PROJECT REPORT

PROJECT: EFFECTIVE AUTOMATED PREDICTION OF VERTEBRAL COLUMN PATHOLOGIES.

MENTOR: MR. ASHWANI THAPLIYAL

Submitted by:

Ankush Kamboj

ACKNOWLEDGEMENT

I have taken efforts in this project. However, it would not have been possible without the kind support and help of my mentor Mr. Ashwini Thapliyal. I would like to extend my sincere thanks to him for guiding me through the field of Medical Diagnosis, for constant supervision, motivation as well as for providing necessary information regarding the project.

I would also like to express my gratitude towards the members of Oil and Natural Gas Corporation for their kind co-operation and encouragement which helped me in completion of this project.

CERTIFICATE

This is to certify that this project titled “Effective Automated Prediction of Vertebral Column Pathologies” submitted by Ankush Kamboj (DTU/2K15/CO/032) in partial fulfilment for the requirements for the award of Bachelor of Technology Degree in Computer Engineering (COE) at Delhi Technological University is an authentic work carried out by the student under my supervision and guidance. To the best of my knowledge, the matter embodied in the report has not been submitted to any other university or institute for the award of any degree or diploma.

Mr. Ashwani Thapliyal
Oil and Natural Gas Corporation

ABSTRACT

Medical science is characterized by the correct diagnosis of a disease and its accurate classification to avoid complexities at treatment/medication stage. This is often accomplished by a physician based on experience without much signal processing aids. It is envisioned that a sophisticated and intelligent medical diagnostic/classification system may be helpful in making right decisions

Classifying data is one of the most common task in Machine learning. Machine learning provides one of the main features for extracting knowledge from large databases from enterprises operational databases. Machine Learning in Medical Health Care is an emerging field of very high importance for providing prognosis and a deeper understanding of medical data. Most machine learning methods depend on a set of features that define the behavior of the learning algorithm and directly or indirectly influence the performance as well as the complexity of resulting models.

We investigate different algorithms such as ***KNearest Neighbors, Random forest, Logistic Regression*** *along with* ***back-propagation Neural Network (BPNN*** *for the diagnosis of spondylolisthesis and Disk-Hernia. We then use this model to deploy our application to automate priority-based follow-up appointment scheduling and also forward referral application to make appointments with specialty hospitals.*

The system has been implemented in Python platform and trained using proprietary dataset and benchmark dataset from UCI machine learning repository.

CONTENTS

INTRODUCTION

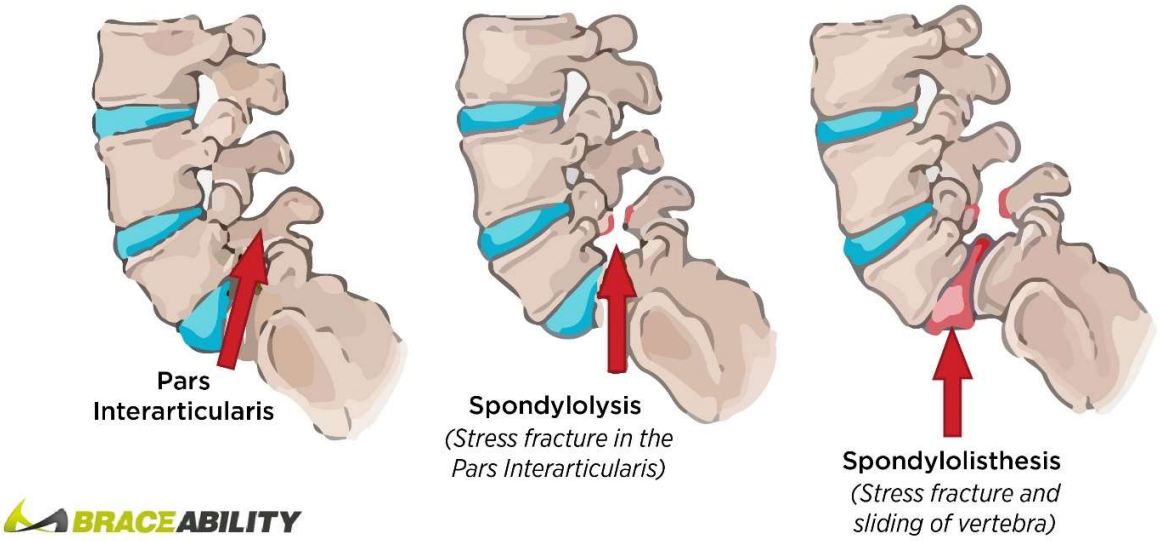
Vertebral Column

Various physical diseases come with age, of which many are inherited and inevitable. The vertebral column plays an important role in body motion, which supports the head and trunk and protects the nerves governing body activities and sensations. In addition, it is the main transfer passage of the nerves

Vertebral column is an integral part of human body. It is a structure which consists of usually thirty-three vertebrates, out of which twenty-four are articulating and nine are fused vertebras. It is found in the dorsal aspect of the torso and is separated by number of intervertebral discs. These thirty-three vertebras are divided in five different groups which include seven vertebrae in cervical curve, twelve vertebrae in thoracic curve, five vertebrae in lumbar curve and nine vertebrae in sacral curve. Intervertebral discs are interposed between the vertebral bodies, and serve not only as shock absorber for the column but also provide the normal mobility between the adjacent vertebrae. Each disc consists of a soft central portion of spongy material. The two vertebral disorders discussed in this paper are disc hernia and spondylolisthesis. Disc hernia is an intervertebral disc protrusion that is produced by the effect of flexion force acting upon the most mobile portions of the spine. A sudden strain with the spine in an unguarded position will rupture the tough annulus, allowing portions of the torn annulus and soft nucleus to escape into the spinal canal. Spondylolisthesis is a condition in which a lower lumbar vertebra, usually the fifth slips forward through the plain of the intervertebral disc below it and so carries with it the whole of the upper portion of the spine.

Experiment Setup

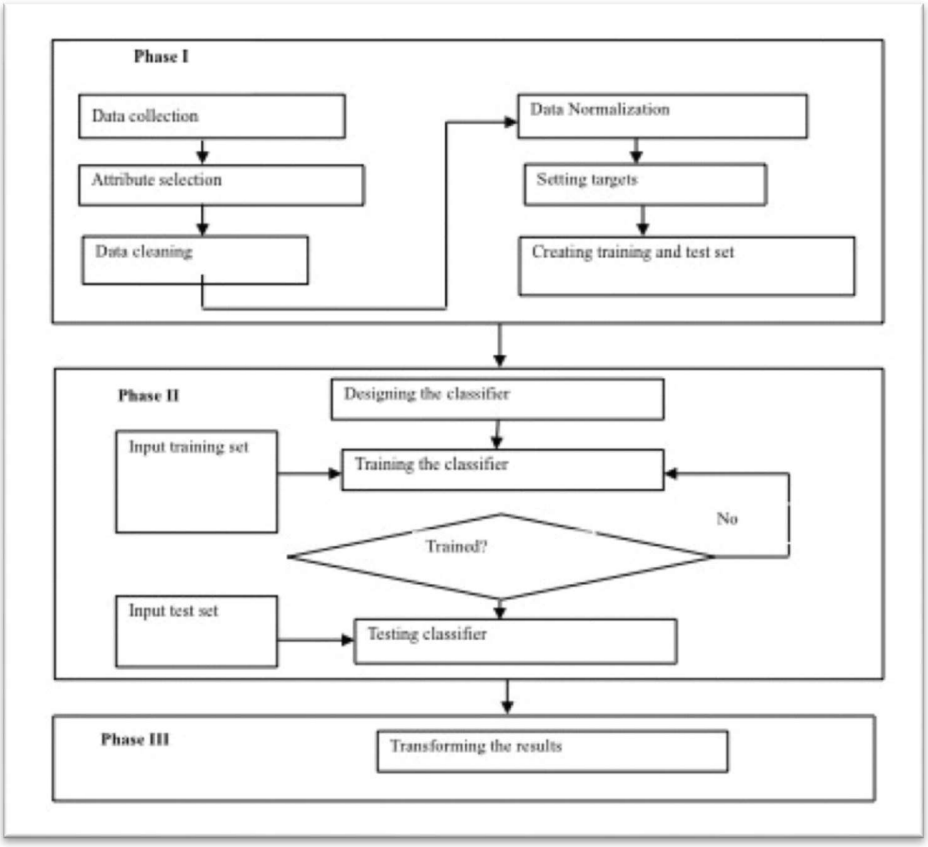
The problem area of vertebral column in humans is spinal cord. Here we have taken data set having values for six biomechanical features used to classify orthopedic patients into 3 classes (normal, disk hernia or spondylolisthesis) or 2 classes (normal or abnormal). The first task involves in categorizing patients as fit in to one out of three categories: Normal (100 patients), Disk Hernia (60 patients) or Spondylolisthesis (150 patients). For the second task, the categories Disk Hernia and Spondylolisthesis were combined into a single category labelled as 'abnormal'. Thus, the second task consists in classifying patients as fitting to one out of two categories: Normal (100 patients) or Abnormal (210 patients). Each patient is characterized in the data set by six biomechanical attributes resultant from the shape and alignment of the pelvis and lumbar spine (in this order): pelvic occurrence, pelvic tilt, lumbar lordosis position, sacral slope, pelvic radius and mark of Spondylolisthesis. The following settlement is used for the class labels: DH (Disk Hernia), Spondylolisthesis (SL), Normal (NO) and Abnormal (AB). A herniated disk and Spondylolisthesis are two possibly painful circumstances that can affect the steadiness and function of the spinal column. While herniation affects the discs between the spinal bones (vertebrae), Spondylolisthesis affect the bones themselves.



PROPOSED METHOD FOR DIAGNOSIS OF VERTEBRAL COLUMN DISORDERS

A: Methodology:

We have used four classifiers to diagnose vertebral column disorders, which includes a kNN classifier, a logistic regression model, a backpropagation neural network and random forest classifier. The disorders are classified as disc hernia spondylolisthesis and normal patients. The methodology used by all the classifiers is divided in to three phases. The detail of these phases is discussed below. Figure 1 presents the block diagram of these phases.



Phase 1: Pre-Processing

In this phase the data is first collected. During the collection the attributes are selected after discussing them with the doctor. We have incorporated all the attributes as they are significant and have a huge influence on the class labels. Next the data is analyzed and cleaned. In the cleaning process the missing and noisy values are handled. Afterwards the data is normalized. Subsequently, each sample in the data set is allotted its targets class. The entire data set is divided into two sets, the training and the test sets. In case of the validation set, the training set is further divided in to two sub groups, the training group and the validation group. After this the dataset is ready to be provided to the classifiers.

Phase 2: Design Classifier

After the pre-processing is complete, the data is presented to the classifiers. All the classifiers belong to the supervised learning category. For the supervised classifier the training set includes both the samples and their targets. These trained classifiers learn the pattern from the samples provided to them and diagnose the correct disorder. Each classifier is trained two times with two different datasets; i.e. the 60% ratio and the 10-fold cross validation. The performance of all four classifiers is evaluated. If the classifier is not trained the training process is repeated with different structure and functions.

Phase 3: Post-Processing

In this phase the results of the trained classifiers are converted into a form that is easily understandable. The results show whether a person is normal or has a vertebral column disorder and the type of disorder either disc hernia or spondylolisthesis.

B: Material

The data set used for classifying vertebral column disorders is taken from the UCI machine learning database. The dataset contains 310 samples out of which 60 samples are of disc hernia, 150 samples of spondylolisthesis and 100 normal samples. Each sample has six attributes. All the attributes are numerical. The second and sixth attribute
contains some samples having negative values as well. All these attributes are used to classify between disc hernia, spondylolisthesis and normal patients. The attribute along with their minimum and maximum values are presented in table I and **figure**.

Attributes	**Values**	
	Minimum	*Maximum*
Pelvic Incidence	26.1479	129.834
Pelvic Tilt	-3.7599	49.4319
Lumbar Lordosis Angle	14.00	125.7424
Sacral Slope	13.3669	121.4296
Pelvic Radius	70.0826	163.071
Degree Spondylolisthesis	-11.0582	418.5431

CLASSIFIERS STRUCTURE

A. Feedforward Backpropagation Neural Network

The NN we used has an architecture 6-5-4-3. The input layer has 6 nodes because each sample has six features and so there are six values in each input record. The hidden and the output layers have 5, 4 and 3 neurons respectively. These neurons perform all the computation. The activation or transfer function we used for both the hidden and output layer neurons is hyperbolic tangent sigmoid function. It is relatively faster than the other activation functions.

$$a = tansig(n) = \frac{2}{1 + e^{(-2*n)}} - 1$$

Where **n** is a value of the hidden and output layer neurons given to the hyperbolic tangent sigmoid activation function. The error is calculated using the mean squared error performance function:

$$Eav = 1/N \sum_{n=1}^{N} E(n)$$

Where $\mathbf{E_{av}}$ is the average error of the network, **N** is the total number of training samples and **E(n)** is the network error. The learning algorithm used is the gradient descent with momentum weight and bias learning function. It
changes the weights in such a way that it reduces the error between the output and the targets. The learning function
is:

$$dW = mc * dWprev + (1 - mc) * lr * gW$$

Where **dW** is the weight change for a particular neuron from its input and error, $\mathbf{dW_{prev}}$ is previous weight change, **lr** is the learning rate in our case it is 0.01**, mc** is momentum constant which is 0.9 in our situation and **W** is either the weight or bias. The weights are changed based on the gradient decent method.
The training algorithm we used is a variant of backpropagation algorithm which is levenberg-marquardt backpropagation. It is based on the error correction rule. The backpropagation algorithm uses the jacobian matrix which contains first derivative of network error with respect to the weights and bias. The Levenberg–Marquardt algorithm is a combination of steepest descent and the gauss–newton algorithm. Its updating rule is:

$$w_{k+1} = w_k - [J^T J + \mu I]^{-1} J^T e$$

Where **w** is the weight, **J** is the jacobian matrix, **T** means transpose, μ is the combination coefficient which is always
positive, I is the identity matrix, and e is the error. The $\mathbf{J^t}\boldsymbol{e}$ is the gradient. The combination coefficient allows the Levenberg–Marquardt algorithm to switch between the steepest decent and netwon algorithms during the training process. The training set provided to FFNN is divided in two parts the training set and the validation set. The validation set monitors the training process and stops training when the error starts to increase instead of decrease. Both the data distribution (80% ratio and 10-fold cross validation) used the same FFNN architecture, training, learning, activation and performance functions.
The weights are converged after 100 epochs in case of 80% ratio data distribution approach whereas in 10-fold approach weights are converged after 50 epochs.

B: Logistic Regression

Logistic regression, despite its name, is a linear model for classification rather than regression. Logistic regression is also known in the literature as logit regression, maximum -entropy classification (MaxEnt) or the log-linear classifier. In this model, the probabilities describing the possible outcomes of a single trial are modeled using a logistic function.

The implementation of logistic regression in scikit-learn can be accessed from class Logistic-Regression. This implementation can fit a multiclass (one-vs-rest) logistic regression with optional L2 or L1 regularization.

As an optimization problem, binary class L2 penalized logistic regression minimizes the following cost function:

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Similarly, L1 regularized logistic regression solves the following optimization problem:

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

Definition of the logistic function:

An explanation of logistic regression can begin with an explanation of the standard logistic function. The logistic function is useful because it can take an input with any value from negative to positive infinity, whereas the output always takes values between zero and one and hence is interpretable as a probability. The logistic function is defined as follows:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

C: Fuzzy k-Nearest Neighbours

In pattern recognition, the kNearest Neighbors algorithm (or kNN for short) is a nonparametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. kNN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The kNN algorithm is among the simplest of all machine learning algorithms. Both for classification and regression, it can be useful to assign weight to the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbor a weight of 1/d, where d is the distance to the neighbor. The neighbors are taken from a set of objects for which the class (for kNN classification) or the object property value (for kNN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required. A shortcoming of the kNN algorithm is that it is sensitive to the local structure of the data.

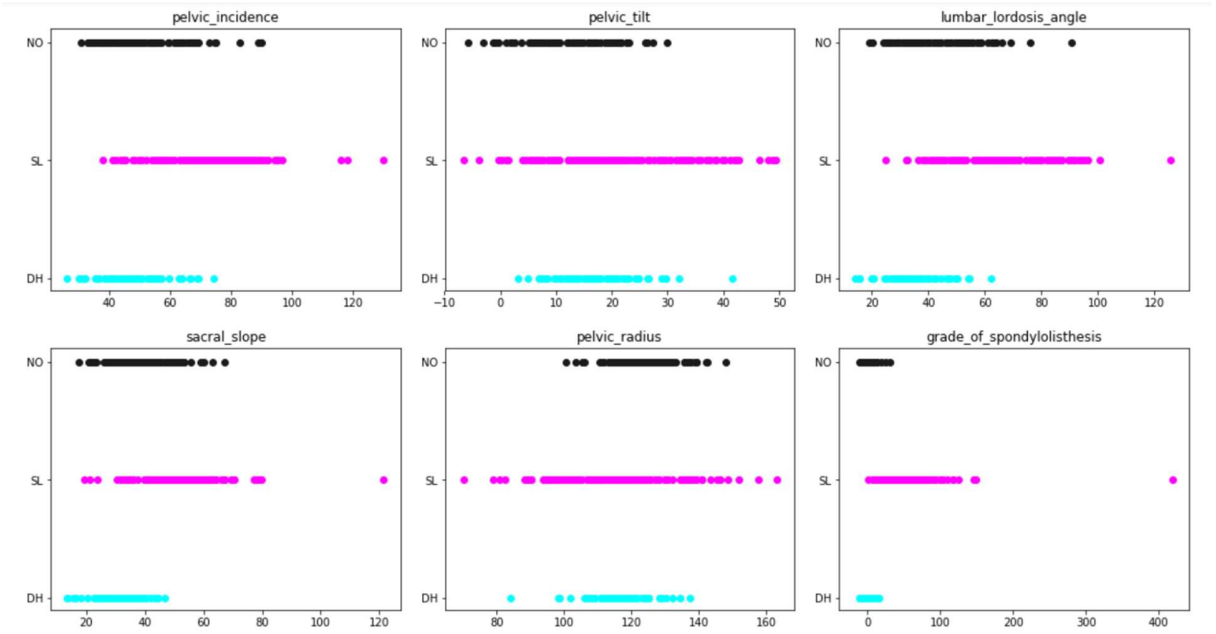
The basis of the algorithm is to assign membership as a function of the vector's distance from its k-nearest neighbors and those neighbors' memberships in the possible classes. The fuzzy algorithm is similar to the crisp version in the sense that it must also search the labeled sample set for the k-nearest neighbors. Beyond obtaining these k samples, the procedures differ considerably. While the fuzzy Kk-nearest neighbor procedure is also a classification algorithm the form of its results differ from the crisp version. The fuzzy k-nearest neighbor algorithm assigns class membership to a sample vector rather than assigning the vector to a particular class. The advantage is that no arbitrary assignments are made by the algorithm. In addition, the vector's membership values should provide a level of assurance to accompany the resultant classification.

D: Random Forests

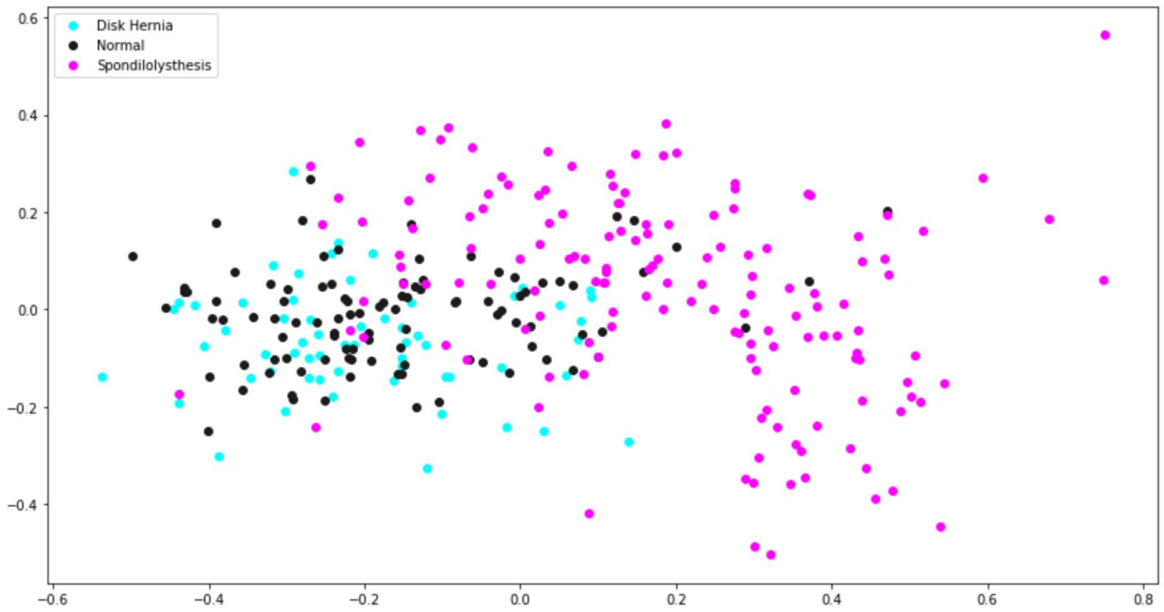
Random forests are built by combining the predictions of several trees, each of which is trained in isolation. Unlike in boosting (Schapire & Freund, 2012) where the base models are trained and combined using a sophisticated weighting scheme, typically the trees are trained independently and the predictions of the trees are combined through averaging. There are three main choices to be made when constructing a random tree. These are the method for splitting the leafs, the type of predictor to use in each leaf, and the method for injecting randomness into the trees. Specifying a method for splitting leafs requires selecting the shapes of candidate splits as well as a method for evaluating the quality of each candidate. Typical choices here are to use axis aligned splits, where data are routed to subtrees depending on whether or not they exceed a threshold value in a chosen dimension; or linear splits, where a linear combination of features are thresholded to make a decision. The threshold value in either case can be chosen randomly or by optimizing a function of the data in the leafs. In order to split a leaf, a collection of candidate splits are generated and a criterion is evaluated to choose between them. A simple strategy is to choose among the candidates uniformly at random, as in the models analyzed in Biau et al. (2008). A more common approach is to choose the candidate split which optimizes a purity function over the leafs that would be created. The most common choice for predictors in each leaf is to use the average response over the training points which fall in that leaf. Criminisi et al. (2011) explore the use of several different leaf predictors for regression and other tasks, but these generalizations are beyond the scope of this paper. We consider only simple averaging predictors here. Injecting randomness into the tree construction can happen in many ways. The choice of which dimensions to use as split candidates at each leaf can be randomized, as well as the choice of coefficients for random combinations of features. In either case, thresholds can be chosen either randomly or by optimization over some or all of the data in the leaf.

DATASET ANALATICS

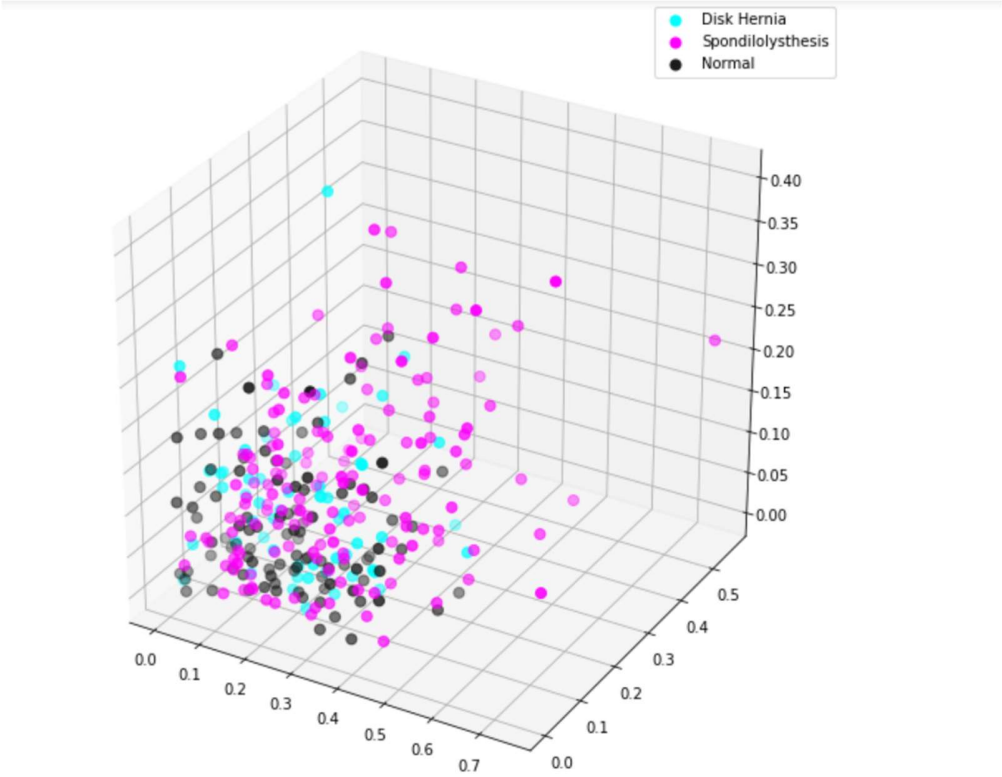
A: Cross-Sections



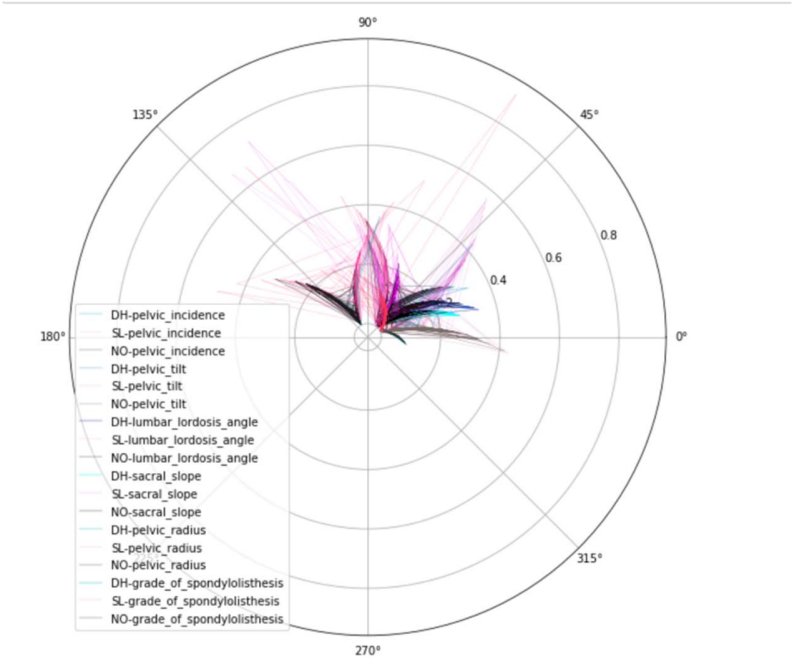
B: PCA (2-components) Plot



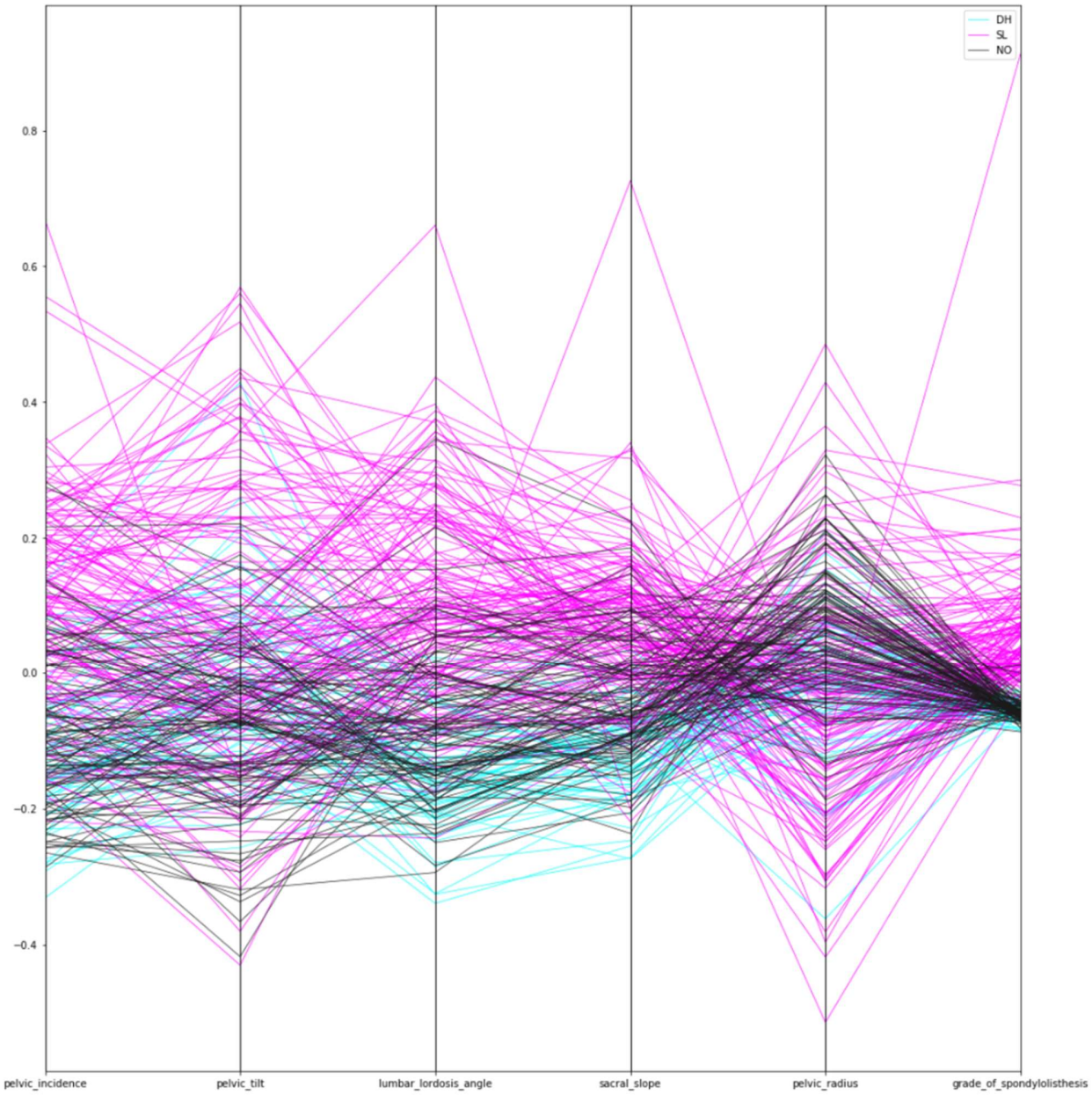
C: PCA (3-components) Plot:



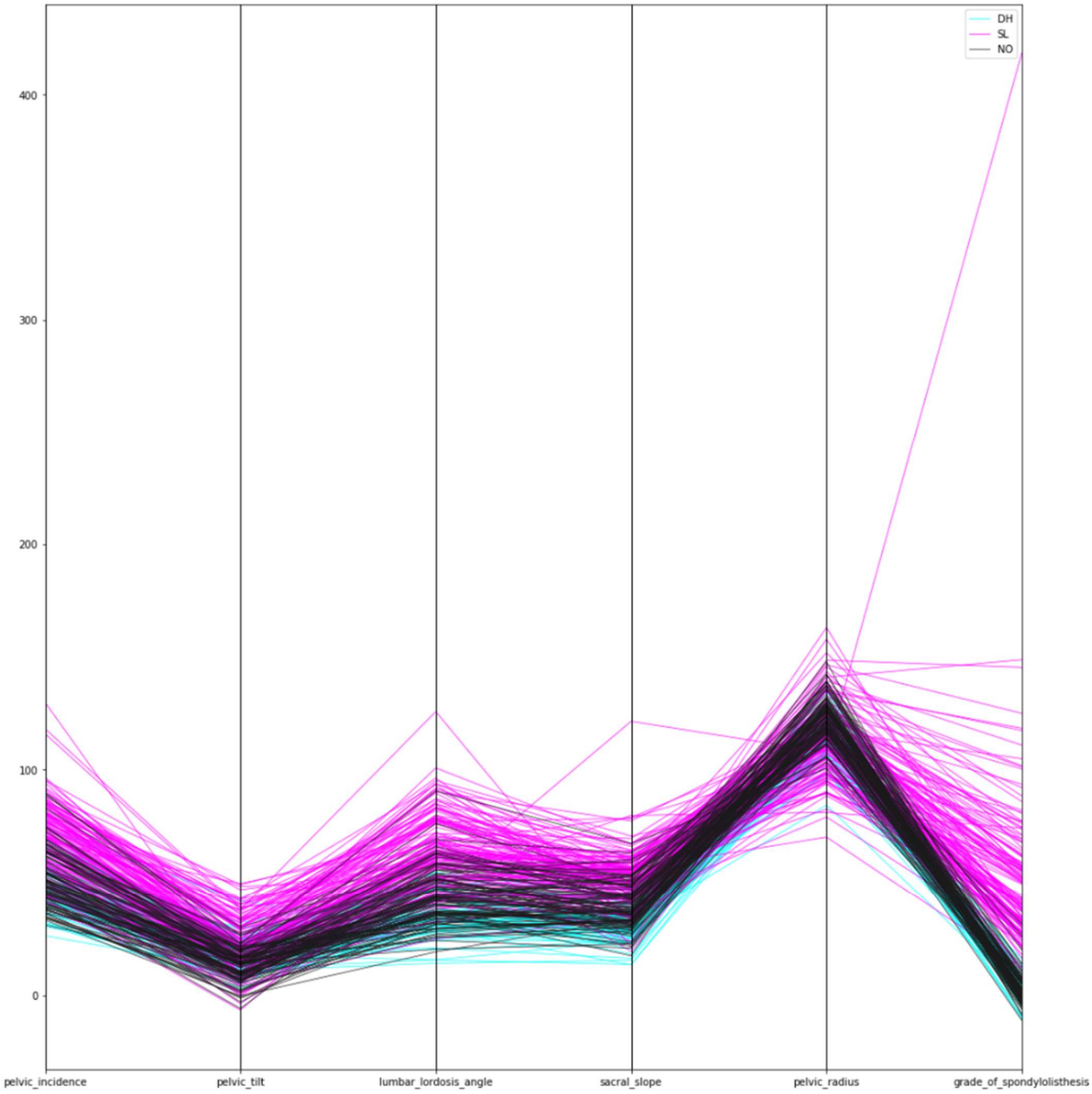
D: Polar Representation:



E: Parallel Coordinates (Normalized):



F: Parallel Coordinates:



ALGORITHM IMPLEMENTATION

In [1]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.preprocessing import normalize,scale
from sklearn.decomposition import PCA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA as sklearnPCA
from pandas.plotting import parallel_coordinates
```

In [2]:
```
#importing dataset and converting to datasframe
header = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacr
al_slope', 'pelvic_radius', 'grade_of_spondylolisthesis', 'label']
df = pd.read_csv('./vertebral_column_data/column_3C.dat', sep=' ', header=N
one, names=header)
print(df.dtypes)
```

```
pelvic_incidence             float64
pelvic_tilt                  float64
lumbar_lordosis_angle        float64
sacral_slope                 float64
pelvic_radius                float64
grade_of_spondylolisthesis   float64
label                         object
dtype: object
```

In [3]:
```
# extracting features and lables
x = df.iloc[:,0:6]
y = df.iloc[:,6]

# split into training and test subsets
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
print(x_test.shape)
```

```
(124, 6)
```

In [4]:
```
model = KNeighborsClassifier(n_neighbors=5, weights='distance')
```

In [5]:
```
#10-fold cross validation
scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
# print scores
print ("10-Fold Accuracy : ", scores.mean()*100)
```

```
10-Fold Accuracy :  83.5483870967742
```

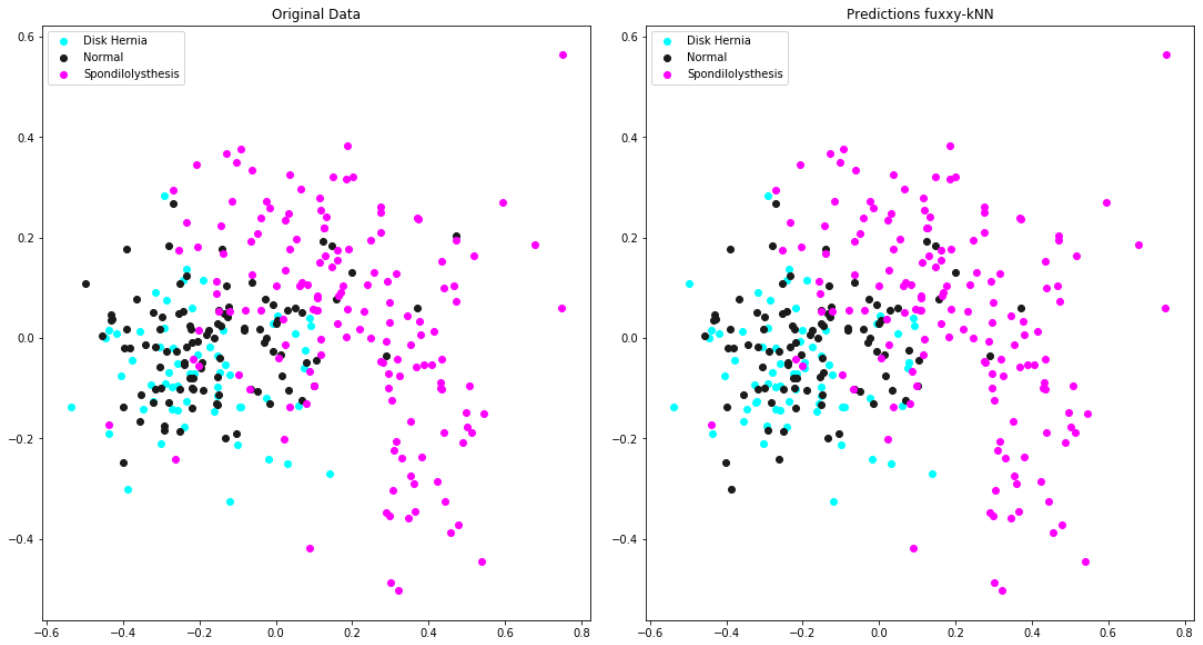
In [6]:
```
#creation of the confusion matrix
model.fit(x_train,y_train)
print ("Testing Accuracy : ",model.score(x_test, y_test)*100)

color_dict = {'DH' : 'cyan', 'SL' : 'magenta', 'NO' : '#1B1B1B'}
predicted_test = model.predict(x_test)
predicted = model.predict(x)
predicted_test_proba = model.predict_proba(x_test)
```

```
Testing Accuracy :  85.48387096774194
```

In [7]:
```
# confusion matrix
cm = metrics.confusion_matrix(y_test, predicted_test, labels=['DH', 'NO',
'SL'])
print (cm)
print()
print (metrics.classification_report(y_test, predicted_test))
```

```
[[18  4  0]
 [10 28  2]
 [ 1  1 60]]

             precision    recall  f1-score   support

         DH       0.62      0.82      0.71        22
         NO       0.85      0.70      0.77        40
         SL       0.97      0.97      0.97        62

avg / total       0.87      0.85      0.86       124
```

In [8]:
```
# mean normalization and feature scaling
x_norm = (x - x.mean())/(x.max() - x.min())

# principle component analysis
pca = PCA(n_components=2) #2-dimensional PCA
# print(pca.fit_transform(x_norm))
transformed = pd.DataFrame(pca.fit_transform(x_norm))
# print(transformed.iloc[[0,1,2],:])
# plot
fig = plt.figure(figsize=(15,8))
ax1 = fig.add_subplot(121)
ax1.scatter(transformed[y=='DH'][0], transformed[y=='DH'][1], label='Disk H
ernia', c='cyan')
ax1.scatter(transformed[y=='NO'][0], transformed[y=='NO'][1], label='Norma
l', c='#1B1B1B')
ax1.scatter(transformed[y=='SL'][0], transformed[y=='SL'][1], label='Spondi
lolysthesis', c='magenta')
ax1.legend()
ax1.set_title('Original Data')

ax2 = fig.add_subplot(122)
ax2.scatter(transformed[predicted=='DH'][0], transformed[predicted=='DH'][1
], label='Disk Hernia', c='cyan')
ax2.scatter(transformed[predicted=='NO'][0], transformed[predicted=='NO'][1
], label='Normal', c='#1B1B1B')
ax2.scatter(transformed[predicted=='SL'][0], transformed[predicted=='SL'][1
], label='Spondilolysthesis', c='magenta')
ax2.legend()
ax2.set_title('Predictions fuxxy-kNN')

plt.tight_layout(pad=0.4, w_pad=1.5, h_pad=2.0)
plt.show()
```



In [1]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.decomposition import PCA

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

In [2]:
```
header = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacr
al_slope', 'pelvic_radius', 'grade_of_spondylolisthesis', 'label']
df = pd.read_csv(filepath_or_buffer='./vertebral_column_data/column_3C.dat'
, header=None, sep=' ', names=header)
print(df.dtypes)
```

```
pelvic_incidence             float64
pelvic_tilt                  float64
lumbar_lordosis_angle        float64
sacral_slope                 float64
pelvic_radius                float64
grade_of_spondylolisthesis   float64
label                         object
dtype: object
```

In [3]:
```
x = df.iloc[:, 0:6]
y = df.iloc[:, 6]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.4)
```

In [4]:
```
# inv of regularization constant
c = 3 #(lambda = 1/3)

model = LogisticRegression(C=c, multi_class='ovr')
```

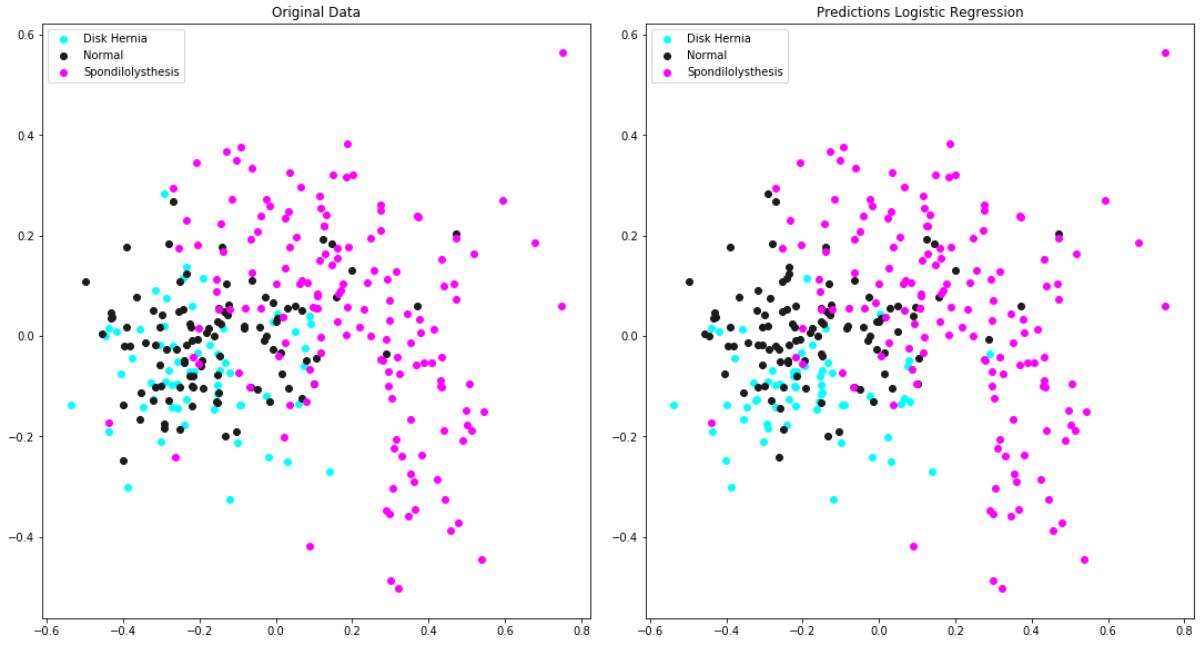
In [5]:
```
# 10 fold cross-validation
scores = cross_val_score(model, x, y, scoring='accuracy', cv=10)
print('10 fold accuracy: {}'.format(scores.mean()*100))
```

```
10 fold accuracy: 83.87096774193547
```

In [6]:
```
# creation of confusion matrix
model = model.fit(x_train, y_train)
print('Testing accuracy: {}'.format(model.score(x_test, y_test)*100))

predicted = model.predict(x)

cm = metrics.confusion_matrix(y, predicted, labels=['DH', 'NO', 'SL'])
print(cm)

print(metrics.classification_report(y, predicted))
```

```
Testing accuracy: 84.67741935483872
[[ 41  17   2]
 [ 17  81   2]
 [  2   3 145]]
             precision    recall  f1-score   support

         DH       0.68      0.68      0.68        60
         NO       0.80      0.81      0.81       100
         SL       0.97      0.97      0.97       150

avg / total       0.86      0.86      0.86       310
```

In [11]:
```
# mean normalization and feature scaling
x_norm = (x - x.mean())/(x.max() - x.min())

# principle component analysis
pca = PCA(n_components=2) #2-dimensional PCA
# print(pca.fit_transform(x_norm))
transformed = pd.DataFrame(pca.fit_transform(x_norm))
predicted = pd.DataFrame(predicted[:]).iloc[:, 0]

# print(transformed[y=='DH'])

# plot
fig = plt.figure(figsize=(15,8))
ax1 = fig.add_subplot(121)
ax1.scatter(transformed[y=='DH'][0], transformed[y=='DH'][1], label='Disk H
ernia', c='cyan')
ax1.scatter(transformed[y=='NO'][0], transformed[y=='NO'][1], label='Norma
l', c='#1B1B1B')
ax1.scatter(transformed[y=='SL'][0], transformed[y=='SL'][1], label='Spondi
lolysthesis', c='magenta')
ax1.legend()
ax1.set_title('Original Data')

ax2 = fig.add_subplot(122)
ax2.scatter(transformed[predicted=='DH'][0], transformed[predicted=='DH'][1
], label='Disk Hernia', c='cyan')
ax2.scatter(transformed[predicted=='NO'][0], transformed[predicted=='NO'][1
], label='Normal', c='#1B1B1B')
ax2.scatter(transformed[predicted=='SL'][0], transformed[predicted=='SL'][1
], label='Spondilolysthesis', c='magenta')
ax2.legend()
ax2.set_title('Predictions Logistic Regression')

plt.tight_layout(pad=0.4, w_pad=1.5, h_pad=2.0)
plt.show()
```



In [1]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn import metrics

import numpy as np
import pandas as pd
```

In [2]:
```
dataset = pd.read_csv('./vertebral_column_data/column_3C.dat', delimiter =
' ', header = None)
X = dataset.iloc[:,0:-1].values
Y = dataset.iloc[:,-1].values
```

In [3]:
```
le = LabelEncoder()
Y = le.fit_transform(Y)
#Y = np.reshape(Y,(310,1))
#onehotencoder = OneHotEncoder(categorical_features=[0])
#Y = onehotencoder.fit_transform(Y).toarray()

X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random
_state = 0)
```

In [4]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = RandomForestClassifier(n_estimators=10, criterion = 'entropy',
random_state = 0)
classsifier = classifier.fit(X_train,Y_train)
```

In [5]:
```
Y_pred = classifier.predict(X_test)

reverse = dict(zip(range(3),['DH','NO','SL']))
Y_test = np.vectorize(reverse.get)(Y_test)
Y_pred = np.vectorize(reverse.get)(Y_pred)

cm = confusion_matrix(Y_test, Y_pred)

print(cm)
print(metrics.classification_report(Y_test, Y_pred))
```

```
[[ 3  9  1]
 [ 1 16  2]
 [ 0  1 29]]
             precision    recall  f1-score   support

         DH       0.75      0.23      0.35        13
         NO       0.62      0.84      0.71        19
         SL       0.91      0.97      0.94        30

avg / total       0.78      0.77      0.74        62
```

In [1]:
```
from sklearn.preprocessing import LabelEncoder,OneHotEncoder, StandardScale
r
from sklearn.model_selection import train_test_split

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

import keras
from keras.layers import Dense
from keras.models import Sequential
```

```
Using TensorFlow backend.
```

In [2]:
```
dataset = pd.read_csv('./vertebral_column_data/column_3C.dat', delimiter =
' ', header = None)
X = dataset.iloc[:,0:-1].values
Y = dataset.iloc[:,-1].values

le = LabelEncoder()
Y = le.fit_transform(Y)
Y = np.reshape(Y,(310,1))
onehotencoder = OneHotEncoder(categorical_features=[0])
Y = onehotencoder.fit_transform(Y).toarray()
```

In [3]:
```
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size = 0.2,random
_state = 0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = Sequential()

classifier.add(Dense(units=5, input_dim = 6, kernel_initializer='uniform',
activation='relu'))

classifier.add(Dense(units=4, kernel_initializer='uniform', activation='rel
u'))

classifier.add(Dense(units=3,  kernel_initializer='uniform', activation='so
ftmax'))

classifier.compile(optimizer = 'adam', loss = 'categorical_crossentropy',me
trics = ['accuracy'] )
```

In [4]:
```
classifier.fit(X_train,Y_train,epochs = 100, batch_size = 10)
```

