Drake Svoboda - Homework 2

Problem 1

In [1]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets
from sklearn.decomposition import PCA
import numpy as np
from numpy import linalg as LA

# import some data to play with
iris = datasets.load_iris()
```

In [2]:
```
print(iris.feature_names)
print(iris.data[0:10]) # Show 10 exampes
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)
']
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]]
```

In [3]:
```
def covariance(data):
    data -= np.mean(data, axis=0)
    n = data.shape[0]
    return (np.matmul(data.transpose(), data) / (n - 1))
```

In [4]:
```
S = covariance(iris.data)
S
```

Out[4]:
```
array([[ 0.68569351, -0.03926846,  1.27368233,  0.5169038 ],
       [-0.03926846,  0.18800403, -0.32171275, -0.11798121],
       [ 1.27368233, -0.32171275,  3.11317942,  1.29638747],
       [ 0.5169038 , -0.11798121,  1.29638747,  0.58241432]])
```

In [5]:
```
w,v = LA.eig(S)
w,v
```

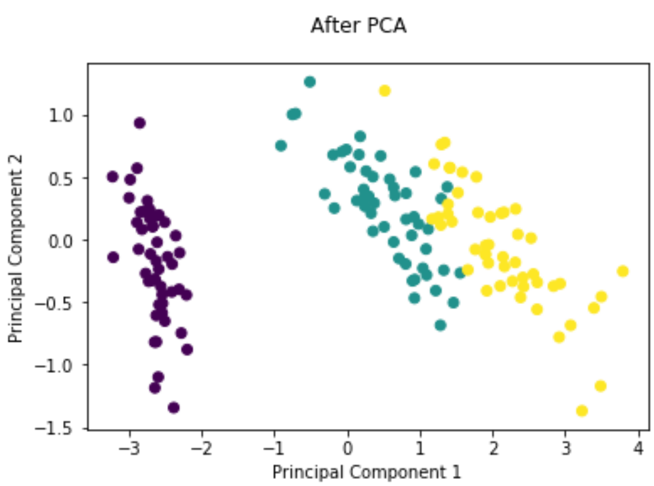
Out[5]:
```
(array([4.22484077, 0.24224357, 0.07852391, 0.02368303]),
 array([[ 0.36158968, -0.65653988, -0.58099728,  0.31725455],
        [-0.08226889, -0.72971237,  0.59641809, -0.32409435],
        [ 0.85657211,  0.1757674 ,  0.07252408, -0.47971899],
        [ 0.35884393,  0.07470647,  0.54906091,  0.75112056]]))
```

In [6]:
```
data_ = np.matmul(iris.data, v)
data_[0:10] # Show 10 examples
```

Out[6]:
```
array([[-2.68420713e+00, -3.26607315e-01, -2.15118370e-02,
         1.00615724e-03],
       [-2.71539062e+00,  1.69556848e-01, -2.03521425e-01,
         9.96024240e-02],
       [-2.88981954e+00,  1.37345610e-01,  2.47092410e-02,
         1.93045428e-02],
       [-2.74643720e+00,  3.11124316e-01,  3.76719753e-02,
        -7.59552741e-02],
       [-2.72859298e+00, -3.33924564e-01,  9.62296998e-02,
        -6.31287327e-02],
       [-2.27989736e+00, -7.47782713e-01,  1.74325619e-01,
        -2.71468037e-02],
       [-2.82089068e+00,  8.21045110e-02,  2.64251085e-01,
        -5.00996251e-02],
       [-2.62648199e+00, -1.70405349e-01, -1.58015103e-02,
        -4.62817610e-02],
       [-2.88795857e+00,  5.70798026e-01,  2.73354061e-02,
        -2.66154143e-02],
       [-2.67384469e+00,  1.06691704e-01, -1.91533300e-01,
        -5.58909660e-02]])
```

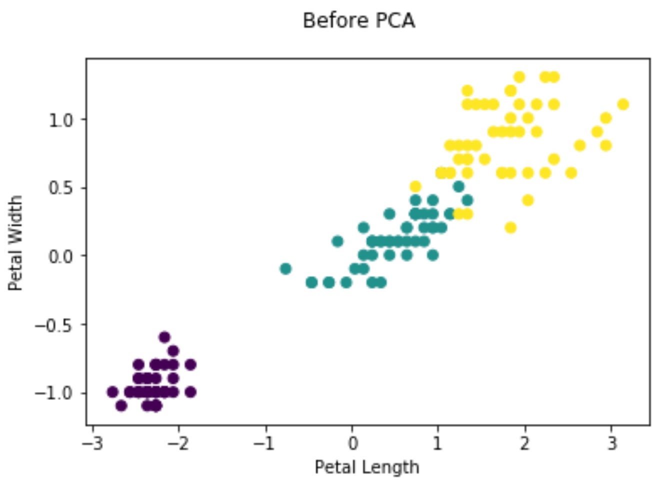
In [7]:
```
fig, ax = plt.subplots()
fig.suptitle('After PCA')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
ax.scatter(data_[:,0], data_[:,1], c=iris.target)
```

Out[7]: <matplotlib.collections.PathCollection at 0x1a13552390>



In [8]:
```
fig, ax = plt.subplots()
fig.suptitle('Before PCA')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
ax.scatter(iris.data[:,2], iris.data[:,3], c=iris.target)
```

Out[8]: <matplotlib.collections.PathCollection at 0x1a13662780>



Before PCA separates the data better for vertical decision boundaries. Before PCA might also have the edge for more complicated linear boundaries. I would say that the data is better separeted before PCA

Problem 2

a)

Larger values of r equate to smaller distances. The orientation of the vector connecting the two points will affect the difference between two minkowski distances. If the vector is perpendicular to an axis, then the values for all minkowski distances will be equal

In [9]:
```
def minkowski(x1, y1, x2, y2, r):
    return (abs(x2 - x1)**r + abs(y2 - y1)**r)**(1/float(r))
```

In [10]:
```
thetas = np.arange(0, np.pi / 2, np.pi / 64)
```

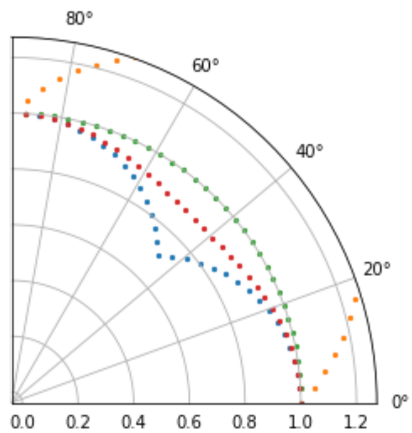
In [11]:
```
coords = [(np.cos(theta), np.sin(theta)) for theta in thetas]
```

In [12]:
```
L_max = [max(x, y) for x, y in coords]
```

In [13]:
```
fig = plt.figure()
ax = fig.add_subplot(111, polar=True)
ax.set_thetamin(0)
ax.set_thetamax(90)

