Module 3, Lecture 3: Some Extensions of Classical Neural Networks

M. Vidyasagar

Distinguished Professor, IIT Hyderabad
Email: m.vidyasagar@iith.ac.in
Website: www.iith.ac.in/∼m_vidyasagar/

Outline

IIT Hyderabad

Outline

IIT Hyderabad

The Vapnik-Chervonenkis (VC)-Dimension

With every neural network *architecture*, we can associate an intege, call it d, referred to as the **Vapnik-Chervonenkis (VC)-Dimension**. In its simplest form it applies to *classification* NNs.

It is a measure of the "richness" of the architecture, and it is *monotonic*.

So if we take a NN architecture and add a few more layers and/or neurons, the VC-dimension can only go up.

Before we define this concept, we answer the question: What is it good for?

VC-Dimension and the Learning Rate

Suppose d is the VC-dimension of a NN architecture (or an upper bound). Given a set of labelled samples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$ where $y_i \in \{-1, 1\}$.

Train the NN by adjusting the weights to minimize the *fraction of misclassified samples*.

$$\min_{\mathbf{w}} J(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} \#(y_i \neq f(\mathbf{w}, \mathbf{x}_i)).$$

Let J^* denote this minimum. Suppose we correctly classify all training inputs, so that the error $J^* n = 0$.

VC-Dimension and the Learning Rate (Cont'd)

Given an accuracy ϵ, δ, define

$$m_0(\epsilon, \delta) = \max \left\{ \frac{8d}{\epsilon} \log_2 \frac{8e}{\epsilon}, \frac{4}{\epsilon} \log_2 \frac{2}{\delta} \right\},$$

where d is the VC-dimension of the NN architecture (or an upper bound).

If the network is trained with at least m_0 samples, then a randomly selected *test input* is correctly classified with accuracy ϵ, with probability $\geq 1 - \epsilon$.

The VC-dimension of a perceptron with n inputs is $n + 1$, and $J^* = 0$ if the data is linearly separable.

Training and Testing Error

The overall error is a sum of the training error and the "generalization error."

The training error depends on the quality of the algorithm used to minimize it.

The generalization error depends on the NN architecture.

There is a lot of analysis of the VC-dimension of NN architectures.

VC-Dimension of Neural Networks

What is the VC-dimension?

Suppose that we have a neural network architecture, and a collection of m vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$.

There are 2^m different ways to assign labels $y_i \in \{-1, 1\}$ for each $\mathbf{x}_i$.

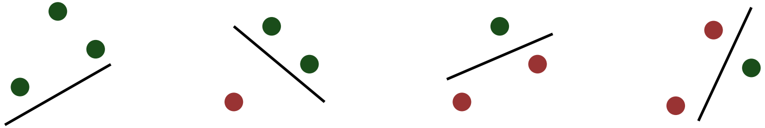
Is the NN architecture rich enough that, for *each* of these 2^m possible labellings, the training error can be made zero by suitably adjusting the weights?

As m is increased, this becomes more and more difficult.

The VC-dimension is the *largest value* of m such that, for some collection of vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$, *everyone* of the 2^m labellings can be achieved.

VC-Dimension of a Perceptron with Two Inputs

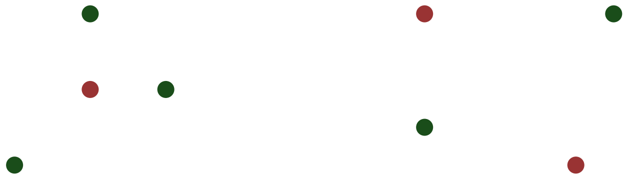
Take 3 points, which do not lie on a straight line. All $2^3 = 8$ labellings can be achieved by a suitably chosen perceptron.



Other four labellings are just mirror images of these. So the VC-dimension is ≥ 3.

VC-Dimension of a Perceptron with Two Inputs (Cont'd)

Now suppose we have four vectors in $\mathbb{R}^2$. There are two cases: (a) One of the vectors is inside the triangle formed by the other three. (b) None of the vectors is inside the triangle formed by the other three.



(a) Every half-plane that contains all three green dots must also contain the maroon dot. (b) This is just the XOR example.

