CSCE 970: Pattern Recognition	Spring Semester, 1999

Topic 4: Non-Linearly Separable Classes and Nonlinear Classifiers

Lecturer: Stephen Scott | *Scribe: Hongling Wang*

1 Introduction

Perceptron Algorithm and Winnow/Exponentiated Gradient are good ways to find classifier for lineary separable classes. But, in practice, some classes are not linearly separable. In this topic, we first see some ideas about how to classify the non-linearly separable classes, then, we give how to use multiple layers of perceptrons to build nonlinear classifiers for non-linearly separable classes.

2 Non-linearly Separable Classes

If a hyperplane exists to completely separate the classes, we can use perceptron or Winnow/Exponentiated Gradient to build a linear classifier. In many cases, no such hyperplane exists to separate the classes, so we can't use these algorithms directly. There are several ways in which we can deal with these cases.

2.1 Add Extra Inputs

A basic requirement for the convergence of the perceptron algorithm is the linear separability of the classes. If this is not true, the perceptron algorithm doesn't converge. If the classes are not linearly separable under the original input, we can add extra inputs to get a new feature vector spaces. For example, if the classes are not linearly separable in feature vector space $[x_1, x_2]$, we can try the following feature vector space,

$$\mathrm{X} = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

Then, maybe the classes are linearly separable in the new feature space. This method is especially useful with Winnow's logarithmic bounds.

2.2 Winnow's agnostic results for non-linear separable classes

For non-linearly separable classes, we can still use Winnow/Exponentiated Gradient algorithm to train the classifier. But, we will have to live with errors, that is, the true output will not always be equal to the desired ones after we get the best classifier from the training data.

However, in on-line setting, Winnow/Exponentiated Gradient's total number of prediction mistakes provably not much worse than best linear classifier. Mistake bound for Winnow/Exponentiated Gradient is related to total distance under $\|\cdot\|_1$ that feature vectors must be moved to make best classifier perfect. This is demonstrated in figure 1.

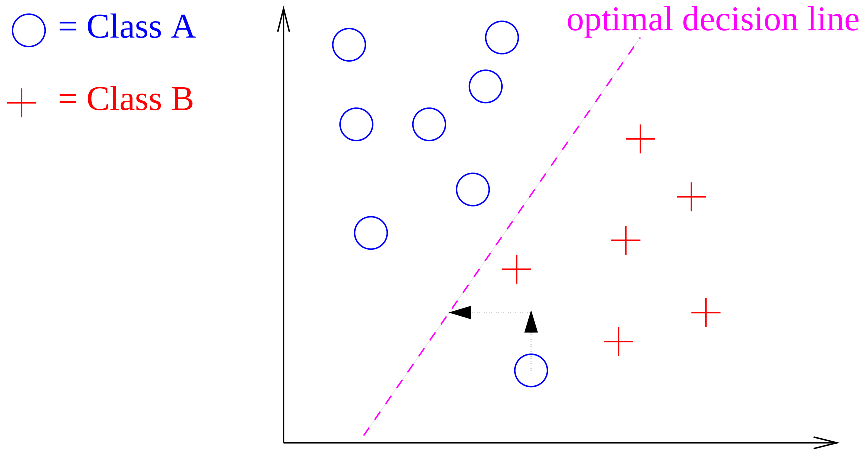

Figure 1: Mistake Bound for Winnow

2.3 Least Squares Methods

In perceptron algorithms, we train classifier according to the following method,

$$
\begin{aligned}
w_i(t+1) &= w_i(t) + \eta\left(y - \sum_{i=1}^{l} w_i(t)x_i(t)\right)x_i(t) \\
&= w_i(t) + \eta(y - \mathrm{w}(t)^T \cdot \mathrm{x}(t))x_i(t) \\
&= w_i(t) + \eta(y - \hat{y})x_i(t)
\end{aligned}
$$