ax.scatter(thetas, L_max, s=5)
for r in range(1, 4):
    ax.scatter(thetas, [minkowski(0, 0, x, y, r) for x, y in coords], s=5)

# Plot the distance for various values of r and theta between the
# center of a unit circle and a point on the unit circle.
# Euclidian distance is in green and represents the unit circle.
# Orange is manhattan and blue is L_max
```



b)

The values aproach 12 from above

In [14]:
```
[minkowski(0, 0, 5, 12, r) for r in range(1, 16)]
```

Out[14]:
```
[17.0,
 13.0,
 12.28264235951734,
 12.089418031483874,
 12.02999053739867,
 12.010442816140744,
 12.003734212737811,
 12.001362162049496,
 12.000504620057733,
 12.00018925092342,
 12.00007168890213,
 12.000027381652155,
 12.00001053148139,
 12.000004074692761,
 12.000001584604773]
```

c)

$$Cos(x, y) = \frac{x \cdot y}{||x||\ ||y||} = \frac{\Sigma x_k y_k}{\sqrt{\Sigma x_k^2}\sqrt{\Sigma y_k^2}}$$
$$Euclidean(x, y) = \sqrt{\Sigma(x_k - y_k)^2}$$
$$= \sqrt{\Sigma(x_k^2 - 2x_k y_k + y_k^2)}$$
$$When\ ||x|| = ||y|| = 1$$
$$Euclidean(x, y) = \sqrt{2 - 2\Sigma(x_k y_k)}$$
$$= \sqrt{2 - 2Cos(x, y)}$$

Problem 3

a)

Plot	x-axis	y-axis
Histogram	attribute values	count
Box Plot	Attributes	attribute values
Percentile Plot	percentile	attribute values
Scatter Plot	attribute values	attribute values
Data Matrix Plot	Attributes	Attributes
Correlation Matrix Plot	Attributes	Attributes
Parallel Coordinates Plot	Attributes	attribute values

b)

*	Chicago	Detroit	Total
Apples	100	200	300
Bananas	300	400	700
Total	400	600	1000

c)

*	Location 1	Location 2	Location 3	Total
Product 1	10	0	6	16
Product 2	5	22	0	27
Total	15	22	6	43

In [15]:
```
def gini(c0, c1):
    return 1 - c0**2 - c1**2

def weighted_gini(c0, c1, weight):
    return weight * gini(c0, c1)
```

In [16]:
```
print("a) Overall ", gini(10/20, 10/20))

b = sum([weighted_gini(1/1, 0/1, 1/20) for i in range(10)]) + sum(weighted_gini(0/
1, 1/1, 1/20) for i in range(10))
print("b) ID      ", b)

print("c) Gender  ",
      weighted_gini(4/10, 6/10, 10/10)    # F
      + weighted_gini(6/10, 4/10, 10/10)) # M

print("d) Car     ",
      weighted_gini(1/4, 3/4, 4/20)     # Family
      + weighted_gini(8/8, 0/8, 8/20)   # Sports
      + weighted_gini(1/8, 7/8, 8/20)) # Luxury

print("e) Shirt   ",
      weighted_gini(3/5, 2/5, 5/20)     # Small
      + weighted_gini(3/7, 4/7, 7/20)   # Medium
      + weighted_gini(2/4, 2/4, 4/20)   # Large
      + weighted_gini(2/4, 2/4, 4/20)) # Extra Large
```

