

**DATA VISUALISATION – CSE3020**

PROJECT REPORT

**TITLE:**

Analysis and Prediction of Vertebral Column Abnormalities

**SUBMITTED TO:**

Prof. Pattabiraman V., VIT CHENNAI.

**NAME AND REG.NO:**

Ayush Sharma - 15BCE1335

**Abstract**

The spine, or backbone, is made up of small bones (vertebrae) stacked -- along with discs -- one on top of another. A healthy spine when viewed from the side has gentle curves to it. The curves help the spine absorb stress from body movement and gravity.

When viewed from the back, the spine should run straight down the middle of the back. When abnormalities of the spine occur, the natural curvatures of the spine are misaligned or exaggerated in certain areas. These misalignments could be used to detect and predict the abnormalities in the vertebral column.

**Introduction**

In this project, I have used the vertebral column dataset available on UCI repository to analyse and predict vertebral column abnormalities using machine learning models. I have first done some pre-processing on the dataset followed by the implementation of certain data visualisation concepts to gain some insight in the dataset.

The dataset consists of 310 instances of patients belonging to 3 classes: Normal (N), Disk Hernia (DH) & Spondylolisthesis (SP).

These patients are represented using 6 biomechanical attributes derived from shape and orientation of the vertebral column which are discussed thoroughly in the dataset section.

**Objective**

My objective in this project is to apply certain data visualisation techniques to gain some insight about this vertebral column dataset and to get some inferences that would help in applying machine learning for the analysis phase.

My main objective is to use machine learning models to classify the patients in the following classes: Normal(N), Disk Hernia(DH) & Spondylolisthesis(SP). Furthermore, DH and SP patients can be clubbed together to form a new class of Abnormal (Ab) patients.

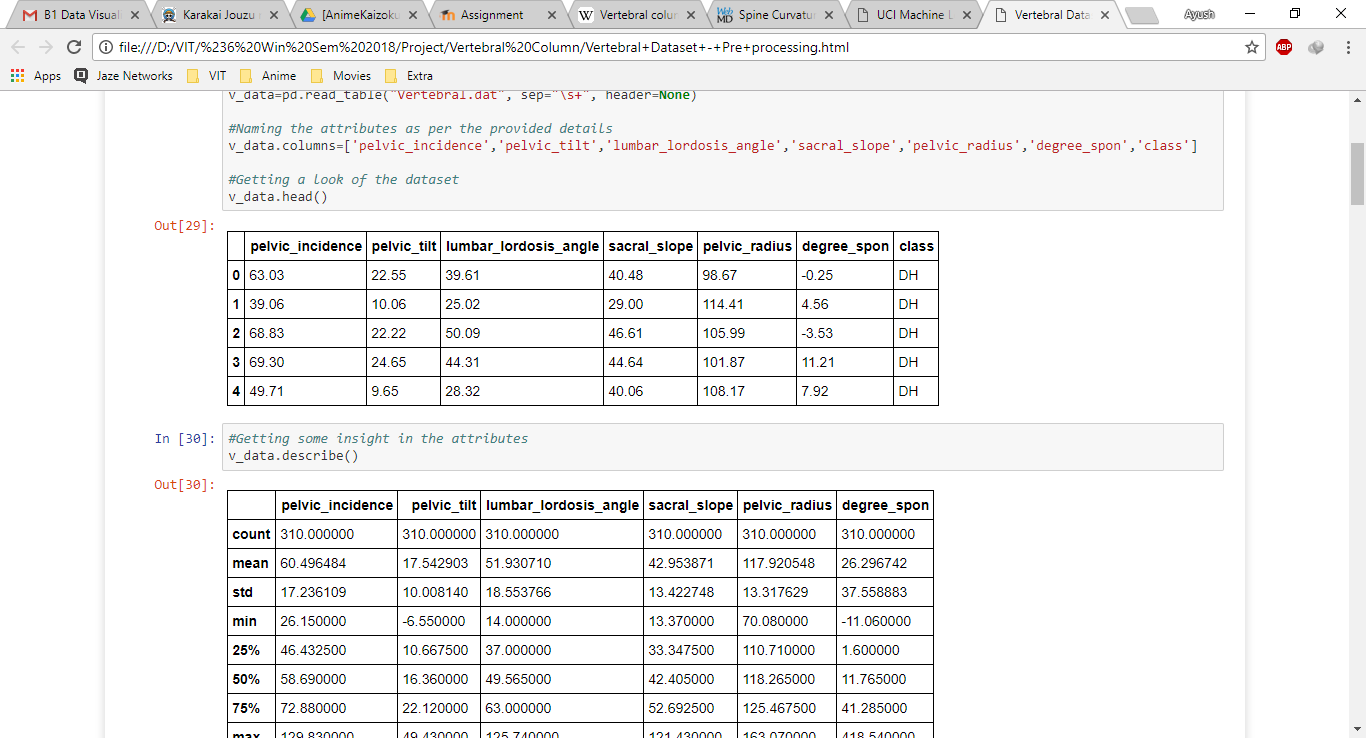
Hence, this dataset can be used as binary classification as well as multi-class classification.

