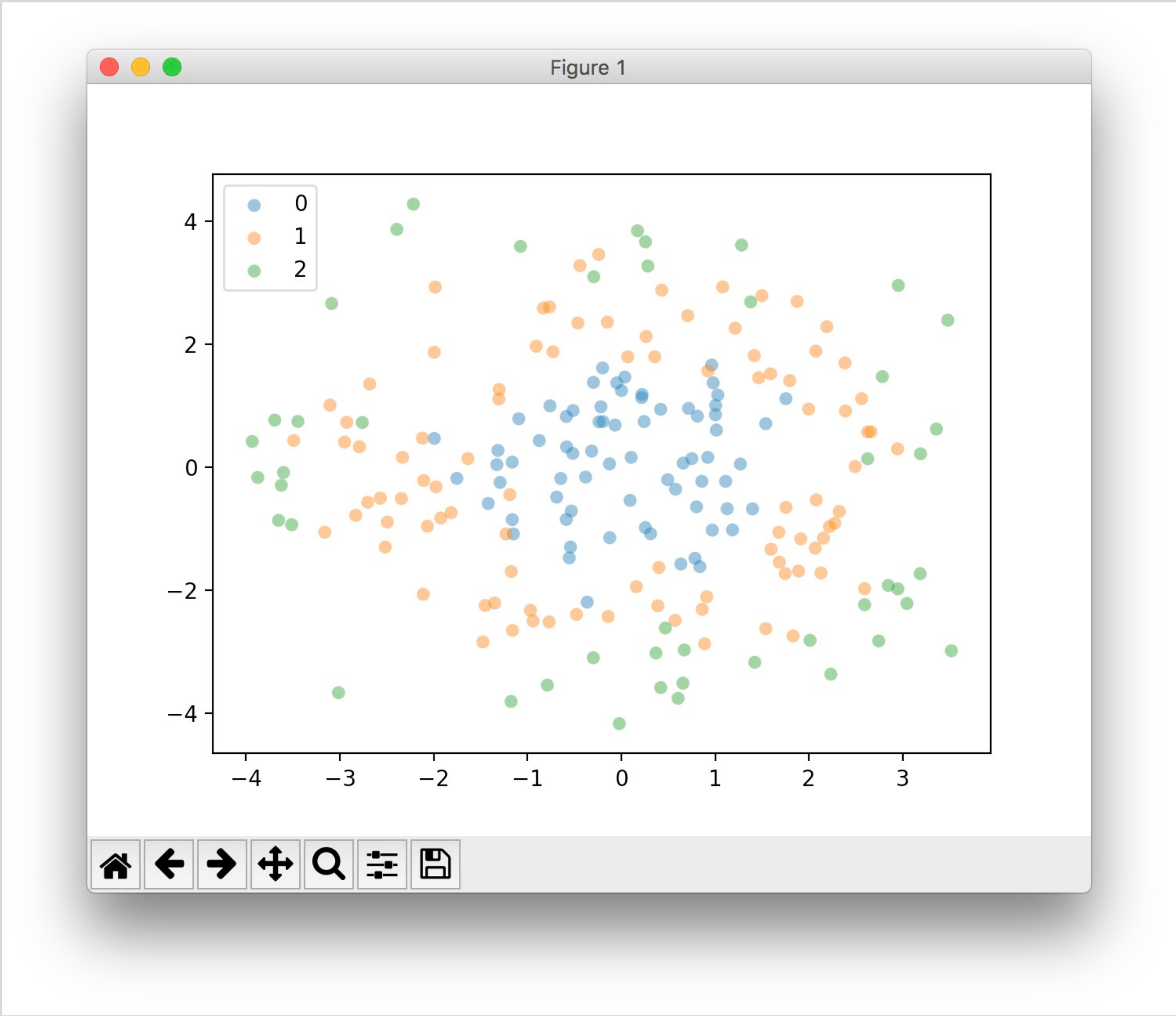
Machine Learning - Homework 5

Program Output

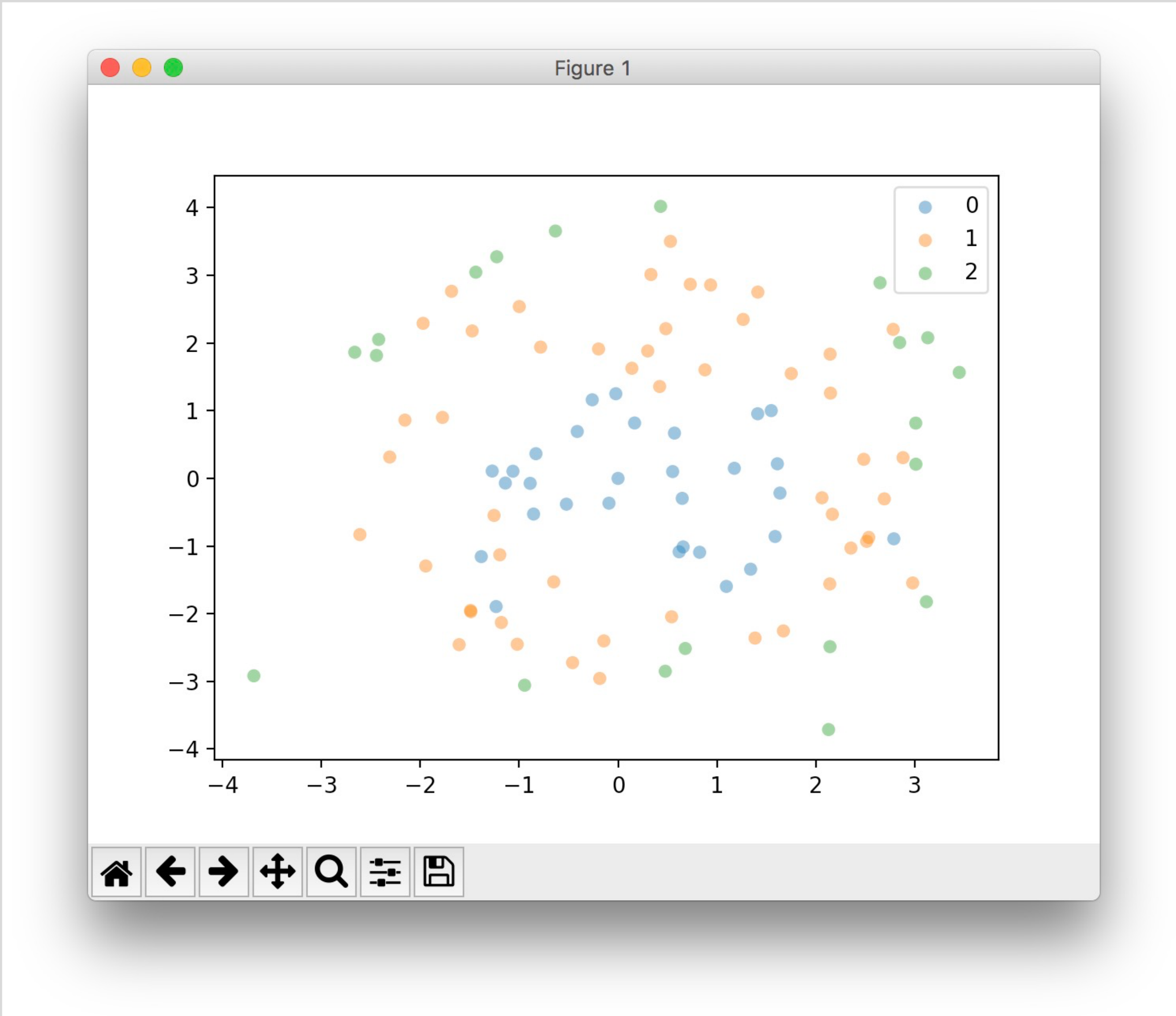
```
Part C Covariance Matrix
[[24  5  1]
 [ 8 33  9]
 [ 0  5 15]]
Error = 0.28

Part D Covariance Matrix
[[27  3  0]
 [ 6 36  8]
 [ 0  2 18]]
Error = 0.19
```

