MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.034—Artificial Intelligence
Recitation 8, November 2, 2001

Agenda

1. Learning: Types of problems; classification vs. regression problems
2. Representation issues
3. Features and Nearest Neighbors: Note the way the plane gets tiled
4. Learning: ID trees and information; from trees to rules
5. Learning: information theory
6. ID tree example

Learning: general issues

One general view of learning is this: finding a *function* based on past inputs or *examples* that can be used to *predict* future inputs. We are given a bunch of data with some features, and we want to predict some feature of interest:

1. Learning how to pronounce words
2. Learning how to throw a ball, ride a bicyle, play piano
3. Learning how to diagnose diseases, predict bankruptcy, etc.
4. Learning how to solve math problems
5. Learning how to predict movie choices

One way to carve up types of learning problems depends on the following general factors (applicable to all learning situations):

1. Representational adequacy: you can't learn something if you can't represent it.
2. Complexity control: how to avoid overfitting (predict 5 points with 5 parameters)
3. Feature choice

There are also ways to classify learning problems themselves, depending on:

1. The *kind* of data available for learning — *positive* examples or positive and *negative* examples (target is KNOWN).

2. The representation of the function itself — as points, a net, and so forth.
3. The availability of prior information
4. The type of prediction to be made — *discrete* (like a classification), or *continuous*.

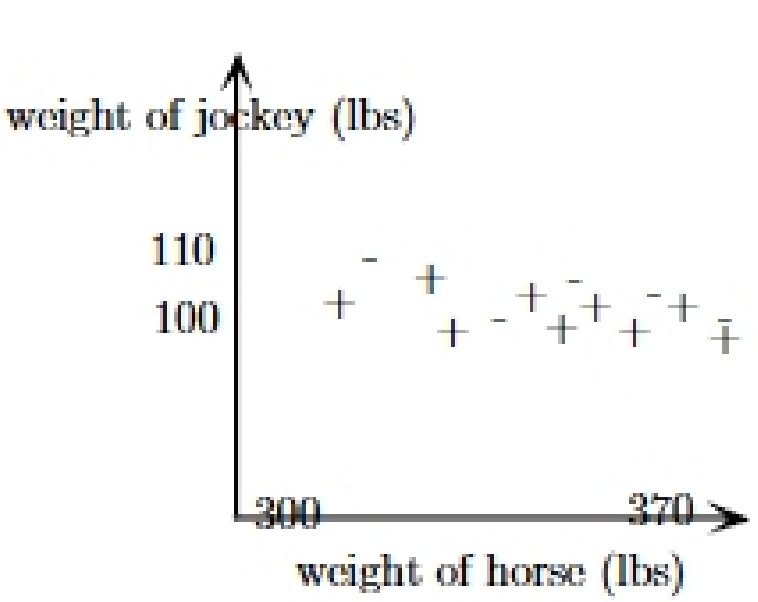
KEY QUESTION: HOW TO REPRESENT WHAT IS LEARNED

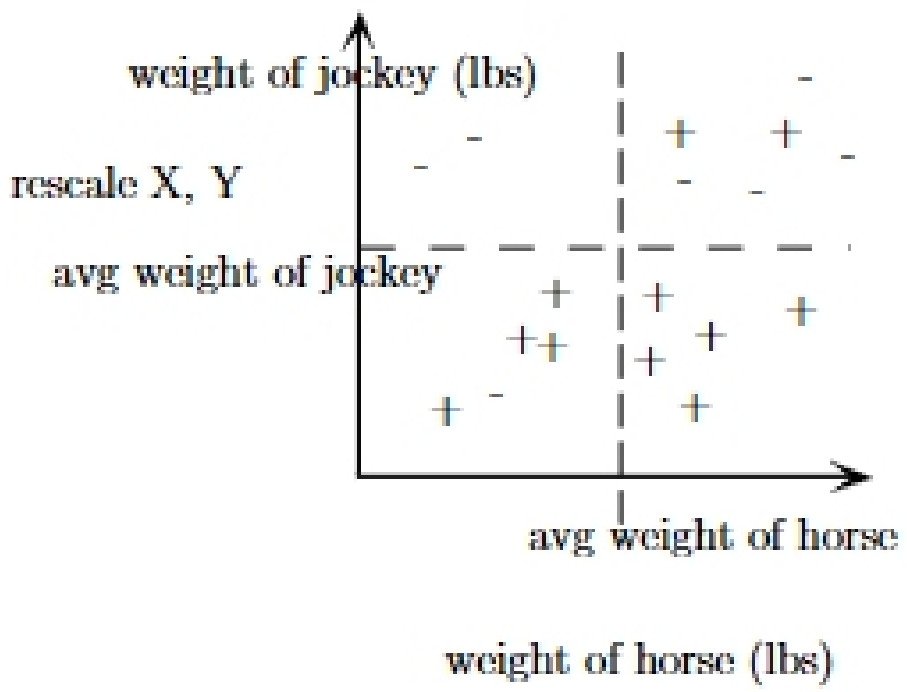
Learning: features and nearest neighbors

The *choice* of features determines whether a function may be learned by a representation. If the input is not correlated with the desired output, nothing can be learned. Even so, if a correlation exists, it might not be captured by the particular *feature choice*, or, even with a good set of features, *scaling* can matter.

Example of feature choice: polar vs. (x, y) coordinates.

Example of *scaling*: Predict outcome of horse race based on weight of horse and weight of jockey.





Nearest neighbors

KEY ASSUMPTION: The **stereotype** principle — THINGS THAT ARE ALIKE IN A FEW WAYS ARE ALIKE IN ALL OTHER WAYS.

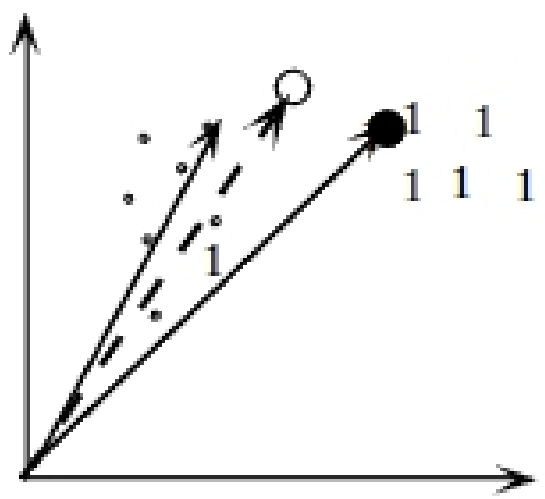
Application: Supervised learning/classification

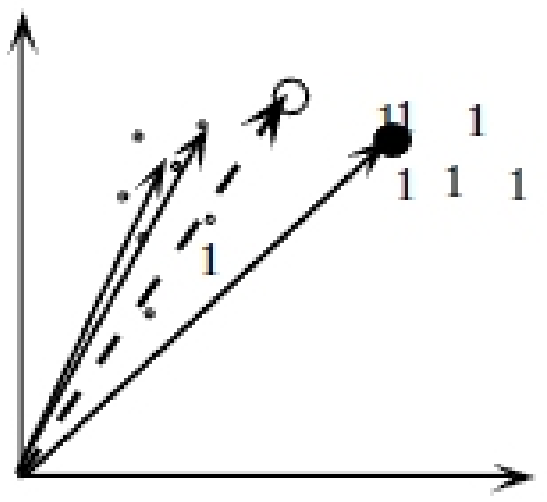
Training: Store *all* feature vector points in the training set, each labeled with its class.

Prediction: Given a *new* feature vector, find the "nearest" stored feature vector and return the result. Note the way it "tiles" the plane.

Advantages: Fastest possible training. Little bias — little preconception about the function that works. All features assumed equal — what does this imply about scaling?

Disadvantages: How to pick the distance metric (how to define *near*). No feature selection or extrapolation (doesn't do generalization). Prediction is costly when there are many features (but we can speed this up). Also sensitive to errors.





Extensions:

1. To reduce sensitivity to noise: pick k neighbors and have them vote, e.g., 3 nearest neighbors. This is k-nearest neighbors method. (see figure).

2. To speed up time for prediction, use a different type of representation (see next lecture: $K-D$ trees, splitting data along selected feature values.

3. To choose k and the metric, can do *cross-validation*: use one bunch of data for training, used to choose the free parameters, and another set is held back, and used for *validation*, used to estimate predictive performance.

   Cross-validation: divide training data into m subsets. Cycle through these subsets using each subset for the validation set, and the remaining subsets for the training set. The validation set is thus $1/m$ of the data, training set is $(m-1)/m$ of the data. Then we average the performance on each of the m subsets and use this as the estimated prediction performance.

   Bootstrap: Treat the training data as representing the actual input distribution, so sample from this distribution, with replacement, to obtain training and validation sets. Use average and deviation to characterize performance, along with confidence bounds.

Classification or ID Trees

Strategy: Divide feature space into boxes that are uniform with respect to labels. Do splitting along each axis, recursively, to define a *tree*.

Example: