Nonlinear Feature Engineering

Instructor: Hui Guan

Slides adapted from: https://github.com/jermwatt/machine_learning_refined

Outline

- Nonlinear Learning: modeling principles
- Nonlinear multi-output regression
- Nonlinear two-class classification
- Nonlinear multi-class classification
- Nonlinear unsupervised learning

Modeling principles

In Chapter 5 we detailed the basic linear model for regression as

$$\text{model}\left(\mathbf{x}, \mathbf{w}\right) = w_0 + x_1 w_1 + \cdots + x_N w_N$$

or more compactly as $\text{model}\left(\mathbf{x}, \mathbf{w}\right) = \dot{\mathbf{x}}^T \mathbf{w}$, denoting

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \quad \text{and} \quad \dot{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

- Using a generic dataset of P input-output pairs $\{(\mathbf{x}_p, y_p)\}_{p=1}^{P}$ we then minimized a proper regression cost function, e.g., the Least Squares cost below in order to find optimal values for the parameters of our linear model (here, the vector $\mathbf{w}$).

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\dot{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

- We can move from linear to general nonlinear regression by swapping out the linear model $\text{model}\,(\mathbf{x}, \mathbf{w}) = w_0 + x_1 w_1 + \cdots + x_N w_N$ with a nonlinear one, for instance a single nonlinear function f that can be parameterized or unparameterized.

$$\text{model}\,(\mathbf{x}, \Theta) = w_0 + f_1(\mathbf{x})\, w_1 + f_2(\vec{x})\, w_2 + \cdots + f_B(\vec{x})\, w_B$$

- In the jargon of machine learning ‘f’ is often called a nonlinear *feature transformation*.
- Here the set Θ represents all tunable parameters - both those potentially internal parameters of the function ‘f’ and those weights in the linear combination.

- In general we could create a nonlinear model that is the weighted sum of B nonlinear functions of our input as

$$\text{model}\left(\mathbf{x}, \Theta\right) = w_0 + f_1\left(\mathbf{x}\right) w_1 + f_2\left(\mathbf{x}\right) w_2 + \cdots + f_B\left(\mathbf{x}\right) w_B$$

- Here $f_1, f_2, \cdots f_B$ are nonlinear parameterized or unparameterized functions - or *feature transformations*.
- Once again w_0 through w_B (along with any additional weights internal to the nonlinear functions) are represented in the weight set Θ and must be tuned properly.

- In general we could create a nonlinear model that is the weighted sum of B nonlinear functions of our input as

$$\text{model}\left(\mathbf{x}, \Theta\right) = w_0 + f_1\left(\mathbf{x}\right) w_1 + f_2\left(\mathbf{x}\right) w_2 + \cdots + f_B\left(\mathbf{x}\right) w_B$$

- Here $f_1, f_2, \ldots f_B$ are nonlinear parameterized or unparameterized functions - or *feature transformations*.
- Once again w_0 through w_B (along with any additional weights internal to the nonlinear functions) are represented in the weight set Θ and must be tuned properly.

- In analogy to the linear case, here we too can compactly denote the generic nonlinear model by taking a 1 on top of the vector of nonlinear feature transformation as

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_B \end{bmatrix} \qquad \underset{\sim}{\dot{\mathbf{f}}} = \begin{bmatrix} 1 \\ f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_B(\mathbf{x}) \end{bmatrix} \qquad \dot{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

- With this notation we can write our generic nonlinear model compactly as

$$\text{model}\left(\mathbf{x}, \Theta\right) = \dot{\mathbf{f}}^{T}\mathbf{w}$$

model (X, W) = $\dot{X}^T W$

- To tune the parameters of our general nonlinear model we minimize a proper regression cost function over Θ , e.g., the Least Squares cost

$$g\left(\Theta\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\dot{\mathbf{f}}_{p}^{T}\mathbf{w} - y_p\right)^2$$

Q1: How to determine the value of B?
Q2: How to determine the f function to use?

① polynomial func ② NN ③ tree

$$\dot{f} = \begin{bmatrix} 1 \\ f_1(\vec{x}) \Rightarrow x_1^d \\ f_2(\vec{x}) \Rightarrow x_1 x_2^{d-1} \\ \vdots \\ f_B(\vec{x}) \end{bmatrix}$$

- With this notation we can write our generic nonlinear model compactly as

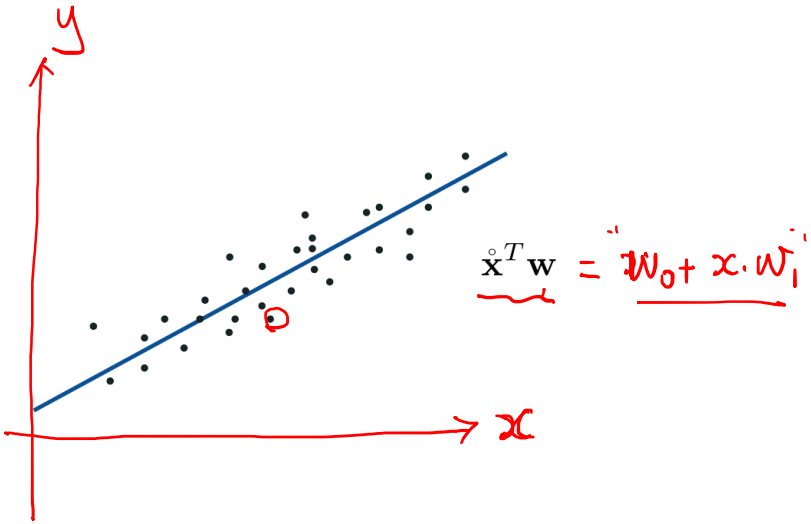
$$\text{model}\left(\mathbf{x}, \Theta\right) = \dot{\mathbf{f}}^T \mathbf{w}$$