If we don't threshold dot product during training and allow η to vary in each trial, we get the following Least Mean Square (LMS) Algorithm,

$$\mathrm{w}(t+1) = \mathrm{w}(t) + \eta_t \mathrm{x}(t)(y(t) - \mathrm{w}(t)^T \cdot \mathrm{x}(t))$$

We let η_t=$1/t$, apparently, if we train the weight vector in this way, then we have,

$$\lim_{t\to\infty} P(\mathrm{w}(t) = \mathrm{w}^*) = 1$$

Where,

$$\mathrm{w}^* = \arg\min\left\{\mathrm{E}\left[\left|y - \mathrm{w}^T \cdot \mathrm{x}\right|^2\right]\right\},\ \mathrm{w} \in \Re_l$$

is vector minimizing mean square error(MSE).

2.4 Multiclass learning: Kessler's Construction

So far we have dealt with the two-classes case. The generalizaion to an M-class task is straightforward. A linear discriminant function $w_i, i = 1, 2, \cdots, M$, is defined for each of the classes. A feature vector x (in the $(l+1)$-dimmesional space to account for threshold) is classified in class ω_i if

$$w_i^T x > w_j^T,\ \forall j \neq i$$

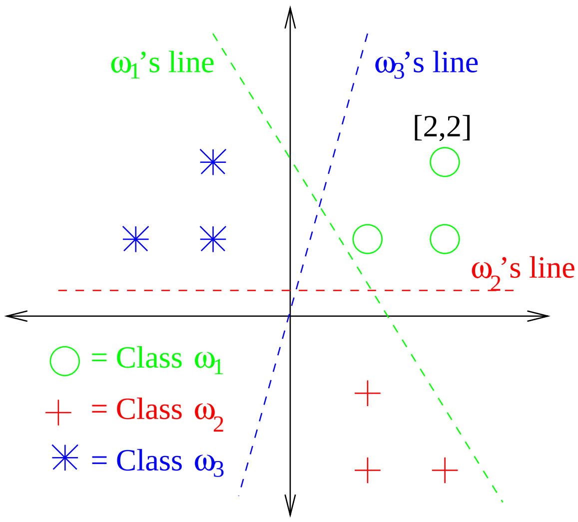

Figure 2: Multiclass

See the classes in the figue 2,

For a feature vector x=$[2,2,1]^T$ of class ω_1, we have,

$$\sum_{i=1}^{l+1} w_{1i}x_i > \sum_{i=1}^{l+1} w_{2i}x_i \text{ AND } \sum_{i=1}^{l+1} w_{1i}x_i > \sum_{i=1}^{l+1} w_{3i}x_i$$

We can map x to

$$\mathrm{x}_1 = [\overbrace{2,2,1}^{orig.}, \overbrace{-2,-2,-1}^{neg}, \overbrace{0,0,0}^{pad}]^T$$

$$\mathrm{x}_2 = [2,2,1,0,0,0,-2,-2,-1]^T$$

and let

$$\mathrm{w} = [\overbrace{w_{11}, w_{12}, w_{10}}^{\mathrm{w}_1}, \overbrace{w_{21}, w_{22}, w_{20}}^{\mathrm{w}_2}, \overbrace{w_{31}, w_{32}, w_{30}}^{\mathrm{w}_3}]^T$$

If we have w^{*T} . $\mathrm{x}_1 > 0$ and w^{*T} . $\mathrm{x}_2 > 0$, then,

$$\sum_{i=1}^{l+1} w^*_{1i}x_i > \sum_{i=1}^{l+1} w^*_{2i}x_i \text{ AND } \sum_{i=1}^{l+1} w^*_{1i}x_i > \sum_{i=1}^{l+1} w^*_{3i}x_i$$

In general, we can map $(l+1)\times 1$ feature vector x to $\mathrm{x}_1, \mathrm{x}_2, \cdots, \mathrm{x}_{M-1}$, each of size $(l+1)M \times 1$. For each of the training feature vector x$\in w_i$, we can put x in ith block and -x in jth block, the rest blocks are set to 0s. Repeat this procedure for each $j \neq i$. In this manner, we train the weight vector using each training feature vector and find weights for new feature vector space.

