

## 0. Basics

The general goal of machine learning = make accurate predictions about *unknown* data after being trained on *known* data.

There are two kinds of training: **supervised**, where the desired output is provided along with the input; and **unsupervised**, where the desired output is not provided. We will focus on **supervised** learning methods here.

Data comes in the form of examples, in the format:  $(x_1, \dots, x_n, y)$

Here,  $x_1, \dots, x_n$ , are also known as **features, inputs,** or **dimensions**, while  $y$  is the desired (or observed) output or class label. A feature is a descriptor or property used to characterize the input for learning. We call the space where feature values define the coordinate axes a **feature space**. The input vector for each example defines a point in feature space

Both the  $x$ 's and the  $y$ 's can be **discrete** (taking on values from, say,  $\{0, 1\}$  or some fixed set of label names or classes) or **continuous**.

In machine learning **training** we are given some (finite) set of  $(x_1, \dots, x_n, y)$  tuples. From this we output some learned classification or prediction function.

**Note** that K-Nearest Neighbors (KNN) and ID Trees are both **supervised, classification** learning algorithms

In machine learning **testing** we are given just  $(x_1, \dots, x_n)$  and the goal is to **predict**  $y$  with high accuracy.

**Training error** is the classification error measured using training data to test.

**Testing error** is classification error on data not seen in the training phase.

**Checking for over-fitting - Cross-validation:** split sample data into  $N$  subsets, use each subset as test set, the rest as training set; use average and standard deviation of performance on test sets to characterize prediction performance.

## 1. k-Nearest Neighbors

**Training** – Store all feature vectors in the training set, along with each class label.

**Prediction** – Given a query feature vector, find “nearest” stored feature vector and return the associated class.

$$\text{“Distance”} = \sqrt{w_1(v_{a1} - v_{b1})^2 + w_2(v_{a2} - v_{b2})^2 + \dots + w_n(v_{an} - v_{bn})^2}$$

$v_{a1}$  is the value of feature 1 in vector  $a$

$v_{b1}$  is the value of feature 1 in vector  $b$

...

$w_n$  is the weight for feature  $n$  (see below for some common metrics used for distance and other points about weighting)

1-NN: Given an unknown point, pick the closest 1 neighbor by some distance measure.

Class of the unknown is the 1-nearest neighbor's label.

$k$ -NN: Given an unknown, pick the  $k$  closest neighbors by some distance function.

Class of unknown is the **mode** of the  $k$ -nearest neighbor's labels.

$k$  is usually an odd number to facilitate tie breaking.

Normalization? To separate values clustered close together, divide by the standard deviation

Relevant features? All features are used; to find relevant ones, have to cross-validate, dropping features out.

What's the  $k$ ? Can find best value using cross-validation

Voting for vectors?  $k$ -Nearest Neighbors votes on class for query feature vector; reduces sensitivity to noise

$k$ -NN fixes a set of **decision boundaries** for whether a point is/is not in a given class. (We will see that other learning methods also fix decision boundaries).

### How to draw 1-NN decision boundaries

Decision boundaries are defined as lines on which it is **equally likely** for a data point to be in any of the classes

1. Examine the region where you think decision boundaries should occur.
2. Find oppositely labeled points (+/-) and connect them, forming a line.
3. Draw perpendicular bisectors of these lines. (Use a pencil)
4. Extend and join all bisectors. Erase extraneously extended lines.
5. Remember to **draw boundaries to the edge of the graph** and indicate it with arrows! (a very common mistake).
6. Your 1-NN boundaries generally should have sharp edges and corners (otherwise, you are doing something wrong or drawing boundaries for a higher order  $k$ -NN).

**Let's practice drawing  $k$ -NN boundaries. Turn to the end of the handout where we show you how the 'recipe' works; then we have a practice problem for you to try.**

Here are some standard distance metrics to use

<b>Euclidean Distance (common)</b>	$D(\vec{w}, \vec{v}) = \sqrt{\sum_i^n (w_i - v_i)^2}$
Manhattan Distance (Block distance) - Sum of distances in each dimension	$D(\vec{w}, \vec{v}) = \sum_i^n  w_i - v_i $
Hamming Distance - Sum of differences in each dimension	$D(\vec{w}, \vec{v}) = \sum_i^n I(w_i, v_i)$ $I(x, y) = 0$ if identical, 1 if different.
Cosine Similarity - Used in Text classification; words are dimensions; documents are vectors of words; vector component is 1 if word $i$ exists.	$D(\vec{w}, \vec{v}) = \frac{\vec{w} \cdot \vec{v}}{\ \vec{w}\  \ \vec{v}\ } = \cos \theta$

Note that it is also sometimes helpful to *transform* the data from one space to another. For example, if data are scattered in ring-like patterns of classes, then a transformation to polar coordinates typically helps. (Why?)

This is true of the practice problem we just did, as we will show in more detail below when using another learning method, ID trees.

### Nearest neighbors, optional: How to weigh dimensions differently

In Euclidean distance all dimensions are treated the same. But in practice not all dimensions are equally important or useful!

Example: Suppose we represent documents as vectors of words. Consider the task of classifying documents related to *Red Sox*. If all words are equal, then the word *the* weighs the same as the word *Sox*. But almost every English document contain the word *the*. But only sports related documents have the word *Sox*. So we want the *k*-NN distance metrics to weight **meaningful** words like *Sox* more than functional words like *the*.

For text classification, a weight scheme used to make some dimensions (words) more important than others is known as: TF-IDF

$$tf \cdot idf(w_i, d) = tf(w_i, d) \cdot idf(w_i)$$

$$tf(w_i, d) = \frac{\#(w_i) \in d}{|d|}$$

$$idf(w_i) = \log \frac{|D|}{\#d \in D \text{ with } w_i}$$

Here:

tf: Words that occur frequently should be weighed more.

idf: Words that occur in all the documents (functional-words like the, of etc) should be weighed less.

**Using this weighing scheme with a distance metric, knn would produce better (more relevant) classifications.**

**Another way to vary the importance of different dimensions is to use: Mahalanobis Distance**

$$D(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})S^{-1}(\vec{x} - \vec{y})}$$

**Here S is a covariance matrix. Dimensions that show more variance are weighted more heavily.**

## 2. Identification Trees (ID trees or decision trees)

Algorithm: Build a decision tree by **greedily** picking the “lowest disorder” feature tests. The best split for a set of data *minimizes* the average disorder (more precisely, we want the split that decreases the average disorder the most). We define these terms immediately below.

**Training – Divide** the feature space into boxes that have uniform labels. Split the space recursively along each axis to define a tree. (Note this forms a set of boundaries that ‘tile’ the plane in terms of perpendiculars.)

NOTE: This algorithm is greedy (local hill climbing) so it does **not** guarantee that the tree will have the minimum total disorder!

The notion of “disorder” is defined using **entropy,  $H$** .

We define the entropy (disorder), following Shannon’s definition, of a discrete random variable  $X$  that has the probability mass function  $p$ , as follows:

$$-\sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

So for example, suppose a drawer contains 3 red socks and 7 green socks. Then the entropy of this collection of socks in one drawer is (see the graph on the next page for a plot of this function where there are only two classes and one ‘bin’):

$$-3/10 \log_2 3/10 - 7/10 \log_2 7/10 = -0.3(-1.7369) - 0.7(-0.5145) = +0.902570$$

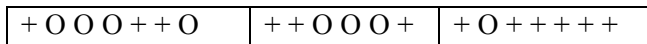
Note that the disorder here is at a maximum when the two kinds of socks are equally distributed; and a minimum when either color is absent (uniform color), so the probability of one possibility is 0, and  $-p \log_2 p$  of the other color is  $1 \times 0 = 0$ , so  $H$  is 0.

For ID trees, we will need to find the *weighted average* of disorder across a *set of classes or ‘bins’*. The *average entropy or disorder for a split* = Entropy for *each* region (bin) times the fraction of the total data points that are in that region (bin) – a weighted average of the disorder, weighted by the # of data points in each class or bin.

$$\text{Average disorder} = \sum_b \left( \frac{n_b}{n_t} \right) \times \left( \sum_c - \frac{n_{bc}}{n_b} \log_2 \left( \frac{n_{bc}}{n_b} \right) \right)$$

$n_b$  is the total number of samples in branch  $b$   
 $n_t$  is the total number of samples in all branches  
 $n_{bc}$  is the total of samples in branch  $b$  of class  $c$

**Let’s practice calculating this. A simple example with 3 bins (classes), and 2 possibilities, + or O:**



We calculate the entropy  $H$  in each of the three classes:

Class 1: 3 +, 4 O, 8 total, so + probability is  $3/8 = 0.375$ , so from our 2<sup>nd</sup> graph:  $H_1 = 0.95$

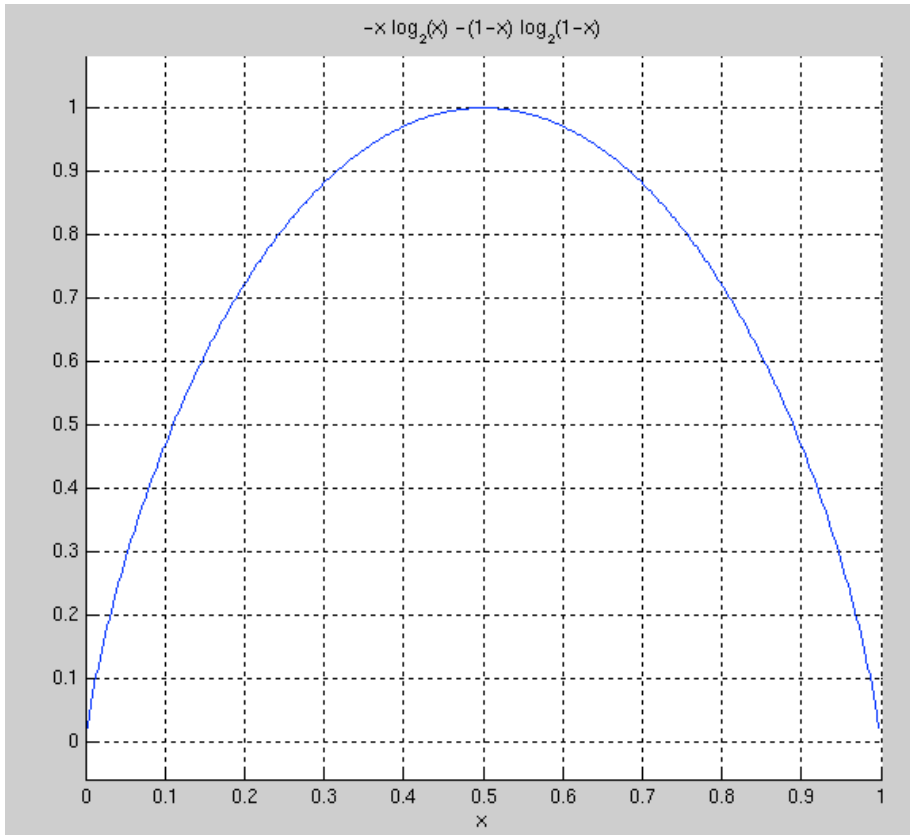
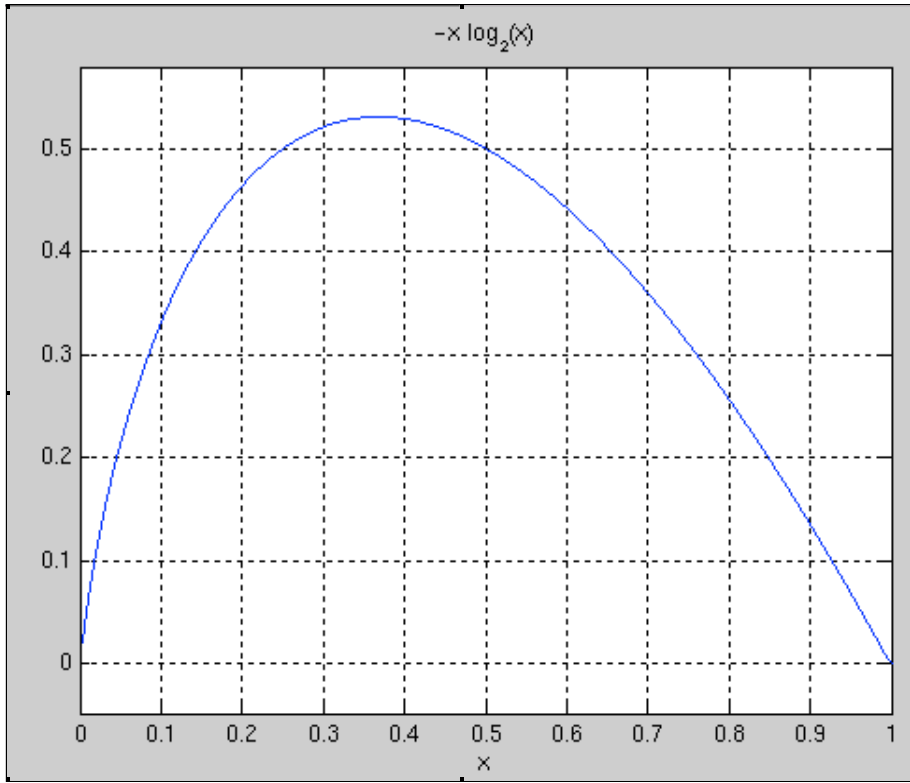
Class 2: 3 +, 2 O, 6 total, so + probability is  $3/6 = 1/2$ , so  $H_2 = 1/2 + 1/2 = 1.0$

Class 3: 7 +, 1 O, 8 total, so + probability is  $7/8 = 0.875$ , so  $H_3 = 0.54$

Now we compute the *weighted average* of these three  $H$  values. There are 8+6+8 objects in all, or 22, so:

$$8/22(H_1) + 6/22(H_2) + 8/22(H_3) = 0.36(0.95) + 0.18(1) + 0.36(0.54) = 0.34 + 0.18 + 0.19 = 0.71$$

This is the *average disorder* of this particular split into 3 classes. This is the number used to ‘drive’ the algorithm, which attempts to find the split that achieves the *lowest* average disorder.



See also the table of binary entropy values a few pages later on.

### Example formulas.

The disorder equation for a test with two branches, left and right, ( $l, r$ ), with each branch having 2 (binary) classes or bins.

Let  $a$  = count of class 1 on the left side;  $b$  = count of class 2 on the left side;

Let  $c$  = count of class 1 on the right side;  $d$  = count of class 2 on the right side

$a + b = l$   $c + d = r$ ;  $r+l = T$

$$\text{Disorder} = \frac{l}{T} \left( \left[ -\frac{a}{l} \log_2 \frac{a}{l} \right] + \left[ -\frac{b}{l} \log_2 \frac{b}{l} \right] \right) + \frac{r}{T} \left( \left[ -\frac{c}{r} \log_2 \frac{c}{r} \right] + \left[ -\frac{d}{r} \log_2 \frac{d}{r} \right] \right)$$

For a test with 3 branches, and 2 binary class outputs (this is the formula for the example we explicitly did earlier):

$$\text{Disorder} = \frac{b_1}{T} H\left(\frac{a}{b_1}\right) + \frac{b_2}{T} H\left(\frac{c}{b_2}\right) + \frac{b_3}{T} H\left(\frac{e}{b_3}\right)$$

$a$  = count of class 1 on branch 1       $b$  = count of class 2 on branch 1

$c$  = count of class 1 on branch 2       $d$  = count of class 2 on branch 2

$e$  = count of class 1 on branch 3       $f$  = count of class 2 in branch 3

$a+b = b_1$      $c + d = b_2$      $e + f = b_3$

### Homogeneous Partitioning Trick

A time-saving heuristic shortcut to picking the lowest disorder test.

1. Pick tests that break the space into a *homogeneous portion* and a *non-homogeneous* portion
2. Pick the test that partitions out the **largest** homogeneous portion; that test will most likely have the lowest disorder.

**Caution!** when the homogeneous portions are **about the same size**, you should compute the full disorder value. This is where this shortcut might break down!

**ID trees and Prediction** – Test features of a query feature vector according to the identification tree generated during training, return the class at the leaf of the tree.

Relevant features? Irrelevant features are ignored because have large disorders.

Whose Razor? Occam's: The world is inherently simple. Choose the smallest consistent tree.

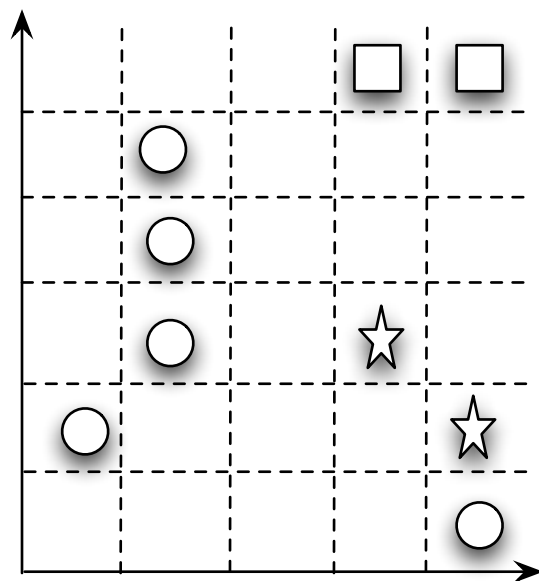
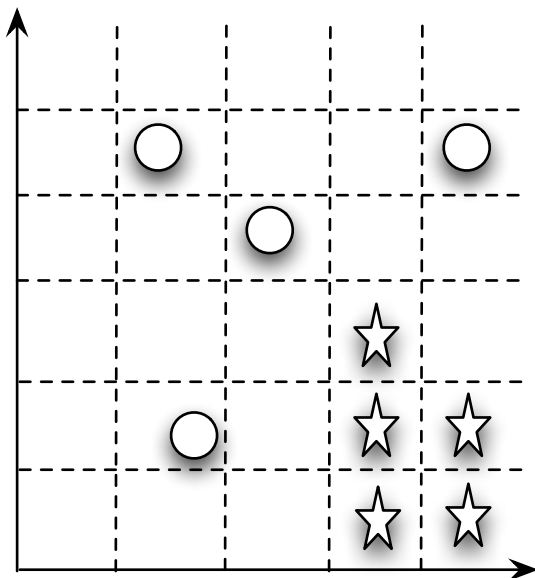
Why greedy? Finding the simplest tree is computationally intractable; so we use a greedy search using minimum average disorder as a heuristic.

**Table of common Binary Entropy values**

Note: because  $H(x)$  is a symmetric function, i.e.  $H(1/3) = H(2/3)$ , fractions  $> 1/2$  are omitted.

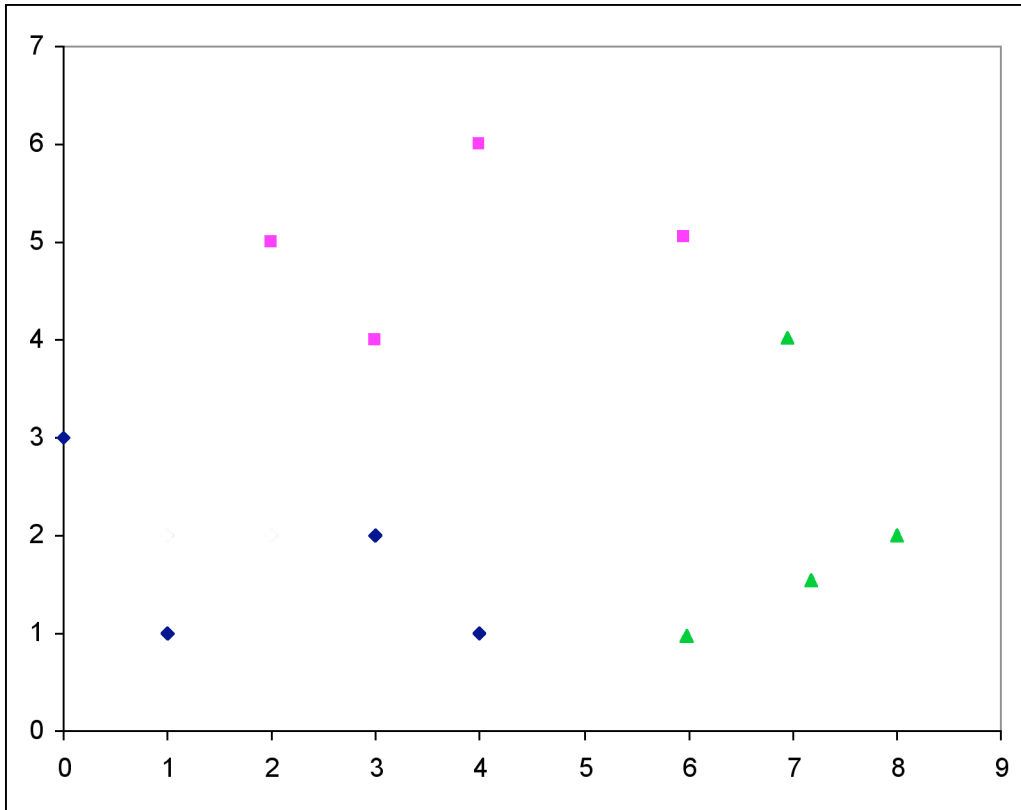
/3 to /9				/10 to /13			
numerator	denominator	fraction	$H(\text{fraction})$	numerator	denominator	fraction	$H(\text{fraction})$
1	3	0.33	0.92	1	10	0.10	0.47
2	3	0.67	0.92	2	10	0.20	0.72
1	4	0.25	0.81	3	10	0.30	0.88
2	4	0.50	1.00	4	10	0.40	0.97
1	5	0.20	0.72	1	11	0.09	0.44
2	5	0.40	0.97	2	11	0.18	0.68
3	5	0.60	0.97	3	11	0.27	0.85
1	6	0.17	0.65	4	11	0.36	0.95
2	6	0.33	0.92	5	11	0.45	0.99
3	6	0.50	1.00	1	12	0.08	0.41
1	7	0.14	0.59	2	12	0.17	0.65
2	7	0.29	0.86	3	12	0.25	0.81
3	7	0.43	0.99	5	12	0.42	0.98
1	8	0.13	0.54	1	13	0.08	0.39
2	8	0.25	0.81	2	13	0.15	0.62
3	8	0.38	0.95	3	13	0.23	0.78
4	8	0.50	1.00	4	13	0.31	0.89
1	9	0.11	0.50	5	13	0.38	0.96
2	9	0.22	0.76	6	13	0.46	1.00
3	9	0.33	0.92				
4	9	0.44	0.99				

Try some sample 'cuts' in these two figures....which is the best single cut(s) in each? Why? And the next cut?

