Aman Chauhan - achauha3

Khantil Choksi - khchoksi

Q1 - PCA

Import the libraries

```
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
import numpy as np
import os
%matplotlib inline
```

(A) Load the Data

```
def data_and_headers(filename):
    data = None
    with open(filename) as fp:
        data = [x.strip().split(',') for x in fp.readlines()]
    headers = data[0]
    headers = np.asarray(headers)
    class_field = len(headers) - 1
    data_x = [[float(x[i]) for i in range(class_field)] for x in data[1:]]
    data_x = np.asarray(data_x)
    data_y = [[str(x[i]) for i in range(class_field, class_field + 1)] for x in data[1:]]
    data_y = np.asarray(data_y)
    return headers, data_x, data_y
```

```
headers, train_x, train_y = data_and_headers('Data' + os.sep + 'hw2q1_train.csv')
headers, test_x, test_y = data_and_headers('Data' + os.sep + 'hw2q1_test.csv')
```

```
print('Training Data')
print('Number of features - ' + str(train_x.shape[1]))
print('Number of target features - ' + str(train_y.shape[1]))
print('Number of observations - ' + str(train_x.shape[0]))
print('Number of observations in category R - ' +
str(train_y[train_y=='R'].shape[0]))
print('Number of observations in category M - ' +
str(train_y[train_y=='M'].shape[0]))
print()
print('Testing Data')
print('Number of features - ' + str(test_x.shape[1]))
print('Number of target features - ' + str(test_y.shape[1]))
print('Number of observations - ' + str(test_x.shape[0]))
print('Number of observations in category R - ' +
str(test_y[test_y=='R'].shape[0]))
print('Number of observations in category M - ' +
str(test_y[test_y=='M'].shape[0]))
```

```
Training Data
Number of features - 60
Number of target features - 1
Number of observations - 156
Number of observations in category R - 73
Number of observations in category M - 83

Testing Data
Number of features - 60
Number of target features - 1
Number of observations - 52
Number of observations in category R - 24
Number of observations in category M - 28
```

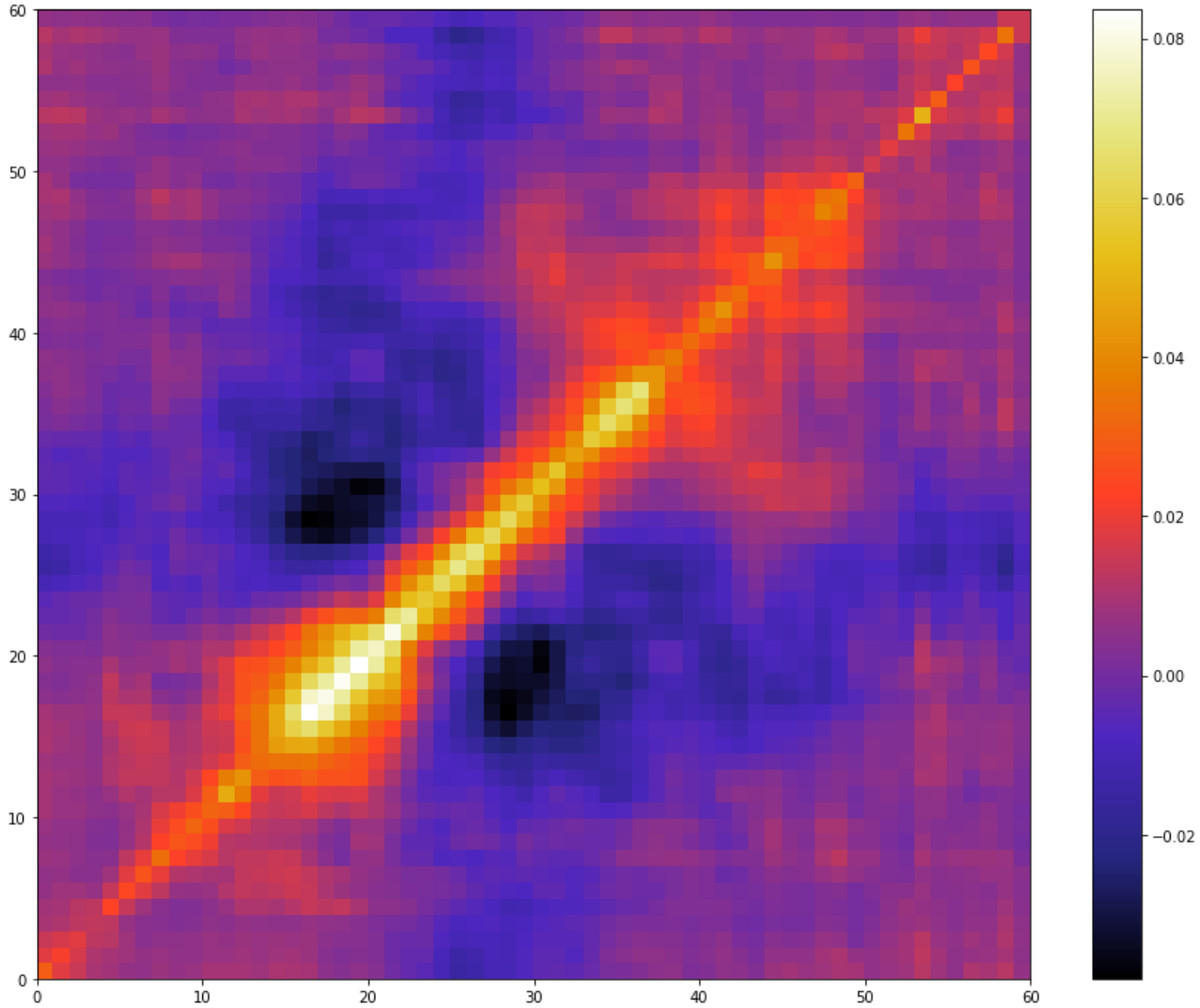
(B) Normalization and PCA

```
def normalize(data, minima, maxima):
    normal = np.copy(data)
    normal = (normal - minima) / (maxima - minima)
    return normal
```

```
normal_train = normalize(train_x, np.amin(train_x, axis=0), np.amax(train_x,
axis=0))
normal_test = normalize(test_x, np.amin(train_x, axis=0), np.amax(train_x,
axis=0))
```

(i) Covariance of Training Dataset

```
covariance = np.cov(normal_train, rowvar=False)
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(12)
fig.set_figwidth(15)
im = axes.pcolor(covariance, cmap='CMRmap')
fig.colorbar(im, ax=axes)
plt.show()
```



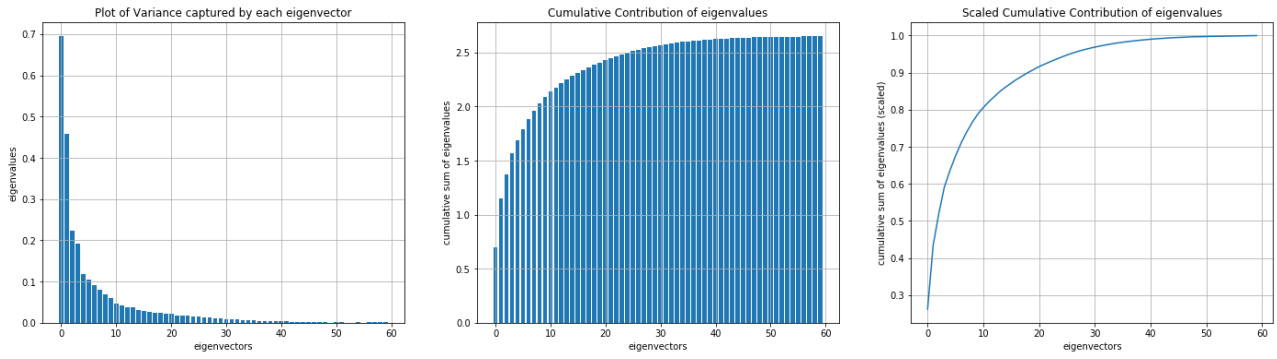
(ii) Eigenvalue and Eigenvectors

```
print('Size of covariance matrix - ' + str(covariance.shape))
w,v = np.linalg.eig(covariance)
print('Top 5 Eigenvalues - ' + ', '.join(['{:.3f}'.format(x) for x in
w[:5]]))
```

```
Size of covariance matrix - (60, 60)
Top 5 Eigenvalues - 0.695, 0.457, 0.223, 0.191, 0.118
```

(iii) Plot of Eigenvalues

```
fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_figheight(6)
fig.set_figwidth(24)
axes[0].bar(np.arange(60), w)
axes[0].xaxis.grid()
axes[0].yaxis.grid()
axes[0].set_xlabel('eigenvectors')
axes[0].set_ylabel('eigenvalues')
axes[0].set_title('Plot of Variance captured by each eigenvector')
axes[1].bar(np.arange(60), np.cumsum(w))
axes[1].xaxis.grid()
axes[1].yaxis.grid()
axes[1].set_xlabel('eigenvectors')
axes[1].set_ylabel('cumulative sum of eigenvalues')
axes[1].set_title('Cumulative Contribution of eigenvalues')
axes[2].plot(np.arange(60), np.cumsum(w)/np.sum(w))
axes[2].xaxis.grid()
axes[2].yaxis.grid()
axes[2].set_xlabel('eigenvectors')
axes[2].set_ylabel('cumulative sum of eigenvalues (scaled)')
axes[2].set_title('Scaled Cumulative Contribution of eigenvalues')
plt.show()
```



Number of Eigenvectors - 10

This is because if we look at the scaled cumulative contribution of the eigen values, we can see that the first 10 eigenvalues cover almost 80% of all variance in the data set. So 10 seems to be a good choice for selecting the number of eigenvectors.

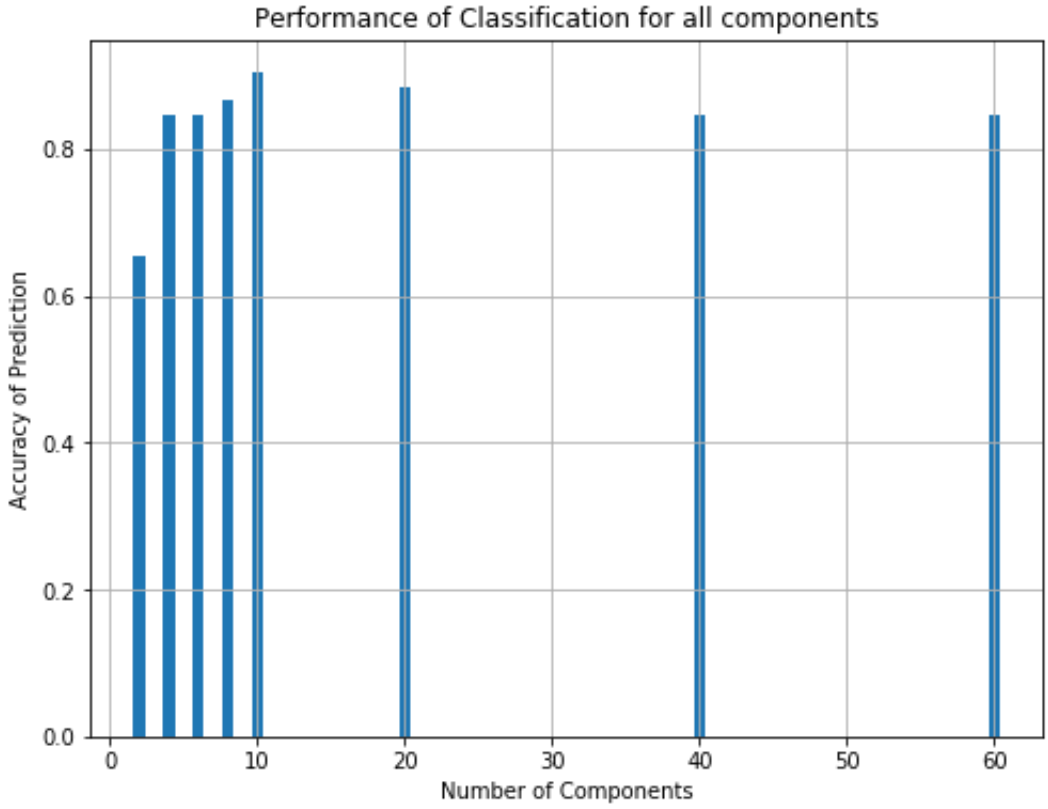
(iv) PCA with KNN

```
ncomps = [2,4,6,8,10,20,40,60]
cls = [KNeighborsClassifier(n_neighbors=3,
metric='euclidean').fit(np.matmul(normal_train, v[:,:ncomps[i]]),
np.ravel(train_y)) for i in range(len(ncomps))]
```

```
pred4 = cls[4].predict(np.matmul(normal_test, v[:,:ncomps[4]]))
true4 = np.ravel(test_y)
with open('q1iv.csv','w') as fp:
    fp.write(', '.join(['Component' + str(i) for i in
range(1,ncomps[4]+1)])+', Class, Class\n')
    transformed = np.matmul(normal_test, v[:,:ncomps[4]])
    for i in range(len(transformed)):
        fp.write(', '.join([str(x) for x in transformed[i]]))
        fp.write(', ' + str(true4[i]))
        fp.write(', ' + str(pred4[i]) + '\n')
```

You can find the output of the above code in file q1iv.csv

```
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(6)
fig.set_figwidth(8)
accuracies = [cls[i].score(np.matmul(normal_test, v[:,:ncomps[i]]),
np.ravel(test_y)) for i in range(len(ncomps))]
axes.bar(ncomps, accuracies)
axes.set_xlabel('Number of Components')
axes.set_ylabel('Accuracy of Prediction')
axes.set_title('Performance of Classification for all components')
plt.grid()
plt.show()
print('Accuracy - ')
print('Components\tAccuracy')
for i in range(len(ncomps)):
    print('{}\t\t{:.4f}%'.format(ncomps[i],100*accuracies[i]))
```




```
Accuracy -
Components  Accuracy
2         65.3846%
4         84.6154%
6         84.6154%
8         86.5385%
10        90.3846%
20        88.4615%
40        84.6154%
60        84.6154%
```

Reasonable Number of PCA components - 10

The reasons are 2-fold - 1) We predicted it above when we plotted the eigenvalues and 2) It gives the best accuracy out of the given number of components.

