Introduction to Machine Learning

Mahammad Valiyev

08.01.2022

Contents and timeline

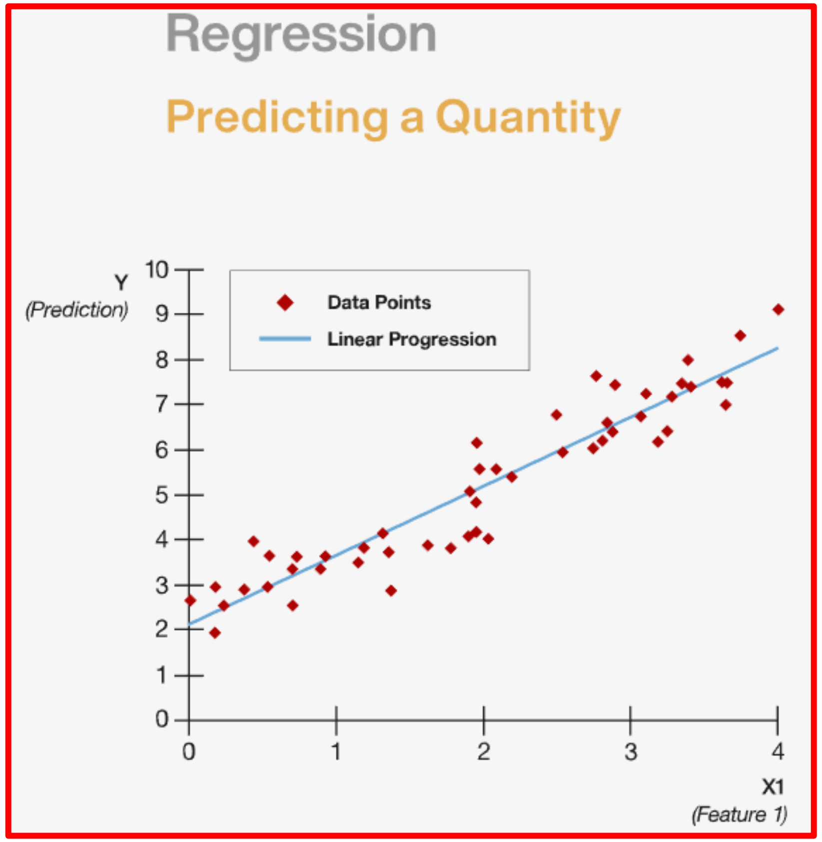
Part 2:

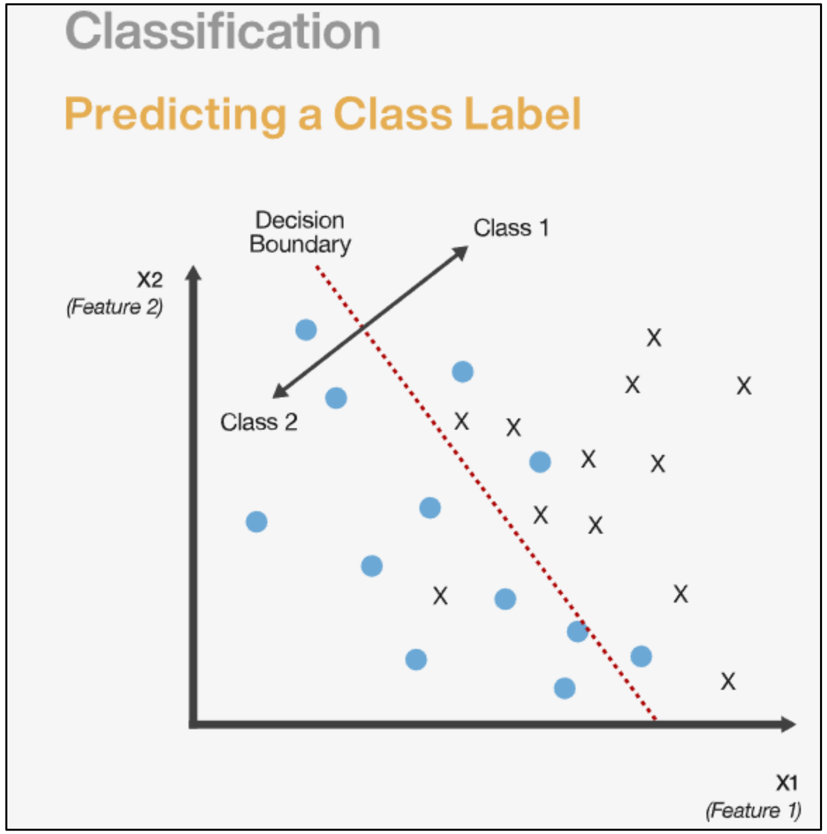
Overview of Machine Learning algorithms

Supervised learning

Supervised learning: learning a functional mapping from input to outputs

- **Regression**: output is continuous variable
- **Classification**: output is categorical variable (2 categories: binary, or more: multiclass)



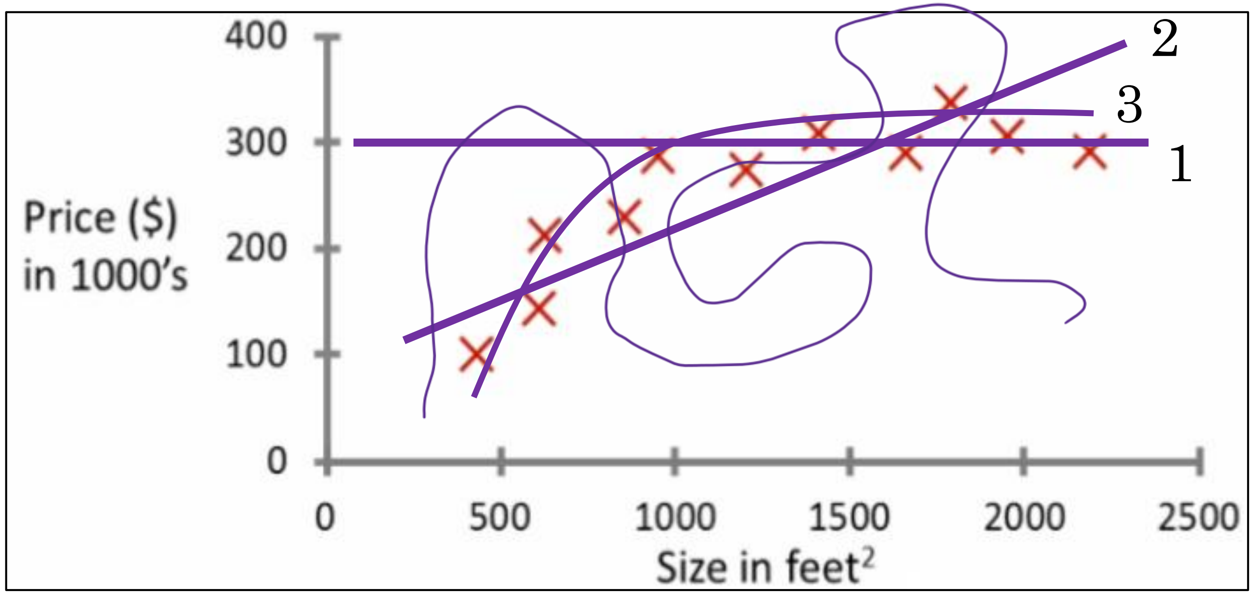


Regression problem

- **Example:** predict price of a house using historical data

Size of a house ($feet^2$)	Price of a house (1000's \$)
2104	460
1416	232
1534	315
…	…

Housing prices data

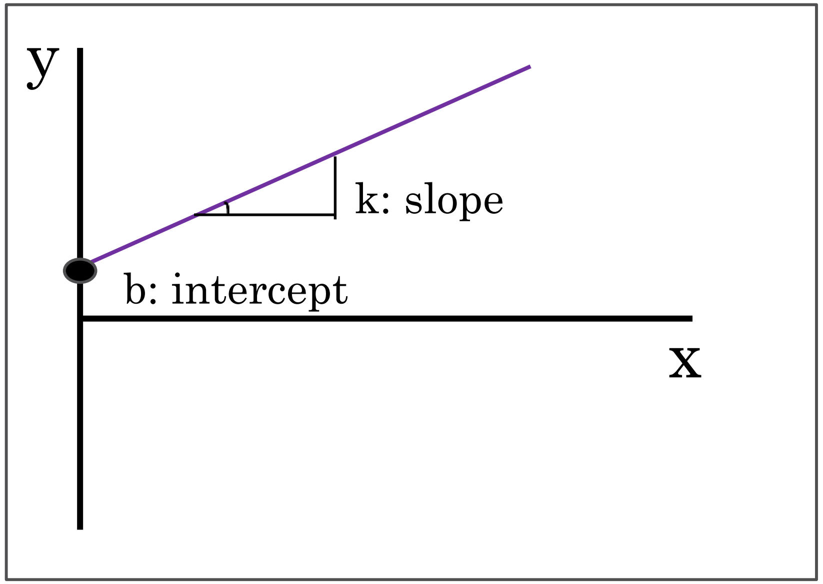


Visualization of data

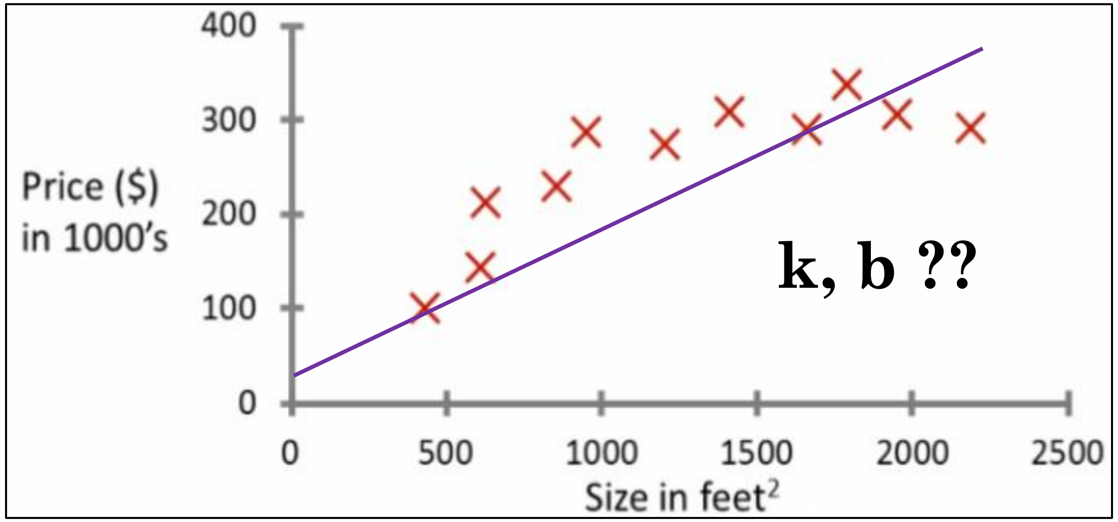
- Different type of models can be used for housing price prediction
- Models vary by underlying assumptions, number of parameters etc
- Example of models relevant to this problem: constant function, linear/quadratic functions

Linear regression

- **Model:** $y(k, b) = kx + b$
  - Parameters: k, b
  - Input: x
  - Output: y



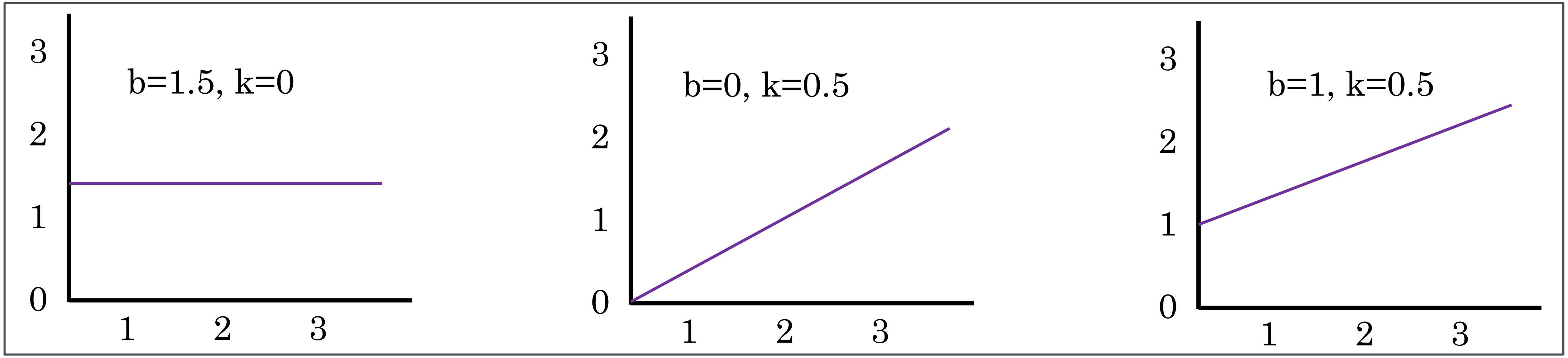
Input (x): size of a house ($feet^2$)	Output (y): price of a House (1000's $)
2104	460
1416	232
1534	315
...	...

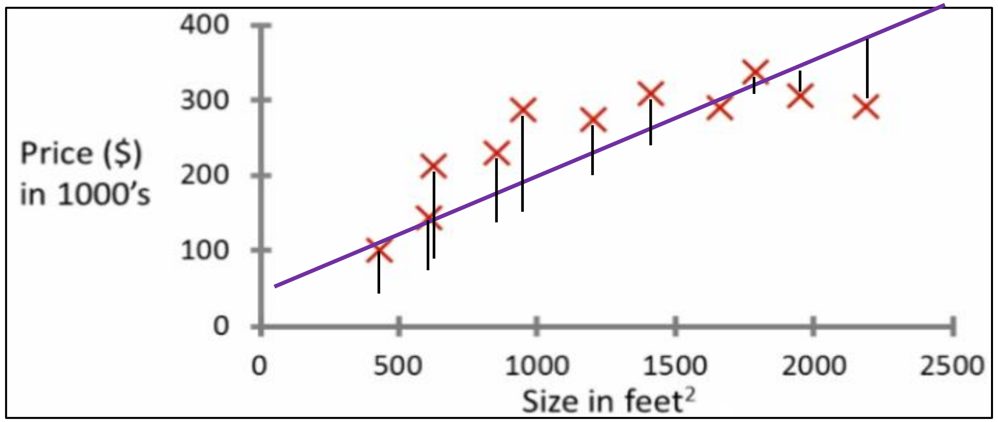


- We chose to model the problem using univariate linear regression model
- Now, given the data, parameters (k, b) of the model need to be estimated..

Estimating parameters of a linear regression model

Model: $y(k, b) = kx + b$





Goal: $\min_{k,b} J \qquad J(k,b) = \frac{1}{2N}\sum_{i=1}^{N} \left((kx_i + b) - y_i\right)^2$

J: objective/cost function

k, b: parameters of a model

x_i , y_i: input and output for each training data

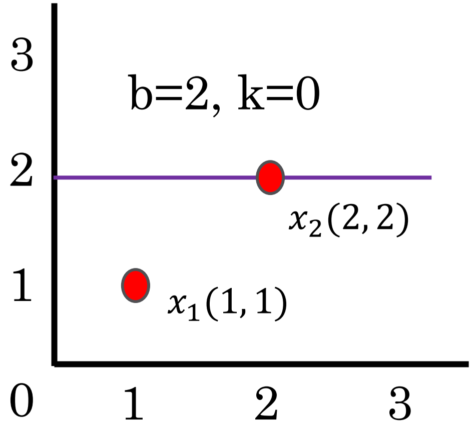
N: number of data points

Intuition: Choose k and b such that, data points are as close as possible (minimum vertical distance) to the fitted line

Cost function intuition

Model: $y(k,b) = kx + b$

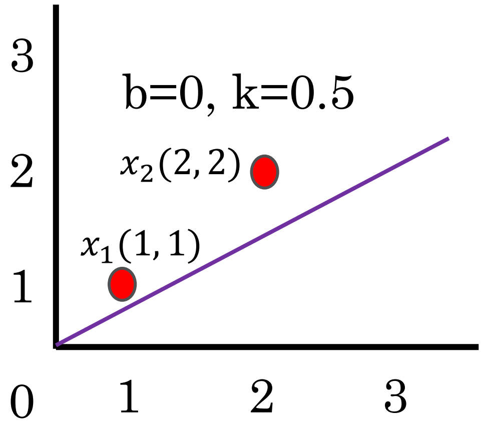
Cost function: $J(k,b) = \frac{1}{2N}\sum_{i=1}^{N} ((kx_i + b) - y_i)^2$



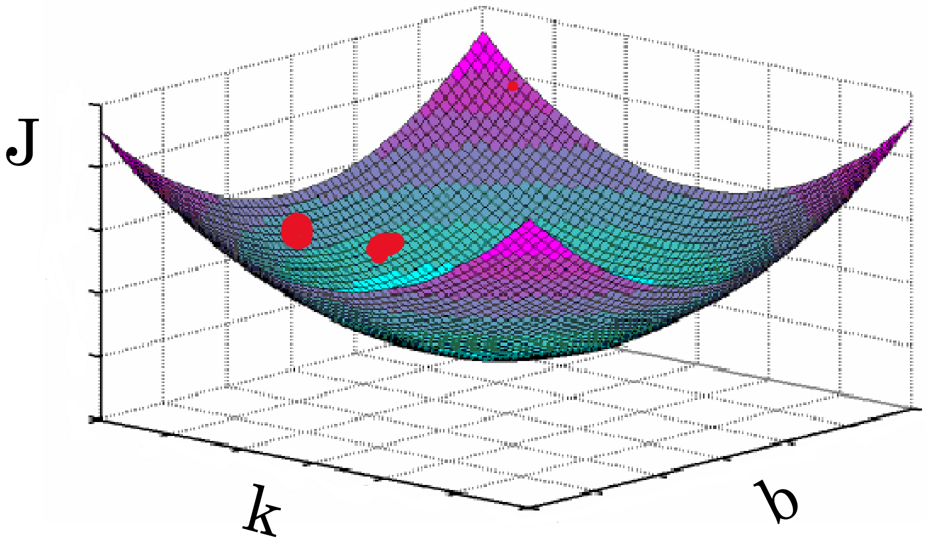
Data	Fit
$x_1(1,1)$	$x_1(1,2)$
$x_2(2,2)$	$x_2(2,2)$

$$J_1 = \frac{1}{2*2}(((0*1+2)-1)^2 + ((0*2+2)-2)^2) = 0.25$$

$$J_2 = \frac{1}{2*2}(((0.5*1+0)-1)^2 + ((0.5*2+0)-2)^2) = 0.31$$



Data	Fit
$x_1(1,1)$	$x_1(1,0.5)$
$x_2(2,2)$	$x_2(2,1)$



Estimating parameters of a linear regression model

Goal: minimize $J(k,b) = \frac{1}{2N}\sum_{i=1}^{N} ((kx_i + b) - y_i)^2$ by varying k and b

Gradient descent algorithm:

- Start with initial guess for k, b
- Update k and b until specified criteria is satisfied:
    - No significant change in J
    - No significant change in k, b
    - Specified number of iterations has been reached

Pseudocode for gradient descent

Initialize k, b, ε, α

while $(J_{n-1} - J_n) > \varepsilon$

$k_n = k_{n-1} - \alpha * \frac{\partial J(k_{n-1}, b_{n-1})}{\partial k}$

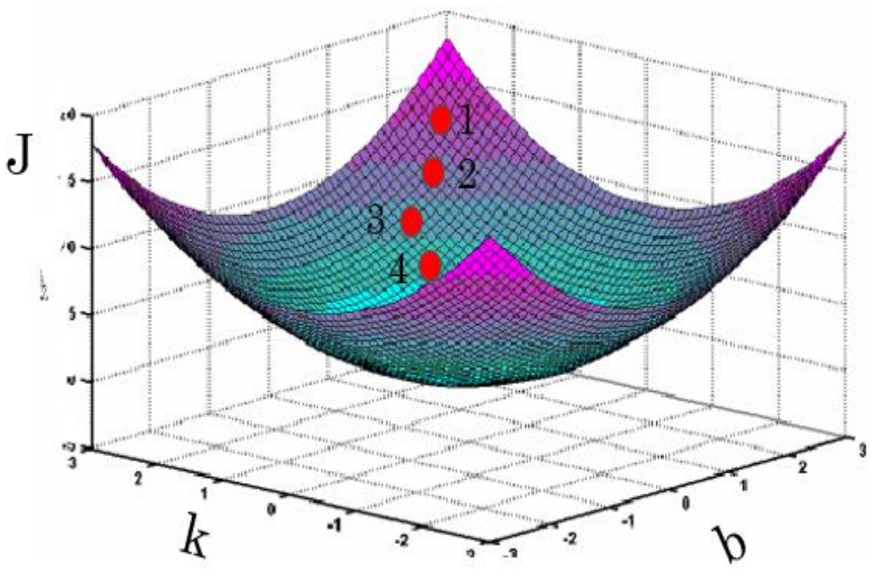
$b_n = b_{n-1} - \alpha * \frac{\partial J(k_{n-1}, b_{n-1})}{\partial b}$

end

Return k, b

- $\frac{\partial J}{\partial k} = \frac{1}{2N}\sum_{i=1}^{N} ((kx_i + b) - y_i)^1 * 2 * x_i$
- $\frac{\partial J}{\partial b} = \frac{1}{2N}\sum_{i=1}^{N} ((kx_i + b) - y_i)^1 * 2$

- Gradient is the direction of the steepest ascent, so opposite of gradient is the direction of steepest descent
- Proof: $\mathrm{grad}(f(a)) \cdot \vec{v} = |\mathrm{grad}(f(a))||\vec{v}|\cos(\theta)$
  directional derivative is maximized when $\cos(\theta) = 1$, meaning at gradient direction



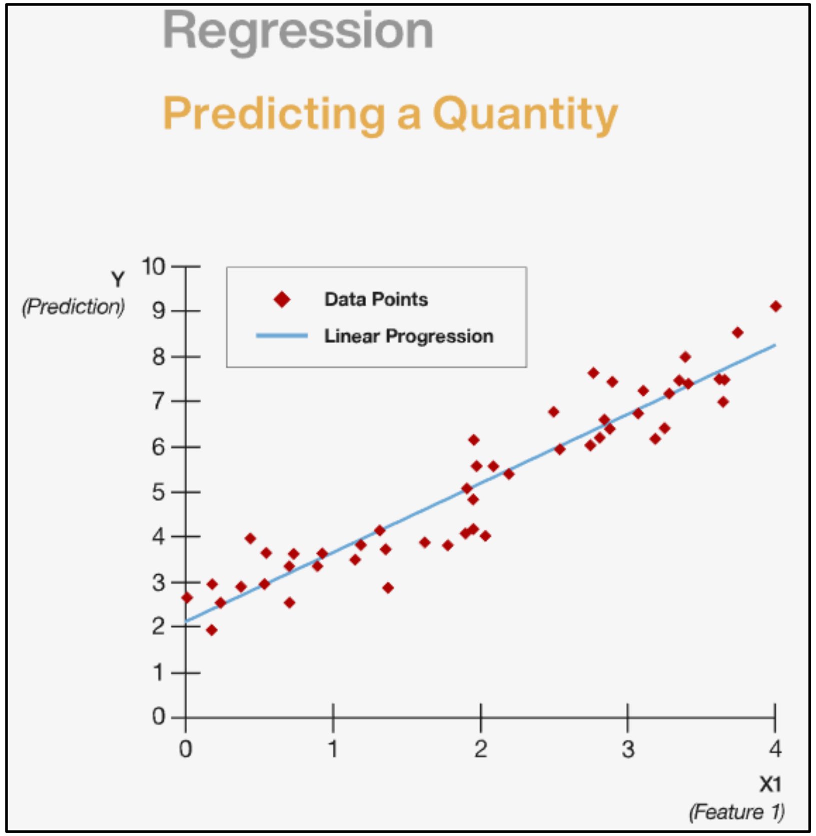
Linear regression summary

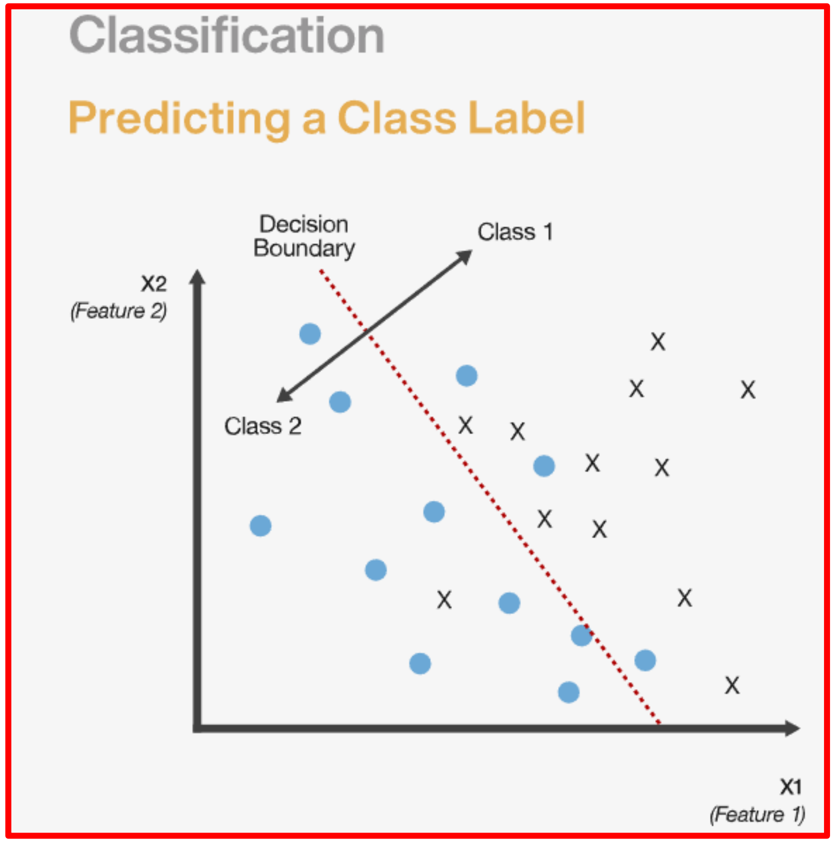
- Linear regression assumes a linear relationship between input and output
- Linear regression can be extended to:
  - multiple inputs: multilinear regression $\boxed{y = b_0 + b_1x_1 + b_2x_2 + \ldots + b_nx_n}$
  - categorical (binary or multiclass) inputs: +1 for 1st category, -1 for the 2nd
  - nonlinear features (still linear!): polynomial regression $\boxed{y = b_0 + b_1x_1 + b_2x_1^2 + \ldots + b_nx_1^n}$
  - nonlinear regression: $\boxed{\hat{y} = \theta_0 + {\theta_2}^2 x}$ $\boxed{\hat{y} = \theta_0 + \theta_1{\theta_2}^x}$ $\boxed{\hat{y} = \log(\theta_0 + \theta_1x + \theta_2x^2 + \theta_3x^3)}$
- Advantages of linear regression:
  - easy to understand, implement and interpret
  - performs well for linearly separable data
- Disadvantages of linear regression:
  - Most of real-world problems are non-linear, so linearity assumption does not hold
  - Sensitive to outliers
  - Assumption of independence of inputs (affects interpretability)

Supervised learning

Supervised learning: learning a functional mapping from input to outputs

- **Regression**: output is continuous variable
- **Classification**: output is categorical variable (2 categories: binary, or more: multiclass)



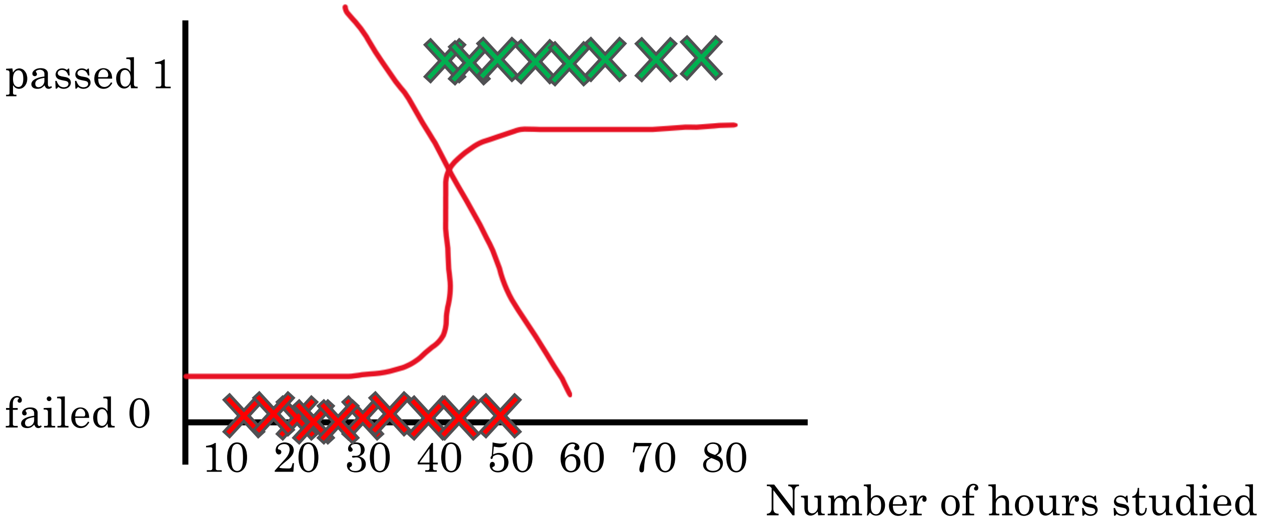


Classification problem

- **Example:** predict whether student will pass or fail an exam based on hours studied

Number of hours student studied	Passed/Failed
60	P
40	F
57	P
…	…

Housing prices data

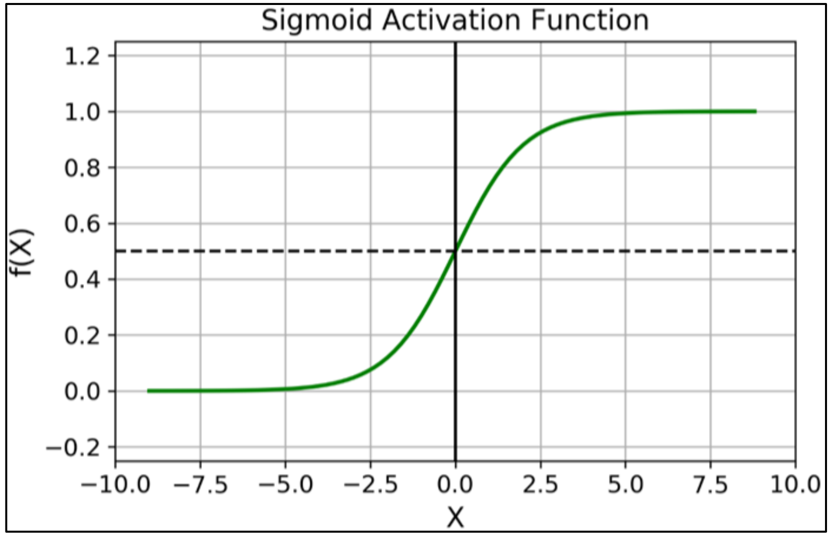


Visualization of data

- Model should predict either 0 or 1
- Classification problem can be binary or multiclass problem

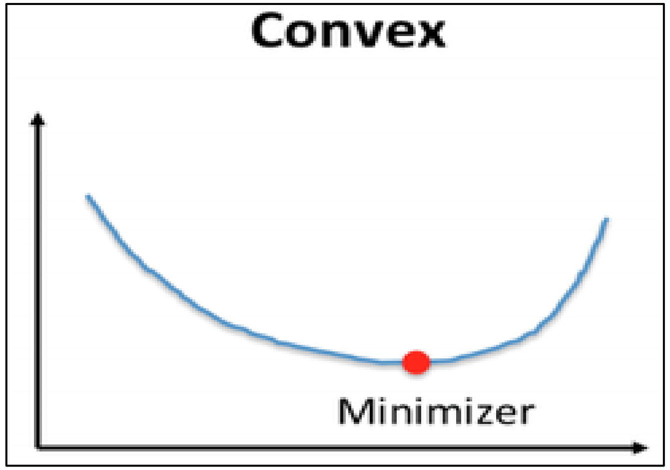
Logistic regression model

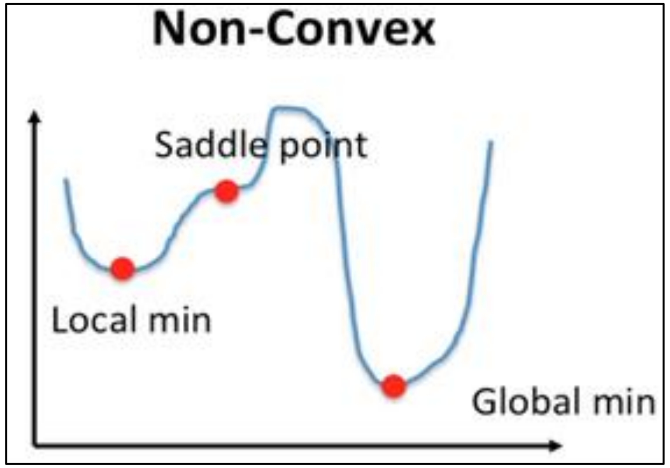
- **Linear regression model:** $y(k,b) = kx + b$
  - Outputs a continuous variable, so can't be used for classification
- **Logistic regression model:** $y(k,b) = f(kx + b)$,
  - where $f(z) = \frac{1}{1+e^{-z}}$ - **sigmoid function**
- So, model becomes: $y(k,b) = \frac{1}{1+e^{-(kx+b)}}$
- Output of sigmoid is in range (0,1)
  - Outputs represents probability of output being = 1
  - Threshold of 0.5 can be used to assign labels
  - Threshold can be modified for imbalanced datasets



Estimating parameters of a logistic regression model

- **Linear regression model:** $\mathrm{y(k,b)} = kx + b$
- **Logistic regression model:** $\mathrm{y(k,b)} = \frac{1}{1+e^{-(kx+b)}}$
- **Linear regression cost function:** $J(k,b) = \frac{1}{2N}\sum_{i=1}^{N} ((kx_i + b) - y_i)^2 = \frac{1}{2N}\sum_{i=1}^{N} Cost(\, y_{i:k,b_{fit}}(x_i)\, ,\, y_{i_{data}})$
- **General cost function for any model:** $J(\theta) = \frac{1}{2N}\sum_{i=1}^{N} Cost(\, y_{i:\theta_{fit}}(x_i))\, ,\, y_{i_{data}})$
- Cost function is measure of difference between fitted and actual data
- In Machine Learning, always, the goal is minimization of cost function
- If linear regression cost function is used for logistic regression -> non-convex cost function





Estimating parameters of a logistic regression model

- Defining a convex cost function for logistic regression..

- $Cost(\, y_{\theta_{fit}}(x),\ y_{data}) = \begin{cases} -\log\left(\, y_{\theta_{fit}}(x)\right) if\ y = 1 \\ -\log\left(1 - \, y_{\theta_{fit}}\,(x)\right) if\ y = 0 \end{cases} = -y \log\left(\, y_{\theta_{fit}}(x)\right)$ - *(1-y)* $\log\left(1 - \, y_{\theta_{fit}}\,(x)\right)$

Logistic regression cost function:

- $J(k, b) = -\frac{1}{N}\sum_{i=1}^{N} \log\left(\frac{1}{1+e^{-(kx_i+b)}}\right) * y_i + \log\left(1 - \frac{1}{1+e^{-(kx_i+b)}}\right) * (1 - y_i)$

Linear regression cost function:

- $J(k, b) = \frac{1}{2N}\sum_{i=1}^{N}\ ((kx_i + b) - y_i)^2$

- **Linear regression model:** $\mathrm{y(k, b)} = kx + b$

- **Logistic regression model:** $\mathrm{y(k, b)} = \frac{1}{1+e^{-(kx+b)}}$

- So, linear and logistic regression have **different model representations** and **different cost functions**

- Logistic regression parameters are also estimated using **gradient descent**

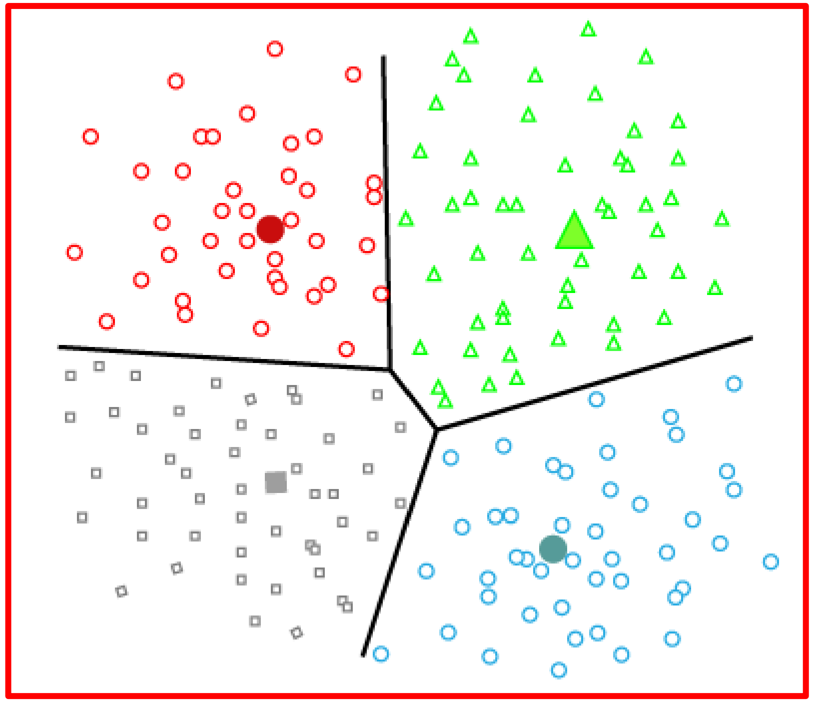
Logistic regression summary

- Logistic regression is a simple linear classification model
- Logistic regression can be extended to:
  - multiple inputs: multilinear logistic regression
  - categorical inputs
  - nonlinear features (still linear!): polynomial logistic regression
  - nonlinear model parameters: usually other models are used for complex problems
  - multiclass classification problems
- Advantages of linear regression:
  - easy to understand, implement and interpret
  - performs well for linearly separable data
- Disadvantages of linear regression:
  - Most of real-world problems are non-liner, so linearity assumption does not hold
  - Sensitive to outliers
  - Assumption of independence of inputs (affects interpretability)

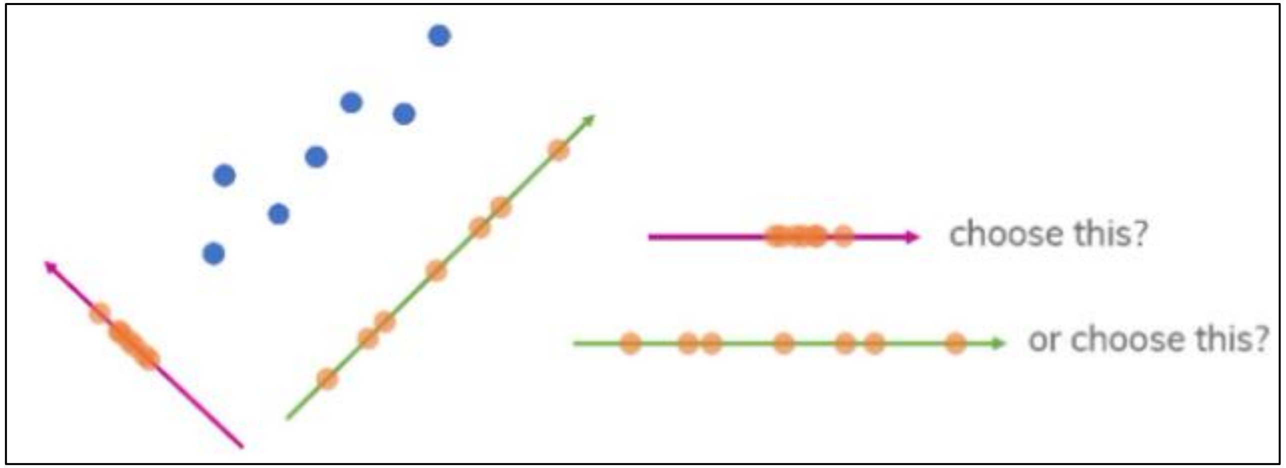
Unsupervised learning

Unsupervised learning: learning patterns from unlabeled data

- **Clustering**: dividing data into a number of groups
- **Dimensionality reduction**: reducing number of input variables



Clustering



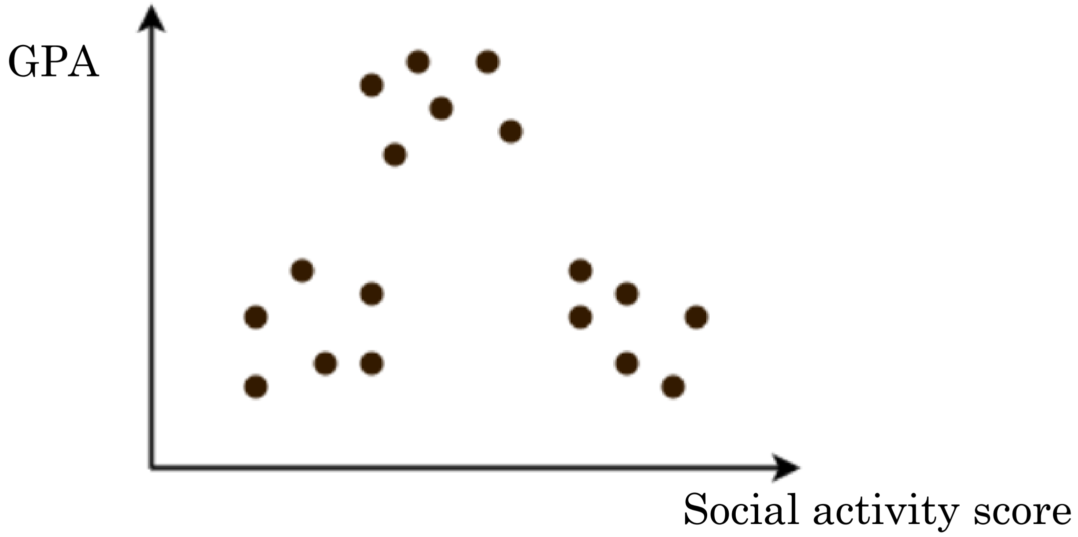
Dimensionality reduction

Clustering problem

- **Example**: classify students into groups based on GPA and social activity scores

GPA	Social activity score
85	84
78	97
96	72
…	…

Student data



Visualization of data

- Students in the same group (cluster) should be more similar to each other than to those in other groups (clusters)

K-means algorithm

- K-means is the most widely used clustering algorithm
- Basic steps in k-means algorithm:
  1. Randomly assign class centroids (numbers of centroids = number of classes)
  2. Assign each data point to some class based on distance to class centroids
  3. Update centroids
  4. Repeat steps 2, 3 until convergence
- Pseudocode

for $k = 1$ **to** K **do**
    $\mu_k \leftarrow$ some random location   // randomly initialize mean for kth cluster
end for
repeat
    for $n = 1$ **to** N **do**
        $z_n \leftarrow \mathrm{argmin}_k \, ||\boldsymbol{\mu}_k - \boldsymbol{x}_n||$ // assign example n to closest center
    end for
    for $k = 1$ **to** K **do**
        $\boldsymbol{\mu}_k \leftarrow \textsc{mean}(\{ \boldsymbol{x}_n : z_n = k \})$ // re-estimate mean of cluster k
    end for
until converged
return z // return cluster assignments

$$d = |\mathbf{x} - \mathbf{y}| = \sqrt{\sum_{i=1}^{n} |x_i - y_i|^2}$$

Euclidean distance between two points

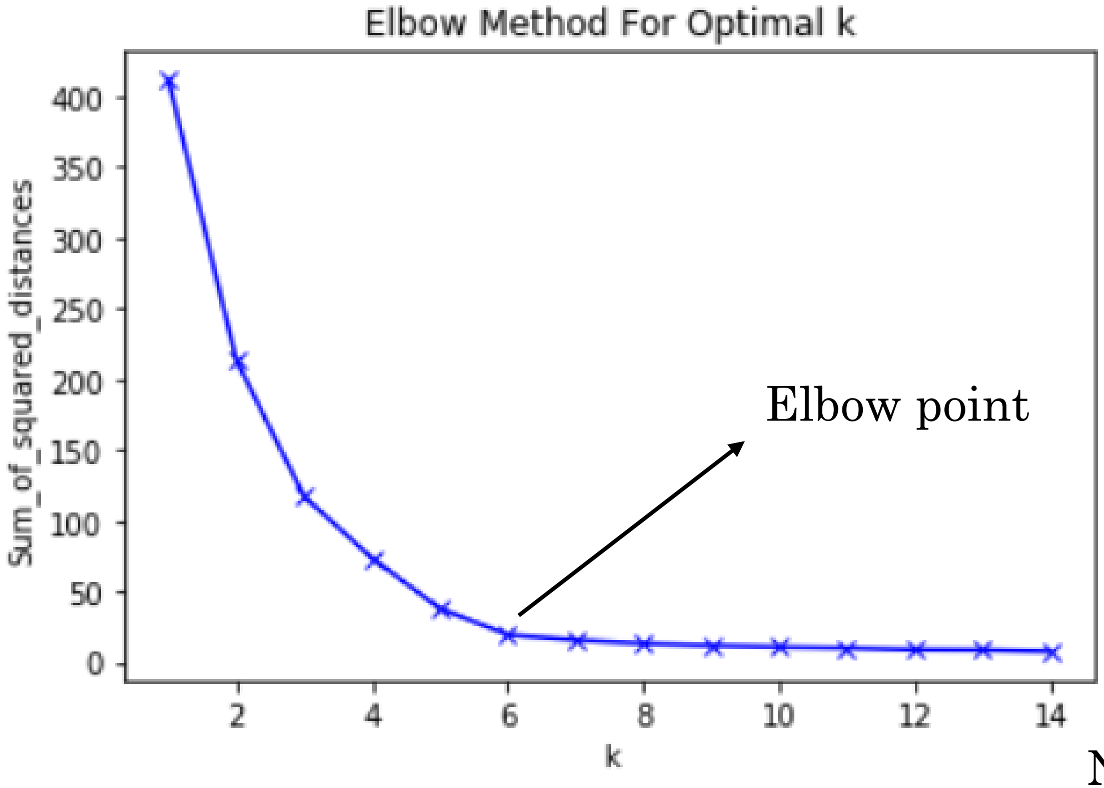
Need to store:
1) coordinates of cluster centroids
2) class of each data point

Convergence criteria:
- Minor change in cluster centroids
- Number of iterations allowed

Choosing number of clusters

- Domain-expertise: e.g. you now for certain how many groups are in data
- Data visualization: plot and see
- Elbow method: sometimes no clear elbow on a plot

Sum of within cluster deviations for each cluster



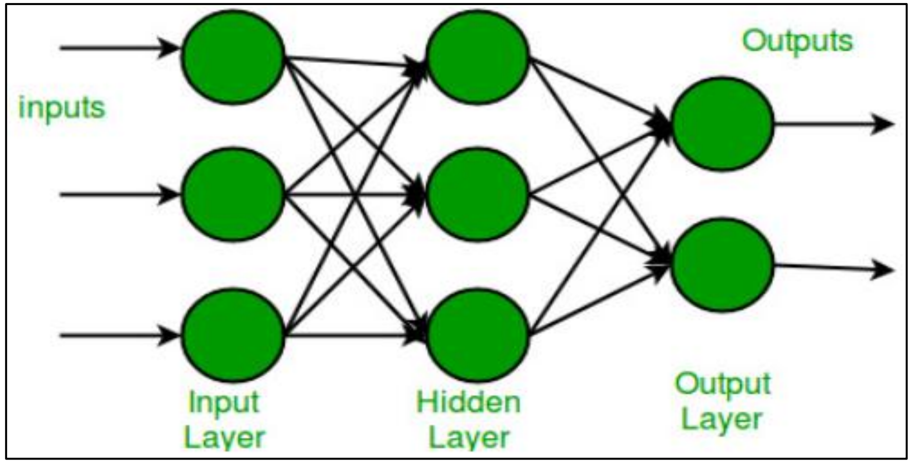
Number of classes

K-means summary

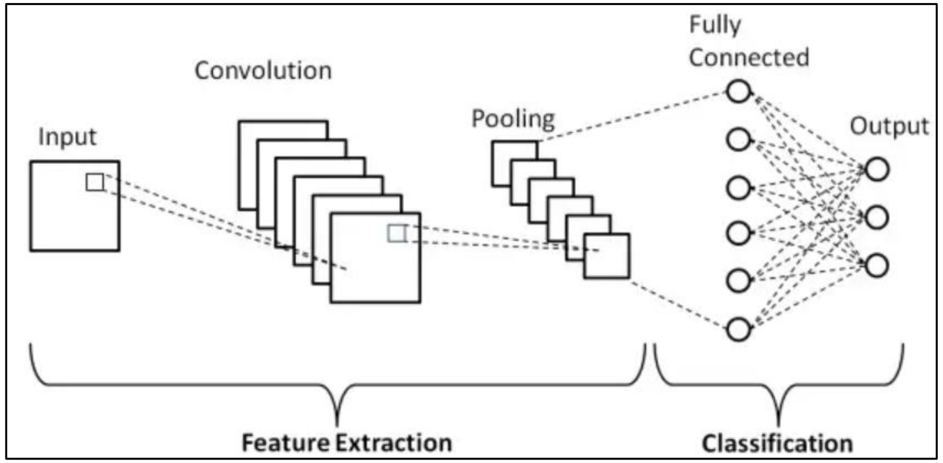
- K-means is the most widely used clustering algorithm
- Advantages
  - Simple to understand, interpret and implement
  - Computationally efficient (scales to large datasets)
- Disadvantages
  - Number of clusters need to be chosen
  - Sensitive to initialization
  - Sensitive to outliers
  - Spherical clustering only (can’t be used for overlapping clusters)

Deep Learning

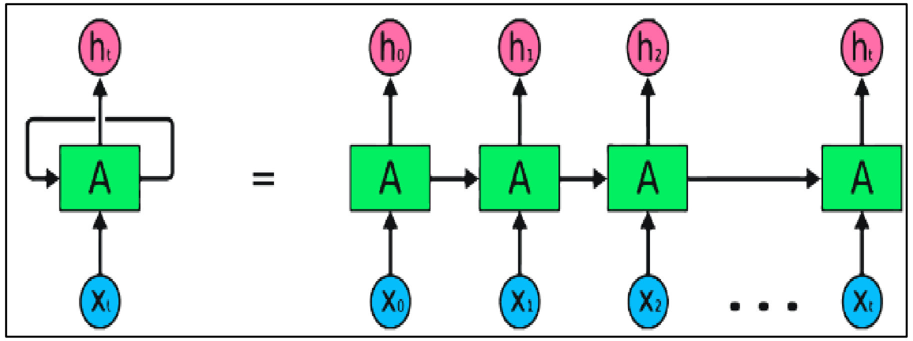
- subset of machine learning that involve use of neural networks



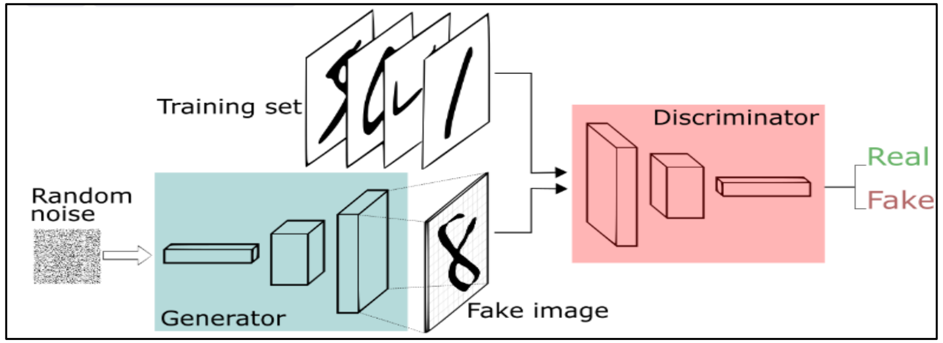
Multilayer perceptron: tabular data



CNN: image data



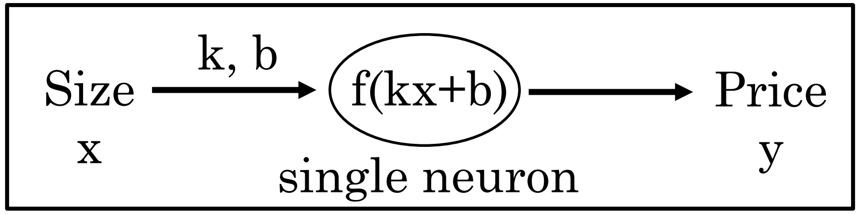
RNN: sequential (timeseries, text) data



GAN: (mostly) image and audio data

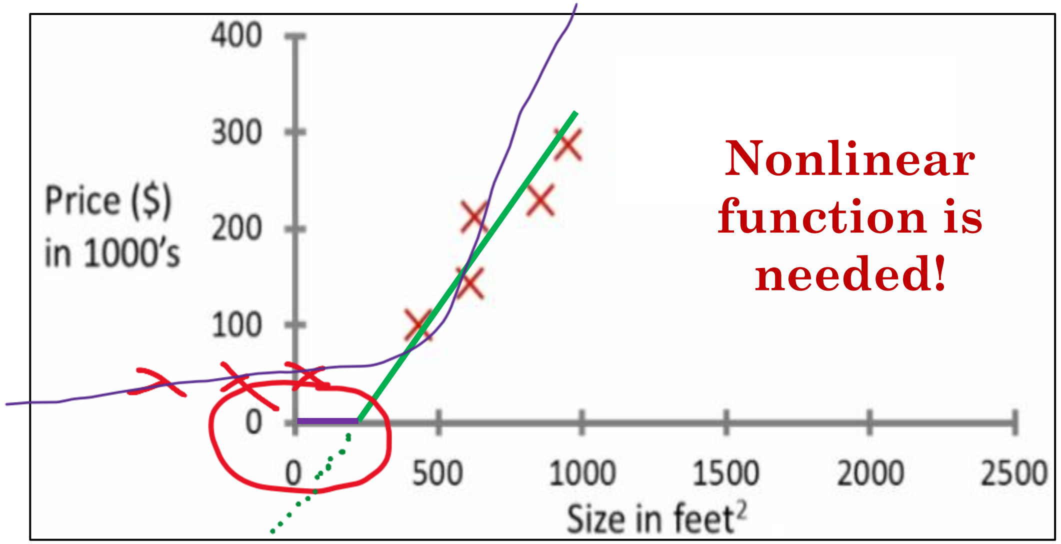
Deep Learning

- **Linear regression:** $y(k,b) = kx + b$
  - k, b: parameters
- **Single unit of a neural net**: y(k,b) = f(kx+b)
  - f: activation function

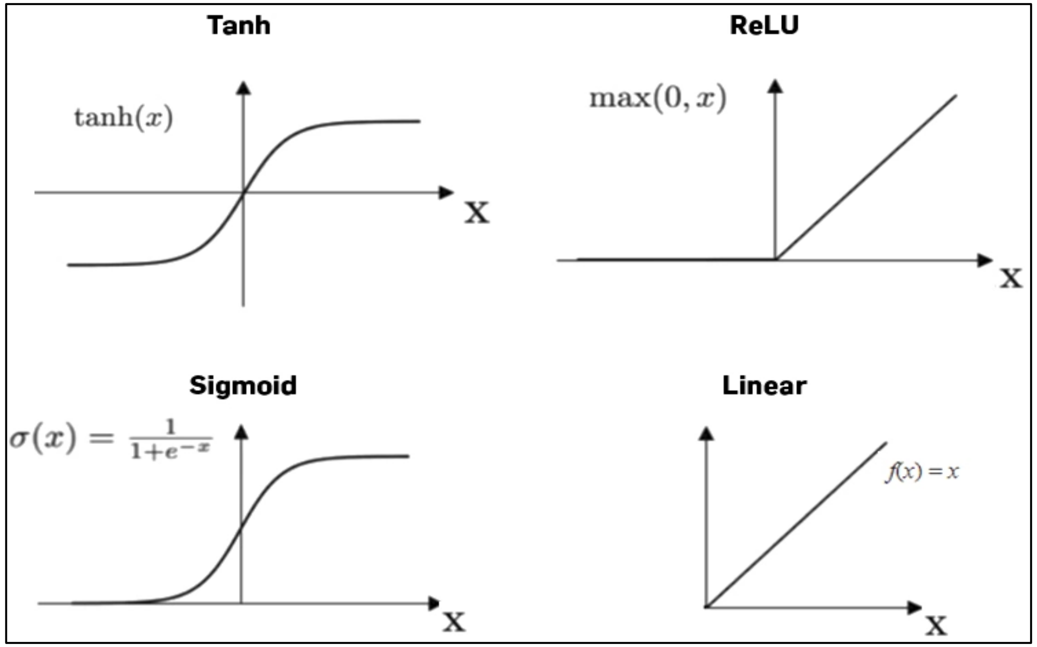


X: Size of a house ($feet^2$)	Y: Price of a house (1000's $)
2104	460
1416	232
1534	315
…	…

Housing prices data

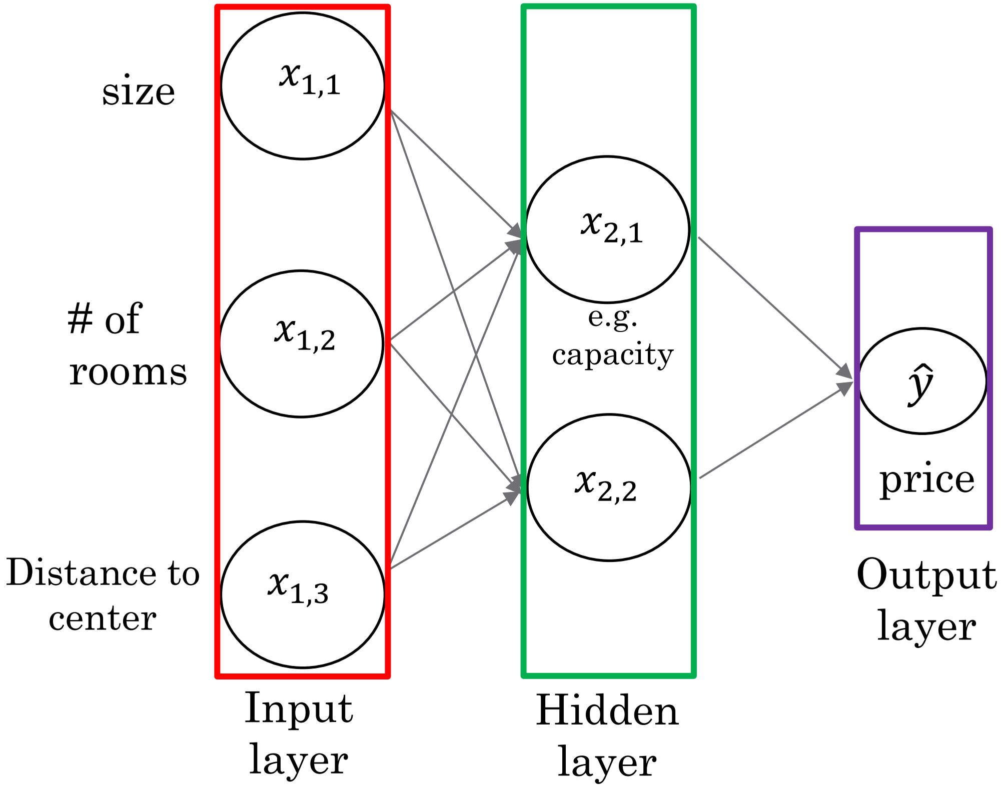


house price prediction



activation functions

Deep Learning



neural network with 1 hidden layer

2 type of computations:

- linear combinations: $\sum_i^n (k_i * x_i + b)$
- activations: f(x)

Forward propagation

Equations

- $x_{2,1} = f\left(k_{1,1,1} * x_{1,1} + k_{1,2,1} * x_{1,2} + k_{1,3,1} * x_{1,3} + b_{2,1}\right) =$

f $(\sum_1^3 (k_{1,j,1} * x_{1,j} + b_{2,1})$

- $x_{2,2} = f\left(k_{1,1,2} * x_{1,1} + k_{1,2,2} * x_{1,2} + k_{1,3,2} * x_{1,3} + b_{2,2}\right) =$

f $(\sum_1^3 (k_{1,j,2} * x_{1,j} + b_{2,2})$

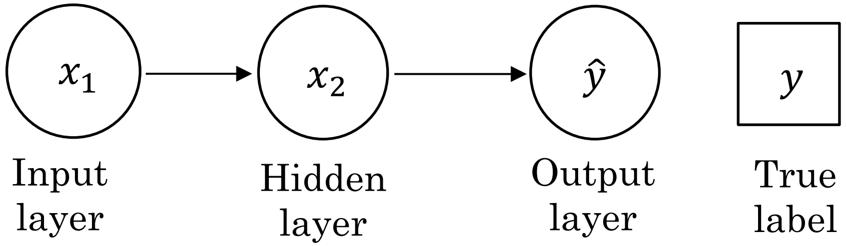
- $\hat{y} = f\,(k_{2,1,1} * x_{2,1} + k_{2,2,1} * x_{2,2} + b_{3,1}) =$

f $(\sum_1^2 (k_{2,j,1} * x_{2,j} + b_{3,1})$

Notation

- $k_{i,j,k}$: weight parameters connecting nodes
- $x_{i,j}$: input and hidden features
- $b_{i,j}$: bias parameters for each node
  - i: layer index
  - j: index of source node
  - k: index of receiver node
- f: activation function

Deep Learning



neural network with 1 hidden layer

Neural network implementation pseudocode

1. Define layer sizes
2. Initialize parameters (k,b) randomly
3. Forward propagation
4. Calculate cost
5. Update parameters using
    - backpropagation
    - gradient descent
6. Repeat step 5 until convergence

Forward propagation:

- Hidden layer: 1) $z_2 = k_1 * x_1 + b_1$ 2) $x_2 = f(z_2)$
- Output layer: 1) $z_3 = k_2 * x_2 + b_2$ 3) $\hat{y} = f(z_3)$

Cost: $C = (\hat{y} - y)^2$

- Cost C is function of parameters k_1, k_2, b_1, b_2
- k_1, k_2 - weights, b_1, b_2 biases

Backpropagation:

- $\frac{\partial C}{\partial k_1} = \frac{\partial C}{\partial y} \times \frac{\partial y}{\partial z_3} \times \frac{\partial z_3}{\partial x_2} \times \frac{\partial x_2}{\partial z_2} \times \frac{\partial z_2}{\partial k_1}$

Gradient descent:

- $k_1 = k_1 - \alpha * \boxed{\frac{\partial C}{\partial k_1}}$

Deep Learning summary

- Applicable to all 3 ML subcategories (supervised, unsupervised, reinforcement learning)
- Advantages
  - More powerful models, applied to solve more complex problems
  - Can be used to solve a large variety of problems
  - Can be used for nonstructured data (image, video, text, audio)
  - No feature engineering is needed
- Disadvantages
  - Requires a lot of data
  - Requires a lot of computing resources (especially for large problems)
  - Black box models (not interpretable)
  - A lot of hyperparameters to tune for having a good performance

Code example: Linear regression

- **Model:** $y(k, b) = kx + b$
  - Parameters: k, b
  - Input: x
  - Output: y

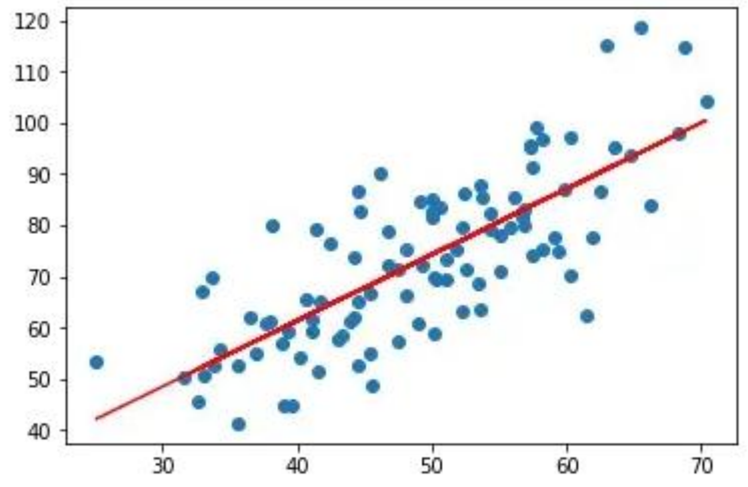
$$J(k,b) = \frac{1}{2N}\sum_{i=1}^{N} \left((kx_i + b) - y_i\right)^2$$

Initialize k, b, ε, α
 while $(J_{n-1} - J_n) > \varepsilon$
  $k_n = k_{n-1} - \alpha * \frac{\partial J(k_{n-1}, b_{n-1})}{\partial k}$
  $b_n = b_{n-1} - \alpha * \frac{\partial J(k_{n-1}, b_{n-1})}{\partial b}$
 end
Return k, b

```python
import numpy as np
import matplotlib.pyplot as plt  # To visualize
import pandas as pd  # To read data
from sklearn.linear_model import LinearRegression
```

```python
data = pd.read_csv('data.csv')  # load data set
X = data.iloc[:, 0].values.reshape(-1, 1)  # values converts it into a numpy array
Y = data.iloc[:, 1].values.reshape(-1, 1)  # -1 means that calculate the dimension of rows, but have 1 column
linear_regressor = LinearRegression()  # create object for the class
linear_regressor.fit(X, Y)  # perform linear regression
Y_pred = linear_regressor.predict(X)  # make predictions
```

```python
plt.scatter(X, Y)
plt.plot(X, Y_pred, color='red')
plt.show()
```



Data set in blue, Regression line in red

Credits:

- https://towardsdatascience.com/linear-regression-in-6-lines-of-python-5e1d0cd05b8d

Code example: Neural network (MLP)

```
#Dependencies
import keras
from keras.models import Sequential
from keras.layers import Dense

# Neural network
model = Sequential()
model.add(Dense(16, input_dim=20, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(4, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])
```

```
history = model.fit(X_train, y_train, epochs=100, batch_size=64)
```

```
Epoch 1/100
1600/1600 [==============================] - 1s 600us/step - loss:
1.3835 - acc: 0.3019
Epoch 2/100
1600/1600 [==============================] - 0s 60us/step - loss:
1.3401 - acc: 0.3369
Epoch 3/100
1600/1600 [==============================] - 0s 72us/step - loss:
1.2986 - acc: 0.3756
Epoch 4/100
1600/1600 [==============================] - 0s 63us/step - loss:
1.2525 - acc: 0.4206
Epoch 5/100
1600/1600 [==============================] - 0s 62us/step - loss:
1.1982 - acc: 0.4675
.
.
.
```

```
y_pred = model.predict(X_test)
```

Credits:

- https://towardsdatascience.com/building-our-first-neural-network-in-keras-bdc8abbc17f5

Machine Learning algorithms

- **Supervised**
  - Regression
    - Linear regression and extensions (ridge, lasso)
    - K-nearest neighbors
    - Support vector machine and its extensions (kernels)
    - Decision trees and its extensions (ensemble methods)
  - Classification
    - Logistic regression
    - K-nearest neighbors
    - Support vector machine and its extensions (kernels)
    - Naïve Bayes
    - Decision trees and its extensions (ensemble methods)
- **Unsupervised**
  - Clustering
    - K-means
  - Dimensionality reduction
    - Principal component analysis
- **Reinforcement Learning**
- **Deep Learning**: can be used for supervised, unsupervised and reinforcement learning
  - Multilayer perceptron (regression and classification)
  - Autoencoders (dimensionality reduction)
  - Convolutional neural networks (regression and classification)
  - Recurrent neural networks (regression and classification)
  - Generative adversarial neural networks (unsupervised: new data generation)
  - Transformers (regression and classification)

References and further resources

Machine Learning Specialization:

- https://www.coursera.org/specializations/machine-learning-introduction

Deep Learning Specialization:

- https://www.coursera.org/specializations/deep-learning?

3Blue1Brown, Neural Networks playlist:

- https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Recap

- Regression: Linear regression
- Classification: Logistic regression
- Clustering: K-means
- Deep Learning: Multilayer perceptron

Thank you