3 머신러닝

이 장에서 다룰 내용

1. 머신러닝 개요
2. 데이터 전처리
3. Scikit-learn
4. 머신러닝 알고리즘

1. 머신러닝 개요

머신 러닝(Machine Learning) 정의

□ 머신 러닝(Machine Learning)이란?

- 아서 사무엘(Arthur samuelm 1959)
  - "머신러닝은 명시적인 프로그래밍 없이 학습하는 능력을 갖추게 하는 연구 분야이다."
- 톰 미첼(Tom Mitchell, 1997)
  - "어떤 작업 T에 대한 컴퓨터 프로그램의 성능을 P로 측정 했을때 경험 E로 성능이 향상 되었다면 이 컴퓨터 프로그램은 작업 T와 성능측정 P에 대해 경험 E로 학습한 것이다."
- Ex1) 스팸필터
  - 스팸 메일과 일반 메일의 샘플을 이용해 스팸 메일 구분법을 배울 수 있는 머신런닝 프로그램
  - 시스템이 학습하는데 사용하는 샘플을 훈련데이터라고 한다.
  - T : 새로운 메일이 스팸매일인지 구분하는 것
  - E : 훈련데이터
  - P : 정확히 분류한 메일의 비율. 성능 측정을 정확도(accuarcy)라고 한다.
- Ex2)

http://www.edtpatips.com/wp-content/uploads/2014/12/teaching-strategy-184317176-1440x1008.jpg

Example: A program for soccer tactics

T : Win the game
P : Goals
E : (x) Players' movements, (y) Evaluation

머신 러닝 사례

머신 러닝 사용 사례

- 생산 라인에서 제품 이미지를 분석해서 자동으로 분류하기 - 합성곱 신경망(CNN) 사용
- 뇌를 스캔하여 종양 진단하기 - CNN을 사용해 이미지의 각 픽셀을 분류.(종양의 정확한 위치와 모양을 결정)
- 자동으로 뉴스 기사 분류하기 - 자연어 처리(NLP) 작업중 하나로 텍스트 분류 작업이다. RNN이나 CNN을 사용하여 해결 가능
- 다양한 성능 지표를 기반으로 내년도 회사의 수익 예측하기 - 회귀(Regression)작업으로 선형회귀나 다항회귀, SVM, 랜덤 포레스트등과 같은 회귀모델을 사용해서 해결 가능
- 구매 이력을 기반으로 고객을 나누고 각 집합마다 다른 마케팅 전략을 계획하기 - 군집(Clustering) 작업
- 신용카드 부정거래 감지하기 - 이상치 탐지 작업

머신 러닝 종류

- 머신러닝 시스템의 종류
  - 사람의 감독하에 훈련하는 것인지 아닌지에 따라 분류
    - 지도
    - 비지도
    - 준지도
    - 강화학습
  - 실시간으로 점진적인 학습을 하는 것인지 아닌지에 따른 분류
    - 온라인 학습
    - 배치 학습
  - 단순하게 알고 있는 데이터 포인터와 새 데이터 포인터를 비교하는 것인지 또는 훈련 데이터 셋에서 패턴을 발견하여 예측 모델을 만드는지에 따른 분류
    - 사례 기반 학습
    - 모델 기반 학습

머신 러닝 종류

지도학습(Supervised Learning)

- 지도학습에는 알고리즘에 주입하는 훈련 데이터에 레이블(Lable)이라는 원하는 답이 포함되어야 한다.
  - 분류(Classification)가 전형적인 지도 학습 작업이며, 숫자 인식을 좋은 예로 들 수 있다.
  - 예측 변수(Predictor Variable)라 불리는 특성(Feature, 주행거리, 연식, 브랜드등)을 사용해 가격 같은 최종적인 타깃 수치를 예측한다. 위와 같은 종류의 작업을 회귀(Regression)라고 부른다.
- 대표적인 지도 학습 알고리즘
  - K-최근접 이웃(k-Nearest Neighbors)
  - 선형 회귀(Linear Regression)
  - 로지스틱 회귀(Logistic Regression)
  - 서포트 벡터 머신(SVM, Support Vector Machine)
  - 결정 트리(Decision Tree)와 랜덤 포레스트(Random Forest)
  - 신경망(Neural Network)

머신 러닝 종류

비지도 학습(Unsupervised Learning)

