CS-541: Artificial Intelligence
Lecture 2

Abdul Rafae Khan
Department of Computer Science
Stevens Institute of Technology
akhan4@stevens.edu

January 31, 2022

Overview

1. Machine Learning
2. Supervised vs Unsupervised Learning
3. Supervised Learning
4. Linear Models
   - 4.1 Regression & Classification
   - 4.2 Loss function
   - 4.3 Gradient Descent
5. Non-Linear Models

Machine Learning

Learn from the experiences using the available data
Apply the learning to any unknown data

Machine Learning

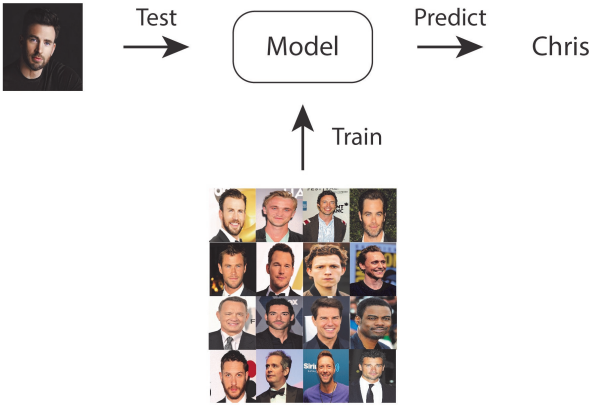
Train model parameters on data samples (training data)

Machine Learning

Train model parameters on data samples (training data)
Trained model can be a function f or a neural network

Machine Learning

Train model parameters on data samples (training data)
Trained model can be a function f or a neural network
Then test on unseen sample and get the output (test data)



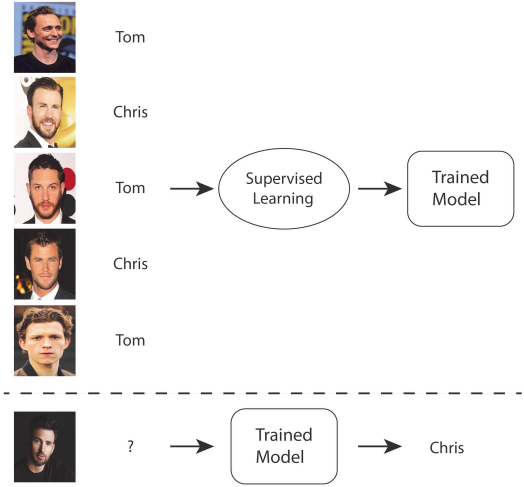
Supervised vs Unsupervised Learning

Two main types of learning approaches

- Supervised learning
  - The data is labelled/annotated
- Unsupervised learning
  - The data is unlabelled

Supervised vs Unsupervised Learning

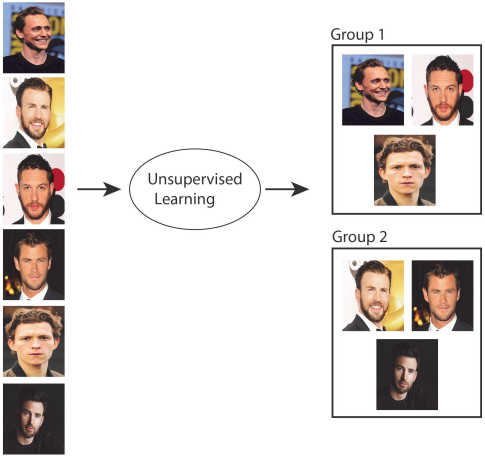
Training data includes observations and true labels (annotated data)



Supervised vs Unsupervised Learning

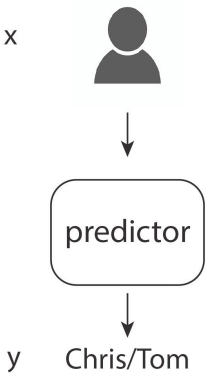
Training data is unannotated, i.e. no true label available

Typically we can only group the data samples



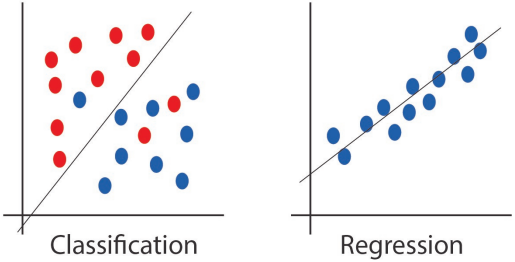
Supervised Learning

We have an input x and we want to get a specific output y



Supervised Learning

Two main types of algorithms in supervised learning



Supervised Learning

We have an input x and we want to get a specific output y

- Binary classification: $y = \{0, 1\}$

E.g. Spam filtering

Supervised Learning

We have an input x and we want to get a specific output y

- Binary classification: $y = \{0, 1\}$

E.g. Spam filtering

- Regression: $y \in \mathbb{R}$

E.g. Housing price

Supervised Learning

We have an input x and we want to get a specific output y

- Binary classification: $y = \{0, 1\}$

E.g. Spam filtering

- Regression: $y \in \mathbb{R}$

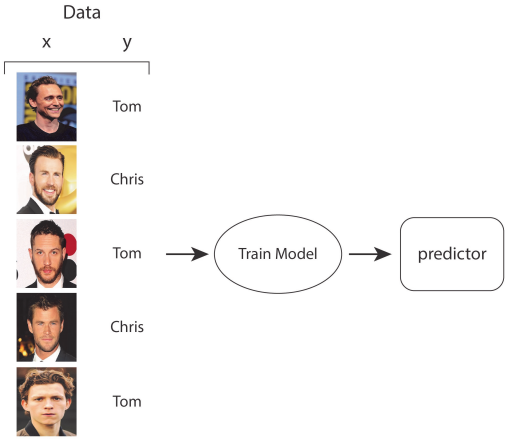
E.g. Housing price

- Multi-class classification: $y = \{0, 1, \cdots, n\}$

E.g. Object recognition

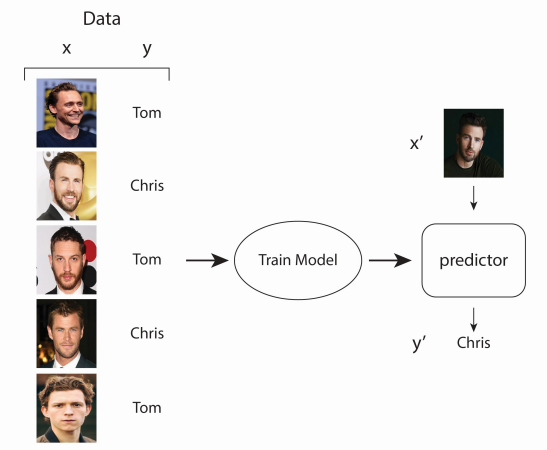
Supervised Learning

Train predictor on lots of training data



Supervised Learning

Test the predictor on unknown data and compare the result

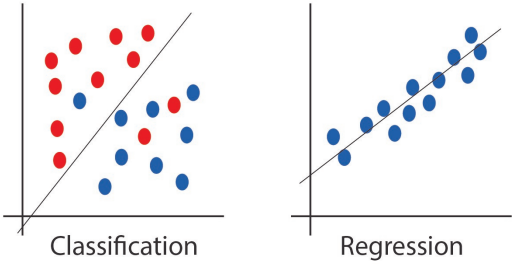


Linear Models

The simplest form of machine learning models
We assume that our model can be represented by a straight line
Easy to train and interpret

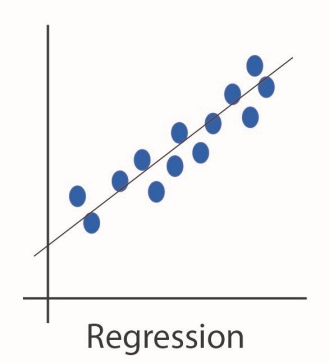
Linear Models

Classification: A straight line can separate the data into two classes
Regression: A straight line can best describe the data points



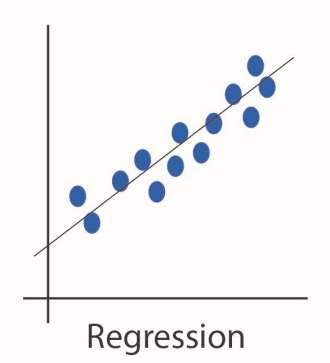
Linear Regression

Lets start with a linear regression model



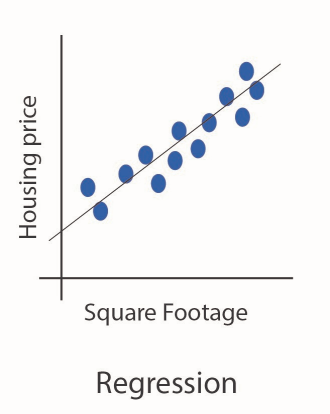
Linear Regression

Lets start with a linear regression model
Train a linear regression line which best fit the data
Model is of the form $y = wx + b$



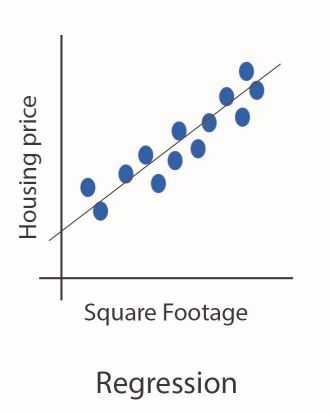
Linear Regression

Lets start with a simple regression task
Train a linear regression line which best fit the data
Model is of the form $y = wx + b$
x is the square footage & y is the housing price



Linear Regression

Lets start with a linear regression model
Train a linear regression line which best fit the data
Model is of the form $y = wx + b$
x is the square footage & y is the housing price
w & b are the model parameters we want to learn



Linear Regression

input feature: $x = \{\text{square footage}\}$

$$y = wx + b$$

Linear Regression

However housing price does not only depend upon area, e.g. bedrooms, bathrooms, year built, garage area, $\cdots$

input features: $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$

Linear Regression

input features: $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}$
model parameters: $\boldsymbol{w} = \{w_1, w_2, \cdots, w_n\}$

$$y = w_1 x_1 + w_2 x_2 + w_3 x_3 + \cdots + w_n x_n + b$$
$$y = \boldsymbol{w}\boldsymbol{x} + b$$

Linear Regression

To simplify the equation,
input features: $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n, 1\}$
model parameters: $\boldsymbol{w} = \{w_1, w_2, \cdots, w_n, b\}$

$$y = \boldsymbol{w}\boldsymbol{x}$$

Linear Regression

To simplify the equation,
input features: $x = \{x_1, x_2, \cdots, x_n, 1\}$
model parameters: $w = \{w_1, w_2, \cdots, w_n, b\}$

$$y = \boldsymbol{w}\boldsymbol{x}$$
$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

$h_{\boldsymbol{\theta}}(x)$ is read h of x given $\boldsymbol{\theta}$
$\boldsymbol{\theta}$ represents the model parameters

Linear Regression

How do we train the model parameters?
Ideally,

$$wx = y$$

Linear Regression

How do we train the model parameters?
Ideally,

$$wx = y$$

However,

$$wx = \hat{y}$$

Linear Regression

Loss is

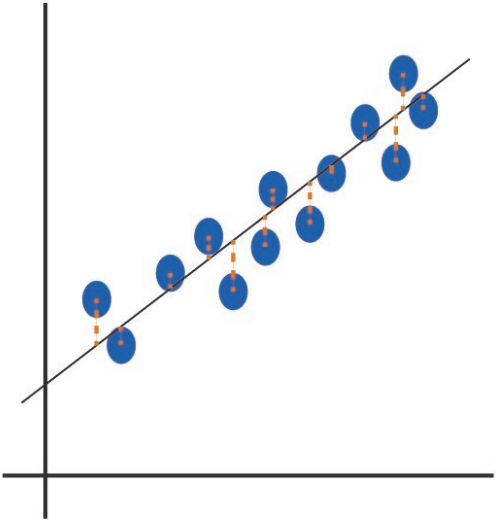
$$L(\boldsymbol{w}, y) = y - \hat{y}$$

Model Loss

$$\begin{aligned} L(\boldsymbol{w}) &= \sum_{\boldsymbol{x} \in X} (y - \hat{y})^2 \\ &= \sum_{\boldsymbol{x} \in X} (y - \boldsymbol{w}\boldsymbol{x})^2 \end{aligned}$$

Loss function

Sum of all the squared distances



Loss function

E.g., we have 3 samples in our training data D

$$D = \begin{bmatrix} 2 & 3 \\ 5 & 7 \\ 9 & 6 \end{bmatrix}$$

Loss function

E.g., we have 3 samples in our training data D
1st column is the feature (x) and the 2nd is the actual value (y)

$$D = \begin{array}{c} \begin{bmatrix} x & y \end{bmatrix} \\ \hline \begin{bmatrix} 2 & 3 \\ 5 & 7 \\ 9 & 6 \end{bmatrix} \end{array}$$

Loss function

If we select $w = 0.5$ and $b = 0$

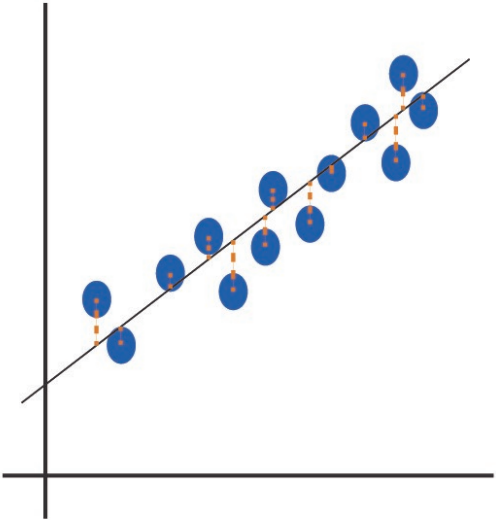
$$
\begin{aligned}
L(w) = &\sum_{x \in X} (y - wx)^2 \\
= &(3 - (2 * 0.5 + 0))^2 + (7 - (5 * 0.5 + 0))^2 \\
&+ (6 - 9 * 0.5)^2 \\
= &26.5
\end{aligned}
$$

Loss function

```
# Sum of squared error loss
def calculate_loss(w,X,Y):
    n = len(Y)
    sse_loss = 0
    for i in range(n):
        y_hat = X[i]*w
        sse_loss += (Y[i]-y_hat)*(Y[i]-y_hat)
    return sse_loss
```

Loss function

Sum of all the squared distances



Loss function

Common loss functions

- Sum of Squared Error (SSE) $= \sum_{i=1}^{N}(y - \hat{y})^2$
- Mean Squared Error (MSE/L2 loss)$= \frac{1}{n}\sum_{i=1}^{N}(y - \hat{y})^2$
- Mean Absolute Error (MAE/L1 loss) $= \frac{1}{n}\sum_{i=1}^{N}|y - \hat{y}|$
- Root Mean Squared Error (RMSE) $= \sqrt{\frac{1}{n}\sum_{i=1}^{N}(y - \hat{y})^2}$

Loss function

Loss function

$$
\begin{aligned}
L(\boldsymbol{w}) &= \sum_{\boldsymbol{x} \in X} (y - \hat{y})^2 \\
&= \sum_{\boldsymbol{x} \in X} (y - \boldsymbol{w}\boldsymbol{x})^2
\end{aligned}
$$

Objective is to **minimize** the loss function

$$
\begin{aligned}
\boldsymbol{w}^* &= \arg\min_{\boldsymbol{w}} L(\boldsymbol{w}) \\
&= \arg\min_{\boldsymbol{w}} \sum_{\boldsymbol{x} \in X} (y - \boldsymbol{w}\boldsymbol{x})^2
\end{aligned}
$$

Loss function

How to calculate optimal $\boldsymbol{w}^*$?
Update $\boldsymbol{w}$ for T times (epoch)
Or until no change is observed (model converges)

Gradient Descent

How to calculate optimal $\boldsymbol{w}$?
Update $\boldsymbol{w}$ for T times (epoch)
Or until no change is observed (model converges)
For every step $t+1$ update $\boldsymbol{w}$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \cdot \nabla L(\boldsymbol{w}_t)$$

Gradient Descent

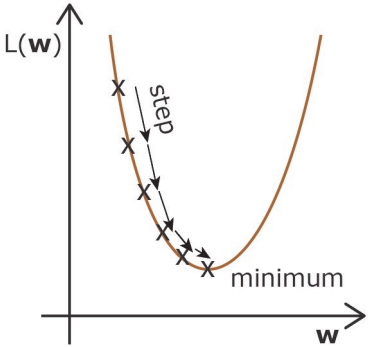
How to calculate optimal $\boldsymbol{w}$?
Update $\boldsymbol{w}$ for T times (epoch)
Or until no change is observed (model converges)
For every step $t+1$ update $\boldsymbol{w}$

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \cdot \nabla L(\boldsymbol{w}_t)$$

$\boldsymbol{w}_t$ is the previous timestep's model parameters
$\nabla L(\boldsymbol{w}_t)$ is the gradient of loss function at time t
η is the step size

Gradient Descent

1. Start with a random value of $\boldsymbol{w}_0$
2. Compute the gradient of the loss function; $\nabla L(\boldsymbol{w})$ for all $\boldsymbol{x} \in X$
3. Move η step in the gradient direction
4. Repeat until $\boldsymbol{w}_{t+1}$ and $\boldsymbol{w}_t$ are same/very close



Gradient Descent

Issue with gradient descent,
It has to go over all the input data before parameter update
Therefore, it takes a very long time
This issue can be solved by **stochastic gradient descent**

Gradient Descent

The idea is simple,

1. Divide the training data into multiple batches
2. Update the model parameters after every batch

This makes the model training much faster

Gradient Descent

What should be a suitable value for η (the learning rate)?

Gradient Descent

What should be a suitable value for η (the learning rate)?
Larger η, we may over step the minimum value
Smaller η, we need to make a lot of steps to reach the minimum

Gradient Descent

```
def gradient_descent(X,Y,learning_rate=0.01,epochs=100):
    n = len(Y)
    losses = np.zeros(epochs)
    W = np.zeros(epochs)
    for t in range(1,epochs):
        sum_of_loss_gradient = 0
        for i in range(n):
            y_hat = X[i]*W[t-1]
            sum_of_loss_gradient += X[i]*(Y[i]-y_hat)
        loss_gradient = -2*sum_of_loss_gradient
        W[t] = W[t-1]-learning_rate*loss_gradient
    return W, losses
```

Gradient Descent

If our model is $y = wx + b$

Then best value using SSE is $w = 0.39189$ and $b = 3.24324$

$$\begin{aligned} L(w) =& (3 - (2 * 0.39189 + 3.24324))^2 \\ &+ (7 - (5 * 0.39189 + 3.24324))^2 \\ &+ (6 - (9 * 0.39189 + 3.24324))^2 \\ \approx& 4.88 \end{aligned}$$

Predicting

Lets say we get a new value $x' = 4$

As our model parameters are $w = 0.39189$ and $b = 3.24324$

$$\begin{aligned} y' &= wx' + b \\ &= 0.39189 * 4 + 3.24324 \\ &\approx 4.81 \end{aligned}$$

15 mins Break!

Recap

Machine Learning

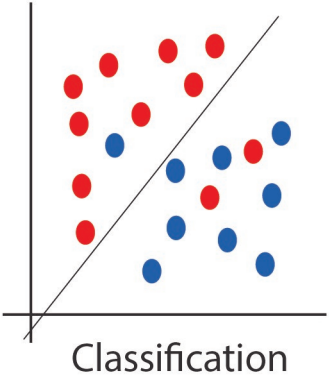
- Suvervised Learning: Data + Class labels
- Unsuvervised Learning: Data only

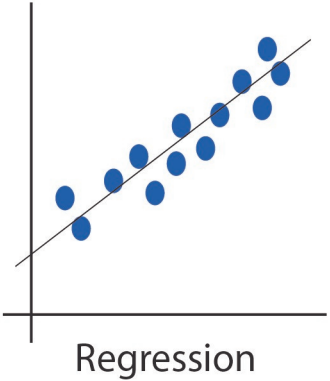
Types of Suvervised Learning algorithms

- Regression
  - Linear Regression: $y = wx + b$
  - Loss function: $L(w) = \sum_x (y - wx)^2$
  - Gradient Descent: $w_{t+1} = w_t - \eta \cdot \nabla L(w)$

Classification

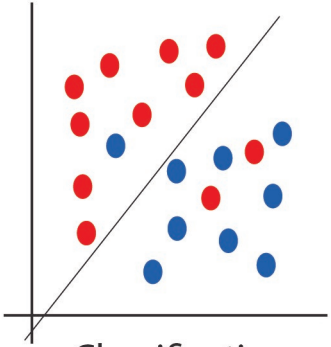
Map data samples to class labels



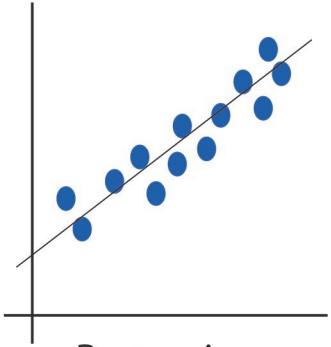


Classification

Map data samples to class labels
Unlike regression, labels cannot be ordered
E.g. Red is higher than Blue



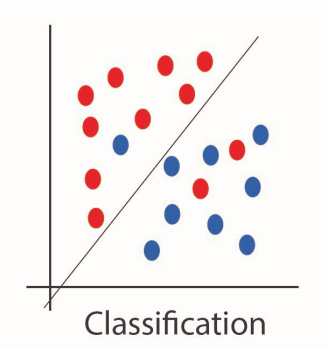
Classification



Regression

Linear Classifier

- Similar to regression,
we want to train a straight line
- However, it should separate the 2 sets of points rather than fit the points.



Linear Classifier

Linear Regression

$$y = wx$$

Linear Regression for classification

$$y = f(wx)$$

Linear Classifier

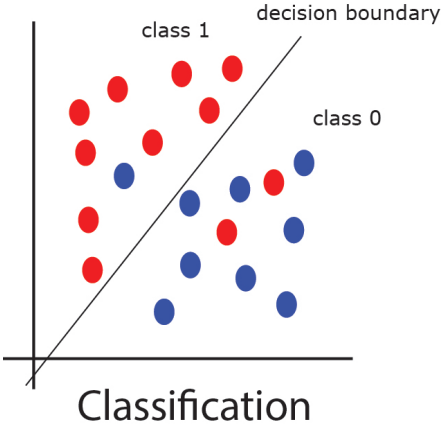
- $f(x)$ is the threshold function
- Can also be written as $f_w(x)$
- Maps values to either class 0 or 1

$$f_w(x) = \begin{cases} 1 \text{ if } wx \geq 0 \\ 0 \text{ if } wx < 0 \end{cases}$$

Linear Classifier

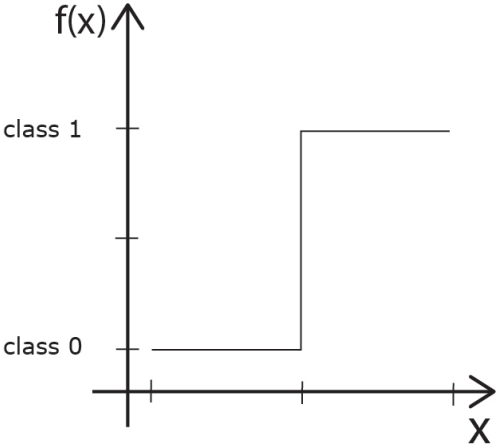
Hard decision boundary

$$f_w(x) = \begin{cases} 1 \text{ if } wx \geq 0 \\ 0 \text{ if } wx < 0 \end{cases}$$



Linear Classifier

Plot of threshold function $f(x)$ against x



Logistic Regression

Use logistic function as f

$$f(x) = \frac{1}{1 + e^{-wx}}$$

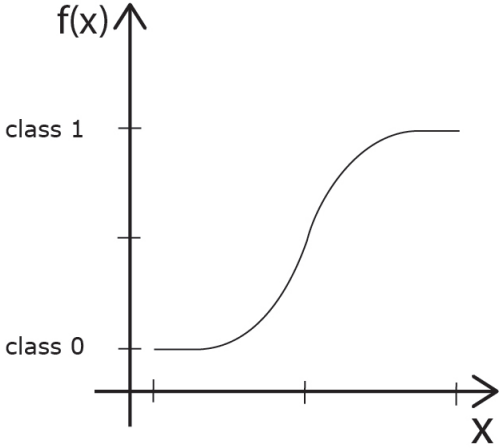
Logistic Regression

Use logistic function as f
Do not get confused with the term 'regression'
It is a classification model

$$f(x) = \frac{1}{1 + e^{-wx}}$$

Logistic Regression

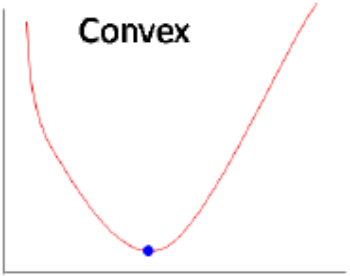
- Gives a smoother decision boundary
- Usually written as $\sigma(z)$
- Gives the probability of x being part of a specific class $P(y|x)$

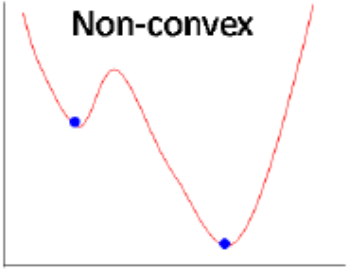


Loss function

- We want to assign loss for incorrect prediction

  e.g. predicting class 1 when the true class is 0
- We want the loss function to be convex





Loss function

We want to assign loss for incorrect prediction
e.g. predicting class 1 when the true class is 0

$$L(f, y_i) = \begin{cases} -\log(f) & \text{if } y_i = 1 \\ -\log(1-f) & \text{if } y_i = 0 \end{cases}$$

Can be written in a single line

$$L(f, y_i) = -y_i \log(f) - (1 - y_i) \log(1 - f)$$

Loss function

For the complete data we average over loss of each sample

$$L(f, y) = \frac{1}{N} \sum_{i=1}^{N} L(f, y_i)$$

Loss function

For the complete data we average over loss of each sample

$$L(f, y) = \frac{1}{N} \sum_{i=1}^{N} L(f, y_i)$$

Gradient is

$$\nabla L(f, y) = \frac{1}{N} \sum_{i=1}^{N} x_i (f - y_i)$$

Gradient Descent

For the complete data we average over loss of each sample

$$L(f, y) = \frac{1}{N} \sum_{i=1}^{N} L(f, y_i)$$

Gradient is

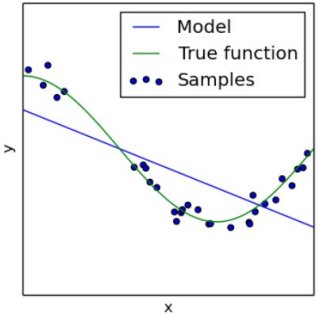
$$\nabla L(f, y) = \frac{1}{N} \sum_{i=1}^{N} x_i (f - y_i)$$

Finally

$$w_{t+1} = w_t - \eta \cdot \nabla L(f, y)$$

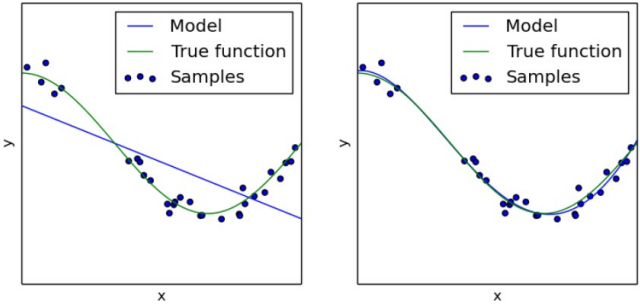
Non-Linear Models

Usually linear models are not the best fit



Non-Linear Models

Usually linear models are not the best fit
We need a non-linear model



Non-Linear Models

Linear input feature: $\boldsymbol{x} = \{x_1, x_2\}$

Polynomial Models

Linear input feature: $\boldsymbol{x} = \{x_1, x_2\}$

Polynomial input feature: $\phi(\boldsymbol{x}) = \{1, x_1, x_2, x_1^2, x_2^2, x_1x_2, x_1^3, x_2^3, x_1^2x_2, x_1x_2^2, \cdots\}$

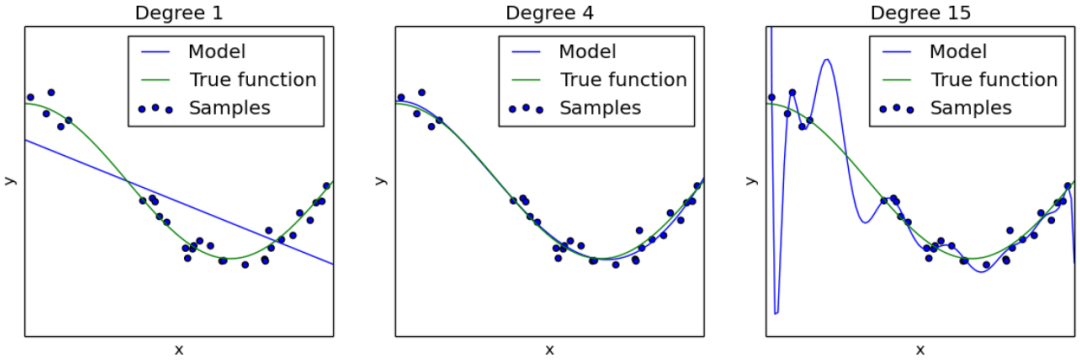
- Degree 0 : 1
- Degree 1 : x_1, x_2
- Degree 2 : x_1^2, x_2^2, x_1x_2
- Degree 3 : $x_1^3, x_2^3, x_1^2x_2, x_1x_2^2$

Polynomial Models

Polynomial Regression Model:

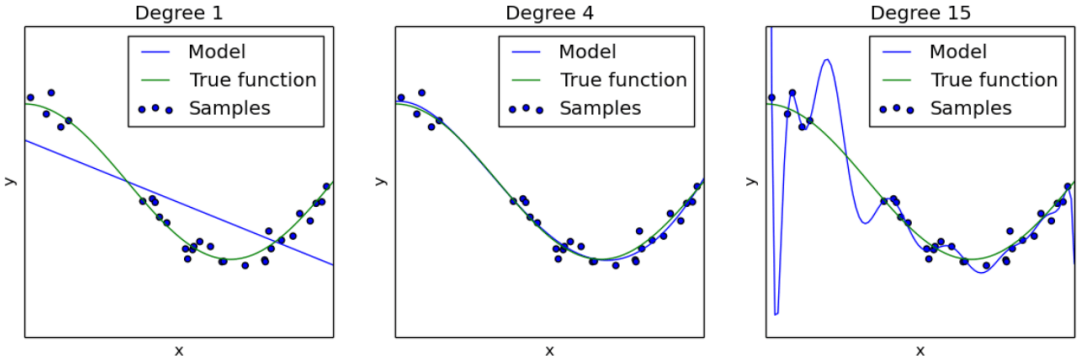
$$
\begin{aligned}
y =& \boldsymbol{w}\phi(\boldsymbol{x}) \\
y =& w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1^2 \\
& + w_4 x_2^2 + w_3 x_1 x_2 + \cdots
\end{aligned}
$$

Polynomial Models



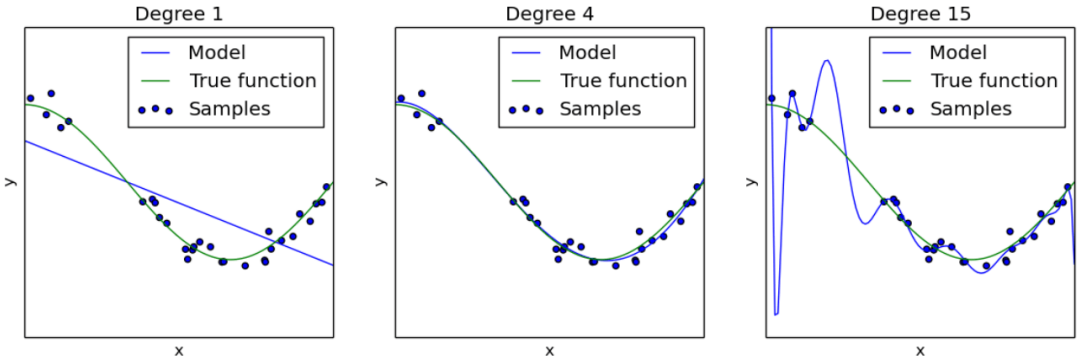
Polynomial Models

Both Degree 1 and Degree 15 are not good models



Polynomial Models

Both Degree 1 and Degree 15 are not good models
Why not Degree 15?



The End