1. **Nearest neighbor:** Consider the set of training examples in the diagram below.

![Diagram of training examples](image)

(a) Draw the decision boundaries for the 1-NN algorithm assuming that we are using the standard Euclidean distance to compute nearest neighbors. A + indicates a positive example and a x indicates a negative example.

(b) How will point (8,1) be classified?

(c) How will point (8,8) be classified?

2. **Nearest neighbor and linear regression:** Exercise 2.6 from the text, part (a) only.

3. **Decision trees and linear regression:** Classic decision tree induction algorithms split real-valued data perpendicular to the axis using tests involving inequalities such as \( x_i > 10 \). Describe how to modify these algorithms to allow for oblique splits, being sure to cover the form of the tests used in decision nodes and the algorithm/metric used to choose the test at a node.
4. **Decision trees:** Consider the following decision tree:

```
   x1<25
    /  
   x2<15  x2<15
   /     /  
  x1<10 x1<5 x1<10 x1<5
  /     /  /     /  
 A     B  C     D  E     F
```

(a) Draw the decision boundaries defined by this tree. Each leaf of the tree is labeled with a letter. Write this letter in the corresponding region of the instance space.

(b) Give another decision tree that is syntactically different but defines the same decision boundaries. This demonstrates that the space of decision trees is syntactically redundant. Is this redundancy a statistical problem (i.e., does it affect the accuracy of learned trees)? Is it a computational problem (i.e., does it increase the computational complexity of finding an accurate tree)?

5. **Decision trees:** This problem will give you experience with c4.5, the most widely used decision tree algorithm. The code for c4.5 can be found at the following URL:

http://www.cs.umbc.edu/~oates/classes/04/ml/c4.5r8.tar

Retrieve the above file to a Linux machine. For the sake of concreteness, suppose you put it in ~/c4.5. Now issue the following commands at the shell prompt:

```
cd ~/c4.5
tar -xvf c4.5r8.tar
cd R8/Src
make
cc -o train-test train-test.c
```

You should now have an executable named c4.5 and one named train-test. The train-test program is used to randomly split a dataset into training and testing instances. If you run the program with no command line arguments you will get the following usage message:

**Usage:** train-test filename percent_train percent_test

You specify the name of a file containing all available instances, the percentage of those instances that are to be used for training and the percentage that are to be used for testing. Try the following commands:

```
cd ~/c4.5/R8/Data
~/c4.5/R8/Src/train-test VOTE 70 30
~/c4.5/R8/Src/c4.5 -f VOTE -u
```
Instances in the VOTE dataset describe how members of the U.S. congress voted on various bills. The task is to predict what political party the member belongs to (Democrat or Republican) based on the way they voted. The train-test command above creates a file named VOTE.data, which is used for building the decision tree, and a file named VOTE.test, which is used for testing the tree. There is also a file called VOTE.names which tells c4.5 about the dataset, i.e. the names of the attributes, the values they can take on, and the possible class labels. VOTE.data contains 70% of the instances in VOTE and VOTE.test contains 30% of the instances.

The c4.5 command above runs c4.5 on the VOTE data by using the -f flag to specify the file stem (the program will look for VOTE.data and VOTE.names) and using the -u flag to tell c4.5 that there is test data (the program will look for VOTE.test).

Please do each of the following:

- Run c4.5 on the VOTE data as described above (i.e. using a 70/30 split of the data). Examine the resulting decision tree and briefly describe in words the structure that it discovered.
- Run c4.5 on the vote data using 10%, 20%, 30%, 40%, 50%, 60% and 70% of the data for training. In each case use 30% of the data for testing. Because the random assignment of instances to the training and testing sets introduces variance, run c4.5 five time for each training set size. Plot the mean, minimum and maximum for each of the following quantities as a function of training set size: tree size before pruning, tree size after pruning, accuracy of the unpruned tree on the testing instances, and accuracy of the pruned tree on the testing instances. Describe briefly what the plots show and about the behavior of c4.5 and its pruning algorithm.
- Modify the routine named FormTree in build.c to select an attribute for splitting at random rather than using information gain. Look for a comment that says Add line of code to set BestAtt randomly here. That’s where you should make the modification. Construct plots of tree size before pruning, tree size after pruning, accuracy of the unpruned tree on the testing instances, and accuracy of the pruned tree on the testing instances just as before with the modified splitting algorithm. Describe how the change affected the performance of the algorithm.