```
a) Overall   0.5
b) ID        0.0
c) Gender    0.96
d) Car       0.16250000000000003
e) Shirt     0.49142857142857144
```

f) Car type is the best as it produces the most favorable Gini index and will generalize better than ID

Problem 5

a)

In [17]:
```
def entropy(c0, c1):
    part_c0 = 0 if c0 == 0 else - c0 * np.log2(c0)
    part_c1 = 0 if c1 == 0 else - c1 * np.log2(c1)

    return part_c0 + part_c1

def weighted_entropy(c0, c1, weight):
    return weight * entropy(c0, c1)
```

In [18]:
```
root = entropy(4/9, 5/9)
root
```

Out[18]:
```
0.9910760598382222
```

b)

In [19]:
```
a1 = weighted_entropy(3/4, 1/4, 4/9) + weighted_entropy(1/5, 4/5, 5/9)
a1_gain = root - a1
a1_gain
```

Out[19]:
```
0.22943684069673975
```

In [20]:
```
a2 = weighted_entropy(2/5, 3/5, 5/9) + weighted_entropy(2/4, 2/4, 4/9)
a2_gain = root - a2
a2_gain
```

Out[20]:
```
0.007214618474517431
```

c)

In [21]:
```
data = [[1, 1, 1, 1],
        [1, 1, 6, 1],
        [1, 0, 5, 0],
        [0, 0, 4, 1],
        [0, 1, 7, 0],
        [0, 1, 3, 0],
        [0, 0, 8, 0],
        [1, 0, 7, 1],
        [0, 1, 5, 0]]
```

In [22]:
```
splits = [1, 2, 3, 4, 5, 6, 7]
entropies = {}

for split in splits:
    less_c0 = [x for x in data if (x[2] <= split and x[3] == 1)]
    less_c1 = [x for x in data if (x[2] <= split and x[3] == 0)]
    n_less = len(less_c0) + len(less_c1)
    e_less = weighted_entropy(len(less_c0)/n_less, len(less_c1)/n_less, n_less/9)

    greater_c0 = [x for x in data if (x[2] > split and x[3] == 1)]
    greater_c1 = [x for x in data if (x[2] > split) and x[3] == 0]
    n_greater = len(greater_c0) + len(greater_c1)
    e_greater = weighted_entropy(len(greater_c0)/n_greater, len(greater_c1)/n_grea
ter, n_greater/9)


    entropies[split] = (e_less + e_greater)

entropies
```

Out[22]:
```
{1: 0.8483857803777467,
 2: 0.8483857803777467,
 3: 0.9885107724710845,
 4: 0.9182958340544896,
 5: 0.9838614413637048,
 6: 0.9727652780181631,
 7: 0.8888888888888888}
```

In [23]:
```
gains =  root - list(entropies.values())
gains
```

Out[23]:
```
array([0.14269028, 0.14269028, 0.00256529, 0.07278023, 0.00721462,
       0.01831078, 0.10218717])
```

d)

It is best to spit on a2

Problem 6

a)

$$Classifactionerror(root) = 1 - \frac{50}{100} = .5$$

For attribute A

*	T	F	Total
+	25	25	50
-	0	50	50
Total	25	75	100

$$ClassificationError(A) = (1 - \frac{25}{25}) * \frac{25}{100} + (1 - \frac{50}{75}) * \frac{75}{100} = .25$$
$$Gain(A) = .5 - .25 = .25$$

For attribute B

*	T	F	Total
+	30	20	50
-	20	30	50
Total	50	50	100

$$ClassificationError(B) = 1 - \frac{60}{100} = .4$$
$$Gain(A) = .5 - .4 = .1$$

For attribute C

*	T	F	Total
+	25	25	50
-	25	25	50
Total	50	50	100

$$ClassificationError(B) = 1 - \frac{50}{100} = .5$$
$$Gain(C) = .5 - .5 = 0$$

Splitting on A produces the highest gain

b)

After the split

In [24]:
```
#      A
#     / \
#    T   F
#   /     \
# +        -
```

The true leaf node is pure and does not split

For the F tree node:

For attribute B

B	T	F	Total
+	25	0	25
-	20	30	50
Total	45	30	75

$$ClassifactionError(B) = 1 - \frac{55}{75} = .2666$$
$$Gain(B) = (1 - 50/75) - .2666 = .0666$$

For attribute C

C	T	F	Total
+	0	25	25
-	25	25	50
Total	25	50	75

$$ClassifactionError(C) = 1 - \frac{50}{75} = .3333$$
$$Gain(C) = (1 - 50/75) - .25 = 0$$

c)

20 instances are misclassified

In [ ]:

In [ ]: