In [17]:

```
import sklearn
import numpy as np
import operator
from tqdm import tqdm
import pandas as pd
```

In [18]:

```
class KNeighborsClassifier():

    def __init__(self, n_neighbors=5): # this is the constructor
        self.neighbors = n_neighbors

    def fit():
        pass

    def predict():
        pass

    def euclidian_dist(self, point_1, point_2): # a function to calculate the euclidian distance
        dist = 0.0
        for i in range(len(point_1) - 1): # -1 because the last element is the class
            dist += pow(point_1[i] - point_2[i], 2) #using the pow function to calculate the power of a number as the
        return np.sqrt(dist)

    def calc_distances(self, data, new_point):
        distances = []
        neighbors = []
        for i in data:
            distances.append((i, self.euclidian_dist(new_point, i))) #appending the distance to the list
        distances.sort(key=operator.itemgetter(1)) #sorting the list by the second element of the tuple
        for i in range(self.neighbors): #getting the first k elements of the list
            neighbors.append(distances[i][0]) #appending the first k elements of the list to the neighbors list
        return neighbors

    def find_majority(self, neighbors, train_X, train_y): #a function to find the majority class
        iter_y = []
        for i in neighbors:
            iter_y.append(train_y[np.where(train_X == i)[0][0]]) #getting the index of the element in the train_X lis
        return max(iter_y)

    def fit(self, train_X, train_y):
        set_of_classes = set(train_y) #getting the set of classes
        self.classes = 0; #initializing the number of classes
        for i in tqdm(set_of_classes): #iterating through the set of classes
            self.classes += 1
        self.X = train_X
        self.y = train_y
        self.data_len = len(train_X) #getting the length of the data

    def predict(self, test_y):
        y_pred = []
        neighbors = []
        for i in tqdm(test_y): #iterating through the test data
            neighbors = self.calc_distances(self.X, i)  #getting the neighbors using the calc_distances function
            y_pred.append(self.find_majority(neighbors, self.X, self.y)) #getting the majority class using the find_
        return y_pred
```

Praneetha - CB.EN.U4AIE21147

In [3]:

```
data = pd.read_csv('/home/kalyan/gitrepo/alma-mater/Sem3/PML/echocardiogram.csv',sep=',')
data = data.dropna()
#missing value treatment
data = data.dropna()
data1 = data
#remove name column
data1 = data1.drop('name',axis=1)
data1.head()
np.random.seed(1234)
index = np.random.choice(np.arange(data1.shape[0]), size=int(data1.shape[0]*0.5))
train = data1.iloc[index]
test = data1.iloc[-index]
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data1.iloc[:, :-1].values
y = data1.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)

clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
100%|██████████| 2/2 [00:00<00:00, 26214.40it/s]
```

```
0.7096774193548387
```

Kalyana Sundaram - CB.EN.U4AIE21120

In [4]:

```
data = pd.read_csv('/home/kalyan/gitrepo/alma-mater/Sem3/PML/Dropout_Academic Success - Sheet1.csv',sep=',')
#missing value treatment
data = data.dropna()

#target column is what we want to predict
target = data['Target']
target

#we assign 1 for Graduate and 0 for Dropout
target = target.replace('Graduate',1)
target = target.replace('Dropout',0)

#model building
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
(2212, 36) (2212, 36) (2212,) (2212,)
```

```
100%|██████████| 3/3 [00:00<00:00, 58798.65it/s]
```

```
0.4914104882459313
```

Sarvesh Shashikumar - CB.EN.U4AIE21163

In [115]:

```python
import cv2
#import paths
from imutils import paths
import os
import numpy as np
def createImageFeatures(image, size=(32, 32)):
    # resize the image
    image = cv2.resize(image, size)

    # flatten the image
    pixel_list = image.flatten()
    return pixel_list

print("Reading all images")
image_paths = list(paths.list_images("/home/kalyan/DataSets/Dogs&Cats/train"))
raw_images = []
labels = []

#take randomly 100 images of cats and dogs
np.random.seed(42)
image_paths = np.random.choice(image_paths, size=(100), replace=False)

# loop over the input images
for (i, image_path) in enumerate(image_paths):
    image = cv2.imread(image_path)
    label = image_path.split(os.path.sep)[-1].split(".")[0]
    # extract raw pixel intensity "features
    pixels = createImageFeatures(image)
    raw_images.append(pixels)
    labels.append(label)

print("Number of images: {}".format(len(raw_images)))
raw_images = np.array(raw_images)
labels = np.array(labels)
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(raw_images, labels, test_size=0.8)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
Reading all images
Number of images: 100
(20, 3072) (80, 3072) (20,) (80,)

100%|██████████| 2/2 [00:00<00:00, 31300.78it/s]
  0%|          | 0/80 [00:00<?, ?it/s]/home/kalyan/miniconda3/lib/python3.7/site-packages/ipykernel_launcher.py:15:
RuntimeWarning: overflow encountered in ubyte_scalars
  from ipykernel import kernelapp as app
100%|██████████| 80/80 [00:13<00:00,  5.83it/s]

0.4875
```

Subikksha - CB.EN.U4AIE21167

In [137]:

```
data = pd.read_csv('/home/kalyan/gitrepo/alma-mater/Sem3/PML/Disease.csv',sep=',')
#print all classes in prognosis
classes = (data['prognosis'].unique())

class_dict = {}
for i in range(len(classes)):
    class_dict[classes[i]] = i

data['prognosis'] = data['prognosis'].map(class_dict)

#drop Unnamed: 133 column
data = data.drop('Unnamed: 133',axis=1)
data

#given symptoms predict the probable disease
symptoms = data.columns[:-1]
symptoms

#model building
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
(984, 132) (3936, 132) (984,) (3936,)

100%|██████████| 41/41 [00:00<00:00, 327555.17it/s]
100%|██████████| 3936/3936 [03:50<00:00, 17.07it/s]

0.025152439024390245
```

Kaushik Jonnada - CB.EN.U4AIE21122

In [150]:

```
#loading data
data = pd.read_csv('/home/kalyan/gitrepo/alma-mater/Sem3/PML/kr-vs-kp.data',sep=',')
data.head()

#missing value treatment
data = data.dropna()
data

#won = 1, nowin = 0
data['won'] = data['won'].replace('won',1)
data['won'] = data['won'].replace('nowin',0)

#settings all f to 1 and t to 0
data = data.replace('f',1)
data = data.replace('t',0)

#changing values with l to 1 and g to 0
data = data.replace('l',1)
data = data.replace('g',0)

#changing values with n to 1 and b to 0 and w to 2
data = data.replace('n',1)
data = data.replace('b',0)
data = data.replace('w',2)

#model building
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.8)

print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

clf = KNeighborsClassifier(n_neighbors=5)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test, y_pred))
```

```
(639, 36) (2556, 36) (639,) (2556,)

100%|██████████| 2/2 [00:00<00:00, 44150.57it/s]
100%|██████████| 2556/2556 [00:30<00:00, 83.03it/s]

0.4769170579029734
```