```
Epoch 1/100
248/248 [==============================] - 0s 951us/step - loss: 1.0961 - a
cc: 0.4718
Epoch 2/100
248/248 [==============================] - 0s 118us/step - loss: 1.0910 - a
cc: 0.4839
Epoch 3/100
248/248 [==============================] - 0s 144us/step - loss: 1.0857 - a
cc: 0.4839
Epoch 4/100
248/248 [==============================] - 0s 165us/step - loss: 1.0784 - a
cc: 0.4839
Epoch 5/100
248/248 [==============================] - 0s 128us/step - loss: 1.0664 - a
cc: 0.4839
Epoch 6/100
248/248 [==============================] - 0s 117us/step - loss: 1.0460 - a
cc: 0.4839
Epoch 7/100
248/248 [==============================] - 0s 127us/step - loss: 1.0136 - a
cc: 0.5081
Epoch 8/100
248/248 [==============================] - 0s 128us/step - loss: 0.9663 - a
cc: 0.6210
Epoch 9/100
248/248 [==============================] - 0s 126us/step - loss: 0.9105 - a
cc: 0.6774
Epoch 10/100
248/248 [==============================] - 0s 150us/step - loss: 0.8507 - a
cc: 0.7016
Epoch 11/100
248/248 [==============================] - 0s 129us/step - loss: 0.7976 - a
cc: 0.7177
Epoch 12/100
248/248 [==============================] - 0s 117us/step - loss: 0.7513 - a
cc: 0.7097
Epoch 13/100
248/248 [==============================] - 0s 137us/step - loss: 0.7165 - a
cc: 0.6895
Epoch 14/100
248/248 [==============================] - 0s 121us/step - loss: 0.6892 - a
cc: 0.6935
Epoch 15/100
248/248 [==============================] - ETA: 0s - loss: 0.5978 - acc: 0.
700 - 0s 135us/step - loss: 0.6687 - acc: 0.6895
Epoch 16/100
248/248 [==============================] - 0s 130us/step - loss: 0.6530 - a
cc: 0.6935
Epoch 17/100
248/248 [==============================] - 0s 121us/step - loss: 0.6395 - a
cc: 0.6976
Epoch 18/100
248/248 [==============================] - 0s 131us/step - loss: 0.6279 - a
cc: 0.7016
Epoch 19/100
248/248 [==============================] - 0s 118us/step - loss: 0.6186 - a
cc: 0.7016
```

```
Epoch 20/100
248/248 [==============================] - 0s 127us/step - loss: 0.6091 - a
cc: 0.7056
Epoch 21/100
248/248 [==============================] - 0s 131us/step - loss: 0.6008 - a
cc: 0.7056
Epoch 22/100
248/248 [==============================] - 0s 159us/step - loss: 0.5931 - a
cc: 0.7097
Epoch 23/100
248/248 [==============================] - 0s 123us/step - loss: 0.5855 - a
cc: 0.7097
Epoch 24/100
248/248 [==============================] - 0s 120us/step - loss: 0.5770 - a
cc: 0.7137
Epoch 25/100
248/248 [==============================] - 0s 116us/step - loss: 0.5689 - a
cc: 0.7218
Epoch 26/100
248/248 [==============================] - 0s 154us/step - loss: 0.5601 - a
cc: 0.7218
Epoch 27/100
248/248 [==============================] - 0s 116us/step - loss: 0.5505 - a
cc: 0.7258
Epoch 28/100
248/248 [==============================] - 0s 132us/step - loss: 0.5399 - a
cc: 0.7258
Epoch 29/100
248/248 [==============================] - 0s 127us/step - loss: 0.5308 - a
cc: 0.7339
Epoch 30/100
248/248 [==============================] - 0s 121us/step - loss: 0.5212 - a
cc: 0.7419
Epoch 31/100
248/248 [==============================] - 0s 122us/step - loss: 0.5118 - a
cc: 0.7379
Epoch 32/100
248/248 [==============================] - 0s 130us/step - loss: 0.5030 - a
cc: 0.7419
Epoch 33/100
248/248 [==============================] - 0s 115us/step - loss: 0.4946 - a
cc: 0.7460
Epoch 34/100
248/248 [==============================] - 0s 122us/step - loss: 0.4870 - a
cc: 0.7460
Epoch 35/100
248/248 [==============================] - 0s 115us/step - loss: 0.4800 - a
cc: 0.7460
Epoch 36/100
248/248 [==============================] - 0s 132us/step - loss: 0.4729 - a
cc: 0.7581
Epoch 37/100
248/248 [==============================] - 0s 135us/step - loss: 0.4662 - a
cc: 0.7621
Epoch 38/100
248/248 [==============================] - 0s 138us/step - loss: 0.4600 - a
cc: 0.7661
```

```
Epoch 39/100
248/248 [==============================] - 0s 119us/step - loss: 0.4542 - a
cc: 0.7702
Epoch 40/100
248/248 [==============================] - 0s 136us/step - loss: 0.4490 - a
cc: 0.7742
Epoch 41/100
248/248 [==============================] - 0s 112us/step - loss: 0.4439 - a
cc: 0.7782
Epoch 42/100
248/248 [==============================] - 0s 128us/step - loss: 0.4392 - a
cc: 0.7823
Epoch 43/100
248/248 [==============================] - 0s 120us/step - loss: 0.4357 - a
cc: 0.7823
Epoch 44/100
248/248 [==============================] - 0s 116us/step - loss: 0.4318 - a
cc: 0.7823
Epoch 45/100
248/248 [==============================] - 0s 122us/step - loss: 0.4286 - a
cc: 0.7823
Epoch 46/100
248/248 [==============================] - 0s 112us/step - loss: 0.4253 - a
cc: 0.7823
Epoch 47/100
248/248 [==============================] - 0s 127us/step - loss: 0.4231 - a
cc: 0.7863
Epoch 48/100
248/248 [==============================] - 0s 118us/step - loss: 0.4203 - a
cc: 0.7863
Epoch 49/100
248/248 [==============================] - 0s 116us/step - loss: 0.4181 - a
cc: 0.7863
Epoch 50/100
248/248 [==============================] - 0s 128us/step - loss: 0.4151 - a
cc: 0.7863
Epoch 51/100
248/248 [==============================] - 0s 124us/step - loss: 0.4127 - a
cc: 0.7863
Epoch 52/100
248/248 [==============================] - 0s 103us/step - loss: 0.4111 - a
cc: 0.7863
Epoch 53/100
248/248 [==============================] - 0s 113us/step - loss: 0.4091 - a
cc: 0.7863
Epoch 54/100
248/248 [==============================] - 0s 127us/step - loss: 0.4074 - a
cc: 0.7863
Epoch 55/100
248/248 [==============================] - 0s 95us/step - loss: 0.4051 - ac
c: 0.7863
Epoch 56/100
248/248 [==============================] - 0s 141us/step - loss: 0.4030 - a
cc: 0.7863
Epoch 57/100
248/248 [==============================] - 0s 116us/step - loss: 0.4008 - a
cc: 0.7863
```

```
Epoch 58/100
248/248 [==============================] - 0s 109us/step - loss: 0.4000 - a
cc: 0.7863
Epoch 59/100
248/248 [==============================] - 0s 130us/step - loss: 0.3976 - a
cc: 0.7903
Epoch 60/100
248/248 [==============================] - 0s 116us/step - loss: 0.3953 - a
cc: 0.7863
Epoch 61/100
248/248 [==============================] - 0s 119us/step - loss: 0.3928 - a
cc: 0.8024
Epoch 62/100
248/248 [==============================] - 0s 116us/step - loss: 0.3911 - a
cc: 0.8065
Epoch 63/100
248/248 [==============================] - 0s 126us/step - loss: 0.3894 - a
cc: 0.8065
Epoch 64/100
248/248 [==============================] - 0s 109us/step - loss: 0.3874 - a
cc: 0.8024
Epoch 65/100
248/248 [==============================] - 0s 114us/step - loss: 0.3860 - a
cc: 0.8065
Epoch 66/100
248/248 [==============================] - 0s 110us/step - loss: 0.3839 - a
cc: 0.8145
Epoch 67/100
248/248 [==============================] - 0s 110us/step - loss: 0.3830 - a
cc: 0.8185
Epoch 68/100
248/248 [==============================] - 0s 109us/step - loss: 0.3806 - a
cc: 0.8185
Epoch 69/100
248/248 [==============================] - 0s 115us/step - loss: 0.3801 - a
cc: 0.8145
Epoch 70/100
248/248 [==============================] - 0s 110us/step - loss: 0.3772 - a
cc: 0.8226
Epoch 71/100
248/248 [==============================] - 0s 112us/step - loss: 0.3752 - a
cc: 0.8185
Epoch 72/100
248/248 [==============================] - 0s 117us/step - loss: 0.3742 - a
cc: 0.8226
Epoch 73/100
248/248 [==============================] - 0s 127us/step - loss: 0.3710 - a
cc: 0.8266
Epoch 74/100
248/248 [==============================] - 0s 119us/step - loss: 0.3699 - a
cc: 0.8226
Epoch 75/100
248/248 [==============================] - 0s 109us/step - loss: 0.3677 - a
cc: 0.8266
Epoch 76/100
248/248 [==============================] - 0s 114us/step - loss: 0.3668 - a
cc: 0.8266
```

