Learning Bias & Clustering

Louis Oliphant
CS-540-2
www.cs.wisc.edu/~cs540-2
based on slides by Burr H. Settles

Outline

- Feature Spaces and Learner Bias
- Issues with Datasets
- Clustering

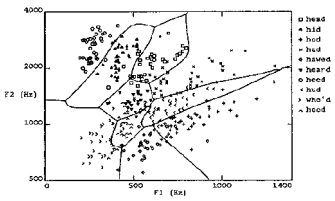
What is a Feature Space?

- So far in artificial intelligence, we've discussed all kinds of high-demensional "spaces," for example:
  - **Search space:** the set of states that can be reached in a search problem
  - **Hypothesis space:** the set of hypothesis that can be generated by a machine learning algorithm
- In this lecture, we'll talk about feature spaces, and the role that they play in machine learning

What is a Feature Space?

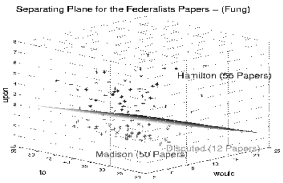
- Chances are, you already understand the idea of feature spaces even if you don't know it yet
- Recall that in our inductive learning framework, we usually represent examples as a vector of features: $\langle x_1, x_2, \ldots, x_n \rangle$
- Each feature can be thought of as a "dimension" of the problem… and each example, then is a "point" in an n-demensional feature space

Illustrative Example: 2D



This is the phoneme disambiguation problem from the neural network lecture: there were only two features (thus 2 "dimensions"), so it is easy to think of each example as a "point" in the 2D feature space.

Illustrative Example: 3D



Here is an example of a 3D feature space: the federalist papers were written in 1787-1788 by Hamilton, Jay, and Madison. The authorship of 12 of those papers are disputed between Hamilton/Madison. Using the frequency of use for 3 words as features, we can consider each of the documents a "point" in 3D feature space.

Learning and Feature Spaces

- So every time we describe a classification learning problem with a feature-vector, we are creating a feature space
- Then the learning algorithms must be *dividing up* that feature space in some way in order label new instances

Learner Bias

- Consider the following training set:

f1	f2	class
1	0	1
0	1	1
1	1	1

- How would you label the test example:

f1	f2	class
0	0	???

Learner Bias

- Consider the following training set:

f1	f2	class
1	0	1
0	1	1
0	0	0

- How would you label the test example:

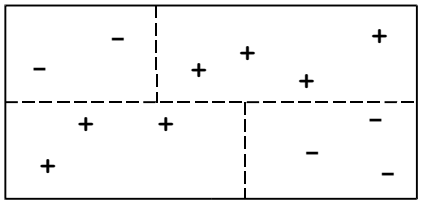
f1	f2	class
1	1	???

- Given just the data there is no reason to prefer one class over another.
- This is true for all datasets! The learner must have some kind of bias in order to make a decision on unseen examples

Decision Trees

- Let's think about decision trees and what they are doing to the feature space:
  - Each feature is a dimension in feature space
  - A decision tree recursively splits up the examples (points in feature space) based on one feature at a time
- So a decision tree essentially draws dividing lines in a dimension of feature space, and recursively subdivides along other dimensions
  - These lines are parallel to the axis of that dimension
  - We say that decision-trees make axis-parallel splits

Decision Trees

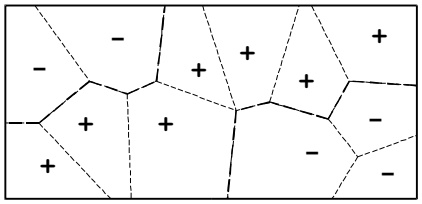


Given the above Venn diagram and these positive and negative training examples, the decision tree will draw axis-parallel boundaries to separate the two classes

k-Nearest Neighbors

- The k-nearest neighbors algorithm is a bit unique in its treatment of feature space:
  - Since it remembers all of the training examples anyway, it partitions the feature space when it is given a test example
  - The boundaries also depend on the value of *k*, the higher *k* is, the more complex and expressive the partitioning can be

k-Nearest Neighbors

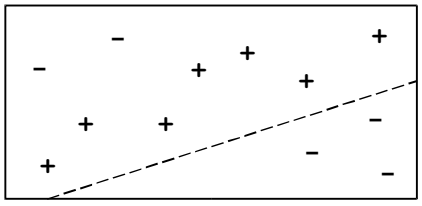


It's easiest to visualize what the basic 1-NN algorithm does: draw a Voronoi diagram, which constructs convex polygons around the examples for a more complex partitioning

Perceptrons

- Recall that a perceptron learns a linear hypothesis function *h*
- So it can only partition the data by drawing a "linear hyperplane"
  - Imagine an *n*-dimensional feature space
  - The perceptron learns an (*n*-1)-dimensional "line" (or "plane" or "surface") that separates the classes

Perceptrons

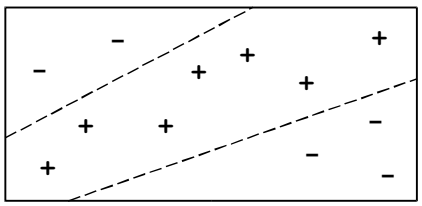


Clearly, simple perceptrons cannot completely separate the positives from the negatives, but they will try to learn a linear hypothesis that does as best they can

Neural Networks

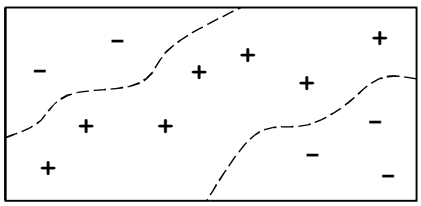
- Recall that as soon as we go from a single perceptron to a full network, the hypothesis function becomes much more expressive
  - With only one hidden layer we can learn any arbitrary classification problem
  - Well, given enough hidden units, anyway