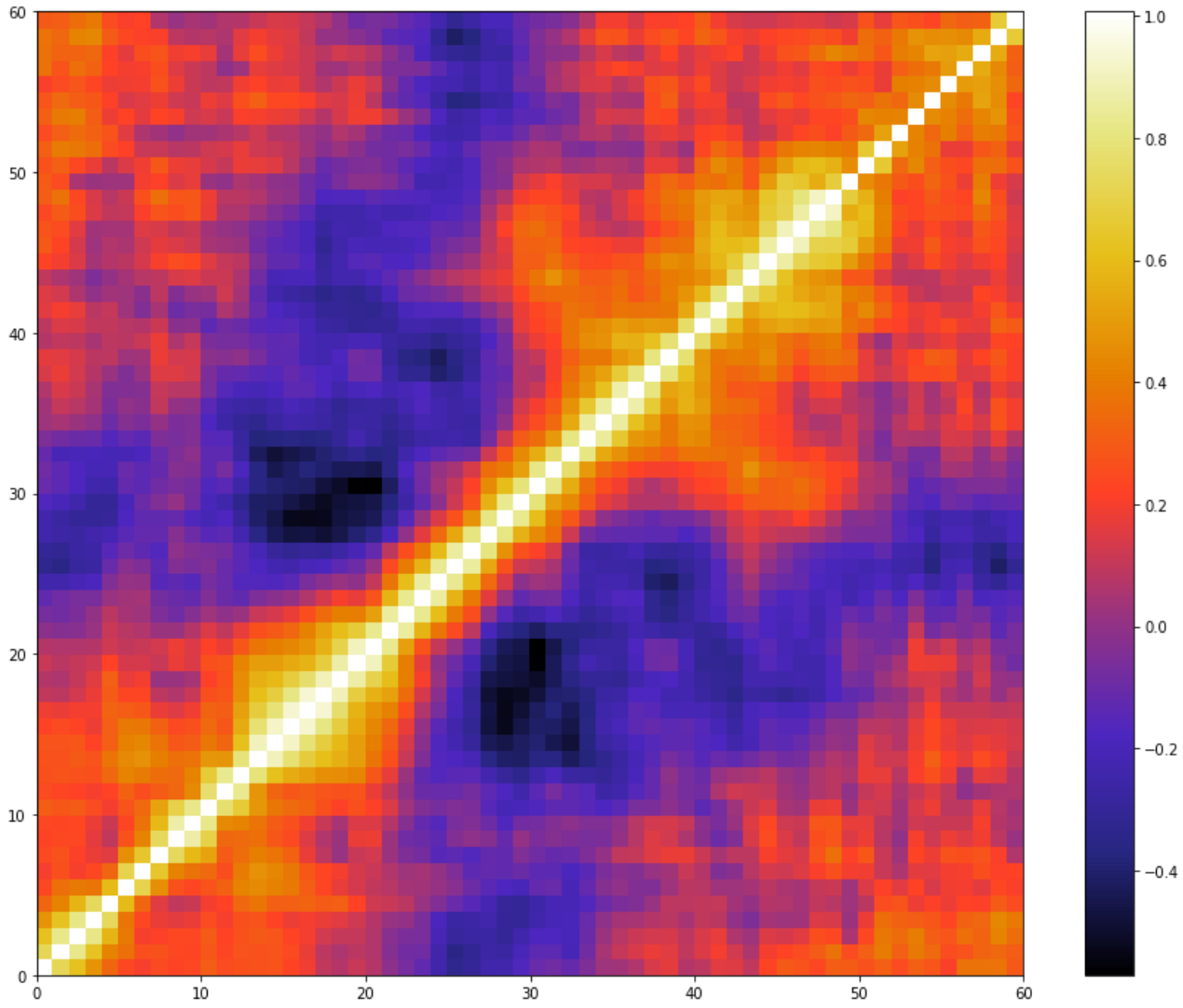
(C) Standardization and PCA

```
def standardize(data, mean, sd):
    standard = np.copy(data)
    standard = (standard - mean) / sd
    return standard
```

```
standard_train = standardize(train_x, np.mean(train_x, axis=0),
np.std(train_x, axis=0))
standard_test = standardize(test_x, np.mean(train_x, axis=0),
np.std(train_x, axis=0))
```

(i) Covariance of Training Dataset

```
covariance = np.cov(standard_train, rowvar=False)
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(12)
fig.set_figwidth(15)
im = axes.pcolor(covariance, cmap='CMRmap')
fig.colorbar(im, ax=axes)
plt.show()
```



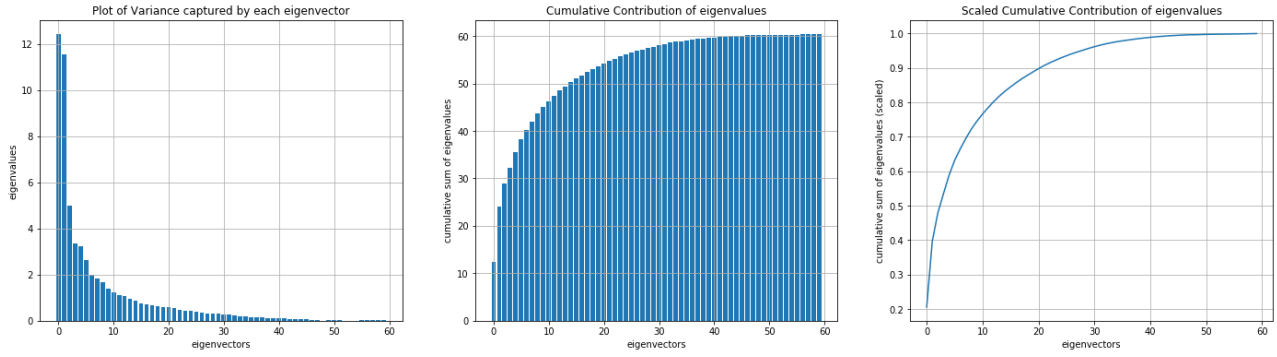
(ii) Eigenvalue and Eigenvectors

```
print('Size of covariance matrix - ' + str(covariance.shape))
w,v = np.linalg.eig(covariance)
print('Top 5 Eigenvalues - ' + ', '.join(['{:.3f}'.format(x) for x in w[:5]]))
```

```
Size of covariance matrix - (60, 60)
Top 5 Eigenvalues - 12.429, 11.558, 4.979, 3.345, 3.226
```

(iii) Plot of Eigenvalues

```
fig, axes = plt.subplots(nrows=1, ncols=3)
fig.set_figheight(6)
fig.set_figwidth(24)
axes[0].bar(np.arange(60), w)
axes[0].xaxis.grid()
axes[0].yaxis.grid()
axes[0].set_xlabel('eigenvectors')
axes[0].set_ylabel('eigenvalues')
axes[0].set_title('Plot of Variance captured by each eigenvector')
axes[1].bar(np.arange(60), np.cumsum(w))
axes[1].xaxis.grid()
axes[1].yaxis.grid()
axes[1].set_xlabel('eigenvectors')
axes[1].set_ylabel('cumulative sum of eigenvalues')
axes[1].set_title('Cumulative Contribution of eigenvalues')
axes[2].plot(np.arange(60), np.cumsum(w)/np.sum(w))
axes[2].xaxis.grid()
axes[2].yaxis.grid()
axes[2].set_xlabel('eigenvectors')
axes[2].set_ylabel('cumulative sum of eigenvalues (scaled)')
axes[2].set_title('Scaled Cumulative Contribution of eigenvalues')
plt.show()
```



Number of Eigenvectors - 20

This is because if we look at the scaled cumulative contribution of the eigen values, we can see that the first 20 eigenvalues cover almost 90% of all variance in the data set. Also, beyond that, all the eigenvectors combined can only contribute 10% more variance. So 20 seems to be a good choice for selecting the number of eigenvectors.

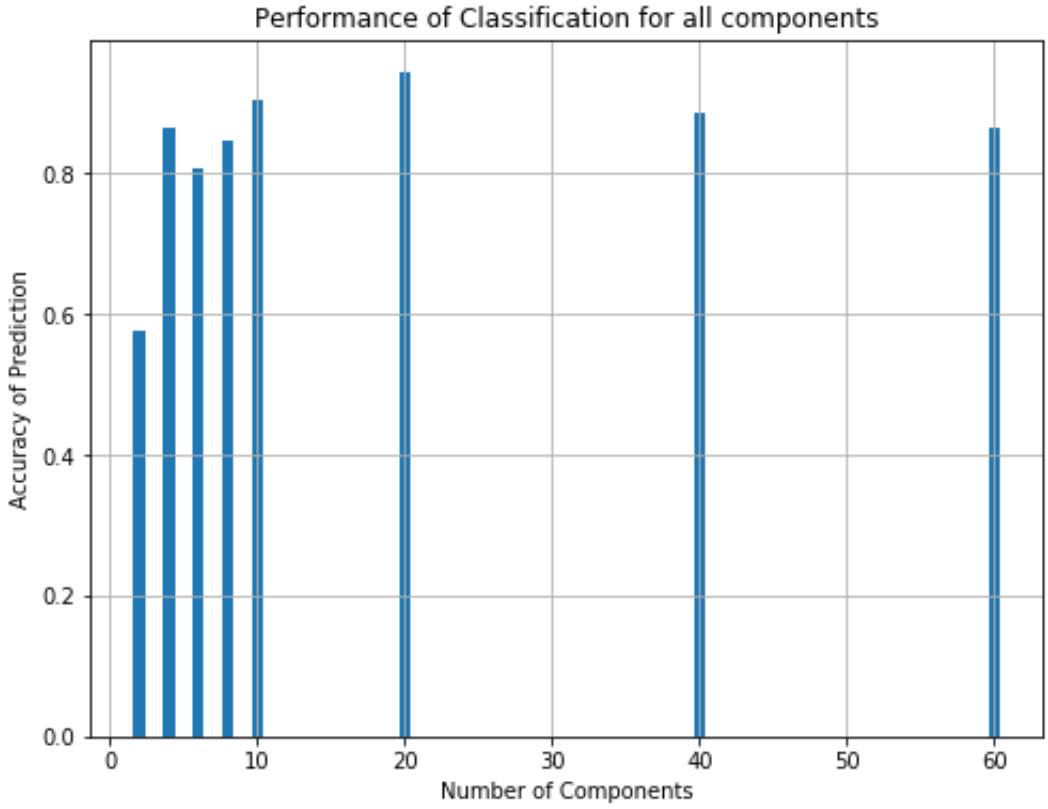
(iv) PCA with KNN

```
ncomps = [2,4,6,8,10,20,40,60]
cls = [KNeighborsClassifier(n_neighbors=3,
metric='euclidean').fit(np.matmul(standard_train, v[:,:ncomps[i]]),
np.ravel(train_y)) for i in range(len(ncomps))]
```

```
pred4 = cls[4].predict(np.matmul(standard_test, v[:,:ncomps[4]]))
true4 = np.ravel(test_y)
with open('q1v.csv','w') as fp:
    fp.write(', '.join(['Component' + str(i) for i in
range(1,ncomps[4]+1)])+', Class, Class\n')
    transformed = np.matmul(standard_test, v[:,:ncomps[4]])
    for i in range(len(transformed)):
        fp.write(', '.join([str(x) for x in transformed[i]]))
        fp.write(', ' + str(true4[i]))
        fp.write(', ' + str(pred4[i]) + '\n')
```

You can find the output of the above code in file q1v.csv

```
fig, axes = plt.subplots(nrows=1, ncols=1)
fig.set_figheight(6)
fig.set_figwidth(8)
accuracies = [cls[i].score(np.matmul(standard_test, v[:,:ncomps[i]]),
np.ravel(test_y)) for i in range(len(ncomps))]
axes.bar(ncomps, accuracies)
axes.set_xlabel('Number of Components')
axes.set_ylabel('Accuracy of Prediction')
axes.set_title('Performance of Classification for all components')
plt.grid()
plt.show()
print('Accuracy - ')
print('Components\tAccuracy')
for i in range(len(ncomps)):
    print('{}\t\t{:.4f}%'.format(ncomps[i],100*accuracies[i]))
```




```
Accuracy -
Components  Accuracy
2        57.6923%
4        86.5385%
6        80.7692%
8        84.6154%
10       90.3846%
20       94.2308%
40       88.4615%
60       86.5385%
```

Reasonable Number of PCA components - 20

The reason is simple - It gives the best accuracy out of the given number of components. If we look at the cumulative plot of eigen values above, we can see that 20 eigenvectors were able to cover almost 90% of the variance. As we keep increasing/decreasing the number of components from this point, the accuracy seems to fall gradually.

(D) Preference

Comparing both procedures, I believe that standardization is better than normalization for this dataset. Not only did PCA + KNN with standardized data give better accuracy, the eigenvalues were much smoother and more interpretable. We can clearly see how many attributes the top 10 eigen vectors were covering, giving a good intuition about the number of eigen vectors/number of PCA components to choose.

Q2 - Decision Trees

The data given is as follows -

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F
7	BLUE	LARGE	HIGH	COOL	F
16	BLUE	LARGE	LOW	COOL	F
17	BLUE	LARGE	HIGH	COOL	F
27	BLUE	LARGE	HIGH	HOT	T
29	BLUE	SMALL	HIGH	HOT	T
33	BLUE	SMALL	LOW	HOT	F
34	BLUE	LARGE	HIGH	HOT	T
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T
10	RED	SMALL	HIGH	COOL	T
11	RED	SMALL	LOW	COOL	F
25	RED	SMALL	HIGH	HOT	T
36	RED	LARGE	HIGH	HOT	T
45	RED	SMALL	LOW	HOT	F
50	RED	LARGE	LOW	HOT	F

(A) ID3

The entropy of Class label is (9F and 7T) -

$$H(Class) = -[\frac{9}{16}\log_2(\frac{9}{16}) + \frac{7}{16}\log_2(\frac{7}{16})] = 0.9887$$

Root Split

Initial Attribute Specific Entropies (6 as split for Attr1) -

$$H(Class|Attr1) = \frac{1}{8}[-(\frac{0}{2}log_2(\frac{0}{2}) + \frac{2}{2}log_2(\frac{2}{2}))] + \frac{7}{8}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] = 0.8750$$
$$IG(Class|Attr1) = H(Class) - H(Class|Attr1) = 0.9887 - 0.8750 = 0.1137$$

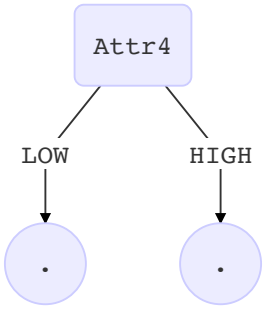
$$H(Class|Attr2) = \frac{1}{2}[-(\frac{3}{8}log_2(\frac{3}{8}) + \frac{5}{8}log_2(\frac{5}{8}))] + \frac{1}{2}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] = 0.9772$$
$$IG(Class|Attr2) = H(Class) - H(Class|Attr1) = 0.9887 - 0.9772 = 0.0115$$

$$H(Class|Attr3) = \frac{1}{2}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] + \frac{1}{2}[-(\frac{3}{8}log_2(\frac{3}{8}) + \frac{5}{8}log_2(\frac{5}{8}))] = 0.9772$$
$$IG(Class|Attr3) = H(Class) - H(Class|Attr1) = 0.9887 - 0.9772 = 0.0115$$

$$H(Class|Attr4) = \frac{1}{2}[-(\frac{7}{8}log_2(\frac{7}{8}) + \frac{1}{8}log_2(\frac{1}{8}))] + \frac{1}{2}[-(\frac{1}{4}log_2(\frac{1}{4}) + \frac{3}{4}log_2(\frac{3}{4}))] = 0.6774$$
$$IG(Class|Attr4) = H(Class) - H(Class|Attr1) = 0.9887 - 0.6774 = 0.3113$$

$$H(Class|Attr5) = \frac{1}{2}[-(\frac{5}{8}log_2(\frac{5}{8}) + \frac{3}{8}log_2(\frac{3}{8}))] + \frac{1}{2}[-(\frac{1}{4}log_2(\frac{1}{4}) + \frac{3}{4}log_2(\frac{3}{4}))] = 0.8828$$
$$IG(Class|Attr5) = H(Class) - H(Class|Attr1) = 0.9887 - 0.8828 = 0.1059$$

So we split at Attr4.



