Feature Spaces

Burr H. Settles
CS-540, UW-Madison
www.cs.wisc.edu/~cs540-1
Summer 2003

1

Announcements

- Homeworks #4/#5 are due next Tuesday (7/29)
- Project papers/reports are due one week from today (Friday, 8/1)
- The near future:
  - Next week we have a few more lectures
  - Mon-Wed of the week afterward will be project presentations (sign up sheet at the front)
  - Review for final exam a week from Thursday (8/7)
  - Final exam in class that Friday (8/8)

2

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

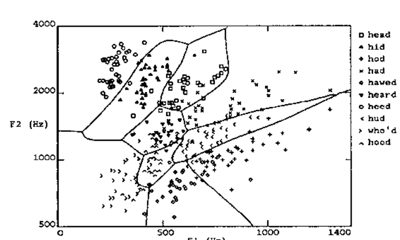
3

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

4

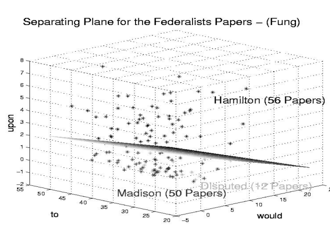
Illustrative Example: 2D



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.

5

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.

6

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 *manipulating* that feature space in some way in order label new instances

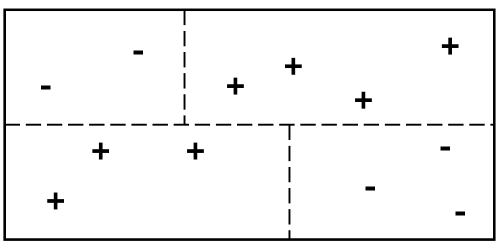
7

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 create axis-parallel splits

8

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

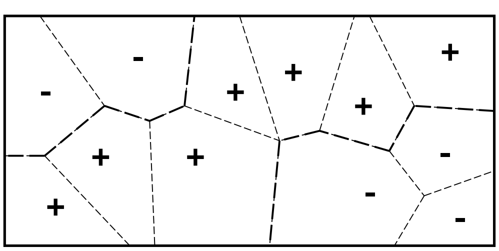
9

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

10

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

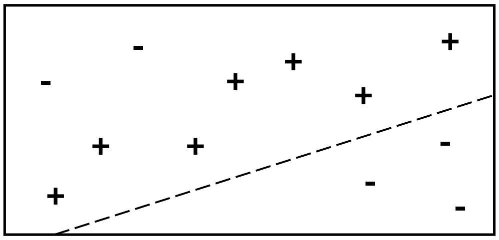
11

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

12

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

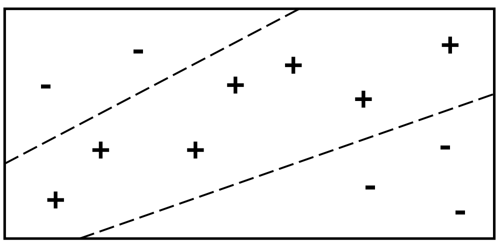
13

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

14

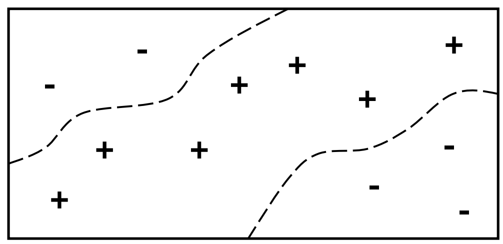
Neural Networks: 2 Hidden Units



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

15

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)

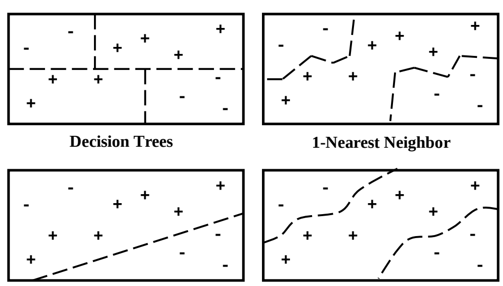
16

Naïve Bayes

- Recall from yesterday that a naïve Bayes classifier learns a linearly separating hyperplane, just as a perceptron would
- The difference in how the line turns out in in the training mechanism:
  - Perceptrons use gradient descent (discriminative training)
  - Naïve Bayes estimates probabilities conditioned on the class label (generative training)

17

Different Learning Models



Decision Trees | **1-Nearest Neighbor** | **Perceptron / Naïve Bayes** | **Neural Network**

18

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

19

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!

20

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

21

Feature Selection

- Perhaps we can learn a lesson from the way decision trees do things: select only the features that seem relevant to the problem!
- This should not only improve the classifier, but might even speed up learning
  - Will result in fewer weights to optimize, fewer probabilities to estimate, dimensions with which to compute distance, etc…

∗ *But how to know what features are important??*

22

Feature Selection

- One possibility is to… well… induce a decision tree and use the features that it used
  - This sometimes works alright
- But since other learning algorithms don't produce the logical tree structure, we have problems:
  - The logical relationship between the features is lost
  - In the worst case, the tree could overfit and use all of the features: then we didn't gain anything

23

Ranked Feature Selection

- Another possibility is to use some scoring function to rank each feature in the problem, and then choose the best *k* features
  - Could choose the top 10%, 25%, 50%, etc.
  - Could also look for statistically significant gaps in the rankings and take those that are above the gap
- So what would make a good ranking function for feature selection for, say, text classification?

