지능형 컴퓨팅과정 포트폴리오 경진대회

과목명 : 인공지능응용프로그래밍
교수 : 강환수교수님
학과 : 컴퓨터정보공학
학번 : 20190708
이름 : 백종수

목차

머리말

본론으로 들어가기 전, 필자는 텐서플로우 머신러닝을 기술하고자 합니다. 이미 강환수 교수님께서 이 부분을 다루셨지만, 더 깊이 있고 자세하게 알아봐 튼튼한 기반을 만들려는 목적이 있습니다. 먼저 머신러닝에 대해 알아보고, 왜 머신러닝을 사용하고자 하는지 알아볼 생각입니다. 그리고 기존 프로그래밍 방식과 머신러닝의 차이점을 파악하고, 분류할 것입니다. 뭐든지 좋은 점만 있는 것이 아니기 때문에 문제점도 알아 보고자 합니다. 이렇게 이론적 기반이 다져졌을 때, 회귀 예제 코드를 보고 시험함으로써 이 포트폴리오는 마치고자 합니다.

Machine Learning(머신러닝) 이란?

해야 할 일(문제) T에 대해서, 그동안 T를 해왔던 경험 E를 바탕으로, 학습하는 System Program을 의미합니다. 해당 Program은 Performance(성능) P를 통해 평가합니다.

쉽게 말해서, 해결하고자 하는 문제 T와 T와 관련되어 축적된 데이터 E를 사용하여 이후의 T를 해결하고자 하는 것을 의미합니다. 이는 특정한 기준으로 설정된 성능 P를 통해 평가할 수 있어야 합니다

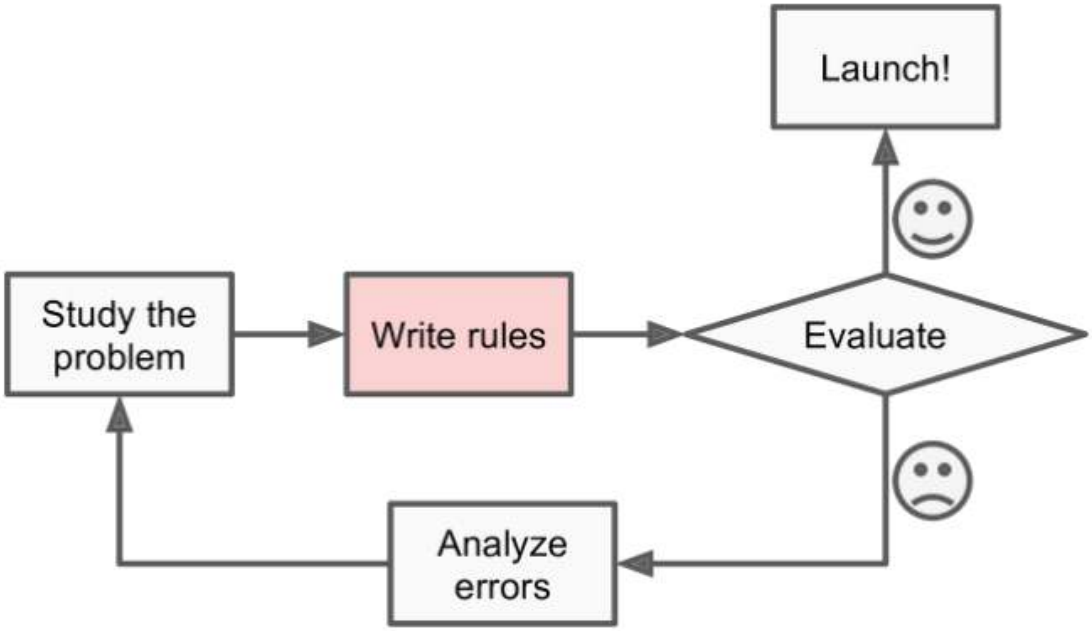
왜 머신러닝을 사용하고자 하는가?

기존의 프로그래밍 방식은 어떠한 규칙을 Case로 나누어서 대응하는 코드를 하나하나 입력하는 방식이었습니다. 새로운 Case가 발생하면, 직접 Case에 대응하는 코드를 추가로 작성해야 했죠.

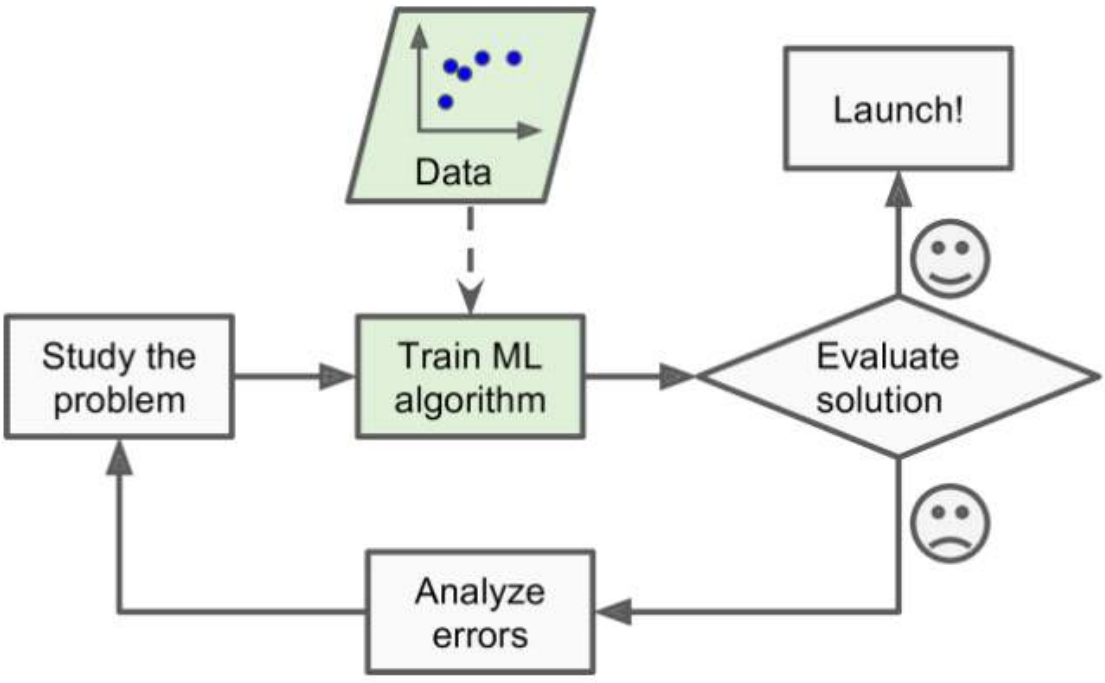
머신러닝 접근법은 기존의 Data (E)를 학습하면서, 각각의 Case (T) 들에 대한 패턴과 규칙들을 찾아내는 방법입니다. 이는 새로운 Case가 발생하면, 학습된 패턴과 규칙들을 바탕으로 따로 코드를 작성하지 않아도 대응할 수 있는 거죠.

기본 방식 vs 머신러닝

그렇다면 기존의 방식과 머신러닝 방식이 어떻게 다른지 알아보겠습니다. 아래 사진은 기존 방식의 흐름도와 머신러닝의 흐름도입니다.



Traditional Approach

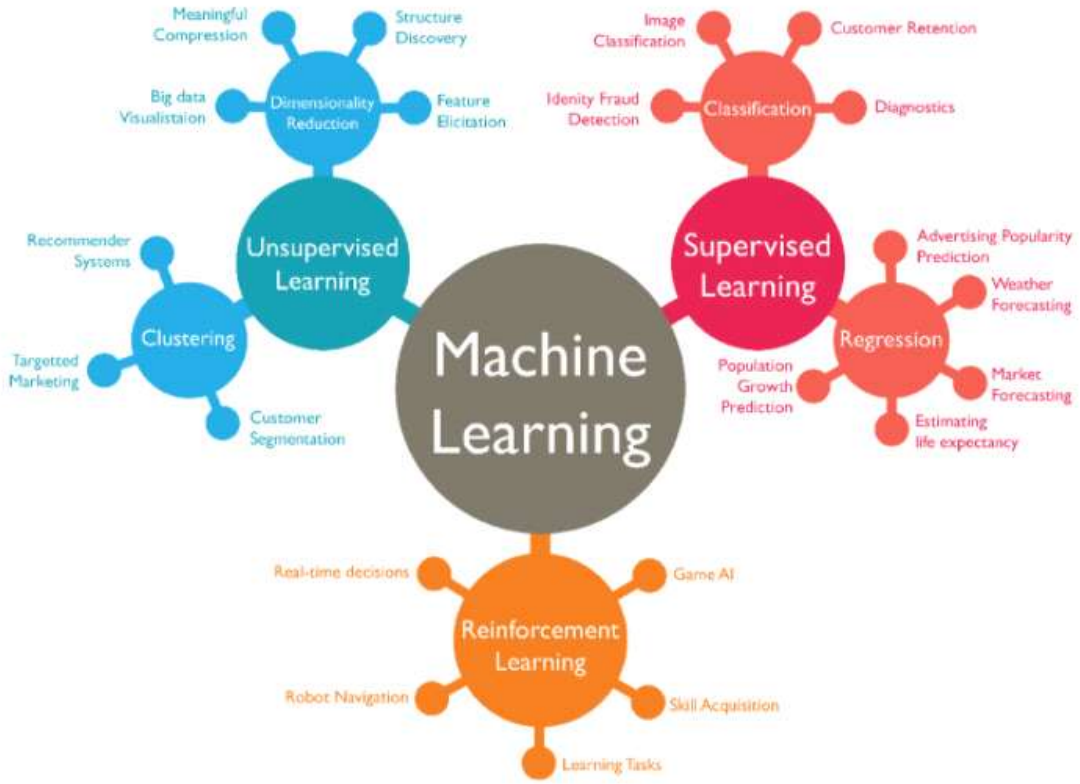


Machine Learning Approach

- 머신러닝은 기존의 프로그래밍에 비해 새로운 규칙 또는 패턴에 대한 유지보수가 쉽습니다.
- 머신러닝은 새로운 변수에 대처가 가능합니다.
- 머신러닝은 상대적으로 더 짧은 Code로 작성 가능합니다.
- 머신러닝은 복잡한 문제에 적용하기에 적합하다고 볼 수 있습니다.
- 평가를 통해 꾸준하게 개선이 될 수 있습니다.

머신러닝 시스템의 분류

머신러닝은 여러 기준에 따라 다양한 분류체계를 가지고 있습니다. 개발자는 머신러닝 시스템 분류를 이해하고, 적절한 방식의 알고리즘을 선정하여 사용하여야 합니다.



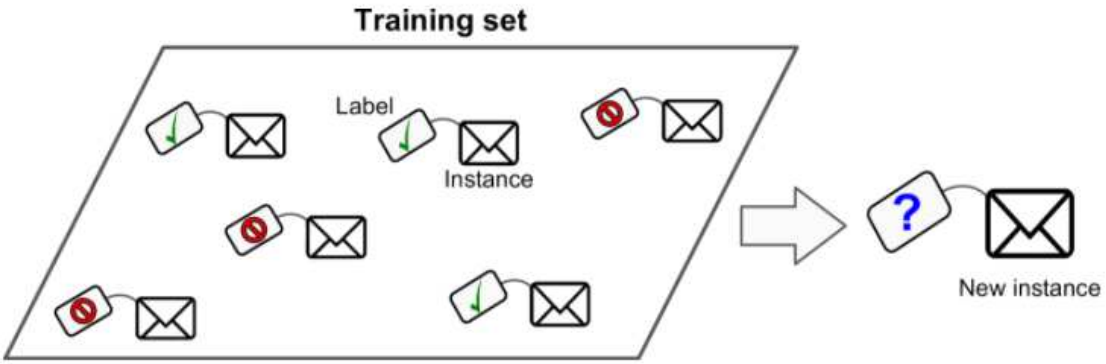
머신러닝분류

■ Human Supervision(사람이 부여한 정답)이 필요한가?

● Supervised Learning : Human Supervision이 있는 경우

Supervised Learning은 각각의 학습을 위한 데이터들이 Label을 가지고 있는 경우 사용됩니다.

Label이란, 문제와 관련된 여러 Instance들이 있을 때, 해당 data가 의미 하는 정답이라고 볼 수 있습니다.



A labeled training set for supervised learning (e.g., spam classification)

위에 그림은 스팸 메일 분류를 위한 방식을 그림으로 표현한 것입니다. 메인안의 내용들(Instance)이 있고, 각각의 메일들에는 기존에 스팸을 구분해 놓은 Label이 있습니다.

여기서 Label은 스팸인지 아닌지 두 가지 경우를 나누어서 나타냈습니다. 기존으 모든 메일 데이터를 Training Set 학습 데이터 세트로 학습시켜서 새로운 메일이 들어왔을 때, Label이 무엇인지 예측하는 방법입니다.

• Classification 분류

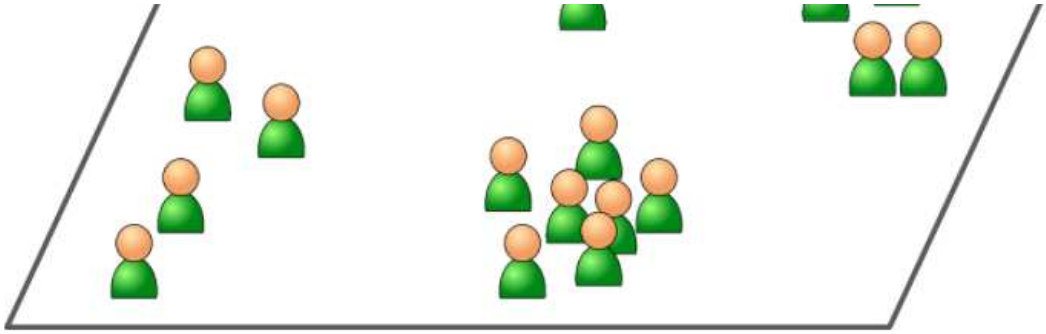
기존 Dataset의 Label들이 class로 나누어진 경우, 새로운 데이터는 어떤 class에 속하는 지를 찾아내는 방식입니다.

• Regression : 회귀

Label들이 수치로 이우어진 경우, 새로운 데이터는 어떠한 수치를 가지게 될지 예측하는 방식을 Regression이라고 합니다.

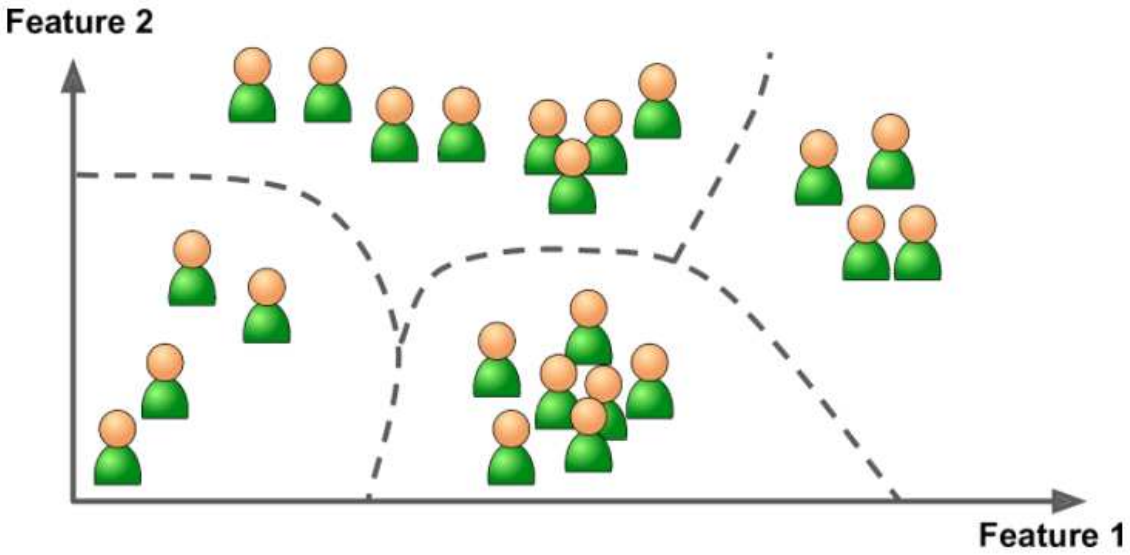
● Unsupervised Learning : Human Supervision이 없는 경우

Supervised와 다른 점은 Label이 없는 학습 데이터를 사용하여 기존의 데이터가 어떠한 형태를 지니고 있는지를 보고 판단하는 방식입니다.



An unlabeled training set for unsupervised learning

기존 데이터의 다양한 Feature들을 다양한 차원으로 분석하여 집단을 형성하는 방식입니다. Clustering 방식이 대표적입니다. 위와 같은 그림의 학습 데이터는 아래의 Clustering 그림으로 나누어집니다.



알고리즘 분류

▲ Clustering Algorithm
- ▲ K - Means
- ▲ Hierarchical Cluster Analysis (HCA)
- ▲ Expectation Maximization

▲ Visualization and dimensionality reduction
- ▲ Principal Component Analysis (PCA)
- ▲ Kernel PCA
- ▲ Locally - Linear Embedding (LLE)

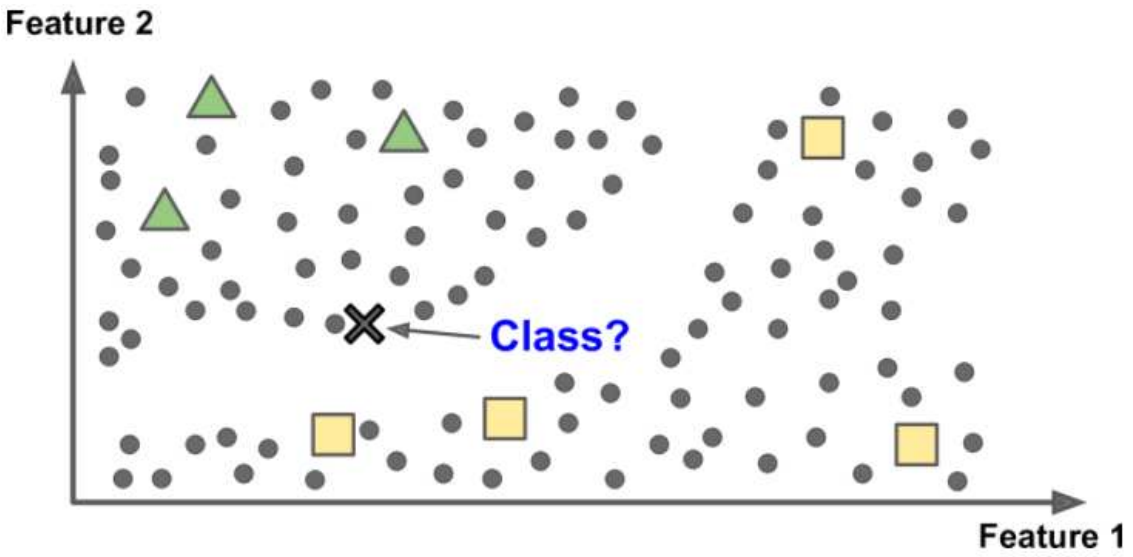
- t – distributed Stochastic Neighbor Embedding (t – SNE)

▲ Association rule learning

- Apriori
- Eclat

● Semi – supervised learning

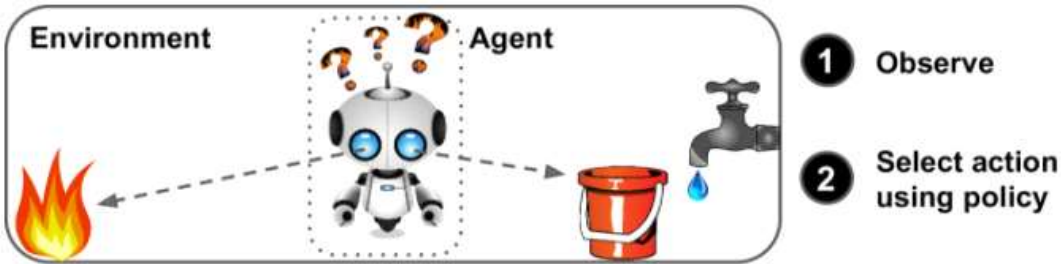
이름 그대로 supervised learning과 unsupervised learning을 합친 방식입니다. 몇몇의 Label이 있는 데이터와 많은 양의 label이 없는 데이터를 사용하여 학습하는 방식입니다.

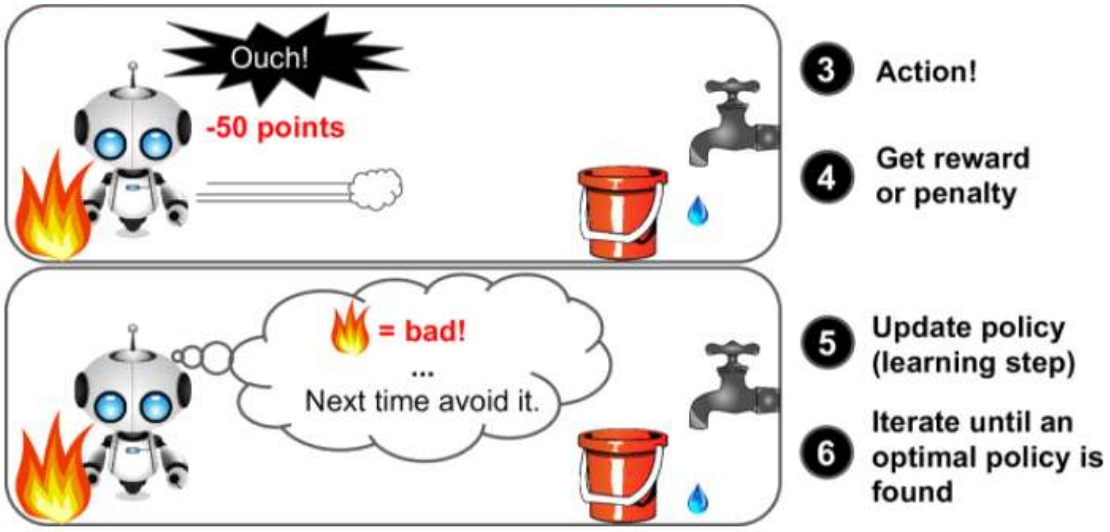


Semisupervised learning

● Reinforcement Learning

Agent가 환경을 인지하고, 정책(Policy)에 맞게 현상황에서의 행동을 선택합니다. 선택한 행동을 수행하고, 보상(Reward) 또는 페널티(Penalty)를 부여하여, 정책을 결과에 맞게 업데이트합니다. 앞의 행동을 계속 반복하여 최적의 정책을 위해 계속 업데이트해나가는 과정을 강화 학습 (Reinforcement Learning)이라고 합니다.



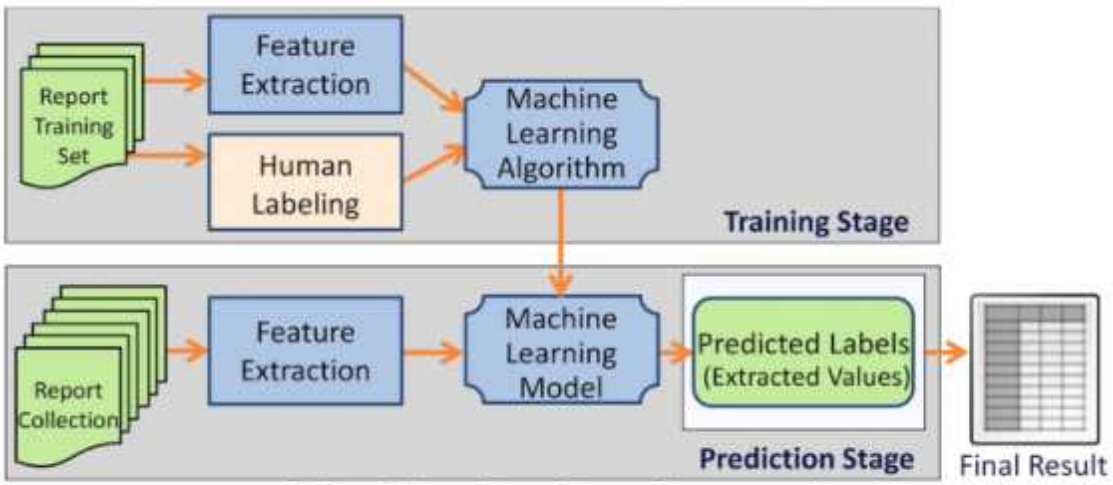


Reinforcement Learning

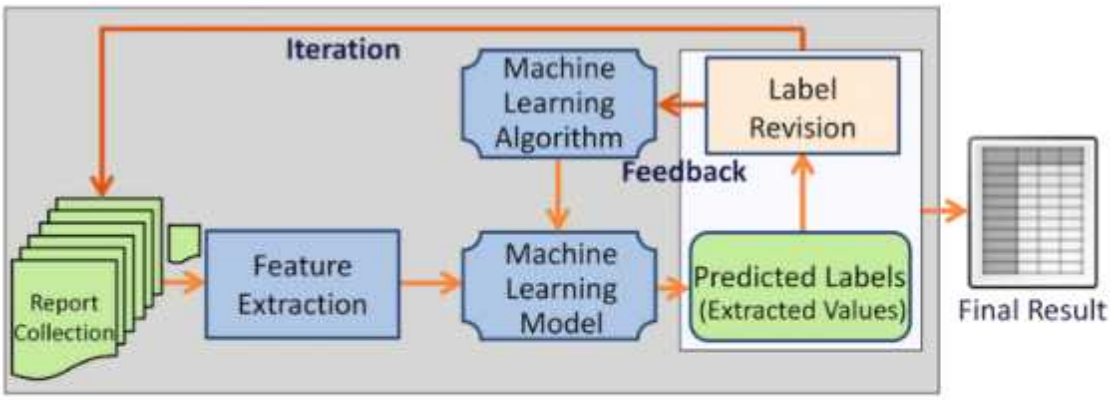
Agent와 Policy를 선정하는 것이 핵심이며, Agent는 환경 인지가 가능하고, 행동을 선택 및 수행하며, 보상 또는 손해라는 결과를 얻어서 이를 통해 학습하여 최적의 결과를 찾아가는 방식으로 업데이트합니다.

대표적인 예로는 DeepMind의 AlphaGo가 있습니다.

■ 학습은 언제 하는가? – 점진적인 학습을 하는가에 대한 여부를 분류



(a) Traditional Batch Machine Learning



(b) Online Machine Learning

Online machine learning versus batch learning Workflow

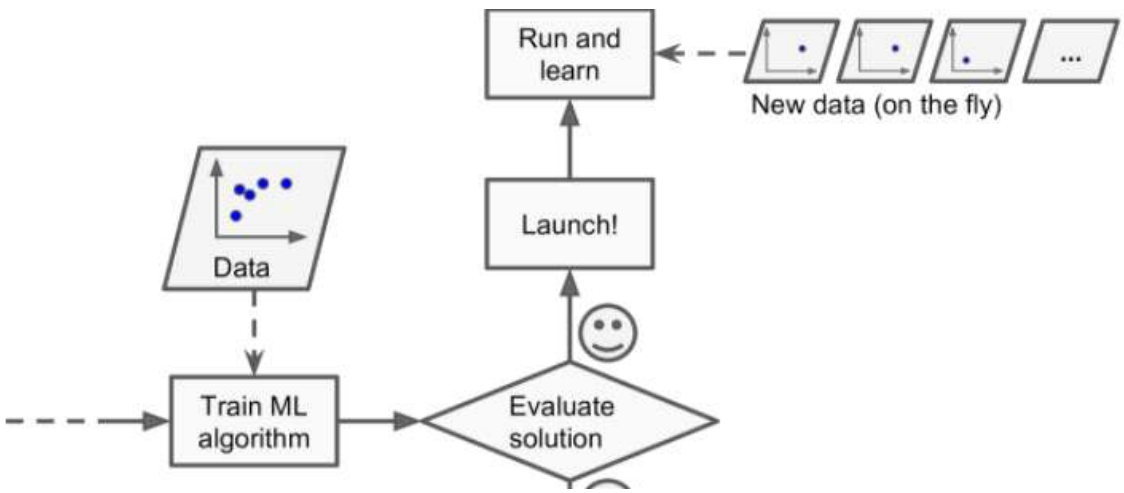
● Batch Learning

점진적으로 학습을 하지 않고, 특정 시점에 사용 가능한 모든 데이터를 사용하여 학습을 하는 방식입니다. 모여있는 모든 학습 데이터를 학습하므로, 많은 시간과 컴퓨팅 파워가 필요합니다. 일반적으로 Offline Learning이라고도 부릅니다.

새로운 데이터로의 학습을 원하면 전체 데이터 세트에 대하여 새로운 모델을 생성하여야 합니다. 그 후 시스템의 모델을 교체하는 방식입니다.

● Online Learning

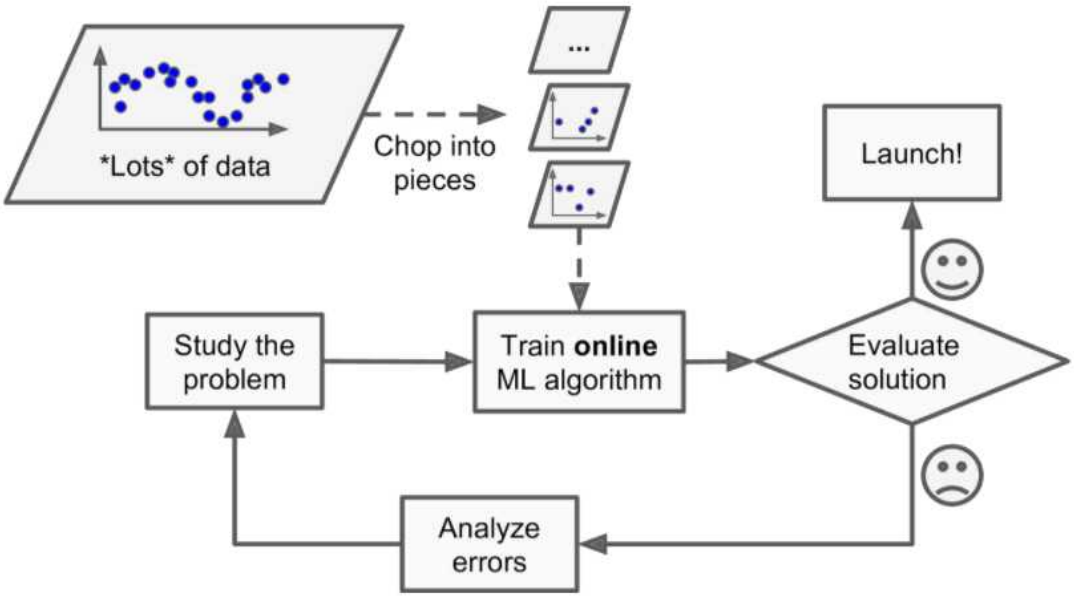
새로운 데이터 Instance들을 순차적으로 개별적 또는 소규모의 그룹으로 점진적 학습을 하는 방식입니다. Batch Learning에 비해 소규모로 학습을 진행하여서, Mini – Batch 라고도 합니다. 각 학습 단계는 빠르고 상대적으로 적은 컴퓨팅 파워를 필요로 합니다. 새로운 데이터가 들어오면 바로 학습을 시작합니다.



Online learning

빠르고 자율적으로 변화해야 하는 시스템에 적합한 훈련 방식으로, 컴퓨팅 리소스가 제한적일 때에도 적합합니다. 학습되지 않은 데이터를 저장할 공간을 절약할 수 있습니다.

Online Learning은 방대한 양의 대이터 세트에서도 사용될 수 있습니다. (모든 데이터 세트가 메인 메모리에 다 들어가지 않는 경우, out - of - core Learning이라고도 부릅니다.) 전체 데이터 세트를 가능한 만큼 메인 메모리로 나누어서 점진적으로 학습하는 방식입니다. 이때, 데이터를 나누어서 학습을 진행하는 것은 Offline에서 진행됩니다. 여기서 말하는 Offline은 Live의 반대로 생각하면 되며, 거대한 데이터 자체가 새로운 데이터이기 때문에 전체적으로 보았을 때는 online적인 Learning입니다.



Using online learning to handle huge datasets

Online Learning에서 중요한 변수 중 하나는 Learning Rate입니다. Learning Rate는 High Rate와 Low Rate로 나뉩니다.

● High Rate

시스템이 새로운 데이터에 빠르게 적응하지만, 이전 데이터를 빠르게 잊어버리는 경향이 생깁니다. 노이즈 데이터나 잘못된 데이터에 반응하여 잘못된 방향으로의 학습 확률이 높아집니다.

● Low Rate

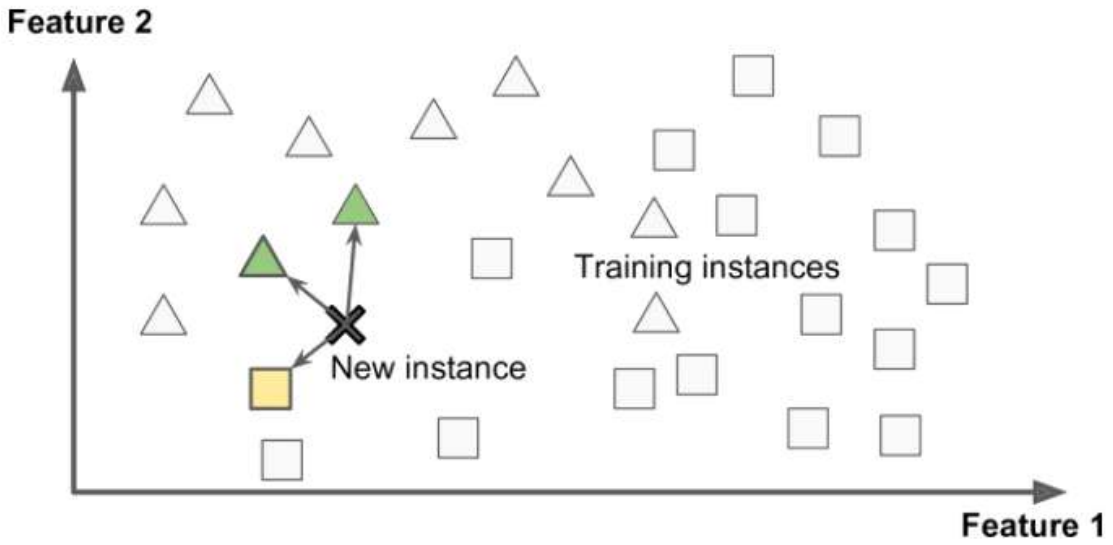
이전 데이터에 대해 관성(유지하는 성질)이 높아진다. 속도가 느려지지만, 새로운 데이터의 노이즈나 이상점에 대해 덜 민감하여 기존의 학습을 상대적으로 유지합니다.

■ Instance VS Model : 어떠한 것에 중점을 둘 것인가?

해당 기준은 시스템을 일반화하는 방법에 따라 학습을 분류합니다. 이는 새로운 데이터에 얼마나 잘 적용이 되었는가를 중요하게 생각합니다.

● Instance – based Learning

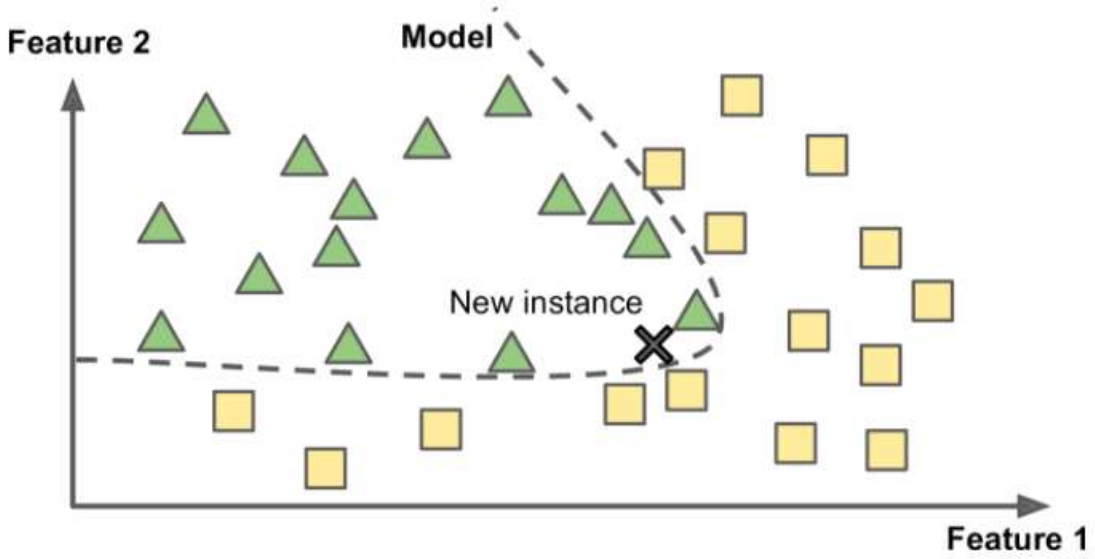
새로운 데이터를 이미 알고 있는 데이터들과의 유사성을 측정하여 판단하는 학습 방식입니다.



Instance-based Learning

● Model – based Learning

기존의 이미 알고 있는 데이터들을 학습하여 모델을 빌드한 후에, 해당 모델을 이용하여 새로운 데이터를 판단하는 학습 방식입니다.



Model-based Learning

요약하자면, 기존의 학습 데이터를 살펴보고, 적합한 모델을 선택하여 학습시키고, 모델을 적용하여 새로운 데이터를 예측하는 방식입니다. 이것은 일반적인 머신러닝 학습 방식입니다.

머신러닝이 해결해야 할 문제점

머신러닝에도 문제점이 있습니다. 이것은 물론 결과 (성능)과도 연관성이 아주 많습니다. 잘못된 예측을 하면 그것은 성능이 낮은 모델이며, 무언가 문제점을 가지고 있다는 점을 의미합니다. 머신러닝의 핵심은 데이터와 알고리즘입니다. 두 가지 관점에서 문제점이 주로 발생하며, 이를 살펴보겠습니다.

● 부족한 학습 데이터 양(Insufficient Quality of Training Data)

많은 개발자, 기업들이 문제를 머신러닝을 통해 풀고자 하지만, 큰 어려움을 겪습니다. 그것은 바로 학습할 데이터가 부족하다는 점입니다. 사실 데이터의 양은 점차 늘어가면서 아주 방대한 양의 데이터가 우리 주변에 있습니다. 하지만 문제를 정의하고 해당 문제를 해결하는 데에는 뚜렷한 목적을 가지고 정제된 데이터가 필요합니다. 이러한 데이터 세트를 형성하는 것이 가장 큰 어려움입니다.

머신러닝 알고리즘은 제대로 동작하기 위해서, 많은 데이터가 필요합니다. 간단한 문제를 해결하기 위해서는 수천 개의 데이터가 기본적으로 필요하며, 더욱이 풀고자 하는 복잡한 문제들은

수백에서 수천만, 필요하다면 더 많은 데이터가 필요합니다.

● 좋지 않은 질의 데이터 (Poor – Quality Data)

Poor - Quality Data

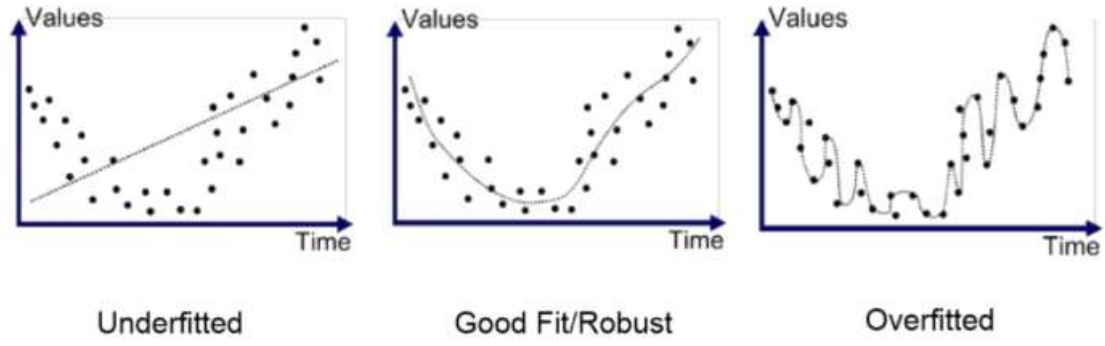
완벽한 데이터 세트는 찾기 힘듭니다. 요구하는 사항을 만족하는 데이터가 있는 반면, errors, outliers, noises가 존재하는 데이터가 있습니다. 이러한 에러와 아웃라이어, 노이즈를 적절하게 대처하는 것은 학습에 중요한 영향을 끼칩니다.

● 연관성이 적은 특징의 사용 (Irrelevant Features)

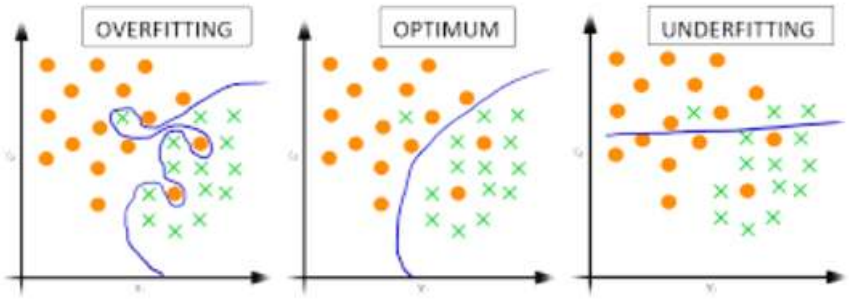
Supervised Learning의 경우, Label이라는 정답지를 가지고 있습니다. 하지만 이 Label과 연관된 특징은 모든 특징 중의 일부일 가능성이 높습니다. 오히려 관련 없는 특징들은 Label을 예측하는 데에 방해만 될 뿐입니다. 특징들을 걸러내지 않고 학습을 한다면 성능이 낮아집니다. 이를 방지하기 위하여 아래와 같은 방식을 사용합니다.

- Feature Selection
- Feature Extraction
- 연관된 새로운 Feature를 생산하여 데이터에 추가하는 방식

● 과적합 (Overfitting)

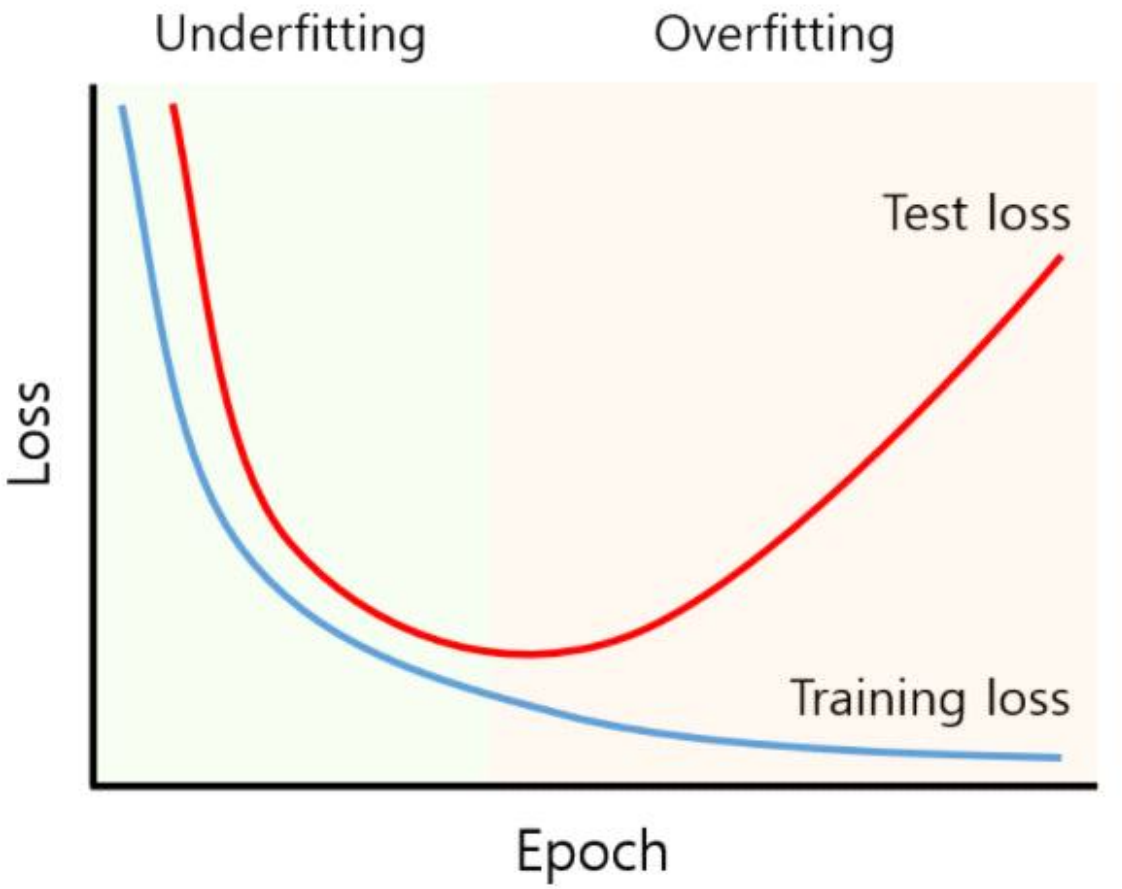


Overfitting in Regression



Overfitting in Classification

인간은 성급한 일반화를 자주 합니다. 머신러닝에서의 모델도 이러한 현상이 발생하는데, 이것을 과적합이라고 합니다. 더 자세하게 말하자면, 학습 데이터를 너무 완벽하게 학습하여 해당 데이터는 반드시 이러한 Label을 가진다고 확신하게 되는 것입니다. 그래서 새로운 데이터를 제대로 판단하지 못하고, 오히려 잘못된 판단을 하는 것입니다. 모델을 학습시킬 때에는, 항상 손실 함수 (Loss Function)을 사용하여 평가합니다. 예를 들어 Regression에서는 MSE라는 실제와 예측의 오차 제곱의 평균값을 사용하는데, 이 값은 0에 가까워질수록 좋은 결과이지만, 또 너무 가깝거나 0이 된다면 과적합이라고 볼 수 있습니다. 오히려 성능이 실제로 테스트할 때는 아주 안 좋게 나옵니다.



이를 해결하기 위한 방법은 다양하게 연구되고 있는데, 노이즈 제거, Feature selection과 데이터 정규화(0에서 1로 정규화하여 사용하는 방식) 같은 방식으로 과적 합의 위험을 줄일 수 있습니다. 또한 overfitting 발생 직전에 학습을 중단하는 방식을 사용합니다(Early stopping). 이를 위해서 validation set을 이용합니다.

회귀

회귀는 기본적인 데이터 분석 방법입니다. 짧게 정의하자면, 가격이나 확률처럼 연속된 실숫값을 정확히 예측하는 것을 목적으로 가진 방법입니다.

● 최소제곱법 (Least Square Method)

• 하나의 함수와 각 데이터의 차를 잔차(Residual)라고 합니다.

• 잔차들의 제곱을 최소화하는 알고리즘을 최소제곱법이라고 합니다.

$$E = \frac{1}{n}\sum_{i=1}^{n}(y - y_{pred})^2$$

• 위와 같은 공식의 최솟값을 찾는 방법입니다. y-y_pred를 잔차(residue)라고합니다.

• 최소제곱법을 사용하여 특정 차수의 함수의 계수를 찾는 것을 회귀선을 구하는 것이라고 볼 수 있습니다.

$$a = \frac{\sum_{i=1}^{n}(x_i - mean(X)) * (y_i - mean(Y))}{\sum_{i=1}^{n}(x_i - mean(X))^2}, \quad b = mean(Y) - a * mean(X)$$

● Linear Regression

• 선형 회귀는 데이터 전체의 경향을 나타내는 2차원의 직선을 예측하는 것을 의미합니다. 데이터의 경향이 선형적인 관계라고 생각되면 사용합니다.

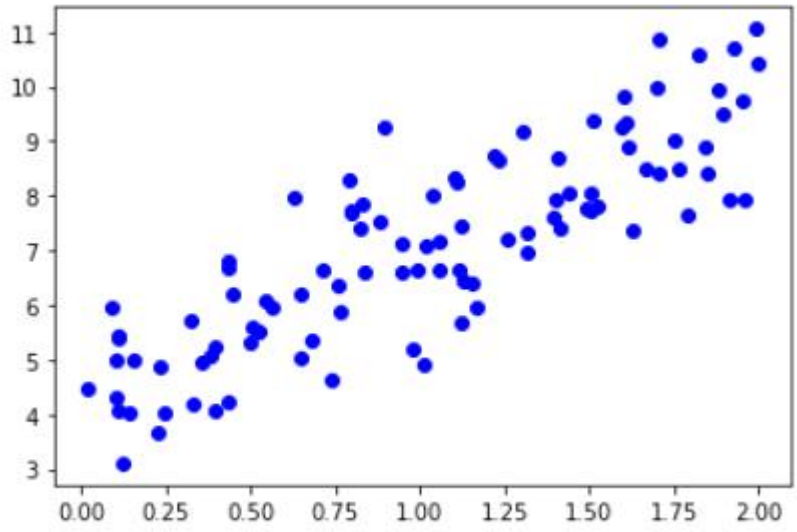
• 정의를 살펴보자면, 종속 변수(dependent variable)와 하나 이상의 독립 변수(independent variavle)사이의 관계를 모델링하는 선형적인(linear) 접근 방식을 의미합니다.

Test

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
```

```
[7]  x = 2 * np.random.rand(100, 1)
     y = 4 + 3 * x + np.random.randn(100, 1)
```

```
[8]  plt.plot(x, y, 'bo')
     plt.show()
```

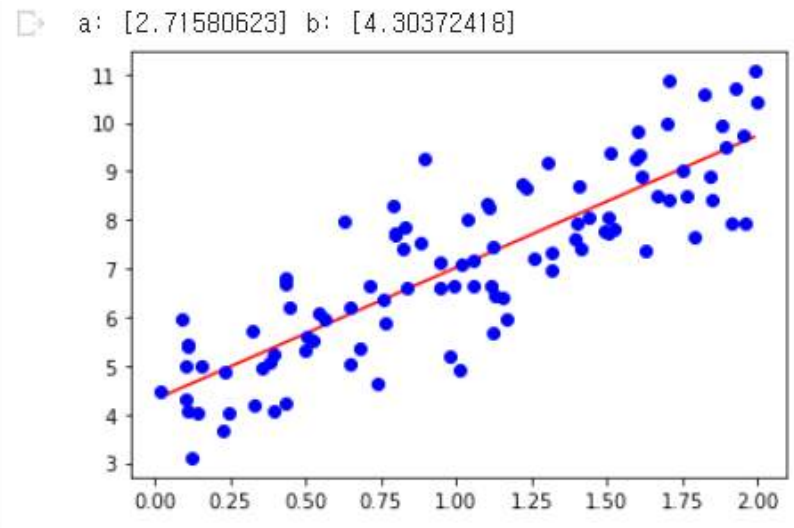


x값은 0부터 2까지의 랜덤한 값을 선언하였고, y=3x+4 경향에 가깝도록 y=3x+4+randn 값을 선언하였습니다.

Numpy를 사용한 선형 회귀

```
x_mean = sum(x) / len(x)
y_mean = sum(y) / len(y)

a = sum([(x_ - x_mean) * (y_ - y_mean) for x_, y_ in list(zip(x, y))])
a /= sum([(x_ - x_mean) ** 2 for x_ in x])
b = y_mean - a * x_mean
print('a:', a, 'b:', b)

x_pred = np.arange(min(x), max(x), 0.01)
y_pred = a * x_pred + b

plt.plot(x_pred, y_pred, 'r-')
plt.plot(x,y,'bo')
plt.show()
```

위에서 설명한 최소제곱법의 공식을 이용하여 코드를 작성하였습니다. x_pred는 x값들을 numpy배열로 바꾼것이고, y_pred는 최소제곱법을 사용하여 계산한 a와 b를 사용하여 계산한 값입니다.



a가 3에 가깝고, y절편인 b가 4에 가깝게 계산되었습니다.

Tensorflow를 사용한 선형 회귀

가중치와 편향을 랜덤 값으로 초기화

```
w = tf.Variable(random.random())
b = tf.Variable(random.random())
```

잔차(residue)를 구하는 함수입니다. 예측 값과 실제 데이터 값의 차이를 제곱한 값의 합을 구하여 이것의 평균을 ruturn합니다.

```
def residue():
  y_pred = w * x + b
  loss = tf.reduce_mean((y - y_pred) ** 2)
  return loss
```

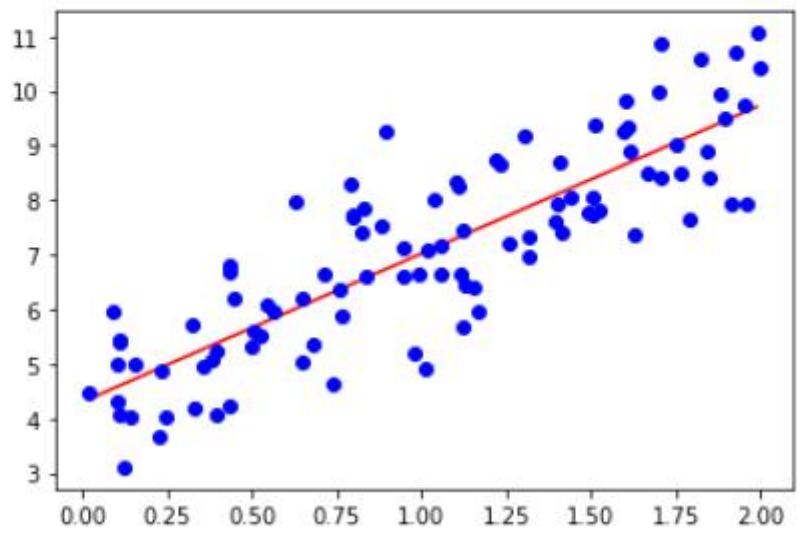
손실을 최소화하는 최적화 함수(optimizer)입니다. 해당 함수는 복잡한 미분 계산 및 가중치 업데이트를 자동으로 진행해주는 함수입니다. 해당 포스팅에서는 Adam 최적화 함수를 선정하였습니다. SGD와 함께 많이 쓰이는 함수로서, 좋은 성능을 보여줍니다. lr(learning rate)는 보통 0.1에서 0.0001 사이 값을 사용하는 데, 0.07을 사용하겠습니다.

```
optimizer = tf.optimizers.Adam(lr=0.07)
for i in range(1000):
  optimizer.minimize(residue, var_list=[w,b])

  if i % 100 == 99:
    print(i, 'w:', w.numpy(), 'b:', b.numpy(), 'loss:', residue().numpy())
```

optimizer.minimize(residue, var_list)는 var_list로 받은 변수 리스트를 residue함수가 최소가 되도록 1000번 학습하는 것입니다. 값의 변화를 확인하기 위해서, 100번에 한번씩 출력해보았습니다.

```
99 w: 3.4616957 b: 3.4488726 loss: 1.1315546
199 w: 3.1214435 b: 3.820652 loss: 0.9980471
299 w: 2.8987622 b: 4.0859327 loss: 0.9509062
399 w: 2.7839928 b: 4.2225533 loss: 0.9405457
499 w: 2.7369242 b: 4.2785854 loss: 0.93903464
599 w: 2.7212348 b: 4.2972617 loss: 0.9388848
699 w: 2.7169585 b: 4.302353 loss: 0.9388747
799 w: 2.7160063 b: 4.303486 loss: 0.9388742
899 w: 2.715834 b: 4.303691 loss: 0.9388743
999 w: 2.7158127 b: 4.3037157 loss: 0.9388743
```

```
x_pred = np.arange(min(x), max(x), 0.01)
y_pred = w * x_pred + b

plt.plot(x_pred, y_pred, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```



위 두 그래프를 보면 numpy로 계산한 a,b와 텐서플로우로 계산한 w,b가 완전히 일치하진 않지만, 아주 유사함을 확인할 수 있습니다.

sklearn을 사용한 선형 회귀

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(x, y)
print(lin_reg.intercept_, lin_reg.coef_)
```

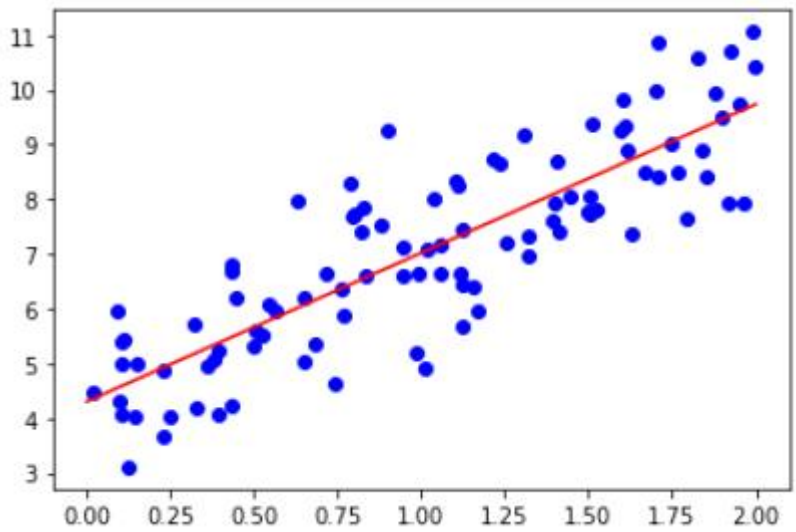
```
[4.30372418] [[2.71580623]]
```

sklearn 라이브러리는 LinearRegression을 위한 자체적인 모델을 제공합니다. fit함수를 통해서 데이터를 입력시켜주면됩니다.

```
x_b = np.c_[np.ones((100, 1)), x]
theta_b = np.linalg.inv(x_b.T.dot(x_b)).dot(x_b.T).dot(y)
print(theta_b)
```

```
[[4.30372418]
 [2.71580623]]
```

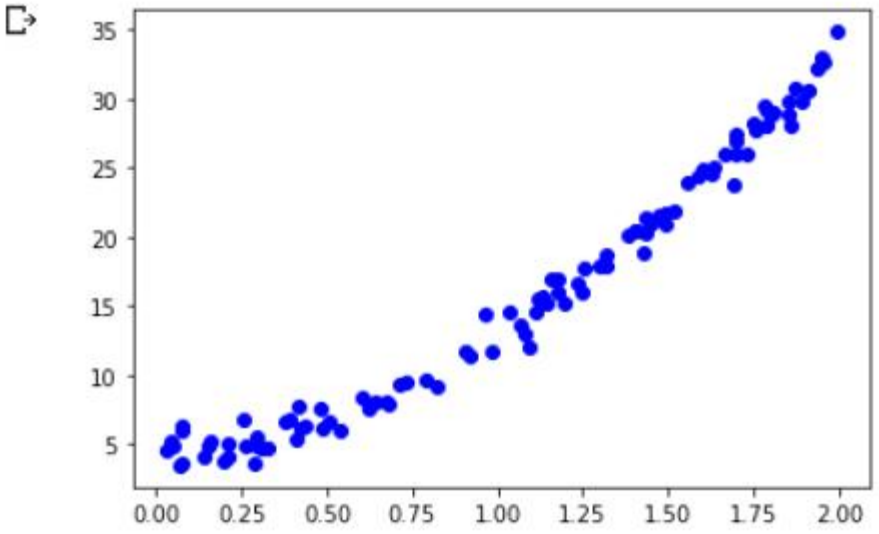
numpy 계산과 일치하게, tensorflow 계산과는 유사하게 결과가 나왔습니다. 이를 그래프로 나타내보면 아래와 같습니다.



Polynomial Regression

다항 회귀(Polynomial Regression)은 선형회귀의 직선을 2차 함수 이상으로 예측하는 것을 의미합니다.

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

x = 2 * np.random.rand(100, 1)
y = 6 * x**2 + 3 * x + 4 + np.random.randn(100, 1)
plt.plot(x, y, 'bo')
plt.show()
```



이번엔 이차함수인 y=6x^2+3*x+4 경향이 나타나도록 랜덤값들을 데이터로 만들어보았습니다.

Tensorflow를 사용한 다항 회귀

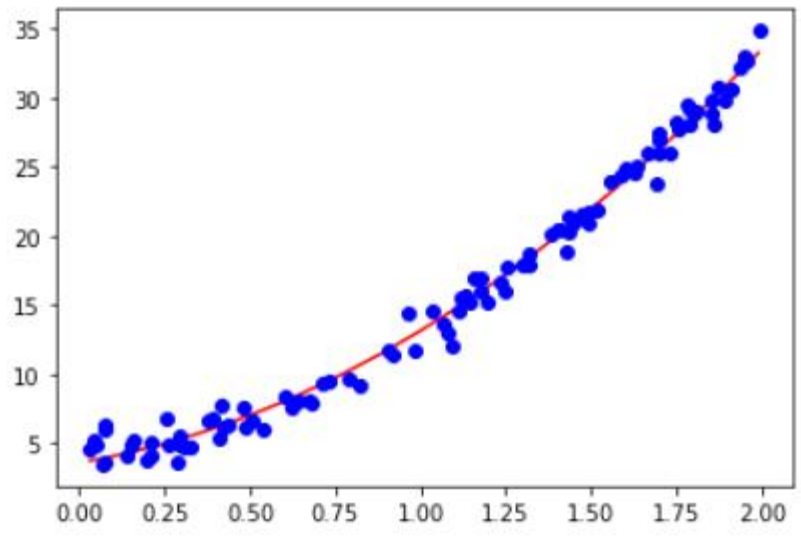
```
import random

w = tf.Variable(random.random())
b = tf.Variable(random.random())
c = tf.Variable(random.random())

def residue():
  y_pred = w * x**2 + b * x + c
  loss = tf.reduce_mean((y - y_pred) ** 2)
  return loss
```

```
optimizer = tf.optimizers.Adam(lr=0.07)
for i in range(1000):
  optimizer.minimize(residue, var_list=[w,b,c])

  if i % 100 == 99:
    print(i, 'w:', w.numpy(), 'b:', b.numpy(), 'c:', c.numpy(), 'loss:', residue().numpy())
```

```
99 w: 4.76735 b: 4.651852 c: 4.183384 loss: 1.1691953
199 w: 4.922726 b: 4.6539755 c: 3.9085171 loss: 0.9946548
299 w: 5.0445185 b: 4.6284957 c: 3.6857173 loss: 0.91065055
399 w: 5.1322393 b: 4.587645 c: 3.5547805 loss: 0.88022554
499 w: 5.1932282 b: 4.5343943 c: 3.4976208 loss: 0.86902547
599 w: 5.2389045 b: 4.4718437 c: 3.4857225 loss: 0.86243576
699 w: 5.278093 b: 4.402465 c: 3.496935 loss: 0.85651505
799 w: 5.315706 b: 4.327943 c: 3.5185158 loss: 0.8505806
899 w: 5.353825 b: 4.249402 c: 3.5444963 loss: 0.844614
999 w: 5.393047 b: 4.1676707 c: 3.5724802 loss: 0.8386885
```

위 결과를 보시면 선형 회귀와 유사한걸 확인할 수 있습니다. 달라진 점은 c라는 변수의 추가와 y식의 변화입니다.

```
x_pred = np.arange(min(x), max(x), 0.01)
y_pred = w * x_pred * x_pred + b * x_pred + c

plt.plot(x_pred, y_pred, 'r-')
plt.plot(x, y, 'bo')
plt.show()
```



학습이 계속 진행되면서 낮은 손실과 정확한 값들을 찾을 수 있습니다.

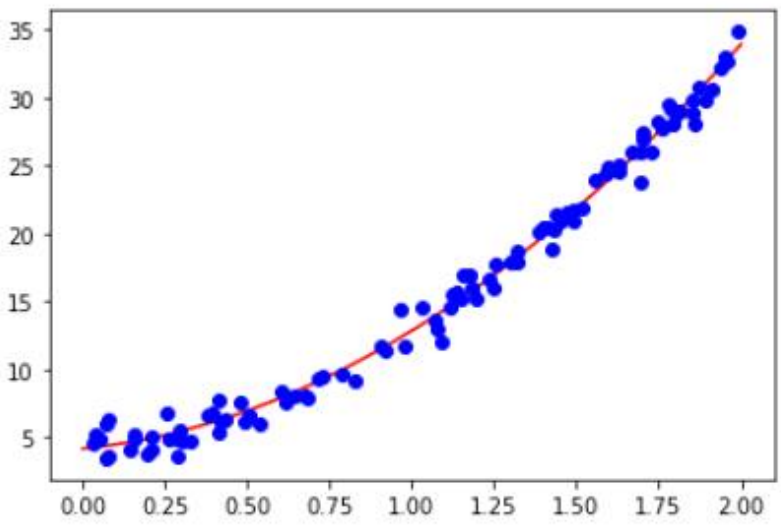
sklearn을 사용한 다항 회귀

sklearn에서는 LinearRegression은 제공합니다. 해당 클래스와 다항 회귀를 위해서 기존 데이터를 다항 회귀에 적합한 데이터로 바꾸어주는 PolynomialFeatures 클래스를 이용하여 다항 호귀 모델을 추정할 수 있습니다.

```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

poly_features = PolynomialFeatures(degree = 2, include_bias=True)
x_poly = poly_features.fit_transform(x)

lin_reg = LinearRegression()
lin_reg.fit(x_poly, y)
print(lin_reg.intercept_, lin_reg.coef_)
```

```
[4.15118489] [[0.         2.49439534 6.19331612]]
```

```
x_new = np.linspace(0, 2, 100).reshape(100,1)
x_new_poly = poly_features.transform(x_new)
y_predict = lin_reg.predict(x_new_poly)

plt.plot(x_new, y_predict, "r-")
plt.plot(x, y, "bo")
plt.show()
```



식의 차수를 늘리고, 변수를 늘리면, 3차 이상의 다항 함수들의 회귀도 가능합니다.

맺음말

머신 러닝의 이론적 부분엔 교수님이 강의하신 부분을 깊게 파고들어가 봤습니다. 참고 자료를 보시면 아시겠지만. 모두 해외 자료이기 때문에 제가 잘 이해하고 흡수 했는지 의심이 갑니다. 미흡하고 부족한 부분을 포트폴리오 작성할 때 공부하며 채우려는 의도가 잘 반영 됐는지 모르겠습니다. 하지만 분명히 기존 지식보단 한 층을 쌓고 뼈대를 굳건히 만드는 계기가 됐다고 자신있게 말할 수 있습니다.

회귀 부분은 여러 회귀를 알아보고 그 값들이 맞는지 확인을 했습니다. 계속 두 방법으로 확인을 했으며, 정확도와 손실 확인으로 제대로 테스트가 됐다는걸 볼 수 있습니다.

역시 공부는 주입식이 아닌 자율적이 훨씬 효율적이고 능률적이라고 생각합니다. 저번 포트폴리오 작성때도 그렇고 이번 포트폴리오 작성때도 그렇듯이 자율이 재미를 낳았습니다. 그 재미는 학습을 흡수시키고요. 이런 선순환이 계속되길 바라고, 교수님의 혜안을 높이 사며 이 포트폴리오를 마치고자 합니다. 감사합니다.

참고 및 출처

Machine Learning - Can We Please Just Agree What This Means
December 4, 2017 at 3:30pm
William Vorhies
https://www.datasciencecentral.com/profiles/blogs/machine-learning-can-we-please-just-agree-what-this-means

Batch Learning VS Online Learning
Figure 1 - uploaded by Fushenh Wang
https://www.researchgate.net/figure/Online-machine-learning-versus-batch-learning-a-Batch-machine-learning-workflow-b_fig1_316818527

O'REILLY
hands-on-machine-learning
https://www.oreilly.com/library/view/hands-on-machine-learning/9781491962282/ch01.html

Dwight Clough
the costs of poor data quality management
https://www.tech4g.com/the-costs-of-poor-data-quality-management/

Understanding Overfitting and Underfitting in Machine Learning
Sep 6, 2019
Aditya Tiwari
https://medium.com/analytics-vidhya/understanding-overfitting-and-underfitting-in-machine-learning-2a2f3577fb27