- 비지도 학습(Unsupervised Learning)은 지도 학습에서 필요했던 레이블이 필요하지 않으며 시스템이 아무런 도움 없이 학습해야 한다.
- 대표적인 비지도 학습 알고리즘
  - 군집(Clustering)
    - K-평균(k-Means)
    - 계층 군집 분석(HCA, Hierarchical Cluster Analysis)
    - 기댓값 최대화(Expectation Maximization)
  - 시각화(Visualization)와 차원 축소(Dimensionality Reduction)
    - 주성분 분석(PCA, Principal Component Analysis)
    - 커널 PCA(Kernel PCA)
  - 연관 규칙 학습(Association Rule Learning)
    - 어프라이어리(Apriori)
    - 이클렛(Eclat)
- 계층 군집 알고리즘을 사용하면 각 그룹을 더 작은 그룹으로 세분화할 수 있다.
- 차원 축소는 너무 많은 정보를 잃지 않으면서 데이터를 간소화하는데 사용된다.
  - 예를 들어 차의 주행거리는 연식과 매우 연관되어 있으므로 차원 축소 알고리즘으로 두 특성을 차의 마모 정도를 나타내는 하나의 특성으로 합칠 수 있다. 이를 특성 추출(Feature Extraction)이라고 한다.
- 연관 규칙 학습은 대량의 데이터에서 특성 간의 흥미로운 관계를 찾는다.
  - 어떠한 상품을 구매한 사람이 다른 상품을 구매하는 경향이 있다는 것을 찾을 때 활용한다.

머신 러닝 종류

준지도 학습(Semisupervised Learning)

- 준지도 학습(Semisupervised Learning)에서는 레이블이 일부만 있어도 데이터를 다룰 수 있다.
- Ex) 구글 포토 호스팅 서비스
  - 이 서비스에 가족사진을 모두 올리면 사람 A는 사진 1, 5, 11에 있고, 사람 B는 사진 2, 5, 7에 있다고 자동으로 인식(군집, Clustering)하며 이는 비지도 학습이다.
  - 이제 이 사람들이 누구인가 하는 정보인데 사람마다 레이블이 하나씩만 주어지면(지도 학습) 사진에 있는 모든 사람의 이름을 알 수 있고, 편리하게 사진을 찾을 수 있다.

강화 학습(Reinforcement Learning)

- 학습하는 시스템을 에이전트(Agent)라고 부르며 환경(Environment)을 관찰해서 행동(Action)을 실행하고 보상(Reward)을 받는다. 시간이 지나면서 가장 큰 보상을 얻기 위해 정책(Policy)이라고 부르는 최상의 전략을 스스로 학습하고 정책은 주어진 상황에서 에이전트가 어떻게 행동해야 하는지를 판단한다.
- Ex) 알파고

머신 러닝 종류

배치 학습(Batch Learning)

- 시스템이 점진적으로 학습할 수 없으며 가용한 데이터를 모두 사용해 훈련시켜야 하는 방식이다.
- 시간과 자원을 많이 소모하여 일반적으로 오프라인에서 사용한다.
- 먼저 시스템을 훈련시키고 제품 시스템에 적용하면 더 이상의 학습 없이 실행된다. 즉, 학습한 것을 적용할 뿐이다. 이를 오프라인 학습(Offline Learning)이라고도 한다.

온라인 학습(Online Learning)

- 데이터를 순차적으로 한 개씩 또는 미니배치(Mini-Batch)라 부르는 작은 묶음 단위로 주입하여 시스템을 훈련시키는 방식으로 매 학습 단계가 빠르고 비용이 적게 들어 시스템은 데이터가 도착하는 대로 즉시 학습할 수 있다.
- 온라인 학습은 연속적으로 데이터를 받고 빠른 변화에 스스로 적응해야 하는 시스템에 적합하다.
- 온라인 학습 시스템에서 중요한 파라미터 중 하나는 변화하는 데이터에 얼마나 빠르게 적응할 것인지 이다. 이를 학습률(Learning Rate)이라고 한다.
- 학습률을 높게 하면 시스템이 데이터에 빠르게 적응하지만 예전 데이터를 금방 잊어버리게 된다. 학습률이 낮으면 시스템의 관성이 더 커져서 더 느리게 학습되지만 새로운 데이터에 있는 잡음이나 대표성 없는 데이터 포인트에 덜 민감해진다.

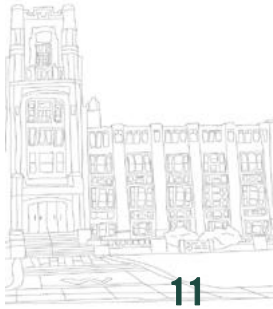
머신 러닝 종류

사례 기반 학습 (Instance-Based Learning)

- 시스템이 사례를 기억함으로써 학습하는 방식이다.
- 스팸 메일과 동일한 메일을 스팸이라고 지정하는 대신 스팸 메일과 매우 유사한 메일을 구분하도록 스팸 필터를 프로그램할 수 있는데 이렇게 하려면 두 메일 사이의 유사도(Similarity)를 측정해야 한다. 두 메일 사이의 매우 간단한 유사도 측정 방법은 공통으로 포함한 단어의 수를 세는 것이며 스팸 메일과 공통으로 가지고 있는 단어가 많으면 스팸으로 분류한다.

모델 기반 학습(Model-Based Learning)

- 샘플들의 모델을 만들어 예측하는 방식이다.



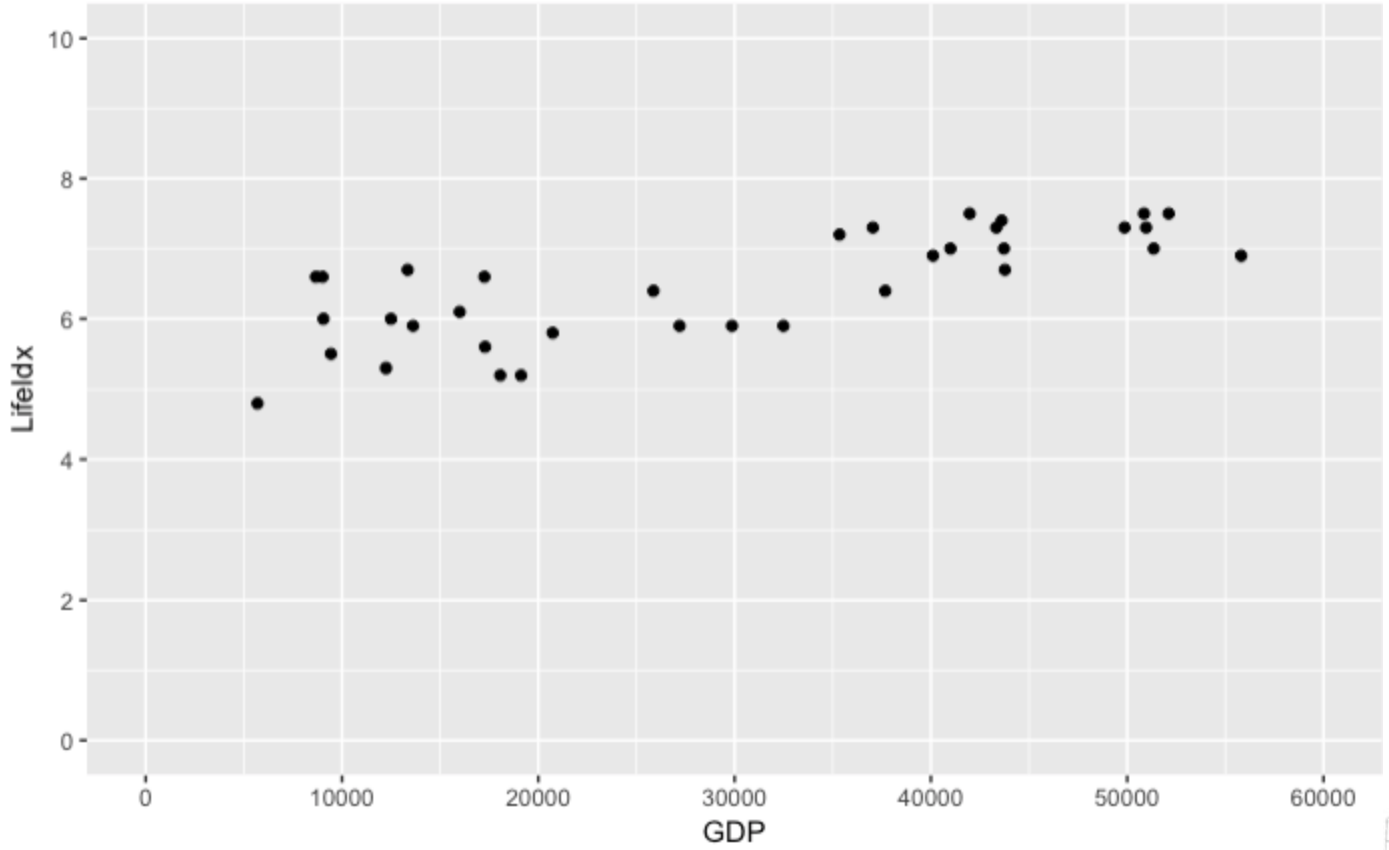
머신 러닝 종류

- 모델 기반 학습의 예시로 돈이 사람을 행복하게 만드는지 알아보도록 하자.
- OECD 웹사이트에서 더 나은 삶의 지표(Better Life Index) 데이터와 IMF 웹사이트에서 1인당 GDP 통계를 내려 받은 다음 두 데이터 테이블을 합치고 1인당 GDP로 정렬한다.

Country <fctr>	LifeIdx <dbl>	GDP <dbl>
Luxembourg	6.9	101994.09
Switzerland	7.5	80675.31
Norway	7.5	74822.11
United States	6.9	55805.20
Denmark	7.5	52114.17
Ireland	7.0	51350.74
Australia	7.3	50961.87
Iceland	7.5	50854.58
Sweden	7.3	49866.27
United Kingdom	6.7	43770.69

머신 러닝 종류

- 위 데이터를 보기 편하게 시각화 하면 다음과 같이 나타난다.



- 삶의 만족도는 국가의 1인당 GDP가 증가할수록 거의 선형으로 상승하는 것을 확인할 수 있었다. 그러므로, 1인당 GDP의 선형 함수로 삶의 만족도를 모델링 해보겠습니다. 이 단계를 모델 선택(Model Selection)이라고 한다. 1인당 GDP라는 특성 하나를 가진 삶의 만족도에 대한 선형 모델(Linear Model)이다.

머신 러닝 종류

- 실습 예제

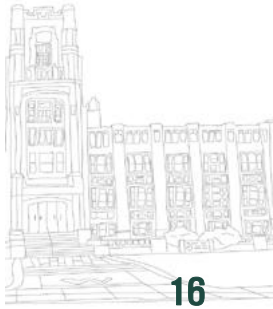
```
import sklearn
import os
datapath = os.path.join("datasets", "lifesat", "")

import matplotlib as mpl
mpl.rc('axes', labelsize=14)
mpl.rc('xtick', labelsize=12)
mpl.rc('ytick', labelsize=12)

# 데이터 다운로드
import urllib
DOWNLOAD_ROOT = "https://raw.githubusercontent.com/rickiepark/handson-ml2/master/"
os.makedirs(datapath, exist_ok=True)
for filename in ("oecd_bli_2015.csv", "gdp_per_capita.csv"):
    print("Downloading", filename)
    url = DOWNLOAD_ROOT + "datasets/lifesat/" + filename
    urllib.request.urlretrieve(url, datapath + filename)
```

머신 러닝 종류

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import sklearn.linear_model
oecd_bli = pd.read_csv(datapath + "oecd_bli_2015.csv", thousands=',')
oecd_bli

gdp_per_capita = pd.read_csv(datapath + "gdp_per_capita.csv",thousands=',',delimiter='₩t',
                             encoding='latin1', na_values="n/a")
gdp_per_capita

oecd_bli = oecd_bli[oecd_bli["INEQUALITY"]=="TOT"]
oecd_bli

oecd_bli = oecd_bli.pivot(index="Country", columns="Indicator", values="Value")
oecd_bli

gdp_per_capita.rename(columns={"2015": "GDP per capita"}, inplace=True)
gdp_per_capita

gdp_per_capita.set_index("Country", inplace=True)
gdp_per_capita

full_country_stats = pd.merge(left=oecd_bli, right=gdp_per_capita, left_index=True,right_index=True)
full_country_stats
```

머신 러닝 종류

```
full_country_stats.sort_values(by="GDP per capita", inplace=True)
full_country_stats

remove_indices = [0, 1, 6, 8, 33, 34, 35]
keep_indices = list(set(range(36)) - set(remove_indices))
keep_indices

country_stats = full_country_stats[["GDP per capita", 'Life satisfaction']].iloc[keep_indices]
country_stats

X = np.c_[country_stats["GDP per capita"]]
X

y = np.c_[country_stats["Life satisfaction"]]
y

# 데이터 시각화
country_stats.plot(kind='scatter', x="GDP per capita", y='Life satisfaction')
plt.show()

model = sklearn.linear_model.LinearRegression() # 선형 모델 선택 . 선형 회귀 모델
model.fit(X, y) # 모델 훈련
X_new = [[22587]]  # 키프로스 1인당 GDP
print(model.predict(X_new)) # 키프로스에 대한 예측 출력 [[ 5.96242338]]
```



나쁜 데이터와 나쁜 알고리즘

러닝 머신의 주요 도전과제

- 우리의 주요 작업은 학습 알고리즘을 선택해서 어떤 데이터에 훈련 시키는 것이므로 문제가 될 수 있는 두가지는 나쁜 데이터와 나쁜 알고리즘이다.

나쁜 데이터

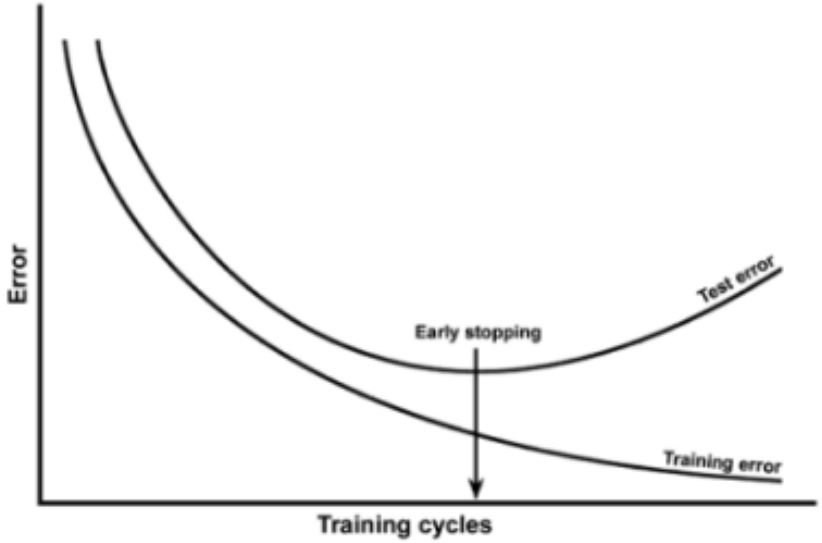
- 충분하지 않은 양의 훈련 데이터
  - 대부분의 머신러닝 알고리즘이 잘 작동하기 위해서는 데이터가 많아야 한다.
  - 이미지나 음성 인식 같은 복잡한 문제의 경우 수백만 개가 필요할 수 있다.
- 대표성 없는 훈련 데이터
  - 일반화가 잘되기 위해서는 우리가 일반화하기 원하는 새로운 사례를 훈련 데이터가 잘 대표하는 것이 중요하다.
  - 샘플이 작으면 샘플링 잡음(Sampling Noise)1이 생기고, 표본 추출 방법이 잘못되면 대표성을 띄지 못할 수 있다. 이를 샘플링 편향(Sampling Bias)이라고 한다.
- 낮은 품질의 데이터
  - 훈련 데이터가 에러, 이상치, 잡음으로 가득하다면 머신러닝 시스템이 내재되어 있는 패턴을 찾기 어려울 수 있다.
  - 다음은 훈련 데이터에 정제가 필요한 경우
    - 일부 샘플이 이상치라는게 명확하다면 그것을 무시하거나 수정하는 것이 좋다.
    - 일부 샘플에 특성이 몇 개 빠져 있다면 무시할지 빠진 값을 채울지 아니면 샘플을 무시할지 결정해야 한다.

나쁜 데이터와 나쁜 알고리즘

- 관련 없는 특성
  - 훈련 데이터에 관련 없는 특성이 적고 관련 있는 특성이 충분해야 학습을 진행할 수 있다.
  - 훈련에 사용할 좋은 특성들을 찾아야 하며, 이를 특성 공학(Feature Engineering)이라고 한다. 특성 공학은 다음과 같은 작업을 포함한다.
    - 특성 선택 (Feature Selection) : 가지고 있는 특성 중에서 훈련에 가장 유용한 특성을 선택한다.
    - 특성 추출 (Feature Extraction) : 특성을 결합하여 더 유용한 특성을 만든다.
    - 새로운 데이터를 수집해 새 특성을 만든다.

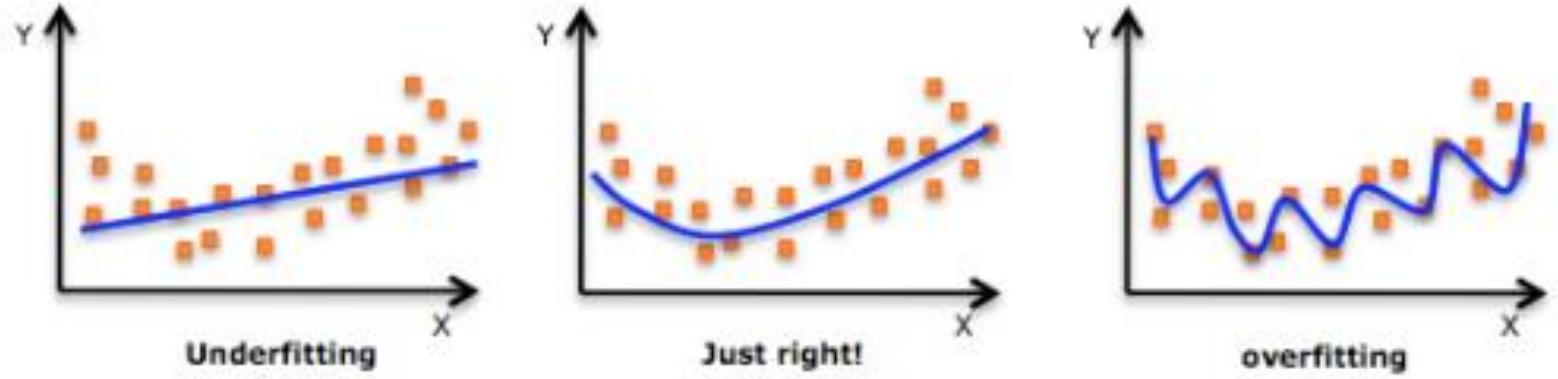
나쁜 알고리즘

- 과대적합(Overfitting)
  - 학습데이터를 과하게 잘 학습한 것을 의미
  - 새로운 데이터나 테스트 데이터에서는 해당 학습한 내용이 제대로 반영되지 못할 수 있다.
  - 학습 데이터에 대해서는 오차가 감소하지만, 실제 데이터에 대해서는 오차가 증가하는 지점이 존재한다.
  - 그림에서는 테스트 에러가 감소하다 갑자기 치솟는 부분에서 과대적합이 발생했다고 볼 수 있다.



나쁜 데이터와 나쁜 알고리즘

- 과대적합의 해결 방법
  - 파라미터 수가 적은 모델을 선택하거나, 모델에 제약을 가하여 단순화시킨다.
  - 훈련 데이터를 더 많이 확보한다.
  - 훈련 데이터의 잡음을 줄인다. (Outlier, Error 제거)
- 과소적합(Underfitting)
  - 과대적합의 반대 개념
  - 델이 너무 단순해서 데이터의 내재된 구조를 학습하지 못할 때 발생한다.
  - 과소적합의 해결 방법
    - 파라미터가 더 많은 강력한 모델을 선택한다.
    - 학습 알고리즘에 더 좋은 특성을 제공한다.
    - 모델의 제약을 줄인다.
- 데이터를 올바르게 학습시키기 위해서는 과대적합과 과소적합의 중간점을 찾는 것이 좋다. 아래 그림에서 보이는 바와 같이 너무 잘 분류해도, 분류하지 못해도 올바른 모델이라고 할 수 없다.



테스트와 검증

테스트와 검증

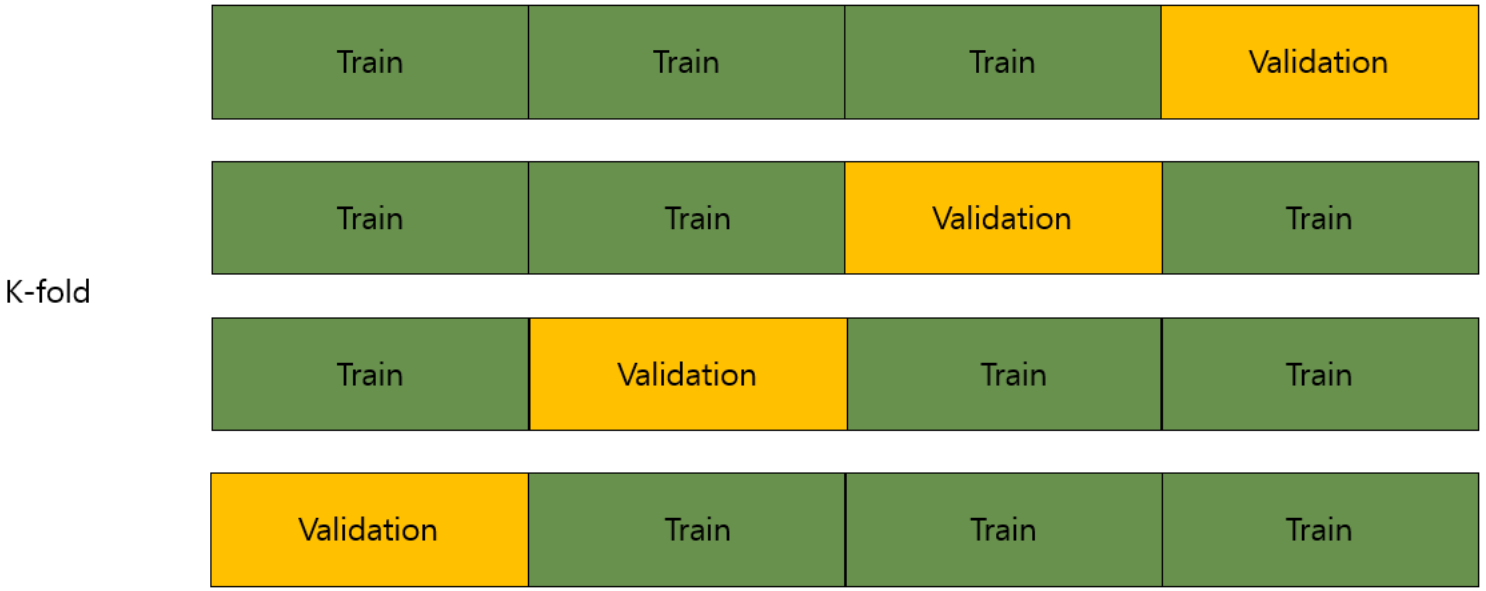
- 모델이 새로운 샘플에 얼마나 잘 일반화 될지 아는 방법
  - 새로운 샘플에 실제로 테스트 하는 방법
    - 실제 서비스에 모델을 넣고 잘 동작하는지 모니터링 하는 것
  - 훈련 데이터를 훈련세트와 테스트 세트로 니누어 테스트 하는 방법
    - 새로운 샘플에 대한 오류 비율을 일반화 오차(Generalization Error)라고 하며 테스트 세트에서 모델을 평가 함으로써 이 오차에 대한 추정값(Estimation)을 얻는다.
    - 이 값이 새로운 샘플에 모델이 얼마나 잘 작동하는지 알려준다.
    - 훈련 오차가 낮고 일반화 오차가 높다면 이는 모델이 훈련에 의해 과적합 되었음을 뜻한다는것을 알 수 있다.
- 하이퍼파라미터 튜닝과 모델 선택
  - 딥러닝 모델을 구축할 때, 훈련 데이터와 테스트 데이터만으로도 훈련의 척도를 판단할 수 있다.
  - 하지만, 훈련 데이터에 대한 학습만을 바탕으로 모델의 설정(Hyperparameter)를 튜닝하게 되면 과대적합(overfitting)이 일어날 가능성이 매우 크다.
  - 또한, 테스트 데이터는 학습에서 모델에 간접적으로라도 영향을 미치면 안 되기 때문에 테스트 데이터로 검증을 해서는 안 된다.
  - 그래서 검증(validation) 데이터셋을 따로 두어 매 훈련마다 검증 데이터셋에 대해 평가하여 모델을 튜닝해야 한다.

테스트와 검증

- 홀드아웃 검증(Hold-out validation)
  - 기본적인 검증 방법으로 단순히 훈련데이터와 테스트 데이터로 나누고, 나눠진 훈련데이터에서 다시 검증 데이터셋을 따로 떼어내는 방법이다.

  - 이 방식의 문재점은 데이터가 적을 때는 각 데이터셋이 전체 데이터를 통계적으로 대표하지 못할 가능성이 높다. 즉, 하나의 데이터셋에 다양한 특징을 지닌 데이터들이 포함되지 않을 수 있다는 것이다.
    - 이를 확인하는 방법은 새롭게 데이터를 셔플링하여 다시 모델을 학습시켰을 때 모델의 성능이 많이 차이가 나면 이 문제라고 볼 수 있다.
- K-겹 교차 검증(K-fold cross-validation)
  - 홀드아웃에 비해 훈련 세트의 분할에 덜 민감한 성능 추정을 얻을 수 있다.
  - 중복을 허락하지 않고 훈련 데이터 셋을 k개의 폴드로 랜덤하게 나눈 뒤, k-1개의 폴드로 모델을 훈련하고 나머지 하나의 폴드로 성능을 평가한다.
  - 이 과정을 k번 반복하여 k개의 모델과 성능 추정을 얻는다.
  - 만족할만한 성능이 나온 하이퍼파라미터를 찾은 후에는 전체 훈련 세트를 사용하여 모델을 다시 훈련하고 독립적인 테스트 세트를 이용하여 최종 성능 추정을 한다.
  - 모델의 총 validation score는 각 훈련의 validation score의 평균이다.

테스트와 검증



- **데이터를 나눌 시 주의점**
  - **대표성 : 훈련 데이터셋과 테스트 데이터셋은 전체 데이터에 대한 대표성을 띄고 있어야 한다.**
  - **시간의 방향 : 과거 데이터로부터 미래 데이터를 예측하고자 할 경우에는 데이터를 섞어서는 안 된다. 이런 문제는 훈련 데이터셋에 있는 데이터보다 테스트 데이터셋의 모든 데이터가 미래의 것이어야 한다.**
  - **데이터 중복 : 각 훈련, 검증, 테스트 데이터셋에는 데이터 포인트의 중복이 있어서는 안 된다. 데이터가 중복되면 올바른 평가를 할 수 없기 때문이다.**

테스트와 검증

K-fold cross-validation 예제(jupyter notebook 사용)

- Ex1)

```
from sklearn.datasets import load_boston
import pandas as pd
boston = load_boston()
dfX = pd.DataFrame(boston.data, columns=boston.feature_names)
dfy = pd.DataFrame(boston.target, columns=["MEDV"])
df = pd.concat([dfX, dfy], axis=1)
df

from sklearn.model_selection import train_test_split
df_train, df_test = train_test_split(df, test_size=0.3, random_state=0)
df_train.shape, df_test.shape

dfX_train, dfX_test, dfy_train, dfy_test = train_test_split(dfX, dfy, test_size=0.3,
            random_state=0)
dfX_train.shape, dfy_train.shape, dfX_test.shape, dfy_test.shape
```

테스트와 검증

- # 교차검증 예제 1

```
from sklearn.model_selection import KFold
import statsmodels.api as sm
import numpy as np
scores = np.zeros(5)
cv = KFold(5, shuffle=True, random_state=0)
for i, (idx_train, idx_test) in enumerate(cv.split(df)):
    df_train = df.iloc[idx_train]
    df_test = df.iloc[idx_test]

    # 선형 회귀분석
    model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.feature_names),
                                                    data=df_train)
    result = model.fit()

    pred = result.predict(df_test)
    rss = ((df_test.MEDV - pred) ** 2).sum()
    tss = ((df_test.MEDV - df_test.MEDV.mean())** 2).sum() rsquared = 1 - rss / tss
    scores[i] = rsquared
    print("학습 R2 = {:.8f}, 검증 R2 = {:.8f}".format(result.rsquared, rsquared))
```

테스트와 검증

- # 교차검증 예제 2

```
from sklearn.metrics import r2_score

scores = np.zeros(5)
cv = KFold(5, shuffle=True, random_state=0)
for i, (idx_train, idx_test) in enumerate(cv.split(df)):
    df_train = df.iloc[idx_train]
    df_test = df.iloc[idx_test]

    model = sm.OLS.from_formula("MEDV ~ " + "+".join(boston.feature_names),
                                                                    data=df_train)
    result = model.fit()

    pred = result.predict(df_test)
    rsquared = r2_score(df_test.MEDV, pred)

    scores[i] = rsquared # 검증 결과 차례로 저장

scores # 검증 결과 출력
```

테스트와 검증

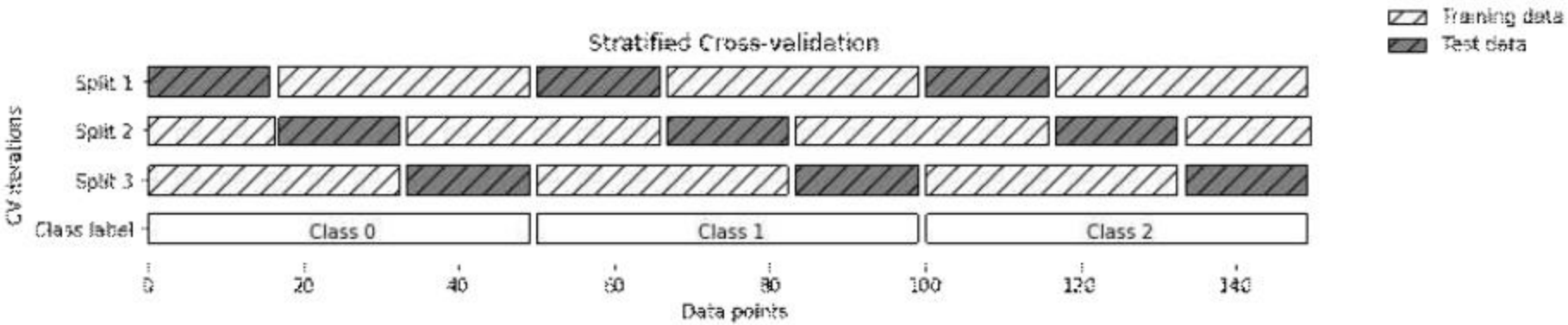
- # 교차검증 예제 3

```
from sklearn.base import BaseEstimator, RegressorMixin
import statsmodels.formula.api as smf
import statsmodels.api as sm
class StatsmodelsOLS(BaseEstimator, RegressorMixin):
    def __init__(self, formula):
        self.formula = formula
        self.model = None
        self.data = None
        self.result = None
    def fit(self, dfX, dfy):
        self.data = pd.concat([dfX, dfy], axis=1)
        self.model = smf.ols(self.formula, data=self.data)
        self.result = self.model.fit()
    def predict(self, new_data):
        return self.result.predict(new_data)

from sklearn.model_selection import cross_val_score
model = StatsmodelsOLS("MEDV ~ " + "+".join(boston.feature_names))
cv = KFold(5, shuffle=True, random_state=0)
cross_val_score(model, dfX, dfy, scoring="r2", cv=cv)
```

테스트와 검증

stratified

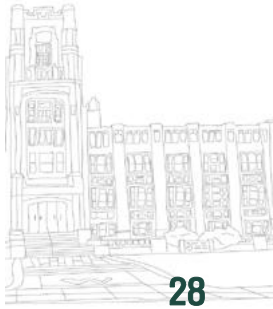
- 트레인셋과 밸리데이션셋을 무작위로 나누기 때문에 hold-out할 때 타겟 클래스가 일정하지 않을 수도 있다.
- 이 경우, 트레인셋과 밸리데이션셋의 데이터 분포가 달라지므로 학습에도 영향을 미친다.
- 기계학습은 학습 데이 터의 분포와 현실 세계 데이터의 분포가 동일하다는 전제가 있다. 기본 전제가 무너지므로 학습 모델의 성능이 떨어지게 되는 것이다.
- 이를 예방하기 위해서 타겟 클래스의 비율이 일정하게 나누어주는 것을 stratified 기법이 라고 한다.



테스트와 검증

- **Ex1)**

```
# KFold 대신 StratifiedKFold 사용
from sklearn.datasets import load_iris
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

data = load_iris()

iris = pd.DataFrame(data=data.data, columns=data.feature_names)

target = pd.Series(data.target, dtype="category")
target =target.cat.rename_categories(data.target_names)
iris["species"] = target
iris.rename({"sepal length (cm)":"sepal_length", "sepal width (cm)": "sepal_width", "petal length
(cm)":"petal_length", "petal width (cm)":"petal_width"}, axis=1, inplace=True)

X_train, X_test, y_train, y_test = train_test_split(iris.iloc[:, :-1],
          iris.iloc[:, -1], test_size=0.33, random_state=42)

model = DecisionTreeClassifier(criterion="gini", splitter="best", max_depth=None,
              min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
              max_features=None, random_state=42, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              class_weight=None)
```



테스트와 검증

```
# KFold 대신 StratifiedKFold 사용
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

cv = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)
results = cross_val_score(model, X_train, y_train, cv=cv)
fin_result = np.mean(results)
for i, _ in enumerate(results):
    print("{}번째 교차검증 정확도: {}".format(i, _))

print("교차검증 최종 정확도: {}".format(fin_result))
```

테스트와 검증

- Ex2) 앞부분은 앞 예제와 동일

```
#  StratifiedKFold 대신 KFold 사용
cv = KFold(n_splits=10, shuffle=True, random_state=42)
results = cross_val_score(model, X_train, y_train, cv=cv)
fin_result = np.mean(results)

for i, _ in enumerate(results):
    print("{}번째 교차검증 정확도: {}".format(i, _))

print("교차검증 최종 정확도: {}".format(fin_result))
```