CH2. 머신러닝 프로젝트 처음부터 끝까지

MADE BY_이다현

주요 단계 정리

1. 큰 그림을 본다
2. 데이터를 구한다
3. 데이터로부터 통찰을 얻기 위해 탐색하고 시각화
4. 머신러닝 알고리즘을 위해 데이터 준비
5. 모델을 선택하고 훈련시키기
6. 모델을 상세하게 조정
7. 솔루션 제시
8. 시스템을 론칭하고 모니터링하고 유지보수

2.2 큰 그림 보기

주제 : 캘리포니아 인구조사 데이터를 사용해 캘리포니아 주택 가격 모델을 만들기

- 블록그룹 = 미국 인구조사국에서 샘플 데이터를 발표하는 데 사용하는 최소한의 지리적 단위 = 구역
- 블록그룹별 인구, 중간소득, 중간 주택가격등을 담은 데이터 셋
- 이 데이터로 모델을 학습시켜 다른 측정 데이터가 주어졌을 때, 구역의 중간 주택 가격을 예측해야 함

▼ 2.2.1 문제정의

- **Q1)** 비즈니스의 '목적'이 무엇인가?

  목적을 아는 것 = 문제를 어떻게 구성할지, 어떤 알고리즘을 선택할지, 모델평가에 어떤 성능지표를 사용할지, 모델 튜닝을 위해 얼마나 노력할지 등을 결정한다

- **Q2)** '현재 솔루션' 은 어떻게 구성되어 있나?

  문제 해결 방법에 대한 정보 + 참고성능으로도 사용

- **문제 정의**

  [1] 레이블 된 (각 샘플이 기대 출력값, 즉 구역의 중간 주택 가격을 가지고 있음) 훈련 샘플이 있음 → 지도학습 작업

  [2] 지도학습 중, 예측을 해야하므로 회귀문제이고, 예측에 사용할 특성이 여러개 이므로 다중회귀

  [3] 각 구역마다 하나의 값(주택가격)을 예측하므로 단변량 회귀 (cf. 구역 별로 여러 값을 예측하면 다변량 회귀)

  [4] 데이터의 연속적 흐름이 없음 + 데이터 크기가 작음 → 일반적인 배치 학습이 적절

- **개념 파이프라인**

  데이터 처리 컴포넌트(component=요소)들이 연속되어 있는 것을 데이터 파이프라인 이라고 한다.

  - 데이터 파이프라인 : 데이터를 한 장소에서 다른 장소로 차례대로 전달하는, 데이터로 구성된 일련의 시스템
  - 머신러닝 파이프라인 : 머신러닝 프로젝트에서 일련의 과정 (load data - data analysis - feature engineering - data validation - data split - build&train model - model validation - model serving) 을 머신러닝 파이프라인이라고 한다.

▼ 2.2.2 성능측정지표 선택

- 회귀 문제의 전형적인 성능 지표 = **평균제곱근오차 (RMSE)** : 오차가 커질수록 예측에 얼마나 많은 오류가 있는지 가늠하게 해줌

표기법 (자세한 내용은 교재 참고!)

- m : 샘플 수
- $x^{(i)}$: i번째 샘플의 전체 특성값 벡터
- $y^{(i)}$: 해당 레이블 (샘플의 기대 출력값)
- X : 데이터셋에 있는 모든 샘플의 모든 특성값(레이블은 제외)을 포함하는 행렬
- *h* : 시스템의 예측함수이며, 가설 이라고도 한다. 시스템이 하나의 샘플 특성 벡터 x 를 받으면 그 샘플에 대한 예측값 y^ = h(x) 를 출력한다
- RMSE(X,h) : 가설 h를 사용하여 일련의 샘플을 평가하는 비용함수

- **평균절대오차** = MAE : 이상치로 보이는 구역이 많은 경우 회귀 문제에 사용하는 성능 지표
- **RMSE vs MAE 비교**

  o 두 성능 지표 모두 예측 값의 벡터와 타깃 값의 벡터 사이의 거리를 재는 방법이다. 거리 측정에는 여러가지 방법(또는 norm노름)이 가능하다.

  o RMSE → 제곱항을 합한 것 = 유클리디안 노름

  o MAE → 절댓값의 합을 계산한 것 = 맨해튼 노름

  o 노름의 지수가 클수록 큰 값의 원소에 치우치며 작은 값은 무시되므로 RMSE가 MAE보다 조금 더 이상치에 민감하다

▼ 2.2.3 가정검사

지금까지 만든 가정을 나열하고 검사해보는 것이 좋다. 이 과정에서 심각한 문제를 일찍 발견할 수도 있다.

📖 2.3 데이터 가져오기

▼ 2.3.1 작업환경 만들기

(생략→ 주피터 노트북 참고)

▼ 2.3.2 데이터 다운로드

(생략→ 주피터 노트북 참고)

▼ 2.3.3 데이터 구조 훑어보기

- `head()` : 처음 다섯 행 확인
- `info()` : 전체 행 수, 각 특성의 데이터 타입과 널이 아닌 값의 개수를 확인
- `df['column_name'].value_counts()` : 범주형 변수에 대해 카테고리와 각 카테고리 별 개수 확인
- `describe()` : 숫자형 특성의 요약정보 제시
- `hist()` : 데이터의 형태를 빠르게 검토하는 방법으로, 숫자형 특성을 히스토그램으로 그려본다. 전체 데이터 셋에 대해 hist() 메서드를 호출하면 모든 숫자형 특성에 대한 히스토그램을 출력한다.
- 히스토그램 그래프에서 확인할 사항

  1) 데이터의 스케일(측정단위) 확인

  2) 최댓값과 최솟값을 한정 지었는지 확인

  3) skewness가 있는 분포를 종모양의 분포가 되도록 변형하기 위해 꼬리가 두꺼운 히스토그램 확인

▼ 2.3.4 테스트 세트 만들기

데이터를 깊게 들여다보기 전에 테스트 세트를 따로 떼어 놓아야 한다. 그리고 테스트 세트를 절! 대! 들여다보면 안된다. 만약 테스트 세트를 들여다보게 되면 테스트 세트에서 겉으로 드러난 어떤 패턴에 속아 특정 머신러닝 모델을 선택하게 될지도 모른다. 그 테스트 세트로 일

반화 오차를 추정하면 매우 낙관적인 추정이 되며 시스템을 론칭했을 때, 기대한 성능이 나오지 않을 것이다. (**데이터 스누핑 편향** 현상)

- **테스트 세트 생성** : 무작위로 어떤 샘플을 선택해 데이터 셋의 20% 정도를 떼어 놓으면 된다.

  → 자세한 과정은 주피터 노트북 참고

  (1) split_train_test 함수 정의해서 생성하기

  (2) 식별자의 해시값을 계산해 데이터셋의 업데이트 이후에도 안정적인 훈련/테스트 분할을 진행

  (3) 사이킷런의 train_test_split 메소드로 생성하기

- **계층적 샘플링** : 순수한 무작위 샘플링 방식은, 데이터 셋이 충분히 크다면 일반적으로 괜찮으나, 그렇지 않으면 샘플링 편향이 생길 가능성이 크다. 따라서 계층적 샘플링을 통해 대표성을 보완한다.
  - 사이킷런의 `StratifiedShuffleSplit` 을 사용할 수 있다.

📖 2.4 데이터 이해를 위한 탐색과 시각화

▼ 2.4.1 지리적 데이터 시각화

위도와 경도의 지리 정보가 있으니 모든 구역을 산점도로 만들어 데이터를 시각화하는 것은 좋은 생각이다.

- 파이썬 산포 그래프 그리기 (scatter)

```
df.plot(kind='scatter', x='변수명', y='변수명' , alpha=숫자, s= , c= ,cmap= )
```

→ s 옵션

- 선택적으로 입력하면 되는데, 마커의 크기를 설정한다. 스칼라로 입력할 경우 마커의 크기는 고정이다. 변수명으로 입력할 경우, 변수의 값의 크기에 따라 마커마다 다른 크기를 선택할 수 있다.

→ c 옵션

- 마커의 색상을 설정한다. 변수명으로 입력할 경우, 변수 값의 크기에 따라 각 마커마다 다른 색상을 설정할 수 있다.

→ cmap 옵션

- c 매개변수에 데이터의 카테고리가 지정되었다면 마커의 색상을 변경하기 위해 미리 정의된 색상표를 이용하면 된다.

▼ 2.4.2 상관관계 조사

- `df.corr()` : 표준 상관계수를 corr() 메서드를 이용해 계산
- 상관계수는 선형적인 상관관계만 측정한다. 비선형적인 관계는 잡을 수 없다. 또한 상관계수는 '기울기' 와 상관 없다. 단지 선형성을 (직선모양을) 띄는지만 보면 된다.
- `scatter_matrix( )` : 특성 사이의 상관관계를 확인하는 다른 방법으로, 숫자형 특성 사이에 산점도를 그려주는 판다스 함수 사용하기. 이때, 대각선 방향은 각 변수 자신에 대한 것이라 그냥 히스토그램을 그려줌

▼ 2.4.3 특성 조합으로 실험

- 정제해야 할 이상한 데이터 확인, 특성 사이에서 상관관계 발견, 꼬리가 두터운 분포의 변수는 로그 스케일 등으로 데이터를 변형
- 머신러닝 알고리즘용 데이터를 준비하기 전 마지막으로 해볼 수 있는 것은 여러 특성의 조합을 시도해보는 것이다.
  - 예를들어 특정 구역의 방 개수는 얼마나 많은 가구 수가 있는지 모른다면 그다지 유용하지 않다. 진짜 필요한 것은 가구당 방 개수이다.

📖 2.5 머신러닝 알고리즘을 위한 데이터 준비

데이터 준비 작업을 함수로 만들어 자동화 하는 이유

- 어떤 데이터 셋에 대해서도 데이터 변환을 손쉽게 가능
- 향후 프로젝트에 사용할 수 있는 변환 라이브러리를 점진적으로 구축

- 실제 시스템에서 알고리즘에 새 데이터를 주입하기 전에 변환시키는데 이 함수를 사용할 수 있음
- 여러가지 데이터 변환을 쉽게 시도할 수 있고, 어떤 조합이 가장 좋은지 확인하는데 편리

▼ 2.5.1 데이터 정제

[1] 결측치 처리하기

- **방법 3가지**
  1. `dropna()` : 해당구역을 제거
  2. `drop()` : 전체 특성을 삭제
  3. `fillna()` : 0, 평균, 중간값 등 어떤 값으로 채우기
- **사이킷런의 SimpleImputer** 로 결측치 처리하기 → 자세한 과정은 주피터 노트북 확인하기
- **사이킷런의 설계 철학** : 일관성 : 모든 객체가 일관되고 단순한 인터페이스를 공유
  1. 추정기 estimator
     - 데이터셋을 기반으로 일련의 모델 파라미터들을 추정하는 객체를 추정기라고 한다. (예를 들어 imputer 객체는 추정기 이다)
     - 추정 자체는 `fit()` 메서드에 의해 수행됨
     - cf) 인스턴스 변수 : 객체지향 프로그래밍에서 객체가 각각 독립적으로 가지고 있는 변수
  2. 변환기 transformer
     - imputer 같이 데이터 셋을 변환하는 추정기를 변환기라고 한다.
     - 변환은 `transform()` 메서드에 의해 수행됨
     - 모든 변환기는 fit() 과 transform()을 연달아 호출하는 것과 동일한 `fit_transform()` 메서드도 가지고 있다.
  3. 예측기 predictor
     - 일부 추정기는 데이터셋에 대해 예측을 할 수 있다. 예를들어 LinearRegression 모델이 예측기가 될 수 있다.
     - `predict()` 메서드는 새로운 데이터셋을 받아 이에 상응하는 예측값을 반환
     - `score()` 메서드는 테스트 세트를 함께 사용해 예측의 품질을 측정
  4. 검사 가능
     - 모든 추정기는 하이퍼파라미터는 공개 인스턴스 변수로 직접 접근 가능. 예를들면 `imputer.strategy` , `imputer.statistics_`

▼ 2.5.2 텍스트와 범주형 특성 다루기

텍스트형 변수에 대해 범주형인지 임의의 텍스트인지 확인하기

- **사이킷런의 OrdinalEncoder 클래스**
  - 범주형이라면, 대부분의 머신러닝 알고리즘은 숫자를 다루므로 카테고리를 텍스트에서 숫자로 변환함
  - categories_ 인스턴스 변수를 사용해 오리지날 인코딩한 카테고리 목록을 얻을 수 있다.
  - 그러나 이 표현방식의 문제는 머신러닝 알고리즘이 가까이 있는 두 값이 떨어져 있는 두 값보다 더 비슷하게 생각한다는 점이다. 순서가 있는 카테고리는 괜찮지만, 단순한 명목 변수는 좋지 않다. 이런 경우에는 카테고리별 이진 특성을 만들어 해결한다. 예를 들어 카테고리가 'INLAND'일 때 한 특성이 1이고 그 외 특성은 0인 방식이다. (=원-핫 인코딩)
- **사이킷런의 원-핫 인코딩 클래스 : OneHotEncoder**
  - 데이터 타입 : **사이파이 희소행렬** (scipy sparse matrix) → 수천 개의 카테고리가 있는 범주형 특성일 경우 매우 효율적이다.
  - 원-핫 인코딩을 하면 열이 수천개인 행렬로 변하고 각 행은 1이 하나뿐이고 그 외에는 모두 0으로 채워져 있을 것이다. 0을 모두 메모리에 저장하는 것은 낭비이므로 희소 행렬은 0이 아닌 원소의 취리만 저장한다.
  - 이 행렬을 넘파이 배열로 바꾸려면 `toarray()` 메서드를 호출하면 된다.
- **Scipy**
  - 과학 기술 계산을 위한 파이썬 라이브러리로, 넘파이 상위에서 구동되는 라이브러리 정도로 이해해도 무방하다.
  - 수치적분 루틴과 미분방정식 해석기, 방정식 근을 구하는 알고리즘, 표준 연속/이산 확률분포와 다양한 통계관련 도구 등을 제공한다.

- **사이파이 희소행렬**
  - 사용하는 이유 : 대규모 행렬을 다룰 때, 메모리 문제가 생긴다면 고려해볼 수 있는 것이 희소행렬이다.
  - 원소 값이 0이 아닌 부분에 대해서만 좌표와 값을 저장하고 나머지는 모두 0으로 간주한다.
  - 희소행렬을 표현하는 방식에는 coo,csr등이 있다.

▼ 2.5.3 나만의 변환기

- 사이킷런에서 유용한 변환기를 많이 제공하지만, 특별한 정제 작업이나 어떤 특성들을 조합하는 등의 작업을 위해 자신만의 변환기를 만들어야 할 때가 있다. 사이킷런은 '덕 타이핑'을 지원하므로 fit() , transform(), fit_transform() 메서드를 구현한 파이썬 클래스를 만들면 된다.
- 덕 타이핑
  - 사람이 오리처럼 행동하면 오리로 봐도 무방하다라는게 덕 타이핑(Duck Typing)이다. 타입을 미리 정하는게 아니라 실행이 되었을 때 해당 Method들을 확인하여 타입을 정한다.
  - example

Example Code

```
class Parrot:
    def fly(self):
        print("Parrot flying")

class Airplane:
    def fly(self):
        print("Airplane flying")

class Whale:
    def swim(self):
        print("Whale swimming")

def lift_off(entity):
    entity.fly()

parrot = Parrot()
airplane = Airplane()
whale = Whale()

lift_off(parrot) # prints `Parrot flying`
lift_off(airplane) # prints `Airplane flying`
lift_off(whale) # Throws the error ``'Whale' object has no attribute 'fly'``
```

▼ 2.5.4 특성 스케일링

데이터에 적용할 가장 중요한 변환 중 하나가 '특성 스케일링' 이다. 몇 가지를 빼고는 머신러닝 알고리즘은 입력 숫자 특성들의 스케일이 많이 다르면 잘 작동하지 않는다. (타깃값에 대한 스케일링은 일반적으로 불필요하다.) 모든 특성의 범위를 같도록 만들어주는 방법으로 min-max 스케일링과 표준화가 널리 사용된다.

- **min-max 스케일링 (=정규화)**

    - 0~1 범위에 들도록 값을 이동하고 스케일을 조정하는 방법이다. 데이터에서 최솟값을 뺀 후 최댓값과 최솟값의 차이로 나누면 이렇게 할 수 있다.
    - 사이킷런에는 이에 해당하는 MinMaxScaler 변환기를 제공한다.
    - 0과 1사이를 원하지 않는다면 feature_range 매개변수로 범위를 변경할 수 있다.
- **표준화**
    - 평균을 뺀 후, 표준편차로 나누어 결과 분포의 분산이 1이 되도록 한다. min-max 스케일링과는 달리 표준화는 범위의 상한과 하한이 없어 어떤 알고리즘에서는 문제가 될 수 있다. 그러나 표준화는 이상치에 영향을 덜 받는다.
    - 사이킷런에는 표준화를 위한 StandardScaler 변환기가 있다.
- 모든 변환기에서 스케일링은 전체 데이터가 아니고 훈련 데이터에 대해서만 fit() 메서드를 적용해야한다. 그런 다음 훈련세트와 테스트 세트 (그리고 새로운 데이터)에 대해 transform() 메서드를 사용한다.

▼ 2.5.5 변환 파이프라인

- 변환 단계가 많으며, 정확한 순서대로 실행되어야 한다. 사이킷런에는 연속된 변환을 순서대로 처리할 수 있도록 도와주는 Pipeline 클래스가 있다.
- 파이프라인은 연속된 단계를 나타내는 이름 , 추정기 쌍의 목록을 입력으로 받는다. 마지막 단계에는 변환기와 추정기를 모두 사용할 수 있고 그 외에는 모두 변환기여야 한다. → 자세한 코드는 주피터 노트북 참고
- 하나의 변환기로 각 열마다 적절한 변환을 적용해 모든 열을 처리할 수 있다면 더 편리하다. 사이킷런에서는 이런 기능을 위해 **ColumnTransformer** 가 추가되었다. → 자세한 코드는 주피터 노트북 참고
- 희소행렬 vs 밀집행렬
    - 희소행렬 : 행렬의 값이 대부분 0인 경우
    - 밀집행렬 : 희소행렬의 반대되는 표현으로 대부분 행렬의 값이 1인 경우
    - 어느 한쪽이 아주 많다면 적은쪽의 인덱스만 별도로 저장하면 메모리 공간을 효율적으로 사용할 수 있다.
    - ColumnTransformer 는 최종행렬의 밀집정도를 추정해 밀집도가 0.3 임곗값보다 낮으면 희소 행렬을 반환한다.
- 여러 변환기를 적용하고 결과를 합쳐주는 또 다른 클래스로 FeatureUnion 클래스가 있다. 하지만 이는 각 변환기에 열을 따로 지정할 수 없고 전체 데이터에 모두 적용된다. 이 문제를 해결하려면 열을 선택해주는 사용자 정의 변환기를 따로 만들어야 한다.

📖 2.6 모델 선택과 훈련

▼ 2.6.1 훈련 세트에서 훈련하고 평가하기

- 예) 선형회귀모델훈련 : `LinearRgression()`

  → 자세한 코드는 주피터 노트북 참고
- 예) 결정트리 : `DecisionTreeRegressor()`

  → 자세한 코드는 주피터 노트북 참고
- 오차 (RMSE)가 큰 경우 : 특성들이 충분한 정보를 제공하지 못했거나, 모델이 충분히 강력하지 못하다는 사실을 말해준다.
- 과소적합을 해결하는 방법 : 더 강력한 모델을 선택, 더 좋은 특성을 주입, 모델의 규제를 감소

▼ 2.6.2 교차 검증을 사용한 평가

결정트리 모델을 평가하는 방법

1) train_test_split 함수를 사용해 훈련 세트를 더 작은 훈련세트와 검증세트로 나누고 더 작은 훈련 세트에서 모델을 훈련 시키고 검증 세트로 모델을 평가하는 방법

2) k-겹 교차검증

* 훈련세트를 폴드(fold)라 불리는 10개의 서브셋으로 무작위로 분할하고, 결정트리 모델을 10번 훈련하고 평가하는데, 매번 다른 폴드를 선택해 평가에 사용하고 나머지 9개 폴드는 훈련에 사용한다. 최종적으로 10개의 평가점수가 담긴 배열이 결과로 나온다.

* 사이킷런의 교차검증 기능은 scoring 매개변수에 (낮을수록 좋은) 비용함수가 아니라 (클수록 좋은) 효용함수를 기대한다. 그래서 평균 제곱오차(MSE)의 반댓값(즉, 음숫값)을 계산하는 neg_mean_squared_error 함수를 사용한다. 평균 제곱 오차가 작을수록 좋은 비용함수 이므로 부호가 반대가 되어야 scoring 매개변수 정의에 맞다.

