Support Vector Machines and Gene Function Prediction

Brown et al. 2000 PNAS.

CS 466

Saurabh Sinha

Outline

- A method of functionally classifying genes by using gene expression data
- Support vector machines (SVM) used for this task
- Tests show that SVM performs better than other classification methods
- Predict functions of some unannotated yeast genes

Motivation for SVMs

Unsupervised Learning

- Several ways to learn functional classification of genes in an unsupervised fashion
- “Unsupervised learning” => learning in the absence of a teacher
- Define similarity between genes (in terms of their expression patterns)
- Group genes together using a clustering algorithm, such as hierarchical clustering

Supervised Learning

- Support vector machines belong to the class of “supervised learning” methods
- Begin with a set of genes that have a common function (the “positive set”)
- … and a separate set of genes known not to be members of that functional class (the “negative set”)
- The positive and negative sets form the “training data”
  - Training data can be assembled from the literature on gene functions

Supervised learning

- The SVM (or any other supervised learner) will learn to discriminate between the genes in the two classes, based on their expression profiles
- Once learning is done, the SVM may be presented with previously unseen genes (“test data”)
- Should be able to recognize the genes as members or non-members of the functional class

SVM versus clustering

- Both use the notion of "similarity" or "distance" between pairs of genes
- SVMs can use a larger variety of such distance functions
  - Distance functions in very high-dimensional space
- SVMs are a supervised learning technique, can use prior knowledge to good effect

Data sets analyzed

Data sets

- DNA microarray data
- Each data point in a microarray experiment is a ratio:
  - Expression level of the gene in the condition of the experiment
  - Expression level of the gene in some reference condition
- If considering *m* microarray experiments, each gene's expression profile is an m-dimensional vector
- Thus, complete data is n x m matrix (n = number of genes)

Data sets

- “Normalization” of data
- Firstly, take logarithms of all values in the matrix (positive for over-expressed genes, negative for repressed genes)
- Then, transform each gene’s expression profile into a unit length vector
- How?

Data sets

- n = 2467, m = 79.
- Data from earlier paper on gene expression measurement, by Eisen et al.
- What were the 79 conditions?
  - Related to cell cycle, “sporulation”, temperature shocks, etc.

Data sets

- Training sets have to include functional labeling (annotation) of genes
- Six functional classes taken from a gene annotation database for yeast
- One of these six was a “negative control”
  - No reason to believe that members of this class will have similar expression profiles
  - This class should not be “learnable”

Support Vector Machine

SVM intro

- Each vector (row) X in the gene expression matrix is a point in an m-dimensional “expression space”
- To build a binary classifier, the simple thing to do is to construct a “hyperplane” separating class members from non-members
  – What is a hyperplane?
  – Line for 2-D, plane for 3-D, … hyperplane for any-D

Inseparability

- Real world problems: there may not exist a hyperplane that separates cleanly
- Solution to this “inseparability” problem: map data to higher dimensional space
  - Example discussed in class (mapping from 2-D data to 3-D)
  - Called the “feature space”, as opposed to the original “input space”
  - Inseparable training set can be made separable with proper choice of feature space

Going to high-d spaces

- Going to higher-dimensional spaces incurs costs
- Firstly, computational costs
- Secondly, risk of “overfitting”
- SVM handles both costs.

SVM handles high-D problems

- Overfitting is avoided by choosing “maximum margin” separating hyperplane.
- Distance from hyperplane to nearest data point is maximized

SVM handles high-D problems

- Computational costs avoided because SVM never works explicitly in the higher-dimensional feature space
  - The “kernel trick”

The kernel trick in SVM

- Recall that SVM works with some distance function between any two points (gene expression vectors)
- To do its job, the SVM only needs “dot products” between points
- A possible dot product between input vectors X and Y is $\sum X_i Y_i$
  - represents the cosine of the angle between the two vectors (if each is of unit length)

The kernel trick in SVM

- The “dot product” may be taken in a higher dimensional space (feature space) and the SVM algorithm is still happy
  - It does NOT NEED the actual vectors in the higher-dimensional (feature) space, just their dot product
- The dot product in feature space is some function of the original vectors X and Y, and is called the “kernel function”

Kernel functions

- A simple kernel function is the dot product in the input space
- The feature space = …
- … the input space
- Another kernel: $(\sum X_i Y_i)^2$
  – A quadratic separating surface in the input space (a separating hyperplane in some higher dimensional feature space)

Soft margins

- For some data sets, SVM may not find a separating hyperplane even in the higher dimensional feature space
- Perhaps the kernel function is not properly chosen, or data contains mislabeled examples
- Use a “soft margin”: allow some training examples to fall on the “wrong” side of the separating hyperplane
- Have some penalty for such wrongly placed examples

Data analysis

Data recap

- 79 conditions (m), ~2500 genes (n), six functional classes
- Test how well each of the functional classes can be learned and predicted
- For each class, test separately; treat it as positive, everything else as negative

Three-way cross validation

- Divide all positive examples into three equal sets, do the same for all negative examples
- Take two sets of positives, two sets of negatives, and train the classifier
- Present the remaining (one set each of) positives and negatives as “test data” and count how often the classifications were correct

Measures of accuracy

- False positives, True positives
- False negatives, True negatives
- This paper uses cost function “C(M)” of learning method M as:
    - C(M) = FP(M) + 2*FN(M)
    - Ad hoc choice
- This paper defines “cost savings” of method M as
    - S(M) = C(N) - C(M), where N is the “null learning procedure” (call everything negative)

Results

SVM outperforms other methods

Class	Method	FP	FN	TP	TN	S(M)
Prot	D-p 1 SVM	21	7	28	2,411	35
	D-p 2 SVM	6	8	27	2,426	48
	D-p 3 SVM	3	8	27	2,429	51
	Radial SVM	2	8	27	2,430	52
	Parzen	21	5	30	2,411	39
	FLD	7	12	23	2,425	39
	C4.5	17	10	25	2,415	33
	MOC1	10	17	18	2,422	26

Brown et al. PNAS 2000; 97; 262-267

SVM does best

- For every class, best performing class is the SVM
- All classifiers fail on the sixth class (negative control), as expected
- SVM does better than hierarchical clustering
- More results in the paper