Machine Learning Summary

1. **Machine Learning 접근 방법**
   1) **Machine Learning 정의**
      - **Field of study that gives computers the ability to learn without being explicitly programmed (Arthur Samuel, 1959)**
        - 명시적인 프로그래밍 없이 데이터만을 제공하여 경험 누적을 통한 성능 개선
        - **Data** 를 제공하여 문제를 해결하거나 기능을 수행하는 모델을 만들어내는 기법
        - **Input : Model Input & Output Dataset**
        - **Output : Model (Algorithm)**

      - **Classical/Conventional Approach (Non-ML / Non-NN)**
        - 목표하는 기능 및 문제에 대해 매번 모델/알고리즘을 재설계해야함.
        - 목표하는 기능 및 문제에 대해 모델/알고리즘의 **Parameter** 를 **Fine-Tuning** 해야함.

      - **Machine Learning / Neural Network Approach**
        - 목표 기능 및 문제를 표현하는 **Dataset** 을 이용한 학습을 통해 모델/알고리즘을 도출함
        - **Dataset** 에 추가 데이터를 업데이트하고, 새로운 추가 데이터를 학습하여 모델/알고리즘의 성능을 개선함.
        - 모델/알고리즘의 **Parameter** 는 학습을 통해 **Tuning** 되기에 학습에 대한 **Hyper Parameter** (반복 학습 횟수 **Epoch**, 1 회 학습 데이터 크기 **Batch Size**, 학습 속도 **Learning Rate** 등)을 조정하여 모델/알고리즘의 학습 과정을 최적화함
        - 모델/알고리즘의 성능 개선을 위해서는 좋은 **Feature** 로 구성된 많은 데이터가 필요하며, 목표하는 기능 및 문제에 따라 매번 다른 학습이 요구됨.

   2) **Machine Learning 기법 분류**
      - **Supervised Learning**
        - **Input Dataset** 과 **Output Label** (**Input** 에 대한 정답)을 모두 제공하여 모델을 학습하는 방법
        - **Input** 데이터에 따라 **Error** 가 최소화될 수 있도록 (**Loss Function** 값이 최소화도록) 적합한 정답을 도출하도록 모델/알고리즘을 학습시킴
        - 정해진 **Label** 을 추정하도록 학습되기에 **Classification** 문제를 해결하는 데에 특화됨
        - **Linear Regression, Logistic Regression, KNN (K-Nearest Neighbors), SVM (Support Vector Machine), Decision Tree, Random Forest, Neural Network**

- **Unsupervised Learning**
  - **Output Label** 없이 **Input Dataset** 만 사용하여 모델을 학습하는 방법
  - **Clustering, Dimensionality Reduction, Association Rule Learning**
- **Semi-Supervised Learning**
  - **Supervised & Unsupervised Learning** 의 조합으로 이루어짐
- **Reinforcement Learning**
  - **Agent** 가 상태**(State)**의 보상 **(Reward)** 최대화하는 방향으로 **Neural Network** 의 **Parameter** 를 최적화함

3) Dataset 구조

	Column 1 (Feature 1)	Column 2 (Feature 2)	· · ·	Column M (Feature M)
Data 1				
Data 2				
⋮				
Data N				

- **Dataset** 은 **NxM** 형태의 **Matrix** 로 구성되어있음**.** 데이터셋은 **N** 의 데이터로 구성되어있으며**,** 각 데이터는 **M** 개의 **Feature** 로 구성되어 표현됨
- **Dataset** 의 **Row Vector** 는 데이터셋의 데이터 **1** 개를 지칭함 (데이터 **N** 개)
- **Dataset** 의 **Column Vector** 는 각 데이터를 표현하는 **Feature** 를 상징함 **(Feature M** 개)
- **Dataset** 의 데이터는 **ML** 및 **Neural Network** 모델에 **1xM** 형태의 벡터로 입력됨

4) Machine Learning Implementation 순서

(1) Dataset 준비

- 목표하는 기능과 문제와 관련된 데이터셋을 확보해야함
- 데이터셋은 **ML/DL** 라이브러리와 연동되기 위해 **NxM Matrix** 형태로 구성되어야함

(2) Dataset Preporcessing

- 데이터셋의 각 **Feature** 는 다른 단위의 값으로 구성됨
- 각 **Feature** 별로 다른 단위를 사용하게 되면서 데이터의 분포가 달라지고 학습에 대한 영향력이 달라지는 **Scale** 문제로 인해 학습 과정 중 **Overfitting** 과 같은 **Fitting** 문제가 발생함
- **Standardization, Normalization** 등 데이터셋의 단위를 통일시키는 **Preprocessing** 을 수행하여 모델/알고리즘의 **Fitting** 을 개선함
- 만약 현재 데이터셋에서 사용하는 **Feature** 가 너무 많아 **Classification** 을 위한 학습이 복잡할 경우 데이터셋을 더 적은 **Feature** 로 재구성하는 **Dimensionality Reduction** 이 요구될 수 있음**.** 이 때**, PCA (Principle Component Analysis)**를 통해 설명력이 높은 소수의 새로운 **Feature** 로 데이터셋을 재구성하여 **Classification** 학습에 좀 더 쉬운 방향으로 유도할 수 있음**.**

(3) Dataset Analysis 를 통한 ML 기법 선정

Machine Learning 과 **Neural Network** 는 데이터셋에 매우 의존적이기 때문에 데이터셋의 **Feature** 성격에 따라 적합한 모델을 선정해야함.

➢ **Multi-Colinearity 존재할 경우**

- 우선적으로 데이터셋의 **Feature** 간 선형성 **(Colinearity)**를 파악해야함.
- 다수의 **Feature** 로 구성된 데이터셋에서 **Feature** 간에 서로 독립적인 것이 이상적이나 **Feature** 간 서로 영향을 주는 **Multi-Colinearity** 가 존재할 수 있음.
- 이를 파악하기 위해 **Feature M** 개에 대한 **MxM Correlation Matrix** 를 구성해서 각 **Feature** 가 서로에게 **Correlation** (선형성 존재여부)을 가지는지 파악함.

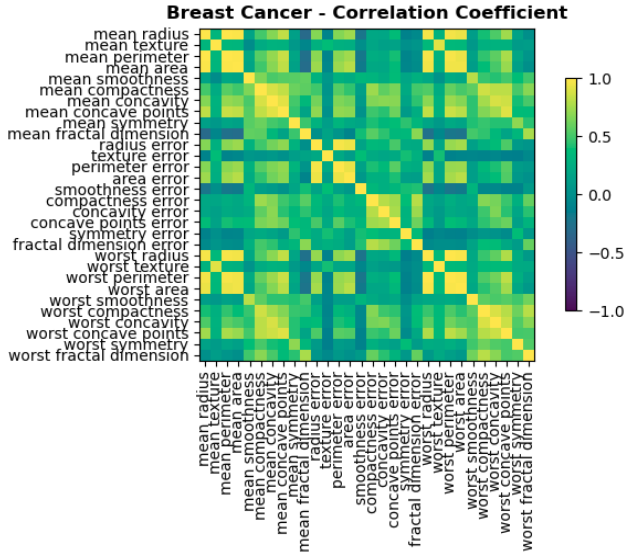


Fig 1: Correlation Matrix (Breast Cancer Dataset)

- **Multi-Colinearity** 가 존재하는 **Feature** 사이에는 **Linear Classifier** 는 적용할 수 없음.
- **Multi-Colinearity** 가 존재하는 **Feature** 사이에 **Classification** 을 위해서는 보다 복잡한 구조의 모델/알고리즘이 요구됨.
  (ex : Neural Network, SVM, Random Forest, Decision Tree)

➢ **데이터셋의 Non-Linearity 존재할 경우**

- 데이터셋 **Feature** 의 물리적 특성이 달라 **Non-Linearity** 가 존재할 수 있음.
  (ex : 2D 이미지 **vs 3D Translation / 2D** 이미지 **vs 3D Rotation)**
- **Non-Linear** 데이터셋에 대해서는 **Linear Classifier** 를 적용할 수 없으며, 더 많은 **Parameter** 를 학습하여 **Non-Linear** 한 추정을 할 수 있는 복잡한 모델/알고리즘이 요구됨. **(ex : Neural Network)**

(4) Training/Validation/Test Dataset 분리

데이터셋은 학습 및 성능 평가를 위해 전체에서 일정 비율로 **Training Set, Validation Set, Test Set** 으로 각각 분리하며, 서로 중첩이 없어야하며 **Random** 하게 선정됨.

- **Training Set**
  - **ML** 모델 학습을 위해 사용하는 데이터셋
- **Validation Set**
  - **ML** 모델 학습에 필요한 **Hyper Parameter** 를 찾기 위해 사용하는 데이터셋
  - 학습이 종료된 후 **Test Set** 이 주어지기 전까지 학습 과정 중 모의 평가를 하기 위해 사용
  - 모의 평가를 통해 **Unseen Data** 에 대한 정확도를 확인하여 **Test Set** 에 대한 정확도를 가늠할 수 있음.
  - 모의 평가를 통해 **Training Set, Validation Set 2** 개에 대해 모두 정확도가 높게 나오게 만듬으로서 **Training Set** 에만 최적화되는 **Overfitting** 을 방지함.
- **Test Set**
  - 학습된 모델을 평가하기 위한 데이터셋

(5) Training 수행 + Optimization

- **Loss Function 선정**
  - 목표하는 **Output** 을 최대한 원하는 정답에 가깝게 만들기 위해 **Error** 를 최소화 시키는 방향으로 학습해야함
  - 사용하는 모델의 **Output** 형태와 **Label/Target Output** 의 형태에 따라 **Error** 의 형태와 종류가 다름. 그러므로 최소화하려는 **Error** 의 형태에 따라 적합한 **Loss Function** 을 선정해서 학습을 수행함.
    (ex : MSE (Mean Squared Error), Cross Entropy Error)

- **Optimizer 선정**

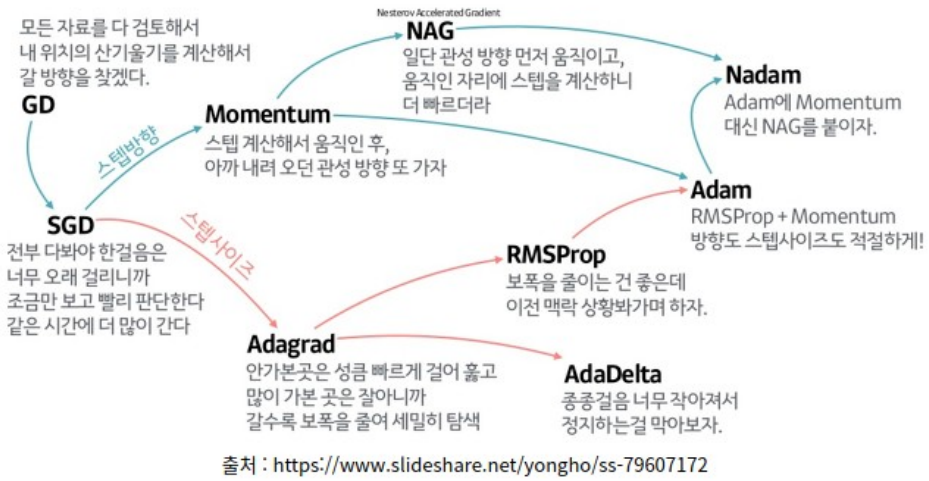


Fig 2: Optimizer Types (https://light-tree.tistory.com/141)

- **ML/NN** 모델 학습 방향과 방식에 따라 **Optimizer** 를 선정하여 사용함

- **Training 수행 및 수행 결과 분석**
  - **Training** 수행 시 **Training Set** 과 **Validation Set** 에 대해 각각 학습을 수행하여 **Overfitting** 을 방지함.
  - **Training Set** 에서의 **Loss Functin, Validation Set** 에서의 **Loss Function** 을 각각 확인하여 **Overfitting, Underfitting** 을 판단함.

5) Machine Learning Dataset Preprocessing 및 Analysis 기법

- **Dataset Preprocessing**
  - ➢ **Dataset Rescaling**
    - **Rescaling 이 필요한 이유**
      - **Feature** 별로 사용하는 단위가 다른 경우 어떤 단위를 사용하는 가에 따라 데이터 분포가 극심하게 달라짐.
        (ex : cm 단위의 데이터는 **m** 단위로 표현할 경우 서로 차이 없이 분포됨)
      - **Distance** 기반 모델 **(ex : SVM)** 경우 사용하는 단위에 따라 성능이 달라지게됨. 그러므로 단위 **Scale** 에 의한 영향을 최소하기 위해 **Feature** 들의 사용 단위/**Scale** 을 통일시켜야함.

    - **Standardization (Z-Score / 표준화)**
      - **Standardization 공식 및 효과**

$x' = \frac{x - \mu}{\sigma}$	X' : 표준화된 데이터 (Z-Score) X : 원본 데이터 μ : 특정 Feature 에 대한 X 의 평균 σ : 특정 Feature 에 대한 X 의 표준편차

◆ **M** 개의 **Feature** 로 구성된 데이터에 대해 각 **Feature** 의 평균과 표준편차를 사용하여 각 **Feature** 에 대해 **Z-Score** 로 데이터를 **Rescaling** 하여 표준화함.

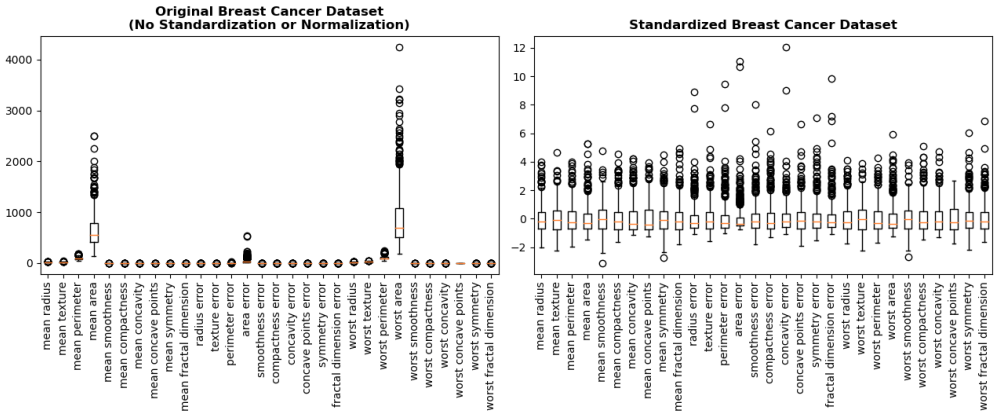


Fig 3: Dataset Standardization

◆ **Standardization** 을 통해 데이터셋의 각 데이터는 **1 x M** 으로 구성된 **Z-Score** 데이터로 재구성됨.

◆ **Standardization** 을 통해 데이터셋의 모든 **Feature** 는 동일한 평균과 표준편차를 가지게 되며, 동일한 범위의 분포 내에 재구성될 수 있음.

◆ **Standardization** 을 통해 각 **Feature** 별로 다른 **Scale** 을 가지는 것을 동일한 분포를 가지게 하여 **Scale** 을 통일시킬 수 있음.

- **사용 API**
  - ◆ **Scikit Learn – StandardScaler 의 fit 함수, transform 함수 (sklearn.preprocessing.StandardScaler)**
    fit 함수를 통해 평균과 표준편차를 구하고, **transform** 함수를 통해 **Feature** 데이터를 **Z-Score** 로 표준화함.

- **Normalization**
  - **Min-Max Normalization 공식 및 효과**

$x' = \frac{x - max(x)}{max(x) - min(x)}$	X' : 정규화된 데이터 X : 원본 데이터 max(X) : 특정 Feature 에 대한 X 의 최대값 min(X) : 특정 Feature 에 대한 X 의 최소값

  - ◆ **M** 개의 **Feature** 로 구성된 데이터에 대해 각 **Feature** 의 최대값과 최소값을 사용하여 각 **Feature** 에 대해 **0 ~ 1** 사이의 값으로 **Rescaling** 하여 정규화함

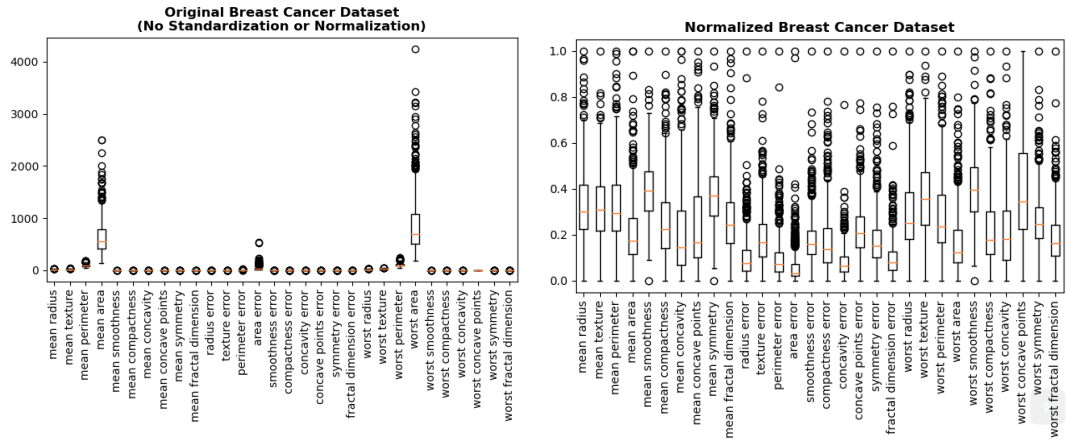


Fig 4: Dataset Normalization

  - ◆ **Min-Max Normalization** 은 데이터셋의 각 데이터를 **0 ~ 1** 사이 값으로 구성된 **1 x M** 데이터로 재구성함.
  - ◆ **Min-Max Normalization** 을 통해 각 **Feature** 별로 다른 **Scale** 을 가지는 것을 **0 ~ 1** 사이 값으로 통일시킴으로서 **Scale** 을 통일시킬 수 있음.

- **사용 API**
  - ◆ **Scikit Learn – MinMaxScaler** 의 **fit** 함수, **transform** 함수 **(sklearn.preprocessing.MinMaxScaler)**
    fit 함수를 통해 최소값과 최대값을 구하고, **transform** 함수를 통해 **Feature** 데이터를 **0 ~ 1** 사이로 정규화함.

➢ Dimensionality Reduction (차원 축소)

I. 차원 축소가 필요한 이유

- 데이터셋의 데이터가 과도하게 많은 **Feature** 로 구성되어있을 시 학습에 필요한 시간과 학습 속도가 매우 느림.
- **Feature** 가 너무 많을 경우 학습하기 쉬운 **Feature** 에만 특화되는 **Overfitting** 이 발생할 수 있음.

II. Simple Projection 의 문제점 – Swiss Roll 문제

- **Feature** 의 개수를 줄이기 위해 데이터를 특정 **Feature (Dimension)**로만 구성된 공간에 **Projection** 함. 그러나 **Feature (Dimension)**을 줄이는 행위는 정보 손실이 발생할 뿐만 아니라 오히려 **Classification** 하기 어려운 분포가 되는 경우가 있음.

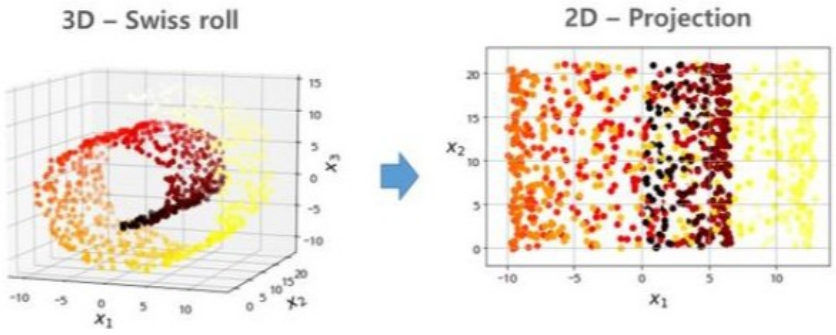
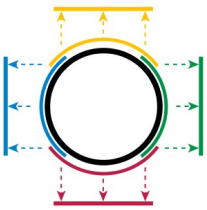


Fig 5: 3D Swiss Roll - 2D Projection

- **3D Swiss Roll** 예제가 대표적인 경우임. **3** 개의 **Feature** 로 구성된 데이터셋은 **3** 차원 좌표계에 표현될 수 있음. 그러나 데이터셋을 **1** 개의 **Feature (Dimension)**을 제거하고 **2D** 평면에 바로 **Projection** 할 경우 각 **Label** 별 데이터가 오히려 섞이는 현상이 발생함. 그러므로 단순히 **Projection** 하는 것은 효과적인 데이터셋 전처리 방법이 아니라는 것을 알 수 있음.

III. Manifold Approach

- **Manifold** 는 다양체라고도 하며 국소적으로 **Euclidean** 공간과 닮은 위상 공간임

	전체 구조는 원이지만 노란색 구간에서만의 미시적인 관점에서 원 전체를 느낄 수 없으며, 미시적으로 직선이라 생각할 수 있음.

- **Manifold** 는 국소적인 구간과 미시적인 관점에서 전체 구조를 다른 구조와 위상적으로 동형이라는 것을 사용하여 데이터셋의 **Feature** 를 재정의하고, 데이터셋을 보다 이해하기 쉬운 구조로 재구성함.

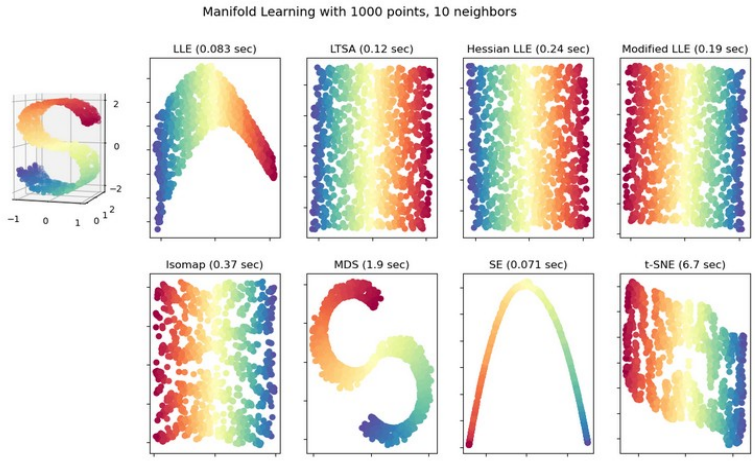


Fig 6: sci-kit learn : Comparison of Manifold Learning Methods

- **Manifold** 를 응용하여 고차원/다차원에서 복잡한 데이터셋을 재구성하고, 이를 기반으로 모델 학습을 수행하는 것을 **Manifold Learning** 이라함.

IV. PCA (Principle Component Analysis)

- **PCA** 는 데이터셋의 전체를 최대한 많이 설명할 수 있는 새로운 **Feature Axis (Dimension Axis)**를 새로 산출하는 방법임.
- **PCA** 는 표준화된 데이터셋이 특정 **Eigen Vector** (고유 벡터)에 **Projection** 되었을 시에 최대한 많은 데이터를 수용되어야함. 최대한 많은 데이터셋을 수용/설명할 수 있는 특정 **Eigen Vector** 를 구하기 위해 **Projection** 된 결과의 **Variance (Eigen Value)**가 최대가 되는 경우를 찾음.

> **※ Eigen Vector 를 사용하는 이유 : 돌아가지 않는 고정된 Axis 필요**
> **새로운 Feature/Dimension Axis 로 사용될 것이기 때문에 Matrix 에 곱해져도 Translation 만 변하고, Phase 가 유지되는 Vector 가 필요함**

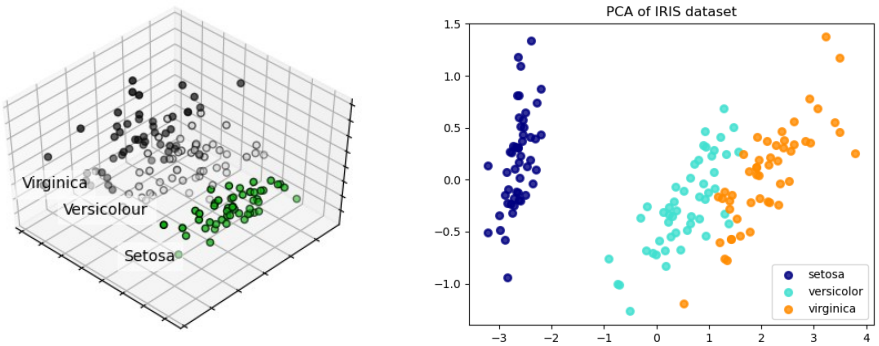


Fig 7: sci-kit learn : PCA Example with Irisi Data-set

- **PCA** 는 다수의 **Eigen Vector** 인 **PC (Principle Component)**를 제공함. **M** 개의 **Feature** 로 구성된 데이터셋은 **PCA** 를 통해 최대 **M** 개의 **PC** 를 생성할 수 있음.
- **1st PC** 는 **PC** 중 제일 설명력이 강한 새로운 **Feature/Dimension Axis** 임. **PCA** 를 통해 구해진 **PC** 를 새로운 **Feature** 로 사용하여 데이터셋을 재구성함으로서 설명력을 최대한 유지하면서 이전보다 더 적은 **Feature** 로 데이터셋을 구성할 수 있음.

- **Dataset Analysis**
  - ➢ **Linearity 판단 : Correlation Matrix**
    - **I. Correlation**
      - 특정 데이터 **X, Y** 간의 선형성 여부를 알려주는 지표
      - 선형성 여부와 강도만 알려줄 뿐 선형성의 크기를 알려주지는 않음 **(Correlation ≠ 기울기)**
    - **II. Correlation Matrix 를 사용하는 이유**
      - 데이터셋의 **Feature** 간 독립적인 것이 이상적이지만 항상 모든 데이터셋의 **Feature** 는 독립적이지 않음**.**
      - 특정 **Feature** 의 변화가 다른 **Feature** 에 영향을 주면서 **Output** 에도 영향을 주기 때문에 그에 따른 **Feature** 간 선형성 관계 여부와 강도를 알아야함**.**
      - **Correlation Matrix** 를 통해 다수 **Feature** 간 선형성 **(Multi-Colinearity)**가 발생할 경우 그에 따라 적합한 모델과 전처리 기법을 사용해야함**.**

6) Machine Learning 설계 주요 중점 : Model Fitting

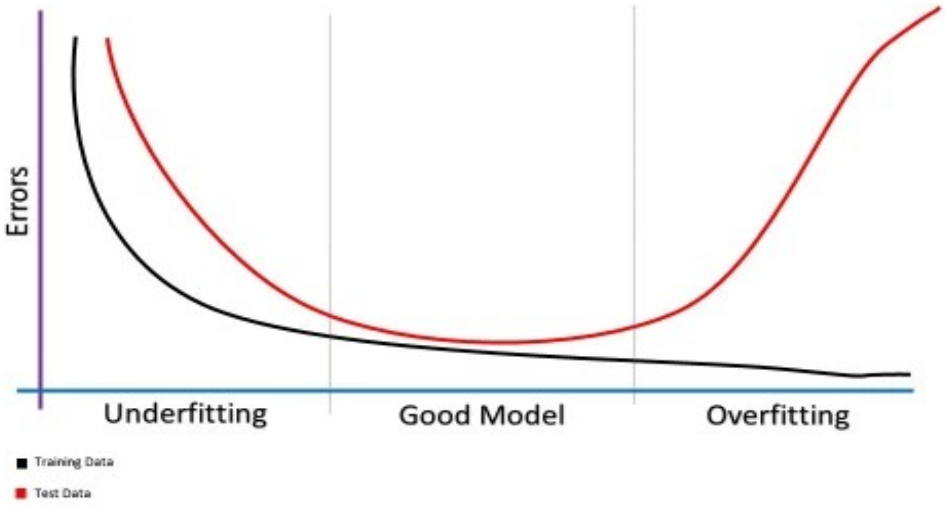


Fig 8: Overfitting vs Underfitting

(https://meditationsonbianddatascience.com/2017/05/11/overfitting-underfitting-how-well-does-your-model-fit/)

- **Overfitting**
  - ➢ 학습 과정 중 **Training Loss** 와 **Validation Loss** 가 서로 유사하지 못하면서 **Validation Loss** 가 **Training Loss** 보다 높은 상황
  - ➢ **발생하는 이유**
    - 과도한 양의 **Feature** 로 구성된 매우 복잡한 모델을 학습하는 과정에서 학습하기 쉬운 **Feature** 에 대해서만 하면서 모델이 **Training Set** 에 특화되어 버리기 때문임**.**
    - **Training Set** 에 특화되어버리기 때문에 다른 데이터셋인 **Validation Set** 에 대해서는 낮은 성능을 보임**.**

➢ **해결방법**

Overfitting 이 발생하는 이유는 데이터셋이 너무 많은 **Feature/Dimension** 으로 구성되어있는 고차원 모델이기 때문임**.** 많은 **Feature/Dimension** 으로 구성된 만큼 데이터를 더 사용하거나 **Feature/Dimension** 을 축소시켜서 모델을 간소화시켜야함

- **데이터셋의 Feature/Dimension 개수를 줄이는 작업 (Dimensionality Reduction** 또는 **Feature** 일부만 선정해서 학습**)을 통해 입력 학습 데이터셋을 간소화시킴**
- **더 많은 학습 데이터를 모아서 제공함**
- **학습 데이터의 노이즈를 줄임**

- **Underfitting**
  - ➢ 학습 과정 중 **Training Loss** 와 **Validation Loss** 가 낮게 유지되지 못하면서 **Training Loss** 가 **Validation Loss** 보다 높거나 같은 상황
  - ➢ **발생하는 이유**
    - 학습에 사용되는 **Feature** 가 너무 적어서 학습 데이터를 모두 표현할 수 있는 모델을 만들지 못하는 상황임**.**
    - **Training Set** 에서 제대로 학습을 하지 못했기에 **Training Set** 에서 낮은 성능을 보이며**, Validation** 에서도 높은 성능을 보이지 못함**.**
  - ➢ **해결방법**

    Underfitting 이 발생하는 이유는 데이터셋에 사용되는 **Feature/Dimension** 이 적어서 모델이 학습할 정보량이 부족하기 때문임**.** 그러므로 데이터셋에 사용되는 **Feature** 의 개수를 증가시켜야함**.**

    - 학습 데이터셋에 더 많은 **Feature/Dimension** 를 추가해서 낮은 차원의 데이터셋을 고차원으로 개선하여 모델이 각 데이터로부터 더 많은 정보를 학습할 수 있게 만듬
    - 기존에 사용하던 **Feature** 보다 더 좋은 **Feature** 를 선정해서 학습함**.**
    - 모델의 제약을 줄임**.**

2. 다양한 기초 Classification 기법

1) Linear Regression

- **주요 특성 및 원리**
  - **X, Y** 사이에 서로 **Linear** 한 관계가 있다는 가정하에 **X, Y** 의 **Linear** 한 관계를 표현할 수 있는 수식 (모델 / **Y** = **f(X)**)을 찾아내는 방법.
  - **X** 는 **N x M** 으로 구성된 데이터셋에서 **1 x M Vector** 형태로 **M** 개의 **Feature** 로 구성된 데이터임. **Y** 는 **X** 의 **Feature** 의 조합으로 구성된 모델의 **Output** 임.
    Linear Regression 모델 : $Y=\sum_{i=1}^{M} X_i = w_0 * X_0 + w_1 * X_1 + w_2 * X_2 \ldots + w_M * X_M$
  - 모든 **ML** 기법의 기본이 되는 모델 학습 방법이며 여러 종류의 데이터셋을 최대한 **Linear Regression** 의 형태에 유사하게 만들어서 학습을 수행하려함.
  - 모델의 **Prediction Y'**와 데이터셋의 정답 **Y** 간의 **Error** 를 최소화하는 방향으로 **Gradient Descent** 를 수행하여 **Training Set** 과 **Validation Set** 에서 최소의 **Error** 를 가지는 **f(X)** 모델을 구함

- **한계점**
  - **Input X** 와 **Output Y** 사이에 서로 **Linear** 한 관계가 있다는 가정하에 모델을 도출하는 학습을 하기 때문에 **Non-Linear** 한 데이터셋, **Multi-Colinearity** 데이터셋에 대해서는 학습할 수 없음.
  - **Non-Linear** 한 상황, **Multi-Colinearity** 상황에 대해 데이터셋의 **Feature** 를 재구성하여 데이터를 **Linear** 한 분포로 유도하는 기법이 요구됨

2) Ridge / Lasso / ElasticNet

- **주요 특성 및 원리**
  - **Regularization (Shrinkage)**
    : Multi-Colinearity 상황에서 **Input X** 의 **Feature** 사이에 **Linearity** 가 발생하여 모델 추정 시 각 **Feature** 에 대해 독립적인 **Weight** 를 보장하지 못하게됨.
    : 이에 대해서 **Regularization** 을 통해 **Multi-Colinearity** 가 발생하는 **Feature** 에 대해 **Penalty** 값을 부여하여 유의미한 **Feature** 만 모델 생성에 기여하도록함.
  - **Ridge**
    : L2 Penalty 를 사용하여 **Multi-Colinearity** 를 가진 **Feature** 의 **Weight** 를 줄임으로서 모델에 대한 기여도를 약화시킴.
    : L2 형태로 **Penalty** 가 들어가기 때문에 완전히 **0** 이 되지 못하고, **0** 에 근사화됨.
  - **Lasso**
    : L1 Penalty 를 사용하여 **Multi-Colinearity** 를 가진 **Feature** 의 **Weight** 를 **0** 으로 수렴하게하여 유의미한 **Feature** 만 남게하는 **Feature Selection** 을 수행함
  - **ElasticNet**
    : Ridge 와 **Lasso** 를 같이 사용하여 **Lasso** 가 과도하게 **Feature** 를 없애는 것을 **Ridge** 를 통해 완화시킴.

- **한계점**
  - **Ridge : Feature** 를 **0** 으로 만들지 않기 때문에 불필요한 **Feature** 를 없애지 못함.
  - **Lasso : Colinear** 한 **Feature** 가 다수 존재하는 경우 **1** 개만 **Penalty** 를 가할 수 있음.

3) KNN (K-Nearest Neighbors)

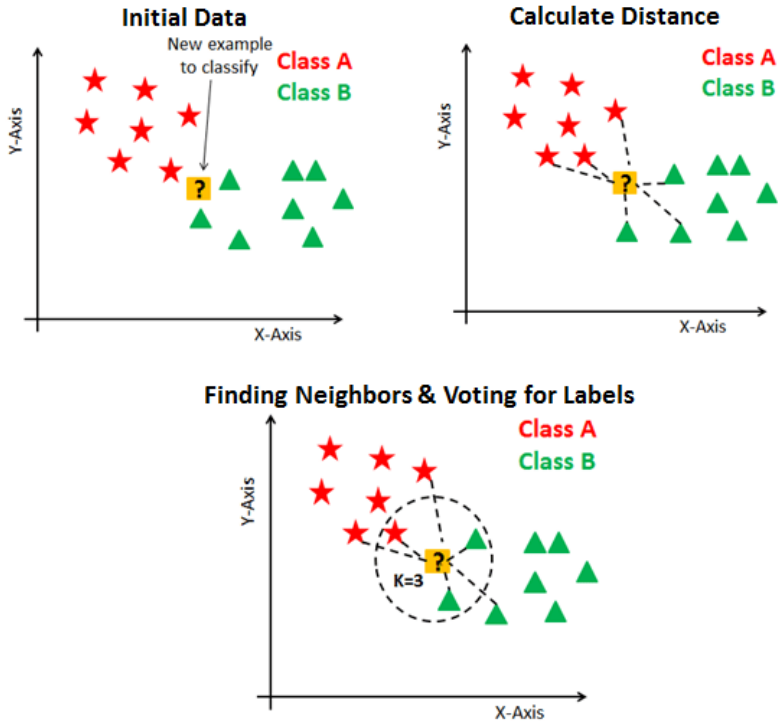


Fig 9: KNN Classification Process

(https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn)

- **주요 특성 및 원리**
  - 전체 데이터셋 분포에서 입력 데이터와 거리가 가까운 데이터를 **K** 개 선정함. 선정된 **K** 개의 데이터의 **Label** 에 대해 **Majority Voting** 을 수행하여 입력 데이터에 대한 **Label** 을 정함.
  - 입력 데이터와 가깝게 배치된 데이터들이 서로 유사한 **Label** 을 가진다는 가정하에 설계된 **Classification** 방식.
  - 별도의 학습 과정이 필요 없으며, 데이터만 충분히 있다면 **Classification** 을 손쉽게 구현할 수 있음.

- **한계점**
  - 효과적인 **Classification** 을 위해서는 많은 데이터를 요구함
  - **Label** 별로 **Linear** 한게 분포된 데이터에 대해서는 효과적이나 **Non-Linear** 하게 분포된 경우 효과적인 **Classification** 을 수행할 수 없음
  - 효과적인 데이터셋의 분포 및 **K** 개 최단 거리 데이터 선정을 위해 상황에 따른 전처리 과정과 최단 거리 연산이 요구됨.

4) Logistic Regression

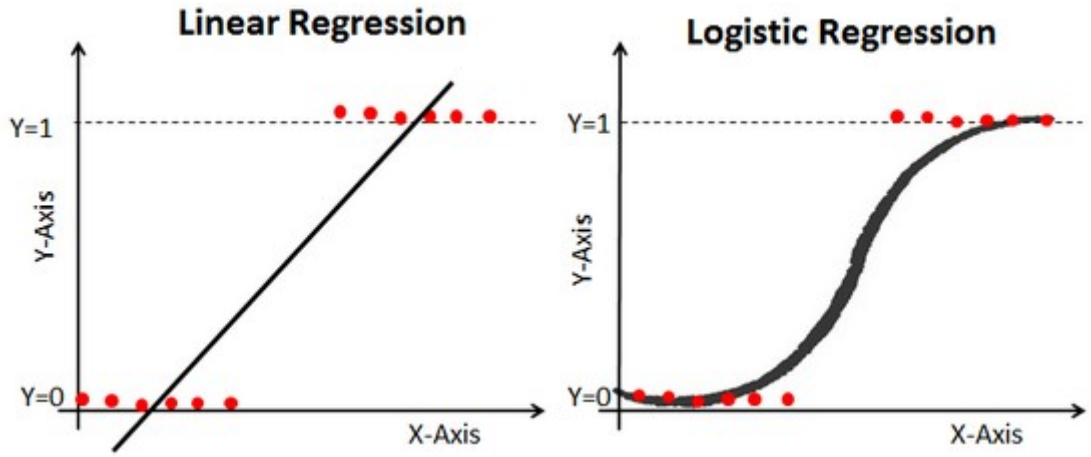


Fig 10: Logistics Regression

(https://nittaku.tistory.com/478)

- **주요 특성 및 원리**
  - **Prediction** 하고자하는 **Output Y** 가 범주형 **(Categorical)**이면서 **0** 또는 **1** 인 경우 사용하는 회귀 분석임**.**
  - 기존의 **Linear Regression** 은 **Continuous Input X, Continuous Output Y** 에 대해 **Fitting** 될 수 있는 모델을 찾는 것이 주요 목적이었음**.** 그러나 **Discrete** 한 **Y** 로 구성된 데이터셋에 대해서는 효과적으로 **Fitting** 할 수 없음**.**
  - **Discrete Output** 에 대한 **Fitting** 을 위해 **Linear** 한 직선이 아닌 **Curve** 를 사용하여 **Fitting** 함**.**
  - 이를 위해 **Input X** 에 대해 각각 **Output Y** 가 발생할 수 있는 확률을 통해 **Odds** 를 구하고**,** **Logit** 함수를 도출하여 **Curve Fitting** 함**.**

- **한계점**
  - **Output Y** 가 **Discrete** 한 상황**(Yes or No / 0 or 1)**에 대해서 적용할 수 있기 때문에 사용 가능한 상황은 매우 제한적임**.**

5) Naive-Bayes Classifier

- **주요 특성 및 원리**
  - **Y** 라는 **Target Output** 이 있을 때**,** **Feature X** 에 대한 확률 분포가 있다고 가정함**.**
  - **Feature** 가 주어졌을 때 확률이 제일 높은 **Label** 을 찾을 확률**(P(Y|X))**을 찾도록 모델을 학습시킴**.** **Bayes Rule** 을 이용하여 간접적으로 데이터로부터 답을 유추하도록 학습됨**.**

3. SVM (Support Vector Machine)

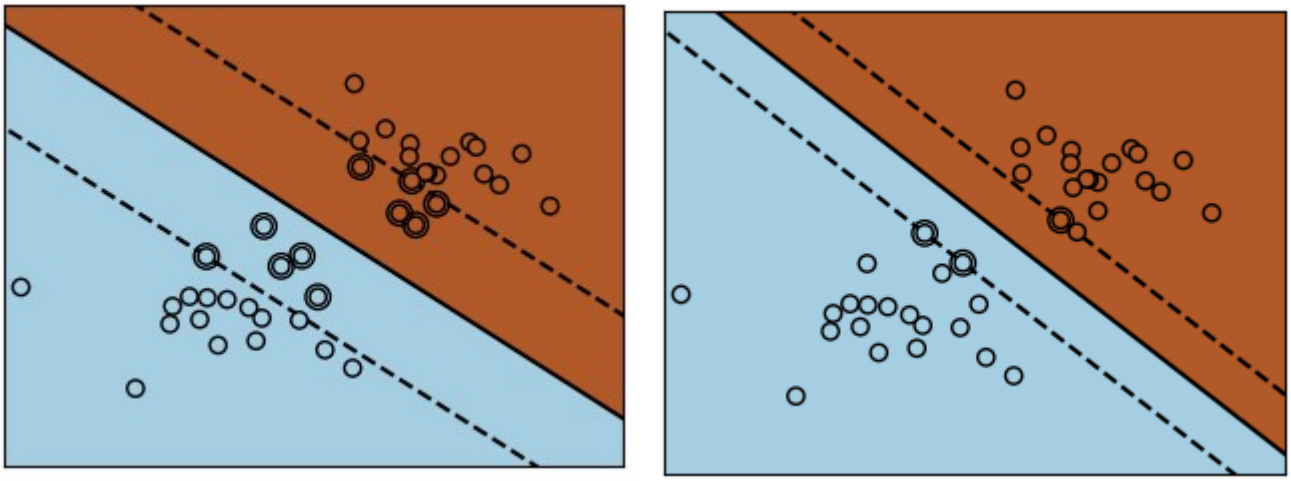


Fig 11: sci-kit learn : SVM Margins Example

1) 주요 특성 및 원리

- **Linear Regression** 은 **Training Set** 내에서 **Prediction Y'**와 **Tareget Output Y** 간의 **Error** 를 최소화 시키는 방향으로 모델/**Decision Boundary** 를 산출하도록 학습됨.
- **SVM** 은 **Linear Regression** 과 다르게 전체 데이터셋에서 **Label** 간의 간격이 최대가 되도록 **Decision Boundary** 를 산출하도록 하는 기법임.
- 전체 데이터셋에 대해서 직접 **Decision Boundary** 를 산출하기 때문에 별도의 학습이 요구되지 않음.
- **Non-Linear** 한 데이터셋 분포에 대해서는 **Kernel Trick** 을 적용하여 **Dimensionality Reduction** 과 유사하게 데이터셋을 좀 더 **Linear** 한 분포로 재구성함.

2) 한계점

- **SVM** 은 기본적으로 **Label** 간의 거리를 최대화하는 방향으로 **Decision Boundary** 를 만들기 때문에 거리를 산출하는 데에 영향을 주는 **Feature** 의 단위에 크게 영향을 받음. 데이터셋의 **Feature** 에 사용되는 단위/**Scale** 에 따라 데이터 분포가 바뀌고 그에 따라 **Decision Boundary** 의 성능이 크게 달라짐. 그러므로 데이터셋에 대한 **Rescaling** 전처리가 요구됨.
- **Non-Linear** 한 데이터셋 분포에서 **Kernel** 의 종류에 따라 성능이 크게 달라짐.

4. Decision Tree

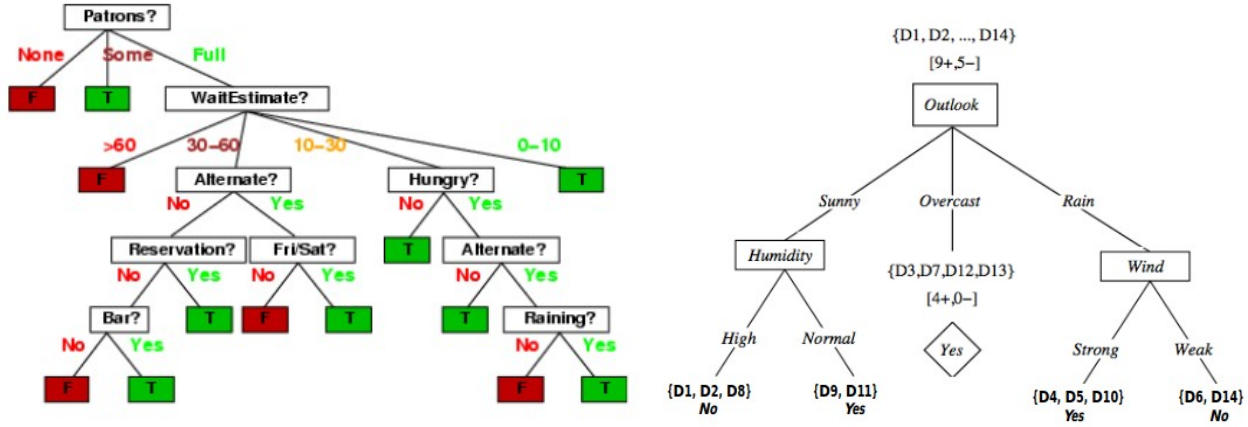


Fig 12: Decision Tree Structure

1) 주요 특성 및 원리

- 스무고개 하듯이 데이터셋의 **Feature** 를 하나씩 확인하면서 각 **Feature** 별로 조건 만족 여부에 따라 **Tree** 를 탐색하면서 **Classification** 을 수행함.
- **Decision Tree** 에서 매 선택을 해야하는 순간에 **Uncertainty** 가 존재함.
- 매번 선택을 할 때마다 **Uncertainty** 가 최대로 감소하는 방향으로 **Tree** 를 탐색함. **(※ 선택을 할 수록 더 확고한 결론에 도달하는 것을 목표함)**
- **Decision Tree** 에서는 **Information** 을 **Uncertainty** 의 양이라 정의하며, **Information** 에 대한 기대값을 **Entropy** 라 지칭함. **Information Gain** 은 선택하면서 **Uncertainty** 가 얼마나 감소하는지 보여주는 지표임.
- **Decision Tree** 는 **Information Gain** 이 큰 방향으로 **Tree** 를 탐색함으로서 선택할 때마다 더 확고한 **Classification** 결론에 도달하게 만드는 **Greedy Algorithm** 임.

2) 주요 문제점

- **Tree** 를 구성하는 **Feature** 를 어떤 것을 사용할 것인가**?**
- **Tree** 의 가지치기 **(Split)**을 어느 지점까지 할 것인가**?**
- 새로운 **Feature** 데이터가 추가될 때마다 새로운 가지치기 **(Split)**이 요구됨.
- **Decision Tree 1** 개가 만들어지면 수정하는 것이 매우 힘듬.

5. Random Forest

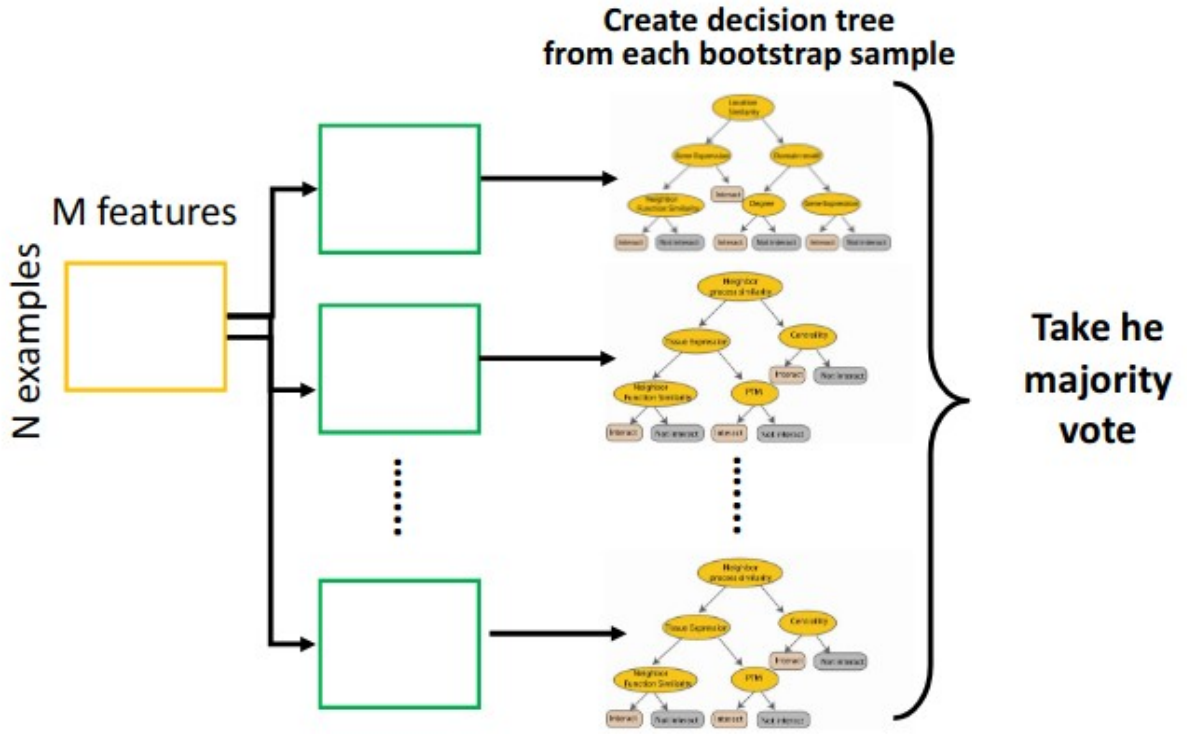


Fig 13: Random Forest Structure / Majority Voting

1) 주요 특성 및 원리

- **Decision Tree** 를 **1** 개 만들면 **Classification** 방식이 고정되는 단점이 있음.
- 이를 보완하기 위해 **Random Forest** 는 데이터셋의 모든 **Feature** 를 사용하는 **Decision Tree** 와 달리 데이터셋의 **Feature** 의 일부를 다양하게 조합하여 여러가지 **Decision Tree** 를 만듬.
  (Bagging : 동일한 데이터셋에 대해 일부 **Feature** 만 사용해서 여러 버전을 준비함 / 중복 추출)
- 다양한 **Feature** 의 조합으로 만들어진 다수의 **Decision Tree** 는 **Input X** 에 대해 각각 **Classification** 결과를 도출함. 도출된 결과에 대해 **Majority Voting** 을 수행하여 최종 **Classification** 결과를 결정함.

2) 한계점

- **Decision Tree** 를 여러 개로 구성되기 때문에 대규모의 **Tree** 를 저장할 수 있는 메모리 공간이 요구됨.
- **Decision Tree** 여러 개를 모두 탐색해야하기 때문에 **Time-Critical** 한 시스템에 적용시키기에는 부적합함.

6. ANN (Artificial Neural Network)

1) Artificial Neuron (Perceptron) Modeling

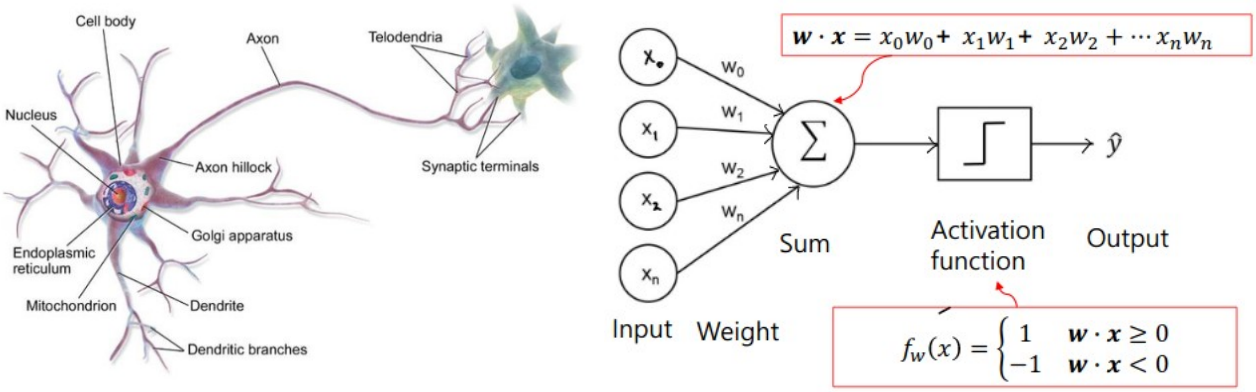


Fig 14: Artificial Neuron (Perceptron) Mathematical Modeling

- 인간의 뉴런은 외부 전기적 자극을 받아 종합되어 외부로 전기적 자극을 방출함.
- **Artificial Neuron (Perceptron)**은 인간의 뉴런의 작동 과정의 수학적 모델임.
  - **외부 전기적 자극을 받아 종합**

    = 입력 데이터에 대해 Weighted Sum 을 계산 (= Linear Regression)
  - **외부로 전기적 자극을 방출**

    = Weighted Sum 을 Activation Function 에 대입하여 다음 단계에 연결된 뉴런에 입력으로 사용
- **Artificial Neuron** 은 **ReLU, Leaky ReLU, Sigmoid** 등 **Non-Linear** 함수를 **Activation** 함수로 사용함. 왜냐하면 **Non-Linear** 한 **Input/Output** 데이터셋에 대해 **Non-Linear** 한 **Prediction** 을 수행하기 위해 **Non-Linear Activation Function** 을 사용함.

2) Perceptron

- **주요 특성 및 원리**
  - **Perceptron** 은 **Prediction Output Y'**와 **Target Y** 의 **Error** 가 최소화 되도록 **Gradient Descent** 를 사용하여 **Weight** 를 최적화하는 학습을 수행함.
  - 단일 **Perceptron** 은 **Input** 에 대한 **Weighted Sum** 에서 **Gradient Descent** 를 통해 **Weight** 를 최적화하기 때문에 **Linear Regression** 과 유사함.
- **한계점**
  - 단일 **Perceptron** 으로는 **Non-Linear** 데이터셋에 대해 효과적인 **Prediction** 을 할 수 없음. 이를 극복하기 위해 다수의 **Perceptron** 을 여러 **Layer** 로 구성한 **Multi-Layer Perceptron (MLP) / Neural Network** 를 사용함.

3) Neural Network / MLP : Multi-Layer Perceptron

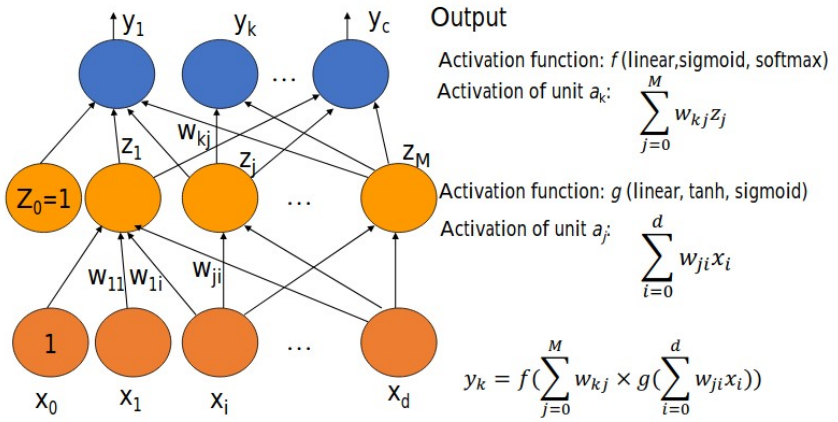


Fig 15: Neural Network / MLP Structure

- **주요 특성 및 원리**
  - **Non-Linear** 데이터셋에 대한 단일 **Perceptron** 의 한계를 극복하기 위해 다수의 **Perceptron** 으로 다중 **Layer** 네트워크를 구성하여 **Linear/Non-Linear** 한 상황 모두에서 **Decision Boundary** 를 생성할 수 있음.
  - 다수의 **Perceptron** 이 연결되어있는 구조이기 때문에 **Linear Regression** 과 유사한 단일 **Perceptron** 과 달리 **Error Optimization** 을 위한 **Gradient Descent** 가 매우 복잡함. 이를 극복하기 위해 **Chain-Rule** 을 기반으로 **Backpropagation** 을 사용함.

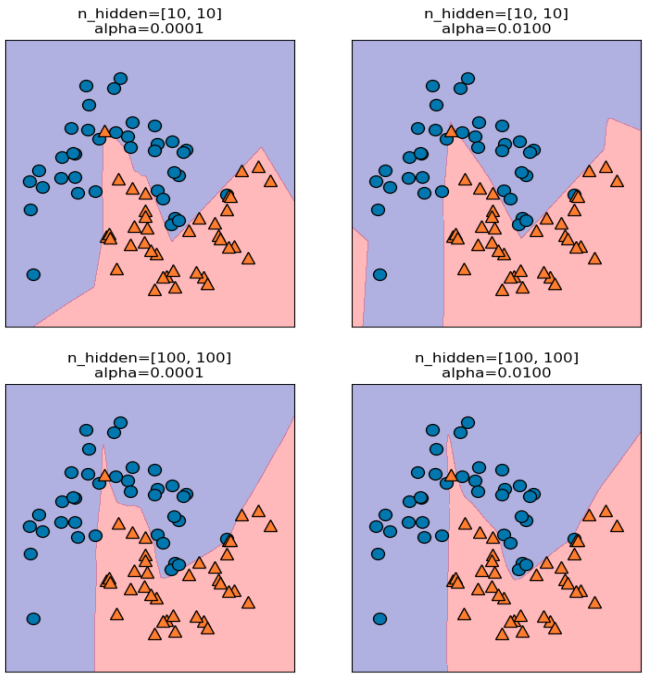


Fig 16: MLP Decision Boundary

  - 더 많은 **Perceptron** 과 **Layer** 로 **Neural Network** 를 구성할수록 **Non-Linear** 한 데이터셋에 대해 더 효과적인 **Non-Linear Decision Boundary** 를 만들 수 있음.

- **Backpropagation**

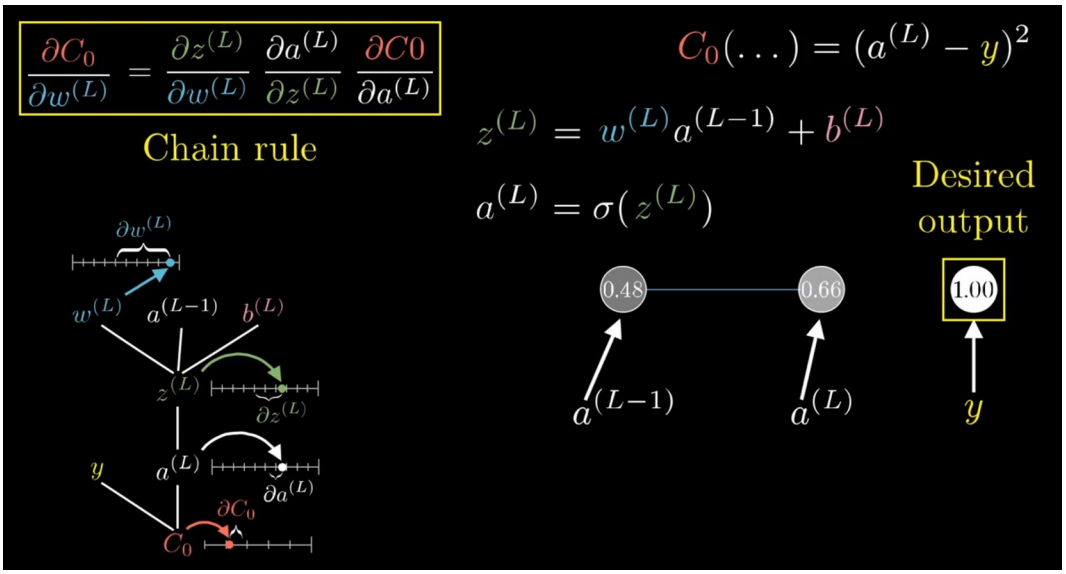


Fig 17: Backpropagation Calculus & Chain Rule
(https://www.youtube.com/watch?v=tIeHLnjs5U8)

- **Prediction Output Y'**과 **Target Output Y** 간의 **Error** 값은 **Neural Network** 최종 **Output Layer** 에서 도출할 수 있음.
- 그러나 역방향으로 **Prediction Y'**의 변화에 따른 **Error** 변화에 비례해서 이전 **Layer** 의 모든 **Weight** 값을 업데이트하기 위해 이전 **Layer** 를 매번 방문하는 것은 매우 비효율적임. 특히 **Layer** 가 늘어날 수록 **Weight** 업데이트에 소요되는 시간이 기하급수적으로 증가함.
- 이를 해결하기 위해 **Prediction Output** 대비 **Error (Loss)**의 변화량(편미분)을 **Chain Rule** 를 통해 각 **Layer** 별 **Input** 대비 **Output** 변화량 (편미분)의 조합으로 재정리하여 보다 쉽게 **Gradient Descent** 에 필요한 편미분을 모두 산출할 수 있음.
- **Prediction Output** 대비 **Error (Loss)**를 편미분하면서 이전 **Layer** 의 **Activated** 된 **Weighted Sum** 의 편미분이 밖으로 연속적으로 꺼내지게됨. 각 **Layer** 별 **Weighted Sum** 의 편미분이 **Backpropagation** 을 통해 모두 구해지기 때문에 모든 **Layer** 의 **Weight** 업데이트에 필요한 업데이트량을 알 수 있음.
- **Backpropagation** 을 통해 어떤 **Weight** 가 다음 **Layer** 의 **Output** (최종적으로 **Error/Loss Fucntion**)에 연속적으로 얼마나 영향을 주는지 알 수 있음.

- **주요 문제점**
  - **Vanishing Gradient**
    - **Deep Learning** 과 같이 매우 많은 **Layer** 로 구성된 **Neural Network** 의 경우 **Output Layer** 에서 **Input Layer** 로 갈 수록 미분하는 횟수가 증가함.
    - 미분 횟수가 너무 증가하게 되면 결국 **Input Layer** 에 가까운 **Weighted Sum** 의 **Error/Loss** 에 대한 편미분은 거의 **0** 에 수렴하게 되면서 **Weight** 업데이트량이 **0** 에 수렴하게됨. 이로 인해 **Input Layer** 쪽에 가까운 **Weight** 가 업데이트 되지 않는 **Vanishing Gradient** 현상이 발생함.
    - 이는 각 **Layer** 의 **Weighted Sum** 이 거치는 **Activation Function** 이 미분을 여러번 거치면서 점점 **0** 에 수렴하기에 발생하는 현상임.
    - 이를 해결하기 위해 미분을 해도 **0** 에 수렴되지 않는 **Non-Linear Function** 인 **ReLU, Leaky ReLU** 등을 **Activation** 함수로 사용함.

  - **매우 많은 데이터량 및 Computing 연산 능력 요구**
    - **Neural Network** 의 경우 **Layer** 가 깊어지고 **Perceptron** 을 많이 사용할수록 그에 따라 사용되는 **Weight** 의 개수, **Backpropagation** 을 위해 저장하는 데이터의 개수가 기하급수적으로 증가함.
    - 또한 **CPU** 의 경우 병렬적인 연산보다는 단일 고속 연산에 특화되어있기 때문에 **CPU** 를 이용한 **Backpropagation** 및 **Neural Network** 학습은 매우 느림.
    - 이를 해결하기 위해서 **CPU** 대신 병렬 연산에 특화된 **GPU** 를 사용해야함. 그리고 **Neural Network** 를 작동시키는 프로그램이 데이터를 시스템에 등재시키기 위해 대량의 **RAM** 와 **VRAM** 을 사용해야함.