**Dataset**

I have used the vertebral column dataset available in UCI repository in the following [link](http://archive.ics.uci.edu/ml/datasets/vertebral+column).

This dataset has 310 instances which represent patients belonging to 3 different classes and 6 biochemical attributes derived from shape and orientation of pelvis and lumbar spine: pelvic incidence, pelvic tilt, lumbar lordosis angle, sacral slope, pelvic radius and grade of spondylolisthesis.

The head view of the dataset is as follows:



**Coding with output Screenshots**

**import** **pandas** **as** **pd**

%**matplotlib** inline

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sb**

*#Reading the data file in .dat format*

*#Using \s+ as space separator*

*#And taking Header= None such that indexing starts from 0th row*

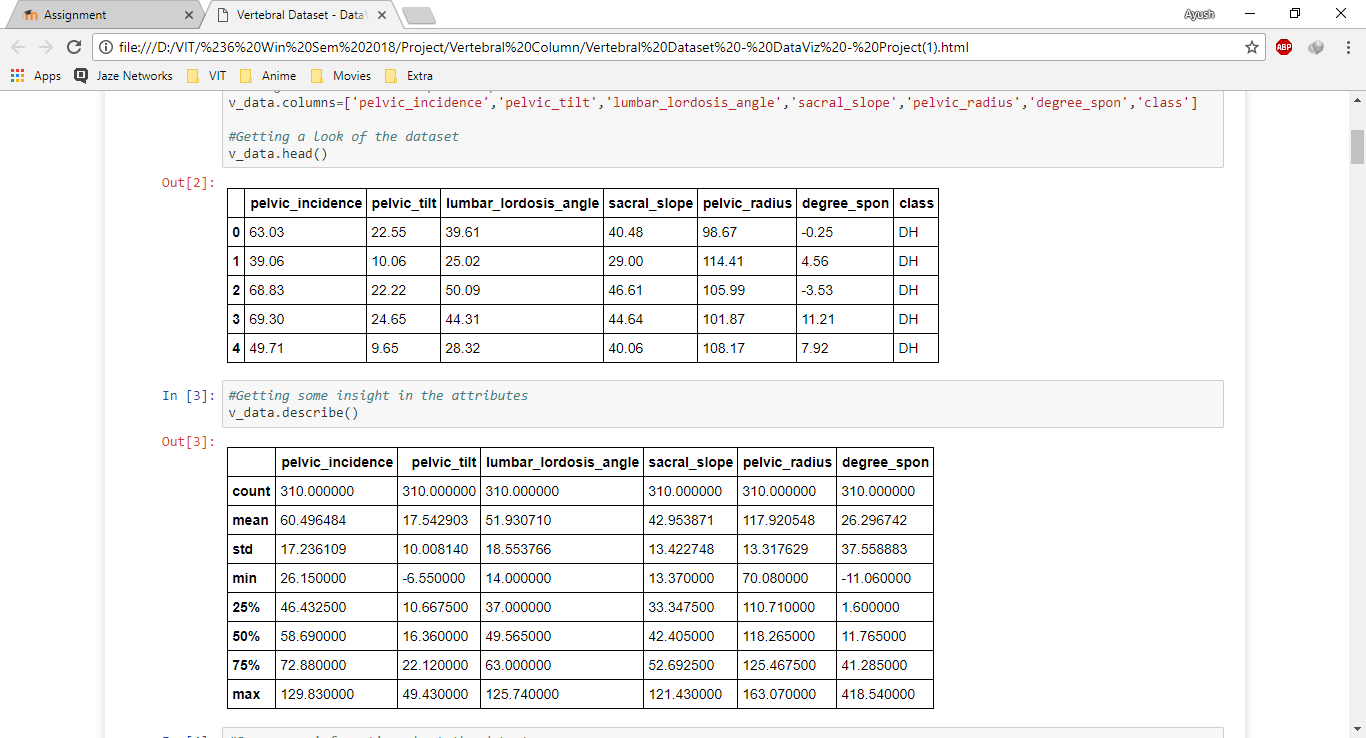
v\_data=pd.read\_table("Vertebral.dat", sep="\s+", header=None)

*#Naming the attributes as per the provided details*

v\_data.columns=['pelvic\_incidence','pelvic\_tilt','lumbar\_lordosis\_angle','sacral\_slope','pelvic\_radius','degree\_spon','class']

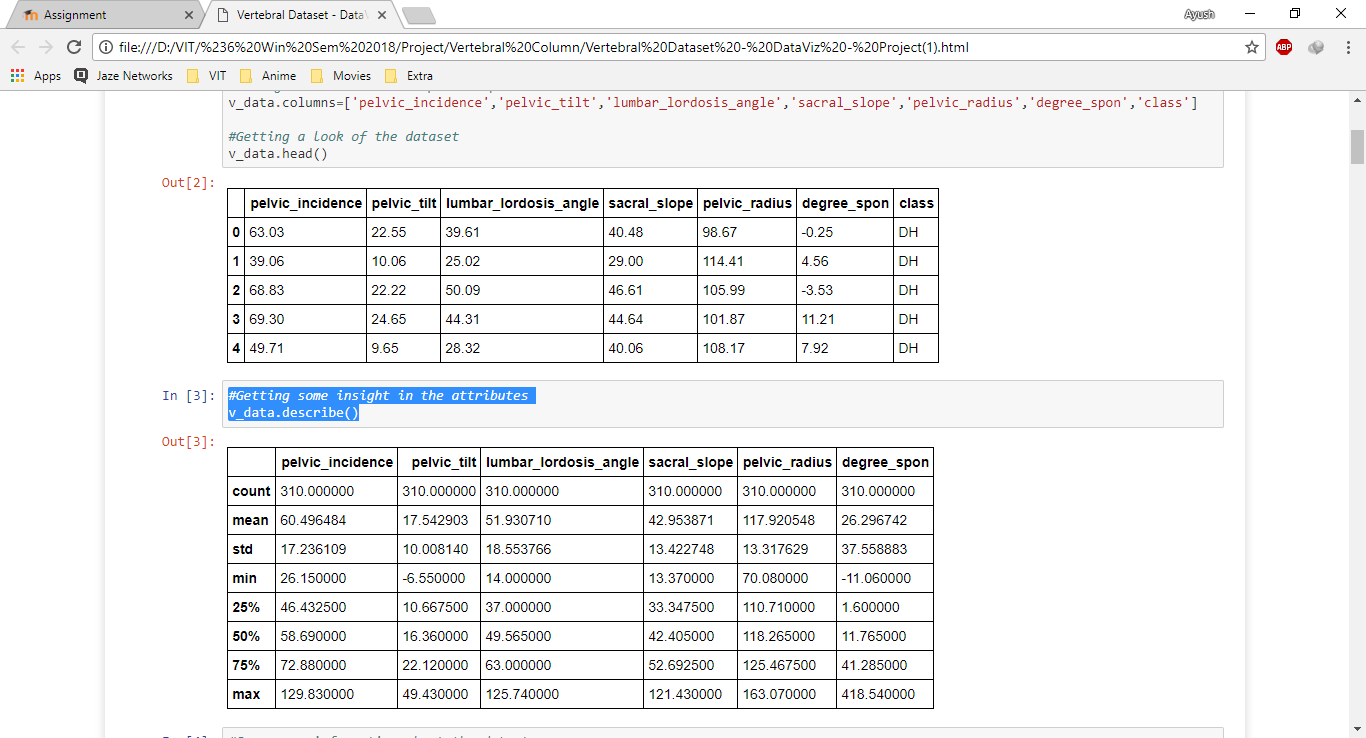
*#Getting a look of the dataset*

v\_data.head()



*#Getting some insight in the attributes*

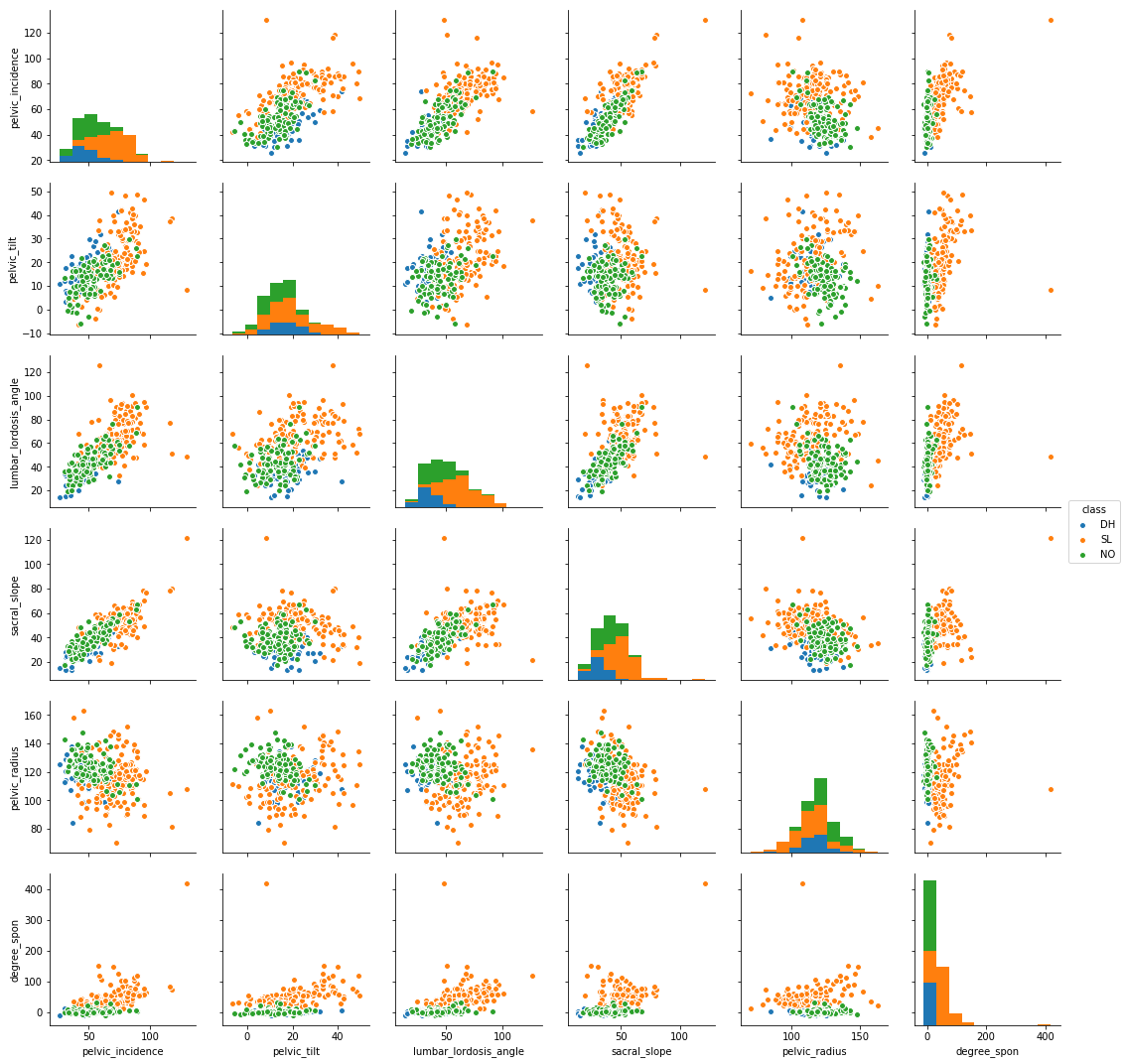
v\_data.describe()



*#Using pariplot from seaborn package*

*#To visualize the relationships among the attributes*

sb.pairplot(v\_data,hue='class')



*# Visualising the histogram of attributes for any outliers*

plt.figure(figsize=(10, 10))

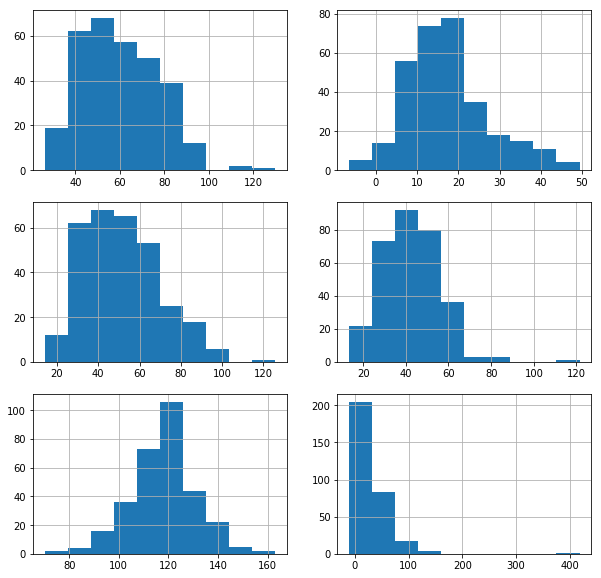
**for** column\_index, column **in** enumerate(v\_data.columns):

**if** column == 'class':

**continue**

plt.subplot(3, 2, column\_index + 1)

v\_data[column].hist()

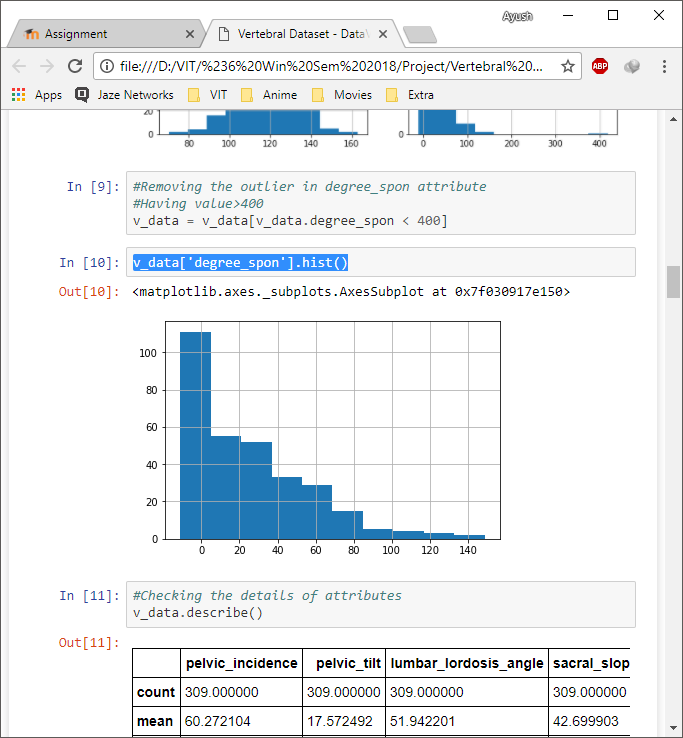


*#Removing the outlier in degree\_spon attribute*

*#Having value>400*

v\_data = v\_data[v\_data.degree\_spon < 400]

v\_data['degree\_spon'].hist()



