(int))
print("y_pred:",y_pred)
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
y_test: [-1 -1 -1  1  1  1  1  1 -1  1  1  1  1 -1  1 -1  1  1  1  1 -1  1 -1  1
 -1  1  1  1 -1 -1 -1 -1 -1  1 -1 -1  1 -1  1  1]
y_pred: [-1 -1 -1  1  1  1  1  1 -1  1  1  1  1 -1  1 -1  1  1  1  1 -1  1 -1  1
 -1  1  1  1 -1 -1 -1 -1 -1  1 -1 -1  1 -1  1  1]
              precision    recall  f1-score   support

        -1.0       1.00      1.00      1.00        17
         1.0       1.00      1.00      1.00        23

    accuracy                           1.00        40
   macro avg       1.00      1.00      1.00        40
weighted avg       1.00      1.00      1.00        40
```

[18]:
```
resolution = 100
x_series = np.linspace(0,5,resolution)
y_series = np.linspace(0,3.5,resolution)

x_mesh,y_mesh = np.meshgrid(x_series,y_series)

x_mesh = x_mesh.reshape(-1,1)
y_mesh = y_mesh.reshape(-1,1)

mesh = np.append(x_mesh,y_mesh,axis=1)
y_pred = SVM.predict(mesh,kernel,sv,sv_y,a,w,b).astype(int)

x_mesh = x_mesh.reshape(resolution,resolution)
y_mesh = y_mesh.reshape(resolution,resolution)
y_pred = y_pred.reshape(resolution,resolution)
```

[19]:
```
plt.figure(figsize=(10,10))
plt.scatter(dataset2_np[:,0],dataset2_np[:,1],c=dataset2_np[:,2])
plt.pcolormesh(x_mesh,y_mesh,y_pred,cmap='viridis',shading='auto',alpha=0.1)
plt.grid(True)
```



[20]:
```
# 4. Is the choosen method yeild a good result? If not, why? (2 pts)
print('Yes the method I used yielded a good result because SVMs with a gaussian␣
 ↪kernel can classify any data as long as they arent overlapping, although␣
 ↪SVMs can have disadvantages such as slow training time, for this small␣
 ↪dataset, it is able to train very quickly')
```

```
Yes the method I used yielded a good result because SVMs with a gaussian kernel
can classify any data as long as they arent overlapping, although SVMs can have
disadvantages such as slow training time, for this small dataset, it is able to
```

```
train very quickly
```