In [3]:

```python
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import csv
from collections import namedtuple
from scipy import stats, io
from scipy.special import expit
from sklearn.feature_extraction import DictVectorizer
```

In [5]:

```python
class Data(object):
    def __init__(self, name):
        if name != 'spam' and name != 'titanic':
            raise ValueError('Incorrect Data')
        self.name = name
        self.path = 'HW5_codes/Q2_decision_tree/datasets/'

        self.data = None
        self.csvTrain = None
        self.csvTest = None
        self.featNames = None
        self.dim = None

        self.trOrig = None
        self.trLabelsOrig = None
        self.train = None
        self.test = None
        self.trLabels = None
        self.val = None
        self.valLabels = None

        self.load_data()
        self.split(.2)

    def load_data(self):
        if self.name == 'spam':
            self.data = io.loadmat(self.path + 'spam-dataset/spam_data.mat')
            self.trOrig = self.data['training_data']
            self.trLabelsOrig = self.data['training_labels'].reshape(-1)
            self.test = self.data['test_data']
            self.featNames = [
            "pain", "private", "bank", "money", "drug", "spam", "prescription",
            "creative", "height", "featured", "differ", "width", "other",
            "energy", "business", "message", "volumes", "revision", "path",
            "meter", "memo", "planning", "pleased", "record", "out",
            "semicolon", "dollar", "sharp", "exclamation", "parenthesis",
            "square_bracket", "ampersand"]
            #self.featNames =
```

```python
        else:
            self.csvTr = pd.read_csv(self.path + 'titanic/titanic_training.csv')
            self.csvTest = pd.read_csv(self.path + 'titanic/titanic_testing_data
.csv')
            self.cleanData()
            self.trLabelsOrig = np.array(self.csvTr.survived)
            self.csvTr = self.csvTr.drop('survived', 1)

            dv = DictVectorizer(sparse = False)
            trainDict = self.csvTr.T.to_dict().values()
            testDict = self.csvTest.T.to_dict().values()
            train = dv.fit_transform(trainDict)
            test = dv.transform(testDict)
            self.trOrig, self.test = train, test
            self.featNames = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
"10"]

    def cleanData(self):
        df = self.csvTr[~self.csvTr['survived'].isna()]
        medianAge = np.median(df[df['age'].notna()]['age'])
        df['age'].fillna(medianAge, inplace = True)
        medianFare = np.median(df[df['fare'].notna()]['fare'])
        df['fare'].fillna(medianFare, inplace = True)
        df = df.drop(['cabin','ticket'], axis = 1)

        self.csvTr = df
        df.to_csv('HW5_codes/Q2_decision_tree/datasets/titanic/titanic_training_
cleaned.csv')


    def split(self,valProp):
        """The valProp is the proportion of the total data that should be valida
tion data."""
        np.random.seed(42)
        totalLen = len(self.trOrig)
        trainSize = totalLen - round(totalLen*valProp)
        randIdx = np.random.permutation(totalLen)
        self.train = self.trOrig[randIdx][:trainSize]
        self.trLabels = self.trLabelsOrig[randIdx][:trainSize]
        self.val = self.trOrig[randIdx][trainSize:]
        self.valLabels = self.trLabelsOrig[randIdx][trainSize:]
        self.dim = self.train.shape

class Node(object):
    def __init__(self,data,label,names,nodeType=None,splitFeature=None,splitThre
sh=None,printSplit=False):
        self.data = data
        self.label = label
        self.leftChild = None
        self.rightChild = None
        self.nodeType = nodeType
        self.splitThresh = splitThresh
        self.splitFeature = splitFeature
```

```
        self.splitValue = None

        self.names = names
        self.split = printSplit


    def traverse(self, row):
        if self.splitFeature > len(row)-1:
            self.splitFeature = 0
        if row[self.splitFeature] <= self.splitThresh:
            self.splitValue = "{} <= {}".format(self.names[self.splitFeature], self.splitThresh)
            if self.split:
                split = "{} <= {}".format(self.names[self.splitFeature], self.splitThresh)
                print(split)
            if self.leftChild.nodeType == 'Leaf':
                return self.getProb()
            return self.leftChild.traverse(row)
        else:
            self.splitValue ="{} > {}".format(self.names[self.splitFeature], self.splitThresh)
            if self.split:
                split = "{} > {}".format(self.names[self.splitFeature], self.splitThresh)
                print(split)
            if self.rightChild.nodeType == 'Leaf':
                return self.getProb()
            return self.rightChild.traverse(row)

    def getProb(self):
        total = len(self.label)
        ones = np.count_nonzero(self.label)
        zeros = total-ones
        prob = (total-zeros)/total
        if prob < .5:
            if self.split:
                print("Answer: Spam")
            return prob
        else:
            if self.split:
                print("Answer: Ham")
            return prob

class DecisionTree(object):
    def __init__(self, maxDepth, minObs, names, printSplit = False):
        self.maxDepth = maxDepth
        self.minObs = minObs
        self.tree = None #will be of type Node
        self.pred = None
        self.split = printSplit
        self.names = names

    def entropy(self, probs):
        """
```

```
        """

        probs: {'0': (num, tot), '1': (num, tot)}
        """
        prob0, prob1 = probs['0'], probs['1']
        if prob0[1] == 0:
            return 0
        elif prob1[1] == 0:
            return 0
        else:
            p0 = prob0[0]/prob0[1]
            p1 = prob1[0]/prob1[1]
            return p0*(1-p0) + p1*(1-p1)

    def information_gain(self, parentEntropy, childProb):
        """
        parentEntropy: value for calculated entropy of the Parent Node
        childProb: (leftProb, rightProb)
        """
        left, right = childProb
        leftTotal = left['0'][1]
        rightTotal = right['0'][1]
        childTotal = leftTotal + rightTotal
        leftWeight = leftTotal * self.entropy(left)
        rightWeight = rightTotal * self.entropy(right)
        sumTotal = leftTotal + rightTotal
        if sumTotal == 0:
            return 0
        return parentEntropy - (leftWeight + rightWeight)/(sumTotal)

    def threshold_test(self, label, featData, featName, parentEntropy):
        """Tests all of the values in the feature data for the best threshold to
split on.
        Returns: the best infoScore given this feature and all of the tested thr
esholds (tH)
                [(feature, tH, info_gain)]
        """
        uniqVals = np.unique(featData)

        infoScores = []
        for tH in uniqVals:
            L_data = label[featData <= tH]
            R_data = label[featData > tH]

            L1_total = np.count_nonzero(L_data)
            L0_total = L_data.shape[0] - L1_total
            R1_total = np.count_nonzero(R_data)
            R0_total = R_data.shape[0] - R1_total

            L_total = L0_total + L1_total
            R_total = R0_total + R1_total

            LProb, RProb = {}, {}
            LProb['0'], LProb['1'] = (L0_total, L_total), (L1_total, L_total)
```

```python
            RProb['0'], RProb['1'] = (R0_total, R_total), (R1_total, R_total)


            childProb = (LProb, RProb)
            info_gain = self.information_gain(parentEntropy, childProb)
            infoScores += [(featName, tH, info_gain)]
        best_score = max(infoScores, key=lambda x: x[2])
        return [best_score]

    def segmenter(self, node):
        """Finds the best feature to split on which maximizes information gain
        returns: (leftNode, rightNode)
        """
        parentProb = {}
        parentProb['0'] = (np.count_nonzero(node.label), len(node.label))
        parentProb['1'] = (len(node.label) - np.count_nonzero(node.label), len(n
ode.label))
        parent_entropy = self.entropy(parentProb)

        infoScores = []
        for featNum in range(node.data.shape[1]):
            infoScores += self.threshold_test(node.label,node.data[:,featNum],fe
atNum,parent_entropy)
        gain_attr = max(infoScores, key=lambda x: x[2])

        featName, tH, info_gain = gain_attr

        index = node.data[:,featName] <= tH
        data_l,  label_l  = node.data[index],  node.label[index]
        data_r, label_r = node.data[~index], node.label[~index]

        leftChild  = Node(data_l,label_l,names = self.names,nodeType = 'Node',sp
litFeature=featName,splitThresh=tH)
        rightChild = Node(data_r,label_r,names = self.names,nodeType = 'Node',sp
litFeature=featName,splitThresh=tH)


        return leftChild, rightChild

    def train(self, data, labels, depth=0):
        """
        data - the data that we want to train (i.e. data.train)
        labels - the labels associated with training (i.e. data.trLabels)
        """
        classRemain = np.unique(labels)
        if len(classRemain) == 1:
            if classRemain == 0:
                return Node(data=None, label=0, names = self.names, nodeType='Le
af', printSplit=self.split)
            if classRemain == 1:
                return Node(data=None, label=1, names = self.names, nodeType='Le
af', printSplit=self.split)
        elif self.minObs >= data.shape[0]:
            return Node(data=None, label=None, names = self.names, nodeType='Lea
```

```
f', printSplit=self.split)

        elif self.maxDepth == depth:
            return Node(data=None, label=None, names = self.names, nodeType='Lea
f', printSplit=self.split)

        else:
            tree = Node(data, labels, names = self.names, nodeType = "Node", pri
ntSplit=self.split)
            l_child, r_child = self.segmenter(tree)
            tree.splitFeature = r_child.splitFeature
            tree.splitThresh = r_child.splitThresh

            tree.leftChild = self.train(l_child.data, l_child.label, depth+1)
            tree.rightChild = self.train(r_child.data, r_child.label, depth+1)
        return tree

    def fit(self, data, labels):
        self.tree = self.train(data, labels)

    def prob(self, data):
        return np.array([self.tree.traverse(row) for row in data])

    def predict(self, data):
        probs = self.prob(data)
        self.pred = (probs > .5).astype(np.int)

    def accuracy(self,labels):
        return np.sum(self.pred.reshape(-1) == labels.reshape(-1))/len(self.pred
)

class RandomForest(object):
    def __init__(self, maxDepth, minObs, numTrees, names):
        self.maxDepth = maxDepth
        self.minObs = minObs
        self.numTrees = numTrees
        self.names = names
        self.pred = None
        self.DT = []

    def fit(self, data, labels):
        totFeat = data.shape[1]
        for tree in range(self.numTrees):
            randFeat = np.random.choice(totFeat,totFeat,replace = False)
            newData = data[:,randFeat]
            dT = DecisionTree(self.maxDepth, self.minObs, names = self.names)
            dT.fit(newData,labels)
            self.DT += [(randFeat, dt)]

    def prob(self, data):
        probLst = []
        for tree in self.DT:
            randfeat, dT = tree
            newData = data[:,randfeat]
```

```
            preds = dT.prob(newData)

            probLst += [preds]
        meanPreds = np.mean(np.array(probLst).T, axis = 1)
        return meanPreds

    def predict(self, data):
        probs = self.prob(data)
        self.pred = (probs > .5).astype(np.int)

    def accuracy(self,labels):
        return np.sum(self.pred.reshape(-1) == labels.reshape(-1))/len(self.pred
)
```

In [14]:

```
data = Data('titanic')
dt = DecisionTree(maxDepth=20, minObs = 5, names = data.featNames, printSplit =
False)
dt.fit(data.train, data.trLabels)
dt.predict(data.val)
dt.accuracy(data.valLabels)
```

Out[14]:

```
0.735
```

In [15]:

```
d = Data('titanic')
rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
rf.fit(d.train, d.trLabels)
rf.predict(d.val)
rf.accuracy(d.valLabels)
```

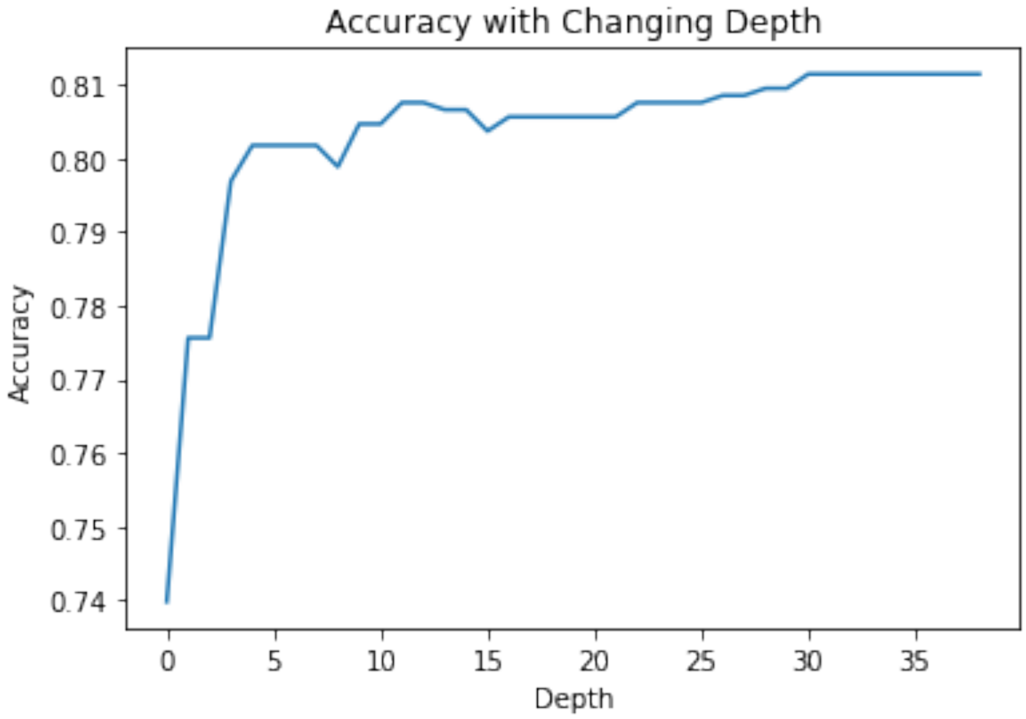
```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:175: RuntimeWarning: invalid value e
ncountered in less_equal
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site
-packages/ipykernel_launcher.py:176: RuntimeWarning: invalid value e
ncountered in greater
```

Out[15]:

```
0.465
```

In [78]:

```
#2.5.3 Train decision trees with 80/20 validation split. Plot depth from 1 to 40 on Validation accuracies
accuracy = []
for depth in range(1,40):
    data = Data('spam')
    dt = DecisionTree(maxDepth=depth, minObs=5, names=data.featNames)
    dt.fit(data.train, data.trLabels)
    dt.predict(data.val)
    acc = dt.accuracy(data.valLabels)
    accuracy.append(acc)
plt.plot(accuracy, label = "Validation Error")
plt.xlabel("Depth")
plt.ylabel("Accuracy")
plt.title("Accuracy with Changing Depth");
```



In [26]:

```
#2.5.2 Report the splits for the a data point.
data = Data('spam')
dt = DecisionTree(maxDepth=20, minObs = 5, names = data.featNames, printSplit = False)
dt.fit(data.train, data.trLabels)
dt.predict(data.train)
dt.accuracy(data.trLabels)
```

Out[26]:

```
0.8037699371677138
```

In [ ]:

```
#Figure out the hyperparameters: Change 'spam' to 'titanic' to check
maxDepth = np.arange(5,40,1)
minObs = np.arange(3,20,1)
for i in maxDepth:
    for j in minObs:
        data = Data('spam')
        dt = DecisionTree(maxDepth=i, minObs = j, names = data.featNames)
        dt.fit(data.train, data.trLabels)
        dt.predict(data.val)
        acc = dt.accuracy(data.valLabels)
        #print("maxDepth: " + str(i) + ', minDepth: ' + str(j) + ', acc: ' + str
(acc))
```

In [21]:

```
#Function to print a Decision Tree
data = Data('titanic')
dt = DecisionTree(maxDepth=3, minObs = 5, names = data.featNames, printSplit = F
alse)
dt.fit(data.train, data.trLabels )
def printTree(node, name, depth = 0):
    #which is node
    ret = "\t"*depth+name+str(node.splitFeature)+" Feat, "+str(node.splitThresh)
+" Val"+"\n\n"
    for child in [node.leftChild, node.rightChild]:
        if child.nodeType == "Leaf":
            continue
        if child == node.leftChild:
            newName = 'LChild: '
        else:
            newName = 'RChild: '
        ret += printTree(child,newName,depth+1)
    return ret
print(printTree(dt.tree, 'Root: ', 0))
```

```
Root: 8 Feat, 0.0 Val

        LChild: 0 Feat, 11.0 Val

                LChild: 10 Feat, 1.0 Val

                RChild: 7 Feat, 1.0 Val

        RChild: 7 Feat, 2.0 Val

                LChild: 5 Feat, 26.0 Val

                RChild: 5 Feat, 23.25 Val
```

In [30]:

```
data = Data('spam')
dt = DecisionTree(maxDepth=14, minObs = 10, names = data.featNames, printSplit =
False)
dt.fit(data.train, data.trLabels)
dt.predict(data.test)
y_pred = dt.pred.reshape(-1)
def results_to_csv(y_test):
    y_test = y_test.astype(int)
    df = pd.DataFrame({'Category': y_test})
    df.index += 1  # Ensures that the index starts at 1.
    df.to_csv('submission.csv', index_label='Id')

results_to_csv(y_pred)
```

In [ ]:

```
```

Matthew Brennan
Cs189 - hw5

Hw5 - Write-Up

“I certify that all solutions are entirely in my own words and that I have not looked at another student’s solutions. I have given credit to all external sources I consulted.”

Q1 Write-Up :

1. In order to extract the frames, I relied on opencv and therefore imported cv2. Using this code:

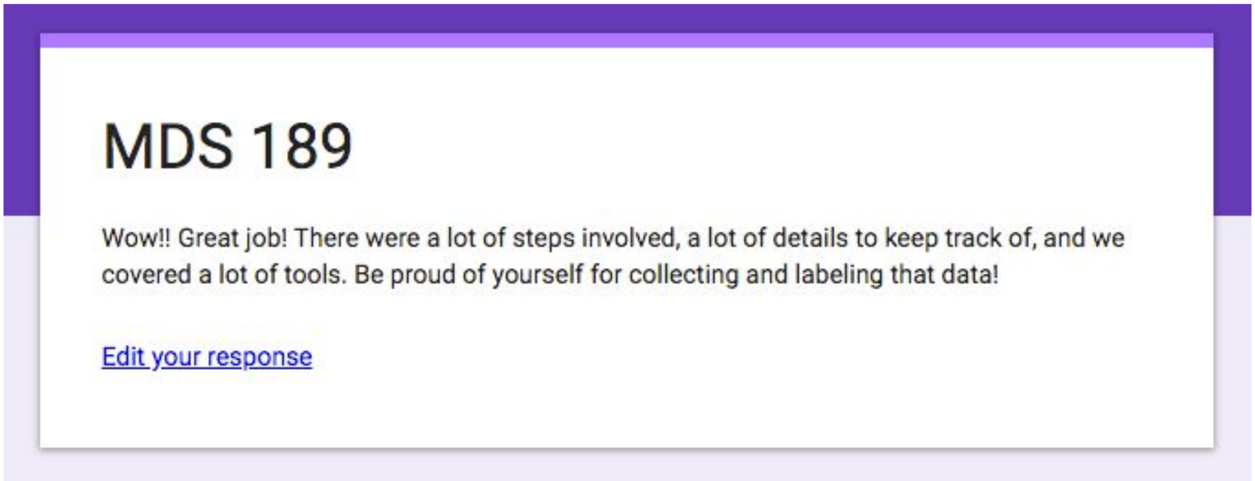
```python
videos = ['reach', 'squat', 'inline',
          'lunge', 'hamstrings', 'stretch',
          'deadbug', 'pushup']
for video in videos:
    vidcap = cv2.VideoCapture("HW5_codes/Q1_mds189_resources/Videos/" + video + ".MOV")
    success, image = vidcap.read()
    count = 1
    success = True
    while success:
        cv2.imwrite('HW5_codes/Q1_mds189_resources/' + video + '/' + '%04d.jpg' % count, image)
        success, image = vidcap.read()
        count += 1
        print("Read a new frame:", success)
```

   I was able to ensure that the frames were being read into the files that I created for each specific folder.

2. Output from running checkData.py: All test cases passed

```
Passed!
Checking key frames......
Passed!
Checking keypoints label file......
Passed!
Checking video label files......
Checking label file matthewbrennan/labels/deadbug.json
Checking label file matthewbrennan/labels/hamstrings.json
Checking label file matthewbrennan/labels/inline.json
Checking label file matthewbrennan/labels/lunge.json
Checking label file matthewbrennan/labels/pushup.json
Checking label file matthewbrennan/labels/reach.json
Checking label file matthewbrennan/labels/squat.json
Checking label file matthewbrennan/labels/stretch.json
{'squat': (1080, 1920, 3), 'reach': (1080, 1920, 3), 'pushup': (1080, 1920, 3), 'inline': (
1080, 1920, 3), 'hamstrings': (1080, 1920, 3), 'stretch': (1080, 1920, 3), 'lunge': (1080,
1920, 3), 'deadbug': (1080, 1920, 3)}
{'squat': (1080, 1920, 3), 'reach': (1080, 1920, 3), 'pushup': (1080, 1920, 3), 'inline': (
1080, 1920, 3), 'hamstrings': (1080, 1920, 3), 'stretch': (1080, 1920, 3), 'lunge': (1080,
1920, 3), 'deadbug': (1080, 1920, 3)}
Passed!
```

3. Google Form Screenshot:


MDS 189

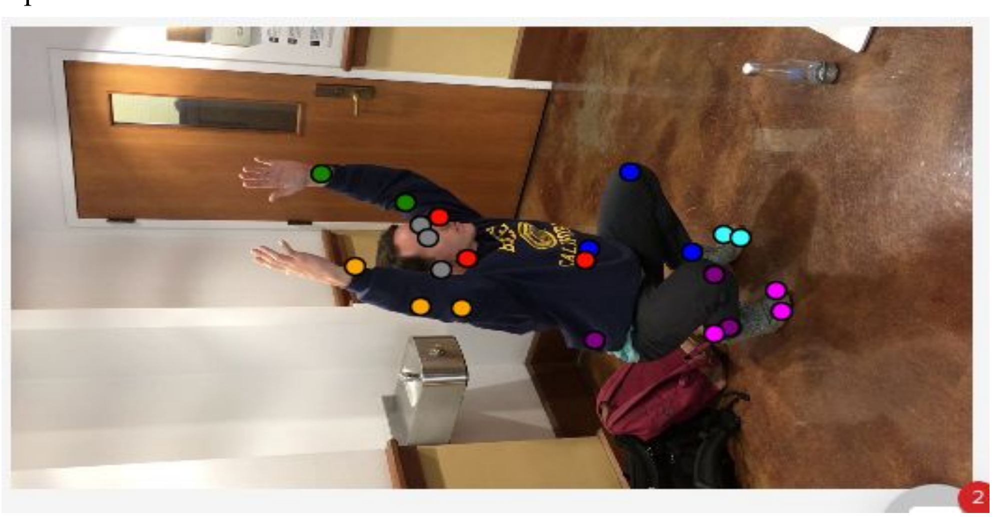
Wow!! Great job! There were a lot of steps involved, a lot of details to keep track of, and we covered a lot of tools. Be proud of yourself for collecting and labeling that data!

Edit your response

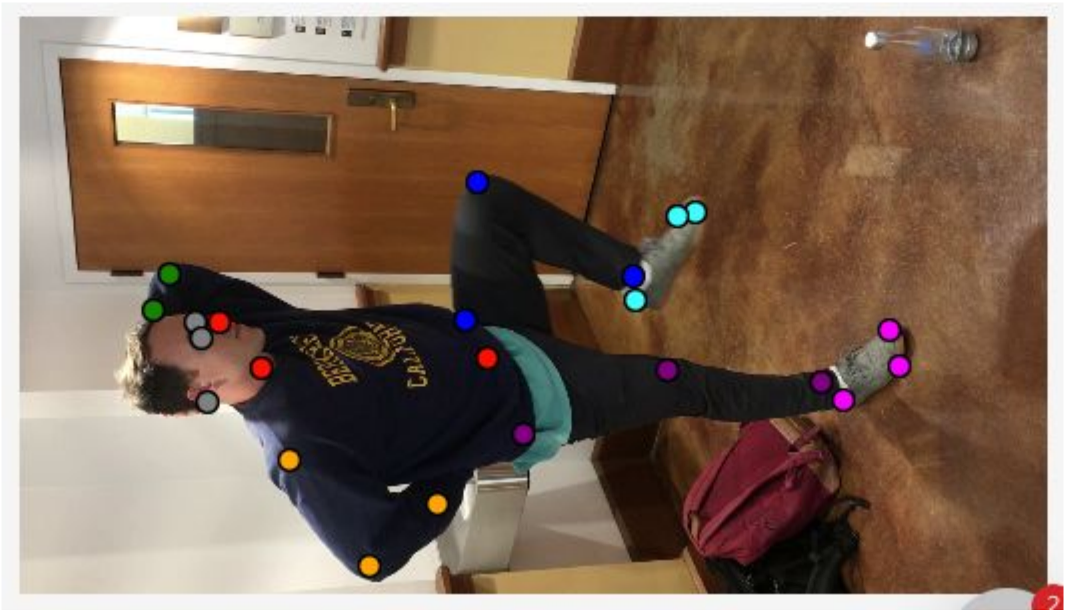
4. Screenshots from LabelBox:

Reach:

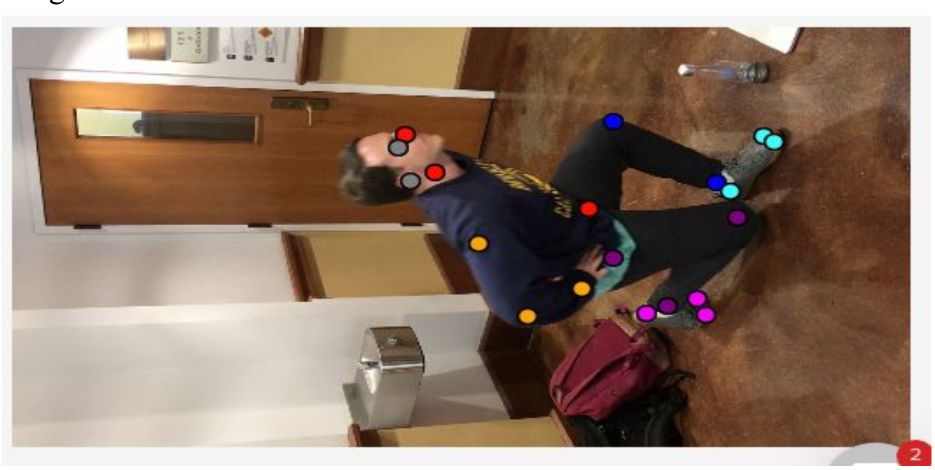
Squat:



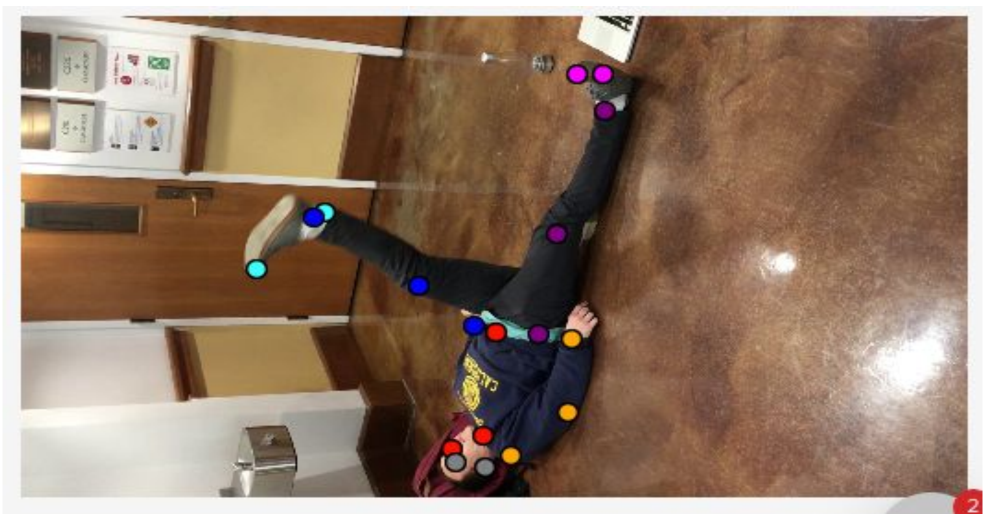
Inline:

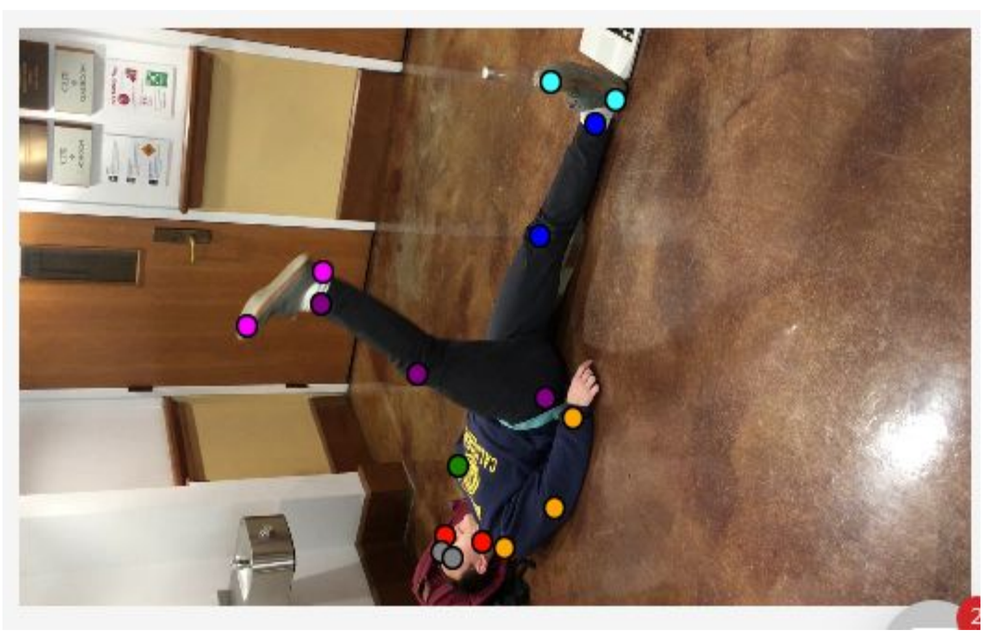


Lunge:

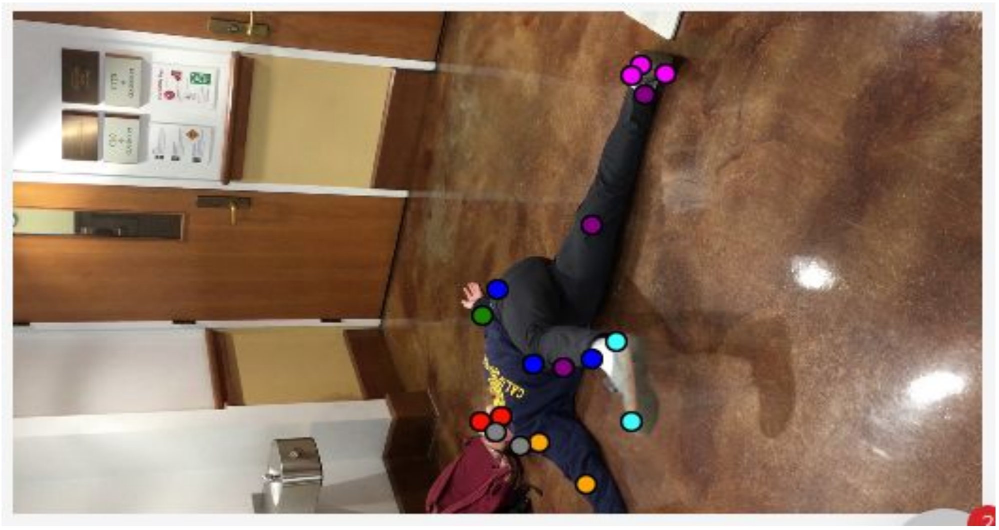


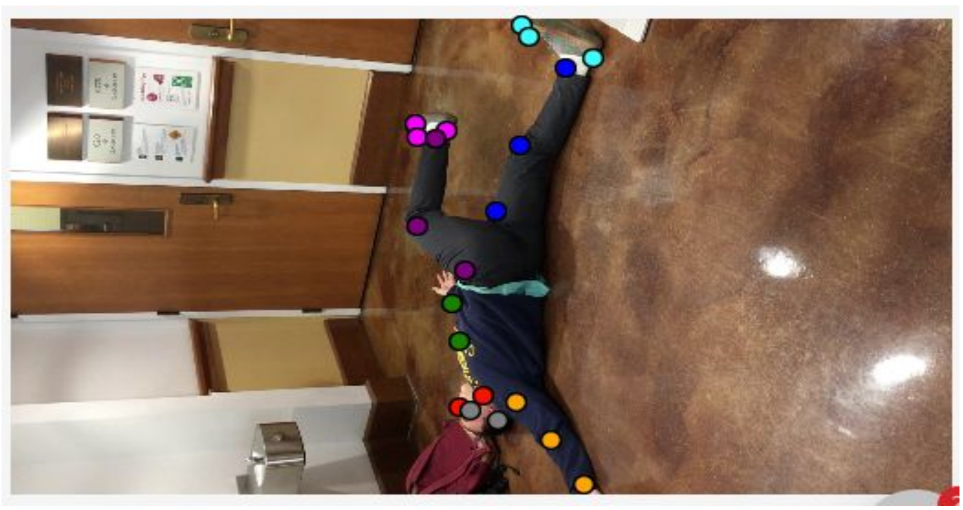
Hamstrings:



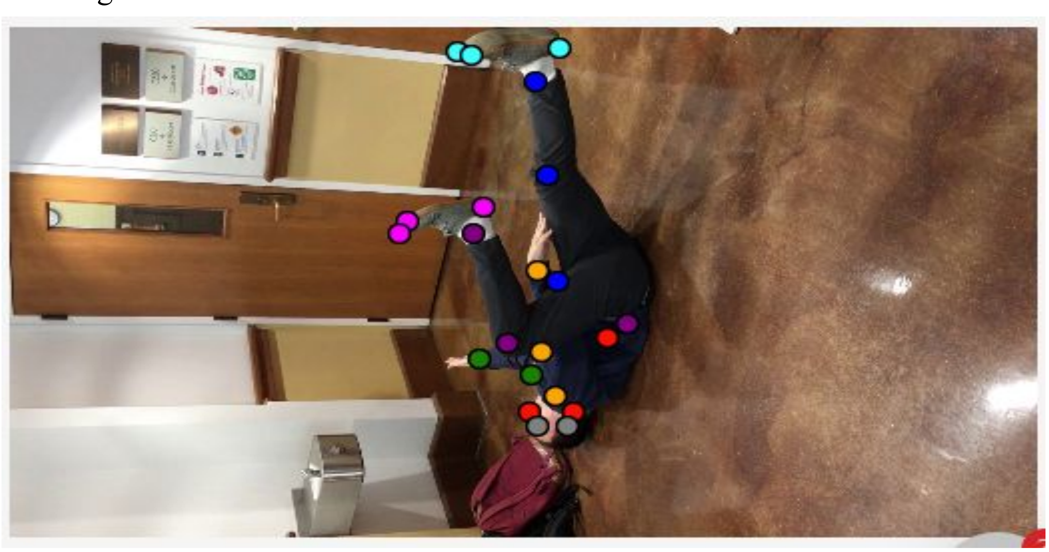


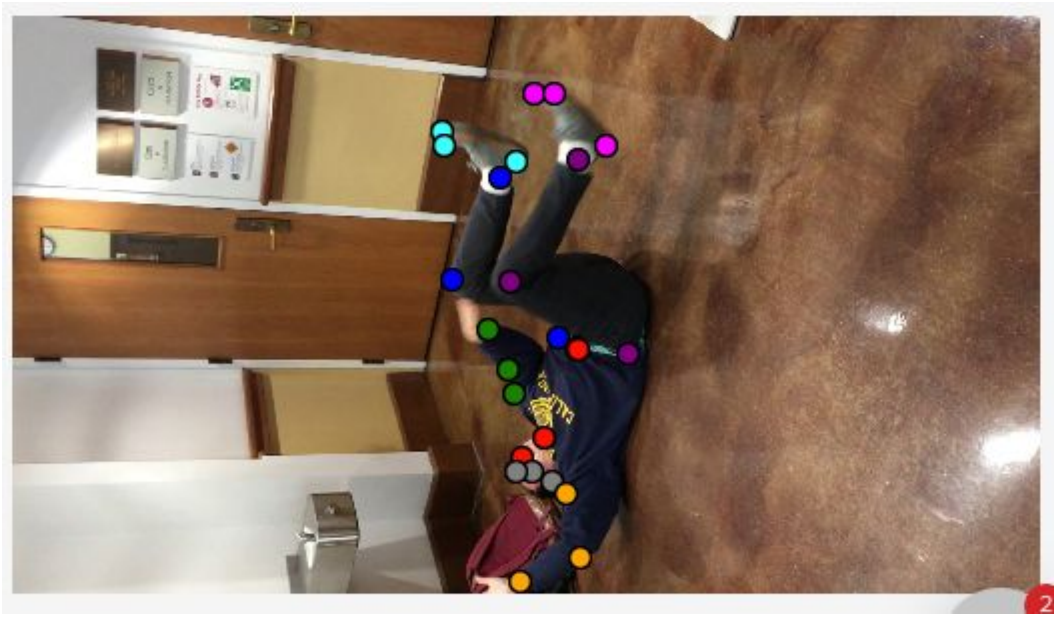
Stretch:



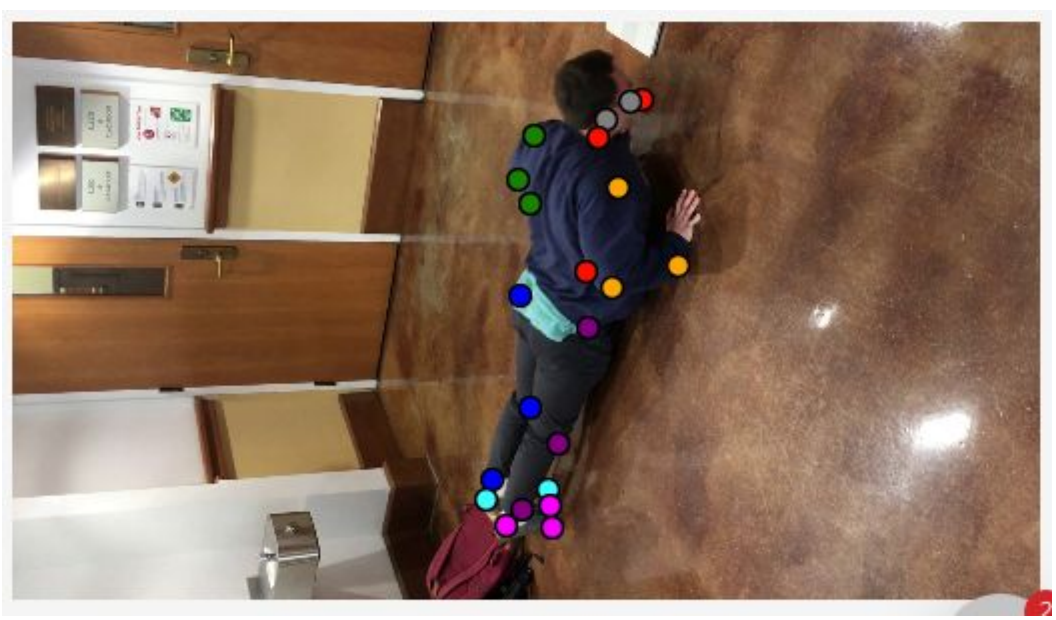


Deadbug:





Pushup:



2.1 Implement Decision Tree: Successfully implemented decision tree. Actual class is attached below with Code.

```
dt = DecisionTree(maxDepth=20, minObs = 5, printSplit = False)
data = Data('spam')
dt.fit(data.train, data.trLabels)
dt.predict(data.train)
dt.accuracy(data.trLabels)
```

```
0.8037699371677138
```

2.2 Implement Random Forests: Successfully implemented Random Forests. Actual class is attached below with Code.

```
: d = Data('spam')
  rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
  rf.fit(d.train, d.trLabels)
  rf.predict(d.val)
  rf.accuracy(d.valLabels)
```

```
: 0.7437137330754352
```

2.3 Describe Implementation Details:

1) In order to deal with categorical features, I opted to use sklearn's DictVectorizer in order to treat all of the categorical data as numbers by assigning an integer value to each of the feature types. Further, in order deal with missing values, the Data class has a method call cleanData() which allows for the missing values to in some columns to be replaced by the median value. Additionally, I found that getting rid of the features 'cabin' and 'ticket' to be rather successful on the Titanic dataset.

2) The stopping criterion that I implemented were based around a maxDepth, depending on how large the tree was meant to be. Therefore, one could change the maxDepth and thus treat this as a hyperparameter that I used Cross Validation in order to tune. Lastly, I added a minObs category, as well as an additional factor to tune on in order to potentially ensure that each tree gets to a certain height.

3) Implementing Random Forests derived mainly from the already created Decision Tree class as expected. The prior work of Random Forests, allowed there to be random features selected with a hyperparameter numFeat in order to switch up the order of the features as well as to mix up the rows. From the intuition on the lecture notes, I discerned that this was often useful in the situation where trees tend to utilize the same exact splits due to an ordering issue. For each of these newly created datasets, I trained a DecisionTree around this data with this order and features and then added this to a list of trees to compile the forest. Further, once a certain number of these random trees have

been created, we can average the probabilities for these trees in order to receive a a probability with full randomness.

4) The training was recursively called and depended on the depth of the problem in order to end the tree, and therefore was run relatively fast and speed of training never served as a real issue.

5) Nothing particularly cool!

2.4 Performance Evaluation:

Spam, Training, DecisionTree:

```
dt = DecisionTree(maxDepth=20, minObs = 5, printSplit = False)
data = Data('spam')
dt.fit(data.train, data.trLabels)
dt.predict(data.train)
dt.accuracy(data.trLabels)
```

```
0.8037699371677138
```

Spam, Training, RandomForests:

```
d = Data('spam')
rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
rf.fit(d.train, d.trLabels)
rf.predict(d.train)
rf.accuracy(d.trLabels)
```

```
0.7039632672788787
```

Spam, Validation, DecisionTree:

```
data = Data('spam')
dt = DecisionTree(maxDepth=20, minObs = 5, names = data.featNames, printSplit = False)
dt.fit(data.train, data.trLabels)
dt.predict(data.val)
dt.accuracy(data.valLabels)
```

```
0.8056092843326886
```

Spam, Validation, RandomForests:

```
d = Data('spam')
rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
rf.fit(d.train, d.trLabels)
rf.predict(d.val)
rf.accuracy(d.valLabels)
```

```
0.7437137330754352
```

Titanic, Training, DecisionTree:

```
data = Data('titanic')
dt = DecisionTree(maxDepth=20, minObs = 5, names = data.featNames, printSplit = False)
dt.fit(data.train, data.trLabels)
dt.predict(data.train)
dt.accuracy(data.trLabels)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/pandas/co
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/index
copy
  self._update_inplace(new_data)
```

```
0.851063829787234
```

Titanic, Training, RandomForests:

```
d = Data('titanic')
rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
rf.fit(d.train, d.trLabels)
rf.predict(d.train)
rf.accuracy(d.trLabels)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/
ng: invalid value encountered in less_equal
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages/
ng: invalid value encountered in greater
```

```
0.3667083854818523
```

Titanic, Validation, DecisionTree:

```
data = Data('titanic')
dt = DecisionTree(maxDepth=20, minObs = 5, names = data.featNames, printSplit = False)
dt.fit(data.train, data.trLabels)
dt.predict(data.val)
dt.accuracy(data.valLabels)
```

```
0.735
```

Titanic, Validation, RandomForests:

```
d = Data('titanic')
rf = RandomForest(maxDepth=15, minObs =5, numTrees = 50, names = d.featNames)
rf.fit(d.train, d.trLabels)
rf.predict(d.val)
rf.accuracy(d.valLabels)
```

```
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages
ng: invalid value encountered in less_equal
/Library/Frameworks/Python.framework/Versions/3.7/lib/python3.7/site-packages
ng: invalid value encountered in greater
```

```
0.465
```

Kaggle:

For Titanic Dataset: MaxDepth = 25, minObs = 18

Name	Submitted	Wait time	Execution time	Score
submission-13.csv	just now	0 seconds	0 seconds	0.83870

Complete

Jump to your position on the leaderboard

For Spam Dataset: MaxDepth = 14, minObs = 12

Name	Submitted	Wait time	Execution time	Score
submission-20.csv	just now	0 seconds	0 seconds	0.78713

Complete

Jump to your position on the leaderboard

2.5 Writeup Requirements for the Spam Dataset:

1) No additional feature implementations used.

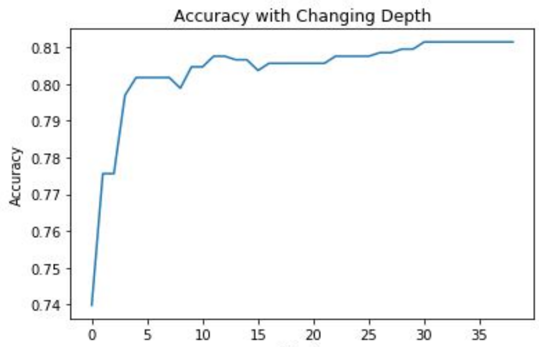
2) For the fifth datapoint of the Spam Data:

```
exclamation <= 0.0
meter <= 0.0
parenthesis > 0.0
private <= 0.0
pain <= 0.0
Answer: Spam
```

3) Generating an 80/20 Validation split and Graphing the validation from maxDepth 1 to 40: MinObs fixed at 5, and my validation split is always set to 80/20 split:

I found that as the depth increases there is an increase in the validation accuracy, yet that this increase is not represented entirely in the kaggle score. When I opted to trying other graphs with changing hyperparameters, the result always depicted a trend toward increasing validation accuracy with increases in depth.

```
#2.5.3 Train decision trees with 80/20 validation split. Plot depth from 1 to 40 on Validation accuracies
accuracy = []
for depth in range(1,40):
    data = Data('spam')
    dt = DecisionTree(maxDepth=depth, minObs=5, names=data.featNames)
    dt.fit(data.train, data.trLabels)
    dt.predict(data.val)
    acc = dt.accuracy(data.valLabels)
    accuracy.append(acc)
plt.plot(accuracy, label = "Validation Error")
plt.xlabel("Depth")
plt.ylabel("Accuracy")
plt.title("Accuracy with Changing Depth");
```



2.6 Writeup Requirements for the Titanic Dataset:

A simple tree that covers the characteristics of the tree where the Root and Children are labeled as well as the split Feature and the split Value.

```
data = Data('titanic')
dt = DecisionTree(maxDepth=3, minObs = 5, names = data.featNames, printSplit = False)
dt.fit(data.train, data.trLabels )
def printTree(node, name, depth = 0):
    #which is node
    ret = "\t"*depth+name+str(node.splitFeature)+" Feat, "+str(node.splitThresh)+" Val"+"\n\n"
    for child in [node.leftChild, node.rightChild]:
        if child.nodeType == "Leaf":
            continue
        if child == node.leftChild:
            newName = 'LChild: '
        else:
            newName = 'RChild: '
        ret += printTree(child,newName,depth+1)
    return ret
print(printTree(dt.tree, 'Root: ', 0))
```

```
Root: 8 Feat, 0.0 Val

        LChild: 0 Feat, 11.0 Val

                LChild: 10 Feat, 1.0 Val

                RChild: 7 Feat, 1.0 Val

        RChild: 7 Feat, 2.0 Val

                LChild: 5 Feat, 26.0 Val

                RChild: 5 Feat, 23.25 Val
```

README: To run the code that is attached, Simply run in Jupyter with the given code and it should work as expected.