*#Creating the heatmap of correlation among attributes and class*

**import** **numpy** **as** **np**

v\_data['class'] = v\_data['class'].map({'DH': 1, 'SL': 2, 'NO': 3})

cols=['pelvic\_incidence','pelvic\_tilt','lumbar\_lordosis\_angle','sacral\_slope','pelvic\_radius','degree\_spon','class']

cm = np.corrcoef(v\_data[cols].values.T)

sb.set(font\_scale=1.5)

hm=sb.heatmap(cm,

cbar=True,

annot=True,

square=True,

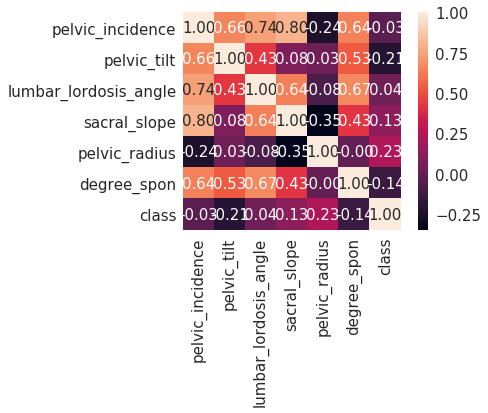
fmt='.2f',

annot\_kws={'size':15},

yticklabels=cols,

xticklabels=cols)

plt.show()



plt.figure(figsize=(10, 10))

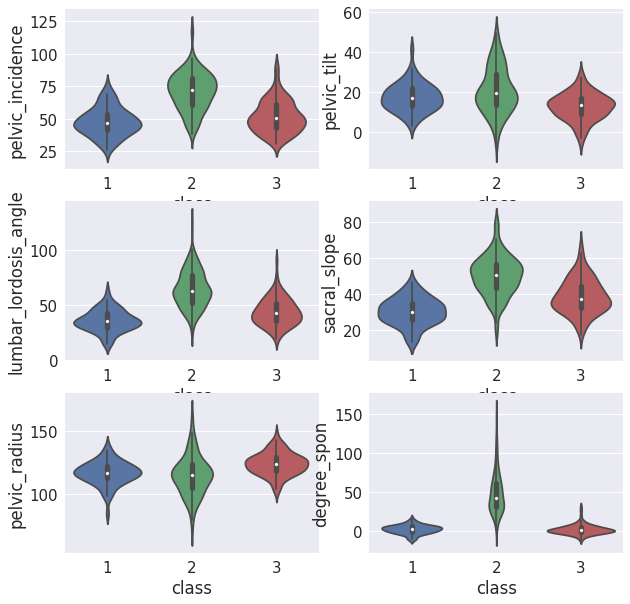
**for** column\_index, column **in** enumerate(v\_data.columns):

**if** column == 'class':

**continue**

plt.subplot(3, 2, column\_index + 1)

sb.violinplot(x='class', y=column, data=v\_data)



plt.figure(figsize=(10, 10))

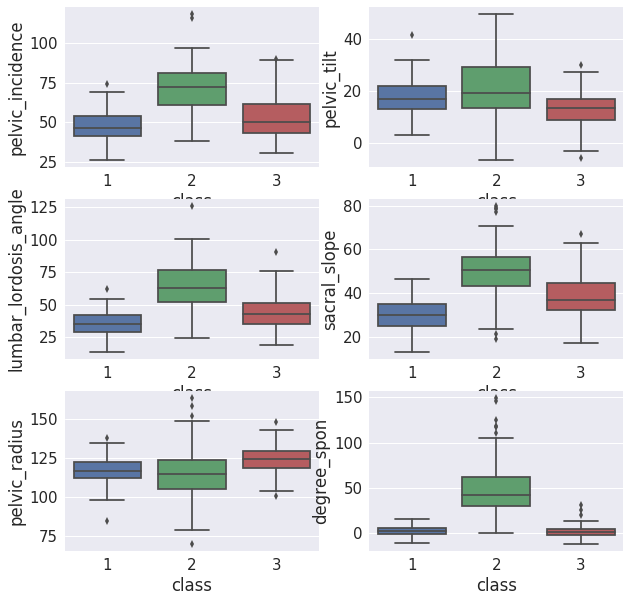
**for** column\_index, column **in** enumerate(v\_data.columns):

**if** column == 'class':

**continue**

plt.subplot(3, 2, column\_index + 1)

sb.boxplot(x='class', y=column, data=v\_data)



*#Training the models for two classes: Normal and Abnormal*

*#Importing necessary libraries for classification*

**from** **sklearn.neighbors** **import** KNeighborsClassifier

**from** **sklearn.tree** **import** DecisionTreeClassifier

**from** **sklearn.ensemble** **import** RandomForestClassifier, ExtraTreesClassifier

**from** **sklearn** **import** svm

*#Defining the function for classification*

*#It includes KNN, Decision Tree, Random Forest, Extra\_Trees, SVM*

**def** models(X\_train,X\_test,y\_train,y\_test):

names = ["Nearest Neighbors", "Decision Tree", "Random Forest","ExtraTreesClassifier","SVM"]

classifiers = [KNeighborsClassifier(20),

DecisionTreeClassifier(max\_depth=2),

RandomForestClassifier(max\_depth=5, n\_estimators=100,max\_features=3),

ExtraTreesClassifier(max\_depth=5,n\_estimators=100,max\_features=3),

svm.SVC(kernel='linear')]

**for** name, clf **in** zip(names, classifiers):

clf.fit(X\_train, y\_train)

score = clf.score(X\_test, y\_test)

**print**(name+' Score:',score)

prediction1 = clf.predict(X\_test)

np.set\_printoptions(precision=2)

**return** 1

*#Splitting the dataset into train and test*

**from** **sklearn.cross\_validation** **import** train\_test\_split

y = v2\_data['class']

X = v2\_data.iloc[:,:6]

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X,y,test\_size=0.3)

y\_train=np.array(y\_train).reshape(-1,1)

*#Training and testing the model*

models(X\_train,X\_test,y\_train,y\_test)

('Nearest Neighbors Score:', 0.8064516129032258)

('Decision Tree Score:', 0.7311827956989247)

('Random Forest Score:', 0.7634408602150538)

('ExtraTreesClassifier Score:', 0.7849462365591398)

('SVM Score:', 0.8279569892473119)

*#Checking feature importance using ID3 algorithm used in decision tree*

*#It takes entropy into account rather than eigen vectors*

*#Hence its computationally cheaper than PCA*

feat\_test=DecisionTreeClassifier(min\_samples\_split=0.1)

feat\_test.fit(X\_train, y\_train)

**print** "**\n\n**Important Features:**\n**",feat\_test.feature\_importances\_

Important Features:

[0.02 0.06 0. 0.16 0.19 0.57]

*#Taking the top 50% important features*

X1 = v2\_data.iloc[:,3:6]

X1\_train,X1\_test,y1\_train,y1\_test = train\_test\_split(X1,y,test\_size=0.3)

y1\_train=np.array(y1\_train).reshape(-1,1)

*#Training the new dataset*

*#Comment: The new accuracy is clearly better.*

models(X1\_train,X1\_test,y1\_train,y1\_test)

('Nearest Neighbors Score:', 0.8172043010752689)

('Decision Tree Score:', 0.7311827956989247)

('Random Forest Score:', 0.8064516129032258)

('ExtraTreesClassifier Score:', 0.7741935483870968)

('SVM Score:', 0.8172043010752689)

*#Training for three classes: NO, DH, SP*

y1 = v\_data['class']

X1 = v\_data.iloc[:,:6]

X1\_train,X1\_test,y1\_train,y1\_test = train\_test\_split(X1,y1,test\_size=0.3)

y1\_train=np.array(y1\_train).reshape(-1,1)

*#Training and testing the model*

models(X\_train,X\_test,y\_train,y\_test)

('Nearest Neighbors Score:', 0.8064516129032258)

('Decision Tree Score:', 0.7311827956989247)

('Random Forest Score:', 0.7526881720430108)

('ExtraTreesClassifier Score:', 0.7849462365591398)

('SVM Score:', 0.8279569892473119)

*#Checking feature importance using ID3 algorithm used in decision tree*

*#It takes entropy into account rather than eigen vectors*

*#Hence its computationally cheaper than PCA*

feat\_test=DecisionTreeClassifier(min\_samples\_split=0.1)

feat\_test.fit(X\_train, y\_train)

**print** "**\n\n**Important Features:**\n**",feat\_test.feature\_importances\_

Important Features:

[0.02 0.06 0. 0.16 0.19 0.57]

*#Taking the top 50% important features*

X1 = v\_data.iloc[:,3:6]

X1\_train,X1\_test,y1\_train,y1\_test = train\_test\_split(X1,y,test\_size=0.3)

y1\_train=np.array(y1\_train).reshape(-1,1)

*#Training the new dataset*

*#Comment: The new accuracy is clearly better.*

models(X1\_train,X1\_test,y1\_train,y1\_test)

('Nearest Neighbors Score:', 0.8817204301075269)

('Decision Tree Score:', 0.7956989247311828)

('Random Forest Score:', 0.8494623655913979)

('ExtraTreesClassifier Score:', 0.8494623655913979)

('SVM Score:', 0.8387096774193549)

*#Taking the best model for this dataset and hyper tuning its parameters*

**from** **sklearn.model\_selection** **import** cross\_val\_score

myList = list(range(1,50))

*# subsetting just the odd ones*

neighbors = list(filter(**lambda** x: x % 2 != 0, myList))

*# empty list that will hold cv scores*

cv\_scores = []

*# perform 10-fold cross validation*

**for** k **in** neighbors:

knn = KNeighborsClassifier(n\_neighbors=k)

scores = cross\_val\_score(knn, X1\_test, y1\_test, cv=10, scoring='accuracy')

cv\_scores.append(scores.mean())

*# changing to misclassification error*

MSE = [1 - x **for** x **in** cv\_scores]

*# determining best k*

optimal\_k = neighbors[MSE.index(min(MSE))]

**print**("The optimal number of neighbors is:"+str(optimal\_k))

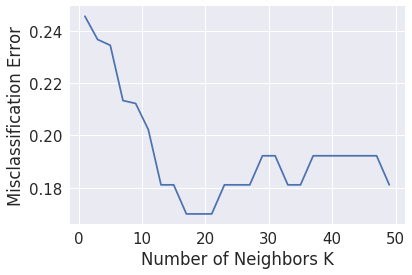
*# plot misclassification error vs k*

plt.plot(neighbors, MSE)

plt.xlabel('Number of Neighbors K')

plt.ylabel('Misclassification Error')

plt.show()



*# Showing the best accuracy for this dataset*

*# using the best parameters for KNN model*

knn = KNeighborsClassifier(n\_neighbors=17)

knn.fit(X1\_train, y1\_train)

score = knn.score(X1\_test, y1\_test)

**print**('Optimized KNN Score:',score)

('Optimized KNN Score:', 0.8924731182795699)

**Some Extra Visualization**

|  |  |
| --- | --- |
| **D:\VIT\#6 Win Sem 2018\Project\Vertebral Column\logic_regression.png** | **D:\VIT\#6 Win Sem 2018\Project\Vertebral Column\cluster.png** |
| **D:\VIT\#6 Win Sem 2018\Project\Vertebral Column\tree1.png** | |

**Predictions**

I used different types of machine learning and ensemble models and achieved the following results.

**Conclusion**

I have visualized the data using python and did some pre-processing on it based on the output of the visualization like removal of outliers (presence of angle >360). I have used multiple machine learning models that cover most of the techniques like bagging, boosting and ensemble models. KNN model was working best for this dataset hence I did some parameter hyper tuning and achieved better accuracy of 89.24%

I have created a model to predict vertebral column abnormality with an accuracy of about 90% using K-nearest neighbour.