General Questions [10 points]:

1. For each of the data sets was the decision tree algorithm able to memorize the training data (i.e., get 100% accuracy when used to classify the training data)? If not, can you give at least one reason that you think it was not able to?

Not all. According to the following tests, it's not 100% accuracy even test on the training data itself. There are many reasons. First, the data cannot be classified by a certain function (e.g. linear function). For example, there's no hyperlane to divide the data space, then we will not get 100% accuracy.

Also there are many records, we can't split due to only one attributes or no records for a split attributes. In those cases, we just use the majority rule, which might mismatch with the original class.

2. Could you use your system to, say, buy stocks or bet on horses? Why or why not?

Yes and no.

If you know all the possible parameters (including the trend of other people's involvement) that might affects the results, yes we can. Even we don't know all the parameters, decision systems can always provide us useful knowledge.

I say no because that in these cases, the training data and test data may not share same set of attributes due to the stochastic environment. If you test something is different with what you trained, it's probably lead you to a bad situation.

Balloon [30 points]:

1. Demonstrate your program on these small data sets. Draw out the trees you generated for the 4 balloon examples. Note that there are no testing or pruning files—we assume your tree will be perfect in these very simple examples.

Dataset: adult-stretch.data (act=4 or age=6)

Tree:

```
act
   4
    `-> 8
   5
   age
      6
       `-> 8
      7
       `-> 9
```

For example: act is the split attribute on value 4 and 5; and 4 led to leaf node which is 8, and 5 need further split; when act=5 then it split on attribute age; age split on value 6 and 7, resulting 8 and 9 separately. Following trees share same structures.

Dataset: adult+stretch.data (act=4 and age=6)

Tree:

```
act
  4
  age
    6
      `-> 8
    7
      `-> 9
  5
    `-> 9
```

Dataset: yellow-small.data (color=0 and size=3)

Tree:

```
color
 1
   `-> 9
 0
  size
    2
      `-> 9
    3
      `-> 8
```

Dataset: yellow-small+adult-stretch.data ((color=0&&size=3)or(act=4&&age=6))

Tree:

```
color
 1
  act
    4
    age
      6
        `-> 8
      7
        `-> 9
    5
      `-> 9
 0
  size
    2
    act
      4
      age
        6
          `-> 8
        7
          `-> 9
      5
        `-> 9
```

3
`-> 8

Hypothyroid and Messidor[30 + 30 points]:

1. What is the majority class accuracy for the test set? (i.e. what is the accuracy to just say "benign" always?)

majority class accuracy on the Hypothyroid test set:

601/632 = 0.9509493670886076

Linux scripts:

```
awk -F, 'BEGIN{sum=0}{if($1==1)sum+=1;}END{print sum}' hypothyroid-test.txt
31
awk -F, 'BEGIN{sum=0}{if($1==0)sum+=1;}END{print sum}' hypothyroid-test.txt
601
```

majority class accuracy on the Hypothyroid training set:

1811/1901= 0.952656496580747

majority class accuracy on Messidor test set: 0.5021645021645022

Linux scripts:

```
awk -F, 'BEGIN{sum=0}{if($20==1)sum+=1;}END{print sum}' messidor-test.txt
116
awk -F, 'BEGIN{sum=0}{if($20==0)sum+=1;}END{print sum}' messidor-test.txt
115
```

majority class accuracy on Messidor training set: 0.5455882352941176

Linux scripts:

```
awk -F, 'BEGIN{sum=0}{if($20==0)sum+=1;}END{print sum}' messidor-train.csv
319
awk -F, 'BEGIN{sum=0}{if($20==1)sum+=1;}END{print sum}' messidor-train.csv
371
```

(Honestly, I don't know the meaning of each attribute. And I also modify both the test and prune files to make them work)

2. How does you tree's test accuracy compare to the majority accuracy?

For Hypothyroid, testing accuracy on test data: 99.0506329114

For Hypothyroid, testing accuracy on training data: 99.3687532877

For messidor, testing accuracy on test data: 61.0692640693

For messidor, testing accuracy on training data: 72.9275362319

All of them are much better than the majority class accuracy.

Comparison:

Dataset	Majority	DT
Hypothyroid Test	0.9509493670886076	0.990506329114
Hypothyroid train	0.952656496580747	0.993687532877
Messidor test	0.5021645021645022	0.610692640693
Messidor train	0.5455882352941176	0.729275362319

3. What do the top 2 levels (the root and the level just below) look like (draw them)?

For Hypothyroid, testing accuracy on test data: 99.0506329114

```
[symptom16 < 15.000]
 [symptom16 < 6.500]
  [symptom24 < 50.000]
   [symptom24 < 49.000]
    [symptom18 < 0.200]
     [symptom1 < 0.000]
      [0.0]
      [symptom1 < 0.000]
       [0.0]
       [0.0]
     [0.0]
    [0.0]
   [symptom1 < -1.000]
    [0.0]
    [0.0]
  [symptom24 < 66.000]
   [symptom4 < 1.000]
    [1.0]
    [0.0]
   [symptom24 < 70.000]
    [0.0]
    [symptom1 < -1.000]
     [0.0]
     [0.0]
 [symptom24 < 65.000]
  [symptom6 < 1.000]
   [symptom8 < 1.000]
    [symptom20 < 63.000]
     [symptom7 < 1.000]
      [symptom22 < 1.120]
       [symptom1 < -1.000]
        [1.0]
        [1.0]
       [symptom22 < 1.130]
        [1.0]
        [symptom1 < -1.000]
         [1.0]
         [1.0]
      [1.0]
     [0.0]
```

```
      [1.0]
    [0.0]
  [symptom16 < 16.000]
    [1.0]
    [symptom2 < 24.000]
      [0.0]
      [symptom1 < -1.000]
        [0.0]
        [0.0]
```

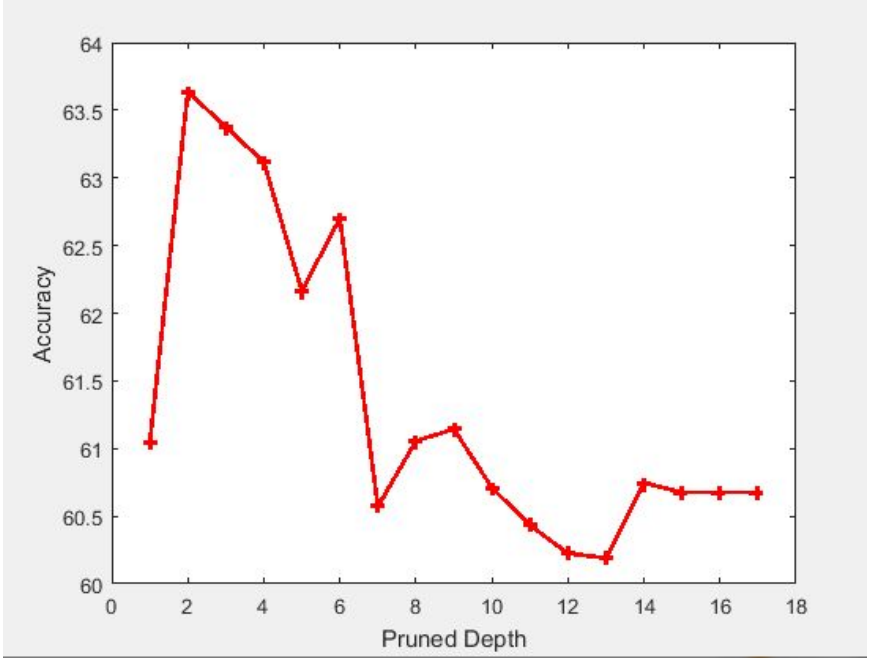
For messidor, the decision tree is:

```
        [a15 < 0.030]
        [a3 < 58.000]
          [a9 < 128.097]
            [a12 < 0.042]
              [a17 < 0.566]
                [1.0]
                [0.0]
              [a12 < 2.565]
                [0.0]
                [0.0]
            [a3 < 21.000]
              [a12 < 1.410]
                [1.0]
                [0.0]
              [a9 < 315.233]
                [1.0]
                [0.0]
          [a10 < 16.224]
            [a18 < 0.090]
              [a18 < 0.090]
                [1.0]
                [0.0]
              [a17 < 0.576]
                [1.0]
                [0.0]
            [a9 < 63.055]
              [a10 < 26.705]
                [1.0]
                [0.0]
              [a3 < 65.000]
                [0.0]
                [1.0]
        [a17 < 0.473]
          [0.0]
          [a9 < 88.279]
            [a10 < 32.798]
              [a3 < 58.000]
                [1.0]
                [1.0]
              [a9 < 77.676]
                [0.0]
                [1.0]
            [a1 < 1.000]
              [1.0]
              [1.0]
```

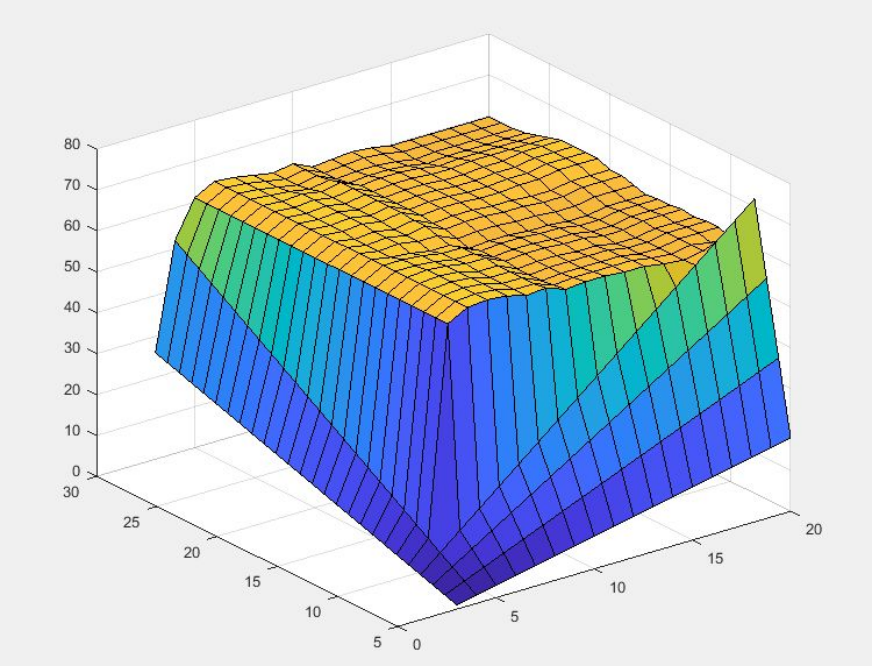
EXTRA CREDIT [20 points]

Use reduced-error pruning. This is a pruning criterion that replaces a subtree with a single leaf when that leaf (which classifies the instance as the majority class of

all instances seen at that leaf during pruning) represents a lower error rate than does the subtree. The pruning procedure thus has the following steps:
Grow the tree fully on the training data.
In a postorder recursive fashion traverse the decision tree, replacing a non-leaf node with a leaf node only when the error rate of the subtree node exceeds the error rate of the leaf node.
Measure the error rate on a separate set of instances (called the pruning data).
I tried different prune parameters, the results shown as the following figures. The decision tree with pruned depth 5, and if an attribute has less than 5 records I just use the majority class to prune, which generate the best result on the test set: 64.0693%



I also tried how the majority rules affect the results, the following figure shows the results. [0,20] means different the depth, [5,30] means when we stop to choose the majority class. Z is the accuracy. It seems that when to stop is not very important to the accuracy. More details please refer to the pruned result files.



EXTRA CREDIT [10 points]

Explore the effect of Bagging. Choose the data set for which you performed the WORST, and see if you can improve your results via bagging.

Here we sampling 5 times to get different training sets using a simplified bootstrap sampling. I use the vote machenism:

```
44      for sample in samples:
45          train_set = list(samples)
46          train_set.remove(sample)
47          train_set = sum(train_set, [])
48          test_set = list()
49          for row in sample:
50              row_copy = list(row)
51              test_set.append(row_copy)
52              row_copy[-1] = None
53          # each sample trained a decision tree
54          predicted = algorithm(train_set, test_set, *args)
55          actual = [row[-1] for row in sample]
56          acc= accuracy_metric(actual, predicted)
57          accuracy.append(acc)
58          votes.append(predicted)
59      win = list()
60      for idx in range(len(votes[0])):
61          tmp=[vote[idx] for vote in votes]
62          win.append(max(set(tmp), key=tmp.count)) # the max voted wins
```

Results for bagging :

Accuracy for different base decision trees:

[59.42028985507246,

66.66666666666666,

67.3913043478261,

63.76811594202898,

63.04347826086957]

Best Accuracy: 67.391%, not too much but is the best for now.

The best decision tree:

```
[a3 < 58.000]
  [a15 < 0.030]
    [a9 < 121.443]
      [a12 < 0.042]
        [a18 < 0.087]
          [1.0]
          [0.0]
        [a11 < 15.347]
          [0.0]
          [0.0]
      [a12 < 3.043]
        [a9 < 274.763]
          [1.0]
          [0.0]
        [a3 < 22.000]
          [0.0]
          [1.0]
    [a7 < 42.000]
      [a17 < 0.479]
        [0.0]
        [a7 < 22.000]
          [1.0]
          [1.0]
      [0.0]
  [a14 < 0.040]
    [a12 < 0.812]
      [a8 < 56.000]
        [a9 < 31.128]
          [1.0]
          [1.0]
        [a10 < 4.805]
          [0.0]
          [1.0]
      [a8 < 32.000]
        [1.0]
        [a10 < 10.949]
          [1.0]
          [0.0]
    [a3 < 62.000]
      [a16 < 0.027]
        [0.0]
        [1.0]
      [a13 < 6.621]
        [a1 < 1.000]
          [1.0]
          [1.0]
        [1.0]
```

EXTRA CREDIT [20 points]

Explore the effect of Boosting. Choose the data set for which you performed the WORST, and see if you can improve your results via implementing the AdaBoost ensemble algorithm.

Based on the previous implementations, we can just make some modification to achieve similar effect like AdaBoost.

Without learning the distribution of the samples, we can re-sampling the to train the decision tree. Although it's not exactly same as re-weighting, like the adaboost, it can also have the same effect.

So the simplest way to observe the effect boosting is just running bagging algorithm several times and each time with a different re-sampling method.