1) **Take Data2 and split it into a Training Set of 210 randomly selected data points. Use the remaining 100 data points as the Testing Set. Train a linear support vector machine (kernel_function = linear) to build a classifier model. Use this model to predict the classes for the data in the Testing Set.**

```
import numpy as np
from sklearn.utils import shuffle
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.metrics import precision_recall_fscore_support
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
import collections
import matplotlib.pyplot as plt
```

```
data1=pd.read_csv('G:\Masters  materials\Spring\IDA\Assignment2\Biomechanical_Data_
column_2C_weka.csv')
list(data1.keys()) //lists column name of datasets.
```

- **OUTPUT**

```
['pelvic_incidence',
'pelvic_tilt numeric',
'lumbar_lordosis_angle',
'sacral_slope',
'pelvic_radius',
'degree_spondylolisthesis',
'class']
```

- **Split it into a Training Set of 210 randomly selected data points**

```
data1=shuffle(data1);
X = data1.loc[:,'pelvic_incidence':'degree_spondylolisthesis'];
y= data1.loc[:,'class'];
X_train,X_test,y_train,y_test = X[:210] , X[210:], y[:210] , y[210:];
```

- **Train a linear support vector machine (kernel_function = linear) to build a classifier model, Use this model to predict the classes for the data in the Testing Set.**

```
from sklearn.svm import SVC
clf = SVC(kernel='linear')
clf.fit(X_train,y_train)
```

```
predict=clf.predict(X_test)
```

In above code,
SVC() function is equivalent to svmtrain function of matlab.
fit() function is used to train the classifier with given X_train,y_train train dataset.
predict() function predicts the output given a test dataset

1a) List all the support vectors of the data set. Examine the number of support vectors for each class and their attribute values and comment on what can be inferred from these values.

- **List all the support vectors of the data set. Examine the number of support vectors for each class**

```
print(clf.support_vectors_)
print("support vector for Abnormal class="+str(clf.n_support_[0])+" support vector for Normal class="+str(clf.n_support_[1]))
print("Indices of support vectors are\n"+str(clf.support_))
```

support_vectors_ attribute displays the support vectors for a SVM classifier.
n_support attribute displays support vectors for each class.
support_ attribute displays the indices of the support vectors.

Output:

Support Vectors for linear svm:

```
[[ 68.83202098  22.21848205  50.09219357  46.61353893 105.9851355   -3.53031731]
 [ 41.72996308  12.25407408  30.12258646  29.475889   116.5857056   -1.24440249]
 [ 69.78100617  13.77746531  57.99999999  56.00354085 118.9306656   17.91456046]
 [ 44.31890674  12.53799164  36.098763    31.78091509 124.1158358    5.41582514]
 [ 41.35250407  16.57736351  30.70619135  24.77514057 113.2666746   -4.49795756]
 [ 70.25043628  10.34012252  76.37007032  59.91031376 119.2370072   32.66650243]
 [ 40.55735663  17.97778407  34.          22.57957256 121.0462458   -1.53738307]
 [ 48.10923638  14.93072472  35.56468278  33.17851166 124.0564518    7.94790486]
 [ 38.69791243  13.44474904  31.          25.25316339 123.1592507    1.42918576]
 [ 78.40125389  14.04225971  79.69426258  64.35899418 104.7312342  12.39285327]
 [ 45.54078988  13.06959759  30.29832059  32.47119229 117.9808303   -4.98712962]
 [ 42.02138603  -6.55494835  67.89999999  48.57633437 111.5857819   27.33867086]
 [ 53.85479842  19.23064334  32.77905978  34.62415508 121.6709148    5.3298432 ]
 [ 43.20318499  19.66314572  35.          23.54003927 124.8461088   -2.91907595]
 [ 65.53600255  24.15748726  45.77516991  41.3785153  136.4403015   16.37808564]
 [ 52.41938511  19.01156052  35.87265953  33.40782459 116.5597709    1.6947051 ]
 [ 46.85578065  15.35151393  38.          31.50426672 116.2509174    1.66270559]
 [ 31.27601184   3.14466948  32.56299592  28.13134236 129.0114183    3.62302007]
 [ 66.28539377  26.32784484  47.49999999  39.95754893 121.2196839   -0.79962447]
 [ 53.43292815  15.86433612  37.16593387  37.56859203 120.5675233    5.9885507 ]
 [ 63.83498162  20.36250706  54.55243367  43.47247456 112.3094915   -0.62252664]
 [ 43.34960621   7.46746896  28.06548279  35.88213725 112.7761866    5.75327746]
 [ 45.44374959   9.9060718   44.99999999  35.53767779 163.0710405   20.31531532]
 [ 32.09098679   6.98937808  35.99819848  25.10160871 132.264735    6.41342771]
 [ 46.44207842   8.39503589  29.0372302   38.04704253 115.4814047    2.0454758 ]
 [ 35.70345781  19.44325311  20.7          16.26020471 137.5406125   -0.26348965]
 [ 54.60031622  21.48897426  29.36021618  33.11134196 118.3433212   -1.47106726]
 [ 37.7319919    9.38629828  41.99999999  28.34569362 135.740926    13.68304672]
 [ 54.92085752  18.96842952  51.60145541  35.952428   125.8466462    2.00164247]
 [ 51.07983294  14.20993529  35.95122893  36.86989765 115.8037111    6.90508996]
 [ 63.79242525  21.34532339  65.99999999  42.44710185 119.5503909   12.38260373]
 [ 33.84164075   5.07399141  36.64123294  28.76764934 123.9452436   -0.19924909]
 [ 49.82813487  16.73643493  28.          33.09169994 121.4355585    1.91330704]
 [ 42.51727249  14.37567126  25.32356538  28.14160123 128.9056892    0.75702014]
 [ 53.68337998  13.44702168  41.58429713  40.23635831 113.9137026    2.73703529]
 [ 51.62467183  15.96934373  35.          35.6553281  129.385308     1.00922834]
 [ 67.80469442  16.55066167  43.25680184  51.25403274 119.6856451    4.86753994]
 [ 46.63786363  15.85371711  39.99999999  30.78414653 119.3776026    9.06458168]
 [ 64.26150724  14.49786554  43.90250363  49.76364169 115.3882683    5.95145437]
 [ 40.34929637  10.19474845  37.96774659  30.15454792 128.0099272    0.45890137]
 [ 48.3189305   17.45212105  47.99999999  30.86680945 128.9803079   -0.91094057]
 [ 57.1458515   16.48909145  42.84214764  40.65676005 113.8061775    5.0151857 ]
```

```
 [ 47.31964755   8.5736803   35.56025198  38.74596726 120.5769719    1.63066351]
 [ 33.04168754  -0.32467846  19.0710746   33.366366   120.3886112    9.35436493]
 [ 38.04655072   8.30166942  26.23683004  29.7448813  123.8034132    3.88577349]
 [ 69.3988184   18.89840693  75.96636144  50.50041147 103.5825398   -0.44366081]
 [ 89.01487529  26.07598143  69.02125897  62.93889386 111.4810746    6.0615084 ]
 [ 47.90356517  13.61668819  36.          34.28687698 117.4490622   -4.24539542]
 [ 42.51561014  16.54121618  41.99999999  25.97439396 120.631941     7.87673069]
 [ 56.10377352  13.10630665  62.63701952  42.99746687 116.2285032   31.17276727]
 [ 61.54059876  19.67695713  52.89222856  41.86364163 118.6862678    4.81503108]
 [ 38.50527283  16.96429691  35.11281407  21.54097592 127.6328747    7.98668323]
 [ 43.11795103  13.81574355  40.34738779  29.30220748 128.5177217    0.97092641]]


 support vector for Abnormal class=26  support vector for Normal class=27

 Indices of support vectors are
[  2   9  24  28  31  41  42  46  47  48  50  52  56  69  89  94  96 102
 109 151 160 181 187 193 194 204   8  15  29  32  58  64  67  76  78  84
  85 101 113 118 131 141 143 147 162 164 171 176 195 198 199 200 205]
```

- **Examine the number of support vectors for each class and their attribute values and comment on what can be inferred from these values.**

From above output it can be seen that number of support vectors for the abnormal class are 26 and support vector for Normal class= 27.