VC-Dimension of a Perceptron with n Inputs

So the VC-dimension of a perceptron on $\mathbb{R}^2$ is three.

On $\mathbb{R}^n$, it is $n+1$. In fact, given *any* set of $n+1$ points that do not lie in a lower-dimensional subspace, *all* 2^{n+1} assignments of labels can be realized by a SVM.

This is the rationale behind higher-order SVMs – By enlarging the number of features, we ensure that the data is linearly separable (in the higher-dimensional space).

VC-Dimension of Multi-Layer Pereptron Networks

If a MLPN contains k individual neurons, and neuron i has n_i adjustble parameters, then the overall VC-dimension is no larger than

$$2k \log_2(ek) \max_i n_i .$$

So the VC-dimension of a MLPN can grow *slightly faster* than the number of weights, but not much faster.

Outline

IIT Hyderabad

Outline

IIT Hyderabad

Traditional Linear Classifier: Reprise

Traditional classifiers start with training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^n, y_i \in \{0, 1\}$. (Note the change in the "label space" from $\{-1, 1\}$ to $\{0, 1\}$.)

If the data set is linearly separable, then *all* training data is correctly classified. Specifically, we find $\mathbf{w}, \theta$ such that the linear discriminant

$$\Delta(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x} - \theta \begin{cases} \geq 0 & \text{if } y_i = 1, \\ < 0 & \text{if } y_i = 0. \end{cases}$$

A new test sample $\mathbf{x}$ is assigned a label of 1 or 0 according as $\Delta(\mathbf{x})$ is ≥ 0 or < 0.

Limitations of Linear Classifiers

- Do not readily adapt to the case where the data is *not* linearly separable.
- Test samples are *unambiguously* assigned a label of 1 or 0, but there is no notion of "confidence."
- Soft-margin SVMs address the first point but not the second.

Logistic regression is a way of addressing both points.

Caution: Logistic regression *should not be used* with linearly separable data – it should be used *only* when data is not linearly separable.

Outline

Convex Sets

A set $S \subseteq \mathbb{R}^n$ is **convex** if

$$x, y \in S \implies \lambda x + (1-l)y \in S, \ \forall \lambda \in [0,1].$$

In words, a straight line connecting any two points in the set is fully contained in S.

Depiction of Convex Sets

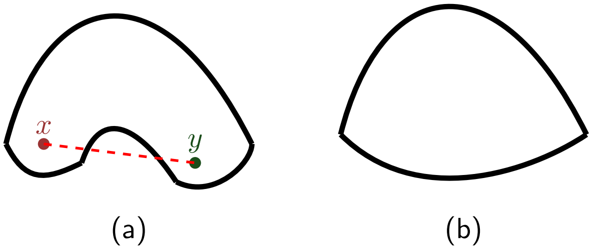


Figure: The set in (a) is not convex but the set in (b) is convex.

Convex Functions

Suppose $S \subseteq \mathbb{R}^n$ is a convex set, and that $f : S \to \mathbb{R}$. We say that the function f is **convex** if

$$f(\lambda \mathbf{w}_1 + (1-\lambda)\mathbf{w}_2) \leq \lambda f(\mathbf{w}_1) + (1-\lambda) f(\mathbf{w}_2), \ \forall \mathbf{w}_1, \mathbf{w}_2 \in S, \lambda \in [0,1].$$

We write $f(\mathbf{w})$ and not $f(\mathbf{x})$ because we will be dealing with convex functions of the *weights*.

Convex Functions: Depiction

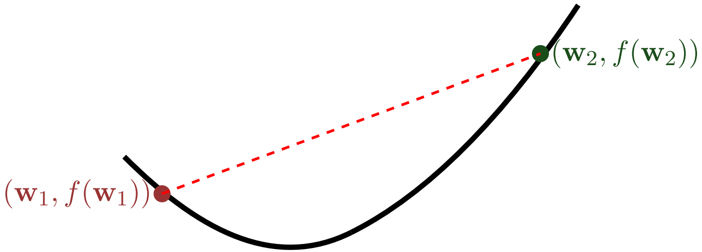

$(\mathbf{w}_1, f(\mathbf{w}_1))$
$(\mathbf{w}_2, f(\mathbf{w}_2))$

Figure: Graph Below Chord Interpretation of a Convex Function

Checking for Convexity

If the function f is twice differentiable, then f is convex if and only if $\nabla^2 f(\mathbf{w})$ has all nonnegative eigenvalues, for all $\mathbf{w}$.

