Object Oriented Programming (OOP)

Properteis
- Method → local function
- Attribute → local variables

Script
- Class initiation

```
class A()
```

- Create a *"instance/object"* to hold the properties of the class A()

```
a = A()
```

- Constructor function: a fuction called when the class is created

```
def __init__():
```

- Reference local variable

```
def display(self, new_name):
    self.name = new_name
    print (self.name)
```

- Call method or attribute: use dot (.)

```
a.name          # Attribute
a.display()     # Method
```

Naming Rules
- Class → start with cap-lock
- Funciton & variable → start with small letter

More info: LINK to files

Linear regression

- Target: Try to find a line that best estimate all data.
- Usage: prediction
- Data should have linear relationship
- General equation: $y = \beta_i x + \beta_0$
  - x: $input\ matrix$
  - β_i: $weight\ matrix\ for\ x$
  - β_0: $y\ intercept$
  - For multi-feature → $y = \beta_1 x_1 + \beta_2 x_2 + \beta_0$
- Loss function: **square mean error** $\boldsymbol{Loss = \frac{1}{n}\sum(y_i - \hat{y}_{i-predict})^2}$
- Programming
  - Initiate model

```
model = LinearRegression()
```

  - Train model

```
model.fit(x, y)
```

  - Access trained variables

```
model.coef_             # gives the weight matrix
model.intercept_        # gives the y intercept
```

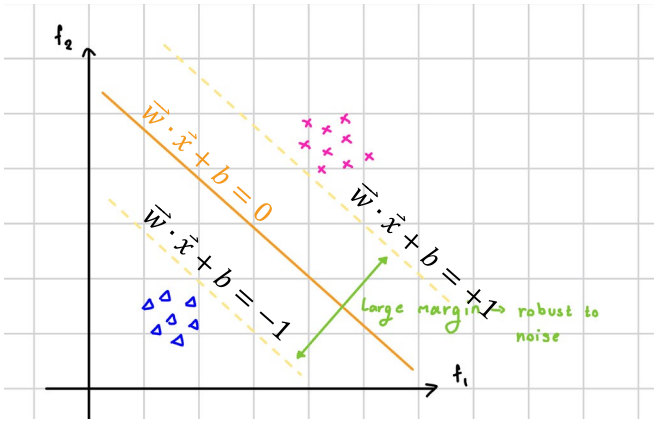
Logistics regression

- Target: try to create a function ($f(x)$) seperating <u>two classes</u>
- Usage: binary classification
- Principle: we linear combination the each feature (x_i) with a constant weight (β_i) and substitute the equation into a sigmoid funciton to predict the value. The output of a sigmoid function is how likely this feature is to be in one class, the value is 0 to 1.
    - $f(x_i) = \beta_i x_i + \beta_0 \rightarrow \beta_i x_i + \beta_{i-1} x_{i-1} + \cdots + \beta_1 x_1 + \beta_0$
    - $sigmoid: S\big(f(x)\big) = \frac{1}{1+e^{-f(x)}}$

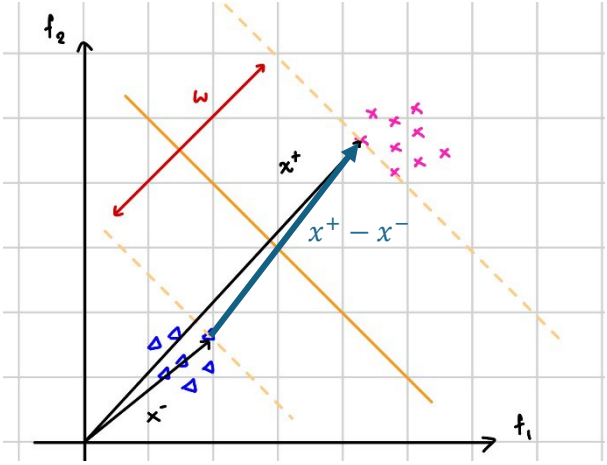
  the output form the sigmoid is them evalute using the principle of **maximum likelihood.**
- Loss function: **negative log-likelihood**

Support Vector Machine (SVM)

- Target: create a fucntion of plane (hyperplane) to seperate classes with <u>maximum margin</u> from one another.
- Usage: classificaiton
- Principle: using dot product to represent the best fit line ($\vec{w} \cdot \vec{x} + b = 0$) and the margin lines ($\vec{w} \cdot \vec{x} + b = \pm 1$),



We try to maximize the margin which is $w = (x^+ - x^-) \cdot \frac{\vec{w}}{||\vec{w}||} \rightarrow \frac{1}{2}||\vec{w}||^2$, we still have a constrain which is the two margin line ($y(\vec{w} \cdot \vec{x} + b) = 1$).



Using Lagrange multiplier, we'll be able to solve $\vec{w}, b$. The $\vec{w}$ is the weight matrix, and b is the bias, this two can form the equation of the best fit line, and the equaiton of the margin line. The margin line that we've got will go through 3 data point from the data set. Those points are called support vectors and are the only three points from the data that influcence the positon of the best fit line.

Lagrane Multiplier: $\mathcal{L} = \frac{1}{2}||\vec{w}||^2 - \sum \lambda_i \left[y(\vec{w} \cdot \vec{x} + b) - 1\right]$

*solving this eq. is the optimization process fo svm

More info: SVM Video

Support Vector Machine (SVM) – con't

- The maximum margin principle makes the SVM more robust, can withstand noises, less likely to get overfit.
- Kernel trick:
  - a technique to reference from 2D space to multi-dimension
  - instead of mapping the data to the multi-dimension space, we use kernel trick to reference and mathamatically calulate.
  - This is a very effeicent way to treat multi-feature / multi-dimenion data, as the calculating space remains in 2D. This reduce the computation requiredment, as more dimensions require more powerful compuation.
- Known kernel function
  - Linear
  - Polynomial
  - RBF (Gaussian)
  - Sigmiod
- Script
  - Initiate model

```
model = svm.SVC(kernel="linear", C=penalty)
```

  - Access trained variables

```
model.coef_             # gives the weight matrix
model.intercept_        # gives the y intercept
```

  - Apply kernel, adjust gamma to change the shape of the curve.

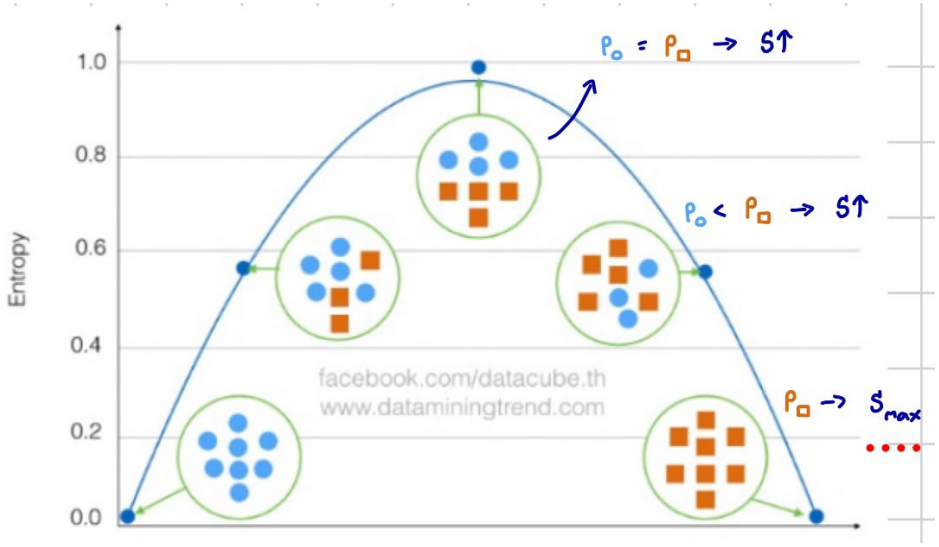
```
Model_1 = svm.SVC(kernel="linear", C=penalty)            # Linear kernel
Model_2 = svm.SVC(kernel="rbf", gamma=0.7, C=C)          # RBF kernel
Model_3 = svm.SVC(kernel="poly", degree=3, gamma="auto") # Polynomial Kernel
```

Addtional (According to Ajarn.)

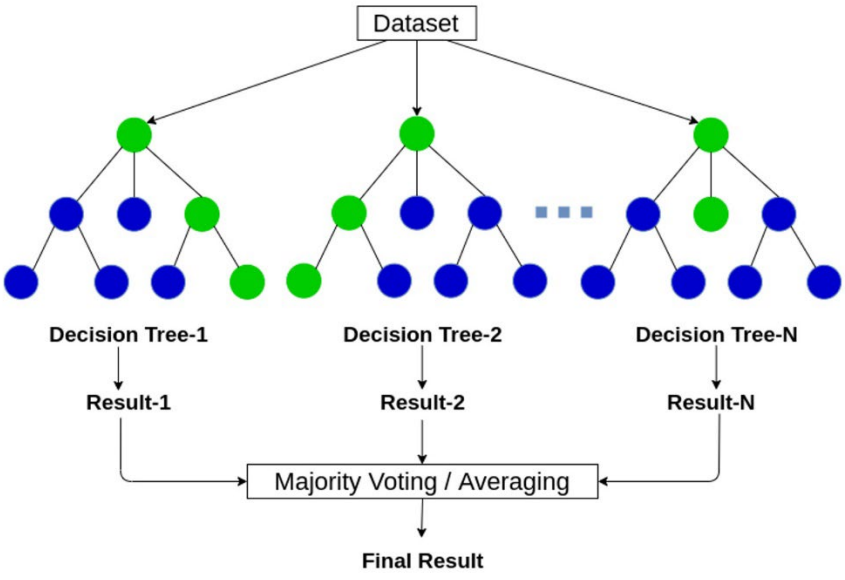
- AI: just a bunch of if-else that we've to set the thershold
- ML: the machine adjust for the best threshold automatically, buy we've to input the correct feature
- DL: automatically choose the best feature and calculate the best threshold -> required resources (dataset, compuational power)

Decision Tree

- Cross entropy:
  - Entropy (S) a measurement of disorder/chaos in the system → more S more choas
  - $\boldsymbol{S = \sum -p_i \log_2 p_i} \rightarrow p_i: propability\ of\ each\ class$
    - $S(flu|headache) \rightarrow entropy\ of\ flu\ when\ given\ data\ of\ headache$
  - Use as loss funciton of decision tree, represent how the data is classified. $p_i\ high, accuracy\ low$. We try to minimize it

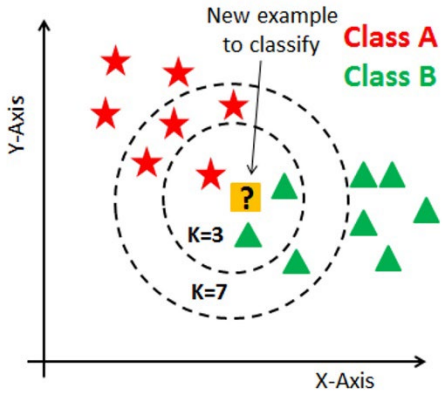


- Example: Entropy Example (page 3) → try to find the case with least entropy
- Using decision tree often create a class with only one member, which often causes a model overfitting. To solve this **Random forest algorithm** is created. This algorithm seperate the data in to multi group and create a tree from each group. The test data then be input to each tree, and the result of each tree will be operate.
  - Major voting: classification
  - Averaging: predeiction

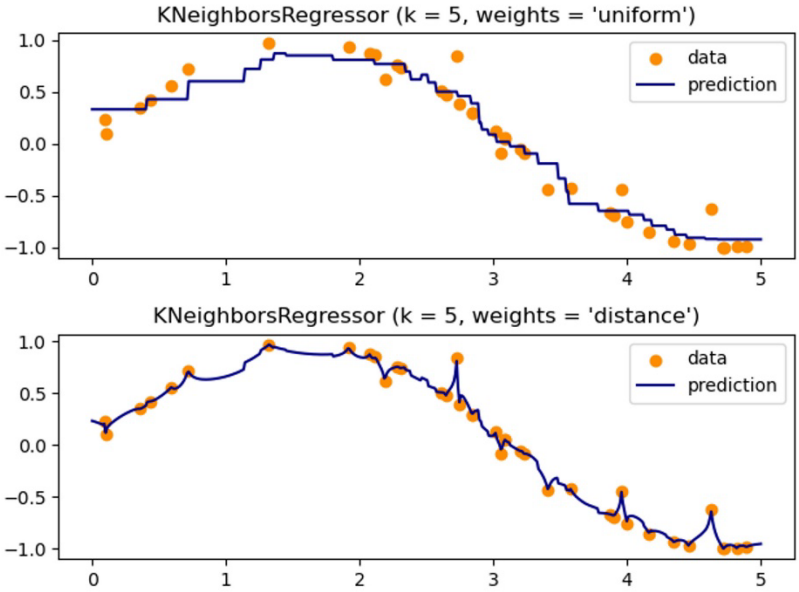


K-nearest neighber

- Target: find the class of input data considering the near by data.
- Usage: classification & predeiction
- Principle: the algorihtm choose **k** nearest data compare to the input data, and count the number of each class in the **k** sample. The data will be considered as the class with the highest quantity in the **k** sample.
- The distance can be calculate many ways, usaully Euler distance.



- For prediciton, the model calculate the predication using the **k** value near by and a human set weight.



- As you can see, training the data for a K-nearest neigber is just collecting the data, as a result, this model required a lot of memomery as all the data has to be stored.

Clustering

- Target: seperate data into k group
- Usage: find relationship between data
- Principle: data in the same class must have some what similar feature
- It's an unsupervise learning, unlike the others, no correct output to fed to the training process.
- Method
  - Assign class
  - Calculate mean
- During training process, the model randomly choose k (human set) data and assign them as the initial mean point of the clusters. Then, the model select the remaining points one by one to calculate the distance from each mean points. This point will then be assign to the cluster that is closest (least distance). The process repeats untill all data are assigned. Then the model recalculate the mean of each cluster and repeat the process from reassigning the points. The model repeat this process until the mean of each cluster doesn't change significantly.
- Method of calculating distance
  - Euler diatance: shortest distance, use for K-mean clustering
    - Suit for data which cluster size are the same.
    - Work well when data is clearly distributed
  - Mahalanobis: use for Gaussian mixer model (GMM)
    - Mahalanobis distance calculaiton is based from normal distribution.
    - $d = \sqrt{(x-\mu)^T S^{-1}(x-\mu)}$
      - x: feature vector
      - μ: mean vector
      - S: convariance matrix
    - This method also consider the variation of the data, it's suit for data that each cluster are difference in size.