In [1]:
```
import numpy as np
# Load data set and code labels as 0 = ’NO’, 1 = ’DH’, 2 = ’SL’
labels = [b'NO', b'DH', b'SL']
data = np.loadtxt('spine-data.txt', converters={6: lambda s: labels.index(s)})
```

In [2]:
```
X, y = data[:,:6], data[:, 6]
```

In [3]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size = 250, test
```

In [5]:
```
## Computes squared Euclidean distance between two vectors.
def l2_dist(x,y):
    return np.sum(np.square(x-y))
```

In [6]:
```
def l1_dist(x,y):
    return np.sum(np.abs(x-y))
```

In [7]:
```
## Takes a vector x and returns the index of its nearest neighbor in X_train
def find_NN(x, dist="l2"):
    if(dist=="l2"):
        # Compute distances from x to every row in train_data
        distances = [l2_dist(x,X_train[i,]) for i in range(len(y_train))]
    else:
        distances = [l1_dist(x,X_train[i,]) for i in range(len(y_train))]
    return np.argmin(distances)

## Takes a vector x and returns the class of its nearest neighbor in X_train
def NN_classifier(x, distance="l2"):
    # Get the index of the the nearest neighbor
    if(distance=="l2"):
        index = find_NN(x, "l2")
    else:
        index = find_NN(x, "l1")
    # Return its class
    return y_train[index]
```

2 a) What error rates do you get on the test set for each of the two distance functions?

In [8]:
```
## Predict on each test data point with l2 distance
test_predictions_l2 = [NN_classifier(X_test[i,],"l2") for i in range(len(y_test)

## Predict on each test data point with l1 distance
test_predictions_l1 = [NN_classifier(X_test[i,],"l1") for i in range(len(y_test)
```

In [9]:
```
## Compute the error with l2 distance
err_positions = np.not_equal(test_predictions_l2, y_test)
error = float(np.sum(err_positions))/len(y_test)

print("Error of nearest neighbor classifier with l2 distance: ", error)
```

```
Error of nearest neighbor classifier with l2 distance:  0.23333333333333334
```

In [10]:
```python
## Compute the error with l1 distance
err_positions = np.not_equal(test_predictions_l1, y_test)
error = float(np.sum(err_positions))/len(y_test)

print("Error of nearest neighbor classifier with l1 distance: ", error)
```

```
Error of nearest neighbor classifier with l1 distance:  0.21666666666666667
```

2 b) For each of the two distance functions, give the confusion matrix of the NN classifier.

In [11]:
```python
import numpy as np

#initializing 10x10 matrix with just zeroes.
dimensions = (3, 3)
confusion_matrix_l2 = np.zeros(dimensions)
#Looping through original test labels and test predictions to create confusion m
for a, b in zip(y_test, test_predictions_l2):
    confusion_matrix_l2[int(a)][int(b)] = confusion_matrix_l2[int(a)][int(b)] +
    #print(str(a)+" "+str(b))

#printing the 10x10 confusion matrix
print("Confusion matrix with l2 distance:")
confusion_matrix_l2
```

```
Confusion matrix with l2 distance:
```

Out[11]:
```
array([[12.,  1.,  3.],
       [ 9.,  9.,  0.],
       [ 1.,  0., 25.]])
```

For l2 distance :

- label 0 i.e NO (Normal) was misclassified 4 times
- label 1 i.e DH (herniated disk) was misclassified 9 times
- label 2 i.e SL (spondilolysthesis) was misclassified 1 time

In [12]:
```python
import numpy as np

#initializing 10x10 matrix with just zeroes.
dimensions = (3, 3)
confusion_matrix_l1 = np.zeros(dimensions)
#Looping through original test labels and test predictions to create confusion m
for a, b in zip(y_test, test_predictions_l1):
    confusion_matrix_l1[int(a)][int(b)] = confusion_matrix_l1[int(a)][int(b)] +
    #print(str(a)+" "+str(b))

#printing the 10x10 confusion matrix
print("Confusion matrix with l1 distance:")
confusion_matrix_l1
```

```
Confusion matrix with l1 distance:
```

Out[12]:
```
array([[14.,  0.,  2.],
       [ 9.,  9.,  0.],
       [ 1.,  1., 24.]])
```

For l1 distance :

- label 0 i.e NO (Normal) was misclassified 2 times
- label 1 i.e DH (herniated disk) was misclassified 9 times
- label 2 i.e SL (spondilolysthesis) was misclassified 2 times

In [ ]:

```

```