Here $\nabla^2 f(\mathbf{w})$ denotes the Hessian matrix:

$$\nabla^2 f(\mathbf{w}) = \left[\frac{\partial^2 f}{\partial w_i \partial w_j} \right].$$

Minimizing a convex function (over a convex set) can be carried out very efficiently even for very large-sized problems.

Outline

IIT Hyderabad

Basic Assumption

Suppose are given labelled samples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, where $\mathbf{x}_i \in \mathbb{R}^n$, $y_i \in \{0, 1\}$. (Note the change of the "label space" from $\{-1, 1\}$ to $\{0, 1\}$.)

Suppose we have a model where the probability is a function of some parameters $\boldsymbol{\beta}$, so that

$$\Pr\{\mathbf{x} \sim 1\} = p(\boldsymbol{\beta}, \mathbf{x}),$$

which implies that

$$\Pr\{\mathbf{x} \sim 0\} = 1 - p(\boldsymbol{\beta}, \mathbf{x}).$$

Forming the Function to be Maximized

Given the training data, we choose the parameters $\boldsymbol{\beta}$ so as to maximize the **log-likelihood**

$$J = \sum_{y_i=1} \log p(\boldsymbol{\beta}, \mathbf{x}_i) + \sum_{y_1=0} \log(1 - p(\boldsymbol{\beta}, \mathbf{x}_i)).$$

Interpretation: Choose the parameter vector $\boldsymbol{\beta}$ so as to maximize the probability that each training sample has the correct label.

We can rewrite the function to be maximized as

$$J = \sum_{i=1}^{m} y_i \log p(\boldsymbol{\beta}, \mathbf{x}_i) + (1 - y_i) \log(1 - p(\boldsymbol{\beta}, \mathbf{x}_i)).$$

Standard Sigmoid: Reprise

Recall the standard sigmoid function:

$$\sigma(z) = \frac{1}{1+\exp(-z)}.$$

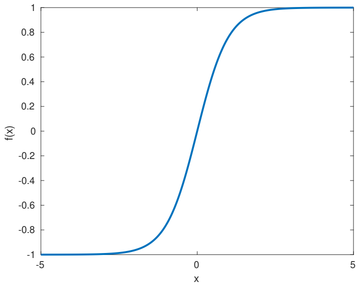


Figure: Standard Sigmoid Function

Sigmoidal Probability

In principle we can use whatever expression we want for $p(\boldsymbol{\beta}, \mathbf{x})$. But a useful choice is

$$p(\boldsymbol{\beta}, \mathbf{x}) = \frac{1}{1 + \exp(-z)},$$

where z has the form

$$z = \sum_{i=1}^{n} w_i x_i + b.$$

Advantage of this formulation:

$$1 - p(\boldsymbol{\beta}, \mathbf{x}) = 1 - \frac{1}{1 + \exp(-z)} = \frac{\exp(-z)}{1 + \exp(-z)} = \frac{1}{1 + \exp(z)}.$$

Augmenting the Input Vectors

We can think of b as the "bias," and $b = -\theta$ which is the "threshold" in the SVM.

The formulation can be streamlined by augmenting each $\mathbf{x}$ by 1, so that

$$\bar{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}, \boldsymbol{\beta} = \bar{\mathbf{w}} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix},$$

so that we have

$$z = \bar{\mathbf{w}}^\top \bar{\mathbf{x}}.$$

Training the Logistic Regressor

Given labelled inputs $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $\mathbf{x}_i \in \mathbb{R}^m, y_i \in \{0,1\}$, we augment each $\mathbf{x}_i$ into $\bar{\mathbf{x}}_i \in \mathbb{R}^{n+1}$, and then choose $\bar{\mathbf{w}} \in \mathbb{R}^{n+1}$ so as to *maximize*

$$J = \sum_{i=1}^m y_i \log p(\bar{\mathbf{w}}, \bar{\mathbf{x}}_i) + (1 - y_i) \log(1 - p(\bar{\mathbf{w}}, \bar{\mathbf{x}}_i)),$$

or equivalently, to *minimize*

$$-J = - \left[\sum_{i=1}^m y_i \log p(\bar{\mathbf{w}}, \bar{\mathbf{x}}_i) + (1 - y_i) \log(1 - p(\bar{\mathbf{w}}, \bar{\mathbf{x}}_i)) \right].$$

Convexity of the Objective Function

The reason why logistic regression can be carried out for really large problems is that the objective function is a *convex* function of the parameter vector $\bar{\mathbf{w}}$.