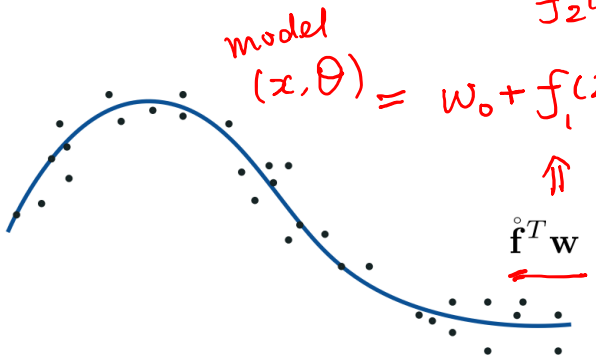
$\vec{x} \in \mathbb{R}^N \Rightarrow \in \mathbb{R}^B$

- To tune the parameters of our general nonlinear model we minimize a proper regression cost function over Θ , e.g., the Least Squares cost

$$g\left(\Theta\right) = \frac{1}{P} \sum_{p=1}^{P} \left(\dot{\mathbf{f}}_p^T \mathbf{w} - y_p \right)^2$$

Abstract illustration of linear versus nonlinear regression





$f_B(x) \cdot w_B$

$f_2(x) w_2 + \cdots$

model $(x, \Theta) = w_0 + f_1(x) w_1 +$

$B = 1$

$f = \sin(\alpha_0 + \alpha_1 \cdot x)$

$\Theta = [\alpha_0, \alpha_1, w_0, w_1]$

Q1: what is B?
Q2: What is f?

$$\text{model}(\mathbf{x}, \Theta) = w_0 + f_1(\mathbf{x}) w_1 + f_2(\mathbf{x}) w_2 + \cdots + f_B(\mathbf{x}) w_B$$

$\text{model}(x, \Theta) = w_0 + \sin(\alpha_0 + \alpha_1 x) \cdot w_1$

Generic Python Implementation

```
# an implementation of our model employing a
# general nonlinear feature transformation
def model(x, theta):

    # feature transformation
    f = feature_transforms(x, theta[0])

    # compute linear combination and return
    a = theta[1][0] + np.dot(f.T, theta[1][1:])

    return a.T
```

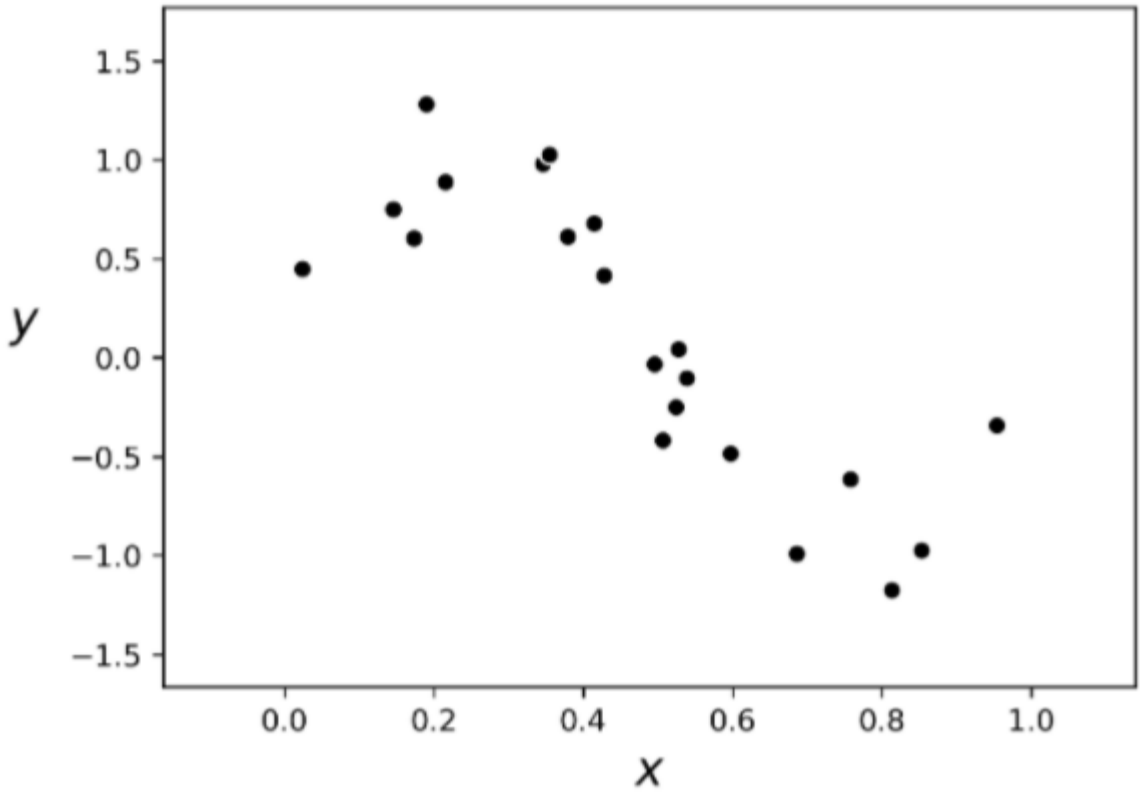
Feature engineering

How do we determine the appropriate nonlinear feature transformations for our model, and their number B?

- In some simple instances we can determine these by **visualizing the data** and relying on our own pattern recognition abilities to determine the appropriate nonlinearities.
- This kind of practice is referred to as **nonlinear feature engineering**.

nonlinear feature learning

Example: Modeling a familiar wave using a parameterized feature transformation


1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
0.0
0.2
0.4
0.6
0.8
1.0
x

- This dataset looks sinusoidal, so we can defensibly propose a `model` consisting of completely *parameterized* sinusoidal function or *feature transformation*

$$f\left(x, \mathbf{w}\right) = \sin\left(w_0 + xw_1\right)$$

- We can then take as our model a linear combination of this nonlinear feature transformation as

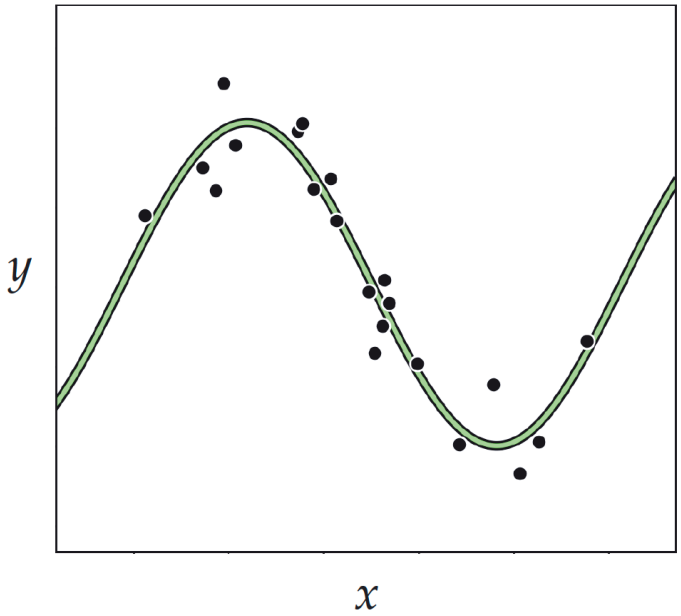
$$\text{model}\left(x, \Theta\right) = w_2 + f\left(x, \mathbf{w}\right) w_3 \, .$$

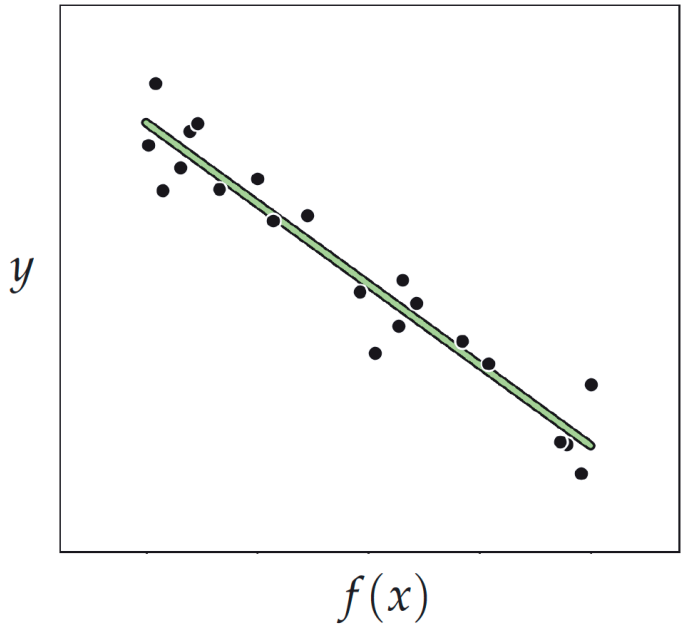
```
def feature_transforms(x, w):

    # compute feature transformation
    f = np.sin(w[0] + np.dot(x.T, w[1:])).T

    return f
```

Tuning the model parameters by minimizing a Least Squares cost via gradient descent, produces the following fit to our original data.





model(x, θ) = $\vec{f}^T \cdot w$

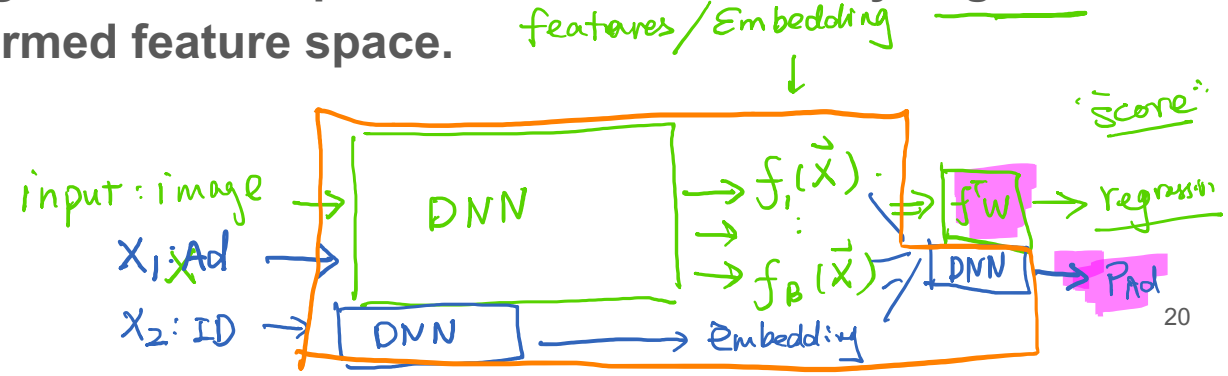
- Notice, with our model completely tuned if plot the points

$$\left(f\left(x_1, \mathbf{w}^\star\right), y_1\right), \left(f\left(x_2, \mathbf{w}^\star\right), y_2\right) \ldots, \left(f\left(x_P, \mathbf{w}^\star\right), y_P\right)$$

  our model fits the transformed data *linearly*.

- This finding is true in general with nonlinear regression problems.

- **A properly designed feature (or set of features) provides a good nonlinear fit in the original feature space and, simultaneously, a good linear fit in the transformed feature space.**



10.3 Nonlinear Multi-Output Regression

Q: how to make it non-linear?

- When dealing with **linear multi-output regression**, we have N dimensional input / C dimensional output pairs $(\mathbf{x}_p, \mathbf{y}_p)$, and our joint linear model for all C regressions takes the form

$$\text{model}\,(\mathbf{x}, \mathbf{W}) = \dot{\mathbf{x}}^T \mathbf{W}$$

- Here the weight matrix $\mathbf{W}$ has dimension $(N+1) \times C$

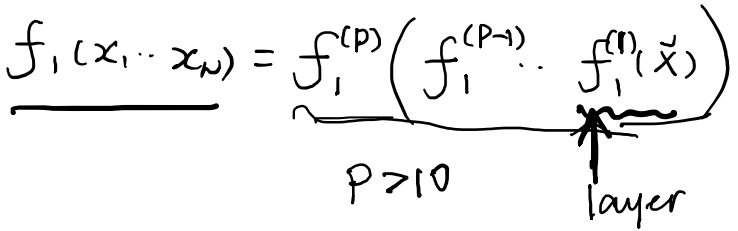
$\dot{X} \in \mathbb{R}^{(N+1) \times P}$

nonlinear regression : $\text{model}(X, \Theta) = \dot{f}^T W$

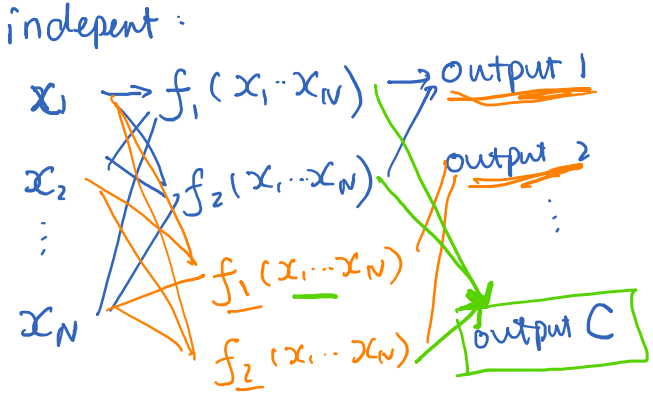
- We can tune the parameters of this joint model *one column at a time* by solving each linear regression ***independently***.
- We can also tune the parameters of $\mathbf{W}$ ***simultaneously*** by minimizing an appropriate regression cost function over the entire matrix, e.g., the Least Square

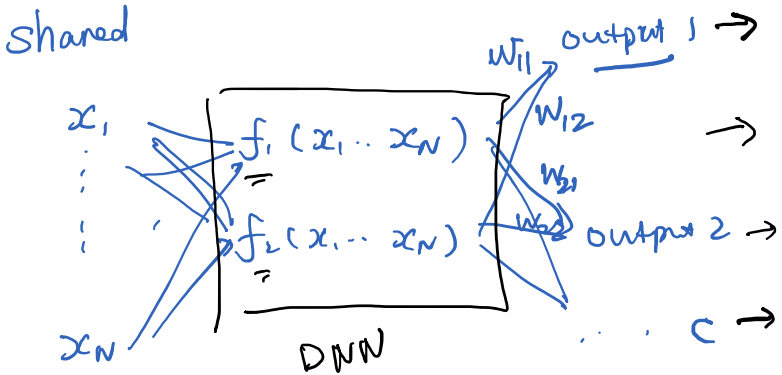
$$g\left(\mathbf{W}\right) = \frac{1}{P}\sum_{p=1}^{P}\left\|\dot{\mathbf{x}}_p^T\mathbf{W} - \mathbf{y}_p\right\|_2^2$$

- Because this model is *linear* the results of either approach result in the same tuning of the parameters.



- With multi-output regression the move from linear to nonlinear modeling mirrors what we have seen previously -- with one small but important wrinkle.
- With C regressions to perform we can choose to either produce C *__independent__* nonlinear models or one *__shared__* nonlinear model for all C regressions.





The independent approach

- If we choose the independent approach - forming C separate nonlinear models - each feature engineering / nonlinear regression is executed precisely as we have seen previously.
- That is, for the c^{th} regression problem we construct a model using (in general) B_c nonlinear feature transformations as

$$\text{model}_c(\mathbf{x}, \Theta_c) = w_{c,0} + f_{c,1}(\mathbf{x})\, w_{c,1} + f_{c,2}(\mathbf{x})\, w_{c,2} + \cdots + f_{c,B_c}(\mathbf{x})\, w_{c,B_c}$$

The feature-sharing approach

- With the feature-sharing approach to nonlinear multi-output regression, we simply use ***the same nonlinear features*** for all C models.
- This is very often done to simplify the chores of nonlinear feature engineering.
- If we choose the same set of B nonlinear features for all C models the c^{th} of which looks like

$$\text{model}_c(\mathbf{x}, \Theta_c) = w_{c,0} + f_1(\mathbf{x})\, w_{c,1} + f_2(\mathbf{x})\, w_{c,2} + \cdots + f_B(\mathbf{x})\, w_{c,B}$$

Employing the same compact notation for our feature transformations as introduced previously we can express each of these models more compactly as

$$\text{model}_c\left(\mathbf{x}, \Theta_c\right) = \dot{\mathbf{f}}^T \mathbf{w}_c.$$

Alternatively we can formulate all C models together in one model $\text{model}\,(\mathbf{x}, \Theta) = \dot{\mathbf{f}}^T \mathbf{W}$ by stacking the linear combination weights from our C nonlinear models into an $(N+1) \times C$ array of the form

$$\mathbf{W} = \begin{bmatrix} w_{0,0} & w_{0,1} & w_{0,2} & \cdots & w_{0,C-1} \\ w_{1,0} & w_{1,1} & w_{1,2} & \cdots & w_{1,C-1} \\ w_{2,0} & w_{2,1} & w_{2,2} & \cdots & w_{2,C-1} \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ w_{B,0} & w_{B,1} & w_{B,2} & \cdots & w_{B,C-1} \end{bmatrix}$$

Note here that the set Θ contains the linear combination weights $\mathbf{W}$ as well as any parameters internal to our feature transformations.

To tune the weights of our joint model we minimize an appropriate regression cost over the parameters in Θ , e.g., the Least Squares

$$g(\Theta) = \frac{1}{P}\sum_{p=1}^{P}\left\|\dot{\mathbf{f}}_p^T\mathbf{W} - \mathbf{y}_p\right\|_2^2$$

feature
-shared

Q: Can we optimize the parameter for one regression output at a time?

$y_1 \cdots y_C$

$\min g_c(\Theta) = \frac{1}{P}\sum \| f_p^T \cdot w_c - y_p \|_2^2$

$w_c = W[c]$

$\Theta = w_c \| w_f$

$f_p(\vec{x} | w_f)$

- If feature transformations used contain *__no internal parameters__*, e.g., if they are polynomial functions, then each of the C individual regression models can be tuned separately.
- Otherwise **when employing *parameterized* features (e.g., neural networks) we must tune all of our model parameters *jointly*, that is learn all C regressions *simultaneously*.**

10.4 Nonlinear Two-Class Classification

- Linear two-class classification uses the linear model

$$\text{model}\left(\mathbf{x}, \mathbf{w}\right) = \dot{\mathbf{x}}^T\mathbf{w}$$

- The linear decision boundary - employing by default label values $y_p \in \{-1, +1\}$ then lies precisely where $\dot{\mathbf{x}}^T\mathbf{w} = 0$
- And label predictions are made as

$$y = \text{sign}\left(\dot{\mathbf{x}}^T\mathbf{w}\right)$$

- To tune the set of weights properly for a given dataset we must minimize a proper two-class classification cost function, e.g., the Softmax/Cross-Entropy cost given by

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1 + e^{-y_p\dot{\mathbf{x}}_p^T\mathbf{w}}\right)$$

- While we previously employed a *linear* model in deriving linear two-class classification schemes, we could have just as well use any ***nonlinear*** model of the general form

$$\text{model}\left(\mathbf{x}, \Theta\right) = w_0 + f_1\left(\mathbf{x}\right) w_1 + f_2\left(\mathbf{x}\right) w_2 + \cdots + f_B\left(\mathbf{x}\right) w_B$$

- Here $f_1, f_2, \ldots f_B$ are nonlinear parameterized or unparameterized functions - or *feature transformations*

- Once again, w_0 through w_B (along with any additional weights internal to the nonlinear functions) are represented in the weight set Θ and must be tuned properly.

- We can express this nonlinear model - using the same compact notation introduced previously - as

$$\text{model}\left(\mathbf{x}, \Theta\right) = \dot{\mathbf{f}}^T \mathbf{w}$$

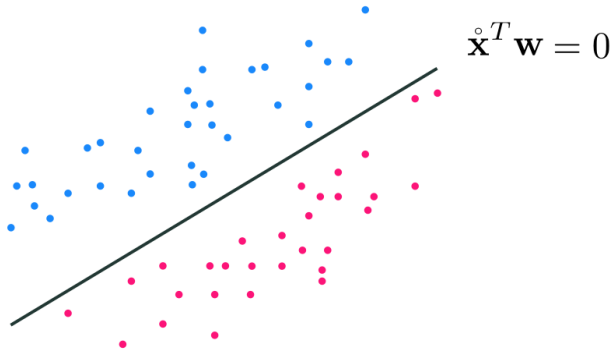
- In complete analogy to the linear case, here our decision boundary consists of all inputs $\mathbf{x}$ where $\dot{\mathbf{f}}^T \mathbf{w} = 0$
- And, likewise predictions are made as

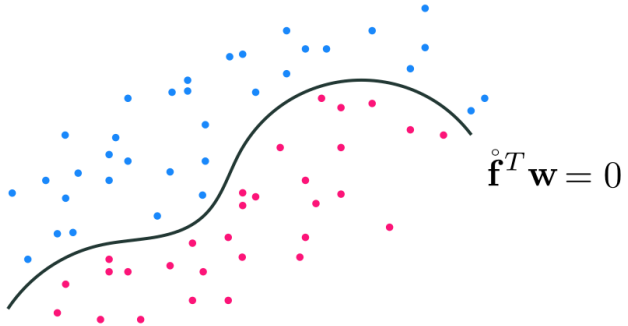
$$y = \text{sign}\left(\dot{\mathbf{f}}^T \mathbf{w}\right)$$

- Likewise, to properly tune the parameters of Θ we must minimize a proper cost function with respect to it, e.g., the Softmax cost

$$g\left(\Theta\right) = \frac{1}{P}\sum_{p=1}^{P}\log\left(1 + e^{-y_p\dot{\mathbf{f}}_p^T\mathbf{w}}\right)$$

- Abstract illustration of linear versus nonlinear two-class classification

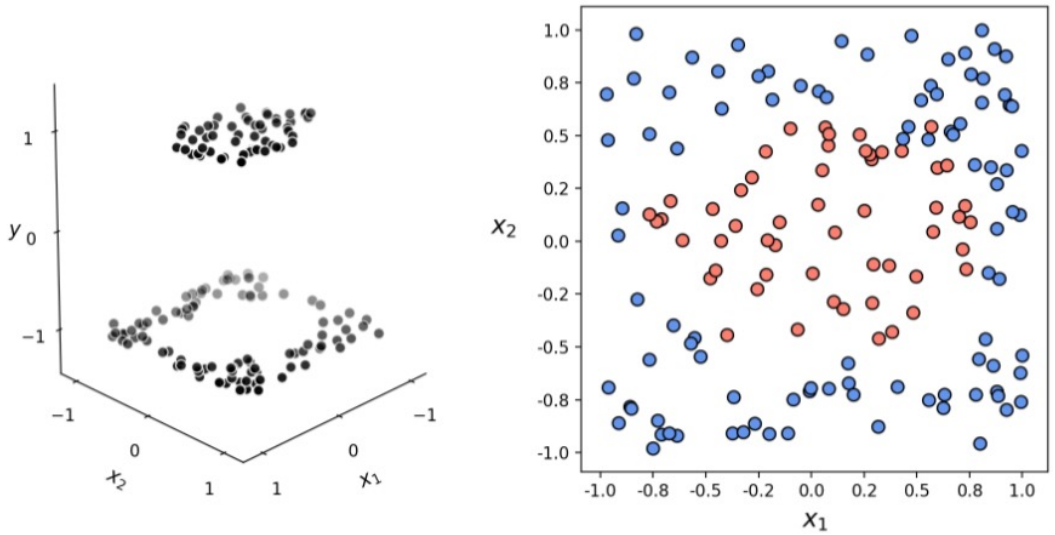




Example: Finding an elliptical boundary separating two classes

Q: what could be the nonlinear classification model?

In this example we examine the following classification dataset with N=2 dimensional input

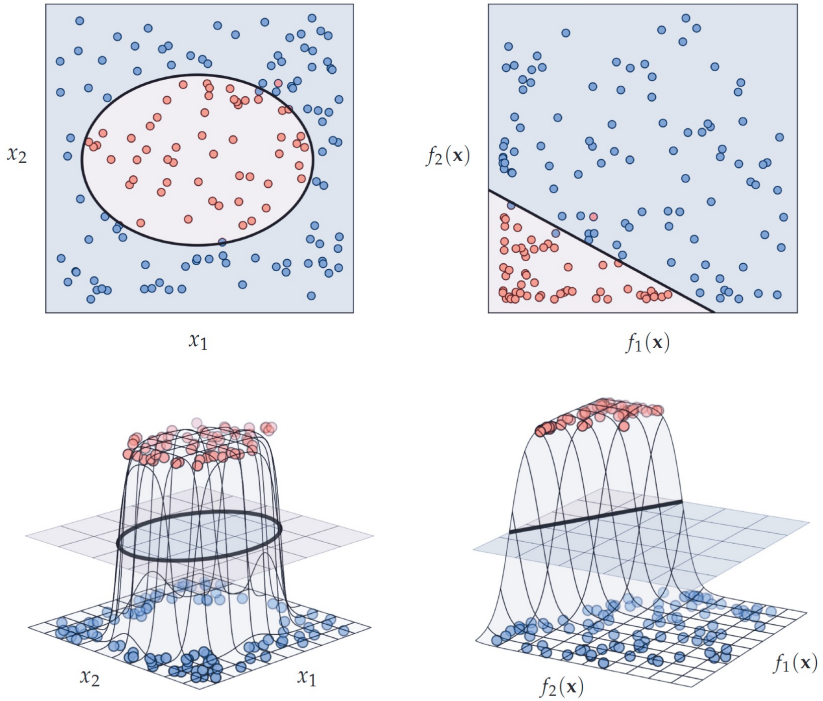


- Visually examining the dataset it appears that some sort of elliptical decision boundary centered at the origin might do a fine job of classification.
- Thus we set our `model` function to the general parameterized form of such an ellipse

$$\text{model}(\mathbf{x}, \Theta) = w_0 + x_1^2 w_1 + x_2^2 w_2$$

- Parsing this formula, we can see that we have used two feature transformations $f_1(\mathbf{x}) = x_1^2$, $f_2(\mathbf{x}) = x_2^2$, and the parameter set $\Theta = \{w_0, w_1, w_2\}$

With our weights tuned - by minimizing the Softmax cost via gradient descent - we can plot the data in its original space (left panels) along with the nonlinear decision boundary provided by the trained predictor, and in the transformed feature space (right panels) where the corresponding decision boundary is linear.



10.5 Nonlinear Multi-Class Classification

With mulit-class classification we have N input/output pairs $(\mathbf{x}_p,\, y_p)$ where by default our label values lie in the set $y_p \in \{0, 1, \ldots, C-1\}$

The joint linear model for all C classifiers takes the form

$$\text{model}\left(\mathbf{x}, \mathbf{W}\right) = \dot{\mathbf{x}}^T\mathbf{W}$$

Here the weight matrix $\mathbf{W}$ has dimension $(N+1) \times C$

Notice that this is precisely the same linear model used with multi-output regression.

We can safely wager that the ***nonlinear*** extension of multi-class classification takes a similar form to that discussed for multi-output regression.

With the parameters of the matrix $\mathbf{W}$ fixed we make predictions based on the *fusion rule*, denoting $\mathbf{w}_c$ as the c^{th} column of $\mathbf{W}$

$$y = \underset{c=0,..,C-1}{\text{argmax}} \ \dot{\mathbf{x}}^T \mathbf{w}_c$$

- We can tune the parameters of the joint model *one column at a time* by solving a sequence of C two-class subproblems *independently*: this is the One-versus-All approach.
- We can also tune the parameters of $\mathbf{W}$ *simultaneously* by minimizing an appropriate multi-class cost function over the entire matrix at once using e.g., the Multiclass Softmax/Cross-Entropy cost

$$g\left(\mathbf{W}\right) = \frac{1}{P}\sum_{p=1}^{P}\left[\log\left(\sum_{c=0}^{C-1} e^{\dot{\mathbf{x}}_p^T\mathbf{w}_c}\right) - \dot{\mathbf{x}}_p^T\mathbf{w}_{y_p}\right]$$

Similar to the linear case, in general we have two main avenues for moving to nonlinear multi-class classification. We can either:

- employ a One-versus-All approach and use precisely the framework for nonlinear two-class classification introduced previously with each of our C two-class subproblems
- or introduce nonlinearity ***jointly*** across all C classifiers and tune all parameters simultaneously

- If we choose a ***single set of features*** for all C of our models we can express the c^{th} of which compactly as

$$\text{model}_c\left(\mathbf{x}, \Theta_c\right) = \dot{\mathbf{f}}^T \mathbf{w}_c$$

- Here the notation $\dot{\mathbf{f}}$ is used to compactly denote our B feature transformations
- Θ_c as the linear combination weights and Θ_c denotes the entire set of parameters (including any parameters internal to the feature transformations)

- Once each model has been tuned properly we predict the label of an input point $\mathbf{x}$ employing the fusion rule across all C of these nonlinear models - mirroring our approach in the linear setting

$$y = \underset{c=0,..,C-1}{\text{argmax}} \; \dot{\mathbf{f}}^T \mathbf{w}_c$$

- To properly tune the parameters we must minimize an appropriate cost, e.g., the Multiclass Softmax

$$g\left(\Theta\right) = \frac{1}{P}\sum_{p=1}^{P}\left[\log\left(\sum_{c=0}^{C-1} e^{\dot{\mathbf{f}}_p^T \mathbf{w}_c}\right) - \dot{\mathbf{f}}_p^T \mathbf{w}_{y_p}\right]$$

10.6 Nonlinear Unsupervised Learning

- The linear Autoencoder model

$$\text{model}\left(\mathbf{x}, \mathbf{C}\right) = \mathbf{C}\ \mathbf{C}^{T}\mathbf{x}.$$

- is parameterized by the $N \times K$ matrix $\mathbf{C}$ that will act as our recovered basis upon minimization of the cost function

$$g\left(\mathbf{C}\right) = \frac{1}{P}\sum_{p=1}^{P}\left\|\mathbf{C}\ \mathbf{C}^{T}\mathbf{x}_{p} - \mathbf{x}_{p}\right\|_{2}^{2}$$

We can decompose the linear Autoencoder model into an *encoding* and *decoding* modeling sequence:

$$\begin{aligned} &\text{encoder model:} \quad \text{model}_e\left(\mathbf{x}, \mathbf{C}\right) = \mathbf{C}^T\mathbf{x} \\ &\text{decoder model:} \quad \text{model}_d\left(\mathbf{v}, \mathbf{C}\right) = \mathbf{C}\mathbf{v} \end{aligned}$$

In this notation our general linear Autoencoder model is then the *composition of these two individual models* as

$$\text{model}\left(\mathbf{x}, \mathbf{C}\right) = \text{model}_d\left(\text{model}_e\left(\mathbf{x}, \mathbf{C}\right), \mathbf{C}\right)$$

To introduce nonlinearity here, we can simply replace the linear encoder/decoder models with nonlinear versions of the generic form

$$\begin{aligned}\text{encoder model:} \quad & \text{model}_e\left(\mathbf{x}, \Theta_e\right) = \mathbf{f}_e\left(\mathbf{x}\right) \\ \text{decoder model:} \quad & \text{model}_d\left(\mathbf{v}, \Theta_d\right) = \mathbf{f}_d\left(\mathbf{v}\right)\end{aligned}$$

Here, $\mathbf{f}_e$ and $\mathbf{f}_d$ are (in general) nonlinear vector-valued functions, with Θ_e and Θ_d denoting their parameter sets.

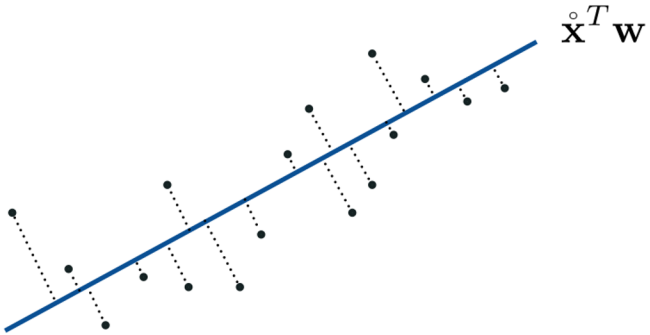
With this notation we can write the **general nonlinear Autoencoder** model simply as

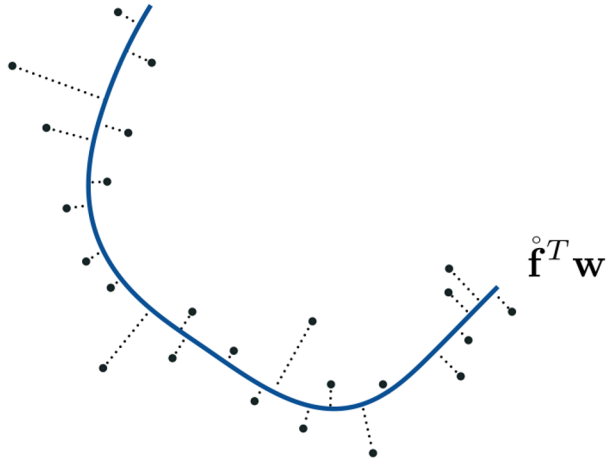
$$\text{model}\left(\mathbf{x}, \Theta\right) = \text{model}_d\left(\text{model}_e\left(\mathbf{x}, \Theta_e\right), \Theta_d\right)$$

We can tune the parameters of Θ (which contains all parameters of Θ_e and Θ_d) by minimizing the Least Squares cost

$$g\left(\Theta\right) = \frac{1}{P}\sum_{p=1}^{P}\left\|\text{model}\left(\mathbf{x}, \Theta\right) - \mathbf{x}_p\right\|_2^2$$

Abstract illustration of (left) linear and (right) nonlinear Autoencoder





Summary

- Nonlinear Learning: modeling principles
- Nonlinear multi-output regression
- Nonlinear two-class classification
- Nonlinear multi-class classification
- Nonlinear unsupervised learning