Graphs

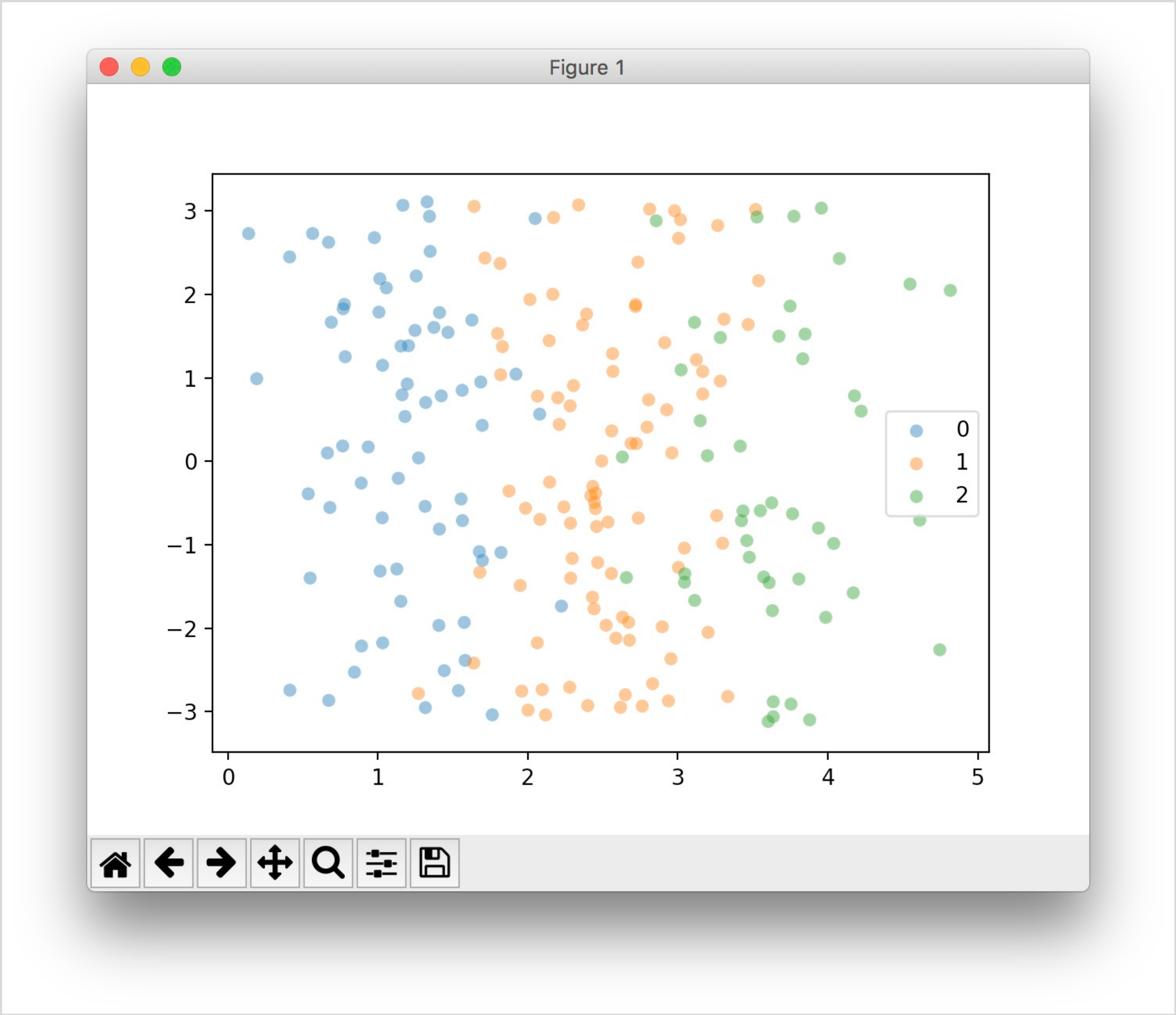
Untransformed Train Points



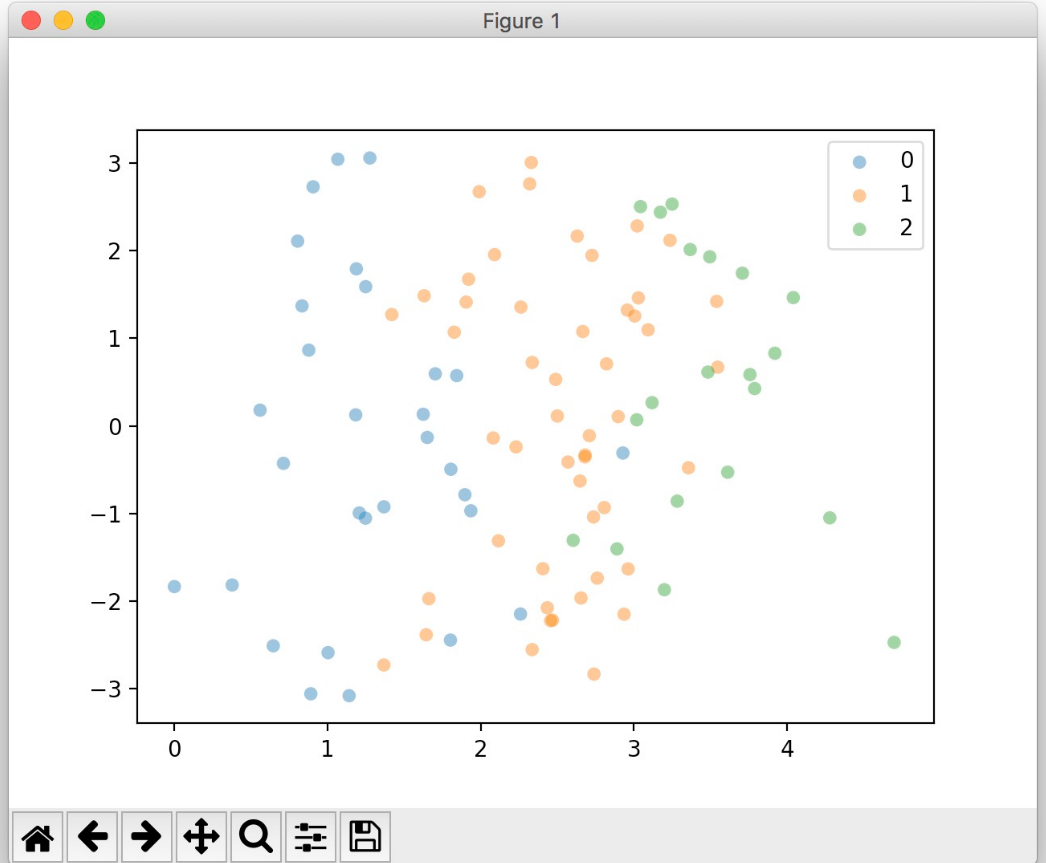
Untransformed Test Points



Polar Transformed Train Points



Polar Transformed Test Points



The polar transformation created a much more accurate classifier because, in polar coordinates, the data is closer to linearly separable. The untransformed data is concentric circles.

homework5.py

```
import numpy as np
from data import load_data, flatten_data
from mahalanobis import discriminant
from confusion_matrix import confusion_matrix
from coordinates import cart2pol, pol2cart
from plot import plot_data
from scipy.stats import norm
from polar_descriminant import mu_estimate

def main():
```

```
train, test = load_data()

prior = 1 / 3

# Num Classes
c = train.shape[0]
# Num Dimensions
d = train[0].shape[1]

means = np.empty((c, d))
covs = np.empty(((c, d, d)))

###### Part A #######

for i in range(c):
    means[i] = np.mean(train[i], axis=0)
    covs[i] = np.cov(train[i], rowvar=0)

predicted = np.array([], dtype=int)

flat_test, labels_test = flatten_data(test, c)

disc_values = np.zeros((100, 3))

####### Part B ######

for i, point in enumerate(flat_test):
    for j in range(c):
        m = discriminant(point, means[j], covs[j], d, prior)
        disc_values[i, j] = m

predicted = np.argmax(disc_values, axis=1)

####### Part C #########

cm, acc = confusion_matrix(labels_test, predicted, c)

print("Part C Covariance Matrix")
print(cm)
```

```python
    print(f"Error = {1 - acc}")

    ####### Part D ##########

    flat_train, labels_train = flatten_data(train, c)

    plot_data(flat_train.T[0], flat_train.T[1], labels_train, c)
    plot_data(flat_test.T[0], flat_test.T[1], labels_test, c)

    r_train, theta_train = cart2pol(flat_train)

    r_test, theta_test = cart2pol(flat_test)

    plot_data(r_train, theta_train, labels_train, c)
    plot_data(r_test, theta_test, labels_test, c)

    means = np.empty(c)
    covs = np.empty(c)
    posterior = np.zeros(c)

    disc_values = np.empty((r_test.shape[0], c))

    for i in range(c):
        means[i], covs[i] = mu_estimate(
            r_train[labels_train == i], 0, 100, .25)

    for i, pt in enumerate(r_test):
        for j in range(c):
            disc_values[i, j] = discriminant(
                pt, means[j], covs[j], d, prior)

    predicted = np.argmax(disc_values, axis=1)

    cm, acc = confusion_matrix(labels_test, predicted, c)

    print("Part D Covariance Matrix")
    print(cm)
    print(f"Error = {1 - acc}")
```

```python
if __name__ == "__main__":
    main()
```