```
Epoch 77/100
248/248 [==============================] - 0s 118us/step - loss: 0.3641 - a
cc: 0.8266
Epoch 78/100
248/248 [==============================] - 0s 113us/step - loss: 0.3625 - a
cc: 0.8266
Epoch 79/100
248/248 [==============================] - 0s 107us/step - loss: 0.3595 - a
cc: 0.8226
Epoch 80/100
248/248 [==============================] - 0s 117us/step - loss: 0.3574 - a
cc: 0.8226
Epoch 81/100
248/248 [==============================] - 0s 112us/step - loss: 0.3560 - a
cc: 0.8266
Epoch 82/100
248/248 [==============================] - 0s 119us/step - loss: 0.3558 - a
cc: 0.8387
Epoch 83/100
248/248 [==============================] - 0s 114us/step - loss: 0.3511 - a
cc: 0.8306
Epoch 84/100
248/248 [==============================] - 0s 112us/step - loss: 0.3498 - a
cc: 0.8266
Epoch 85/100
248/248 [==============================] - 0s 119us/step - loss: 0.3480 - a
cc: 0.8306
Epoch 86/100
248/248 [==============================] - 0s 110us/step - loss: 0.3459 - a
cc: 0.8347
Epoch 87/100
248/248 [==============================] - 0s 110us/step - loss: 0.3430 - a
cc: 0.8347
Epoch 88/100
248/248 [==============================] - 0s 106us/step - loss: 0.3413 - a
cc: 0.8387
Epoch 89/100
248/248 [==============================] - 0s 122us/step - loss: 0.3385 - a
cc: 0.8387
Epoch 90/100
248/248 [==============================] - 0s 102us/step - loss: 0.3374 - a
cc: 0.8427
Epoch 91/100
248/248 [==============================] - 0s 120us/step - loss: 0.3342 - a
cc: 0.8387
Epoch 92/100
248/248 [==============================] - 0s 107us/step - loss: 0.3338 - a
cc: 0.8468
Epoch 93/100
248/248 [==============================] - 0s 121us/step - loss: 0.3308 - a
cc: 0.8508
Epoch 94/100
248/248 [==============================] - 0s 121us/step - loss: 0.3285 - a
cc: 0.8589
Epoch 95/100
248/248 [==============================] - 0s 109us/step - loss: 0.3272 - a
cc: 0.8710
```

```
Epoch 96/100
248/248 [==============================] - 0s 109us/step - loss: 0.3245 - a
cc: 0.8831
Epoch 97/100
248/248 [==============================] - 0s 109us/step - loss: 0.3230 - a
cc: 0.8710
Epoch 98/100
248/248 [==============================] - 0s 120us/step - loss: 0.3209 - a
cc: 0.8871
Epoch 99/100
248/248 [==============================] - 0s 98us/step - loss: 0.3194 - ac
c: 0.8790
Epoch 100/100
248/248 [==============================] - 0s 114us/step - loss: 0.3171 - a
cc: 0.8750
```

Out[4]: <keras.callbacks.History at 0x7fc05b9ded68>

In [5]:
```
Y_pred = classifier.predict(X_test)
Y_pred = (Y_pred>0.5).astype(float)
print(Y_pred)
```

```
[[0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [1. 0. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
```

```
[0. 0. 1.]
[0. 1. 0.]
[1. 0. 0.]
[0. 1. 0.]
[0. 1. 0.]]
```

```
{
  "pelvic_incidence": 33.03,
  "pelvic_tilt": 12.55,
  "lumbar_lordosis_angle": 49.61,
  "sacral_slope":  60.48,
  "pelvic_radius": 88.67,
  "grade_of_spondylolisthesis": -0.75
}
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn import metrics
from sklearn.preprocessing import normalize,scale
from sklearn.decomposition import PCA

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA as sklearnPCA
from pandas.plotting import parallel_coordinates

def knn(report, sample_path='./vertebral_column_data/column_3C.dat'):
    #importing dataset and converting to datasframe
    header = ['pelvic_incidence', 'pelvic_tilt', 'lumbar_lordosis_angle', 'sacral_slope', 'pel
vic_radius', 'grade_of_spondylolisthesis', 'label']
    df = pd.read_csv(sample_path, sep=' ', header=None, names=header)

    features = df.iloc[:,0:6]
    labels = df.iloc[:,6]
    model = KNeighborsClassifier(n_neighbors=5, weights='distance')
    model.fit(features,labels)

    predicted = model.predict(report)
    predicted_proba = model.predict_proba(report)

    return(predicted, predicted_proba)
```

```
import smtplib
from configparser import ConfigParser
from email.mime.text import MIMEText

def sendmail(to, message):
    config = ConfigParser()
    config.read('credentials.ini')

    gmail_user = config.get('email', 'username')
    gmail_password = config.get('email', 'pwd')
    sent_from = gmail_user

    try:
        msg = MIMEText(message)
        msg['Subject'] = 'Medical Report for Vertebral Column Test'
        msg['From'] = gmail_user
        msg['To'] = to

        server = smtplib.SMTP_SSL('smtp.gmail.com', 465)
        server.ehlo()
        server.login(gmail_user, gmail_password)
        server.sendmail(sent_from, to, msg.as_string())
        server.close()

        print('Email sent!')
    except:
        print('Unable to process mail...')
```

```
import json

class Patient:

    def __init__(self, cpf, name, address, telephone, email):
        self.cpf = cpf
        self.name = name
        self.address = address
        self.telephone = telephone
        self.email = email

    @classmethod
    def from_input(cls):
        return cls(
            input('CPF: '),
            input('NAME: '),
            input('Address: '),
            input('Telephone: '),
            input('Email: ')
            )


class VertebralColumnReport:
    def __init__(self, patient, report):
        self.id = patient
        self.pelvic_incidence = report['pelvic_incidence']
        self.pelvic_tilt = report['pelvic_tilt']
        self.lumbar_lordosis_angle = report['lumbar_lordosis_angle']
        self.sacral_slope = report['sacral_slope']
        self.pelvic_radius = report['pelvic_radius']
        self.grade_of_spondylolisthesis = report['grade_of_spondylolisthesis']

    def tolist(self):
        return [self.pelvic_incidence,
            self.pelvic_tilt,
            self.lumbar_lordosis_angle,
            self.sacral_slope,
            self.pelvic_radius,
            self.grade_of_spondylolisthesis]

    def todict(self):
        return {
            'pelvic_incidence' : self.pelvic_incidence,
            'pelvic_tilt' : self.pelvic_tilt,
            'lumbar_lordosis_angle' : self.lumbar_lordosis_angle,
            'sacral_slope' : self.sacral_slope,
            'pelvic_radius' : self.pelvic_radius,
            'grade_of_spondylolisthesis' : self.grade_of_spondylolisthesis
        }
```

```
from patient import Patient, VertebralColumnReport
from FuzzyKnn import knn
from headers import *
from mail import sendmail

label = {'DH' : 'Disk Hernia', 'NO' : 'Normal', 'SL' : 'Spondilolysthesis'}

def printdetails(user, predicted, probability):
    print()
    print('DETAILED REPORT FOR PATIENT {}'.format(user.name))
    print('Diagnosed: {}'.format(label[predicted]))
    accuracy = probability[predicted]
    print('Accuracy: {}%'.format(accuracy))
    print()

    if predicted != 'NO' and accuracy > 75:
        action = 'Booked Follow-up Appointment\nReference for Speciality Hospital also forward
ed.\n'
        print('-'*60)
        # call Appointment Scheduler Module
        print('Follow Up Appointment Scheduled with priority : HIGH')
        print('-'*60)
        # call reference builder
        print('Forwarded report for reference with Speciality Hospital')
        print('-'*60)

    elif (predicted != 'NO' and accuracy > 50):
        action = 'Booked Follow-up Appointment'
        # call Appointment Scheduler Module
        print('-'*60)
        print('Follow Up Appointment Scheduled with priority : MODERATE')
        print('-'*60)

    else :
        action = 'Booked Follow-up Appointment'
        print('-'*60)
        # call Appointment Scheduler Module
        print('Follow Up Appointment Scheduled with priority : LOW')
        print('-'*60)

    msg = 'Diagnosed: {}\nAccuracy: {}\n{}'.format(label[predicted], accuracy, action)
    # print(msg, user.email)
    sendmail(user.email, msg)

def main():
    user = Patient.from_input()
    report_path = input('Report Path: ')

    with open(report_path) as file:
        report = json.load(file)

    vcr = VertebralColumnReport(user, report)
    data = pd.DataFrame(vcr.todict(), index=[0])

    predicted, predicted_proba = knn(data)

    probability = {'DH' : predicted_proba[0][0]*100, 'NO' : predicted_proba[0][1]*100, 'SL' :
predicted_proba[0][2]*100}

    printdetails(user, predicted[0], probability)

if __name__ == '__main__':
    main()
```