Since a nonnegative combination of convex functions is convex, and since $y_i, 1 - y_i \geq 0$ for all i, it is enough to show that the functions $-\log p(\bar{\mathbf{w}}, \bar{\mathbf{x}})$ and $1 - \log p(\bar{\mathbf{w}}, \bar{\mathbf{x}})$ are convex in $\bar{\mathbf{w}}$ for fixed $\mathbf{x}$.

Convexity of the Objective function

Note that

$$-\log p(\bar{\mathbf{w}}, \bar{\mathbf{x}}) = -\log \left(\frac{1}{1 + e^{-\bar{\mathbf{w}}^\top \bar{\mathbf{x}}}} \right) = \log[1 + e^{-\bar{\mathbf{w}}^\top \bar{\mathbf{x}}}] =: h_1(\bar{\mathbf{w}}, \bar{\mathbf{x}}) \text{ say.}$$

For simplicity, assume w, x are scalars. We will show that $-\log p = -\log(w, x)$ is convex by showing that

$$\frac{\partial^2 h_1(w,x)}{\partial w^2}(w,x) \geq 0, \ \forall w.$$

Convexity of the Objective Function (Cont'd)

$$\frac{\partial h_1(w,x)}{\partial w} = -\frac{1}{1+e^{-wx}} \cdot xe^{-wx} = -\frac{x}{e^{wx}+1},$$
$$\frac{\partial^2 h_1(w,x)}{\partial w^2} = \frac{x^2}{(e^{wx}+1)^2} \geq 0, \ \forall w, x.$$

Similarly $-\log[1-p(w,x)]$ is also convex in w.

So the objective function $-J$ is convex in w.

A similar argument works in higher dimensions.

Use of Logistic Regression for Classifying Test Inputs

Suppose $\mathbf{w}$ is chosen to optimize the overall LLR. Now suppose a new test input $\mathbf{x}$ is presented. We assign it labels as follows:

$$\Pr\{y = 1\} = p(\bar{\mathbf{w}}, \bar{\mathbf{x}}) = \frac{1}{1 + \exp(-\bar{\mathbf{w}}^\top \bar{\mathbf{x}})},$$

$$\Pr\{y = 0\} = 1 - p(\bar{\mathbf{w}}, \bar{\mathbf{x}}) = \frac{1}{1 + \exp(\bar{\mathbf{w}}^\top \bar{\mathbf{x}})}.$$

The two probabilities add up to one, and give us a *confidence value* associated with our classification.

Tips on Using Logistic Regression

- The `Matlab` command `glmfit` solves the logistic regression problem with input vectors $\mathbf{x}_1, \ldots, \mathbf{x}_m$ and binary labels $y_1, \ldots, y_n$.
- Form

$$X = \begin{bmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_m \end{bmatrix}, y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}.$$

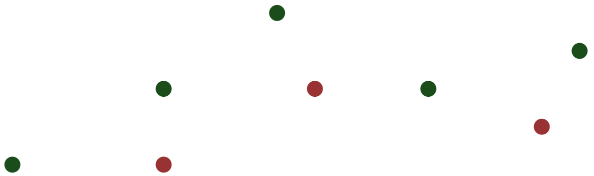
- Write out each $\mathbf{x}_i$ as a row vector.
- The command

```
b = glmfit(X,y,'binomial','link','logit')
```

gives the optimal weight vector $(-\theta, \mathbf{w})$. (Note the minus sign in front of θ.)