Level 1 Split

Attr4=high

Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F
27	BLUE	LARGE	HIGH	HOT	T
29	BLUE	SMALL	HIGH	HOT	T
34	BLUE	LARGE	HIGH	HOT	T
10	RED	SMALL	HIGH	COOL	T
25	RED	SMALL	HIGH	HOT	T
36	RED	LARGE	HIGH	HOT	T

$H(Class|Attr4 = high) = -[\frac{1}{4}\log_2(\frac{1}{4}) + \frac{3}{4}\log_2(\frac{3}{4})] = 0.8113$

Split at 25 for Attr1 -

$H(Class|Attr4 = high, Attr1) = \frac{3}{8}[-(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3}))] + \frac{5}{8}[-(\frac{0}{8}log_2(\frac{0}{8}) + \frac{8}{8}log_2(\frac{8}{8}))] = 0.3443$
$IG(Class|Attr4 = high, Attr1) = H(Class|Attr4 = high) - H(Class|Attr4 = high, Attr1) = 0.8113 - 0.3 = 0.4670$

$H(Class|Attr4 = high, Attr2) = \frac{3}{8}[-(\frac{3}{3}log_2(\frac{3}{3}) + \frac{0}{3}log_2(\frac{0}{3}))] + \frac{5}{8}[-(\frac{2}{5}log_2(\frac{2}{5}) + \frac{3}{5}log_2(\frac{3}{5}))] = 0.6068$
$IG(Class|Attr4 = high, Attr2) = H(Class|Attr4 = high) - H(Class|Attr4 = high, Attr2) = 0.8113 - 0.6068 = 0.2045$

$H(Class|Attr4 = high, Attr3) = \frac{3}{8}[-(\frac{0}{3}log_2(\frac{0}{3}) + \frac{3}{3}log_2(\frac{3}{3}))] + \frac{5}{8}[-(\frac{2}{5}log_2(\frac{2}{5}) + \frac{3}{5}log_2(\frac{3}{5}))] = 0.6068$
$IG(Class|Attr4 = high, Attr3) = H(Class|Attr4 = high) - H(Class|Attr4 = high, Attr3) = 0.8113 - 0.6068 = 0.2045$

$H(Class|Attr4 = high, Attr5) = \frac{3}{8}[-(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3}))] + \frac{5}{8}[-(\frac{0}{8}log_2(\frac{0}{8}) + \frac{8}{8}log_2(\frac{8}{8}))] = 0.3443$
$IG(Class|Attr4 = high, Attr5) = H(Class|Attr4 = high) - H(Class|Attr4 = high, Attr5) = 0.8113 - 0.3443 = 0.4670$

So we split on Attr1 at 25 for Attr4=high.

Attr4=low

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F
16	BLUE	LARGE	LOW	COOL	F
33	BLUE	SMALL	LOW	HOT	F
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T
11	RED	SMALL	LOW	COOL	F
45	RED	SMALL	LOW	HOT	F
50	RED	LARGE	LOW	HOT	F

$H(Class|Attr4 = low) = -[\frac{1}{8}\log_2(\frac{1}{8}) + \frac{7}{8}\log_2(\frac{7}{8})] = 0.5436$

Split at 11 for Attr1 -

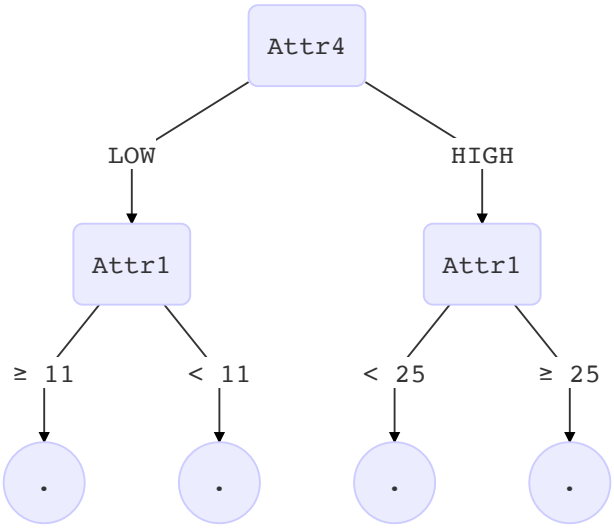
$H(Class|Attr4 = low, Attr1) = \frac{3}{8}[-(\frac{1}{3}log_2(\frac{1}{3}) + \frac{2}{3}log_2(\frac{2}{3}))] = 0.3443$
$IG(Class|Attr4 = low, Attr1) = H(Class|Attr4 = low) - H(Class|Attr4 = low, Attr1) = 0.5436 - 0.3443 = 0.1992$

$H(Class|Attr4 = low, Attr2) = \frac{3}{8}[-(\frac{3}{3}log_2(\frac{3}{3}) + \frac{0}{3}log_2(\frac{0}{3}))] + \frac{5}{8}[-(\frac{1}{5}log_2(\frac{1}{5}) + \frac{4}{5}log_2(\frac{4}{5}))] = 0.4512$
$IG(Class|Attr4 = low, Attr2) = H(Class|Attr4 = low) - H(Class|Attr4 = low, Attr2) = 0.5436 - 0.4512 = 0.0924$

$H(Class|Attr4 = low, Attr3) = \frac{3}{8}[-(\frac{0}{3}log_2(\frac{0}{3}) + \frac{3}{3}log_2(\frac{3}{3}))] + \frac{5}{8}[-(\frac{1}{5}log_2(\frac{1}{5}) + \frac{4}{5}log_2(\frac{4}{5}))] = 0.4512$
$IG(Class|Attr4 = low, Attr3) = H(Class|Attr4 = low) - H(Class|Attr4 = low, Attr3) = 0.5436 - 0.4512 = 0.0924$

$H(Class|Attr4 = low, Attr5) = \frac{3}{8}[-(\frac{0}{3}log_2(\frac{0}{3}) + \frac{3}{3}log_2(\frac{3}{3}))] + \frac{5}{8}[-(\frac{1}{5}log_2(\frac{1}{5}) + \frac{4}{5}log_2(\frac{4}{5}))] = 0.4512$
$IG(Class|Attr4 = low, Attr5) = H(Class|Attr4 = low) - H(Class|Attr4 = low, Attr5) = 0.5436 - 0.4512 = 0.0924$

So we split at Attr1at value 11



Level 2 Split

Attr4=high, Attr1≥25

Attr1	Attr2	Attr3	Attr4	Attr5	Class
27	BLUE	LARGE	HIGH	HOT	T
29	BLUE	SMALL	HIGH	HOT	T
34	BLUE	LARGE	HIGH	HOT	T
25	RED	SMALL	HIGH	HOT	T
36	RED	LARGE	HIGH	HOT	T

As all labels are same(T) this is a leaf node. So we don't proceed further on this branch.

Attr4=high, Attr1<25

Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F
10	RED	SMALL	HIGH	COOL	T

$H(Class|Attr4 = high, Attr1 < 25) = -[\frac{1}{3}\log_2(\frac{1}{3}) + \frac{2}{3}\log_2(\frac{2}{3})] = 0.9183$

$H(Class|Attr4 = high, Attr1 < 25, Attr2) = \frac{1}{3}[-\frac{1}{1}log_2(\frac{1}{1})] + \frac{2}{3}[-\frac{2}{2}log_2(\frac{2}{2})] = 0$
$IG(Class|Attr4 = high, Attr1 < 25, Attr2) = 0.9183$

$H(Class|Attr4 = high, Attr1 < 25, Attr3) = \frac{1}{3}[-\frac{1}{1}log_2(\frac{1}{1})] + \frac{2}{3}[-\frac{2}{2}log_2(\frac{2}{2})] = 0$
$IG(Class|Attr4 = high, Attr1 < 25, Attr3) = 0.9183$

$H(Class|Attr4 = high, Attr1 < 25, Attr5) = \frac{1}{1}[-(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3}))] = 0.9183$
$IG(Class|Attr4 = high, Attr1 < 25, Attr5) = 0$

We split on Attr2.

Attr4=low,Attr1≥11

Attr1	Attr2	Attr3	Attr4	Attr5	Class
16	BLUE	LARGE	LOW	COOL	F
33	BLUE	SMALL	LOW	HOT	F
11	RED	SMALL	LOW	COOL	F
45	RED	SMALL	LOW	HOT	F
50	RED	LARGE	LOW	HOT	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low,Attr1<11

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T

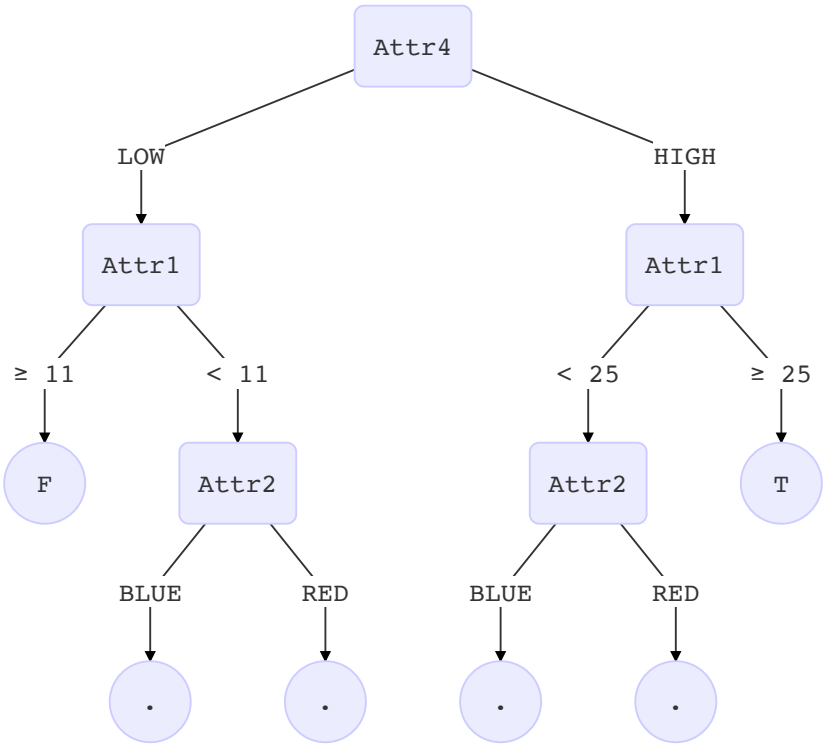
$H(Class|Attr4 = low, Attr1 < 11) = -[\frac{2}{3}\log_2(\frac{2}{3}) + \frac{1}{3}\log_2(\frac{1}{3})] = 0.9183$

$H(Class|Attr4 = high, Attr1 < 11, Attr2) = \frac{1}{3}[-\frac{1}{1}log_2(\frac{1}{1})] + \frac{2}{3}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] = 0.6667$
$IG(Class|Attr4 = high, Attr1 < 11, Attr2) = 0.2516$

$H(Class|Attr4 = high, Attr1 < 11, Attr3) = \frac{1}{3}[-\frac{1}{1}log_2(\frac{1}{1})] + \frac{2}{3}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] = 0.6667$
$IG(Class|Attr4 = high, Attr1 < 11, Attr3) = 0.2516$

$H(Class|Attr4 = high, Attr1 < 11, Attr5) = \frac{1}{1}[-(\frac{2}{3}log_2(\frac{2}{3}) + \frac{1}{3}log_2(\frac{1}{3}))] = 0.9183$
$IG(Class|Attr4 = high, Attr1 < 11, Attr5) = 0$

We split on Attr2.



Level 3 Split

Attr4=high, Attr1<25, Attr2=red

Attr1	Attr2	Attr3	Attr4	Attr5	Class
10	RED	SMALL	HIGH	COOL	T

As all labels are same(T), this is a leaf node. So we don't proceed further along this branch.

Attr4=high, Attr1<25, Attr2=blue

Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low, Attr1<11, Attr2=blue

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low, Attr1<11, Attr2=red

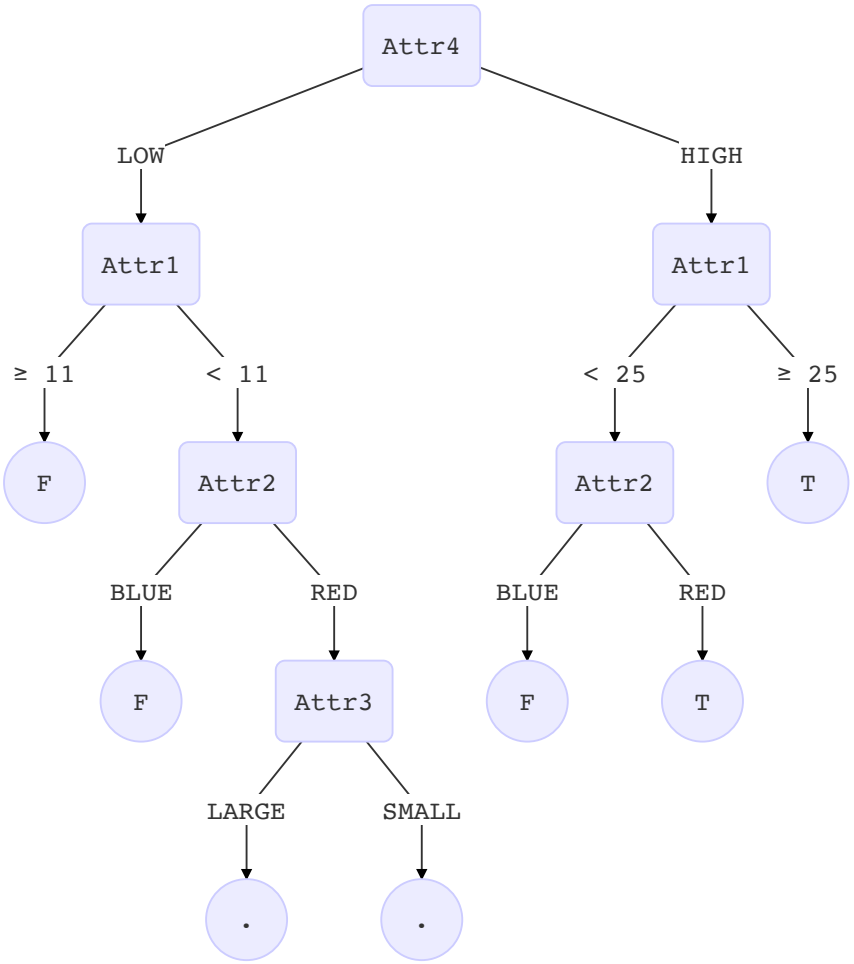
Attr1	Attr2	Attr3	Attr4	Attr5	Class
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T

$H(Class|Attr4 = low, Attr2 = red, Attr1 < 11) = -[\frac{1}{2}\log_2(\frac{1}{2}) + \frac{1}{2}\log_2(\frac{1}{2})] = 1.0000$

$H(Class|Attr4 = low, Attr2 = red, Attr1 < 11, Attr3) = \frac{1}{2}[-(\frac{1}{1}log_2(\frac{1}{1})] + \frac{1}{2}[-(\frac{1}{1}log_2(\frac{1}{1})] = 0.0000$
$IG(Class|Attr4 = low, Attr2 = red, Attr1 < 11, Attr3) = 1.000$

$H(Class|Attr4 = low, Attr2 = red, Attr1 < 11, Attr5) = \frac{2}{2}[-(\frac{1}{2}log_2(\frac{1}{2}) + \frac{1}{2}log_2(\frac{1}{2}))] = 1.0000$
$IG(Class|Attr4 = low, Attr2 = red, Attr1 < 11, Attr5) = 0.0000$

So we split on Attr3.



Level 4 Split

Attr4=low, Attr2=red, Attr1<11, Attr3=large

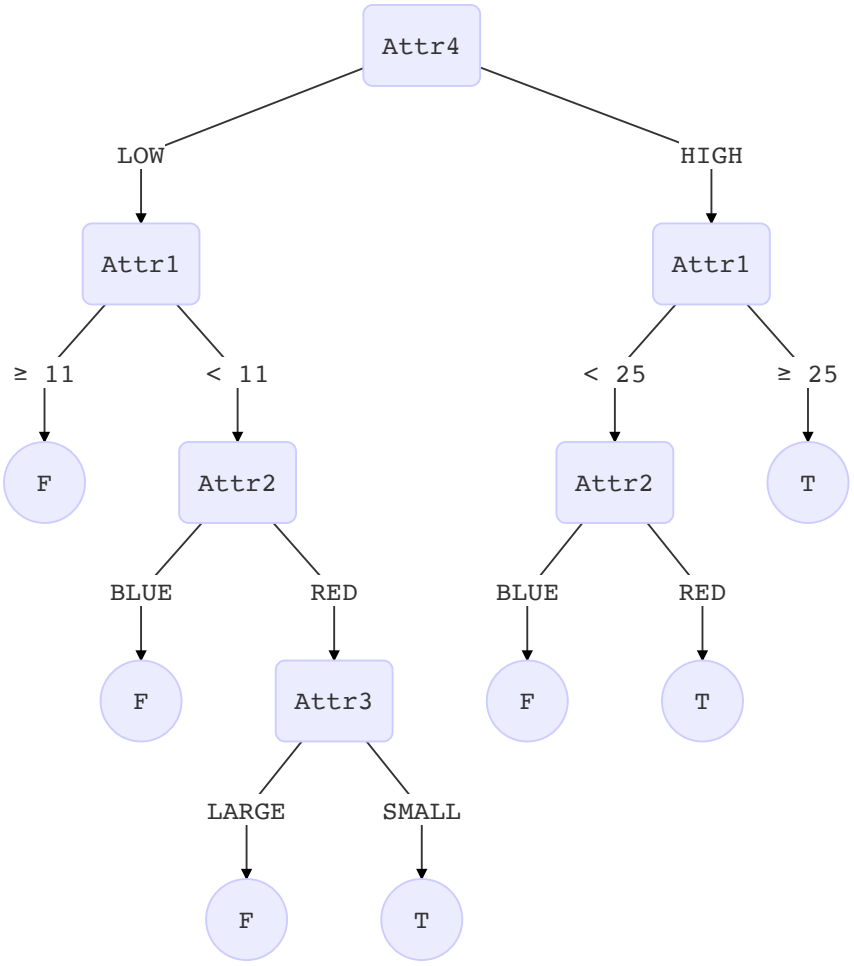
Attr1	Attr2	Attr3	Attr4	Attr5	Class
2	RED	LARGE	LOW	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low, Attr2=red, Attr1<11, Attr3=small

Attr1	Attr2	Attr3	Attr4	Attr5	Class
6	RED	SMALL	LOW	COOL	T

As all labels are same(T), this is a leaf node. So we don't proceed further along this branch.



(B) GINI

Gini Score of Class attribute (9F and 7T) -

$Gini = 1 - (\frac{9}{16})^2 - (\frac{7}{16})^2 = 0.4922$

Root Split

Initial Attribute Specific Gini Splits (6 as split for Attr1) -

$Gini_{split}(Attr1) = \frac{2}{16}[1 - (\frac{0}{2})^2 - (\frac{2}{2})^2] + \frac{14}{16}[1 - (\frac{7}{14})^2 + (\frac{7}{14})^2] = 0.4375$

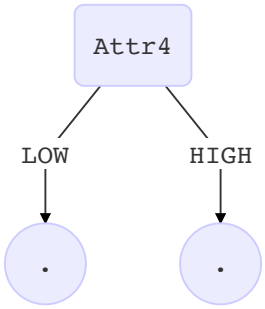
$Gini_{split}(Attr2) = \frac{1}{2}[1 - (\frac{3}{8})^2 - (\frac{5}{8})^2] + \frac{1}{2}[1 - (\frac{4}{8})^2 + (\frac{4}{8})^2] = 0.4843$

$Gini_{split}(Attr3) = \frac{1}{2}[1 - (\frac{3}{8})^2 - (\frac{5}{8})^2] + \frac{1}{2}[1 - (\frac{4}{8})^2 + (\frac{4}{8})^2] = 0.4843$

$Gini_{split}(Attr4) = \frac{1}{2}[1 - (\frac{1}{8})^2 - (\frac{7}{8})^2] + \frac{1}{2}[1 - (\frac{1}{4})^2 + (\frac{3}{4})^2] = 0.2969$

$Gini_{split}(Attr5) = \frac{1}{2}[1 - (\frac{3}{8})^2 - (\frac{5}{8})^2] + \frac{1}{2}[1 - (\frac{1}{4})^2 + (\frac{3}{4})^2] = 0.4218$

The lowest GINI split score is for Attr4. So we split on Attr4.



Level 1 Split

Attr4=low

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F
16	BLUE	LARGE	LOW	COOL	F
33	BLUE	SMALL	LOW	HOT	F
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T
11	RED	SMALL	LOW	COOL	F
45	RED	SMALL	LOW	HOT	F
50	RED	LARGE	LOW	HOT	F

$Gini(Attr4 = low) = 1 - (\frac{1}{8})^2 - (\frac{7}{8})^2 = 0.2187$

Splitting at 11 for Attr1

$Gini_{split}(Attr1|Attr4 = low) = \frac{3}{8}[1 - (\frac{1}{3})^2 - (\frac{2}{3})^2] + \frac{5}{8}[1 - (\frac{5}{5})^2] = 0.1667$

$Gini_{split}(Attr2|Attr4 = low) = \frac{3}{8}[1 - (\frac{3}{3})^2] + \frac{5}{8}[1 - (\frac{4}{5})^2 + (\frac{1}{5})^2] = 0.2000$

$Gini_{split}(Attr3|Attr4 = low) = \frac{3}{8}[1 - (\frac{3}{3})^2] + \frac{5}{8}[1 - (\frac{4}{5})^2 + (\frac{1}{5})^2] = 0.2000$

$Gini_{split}(Attr5|Attr4 = low) = \frac{3}{8}[1 - (\frac{3}{3})^2] + \frac{5}{8}[1 - (\frac{4}{5})^2 + (\frac{1}{5})^2] = 0.2000$

So we split at Attr1 at value 11

Attr4=high

Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F
27	BLUE	LARGE	HIGH	HOT	T
29	BLUE	SMALL	HIGH	HOT	T
34	BLUE	LARGE	HIGH	HOT	T
10	RED	SMALL	HIGH	COOL	T
25	RED	SMALL	HIGH	HOT	T
36	RED	LARGE	HIGH	HOT	T

$Gini(Attr4 = high) = 1 - (\frac{1}{4})^2 - (\frac{3}{4})^2 = 0.375$

Splitting at 25 for Attr1

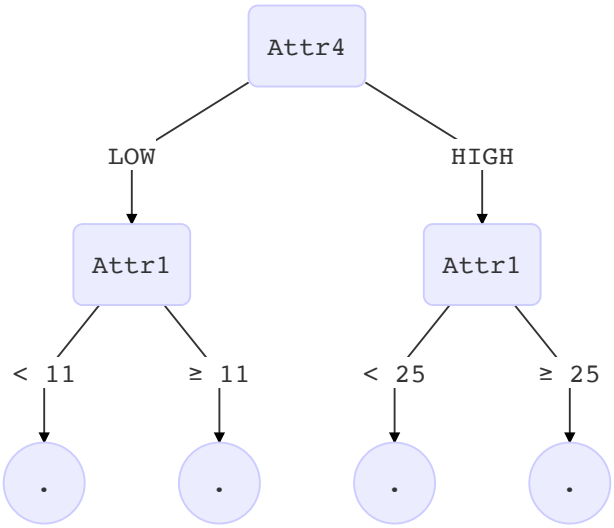
$Gini_{split}(Attr1|Attr4 = high) = \frac{3}{8}[1 - (\frac{1}{3})^2 - (\frac{2}{3})^2] + \frac{5}{8}[1 - (\frac{5}{5})^2] = 0.1667$

$Gini_{split}(Attr2|Attr4 = high) = \frac{3}{8}[1 - (\frac{3}{3})^2] + \frac{5}{8}[1 - (\frac{2}{5})^2 + (\frac{3}{5})^2] = 0.3000$

$Gini_{split}(Attr3|Attr4 = high) = \frac{3}{8}[1 - (\frac{3}{3})^2] + \frac{5}{8}[1 - (\frac{2}{5})^2 + (\frac{3}{5})^2] = 0.3000$

$Gini_{split}(Attr5|Attr4 = high) = \frac{3}{8}[1 - (\frac{1}{3})^2 - (\frac{2}{3})^2] + \frac{5}{8}[1 - (\frac{5}{5})^2] = 1.667$

So we split at Attr1 at value 25



Level 2 Split

Attr4=low, Attr1<11

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T

$Gini(Attr4 = low, Attr1 < 11) = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.4444$

$Gini_{split}(Attr2|Attr4 = low, Attr1 < 11) = \frac{1}{3}[1 - (\frac{1}{1})^2] + \frac{2}{3}[1 - (\frac{1}{2})^2 + (\frac{1}{2})^2] = 0.3333$

$Gini_{split}(Attr3|Attr4 = low, Attr1 < 11) = \frac{1}{3}[1 - (\frac{1}{1})^2] + \frac{2}{3}[1 - (\frac{1}{2})^2 + (\frac{1}{2})^2] = 0.3333$

$Gini_{split}(Attr5|Attr4 = low, Attr1 < 11) = \frac{1}{1}[1 - (\frac{1}{3})^2 + (\frac{2}{3})^2] = 0.4444$

So we split on Attr2

Attr4=low, Attr1≥11

Attr1	Attr2	Attr3	Attr4	Attr5	Class
16	BLUE	LARGE	LOW	COOL	F
33	BLUE	SMALL	LOW	HOT	F
11	RED	SMALL	LOW	COOL	F
45	RED	SMALL	LOW	HOT	F
50	RED	LARGE	LOW	HOT	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=high, Attr1<25

Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F
10	RED	SMALL	HIGH	COOL	T

$Gini(Attr4 = high, Attr1 < 25) = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.4444$

$Gini_{split}(Attr2|Attr4 = high, Attr1 < 25) = \frac{1}{3}[1 - (\frac{1}{1})^2] + \frac{2}{3}[1 - (\frac{1}{1})^2] = 0.0000$

$Gini_{split}(Attr3|Attr4 = high, Attr1 < 25) = \frac{1}{3}[1 - (\frac{1}{1})^2] + \frac{2}{3}[1 - (\frac{1}{1})^2] = 0.0000$

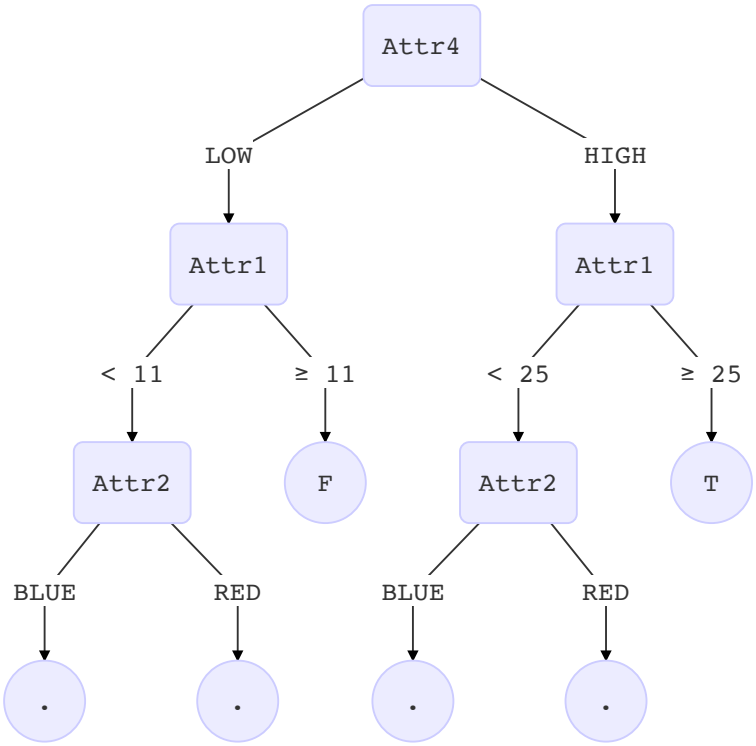
$Gini_{split}(Attr5|Attr4 = high, Attr1 < 25) = \frac{1}{1}[1 - (\frac{1}{3})^2 + (\frac{2}{3})^2] = 0.4444$

So we split on Attr2

Attr4=high, Attr1≥25

Attr1	Attr2	Attr3	Attr4	Attr5	Class
27	BLUE	LARGE	HIGH	HOT	T
29	BLUE	SMALL	HIGH	HOT	T
34	BLUE	LARGE	HIGH	HOT	T
25	RED	SMALL	HIGH	HOT	T
36	RED	LARGE	HIGH	HOT	T

As all labels are same(T), this is a leaf node. So we don't proceed further along this branch.



Level 3 Split

Attr4=low, Attr1<11, Attr2=blue

Attr1	Attr2	Attr3	Attr4	Attr5	Class
3	BLUE	SMALL	LOW	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low, Attr1<11, Attr2=red

Attr1	Attr2	Attr3	Attr4	Attr5	Class
2	RED	LARGE	LOW	COOL	F
6	RED	SMALL	LOW	COOL	T

$Gini(Attr4 = low, Attr1 < 11, Attr2 = red) = 1 - (\frac{1}{3})^2 - (\frac{2}{3})^2 = 0.4444$

$Gini_{split}(Attr3|Attr4 = low, Attr1 < 11, Attr2 = red) = \frac{1}{2}[1 - (\frac{1}{1})^2] + \frac{1}{2}[1 - (\frac{1}{1})^2] = 0.0000$

$Gini_{split}(Attr5|Attr4 = low, Attr1 < 11, Attr2 = red) = \frac{1}{1}[1 - (\frac{1}{2})^2 + (\frac{1}{2})^2] = 0.5000$

So we split on Attr3

Attr4=high, Attr1<25, Attr2=blue

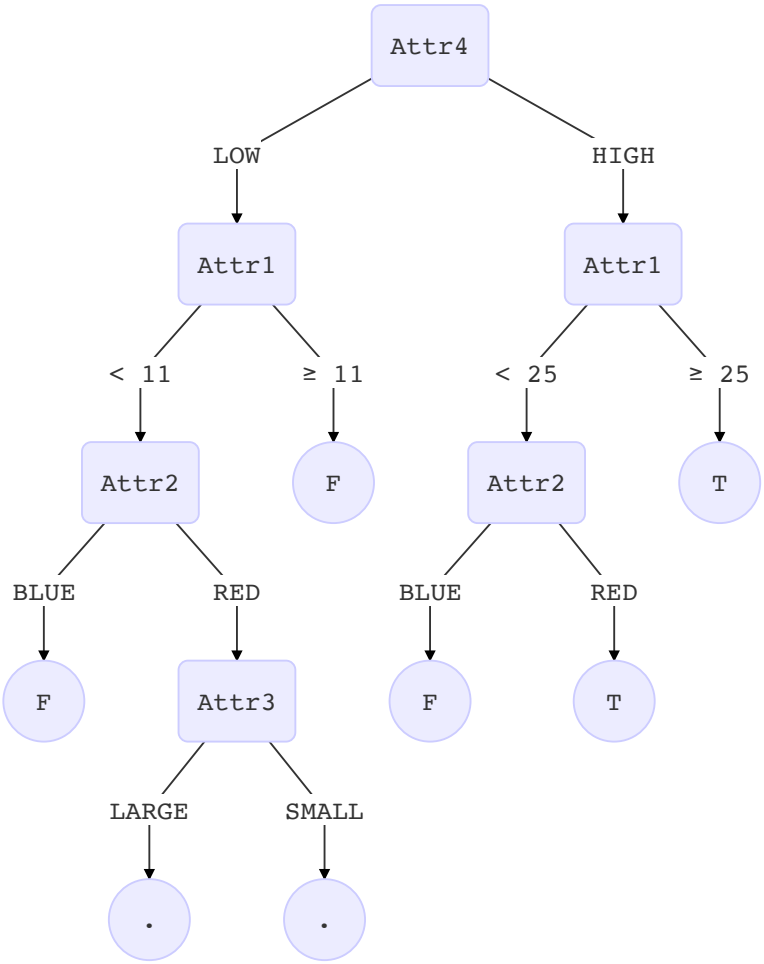
Attr1	Attr2	Attr3	Attr4	Attr5	Class
7	BLUE	LARGE	HIGH	COOL	F
17	BLUE	LARGE	HIGH	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=high, Attr1<25, Attr2=red

Attr1	Attr2	Attr3	Attr4	Attr5	Class
10	RED	SMALL	HIGH	COOL	T

As all labels are same(T), this is a leaf node. So we don't proceed further along this branch.



Split Level 4

Attr4=low, Attr1<11, Attr2=red, Attr3=large

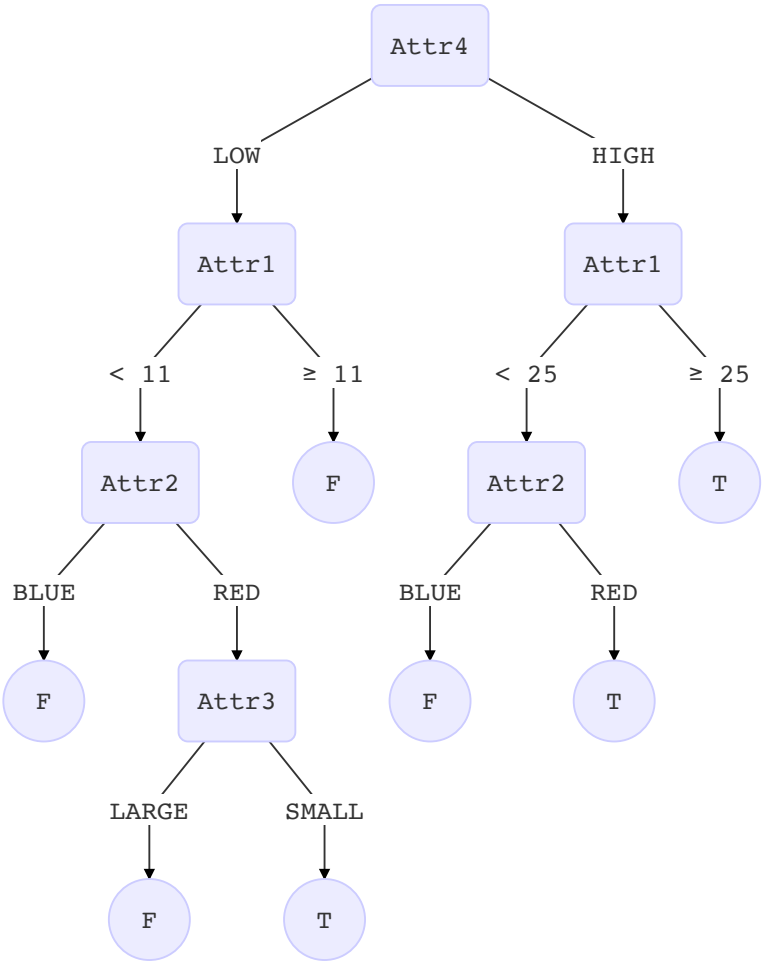
Attr1	Attr2	Attr3	Attr4	Attr5	Class
2	RED	LARGE	LOW	COOL	F

As all labels are same(F), this is a leaf node. So we don't proceed further along this branch.

Attr4=low, Attr1<11, Attr2=red, Attr3=small

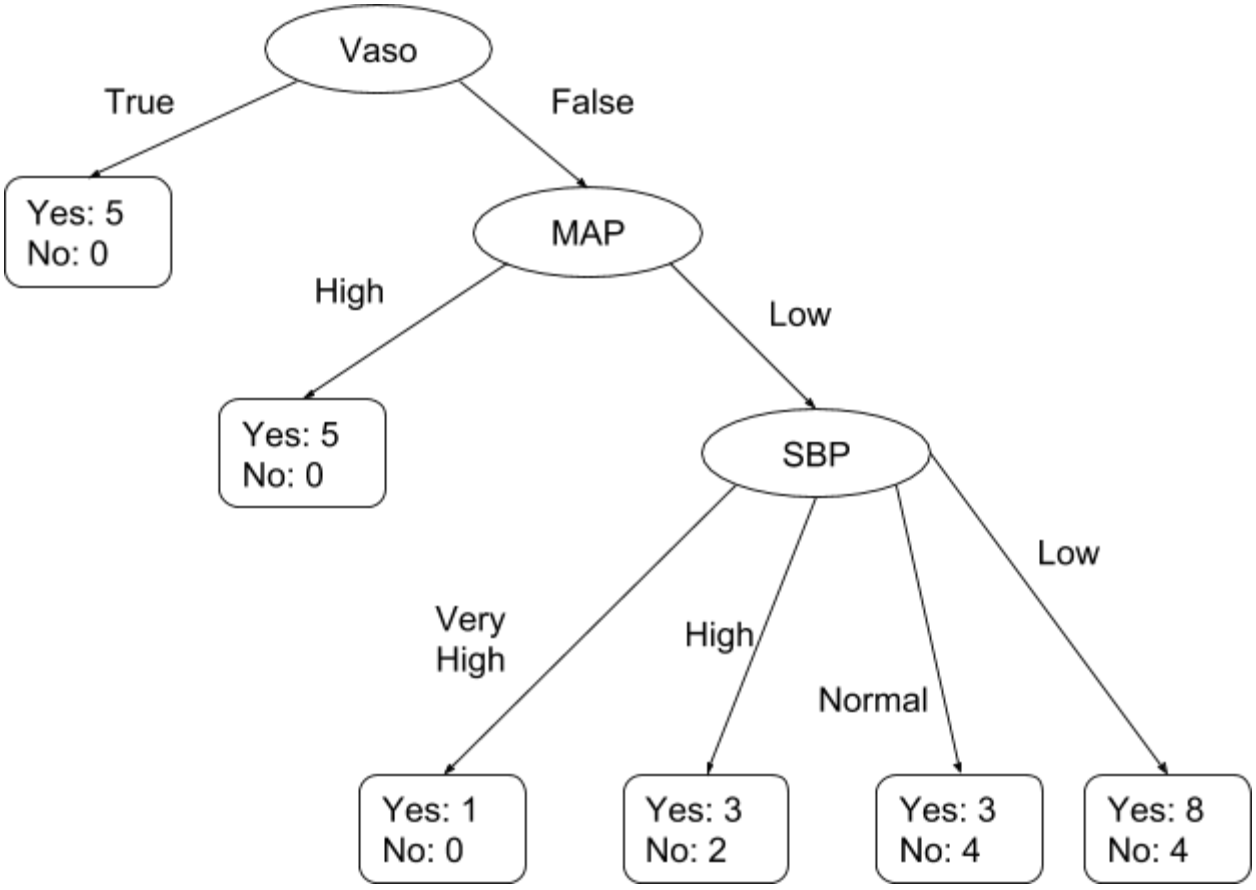
Attr1	Attr2	Attr3	Attr4	Attr5	Class
6	RED	SMALL	LOW	COOL	T

As all labels are same(T), this is a leaf node. So we don't proceed further along this branch.



Khantil Choksi - khchoksi
Aman Chauhan - achauha3

Problem 3:



(a) Post-pruning based on optimistic errors

(i) Optimistic Errors:

Before Splitting	After Splitting
$\frac{10}{25}$	$\frac{9}{25}$

(ii) Based upon the optimistic erros, the subtree should not be pruned. As after splitting based on SBP, the optimistic error reduces. Therefore, we keep the given decision tree retained.

(iii) Classified provided testing dataset (“hw2q3_test.csv”):

Vaso	MAP	SBP	Sepsis Shock	Predicted Class (Based on given retained decision tree)
FALSE	Low	High	Yes	Yes
FALSE	Low	High	Yes	Yes
FALSE	Low	High	Yes	Yes
FALSE	Low	High	No	Yes
TRUE	High	High	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
TRUE	Low	Low	Yes	Yes
FALSE	High	Low	Yes	Yes
FALSE	Low	Normal	Yes	No
FALSE	Low	Normal	Yes	No
FALSE	Low	Normal	No	No
FALSE	Low	Normal	No	No
FALSE	Low	Normal	No	No
FALSE	Low	Normal	No	No
TRUE	High	Normal	Yes	Yes
FALSE	Low	Very High	Yes	Yes
TRUE	Low	Very High	Yes	Yes

Performance Matrix

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	13 (TP) (a)	2 (FN) (b)
	Class = No	1 (FP) (c)	4 (TN) (d)

Accuracy = $\frac{a+d}{a+b+c+d} = \frac{13+4}{13+2+1+4} = \frac{17}{20} = 0.85$

Recall = $\frac{a}{a+b} = \frac{13}{13+2} = \frac{13}{15} = 0.867$

Precision = $\frac{a}{a+c} = \frac{13}{13+1} = \frac{13}{14} = 0.93$

Specificity (TNR) = $\frac{TN}{N} = \frac{TN}{TN+FP} = \frac{4}{4+1} = \frac{4}{5} = 0.8$

Sensitivity (TPR) = $\frac{TP}{P} = \frac{TP}{TP+FN} = \frac{13}{13+2} = \frac{13}{15} = 0.867$

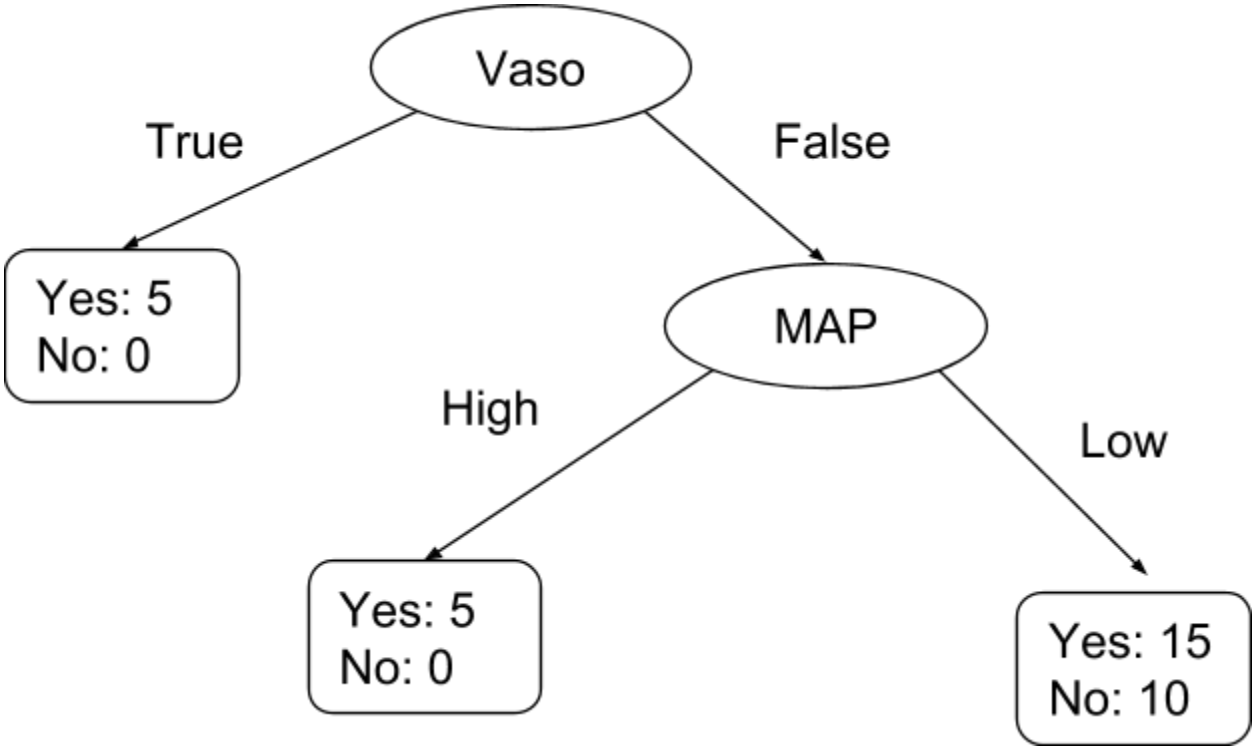
F1 Measure = $2 * \frac{precision * recall}{precision + recall} = 2 * \frac{0.93 * 0.867}{0.93 + 0.867} = 0.8965$

(b) Post-pruning based on pessimistic errors

(i) Pessimistic Errors:

Before Splitting	After Splitting
$\frac{(10 + 1 * 0.5)}{25} = \frac{10.5}{25}$	$\frac{(9 + 4 * 0.5)}{25} = \frac{11}{25}$

(ii) Based upon the pessimistic erros, the subtree should be pruned. As after splitting based on SBP, the pessimistic error increases. Therefore, we prune the given decision tree from SBP. Pruned Decision Tree is as shown below:



(iii) Classified provided testing dataset ("hw2q3_test.csv"):

Vaso	MAP	SBP	Sepsis Shock	Predicted Class (Has Septic Shock)
FALSE	Low	High	Yes	Yes
FALSE	Low	High	Yes	Yes
FALSE	Low	High	Yes	Yes
FALSE	Low	High	No	Yes
TRUE	High	High	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
FALSE	Low	Low	Yes	Yes
TRUE	Low	Low	Yes	Yes

FALSE	High	Low	Yes	Yes
FALSE	Low	Normal	Yes	Yes
FALSE	Low	Normal	Yes	Yes
FALSE	Low	Normal	No	Yes
FALSE	Low	Normal	No	Yes
FALSE	Low	Normal	No	Yes
FALSE	Low	Normal	No	Yes
TRUE	High	Normal	Yes	Yes
FALSE	Low	Very High	Yes	Yes
TRUE	Low	Very High	Yes	Yes

Performance Matrix

		Predicted Class	
		Class = Yes	Class = No
Actual Class	Class = Yes	15 (TP) (a)	0 (FN) (b)
	Class = No	5 (FP) (c)	0 (TN) (d)

Accuracy = $\frac{a+d}{a+b+c+d}$ = $\frac{15+0}{15+0+5+0}$ = $\frac{15}{20}$ = 0.75

Recall = $\frac{a}{a+b}$ = $\frac{15}{15+0}$ = $\frac{15}{15}$ = 1

Precision = $\frac{a}{a+c}$ = $\frac{15}{20}$ = $\frac{15}{20}$ = 0.75

Specificity (TNR) = $\frac{TN}{N}$ = $\frac{TN}{TN+FP}$ = $\frac{0}{0+5}$ = $\frac{0}{5}$ = 0

Sensitivity (TPR) = $\frac{TP}{P}$ = $\frac{TP}{TP+FN}$ = $\frac{15}{15+0}$ = $\frac{15}{15}$ = 1

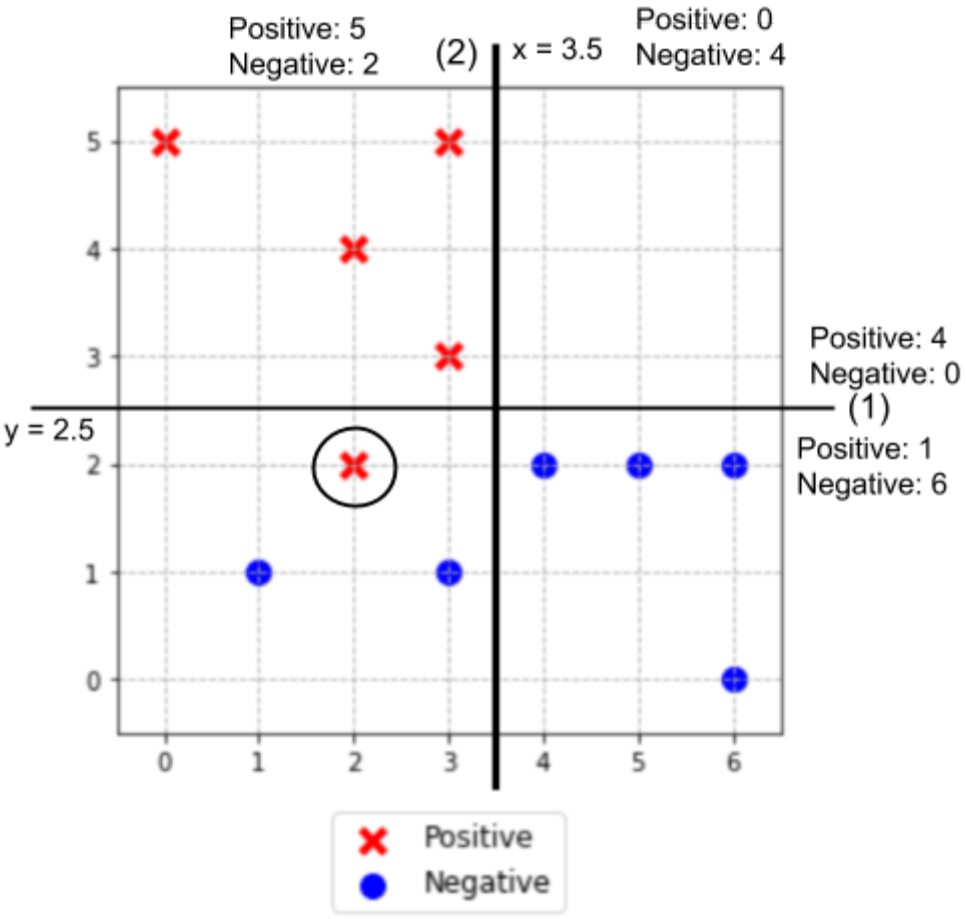
F1 Measure = 2 * $\frac{precision * recall}{precision + recall}$ = 2 * $\frac{0.75 * 1}{0.75 + 1}$ = 0.857

(c) Performance Comparison

	Before Pruning	After Pruning
Accuracy	0.85	0.75
Recall	0.867	1
Precision	0.93	0.75

Here, as the problem mentioned that the sepsis deaths could be prevented with early diagnosis and intervention. So, it is very important to measure TP and TNs are very costly for this example. So, we more care about recall. So, after pruning we have higher recall. So problem 3 (b) (ii) decision tree would be better model for the task of septic shock prediction.

Problem 4:


Positive: 5
Negative: 2
(2)
x = 3.5
Positive: 0
Negative: 4
Positive: 4
Negative: 0
(1)
y = 2.5
Positive: 1
Negative: 6
Positive
Negative

Q4 (C)

Import Libraries

```
import math
```

Iteration 1

```
init_weight = 1 / 11
print(f'Inital Weight: {init_weight}')
```

```
Inital Weight: 0.09090909090909091
```

```
epsilon = 1 * init_weight
print(f'Epsilon: {epsilon}')
```

```
Epsilon: 0.09090909090909091
```

```
alpha = 0.5 * math.log((1-epsilon)/epsilon)
print(f'Alpha: {alpha}')
```

```
Alpha: 1.151292546497023
```

Correctly Classified Points

```
exp_correct = init_weight * math.exp(-1*alpha)
print(f'Correctly Classified  : {exp_correct}')
```

```
Correctly Classified  : 0.028747978728803445
```

```
exp_incorrect = init_weight * math.exp(alpha)
print(f'Incorrectly Classified : {exp_incorrect}')
```

```
Incorrectly Classified : 0.2874797872880345
```

Normalization

```
z = 1 * exp_incorrect + 10 * exp_correct
print(f'Normalization Factor: {z}')
```

```
Normalization Factor: 0.5749595745760689
```

```
w2_correct = exp_correct / z
print(f'W2_Correct: {w2_correct}')
```

```
W2_Correct: 0.049999999999999996
```

```
w2_incorrect = exp_incorrect / z
print(f'W2_InCorrect: {w2_incorrect}')
```

```
W2_InCorrect: 0.5000000000000001
```

Weighted Error

```
weighted_error = w2_incorrect * 1
print(f'Weighted Error: {weighted_error}')
```

```
Weighted Error: 0.5000000000000001
```

Q5 - Naive Bayes and Decision Trees

Import the libraries

```
from sklearn.naive_bayes import MultinomialNB
from IPython.display import display
from IPython.display import SVG
from id3 import export_graphviz
from id3 import Id3Estimator
from sklearn import tree

import matplotlib.pyplot as plt
import numpy as np
import graphviz
import os
%matplotlib inline
```

Load the data

```
def data_and_headers(filename):
    data = None
    with open(filename) as fp:
        data = [x.strip().split(',') for x in fp.readlines()]
    headers = data[0]
    headers = np.asarray(headers)
    class_field = len(headers) - 1
    data_x = [[x[i] for i in range(class_field)] for x in data[1:]]
    data_x = np.asarray(data_x)
    data_y = [[x[i] for i in range(class_field, class_field + 1)] for x in data[1:]]
    data_y = np.asarray(data_y)
    return headers, data_x, data_y
```

```
headers, X, Y = data_and_headers('Data' + os.sep + 'hw2q5.csv')
indexes=[int(x) for x in list(X[:,0])]
X=X[:,1:]
```

(A) K-Fold Splits

```
def createfolds(indexes):
    folds = {i:{'train':[], 'test':[]} for i in range(1,6)}
    for i in range(len(indexes)):
        for j in range(1,6):
            if indexes[i] % 5 == j-1:
                folds[j]['test'].append(i)
            else:
                folds[j]['train'].append(i)
    return folds
```

```
folds = createfolds(indexes)
```

Naive Bayes

```
X = X.tolist()
Y = np.ravel(Y).tolist()
d1={'presbyopic':2, 'pre-presbyopic':1, 'young':0, 'myope':0,
'hypermetrope':1, 'no':0, 'yes':1, 'normal':1, 'reduced':0}
d2={'Yes':1, 'No':0}
X = [[d1[X[i][j]] for j in range(len(X[0]))] for i in range(len(X))]
X = np.asarray(X)
Y = [d2[Y[i]] for i in range(len(Y))]
Y = np.asarray(Y)
```

```
cnt = 0
subheaders = headers[1:-1]
for i in sorted(folds.keys()):
    print('Fold '+str(i))
    nb = MultinomialNB(alpha=1)
    nb=nb.fit(X[folds[i]['train'],:],Y[folds[i]['train']])
    ypred=nb.predict(X[folds[i]['test'],:])
    print('\tTest IID -\t' + ', '.join([str(x) for x in
np.asarray(indexes)[folds[i]['test']]]))
    print('\tActual -\t'+', '.join(['Yes' if x==1 else 'No' for x in
Y[folds[i]['test']]]))
    print('\tPredict -\t' + ', '.join(['Yes' if x==1 else 'No' for x in
ypred]))
    for j in range(len(ypred)):
        if ypred[j]!=Y[folds[i]['test']][j]:
            cnt+=1
    print('\tProbabilities - ')
    dt={0:'Yes', 1:'No'}
    for i in range(len(nb.feature_log_prob_)):
```

```
        for j in range(len(subheaders)):
            print('\t\tP({}|Class={}) = {:.3f}'.format(subheaders[j],
dt[i], np.exp(nb.feature_log_prob_)[i][j]))
print('\nNaive-Bayes 5-fold CV accuracy - '+ str((24-cnt)*100/24) + '%')
```

```
Fold 1
    Test IID -  5, 10, 15, 20
    Actual -    No, Yes, No, Yes
    Predict -   No, No, No, No
    Probabilities -
        P(patient age|Class=Yes) = 0.472
        P(spectacle prescription|Class=Yes) = 0.194
        P(astigmatic|Class=Yes) = 0.222
        P(tear production rate|Class=Yes) = 0.111
        P(patient age|Class=No) = 0.227
        P(spectacle prescription|Class=No) = 0.227
        P(astigmatic|Class=No) = 0.182
        P(tear production rate|Class=No) = 0.364
Fold 2
    Test IID -  1, 6, 11, 16, 21
    Actual -    No, Yes, No, No, No
    Predict -   No, Yes, No, Yes, No
    Probabilities -
        P(patient age|Class=Yes) = 0.452
        P(spectacle prescription|Class=Yes) = 0.226
        P(astigmatic|Class=Yes) = 0.226
        P(tear production rate|Class=Yes) = 0.097
        P(patient age|Class=No) = 0.308
        P(spectacle prescription|Class=No) = 0.154
        P(astigmatic|Class=No) = 0.192
        P(tear production rate|Class=No) = 0.346
Fold 3
    Test IID -  2, 7, 12, 17, 22
    Actual -    Yes, No, Yes, No, Yes
    Predict -   Yes, No, No, No, No
    Probabilities -
        P(patient age|Class=Yes) = 0.444
        P(spectacle prescription|Class=Yes) = 0.222
        P(astigmatic|Class=Yes) = 0.222
        P(tear production rate|Class=Yes) = 0.111
        P(patient age|Class=No) = 0.250
        P(spectacle prescription|Class=No) = 0.200
        P(astigmatic|Class=No) = 0.200
        P(tear production rate|Class=No) = 0.350
Fold 4
```

```
    Test IID -  3, 8, 13, 18, 23
    Actual -    No, Yes, No, No, No
    Predict -   No, Yes, No, Yes, No
    Probabilities -
        P(patient age|Class=Yes) = 0.433
        P(spectacle prescription|Class=Yes) = 0.233
        P(astigmatic|Class=Yes) = 0.233
        P(tear production rate|Class=Yes) = 0.100
        P(patient age|Class=No) = 0.320
        P(spectacle prescription|Class=No) = 0.160
        P(astigmatic|Class=No) = 0.160
        P(tear production rate|Class=No) = 0.360
Fold 5
    Test IID -  4, 9, 14, 19, 24
    Actual -    Yes, No, Yes, No, No
    Predict -   Yes, No, Yes, No, No
    Probabilities -
        P(patient age|Class=Yes) = 0.419
        P(spectacle prescription|Class=Yes) = 0.258
        P(astigmatic|Class=Yes) = 0.226
        P(tear production rate|Class=Yes) = 0.097
        P(patient age|Class=No) = 0.304
        P(spectacle prescription|Class=No) = 0.174
        P(astigmatic|Class=No) = 0.174
        P(tear production rate|Class=No) = 0.348

Naive-Bayes 5-fold CV accuracy - 75.0%
```

Decision Tree

```
headers, X, Y = data_and_headers('Data' + os.sep + 'hw2q5.csv')
indexes=[int(x) for x in list(X[:,0])]
X=X[:,1:]
```

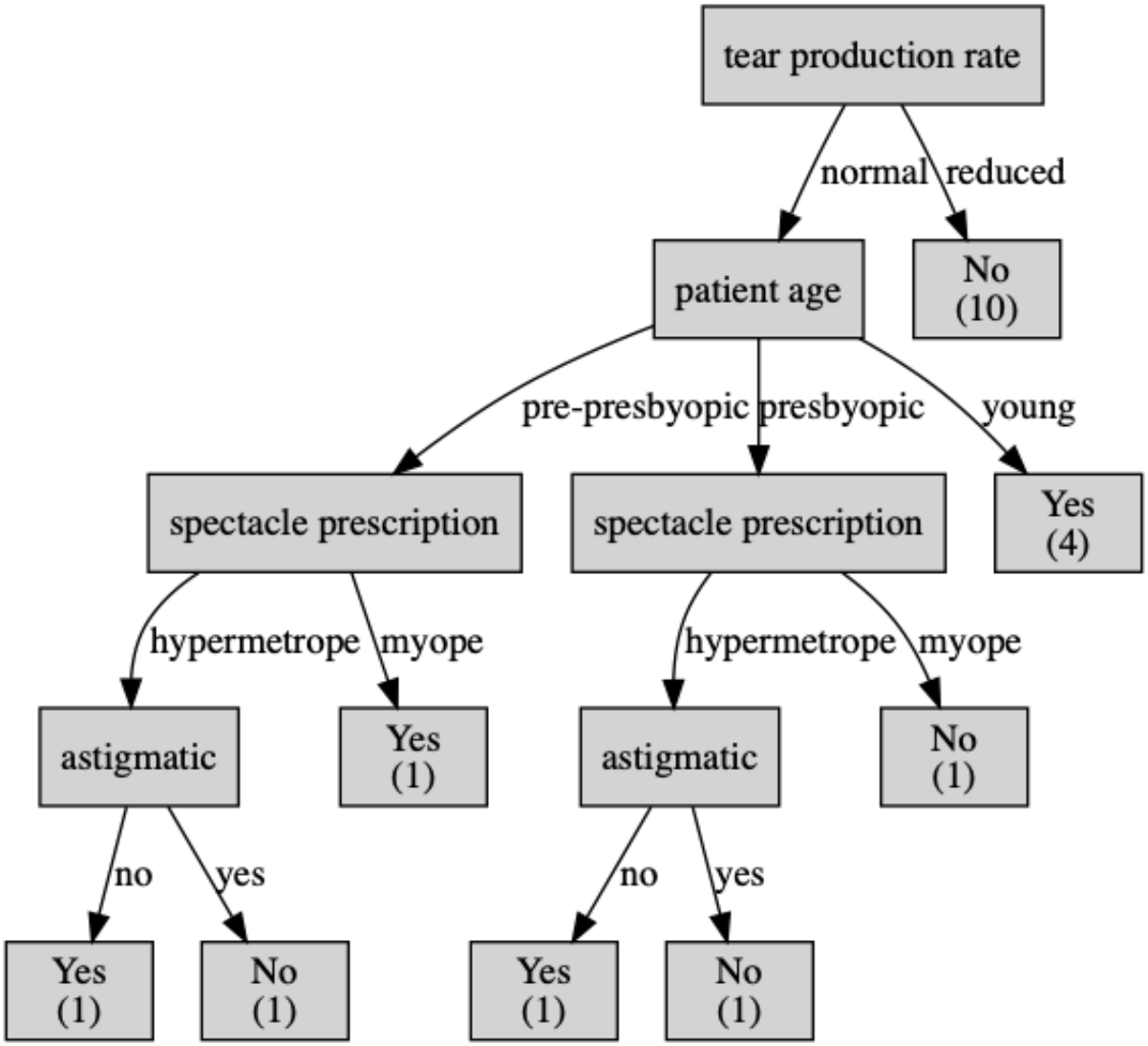
```
cnt = 0
subheaders = headers[1:-1]
for i in sorted(folds.keys()):
    print('Fold '+str(i))
    #dt = tree.DecisionTreeClassifier(criterion='entropy',
splitter='best')
    dt = Id3Estimator(gain_ratio=True)
    dt = dt.fit(X[folds[i]['train'],:],Y[folds[i]['train']])
    ypred=dt.predict(X[folds[i]['test'],:])
```

```
    print('\tTest IID -\t' + ', '.join([str(x) for x in
np.asarray(indexes)[folds[i]['test']]]))
    print('\tActual -\t'+', '.join(np.ravel(Y[folds[i]['test']])))
    print('\tPredict -\t' + ', '.join(ypred))
#     print('\tActual -\t'+', '.join(['Yes' if x==1 else 'No' for x in
Y[folds[i]['test']]]))
#     print('\tPredict -\t' + ', '.join(['Yes' if x==1 else 'No' for x in
ypred]))
    for j in range(len(ypred)):
        if ypred[j]!=Y[folds[i]['test']][j]:
            cnt+=1
    dot_data = export_graphviz(dt.tree_,
'fold'+str(i)+'.dot',feature_names = subheaders)
print('\nDecision Tree 5-fold CV accuracy - '+ str((24-cnt)*100/24) + '%')
```

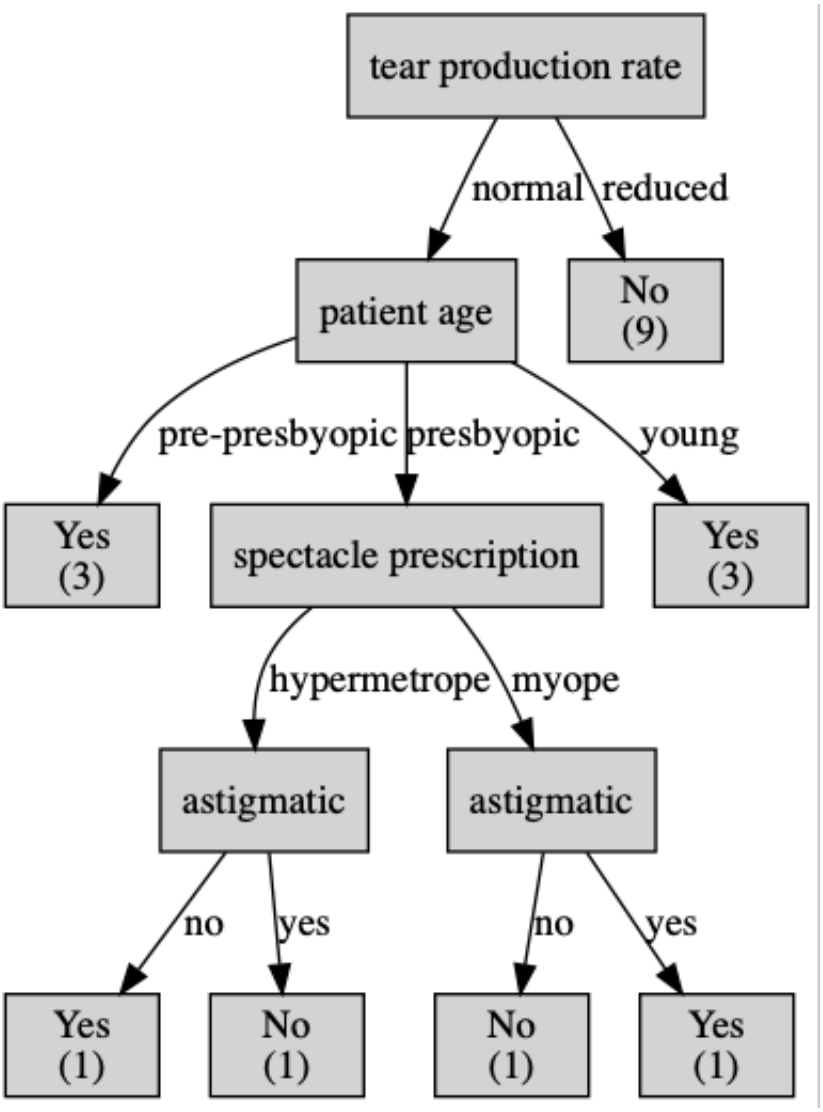
```
Fold 1
    Test IID -  5, 10, 15, 20
    Actual -    No, Yes, No, Yes
    Predict -   No, Yes, No, No
Fold 2
    Test IID -  1, 6, 11, 16, 21
    Actual -    No, Yes, No, No, No
    Predict -   No, Yes, No, Yes, No
Fold 3
    Test IID -  2, 7, 12, 17, 22
    Actual -    Yes, No, Yes, No, Yes
    Predict -   Yes, No, No, No, No
Fold 4
    Test IID -  3, 8, 13, 18, 23
    Actual -    No, Yes, No, No, No
    Predict -   No, No, No, Yes, No
Fold 5
    Test IID -  4, 9, 14, 19, 24
    Actual -    Yes, No, Yes, No, No
    Predict -   Yes, No, No, No, Yes

Decision Tree 5-fold CV accuracy - 66.66666666666667%
```

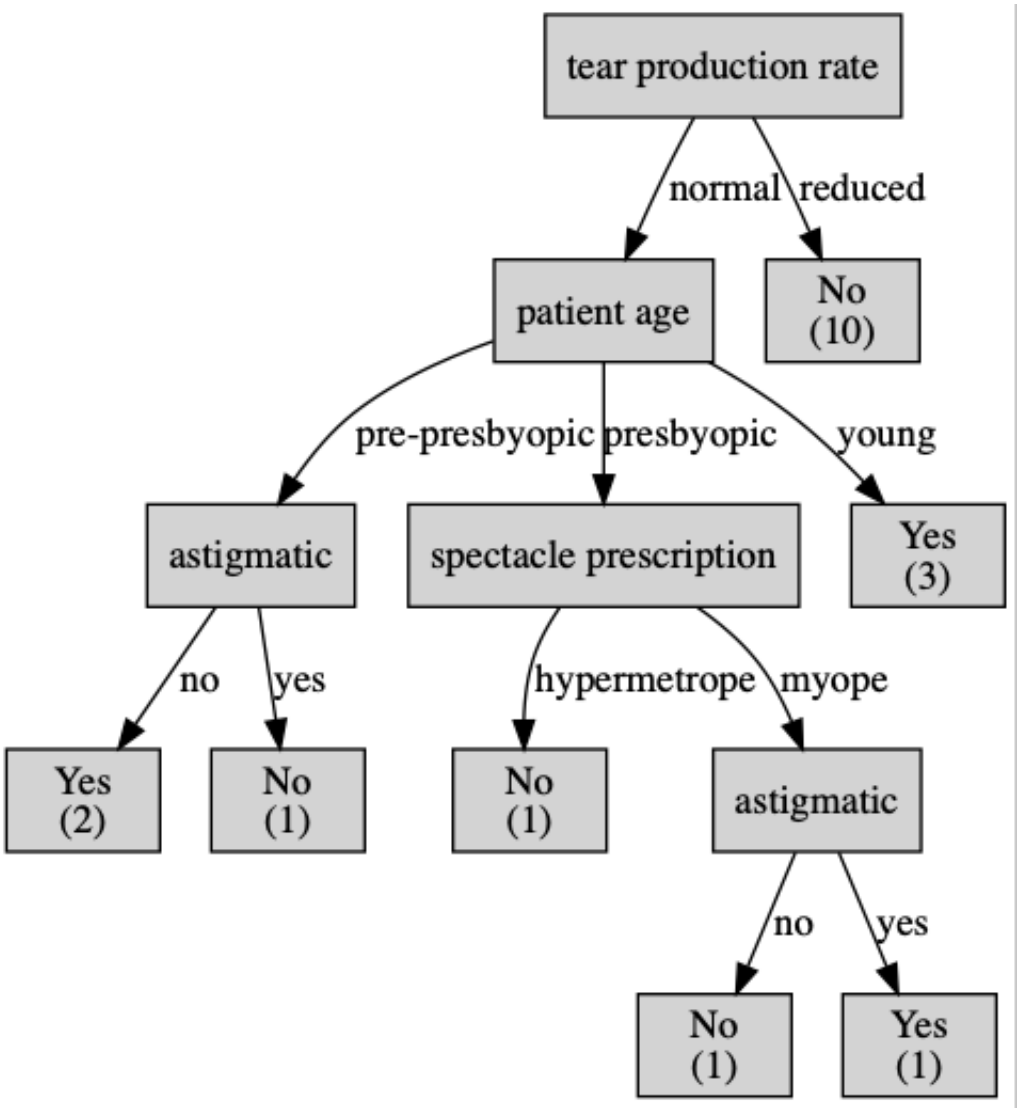
Fold 1


tear production rate
normal
reduced
patient age
No
(10)
pre-presbyopic
presbyopic
young
spectacle prescription
spectacle prescription
Yes
(4)
hypermetrope
myope
hypermetrope
myope
astigmatic
Yes
(1)
astigmatic
No
(1)
no
yes
no
yes
Yes
(1)
No
(1)
Yes
(1)
No
(1)

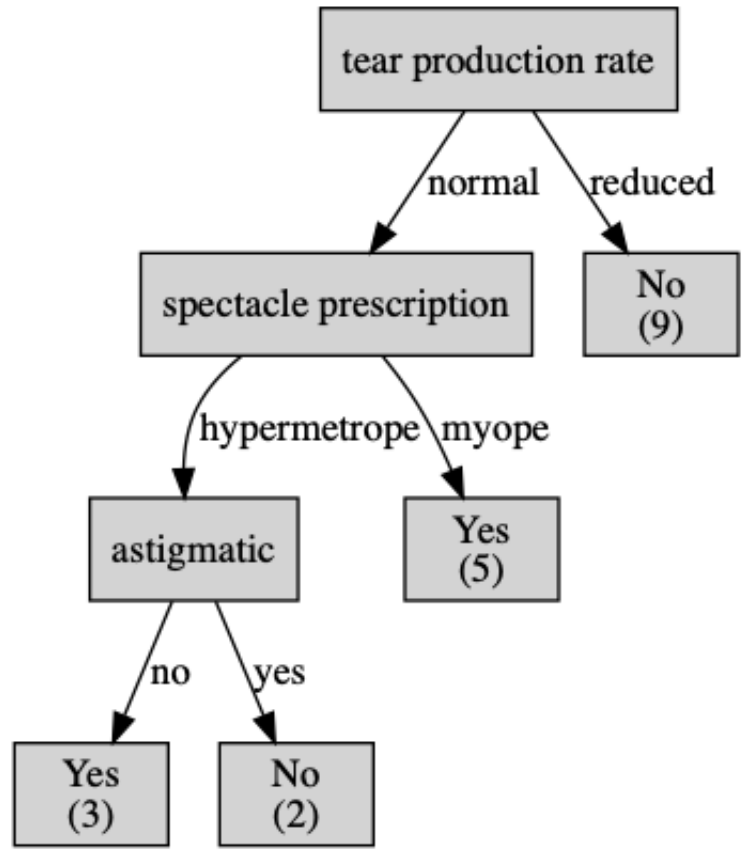
Fold 2


tear production rate
normal
reduced
patient age
No
(9)
pre-presbyopic
presbyopic
young
Yes
(3)
spectacle prescription
Yes
(3)
hypermetrope
myope
astigmatic
astigmatic
no
yes
no
yes
Yes
(1)
No
(1)
No
(1)
Yes
(1)

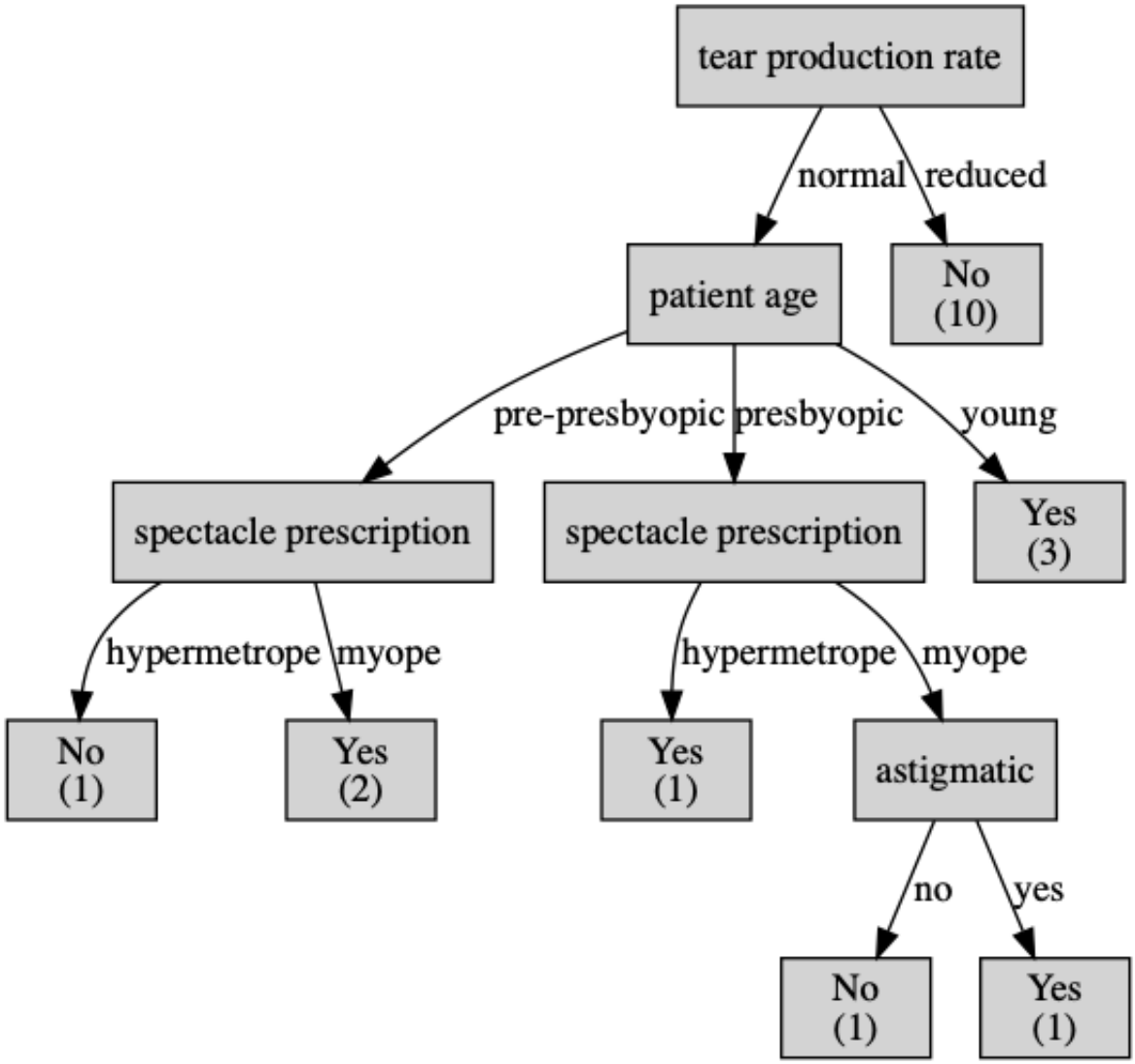
Fold 3


tear production rate
normal
reduced
patient age
No
(10)
pre-presbyopic
presbyopic
young
astigmatic
spectacle prescription
Yes
(3)
no
yes
hypermetrope
myope
Yes
(2)
No
(1)
No
(1)
astigmatic
no
yes
No
(1)
Yes
(1)

Fold 4


tear production rate
normal
reduced
spectacle prescription
No
(9)
hypermetrope
myope
astigmatic
Yes
(5)
no
yes
Yes
(3)
No
(2)

Fold 5



(B) Choosing Model

Based on the above 5-fold CV accuracy, it seems that Naive Bayes is the better model for this dataset.

Naive Bayes for Full Data

```
X = X.tolist()
Y = np.ravel(Y).tolist()
d1={'presbyopic':2, 'pre-presbyopic':1, 'young':0, 'myope':0,
'hypermetrope':1, 'no':0, 'yes':1, 'normal':1, 'reduced':0}
d2={'Yes':1, 'No':0}
X = [[d1[X[i][j]] for j in range(len(X[0]))] for i in range(len(X))]
X = np.asarray(X)
Y = [d2[Y[i]] for i in range(len(Y))]
Y = np.asarray(Y)
```

```
nb = MultinomialNB(alpha=1)
nb=nb.fit(X,Y)
print('Final Model for Naive Bayes')
print('Probabilities - ')
dt={0:'Yes', 1:'No'}
for i in range(len(nb.feature_log_prob_)):
    for j in range(len(subheaders)):
        print('\tP({}|Class={}) = {:.3f}'.format(subheaders[j], dt[i],
np.exp(nb.feature_log_prob_)[i][j]))
```

Model Details -

```
Final Model for Naive Bayes
Probabilities -
    P(patient age|Class=Yes) = 0.450
    P(spectacle prescription|Class=Yes) = 0.225
    P(astigmatic|Class=Yes) = 0.225
    P(tear production rate|Class=Yes) = 0.100
    P(patient age|Class=No) = 0.286
    P(spectacle prescription|Class=No) = 0.179
    P(astigmatic|Class=No) = 0.179
    P(tear production rate|Class=No) = 0.357
```

Q6

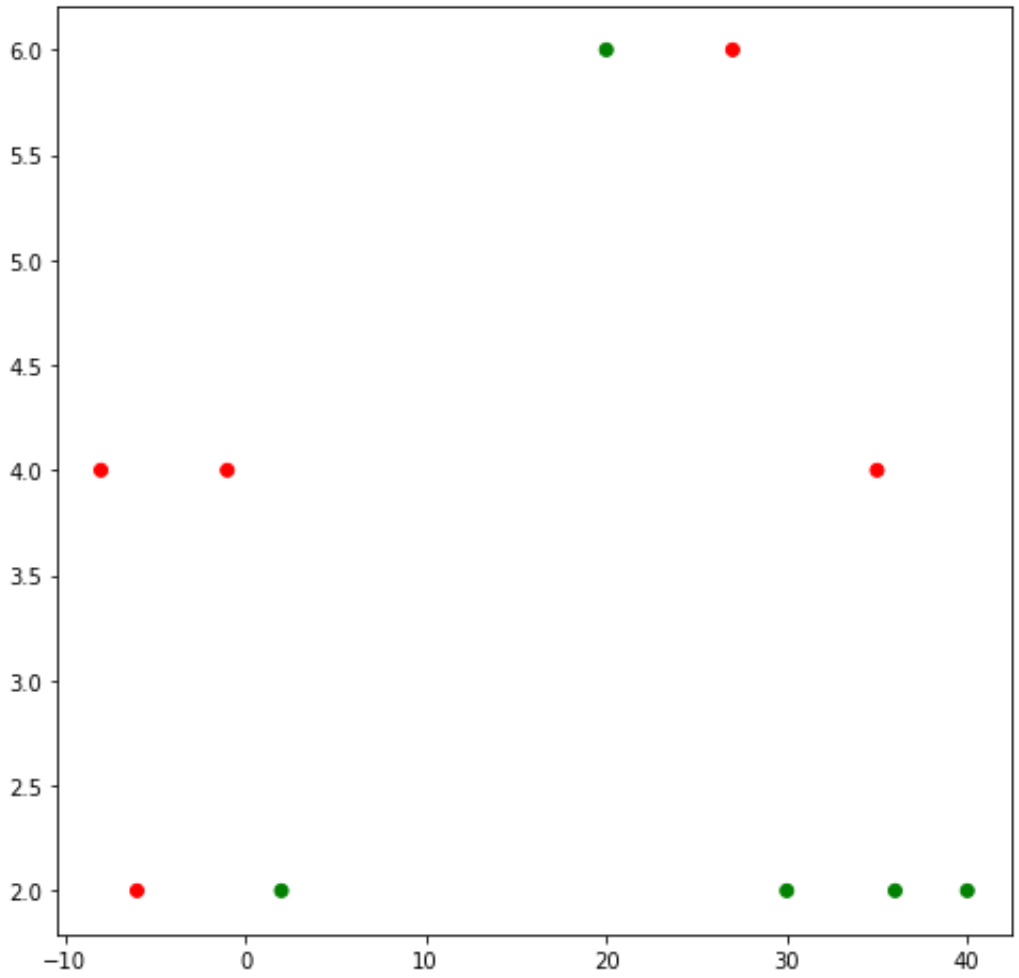
```
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

ids = [1,2,3,4,5,6,7,8,9,10]
x = [27,-6,2,36,-8,40,35,30,20,-1]
y = [6,2,2,2,4,2,4,2,6,4]
label = [-1,-1,1,1,-1,1,-1,1,1,-1]
colors = ['red','green']

fig = plt.figure(figsize=(8,8))
plt.scatter(x, y, c=label, cmap=matplotlib.colors.ListedColormap(colors))

# cb = plt.colorbar()
# loc = np.arange(0,max(label),max(label)/float(len(colors)))
# cb.set_ticks(loc)
# cb.set_ticklabels(colors)
```

```
<matplotlib.collections.PathCollection at 0x11b259128>
```




```
import math
```

```

ids = [1,2,3,4,5,6,7,8,9,10]
x = [27,-6,2,36,-8,40,35,30,20,-1]
y = [6,2,2,2,4,2,4,2,6,4]
label = [-1,-1,1,1,-1,1,-1,1,1,-1]
dataset = list(zip(ids,x,y,label))
```

```
dataset
```

```
[(1, 27, 6, -1),
 (2, -6, 2, -1),
 (3, 2, 2, 1),
 (4, 36, 2, 1),
 (5, -8, 4, -1),
 (6, 40, 2, 1),
 (7, 35, 4, -1),
 (8, 30, 2, 1),
 (9, 20, 6, 1),
 (10, -1, 4, -1)]
```

```
#Helper Method to find euclidian distance between two datapoints
def find_euclidian_distance(x,y):
    sum = 0
    sum += (x[1] - y[1]) ** 2
    sum += (x[2] - y[2]) ** 2
    return math.sqrt(sum)
```

```
find_euclidian_distance(dataset[0],dataset[1])
```

```
33.24154027718932
```

Q6(A)

3 Nearest datapoints for data point ID : 5

```
dist = []
for i in range(0,10):
    dist.append((i+1,find_euclidian_distance(dataset[4],dataset[i])))
# dist
sorted(dist, key=lambda x: x[1])
```

```
[(5, 0.0),
 (2, 2.8284271247461903),
 (10, 7.0),
 (3, 10.198039027185569),
 (9, 28.071337695236398),
 (1, 35.05709628591621),
 (8, 38.05259518088089),
 (7, 43.0),
 (4, 44.04543109109048),
 (6, 48.041648597857254)]
```

```
### ANSWER: Therefore 3 neareset negihbors for datapoint ID: 5 are {2,10,3}
```

```
dist = []
for i in range(0,10):
    dist.append((i+1,find_euclidian_distance(dataset[9],dataset[i])))

sorted(dist, key=lambda x: x[1])
```

```
[(10, 0.0),
 (3, 3.605551275463989),
 (2, 5.385164807134504),
 (5, 7.0),
 (9, 21.095023109728988),
 (1, 28.071337695236398),
 (8, 31.064449134018133),
 (7, 36.0),
 (4, 37.05401462729781),
 (6, 41.048751503547585)]
```

Answer: Therefore 3 neareset negihbors for datapoint ID: 10 are {3,2,5}

Q6 (B)

```
# Experiment
```

```
from sklearn.neighbors import NearestNeighbors
import numpy as np
list_tuples = [(27, 6),
 (-6, 2),
 (2, 2),
 (36, 2),
 (-8, 4),
 (40, 2),
 (35, 4),
 (30, 2),
 (20, 6),
 (-1, 4)]
labels = np.array([-1,-1,1,1,-1,1,-1,1,1,-1])
X = np.array(list_tuples)
# X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
# np.concatenate((X[:i],X[(i+1):]))
```

```
#Leave one out cross validation
total_errors = 0
for i in range(0,10):
    nbrs = NearestNeighbors(n_neighbors=1, algorithm='auto',
metric='euclidean').fit(np.concatenate((X[:i],X[(i+1):])))
    temp_labels = np.concatenate((labels[:i],labels[(i+1):]))
    distances, indices = nbrs.kneighbors([X[i]])

    for nearest_index in indices:
        # print(f'Nearest Index: {nearest_index[0]}')
        print(f'Testing Point: {dataset[i]} -> Nearest Point:
{dataset[nearest_index[0]]}')
        if temp_labels[nearest_index[0]] != dataset[i][3]:
            total_errors += 1

print(f'Leave-one-out-cross-validation error: {total_errors/10}')
```

```
Testing Point: (1, 27, 6, -1) -> Nearest Point: (7, 35, 4, -1)
Testing Point: (2, -6, 2, -1) -> Nearest Point: (4, 36, 2, 1)
Testing Point: (3, 2, 2, 1) -> Nearest Point: (9, 20, 6, 1)
Testing Point: (4, 36, 2, 1) -> Nearest Point: (6, 40, 2, 1)
Testing Point: (5, -8, 4, -1) -> Nearest Point: (2, -6, 2, -1)
Testing Point: (6, 40, 2, 1) -> Nearest Point: (4, 36, 2, 1)
Testing Point: (7, 35, 4, -1) -> Nearest Point: (4, 36, 2, 1)
Testing Point: (8, 30, 2, 1) -> Nearest Point: (1, 27, 6, -1)
Testing Point: (9, 20, 6, 1) -> Nearest Point: (1, 27, 6, -1)
Testing Point: (10, -1, 4, -1) -> Nearest Point: (3, 2, 2, 1)
Leave-one-out-cross-validation error: 0.7
```

Leave-one-out-cross-validation error: 0.7

Q6 (C)

```python
# Create Folds:
def createfolds(num_folds):
    folds = {i:{'train':[], 'test':[]} for i in range(1,num_folds+1)}
    for i in range(1,num_folds+1):
        for j in range(1,11):
            if j % 5 == i-1:
                folds[i]['test'].append(j)
            else:
                folds[i]['train'].append(j)
    return folds
```

```python
folds = createfolds(5)
folds

```

```
{1: {'test': [5, 10], 'train': [1, 2, 3, 4, 6, 7, 8, 9]},
 2: {'test': [1, 6], 'train': [2, 3, 4, 5, 7, 8, 9, 10]},
 3: {'test': [2, 7], 'train': [1, 3, 4, 5, 6, 8, 9, 10]},
 4: {'test': [3, 8], 'train': [1, 2, 4, 5, 6, 7, 9, 10]},
 5: {'test': [4, 9], 'train': [1, 2, 3, 5, 6, 7, 8, 10]}}
```

```python
total_errors = 0
```

```
for i in range(1,6):
    test_data = []
    test_labels = []
    train_data = []
    train_labels = []

    for index in folds[i]['train']:
        # print(f'Index: {index}')
        train_data.append(list_tuples[index-1])
        train_labels.append(labels[index-1])

    train_data_nparray = np.array(train_data)
    print(f'Train Data: {train_data_nparray}')
    # print(f'{train_labels}')

    for index in folds[i]['test']:
        # print(f'Index: {index}')
        test_data.append(list_tuples[index-1])
        test_labels.append(labels[index-1])

    test_data_nparray = np.array(test_data)
    # print(f'Test Data: {test_data_nparray}')

    nbrs = NearestNeighbors(n_neighbors=3, algorithm='auto',
metric='euclidean').fit(train_data_nparray)

    distances, indices = nbrs.kneighbors(test_data_nparray)

    # print(f'indices: {indices}\n')

    k = 0
    for nearest_indices in indices:
        label_total = 0
        for near_indice in nearest_indices:
            # print(f'near_indice: {near_indice}')
            label_total += train_labels[near_indice]

        if label_total > 0:
            predicted_label = 1
        if label_total < 0:
            predicted_label = -1

        # print(f'predicted_label: {predicted_label}')
        if predicted_label != test_labels[k]:
            total_errors += 1

```

```
        k += 1

print(f'5-fold-cross-validation error: {total_errors/10}')
```

```
Train Data: [[27  6]
 [-6  2]
 [ 2  2]
 [36  2]
 [40  2]
 [35  4]
 [30  2]
 [20  6]]
indices: [[1 2 7]
 [2 1 7]]

near_indice: 1
near_indice: 2
near_indice: 7
predicted_label: 1
near_indice: 2
near_indice: 1
near_indice: 7
predicted_label: 1
Train Data: [[-6  2]
 [ 2  2]
 [36  2]
 [-8  4]
 [35  4]
 [30  2]
 [20  6]
 [-1  4]]
indices: [[5 6 4]
 [2 4 5]]

near_indice: 5
near_indice: 6
near_indice: 4
predicted_label: 1
near_indice: 2
near_indice: 4
near_indice: 5
predicted_label: 1
Train Data: [[27  6]
 [ 2  2]
 [36  2]
```

```
 [-8  4]
 [40  2]
 [30  2]
 [20  6]
 [-1  4]]
indices: [[3 7 1]
 [2 5 4]]

near_indice: 3
near_indice: 7
near_indice: 1
predicted_label: -1
near_indice: 2
near_indice: 5
near_indice: 4
predicted_label: 1
Train Data: [[27  6]
 [-6  2]
 [36  2]
 [-8  4]
 [40  2]
 [35  4]
 [20  6]
 [-1  4]]
indices: [[7 1 3]
 [0 5 2]]

near_indice: 7
near_indice: 1
near_indice: 3
predicted_label: -1
near_indice: 0
near_indice: 5
near_indice: 2
predicted_label: -1
Train Data: [[27  6]
 [-6  2]
 [ 2  2]
 [-8  4]
 [40  2]
 [35  4]
 [30  2]
 [-1  4]]
indices: [[5 4 6]
 [0 6 5]]

```

```
near_indice: 5
near_indice: 4
near_indice: 6
predicted_label: 1
near_indice: 0
near_indice: 6
near_indice: 5
predicted_label: -1
5-fold-cross-validation error: 0.7
```

5-fold cross-validation error: 0.7

Q6 (D)

Based on the results of (B) and (C) we can not determine

1. Data points are very less.
2. For 5-fold cross-validation: testing set is size of 2 and training set size is of 8. Which is almost same as for leave-one-out (where ratio for testing is 1 and training 9).
3. So, We can not determine base