Neural Networks: 2 Hidden Units



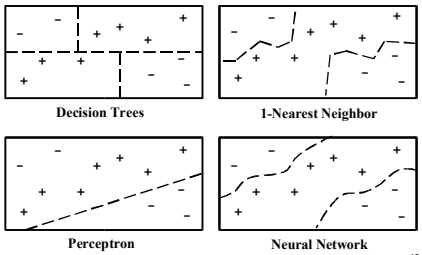
With only two hidden units, a neural network can learn two different hyperplanes to separate the data completely

Neural Networks: ∞ Hidden Units



With an arbitrary number of hidden units, the network can learn a function that is much more expressive, even appearing to be "contoured" (e.g. phoneme example)

Different Learning Models



Issues with Datasets

- Lots of types of datasets
  - missing values
  - noise / errors
  - lots of irrelevant features
  - little training data
  - unbalanced data
- How do these effect which learner to use?
- Let's look at just one issue: lots of features

The Curse of Dimensionality

- The problem of having too many features describing an inductive learning task is the curse of dimensionality
- As we add more features to the problem description, there are more features for the agent to use when constructing its hypothesis
- More features make the model more expressive, but maybe *not all* of these features are even relevant to the concept

Case Study: Text Classification

- One classic problem that illustrates the curse of dimensionality is the text classification task:
  - Each document is an "example"
  - The documents are labeled from a set of topics, which are classes in our inductive learning framework
  - Every word in the vocabulary is a Boolean feature: either it is in the document or not
- A given document can be hundreds of thousands of words long, and most of them will not have anything to do with the topic label!

The Curse of Dimensionality

- Which algorithms suffer most from the curse of dimensionality?
  - ***k*-nearest neighbors:** clearly suffers… it uses all features to compute the distance between examples
  - **Naïve Bayes:** also considers every feature as equally relevant to the concept function
  - **Neural networks:** can reduce the weights of irrelevant features close to zero, but it might take BP a long time to converge, and more likely to find a local minimum
  - **Decision trees:** these seem to do OK… induced trees are usually small, and only use "relevant" features

The Curse of Dimensionality

- A common way to combat to many features is to select just a subset of features to use when build the model
- Known as feature selection
- Many ways to pick the subset
  - Forward chaining
    - keep adding best feature as long as model keeps improving
  - Backward chaining
    - keep removing worst feature as long as model keeps improving

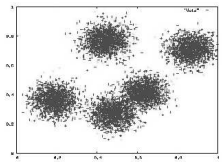
Clustering

- Given:
  - **a dataset** of examples. Each example is an unlabeled continuous-feature vector
  - **k** – the number of groups the examples in the dataset should be grouped into
- Do:
  - group the examples into **k** groups, with each example being in exactly 1 group

Example dataset

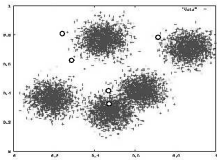
- k=5
- dataset is:

0.841995, 0.663665
0.400491, 0.744203
0.646776, 0.354766
0.50261, 0.744252
0.078588, 0.334003
0.168425, 0.383141
0.842949, 0.816373
0.388433, 0.35108
0.695454, 0.322042
0.43337, 0.295312
0.617841, 0.366261
0.426698, 0.257651
...



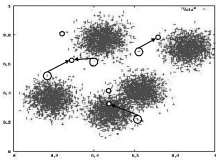
K-Means Algorithm

- Select K initial centroids



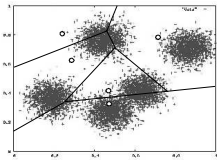
K-Means Algorithm

- Assign each data point to closest centroid



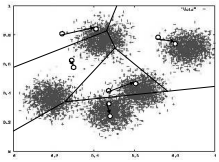
K-Means Algorithm

- Assign each data point to closest centroid



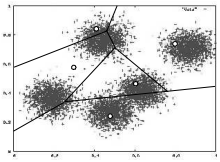
K-Means Algorithm

- Calculate the new centroid for each cluster



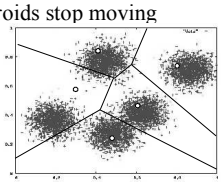
K-Means Algorithm

- Calculate the new centroid for each cluster



K-Means Algorithm

- Repeat:assign nodes to new centroids
- calculate new centroids
- until centroids stop moving



K-Means algorithm

- step 1:Select k initial centroids
- step 2: assign each data point to the closest centroid (uses Euclidean Distance)
- step 3: recalculate the centroid of each group (centroid is the average of each feature for all examples in group)
- repeat step 2 and 3 until convergence

Issues with K-Means

- Picking K
  - What if K where equal to the number of points?
  - What if K=1?
  - Use domain knowledge to pick K
- Local Search, converges to local optimum
- K-Means can be generalized into an Expectation-Maximization algorithm that uses mixtures of Gaussians
  - soft association of data points with centroids

Issues with clustering

- Picking good starting centroids
  - Pick random point x1 from dataset
  - Find point x2 farthest from x1
  - Find x3 farthest from closer x1, x2
  - ...pick k points like this, using them as centroids
- How to assign "labels" for each cluster

Summary

- When describing inductive learning tasks as vector of *n* features, we create a feature space
  - Examples can be though of as "points" in *n*-D space
- Learner Bias
- Feature Selection
- K-Means clustering Algorithm