24

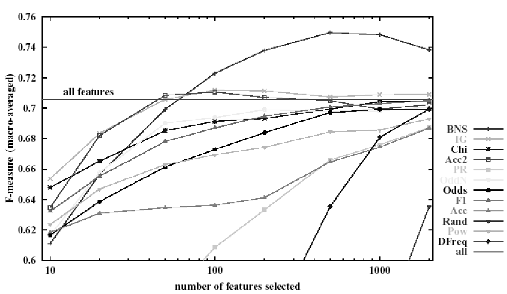
Ranked Feature Selection

- One possibility is to choose the features that have the most consistent values
  - For text classification, this means ranking the features (words) by how often they appear in the corpus (training set)
  - Still doesn't tell us much about correlation between the word and the various topics
  - In fact, if a word appears a lot in every document, then it probably is *not* very informative about the topics!
- Another common ranking function is… believe it or not… information gain!
  - Estimates how well the feature (word) "separates" the data into the different class labels

RFS for Text Classificiation

- G. Forman, "An Extensive Empirical Study of Feature Selection Metrics for Text Classification," Hewlett-Packard Technical Note, 2003
- Compared 12 feature selection methods on the Reuters news articles text classification dataset
  - Included "frequency" and "information gain" feature selection methods
  - Usually use linear models (naïve Bayes, perceptron, support vector machines) for these problems

RFS for Text Classificiation



RFS for Text Classificiation

- Clearly the document frequency metric didn't "make the headlines"
- Information gain, however, was able to reduce the feature set from 12,500 words to 50 words without reducing performance
  - That's a final feature set that's 0.4% the size of the original vocabulary!
- The "federalist papers" example earlier also used feature selection to chose the 3 words

Other Selection Methods

- Unfortunately, most of the features we can rank with a scoring function to must be Boolean (or at least discrete)
- But *k*-nearest neighbors or neural networks deal with continuous features so naturally!
  - Is there some way to do efficiently select continuous features too?

Feature Selection as Search

- The answer is to try and perform an optimization search to find the best feature set:
  - **States:** subsets of the available features
  - **Actions:** add/remove features
  - **Objective function:** maximize performance of the chosen algorithm on the problem
- This sort of feature selection generalizes well to all features, all algorithms, and all problems (even to regression tasks!)

Feature Selection as Search

- There are two common, efficient ways to search for a feature set like this…
  - Forward chaining: begin with an *empty* feature set, and gradually add the feature that most improves performance of the algorithm, until it begins to drop
  - Backward chaining: begin with *all* available features, and remove them one at a time until performance drops
- This solves the dilemma of trying to pick a good initial state, but if we have a slow training algorithm (e.g. neural network) the entire process can take a while…

31

Feature Induction

- Feature selection can help keep hypotheses from becoming *overly* expressive
  - Throws out irrelevant features
  - Often reduces overfitting for naïve Bayes, *k*-nearest neighbors, and sometimes neural networks
- However, sometimes the feature set we have isn't expressive *enough*
  - In this case, we might want to create new features that suit the problem well… this is called feature induction

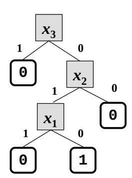
32

Feature Induction

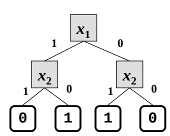
- Consider a decision tree being trained on this data set:

x = 110; $f(x)$ = 0 **x = 010; $f(x)$ = 1**
x = 100; $f(x)$ = 1 **x = 001; $f(x)$ = 0**

ID3 might learn this tree:



But the real concept is probably:



33

Feature Induction

- Features x_1 and x_2 are at the core of the concept function ($x_1 \otimes x_2$)
- But each feature alone yields an information gain of zero (so ID3 won't choose it)
- Perhaps we could make use of a technique that can create a new feature ($x_1 \otimes x_2$), and consider it in learning as well

34

Feature Induction

- Another example: recall that naïve Bayes assumes that each feature is independent
  - For text classification, this means that each word has nothing to do with the other words in the vocabulary
- Consider classifying newspaper articles:
  - If the word "box" occurs in a document, it probably isn't very informative about any news topic
  - Likewise, if the word "office" appears, it might be more about business, but still not too helpful
  - But if we add a new feature, "box office," this feature is now highly correlated with "entertainment"

35

Feature Induction

- In practice the best way to perform feature induction is similar to feature selection:
  - Begin with the base set of features
  - Exhaustively propose new features that are logical operations on the base features
    - Again we have trouble with continuous features, which must be descretized somehow
  - Either rank the new features by some scoring function and add the best *k*, or do an optimization search like forward chaining

36

Feature Induction

- A. McCallum, “Efficiently Inducing Features of Conditional Random Fields,” *UAI*, 2003.
  – Used models called conditional random fields (CRFs) to learn to extract entity names (people, locations, organizations, etc.) from labeled text documents
  – Induced novel Boolean features using a forward-chaining sort of approach

	Per	**Loc**	**Org**	**Misc**	**All**
W/O induction	61.9%	86.7%	63.5%	77.2%	73.3%
With induction	93.2%	92.4%	84.4%	80.0%	89.0%

37

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
- What classification algorithms do is to find an optimal way of partitioning that feature space up into the correct classes
- We can use feature selection to remove many irrelevant features from the feature set
- We can also use feature induction to add new and potentially more relevant features the feature set

38