It can be inferred from above output that, Support vectors are the data points that lie on the margin of the SVM classifier. Compared to other data points, support vectors of both the classes at minimum distances i.e. these points decide to which class a new data instance belongs to.

- **Code for accuracy precision and recall**

```
accuracy = clf.score(X_test,y_test)
result = precision_recall_fscore_support(y_test, predict, pos_label='Normal', average='binary')
print(" Accuracy " +str(accuracy) + "\t\t Precision " +str(result[0])
    +"\n Recall " +str(result[1]))
```

- **Output**

```
Accuracy 0.86    Precision 0.8529411764705882
Recall 0.7631578947368421
```

1b) Run the training and testing five times, each time selecting a different randomly selected set of training instances. Create the confusion matrix for this classifier using average number of true positives, false positives etc.

CODE:

```
clf = SVC(kernel='linear')
Precision = []
Recall = []
FScore = []
Accuracy = []
tp=0
tn=0
fp=0
fn=0
from sklearn.metrics import confusion_matrix
for x in range(0, 5):
   data1=shuffle(data1);
   X = data1.loc[:,'pelvic_incidence':'degree_spondylolisthesis'];
   y= data1.loc[:,'class'];
   X_train,X_test,y_train,y_test = X[:210] , X[210:], y[:210] , y[210:];
   clf.fit(X_train,y_train)
   predict=clf.predict(X_test)
   accuracy = accuracy = clf.score(X_test,y_test)
   result = precision_recall_fscore_support(y_test, predict, pos_label='Normal', average='binary')
   Precision.append(result[0])
   Recall.append(result[1])
   Accuracy.append(accuracy)
   conf_mat = confusion_matrix(y_test, predict)
   print(conf_mat)
   tp= tp + conf_mat[0][0]
   tn= tn + conf_mat[1][1]
   fn= fn + conf_mat[0][1]
   fp= fp + conf_mat[1][0]
avg_conf=np.matrix([[tp/5, fn/5], [fp/5, tn/5]])
print("confusion matrix for this classifier using average number of true positives, false positives etc. =\n "+str(avg_conf))
```

In above code, precision_recall_fscore_support() function calculate precision, recall, f1 score and support given the predicted output, actual output and positive class.

OUTPUT:

```
confusion matrix for this classifier using average number of true
positives,   false positives etc. =
  [[62.  8.]
  [ 6. 24.]]
```

1c) Compute the accuracy, precision and recall values.

CODE:

```
avg_acc = np.mean(Accuracy)
avg_pre = np.mean(Precision)
avg_rec = np.mean(Recall)
print("Average Accuracy " +str(avg_acc) + "\t\t Average Precision " +str(avg_pre)
             +"\nAverage  Recall " +str(avg_rec))
```

OUTPUT:

```
Average Accuracy 0.8559999999999999   Average Precision 0.8001843317972351
Average  Recall 0.760881690904377
```

2) Repeat Q#1 above with the only difference that this time train a non-linear SVM using rbf (radial basis function) for kernerl function

- **Split it into a Training Set of 210 randomly selected data points**
- **Train a linear support vector machine (kernel_function = linear) to build a classifier model, Use this mod el to predict the classes for the data in the Testing Set.**

```
data1=shuffle(data1);
X = data1.loc[:,'pelvic_incidence':'degree_spondylolisthesis'];
y= data1.loc[:,'class'];
X_train,X_test,y_train,y_test = X[:210] , X[210:], y[:210] , y[210:];
from sklearn.svm import SVC
clf = SVC(C = 50,kernel = 'rbf',gamma = 0.0001)
clf.fit(X_train,y_train)
predict=clf.predict(X_test)
```

In above code,
SVC() function is equivalent to svmtrain function of matlab.
fit() function is used to train the classifier with given X_train,y_train train dataset.
predict() function predicts the output given a test dataset.

2a) List all the support vectors of the data set. Examine the number of support vectors for each class and their attribute values and comment on what can be inferred from these values.

- **List all the support vectors of the data set. Examine the number of support vectors for each class**

CODE:

```
print(clf.support_vectors_)
print("support vector for Abnormal class="+str(clf.n_support_[0])+" support vector for Normal class="+str(clf.n_support_[1]))
print("Indices of support vectors are\n"+str(clf.support_))
```

OUTPUT:

SUPPORT VECTORS ARE:

```
[[ 66.28539377  26.32784484  47.49999999  39.95754893 121.2196839   -0.79962447]
 [ 72.64385013  18.92911726  67.99999999  53.71473287 116.9634162   25.38424676]
 [ 76.1472121   21.93618556  82.96150249  54.21102654 123.9320096   10.43197194]
 [ 72.07627839  18.94617604  50.99999999  53.13010236 114.2130126    1.01004051]
 [ 73.63596236   9.71131795  62.99999999  63.92464442  98.72792982  26.97578722]
 [ 45.36675362  10.75561143  29.03834896  34.61114218 117.2700675  -10.67587083]
 [ 43.34960621   7.46746896  28.06548279  35.88213725 112.7761866    5.75327746]
 [ 54.91944259  21.06233245  42.19999999  33.85711014 125.2127163    2.43256144]
 [ 43.79019026  13.5337531   42.69081398  30.25643716 125.0028927   13.28901817]
 [ 47.74467877  12.08935067  38.99999999  35.6553281  117.5120039   21.68240136]
 [ 41.72996308  12.25407408  30.12258646  29.475889   116.5857056   -1.24440249]
 [ 32.09098679   6.98937808  35.99819848  25.10160871 132.264735    6.41342771]
 [ 44.93667457  17.44383762  27.78057555  27.49283695 117.9803245    5.56961959]
 [ 48.10923638  14.93072472  35.56468278  33.17851166 124.0564518    7.94790486]
 [ 43.20318499  19.66314572  35.          23.54003927 124.8461088   -2.91907595]
 [ 41.17167989  17.32120599  33.46940277  23.85047391 116.3778894   -9.56924986]
 [ 65.53600255  24.15748726  45.77516991  41.3785153  136.4403015   16.37808564]
 [ 66.87921138  24.89199889  49.27859673  41.9872125  113.4770183   -2.00589175]
 [ 35.70345781  19.44325311  20.7         16.26020471 137.5406125   -0.26348965]
 [ 38.66325708  12.98644139  39.99999999  25.67681568 124.914118    2.70300805]
```

```
[ 46.85578065  15.35151393  38.          31.50426672 116.2509174    1.66270559]
[ 41.35250407  16.57736351  30.70619135  24.77514057 113.2666746   -4.49795756]
[ 42.02138603  -6.55494835  67.89999999  48.57633437 111.5857819   27.33867086]
[ 45.54078988  13.06959759  30.29832059  32.47119229 117.9808303   -4.98712962]
[ 49.70660953  13.04097405  31.33450009  36.66563548 108.6482654   -7.82598575]
[ 53.43292815  15.86433612  37.16593387  37.56859203 120.5675233    5.9885507 ]
[ 53.85479842  19.23064334  32.77905978  34.62415508 121.6709148    5.3298432 ]
[ 63.83498162  20.36250706  54.55243367  43.47247456 112.3094915   -0.62252664]
[ 65.00796426  27.60260762  50.94751899  37.40535663 116.5811088    7.01597788]
[ 41.76773173  17.89940172  20.0308863   23.86833001 118.3633889    2.06296255]
[ 31.27601184   3.14466948  32.56299592  28.13134236 129.0114183    3.62302007]
[ 55.08076562  -3.75992987  55.99999999  58.84069549 109.9153669   31.77358318]
[ 45.44374959   9.9060718   44.99999999  35.53767779 163.0710405   20.31531532]
[ 52.41938511  19.01156052  35.87265953  33.40782459 116.5597709    1.6947051 ]
[ 55.28585178  20.44011836  34.          34.84573342 115.8770174    3.55837236]
[ 43.1919153    9.9766638   28.93814927  33.21525149 123.4674001    1.74101758]
[ 44.48927476  21.78643263  31.47415392  22.70284212 113.7784936   -0.28412937]
[ 33.84164075   5.07399141  36.64123294  28.76764934 123.9452436   -0.19924909]
[ 48.90290434   5.58758866  55.49999999  43.31531568 137.1082886   19.85475919]