data.py

```python
import numpy as np
from scipy.io.matlab import loadmat

def load_data():

    train, test = loadmat("./test_train_data_class3.mat")["Data"][0][0]

    train = np.array(train[0])
    test = np.array(test[0])

    for i in range(train.shape[0]):
        train[i] = np.transpose(train[i])

    for i in range(test.shape[0]):
        test[i] = np.transpose(test[i])
    return train, test



def flatten_data(data, c):
    # Because MATLAB...
    actual = np.array([], dtype=int)
    flat = np.zeros(2)

    # Flatten Test Array
    for i in range(c):
        for j in range(data[i].shape[0]):
            actual = np.append(actual, i)
            flat = np.vstack([flat, data[i][j]])

    flat = flat[1:len(flat) + 1]

    return flat, actual
```

mahalanobis.py

```
import numpy as np


def mah(x1, x2, cov):

    # Formula distance = √(x1 − x2)T ∑−1 (x1−x2)
    # numpy arrays have built in vector addition & subtraction!
    diff = x1 − x2

    # Vector−Matrix Multiplication
    # first pair (numpy only allows two at a time)

    if np.isscalar(cov):
        inv = 1 / cov
    else:
        inv = np.linalg.inv(cov)

    dist = np.dot(diff, inv)

    dist = np.dot(dist, diff)

    return dist


def discriminant(x, mean, covariance, dimension, prior):

    # g(x) = (−1/2) square(mahalanobis(x, mu)) − (d / 2)ln(2pi)
    #  − (1 / 2)ln(det(cov)) + ln(prior)

    a = (1 / 2) ∗ mah(x, mean, covariance)

    # np.log is natural log
    b = (dimension / 2) ∗ np.log(2 ∗ np.pi)
```

```python
    if np.isscalar(covariance):
        det = covariance
    else:
        det = np.linalg.det(covariance)
    c = (1 / 2) * np.log(det)

    d = np.log(prior)

    return -a - b - c + d
```

confusion_matrix.py

```python
import numpy as np


def confusion_matrix(actual, predicted, num_classes):
    cm = np.zeros((num_classes, num_classes), dtype=int)

    for a, p in zip(actual, predicted):
        cm[a, p] += 1

    acc = (actual == predicted).sum() / len(actual)

    return cm, acc
```

coordinates.py

```python
import numpy as np


def cart2pol(data):
    x = data.T[0]
    y = data.T[1]
    rho = np.sqrt(x**2 + y**2)
    phi = np.arctan2(y, x)
    return rho, phi
```

```python
def pol2cart(rho, phi):
    x = rho * np.cos(phi)
    y = rho * np.sin(phi)
    return(x, y)
```

plot.py

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


def plot_data(r, theta, labels, c):

    fig = plt.figure()
    ax = fig.add_subplot(111)

    for j in range(c):
        ax.scatter(r[labels == j], theta[labels == j],
                   label=f"Class {j}", alpha=0.5, edgecolors="none")

    ax.legend(range(c + 1))
    plt.show()
```

polar_descriminant.py

```python
import numpy as np


def mu_estimate(data, mu_0, sigma_0, variance):
    mu_hat = np.mean(data)
    n = data.shape[0]

    ns = n * sigma_0

    a = ns / (ns + variance)
```

```
    b = variance / (ns + variance)

    mu_n = (a * mu_hat) + (b * mu_0)

    num = sigma_0 * variance
    den = n * sigma_0 + variance

    sigma_n = num / den

    return mu_n, sigma_n
```