2.5 Multiclass learning: Error-Correcting Output Codes

Winnow and Perceptron algorithms learn only binary functions. When we have more than 2 classes, a single binary function is no longer enough. In these cases, we can encode the classes in binary code, then, learn individual bits of binary encoding of the classes.

For the number of classes M, we use at least $\lg M$ binary bits to encode the classes and then have a linear classifier for each of the binary bits. For example, if we have $M = 4$, we use 2 binary bits and two linear classifiers,

Class	Binary Encoding	
	Classifier 1	Classifier 2
ω_1	0	0
ω_2	0	1
ω_3	1	0
ω_4	1	1

We can train each of the classifiers simultaneously. The problem with this method is that it is sensitive to individual classifier errors. If one of the classifiers predicts wrong, then the prediction is wrong. In oder to improve robustness, we can use a set of encodings per class. This method is similar to principle of error-correcting output codes used in communication networks.

3 Nonlinear Classifiers

For non-linearly separable classes, performance of even the best linear classifier might not be good. One method is to add new features that are nonlinear combinations of old ones, but it is not easy to decide what combinations should be introduced, and also the number of new features grows quickly. In many cases when the classes are not linearly separable, if we map the feature vector space to a new feature vector space, maybe the classes are linearly separable in the new feature vector space. Go with this method further, we can use multiple layers of perceptron or winnow to remap the feature vetor space to one that is linearly separable.

3.1 The XOR problem

For the example in the Figure 3, we have 2 classes ω_1 and ω_2. Clearly, we can't use a single linear separator to separate the 2 classes. But, we can do that with intersection of the two following linear separator,

$$g_1(x) = 1 \cdot x_1 + 1 \cdot x_2 - 1/2$$

$$g_2(x) = 1 \cdot x_1 + 1 \cdot x_2 - 3/2$$

We use these two linear separators to separate w_1 and w_2,

$$\begin{aligned} w_1 = & \quad \{\mathrm{x} \in \Re^l : g_1(\mathrm{x}) > 0 \text{ AND } g_2(\mathrm{x}) < 0\} \\ w_2 = & \quad \{\mathrm{x} \in \Re^l : g_1(\mathrm{x}), g_2(\mathrm{x}) < 0 \text{ OR } g_1(\mathrm{x}), g_2(\mathrm{x}) > 0\} \end{aligned}$$

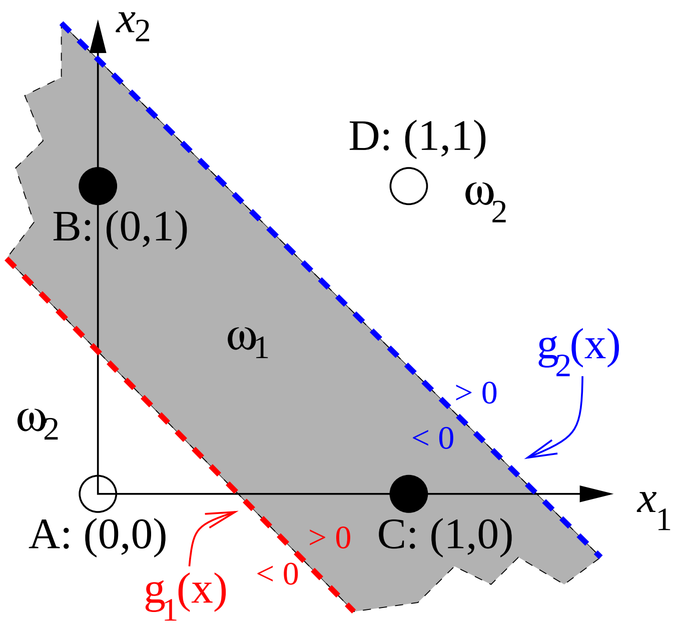

Figure 3: The XOR Problem

If let $y_i = \begin{cases} 0 & \text{if } g_i(\mathrm{x}) < 0 \\ 1 & \text{otherwise} \end{cases}$

We can get the following table for the above example,

Class	(x_1, x_2)	g_1 (x)	y_1	g_2 (x)	y_2
ω_1	B: (0,1)	1/2	1	-1/2	0
ω_1	C: (1,0)	1/2	1	-1/2	0
ω_2	A: (0,0)	-1/2	0	-3/2	0
ω_2	D: (1,1)	3/2	1	1/2	1

Then, we introduce a new feature vector y=$[y_1, y_2]$, and have the following hyperplane for the new feature vector,

$$g(\mathrm{y}) = 1 \times y_1 - 2 \times y_2 - 1/2$$

We can see the classes in the new feature space in the figure 4,
This hyperplane can linearly separate class ω_1 and class ω_2.
In the above example, we remapped vectors x to y such that the classes are linearly separable in the new vector space. We can represent this remapping process in the figure 5,
This example is just a two-layer perceptron or called two-layer feedforward neutral network. In this algorithm, we have a hidden layer. This hidden layer has two neurons. Each neuron outputs 1 if its weighted sum exceeds its threshold and 0 otherwise. In this way, we transform the original feature vector space to a new feature vector space in which the classes are linearly separable.

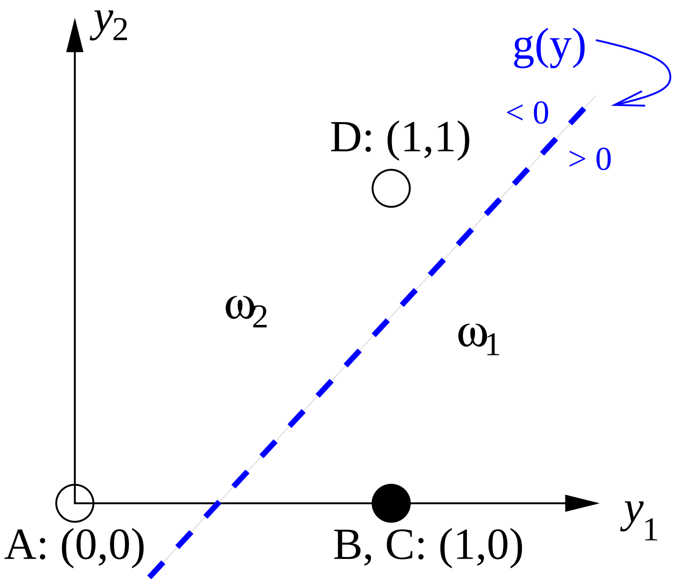

Figure 4: XOR Problem in New Feature Space

3.2 Further Analysis of Two-layer perceptron

In another example, we have classes in the figure 6,
By applying two-layer perceptron, we maps the classes to the following feature vector spaces in figure 7,
In general, we can define the p-dimensional unit hypercube as following,

$$H_p = \{[y_1, \cdots, y_p]^T \in \Re^p, y_i \in [0,1] \text{ for } \forall i\}$$

We define a hidden layer with p neurons using this hypercube. The hidden layer with p neurons maps an l-dimmesional vector x to a p-dimmensional vector y whose elements are corners of H_p. Each of the p neurons corresponds to an l-dimmensional hyperplance. The intersection of the positive or negtive half spaces from these p hyperplans corresponding to a specific class maps to a vertex of H_p. If a hyperplane in the p dimmensional space exists to linearly separate the classes of H_p's vertices, then a perfect two layer network exists. In this case, a 2-layer network can separate classes consisting of unions of certain polyhedra which are not linearly separable in the original feature space.

3.3 Three Layer Network

In the last section, we see the two-layer network which can separate classes consisting unions of certain polyhedra. With two-layer networks, there maybe exists unions of polyhedra not linearly separable on H_p. For the example in last section, if we include 110 in ω_1 using a procedure similar to XOR solution, the classes can't be linearly separated on H_p.

In this case, we can add a second hidden layer of q neutrons to partition H_p into regions based on class and get a three-layer perceptron. Here again, we assume that all regions of interest are formed by intersections of p l-dimmesional half-spaces defined by the p perceptrons.

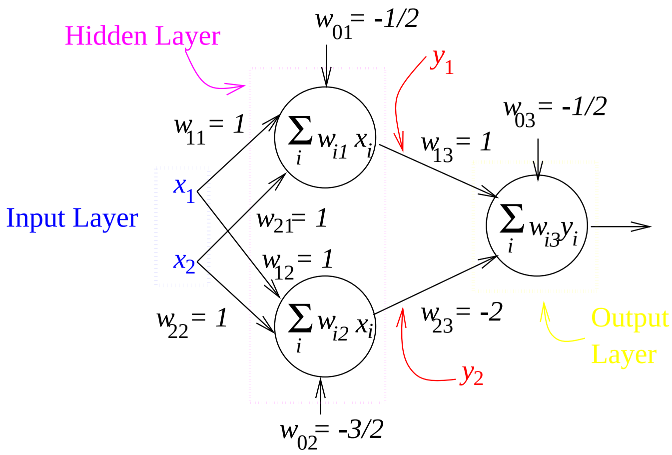

Figure 5: A Two-layer Perceptron

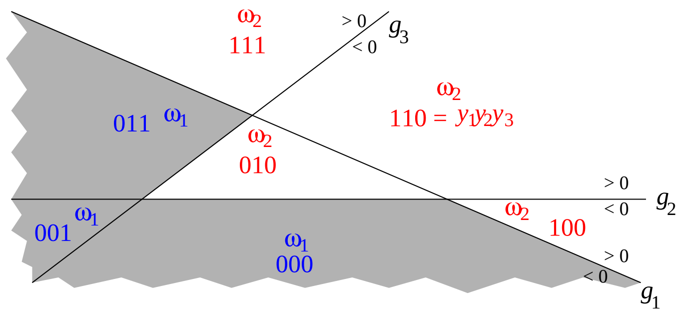

Figure 6: Classes Consisting of Unions of Certain Polyhedra

These are realized by the p neutrons of the first hidden layer, which also perform the mapping of the input space onto the vertices of the H_p hypercube of unit side length. In the second hidden layer, each of the neutrons realizes a hyperplane in the p-dimmensional space. Such an three-layer perceptron can separate classes resulting from any union of polyhedral regions.

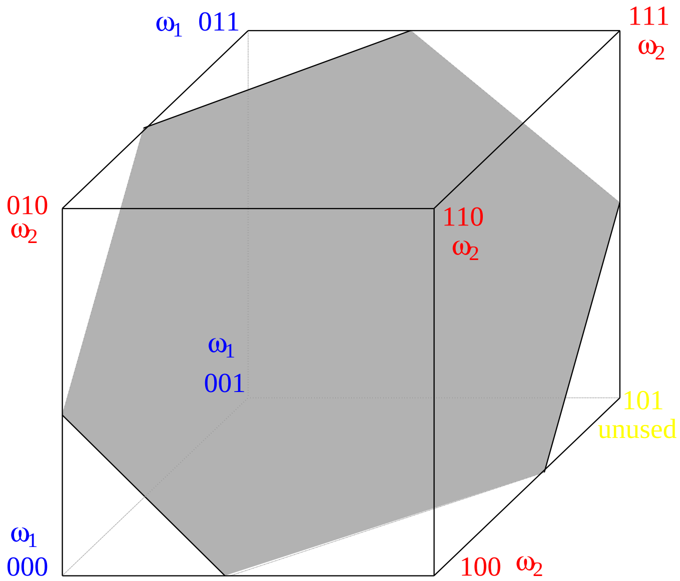

Figure 7: A 2-dimmensional Unit Hypercube