[ 47.90356517  13.61668819  36.          34.28687698 117.4490622   -4.24539542]
[ 51.07983294  14.20993529  35.95122893  36.86989765 115.8037111    6.90508996]
[ 39.65690201  16.20883944  36.67485694  23.44806258 131.922009    -4.96897988]
[ 67.53818154  14.65504222  58.00142908  52.88313932 123.6322597   25.9702063 ]
[ 41.6469159    8.8355491   36.03197484  32.8113668  116.5551679   -6.05453796]
[ 63.92947003  19.97109671  40.17704963  43.95837332 113.0659387  -11.05817866]
[ 66.50717865  20.89767207  31.72747138  45.60950658 128.9029049    1.51720336]
[ 48.80190855  18.01776202  51.99999999  30.78414653 139.1504066   10.44286169]
[ 61.44659663  22.6949683   46.17034732  38.75162833 125.6707246   -2.70787952]
[ 34.64992241   7.51478278  42.99999999  27.13513962 123.9877408   -4.0829376 ]
[ 65.61180231  23.13791922  62.58217893  42.47388309 124.1280012   -4.08329841]
[ 53.91105429  12.93931796  38.99999999  40.97173633 118.1930354    5.07435318]
[ 57.1458515   16.48909145  42.84214764  40.65676005 113.8061775    5.0151857 ]
[ 53.68337998  13.44702168  41.58429713  40.23635831 113.9137026    2.73703529]
[ 33.04168754  -0.32467846  19.0710746   33.366366   120.3886112    9.35436493]
[ 50.75329025  20.23505957  37.          30.51823068 122.343516     2.28848775]
[ 89.83467631  22.63921678  90.56346144  67.19545953 100.5011917    3.04097326]
[ 54.60031622  21.48897426  29.36021618  33.11134196 118.3433212   -1.47106726]
[ 53.93674778  20.72149628  29.22053381  33.21525149 114.365845    -0.42101039]
[ 42.51561014  16.54121618  41.99999999  25.97439396 120.631941     7.87673069]
[ 36.42248549  13.87942449  20.24256187  22.543061   126.0768612    0.17971708]
[ 45.57548229  18.75913544  33.77414297  26.81634684 116.7970069    3.13190992]
[ 50.08615264  13.43004422  34.45754051  36.65610842 119.1346221    3.08948446]
[ 49.82813487  16.73643493  28.          33.09169994 121.4355585    1.91330704]
[ 64.31186727  26.32836901  50.95896417  37.98349826 106.1777511    3.11822129]
[ 89.01487529  26.07598143  69.02125897  62.93889386 111.4810746    6.0615084 ]
[ 37.14014978  16.48123972  24.          20.65891006 125.0143609    7.3664254 ]
[ 63.79242525  21.34532339  65.99999999  42.44710185 119.5503909   12.38260373]
[ 46.63786363  15.85371711  39.99999999  30.78414653 119.3776026    9.06458168]
[ 43.11795103  13.81574355  40.34738779  29.30220748 128.5177217    0.97092641]
[ 42.51727249  14.37567126  25.32356538  28.14160123 128.9056892    0.75702014]]
```

```
Support vector for Abnormal class=35, support vector for Normal class=35
Indices of support vectors are
[  4,  11,  21,  37,  39,  44,  50,  55,  71,  78,  80,  81,  86,
  87,  92, 104, 106, 116, 119, 120, 126, 128, 133, 134, 143, 144,
 149, 151, 168, 175, 194, 196, 200, 206, 207,   2,   7,  12,  43,
  49,  51,  56,  57,  59,  66,  72,  82,  95,  99, 100, 101, 107,
 108, 109, 110, 121, 123, 125, 129, 137, 148, 150, 157, 183, 185,
 187, 199, 205, 208, 209]
```

- **Examine the number of support vectors for each class and their attribute values and comment on what can be inferred from these values.**

From above output it can be seen that number of support vectors for the abnormal class are 35 and support vector for Normal class= 35.
It can be inferred from above output that, Support vectors are the data points that lie on the margin of the SVM classifier. Compared to other data points, support vectors of both the classes at minimum distances i.e. these points decide to which class a new data instance belongs to.

- **`Code for accuracy precision and recall`**

```
accuracy = clf.score(X_test,y_test)
result = precision_recall_fscore_support(y_test, predict, pos_label='Normal', average='binary')
print(" Accuracy " +str(accuracy) + "\t\t Precision " +str(result[0])
  +"\n Recall " +str(result[1]))
```

- **Output**

```
Accuracy 0.87          Precision 0.7714285714285715
Recall 0.84375
```

2b) Run the training and testing five times, each time selecting a different randomly selected set of training instances. Create the confusion matrix for this classifier using average number of true positives, false positives etc.

```
Precision = []
Recall = []
FScore = []
Accuracy = []
tp=0
tn=0
fp=0
fn=0
for x in range(0, 5):
    data1=shuffle(data1);
     X = data1.loc[:,'pelvic_incidence':'degree_spondylolisthesis'];
     y= data1.loc[:,'class'];
     X_train,X_test,y_train,y_test = X[:210] , X[210:], y[:210] , y[210:];
     clf = SVC(C = 50,kernel = 'rbf',gamma = 0.0001)
      clf.fit(X_train,y_train)
      predict=clf.predict(X_test)
     accuracy = clf.score(X_test,y_test)
     result = precision_recall_fscore_support(y_test, predict, pos_label='Normal', average='binary')
     Precision.append(result[0])
      Recall.append(result[1])
     Accuracy.append(accuracy)
     conf_mat = confusion_matrix(y_test, predict)
     print(conf_mat)
     tp= tp + conf_mat[0][0]
     tn= tn + conf_mat[1][1]
     fn= fn + conf_mat[0][1]
      fp= fp + conf_mat[1][0]
avg_conf=np.matrix([[tp/5, fn/5], [fp/5, tn/5]])
print("confusion matrix for this classifier using average number of true positives, false positives etc. =\n
"+str(avg_conf))
```

In above code, **precision_recall_fscore_support()** function calculate precision, recall, f1 score and support given the predicted output, actual output and positive class.

OUTPUT:

```
confusion matrix for this classifier using average number of true posit
ives, false positives etc. =
  [[62.  10.4]
   [ 5.2 22.4]]
```

2c) Compute the accuracy, precision and recall values.

CODE:

```
avg_acc = np.mean(Accuracy)
avg_pre = np.mean(Precision)
avg_rec = np.mean(Recall)
print("Average Accuracy " +str(avg_acc) + "\t\t Average Precision " +str(avg_pre)
      +"\nAverage  Recall " +str(avg_rec))
```

OUTPUT:

```
Average Accuracy 0.8619999999999999 Average Precision 0.783041283041231
Average  Recall 0.8324346405228757
```

3) Compare the performance values obtained in 1c and 2c above with the ones you received for the same data in Problem 1(c) of Homework#1. Comment on the differences and you observe and their possible causes/consequences.

FOR 1C(Linear kernel SVM),

```
Average Accuracy 0.8559999999999    Average Precision 0.800184331797235
Average  Recall 0.760881690904377
```

FOR 2C(RBF kernel SVM),

```
Average Accuracy 0.861999999999999   Average Precision 0.78304128304123
Average  Recall 0.8324346405228757
```

FOR 1(c) of Homework#1(Decision tree),

```
average of accuracy = 0.810000     average of precision = 0.665102
average of recall = 0.548387
```

As can be seen from above values, the accuracy is higher for SVM with rbf kernel than linear kernel SVM and decision tree. Also, the precision and recall values are higher for SVM classifier than decision tree classifier.

Decision trees can't split further if the attribute values are not separable, whereas SVM finds support vector using complex optimization method.
SVM uses the kernel trick to turn a linearly non-separable problem into a linearly separable one, whereas decision tree split input space into hyper-rectangles according to target.
If data is not separable with hyper-rectangles then decision tree won't perform efficiently, whereas complex kernel function can be used to transform data so that it can be classified with SVM classifier.