Outline

Non-Separable Data Set



Setting Up the Problem

The `glmfit` command in `Matlab` permits us to run logistic regression. Input the vectors $\mathbf{x}_i$ as rows and y_i as a label. In our case

$$X = \begin{bmatrix} 0 & 0 \\ 3.5000 & 2.0000 \\ 5.5000 & 1.0000 \\ 2.0000 & 1.0000 \\ 7.5000 & 1.5000 \\ 2.0000 & 0 \\ 4.0000 & 1.0000 \\ 7.0000 & 0.5000 \end{bmatrix}, y = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

Solution Using `glmfit`

$$\begin{bmatrix} -\theta \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 0.3044 \\ -0.6920 \\ 3.9391 \end{bmatrix}.$$

Probability of Each Sample Having $y = 1$.

The probability that each $\mathbf{x}_i$ has label 1 is given by

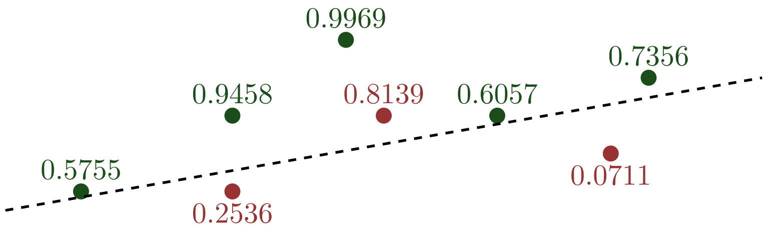
$$p = \frac{1}{1 + \exp(-(w_1 x_1 + w_2 x_2 - \theta))}.$$

In our case

$$\mathbf{p} = \begin{bmatrix} 0.5755 \\ 0.9969 \\ 0.6077 \\ 0.9458 \\ 0.7356 \\ 0.2536 \\ 0.8139 \\ 0.0711 \end{bmatrix}.$$

IIT Hyderabad

Separating Surface Using Logistic Regression



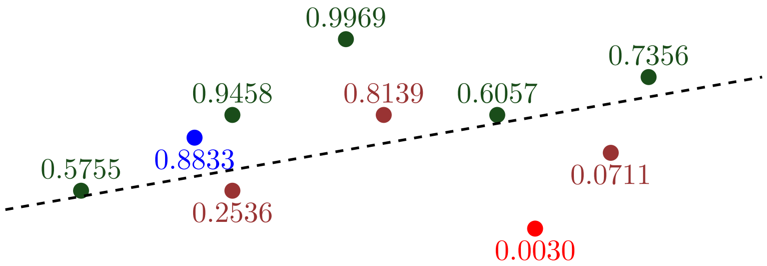
Note: If data is linearly separable, the weights would approach infinity.

Classification of Test Inputs

Given a new test input $\mathbf{z}$,

- Form an augmented vector $\bar{z} = [1 \;\; \mathbf{z}^\top]$.
- Compute $h = \bar{z}b$ (where b comes from the logistic regression).
- Assign $\mathbf{z}$ to the class $y = 1$ with probability $p = 1/(1 + e^{-h})$.

Classification of Test Inputs (Depiction)



The two test inputs are shown in blue and red.

Takeaways From Example

- Logistic regression is not guaranteed to classify all training samples correctly.
- The Boolean label of each training sample has an associated probability.
- When a test sample is assigned a Boolean label, there is once again an associated probability.

Outline

Outline

Ordinal vs. Cardinal Labels

Mutli-Class Classification: Suppose we are given samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$, where $\mathbf{x}_i \in \mathbb{R}^n$, and $y_i \in Y$, a finite set of cardinality ≥ 3. We wish to design a classifier that assigns each test sample unambiguously to one of the elements of Y.

We need to design multiple classifiers, and we need to distinguish between "ordinal" and "cardinal" (also called "categorical") label sets.

The set Y is said to be **ordinal** if there is a natural ordering, and **cardinal (or categorical)** otherwise.

Examples of Ordinal and Cardinal Labels

Suppose wish to describe the prospects of a student passing a course (or a patient recovering). We can assign the prospects into five categories: P (poor), F (Fair), G (Good), V (Very Good), E (Excellent).

Clearly this set is *ordered*, with

$$P \prec F \prec G \prec V \prec E.$$

In contrast, recognizing handwritten characters requires classification into $\{0, \ldots, 9\}$. The natural ordering of integers *does not apply here.* Another example is grouping breast cancer patients into subtypes.

Outline

IIT Hyderabad

Approaches to Multi-Class Classification

By their nature, classifiers work best for *binary* classifiers. So the question becomes: How can multi-class classification be turned into a collection of binary classification problems?

Suppose $|Y| = k$. The idea is to define k distinct (confidence) functions $f_i, i = 1, \ldots, k$. Given a test sample $\mathbf{x}$, it is assigned to class i if

$$f_i(\mathbf{x}) \geq f_j(\mathbf{x}), \ \forall j \neq i.$$

There are several popular methods for generating these confidence functions f_i.

The Code Matrix

Given the set of labels Y (ordinal or cardinal), we associate with it a **code matrix** C that has $|Y|$ rows and l columns.

- Each element of C belongs to $\{-1, 0, 1\}$.
- For each column of C, we construct a binary classifier, to differentiate between the samples with label -1 from the samples labelled $+1$. The samples labelled 0 are ignored.
- Each column gives a binary (or bipolar) classifier with a discriminant function Δ_i.
- These are combined into $|Y|$ different functions $f_1, \ldots, f_{|Y|}$.

Data Replication Method for Ordinal Labels

This method is well-suited for ordinal labels. If there are k labels, then there are k rows and $k-1$ columns, and the j-th classifier discriminates between those samples bearing labels $\{1, \ldots, j\}$ and those bearing labels $\{j+1, \ldots, k\}$. For example, for a four-class problem, the code matrix is

$$\begin{array}{ccccc} & & \Delta_1 & \Delta_2 & \Delta_3 \\ \text{Class 1} & \left(\begin{array}{c} \\ \\ \\ \\ \end{array}\right. & +1 & +1 & +1 \\ \text{Class 2} & & -1 & +1 & +1 \\ \text{Class 3} & & -1 & -1 & +1 \\ \text{Class 4} & & -1 & -1 & -1 \end{array}$$

Data Replication Method (Contd')

The functions f_i are defined as follows:

$$f_i(\mathbf{x}) = \begin{cases} \Delta_1(\mathbf{x}) & \text{if } i = 1 \\ \Delta_i(\mathbf{x}) - \Delta_{i-1}(\mathbf{x}) & \text{if } i \in \{2, \ldots, k-1\} \\ -\Delta_{k-1}(\mathbf{x}) & \text{if } i = k \end{cases}$$

Reminder: A sample $\mathbf{x}$ is assigned the label i if

$$f_i(\mathbf{x}) \geq f_j(\mathbf{x}), \; \forall j \neq i.$$

One vs. Rest Method

If $|Y| = k$, then the j-th row of the code matrix has a $+1$ in row j and a -1 in the others. So the j-th binary classifier discriminates between "in class j" and "not in class j".

For four classes, the code matrix is

$$\begin{array}{c c} & \begin{array}{cccc} \Delta_1 & \Delta_2 & \Delta_3 & \Delta_4 \end{array} \\ \begin{array}{c} \text{Class 1} \\ \text{Class 2} \\ \text{Class 3} \\ \text{Class 4} \end{array} & \left(\begin{array}{cccc} +1 & -1 & -1 & -1 \\ -1 & +1 & -1 & -1 \\ -1 & -1 & +1 & -1 \\ -1 & -1 & -1 & +1 \end{array} \right) \end{array}$$

We take $f_i(\mathbf{x}) = \Delta_i(\mathbf{x})$.

One vs. One Method

In this method, the number of columns l equals "k choose 2," that is, all possible pairs. So with $k = 4$, there are 6 columns, as shown below:

$$\begin{array}{l c} & \begin{array}{cccccc} \Delta_{12} & \Delta_{13} & \Delta_{14} & \Delta_{23} & \Delta_{24} & \Delta_{34} \end{array} \\ \begin{array}{l} \text{Class 1} \\ \text{Class 2} \\ \text{Class 3} \\ \text{Class 4} \end{array} & \left(\begin{array}{rrrrrr} +1 & +1 & +1 & 0 & 0 & 0 \\ -1 & 0 & 0 & +1 & +1 & 0 \\ 0 & -1 & 0 & -1 & 0 & +1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{array} \right) \end{array}$$

With the convention that $b_{ji} = -b_{ij}$, the functions f_i are

$$f_i(\mathbf{x}) = \sum_{j \neq i} b_{ij}(\mathbf{x}).$$