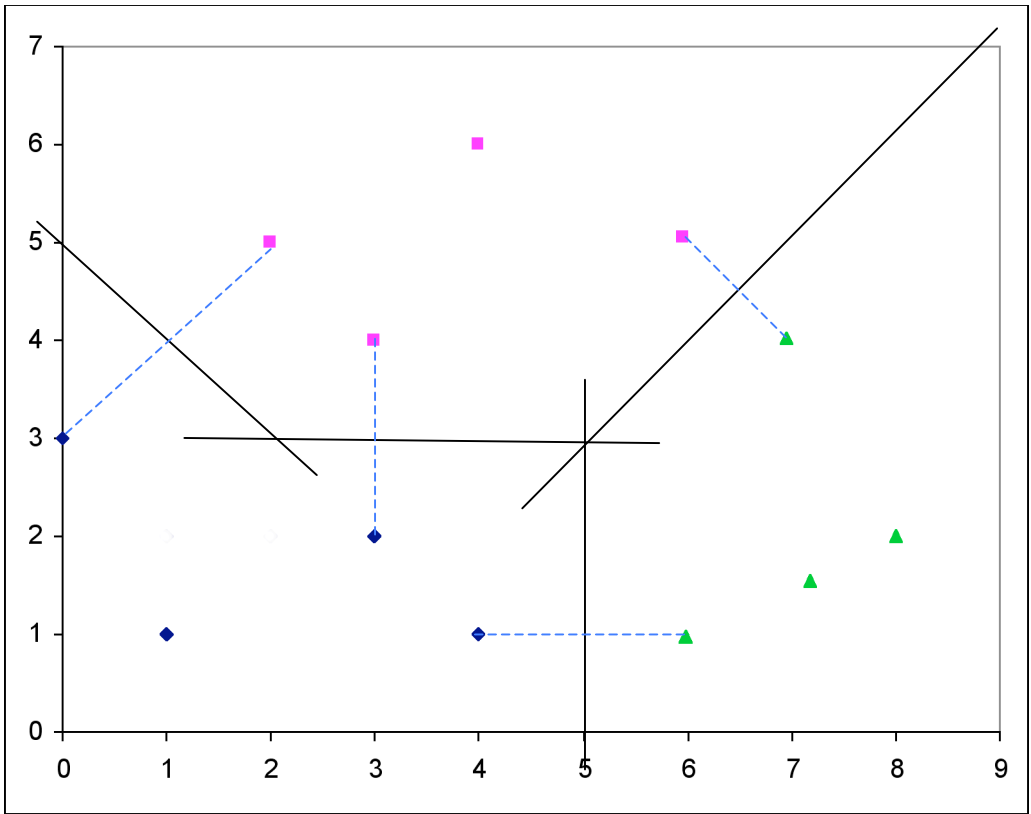


## 6.034 Recitation October 20: Nearest Neighbors, Drawing decision boundaries

Boundary lines are formed by the intersection of perpendicular bisectors of every pair of points. Using pairs of closest points in different classes gives a good enough approximation. (To be absolutely sure about the boundaries, one would draw perpendicular bisectors between each pair of neighboring points to create a region for each point, then consolidate regions belonging to the same class, i.e., remove the boundaries separating points in the same class. This technique is unnecessary for our purposes.)

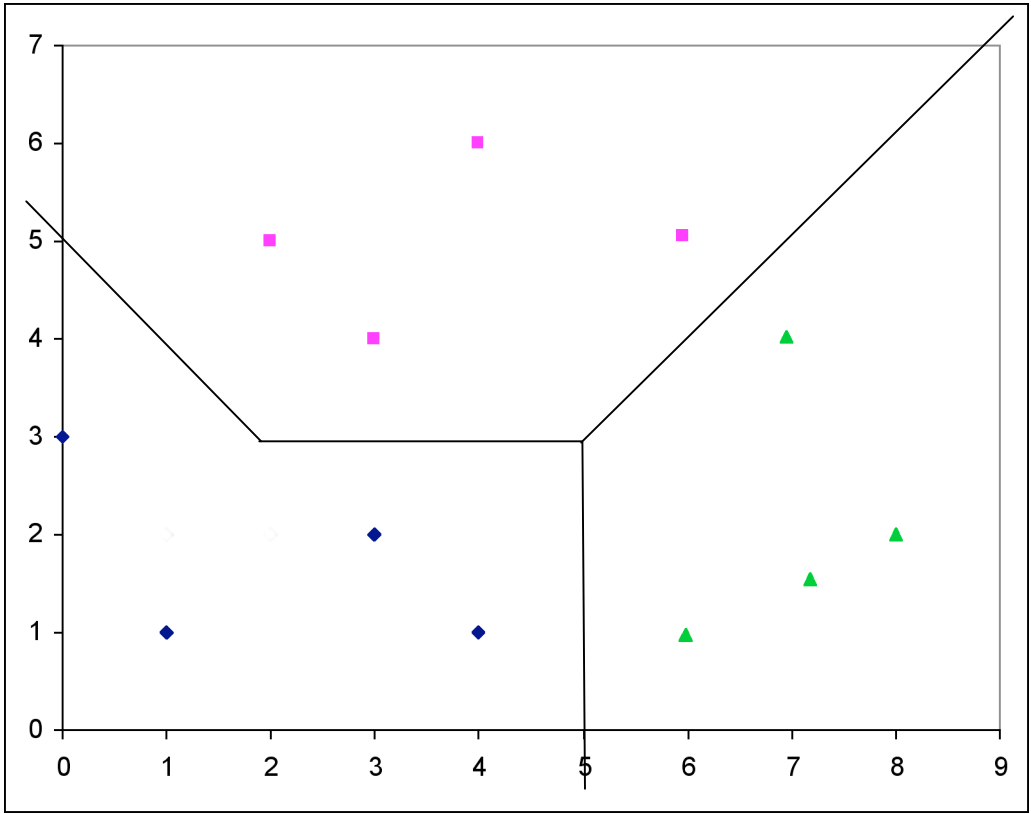






Construct lines between closest pairs of points in different classes.

Draw perpendicular bisectors.



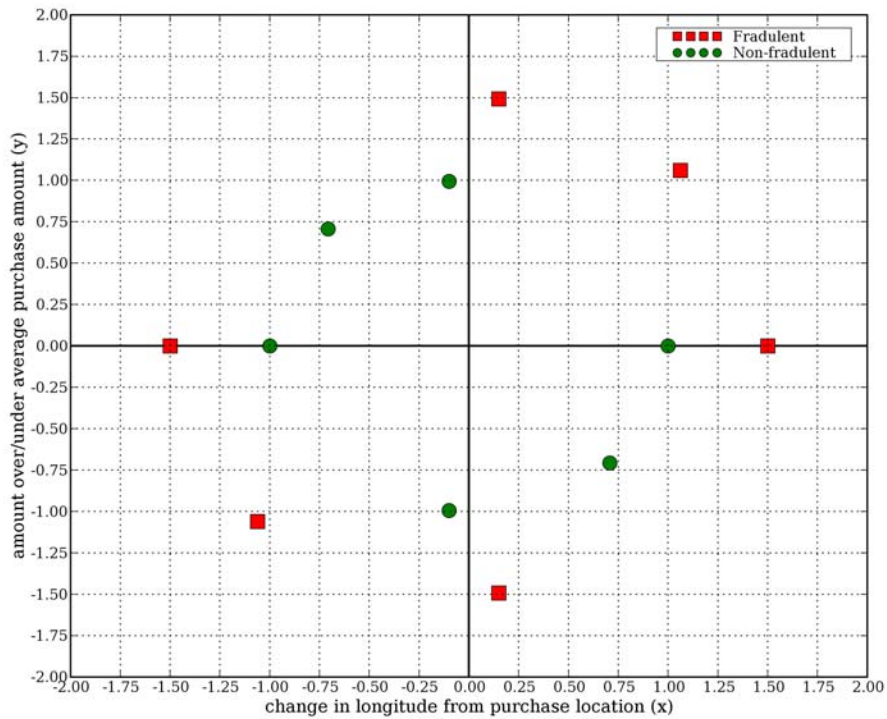
End bisectors at intersections; extend beyond axes (to infinity).

### 3242B3'P gct gw'P gli j dqt u'Rt cevleg'Rt qdrgo '3

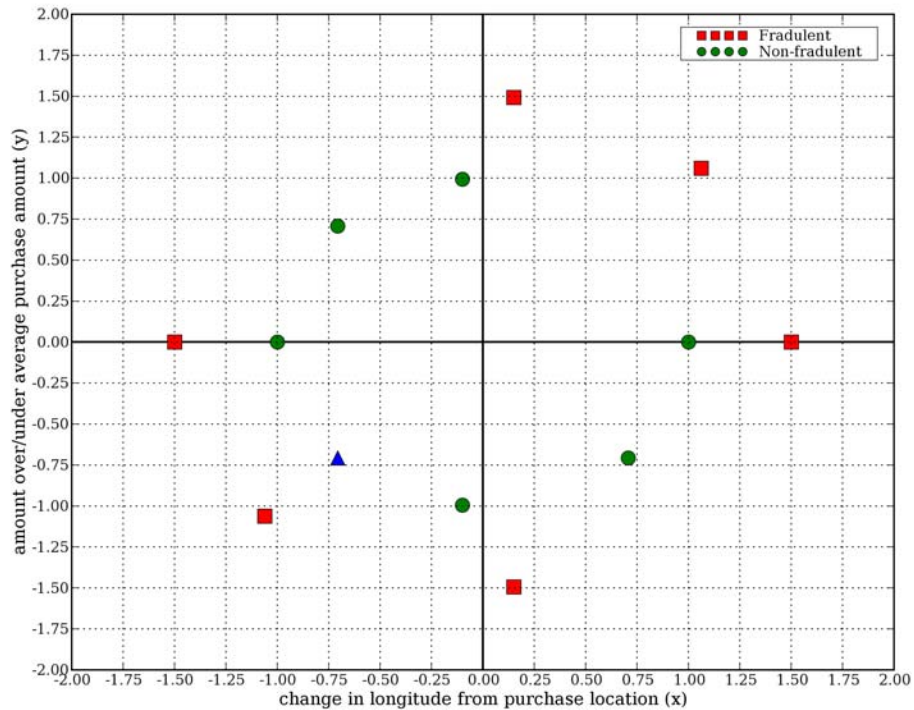
Lucy has been working hard for the credit card companies to detect fraud. They have asked her to analyze a number of classification methods to determine which one is best suited to their problem. The two quantities that they have provided her are the change in longitude from the purchase location to the registered address and the amount that the purchase is over or under the average purchase that the customer usually makes.

#### Part A: Nearest Neighbors (15 pts)

Lucy decides to use nearest neighbors to solve this problem and plots the fraudulent / non-fraudulent data. Squares are fraudulent and circles are non-fraudulent. Sketch the resulting decision boundary on the figure below.



It is the end of the month and Lucy's boss comes over with new data hot off the presses (the triangle). He wants Lucy to analyze whether or not the new charge is fraudulent.



What is the nearest neighbor classification of the new charge, fraudulent or non-fraudulent?

She's not too sure about this classification and decides to rerun it using k-nearest neighbors for  $k=3$  and then for  $k=5$ . Is the charge fraudulent for these values of  $k$ ?

K= 3:

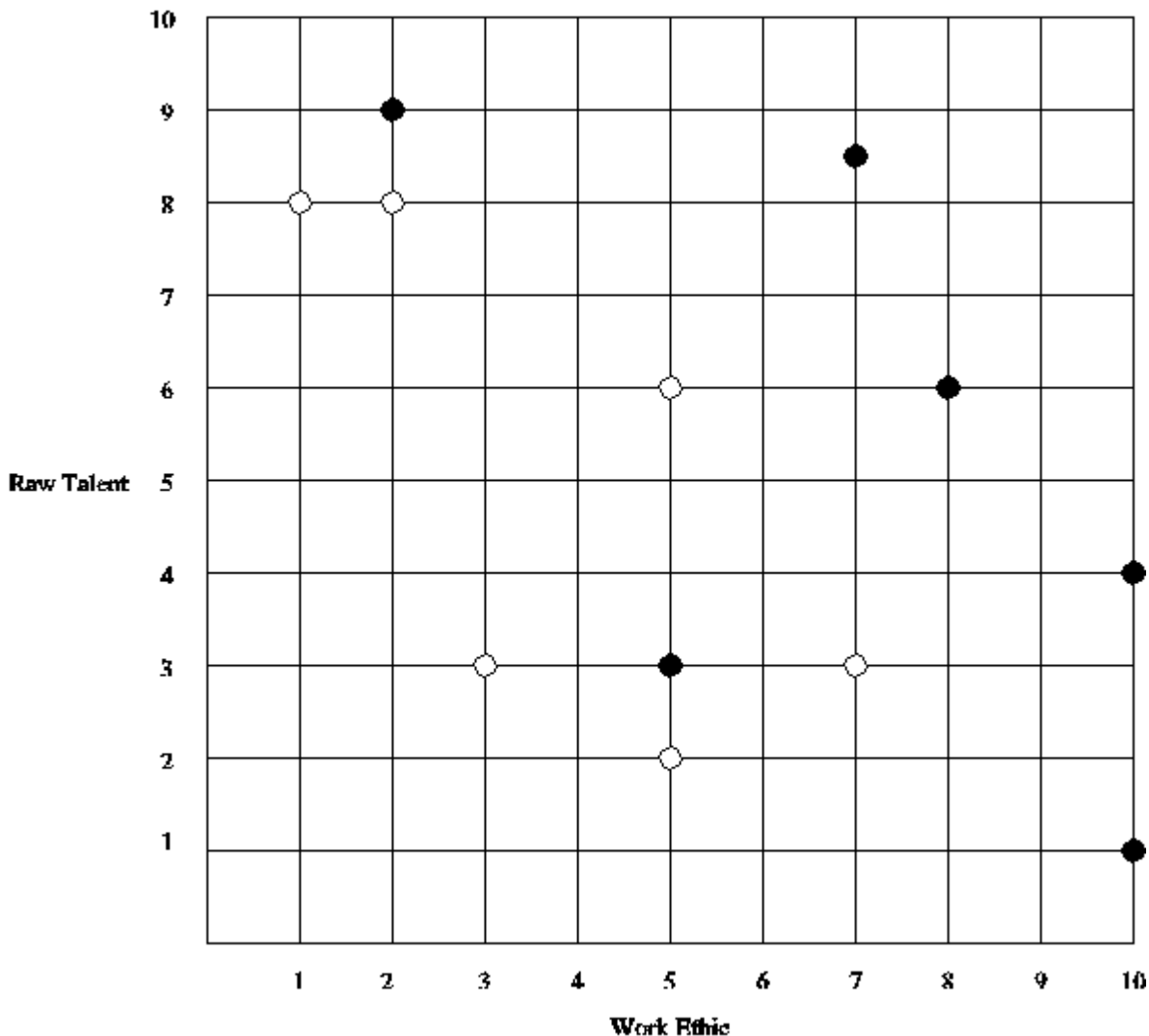
K= 5:

## 6.034 Recitation Thursday, October 20, 2011

### Practice Problem 2: k-Nearest Neighbors

The 6.034 staff has decided to launch a search for the newest AI superstar by hosting a television show that will make one aspiring student an *MIT Idol*. The staff has judged two criteria important in choosing successful candidates: work ethic (W) and raw talent (R). The staff will classify candidates into either potential superstar (black dot) or normal student (open circle) using a nearest-neighbors classifier.

**On the graph below, draw the decision boundaries that a 1-nearest-neighbor classifier would find in the R-W plane.**



# Identification trees Problem 1 (same credit card problem as k-NN above)

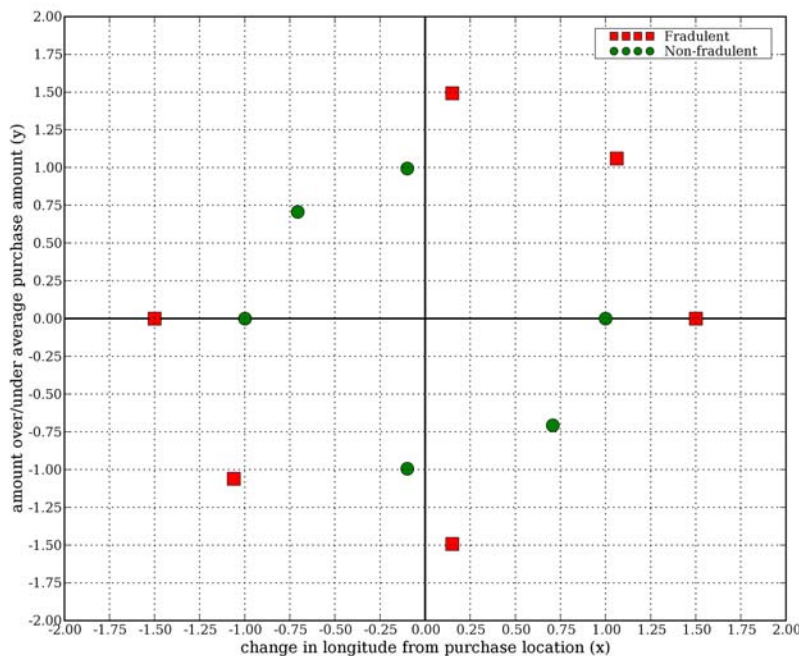
## B1 The boundaries (18 pts)

Lucy now decides that she'll try to use identification trees on the data. There are three likely candidates for splitting the data:  $x=0.0$ ,  $x=-1.01$  and  $x=1.01$ . Note that the  $-1.01$  and  $1.01$  values lie half-way between a square and a circle with nearby  $x$  values. Compute the average disorder for the decision boundary  $x=1.01$ . Your answer may contain logarithms.

Compute the average disorder for the decision boundary  $x=0.0$ . Again, your answer may contain logarithms.

Which of the two decision boundaries,  $x=0.0$  and  $x=1.01$ , is added first?

Sketch all of the decision boundaries on the figure below. Assume that  $x=0.0$  and  $x=1.01$ , in the order you determined above, are the first two decision boundaries selected (this may or may not be true, but assume it is).

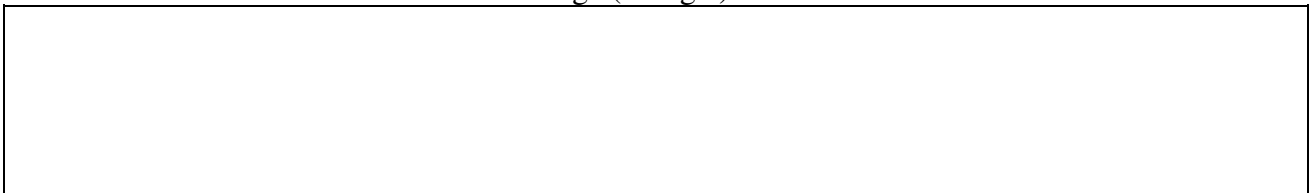


**B2 The identification tree (7 pts)**

Draw the identification tree corresponding to your decision boundaries.

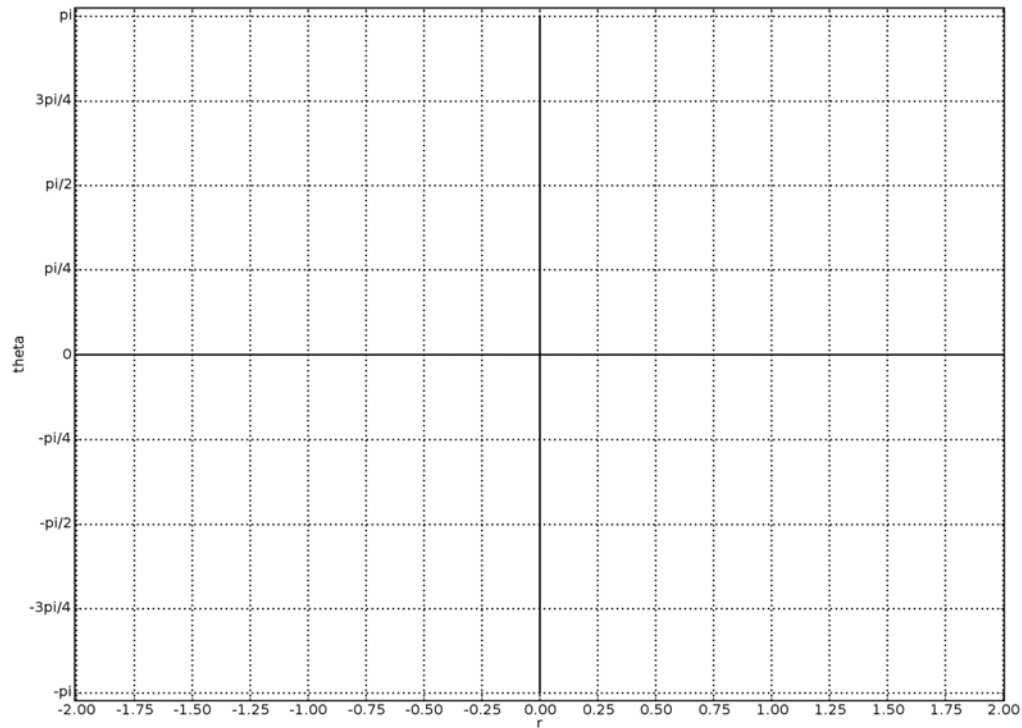


What is the classification of the new charge (triangle)?



### Part C: Polar coordinates (10 pts)

Lucy gets smart and decides to try a different space for each of the points. That is, she converts all of the points to polar coordinates. Sketch the data below. **You may assume that  $r$  value of each point is very close to a multiple of 0.25 and that the theta value of each point is very close to a multiple of  $\pi/4$ .**



How many decision boundaries do we need in this case?

Draw the resulting identification tree and sketch the decision boundary on the graph above.

# Identification Trees Practice Problem 2

## Part B1 (2 Points)

Now, leaving nearest neighbors behind, you decide to try an identification-tree approach. In the space below, you have two possible initial tests for the data. Calculate the average disorder for each test. Your answer may contain  $\log_2$  expressions, but no variables. The graph is repeated below.

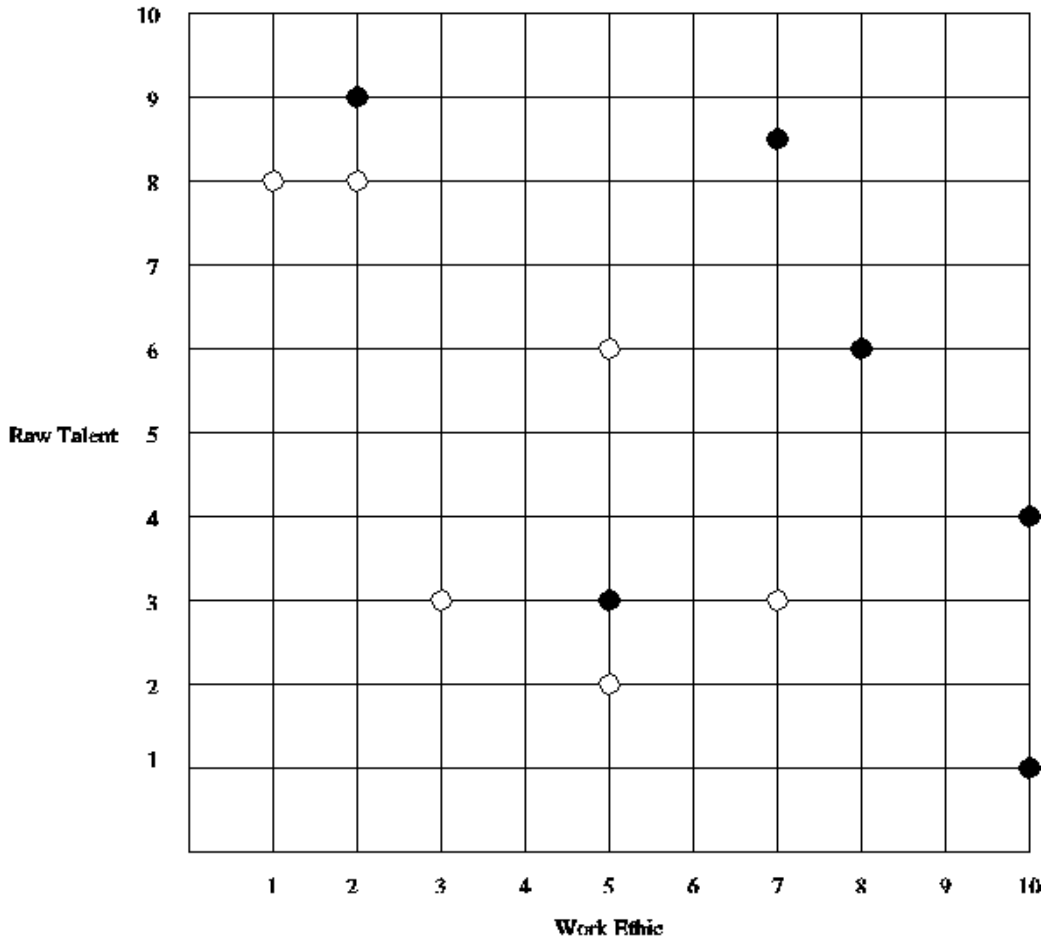
Test A:  $R > 5$ :

Test B:  $W > 6$ :

## Part B2 (2 Points)

Now, indicate which of the two tests is chosen first by the greedy algorithm for building identification trees.

We include a copy of the graph below for your scratch work.





### Part C: Identification Trees (4 Points)

Now, assume  $R > 5$  is the first test selected by the identification-tree builder (which may or may not be correct). Then, draw in all the rest of the decision boundaries that would be placed (correctly) by the identification-tree builder:

