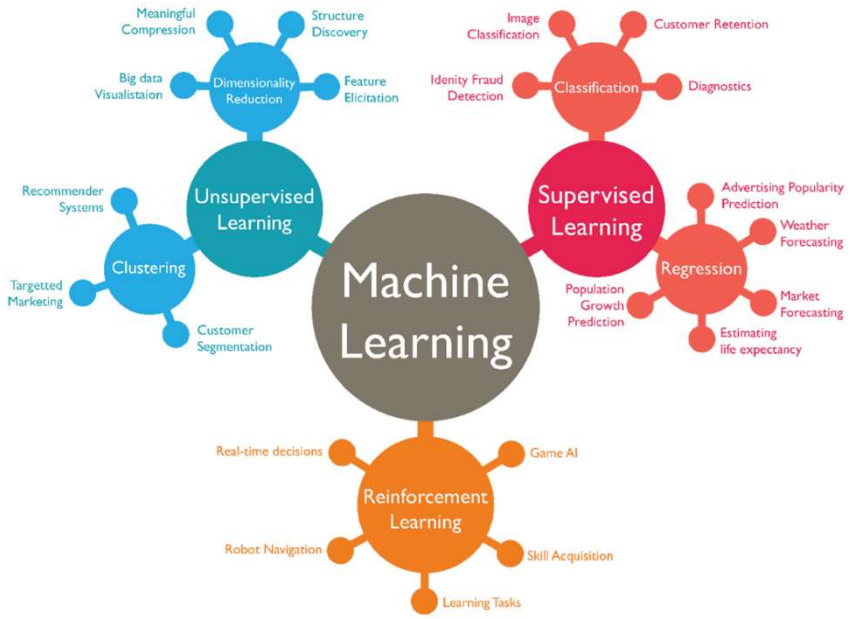
머신러닝과 퍼셉트론

실습 환경

- 복잡성을 줄이고, 개념에 대한 이해도를 높이기 위해 Pop.AI와 PyTorch 병행
  - Pop.AI는 한백전자에서 개발한 CUDA 기반 고수준 머신러닝 라이브러리로 실습장비에 내장됨
    - 모듈 로드 (CUDA 라이브러리를 로드하는데 다소 시간이 걸림)
      - from pop.AI import <클래스>
    - 지원 모델별 클래스
      - 회귀, 분류: Linear_Regression(선형 회귀), Logistic_Regression(로지스틱 회귀), Perceptron(퍼셉트론)
      - 신경망: ANN(인공 신경망), DNN(심층 신경망), CNN(합성 곱신경망), RNN(순환 신경망), DQN(강화 학습)
  - PyTorch는 텐서플로와 함께 오픈소스 기반 머신러닝 전문 프레임워크로 cpu + cuda(gpu) 버전이 실습장비에 내장됨
    - PC 설치 참조 (실습장비의 PyTorch를 업데이트하면 CPU 전용 버전이 설치되므로 주의)
      - sudo pip3 install torch torchvision
  - 학습은 모델을 생성하는 과정에서 학습 횟수 증가와 같은 매개변수 조정이 필요하므로 Jupyter 환경 권장
    - 학습된 모델을 파일로 저장하거나 저장된 모델을 불러와 다시 학습을 이어감 (전이 학습)
  - 예측은 Jupyer 환경을 사용하지 않아도 됨
    - 학습된 모델을 이용해 응용 작성

머신러닝

- 머신러닝은 컴퓨터 시스템이 주어진 데이터를 학습하는 과정으로 지도 학습, 비지도 학습, 강화 학습으로 나눠짐
  - 지도 학습(Supervised Learning)
    - 컴퓨터에게 데이터에 대한 정답(Label)이 무엇인지 알려주면서 학습
    - 정확한 입력(input)과 출력(output) 존재
  - 비지도 학습(Unsupervised Learning)
    - 정답을 알려주지 않고 비슷한 데이터를 군집화하여 미래 예측
    - 라벨링이 되어있지 않은 데이터로부터 패턴이나 형태를 탐색
    - 지도 학습에서 적절한 특징(Feature)을 찾아내기 위한 전처리 방법으로 사용
  - 강화 학습(Reinforcement Learning)
    - 학습 모델이 이전 출력과 비교해 더 나은 결과를 출력할 때 보상
    - 보상을 통해 오차를 줄여 나가는 학습 방법



출처: https://dev.to/afozbek/supervised-learning-vs-unsupervised-learning-4b65

지도 학습

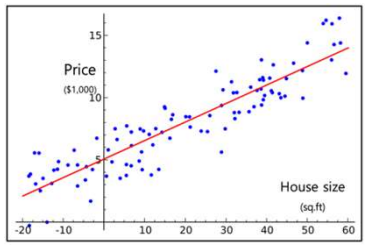
- 지도학습을 하기 위해서는 과거의 데이터 필요
  - 데이터를 독립변수(원인)와 종속변수(결과)로 분리한 후 컴퓨터를 학습시켜 모델 생성
    - 모델: 독립변수와 종속변수의 관계를 컴퓨터에게 학습시키는 과정에서 컴퓨터가 만들어낸 관계를 설명하는 공식
  - 모델을 만들면 아직 결과를 모르는 원인을 모델에 입력했을 때 결과를 순식간에 계산해 알려줌
    - 과거에는 이런 공식을 만들려면 고도의 실험과 높은 수준의 수학 지식 필요 ➡ 소수 엘리트들의 전유물
    - 머신러닝이 등장하면서 과거와는 비교도 할 수 없을 정도로 적은 지식과 노력으로 나만의 공식을 만들어서 사용할 수 있게 됨
  - 학습에 필요한 데이터를 가공하는 과정이 필요하므로 학습 데이터가 단순하고 목푯값의 종류가 적은 문제에 적합
    - 짧은 학습 시간을 투자해 낮은 오차를 얻을 수 있음
    - 복잡한 학습 모델일수록 필요한 데이터양은 급격히 증가
  - 지도 학습 기법은 분류(Classification)와 회귀(Regression)로 나눠지고, 학습 알고리즘은 벡터 기반과 인공 신경망으로 나눠짐
    - 벡터 기반: 회귀 분석과 SVM(Support Vector Machine)
    - 인공신경망 기반: CNN과 RNN

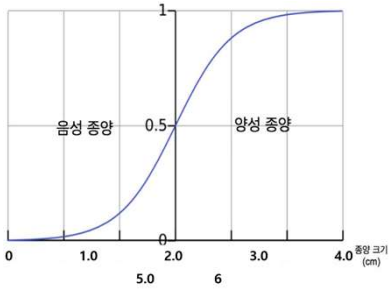
분류

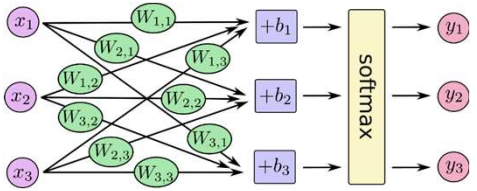
- 독립변수와 종속변수로 나눠진 학습 데이터에서 학습된 기준에 따라 새로운 독립변수를 어떤 종류(Class)로 구분할지 선택
  - 분류 모델은 값과 클래스를 학습해 새로운 값이 입력되면 어떤 클래스를 부여할지 선택
  - 분류 기법은 이진 분류(Binary Classification)와 다중 분류로 구분(Multi-class Classification)
    - 이진 분류
      - 클래스가 True와 False로 이루어져 있는 모델로 데이터를 두 가지로 분류
      - 로지스틱 회귀도 분류의 한 종류
      - 양수와 음수를 구분하는 분류 모델, 목소리를 남성과 여성을 구분하는 분류 모델 등
    - 다중 분류
      - 클래스가 여러 개로 이루어져 있는 모델로 데이터를 여러 클래스로 분류
      - 소프트맥스 회귀도 다중 분류의 한 종류
      - 체온의 정상 범위를 구분하는 분류 모델, 고양이 이미지를 품종별로 구분하는 분류 모델

회귀

- 독립변수와 종속변수로 이루어진 데이터로부터 두 값의 관계를 학습하고, 새로운 독립변수에 대한 종속변수를 예측하는 기법
  - 학습은 데이터로 원인 x(입력값)와 결과 y(출력값)가 주어질 때 두 값의 관계 함수 $y = f(x)$ 를 찾아내는 과정
    - 예측은 새로운 입력값을 관계 함수 $y = f(x)$ 에 대입해 얻은 결과값을 응용에 적용
  - 회귀 기법은 출력값의 범위와 클래스의 개수에 따라 선형(Linear), 로지스틱(Logistic), 소프트맥스(Softmax)로 구분
    - 선형 회귀
      - 입력값과 출력값을 선형적 관계로 모델링하며, 출력값의 범위는 -∞~∞
      - 함수 회귀 모델, 기후 예측 모델, 집 면적 대비 가격 예측 모델 등
    - 로지스틱 회귀
      - 입력값과 출력값을 이항 확률 관계로 모델링하며, 출력값은 0과 1 사이
      - 논리 회귀 모델, 시험 합격/불합격 예측 모델, 종양의 음성/양성 구분 모델 등
    - 소프트맥스 회귀
      - 입력값과 출력값을 다항 확률 관계로 모델링하며, 출력값은 n개로, 각각 0과 1 사이
      - 품종 예측 모델 등







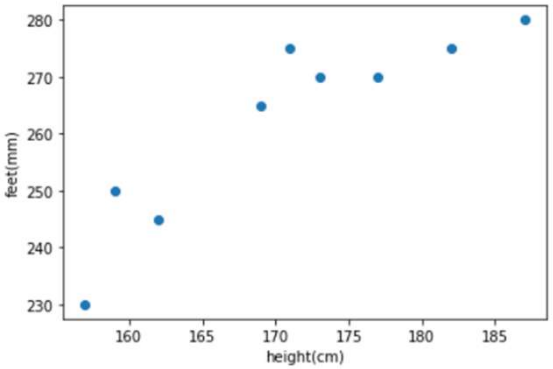
선형 회귀

- 선형 회귀는 입력값이 출력값에 대해 직접적인 관계를 갖는 선형 관계를 가짐
  - 다음과 같이 키와 발 크기에 관한 데이터가 주어질 때, 키에 따른 발 크기 예측을 회귀 모델로 구현

키(cm)	발 크기(mm)	키(cm)	발 크기(mm)	키(cm)	발 크기(mm)
173	270	187	280	177	270
171	275	157	230	159	250
162	245	169	265	182	275

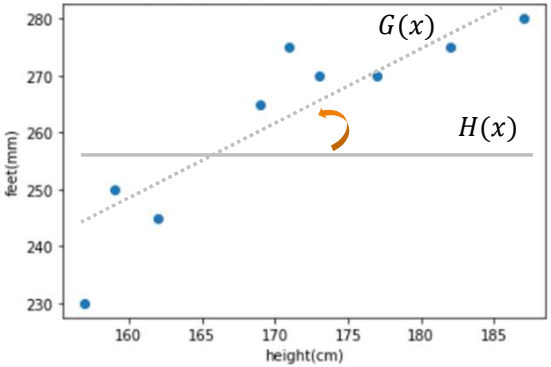
  - 주어진 데이터 시각화

```
01: import matplotlib.pyplot as plt
02:
03: x_data = [[173], [171], [162], [187], [157], [169], [177], [159], [182]]
04: y_data = [[270], [275], [245], [280], [230], [265], [270], [250], [275]]
05:
06: plt.scatter(x_data, y_data)
07: plt.xlabel("height(cm)")
08: plt.ylabel("feet(mm)")
09: plt.show()
```



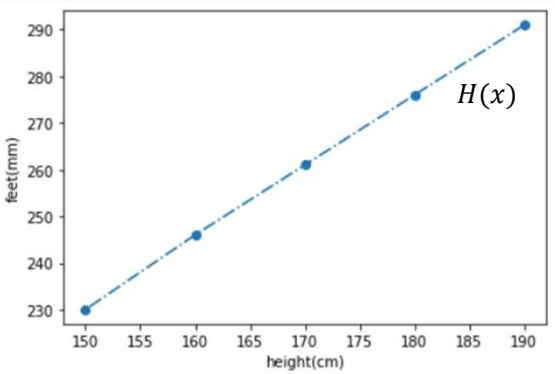
선형 회귀

- 모든 데이터의 관계를 하나의 이상적인 직선으로 표현할 수 있을 때, 이를 $G(x)$ 로 가정
  - 직선 $G(x)$를 구하기 위해 임의로 설정한 직선을 가설(Hypothesis)이라 하고 $H(x)$로 표현
  - 선형 회귀 모델은 데이터 사이 관계를 가장 잘 표현할 수 있는 가설 $H(x)$를 찾는 과정 ➡ 최적화(Optimize)



- 성공적으로 가설 최적화를 끝낸 선형 회귀 모델은 새로운 키 데이터가 주어졌을 때 발 크기가 몇일지 선형적으로 예측할 수 있음

키(cm)	발 크기(mm)	키(cm)	발 크기(mm)
190	291	160	246
180	276	150	230
170	261		



선형회귀

- Linear_Regression 클래로 선형 회귀 모델 생성
  - Linear_Regression(restore=False, ckpt_name="linear_regression_models"): 선형회귀 객체 생성
    - restore: 학습 구분. True이면 기존 모델 이용(전이학습 또는 예측), Flase이면 새 모델 생성. 기본값은 False
    - ckpt_name: 학습 모델을 저장하거나 불러올 폴더 이름. 생략하면 현재 경로의 linear_regression_models
  - Linear_Regression.X_data, Linear_ Regression.Y_data: 학습용 독립 변수와 종속 변수 설정 및 읽기
    - 2차원 리스트 타입로 독립 변수와 종속 변수는 1:1로 대응해야 함
  - Linear_Regression.train(times=100, print_every=10): 학습 진행
    - times: 학습 횟수. 기본값은 100
    - print_every: 학습 과정에서 오차 출력 스탭. 기본값은 10
  - Linear_Regression.run(input=None): 학습된 모델을 이용해 예측
    - input: 새로운 독립 변수. 생략하면 학습용 독립 변수 사용
    - 반환은 예측값(새로운 종속 변수)

```
01: from pop.AI import Linear_Regression
02:
03: x_data = [[173], [171], [162], [187], [157], [169], [177], [159], [182]]
04: y_data = [[270], [275], [245], [280], [230], [265], [270], [250], [275]]
05:
06: linear = Linear_Regression(ckpt_name="height_feet")
07: linear.X_data = x_data
08: linear.Y_data = y_data
09: linear.train(500, 50)
```

```
50 step loss: 368.93804931640625
100 step loss: 50.86083221435547
150 step loss: 49.76646423339844
200 step loss: 49.765838623046875
250 step loss: 49.76567077636719
300 step loss: 49.76565933227539
350 step loss: 49.76569747924805
400 step loss: 49.76560974121094
450 step loss: 49.76557922363281
500 step loss: 49.76546859741211
Training is done.
Time spent: 5.5 s
Training speed: 91.1 step/s
```

선형 회귀

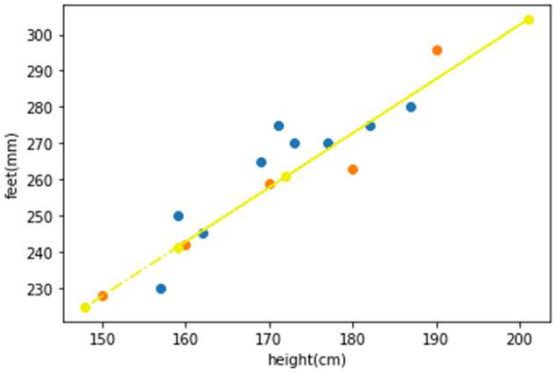
- 모델의 오차를 줄이기 위해 새로운 독립 변수와 종속 변수 사용해 생성된 모델 전이 학습

```
01: from pop.AI import Linear_Regression
02:
03: append_x_data = [[190], [180], [170], [160], [150]]
04: append_y_data = [[296], [263], [259], [242], [228]]
05:
06: linear = Linear_Regression(True, ckpt_name="height_feet")
07: linear.X_data = append_x_data
08: linear.Y_data = append_y_data
09: linear.train(500, 50)
```

```
550 step loss: 0.0878119245171547
600 step loss: 0.08761562407016754
650 step loss: 0.08760855346918106
700 step loss: 0.08760413527488708
750 step loss: 0.08760148286819458
800 step loss: 0.08759927749633789
850 step loss: 0.08759640902280807
900 step loss: 0.08759233355522156
950 step loss: 0.08758692443370819
1000 step loss: 0.08758263289928436
Training is done.
Time spent: 5.0 s
Training speed: 99.7 step/s
```

- 생성된 모델에 새로운 독립 변수를 적용해 종속 변수 예측

```
01: from pop.AI import Linear_Regression
02: import matplotlib.pylab as plt
03:
04: new_x = [[159], [172], [201], [148]]
05:
06: linear = Linear_Regression(True, ckpt_name="height_feet")
07: prediction_y = linear.run(new_x)
08:
09: plt.xlabel("height(cm)")
10: plt.ylabel("feet(mm)")
11: plt.scatter(x_data, y_data)
12: plt.scatter(append_x_data, append_y_data)
13: plt.plot(new_x, prediction_y, color='#f0f000', linestyle="-.", marker='o')
14: plt.show()
```




```
[[241.18515]
 [260.68185]
 [304.1745 ]
 [224.68794]]
```

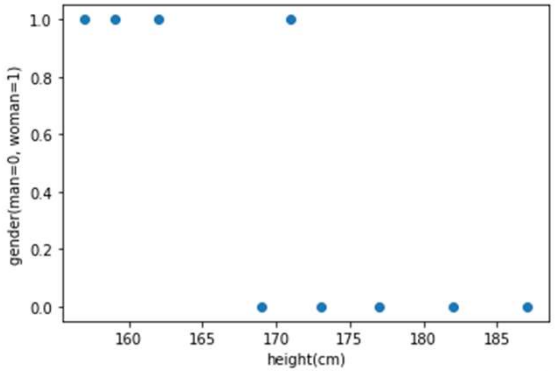
로지스틱 회귀

- 로지스틱 회귀는 입력에 대해 이항 확률 관계를 가지며, 결과를 0~1 사잇값(입력이 True(1)일 확률)으로 표현
  - 다음과 같이 키와 성별에 관한 데이터가 주어졌을 때 키에 따른 성별을 예측하기 위한 회귀 모델 구현

키(cm)	성별	키(cm)	성별	키(cm)	성별
173	0 (남성)	187	0 (남성)	177	0 (남성)
171	1 (여성)	157	1 (여성)	159	1 (여성)
162	1 (여성)	169	0 (남성)	182	0 (남성)

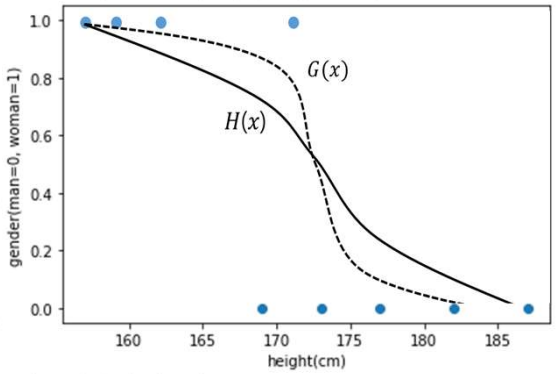
  - 주어진 데이터 시각화

```
01: import matplotlib.pylab as plt
02:
03: x_data = [[173], [171], [162], [187], [157], [169], [177], [159], [182]]
04: y_data = [[0], [1], [1], [0], [1], [0], [0], [1], [0]]
05:
06: plt.scatter(x_data, y_data)
07: plt.xlabel("height(cm)")
08: plt.ylabel("gender(man=0, woman=1)")
09: plt.show()
```



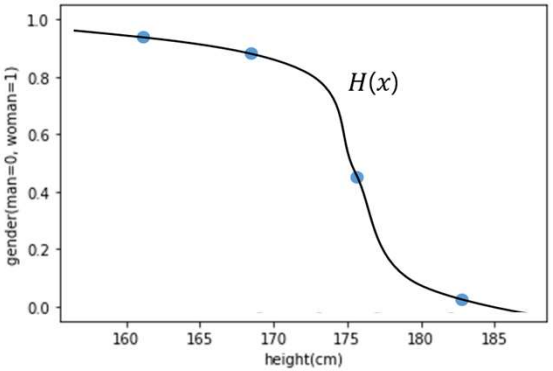
로지스틱 회귀

- 모든 데이터의 관계를 하나의 이상적인 곡선으로 표현할 수 있을 때, 이를 $G(x)$ 로 가정
  - 로지스틱 회귀 모델은 곡선 $G(x)$를 구하기 위해 임의의 곡선 가설 $H(x)$ 설정한 후 가설 $H(x)$를 $G(x)$ 에 가깝도록 최적화



- 성공적으로 가설 최적화를 끝낸 로지스틱 회귀 모델은 새로운 키가 주어지면, 여성(또는 남성)일 확률을 예측할 수 있음

키(cm)	성별	키(cm)	성별
190	0.005	160	0.916
180	0.073	150	0.972
170	0.493		



로지스틱 회귀

- Logistic_Regression 클래스로 선형 회귀 모델 생성
  - Logistic_Regression(input_size=1, restore=False, ckpt_name="logistic_regression_models"): 로지스틱 회귀 객체 생성
    - input: 입력 데이터(최하위 차원) 크기. 기본값은 1
  - Logistic_Regression.X_data, Linear_ Regression.Y_data: 학습용 독립 변수와 종속 변수 설정 및 읽기
  - Logistic_Regression.train(times=100, print_every=10): 학습 진행
  - Logistic_Regression.run(input=None): 학습된 모델을 이용해 예측

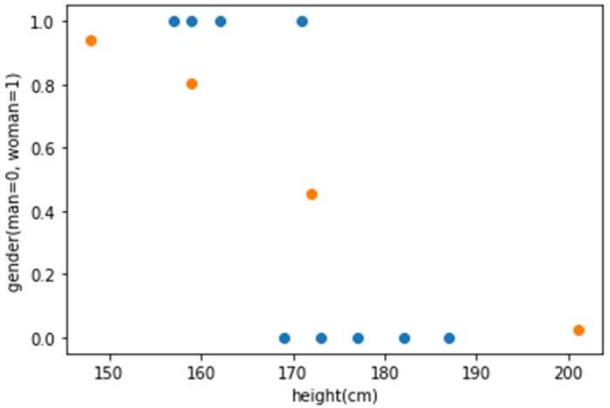
```
01: from pop.AI import Logistic_Regression
02:
03: x_data = [[173], [171], [162], [187], [157], [169], [177], [159], [182]]
04: y_data = [[0], [1], [1], [0], [1], [0], [0], [1], [0]]
05:
06: logistic = Logistic_Regression(ckpt_name="height_gender")
07: logistic.X_data = x_data
08: logistic.Y_data = y_data
10: logistic.train(1500, 300)
```

```
300 step loss: 0.5710358023643494
600 step loss: 0.5021206736564636
900 step loss: 0.460025429725647
1200 step loss: 0.4189497232437134
1500 step loss: 0.38830915093421936
Training is done.
Time spent: 16.4 s
Training speed: 91.3 step/s
```

로지스틱 회귀

- 생성된 모델에 새로운 독립 변수를 적용해 종속 변수 예측

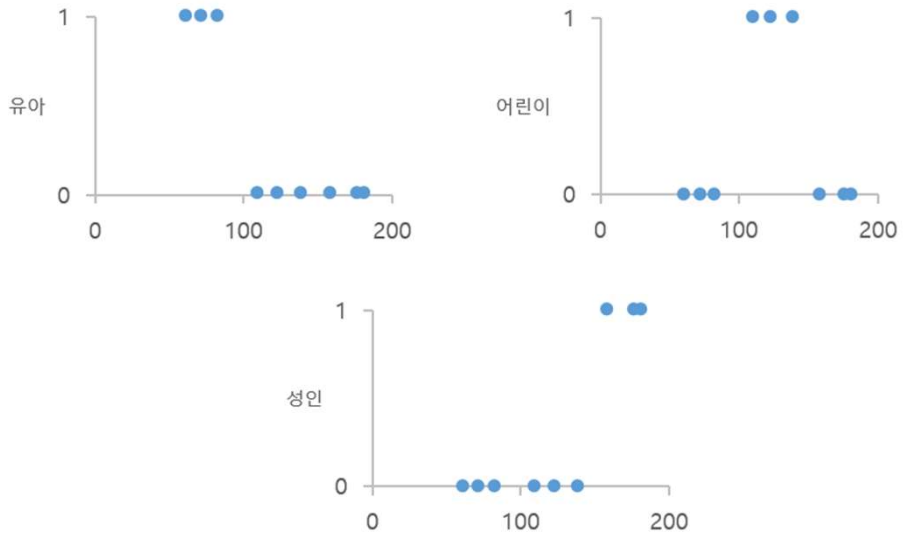
```
01: from pop.AI import Linear_Regression
02: import matplotlib.pylab as plt
03:
04: new_x = [[159], [172], [201], [148]]
05:
06: logistic = Logistic_Regression(restore=True, ckpt_name="height_gender")
07: prediction_y = logistic.run(new_x)
08:
09: plt.xlabel("height(cm)")
10: plt.ylabel("gender(man=0, woman=1)")
11: plt.scatter(x_data, y_data)
12: plt.scatter(new_x, prediction_y)
13: plt.show()
```



소프트맥스 회귀

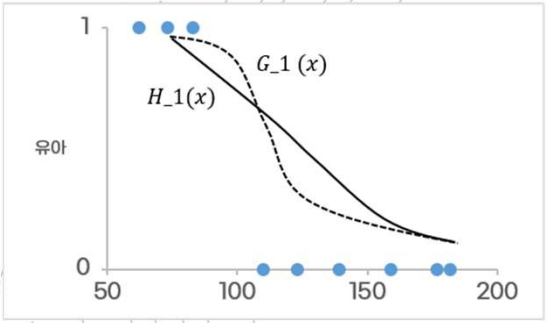
- 소프트맥스 회귀는 입력에 대해 다항 확률 관계를 가짐
  - 입력이 주어졌을 때 각 클래스별로 0~1사잇값으로 표현되며, 모든 클래스 값의 합은 1이 됨
  - 키와 연령층에 관한 데이터가 주어졌을 때 키에 따른 연령층을 예측하기 위한 회귀 모델을 구하고자 함

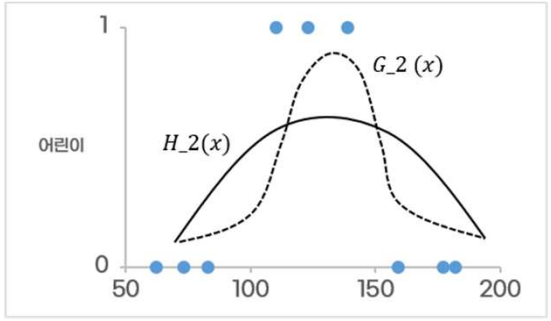
키(cm)	유아	어린이	성인
73	1	0	0
62	1	0	0
83	1	0	0
110	0	1	0
139	0	1	0
123	0	1	0
177	0	0	1
159	0	0	1
182	0	0	1

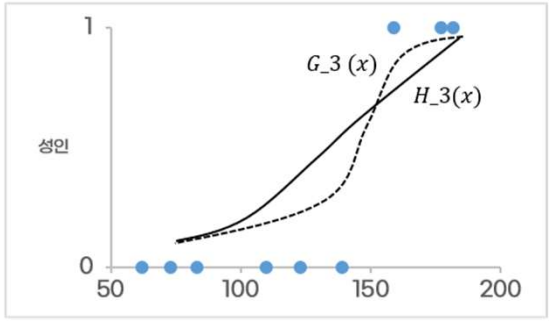


소프트맥스 회귀

- 모든 데이터의 관계를 각 그래프에서 이상적인 곡선으로 표현할 수 있을 때, 이를 $G_i(x)$ 로 가정
  - 소프트맥스 회귀 모델은 곡선 $G_i(x)$를 구하기 위해 임의의 곡선 가설 $H_i(x)$ 설정한 후 가설 $H_i(x)$를 $G_i(x)$ 에 가깝도록 최적화

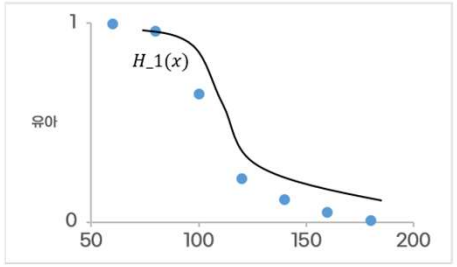


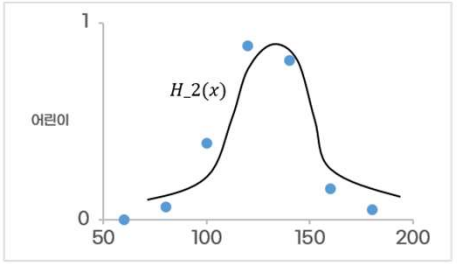


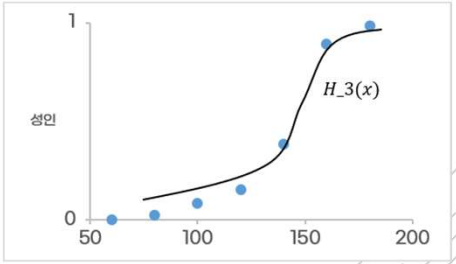


- 최적화에 성공한 소프트맥스 회귀 모델은 새로운 데이터가 주어졌을 때 연령을 예측할 수 있음
  - 모든 클래스 값의 합은 1이어야 하므로 소프트맥스 함수(Softmax Function)를 이용해 각 클래스 값을 조정

키(cm)	유아	어린이	성인
60	0.998	0.002	0
80	0.961	0.068	0.025
100	0.647	0.391	0.083
120	0.217	0.884	0.152
140	0.113	0.807	0.384
160	0.051	0.16	0.891
180	0.008	0.052	0.985

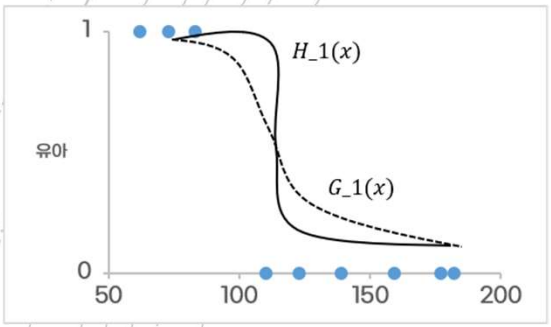


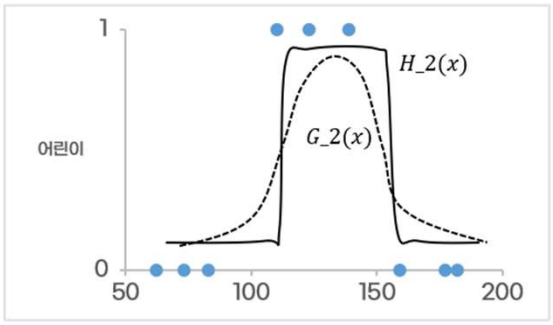


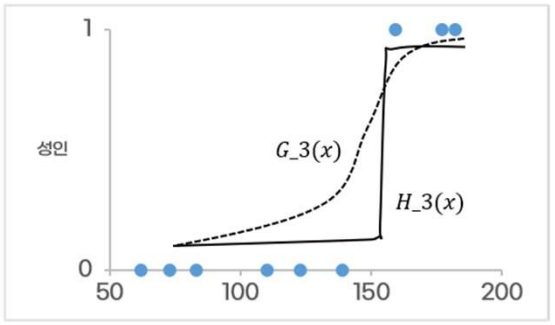


소프트맥스 회귀

- 학습 과정에서 과적합(Overfitting)과 과소적합(Underfitting)이 발생하지 않도록 주의
  - 과적합: 과도한 최적화로 이상적인 최적화 모델 $G_i(x)$를 넘어 극단적인 형태의 모델 $H_i(x)$ 생성
  - 과소적합: 부족한 최적화로 인해 원하는 결과를 얻을 수 없는 상태

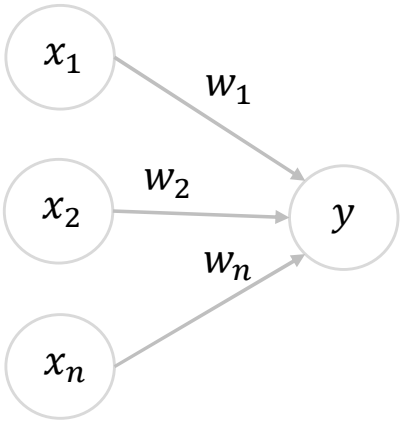






퍼셉트론

- 퍼셉트론은 입력 데이터를 2개의 부류 중 하나로 분류하는 선형 분류기(classifier)
  - 사람의 뇌신경 세포의 동작과정을 흉내 내어 만든 수학적 모델
    - 노드, 가중치, 층(layer)과 같은 개념이 도입되어 딥러닝을 포함하여 현대 신경망의 중요한 구성요소들을 이해하는데 의미가 있음
  - 입력층과 출력층이라는 2개의 층으로 구성된 단순한 구조
    - 입력층: $x_1, x_2, \ldots, x_n$
    - 출력층: y
    - 가중치: $w_1, w_1, \ldots, w_n$
    - 노드: 원
  - n개의 입력에 대한 퍼셉트론은 다음 식으로 표현
    - n개의 입력과 n개의 가중치에 대해 각각 입력과 가중치를 곱한 뒤, 이 값들을 모두 합해 기준과 비교
      - 기준보다 크면 활성화(1)되고, 작으면 비활성화(0)



$$y = \tau(w_1 x_1 + w_2 x_2 + \cdots + w_d x_d)$$

$$\tau(a) = \begin{cases} 0, & \text{if } a <= \beta \\ 1, & \text{if } a > \beta \end{cases}$$

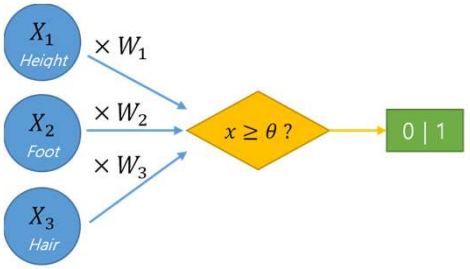
퍼셉트론

- 키, 발 크기, 머리카락 길이, 성별에 관한 데이터가 주어졌을 때 복합적 요인에 따른 성별을 예측하는 퍼셉트론 모델 구현
  - 성별 값은 0~1 사이로 출력되며 그 합이 1이 되도록 소프트맥스 사용

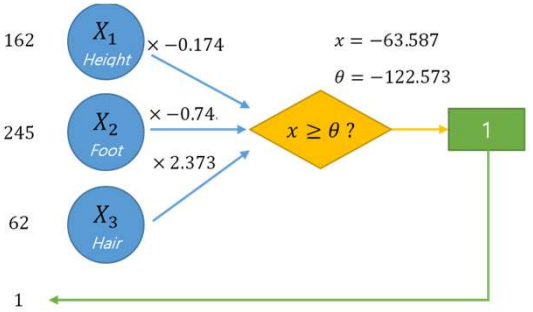
키(cm)	발 크기(mm)	머리카락 길이(cm)	성별
173	270	17	0 (남성)
171	275	51	1 (여성)
162	245	62	1 (여성)
187	280	12	0 (남성)
157	230	47	1 (여성)
169	265	30	0 (남성)
177	270	5	0 (남성)
159	250	32	1 (여성)
182	275	0	0 (남성)

퍼셉트론

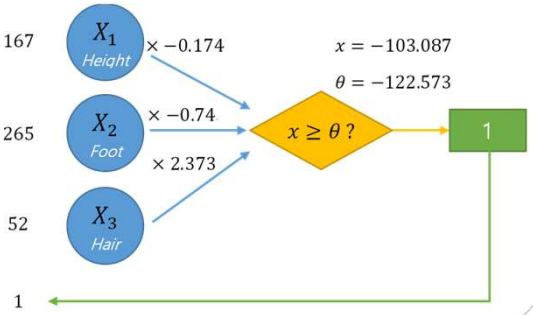
- 각 데이터에 임의의 가중치 W_i를 곱한 다음 모든 데이터를 합한 값 x과 기준값 θ 비교
  - 가중치는 각 데이터 X_i가 결과에 가지는 영향력을 의미



- 반복적으로 출력값과 실제값의 비교를 통해 가중치를 조절하면서 오차를 줄여나감 (최적화)
  - 손실 함수(Loss Function) 또는 비용 함수(Cost Function): 모델의 출력값과 실제값을 비교하는 함수
    - 손실 함수: 하나의 데이터셋에서 비교할 때
    - 비용 함수: 여러 데이터셋을 기준으로 통계를 내거나 성능을 평가할 때



- 최적화에 성공한 퍼셉트론 모델은 새로운 복합 데이터가 주어졌을 때 여성인지 판단
  - 각 데이터에 대한 가중치를 조사해 성별과 관련 있는 데이터를 분석할 수 있음



퍼셉트론

- Perceptron 클래로 퍼셉트론 모델 생성 및 예측
  - Perceptron(input_size, output_size=1, restore=False, ckpt_name="perceptron_models", softmax=True): 퍼셉트론 객체 생성
    - input_size: 입력 데이터(최하위 차원) 크기
    - output_size: 출력 데이터(최하위 차원) 크기
    - softmax: 출력 데이터 크기가 2 이상일 때, True이면 모델의 예측 결과에 소프트맥스 함수를 적용하여 총합이 1이 되도록 조정. False이면 적용 안함

```
01: from pop.AI import Perceptron
02:
03: x_data = [[173,270,17], [171,275,51], [162,245,62], [187,280,12], [157,230,47],
04:                [169,265,30], [177,270,5], [159,250,32], [182,275,0]]
05: y_data = [[0],[1],[1],[0],[1],[0],[0],[1],[0]]
06:
07: perceptron = Perceptron(input_size=3, ckpt_name="complex_gender")
08: perceptron.X_data = x_data
09: perceptron.Y_data = y_data
10: perceptron.train(1000, 100)
11:
12: new_x = [[152,230,28], [152,230,30]]
13:
14: prediction_y = perceptron.run(new_x)
15: print(prediction_y)
```

```
100 step loss: 0.08166109770536423
200 step loss: 0.06007532402873039
300 step loss: 0.05723150074481964
400 step loss: 0.054489120841026306
500 step loss: 0.05184812471270561
600 step loss: 0.049308039247989655
700 step loss: 0.046868398785591125
800 step loss: 0.04452835023403168
900 step loss: 0.04228673875331879
1000 step loss: 0.04014182463288307
Training is done.
Time spent: 20.5 s
Training speed: 48.8 step/s


[[0.42623347]
 [0.8042018 ]]
```