📖 2.7 모델 세부 튜닝

▼ 2.7.1 그리드 탐색

- 가장 단순한 방법은 만족할만한 하이퍼파라미터 조합을 찾을 때까지 수동으로 조정하는 것이겠지만, 시간이 오래 걸릴 수 있다.
- 사이킷런의 **GridSearchCV**를 사용하면, 가능한 모든 하이퍼파라미터 조합에 대해 교차 검증을 사용해 평가하게 된다. 탐색하고자 하는 하이퍼파라미터와 시도해볼 값을 지정하기만 하면 된다. → 자세한 코드는 주피터 노트북

cf) 하이퍼 파라미터 : 모델링 할 때 사용자가 직접 세팅해주는 값을 뜻한다.

▼ 2.7.2 랜덤 탐색

그리트 탐색방법은 비교적 적은 수의 조합을 탐구할 때 괜찮다. 그러나 하이퍼파라미터 탐색 공간이 커지면 **RandomizedSearchCV** 를 사용하는 편이 더 좋다. 그리드서치과 비슷한 방법으로 사용하지만, 가능한 모든 조합을 시도하는 대신 각 반복마다 아이퍼파라미터에 임의의 수를 대입해 지정한 횟수만큼 평가한다.

▼ 2.7.3 앙상블 방법

모델을 세밀하게 튜닝하는 방법은 최상의 모델을 연결해보는 것이다. 결정트리의 앙상블인 랜덤 포레스트가 결정트리 하나보다 더 성능이 좋은 것처럼 말이다. 모델의 그룹(=앙상블) 이 최상의 단일 모델보다 더 나은 성능을 발휘할 때가 많다.

▼ 2.7.4 최상의 모델과 오차 분석

최상의 모델을 분석하면 문제에 대한 좋은 통찰을 얻는 경우가 많다. 예를 들어 랜덤포레스트가 정확한 예측을 만들기 위한 각 특성의 상대적인 중요도를 알려준다. `feature_importances_` 중요도에 대한 정보를 바탕으로 덜 중요한 특성들을 제외할 수 있다.

▼ 2.7.5 테스트 세트로 시스템 평가하기

테스트 세트에서 최종모델을 평가할 차례이다. 테스트세트에서 예측변수와 레이블을 얻은 후, full_pipeline을 사용해 데이터를 변환하고 (테스트 세트에서 훈련하면 안되므로 fit_transform()이 아니라 transform()에서 호출해야 한다!!!!!!!!!!!!!!) 테스트 세트에서 최종 모델을 평가한다.

- `scipy.stats.t.interval()` 을 사용해 오차의 95% 신뢰구간을 구하여 추정값의 정확도를 따져볼 수 있다.
- 하이퍼파라미터 튜닝을 많이 했다면 교차 검증을 사용해 측정한 것보다 조금 성능이 낮은 것이 보통이다. 우리 시스템이 검증 데이터에서 좋은 성능을 내도록 세밀하게 튜닝되었기 때문에 새로운 데이터 셋에는 잘 작동하지 않을 가능성이 크다. 이런 경우가 생기더라도 테스트 세트에서 성능 수치를 좋게 하려고 하이퍼파라미터를 튜닝하려 시도하면 안된다. 향상된 성능은 일반화되기 어렵기 때문이다.

📖 2.8 론칭, 모니터링, 시스템 유지보수

1. 배포
   - 전체 전처리 파이프라인과 예측 파이프라인이 포함된 훈련된 사이킷런 모델을 저장하기. 그리고 훈련된 모델을 상용 환경에서 로드하고 predict() 메서드를 호출해 예측을 만든다.
   - 혹은 클라우드에 배포할 수 있다.
2. 모니터링 코드 작성
   - 일정 간격으로 시스템의 실시간 성능을 체크하고 성능이 떨어졌을 때 알람을 통지할 수 있는 모니터링 코드를 작성해야 한다. 갑작스런 성능감소 뿐만 아니라 긴 시간동안 서서히 성능이 감소하는 것도 감지해야 한다.
   - ex) 고양이와 강아지 사진을 분류하도록 훈련된 모델도, 카메라 성능이 계속 변화함에 따라 재훈련할 필요가 있다.
   - 데이터가 계속 변화하면 데이터셋을 업뎃하고 모델을 정기적으로 다시 훈련해야 하는데, 따라서 전체 과정에서 가능한 많은 것을 자동화 (함수생성) 해야 한다.

     [자동화 할 수 있는 것]
     - 정기적으로 새로운 데이터를 수집하고 레이블 달기
     - 모델을 훈련하고 하이퍼파라미터를 자동으로 세부 튜닝하는 스크립트 작성

        - 업데이트된 테스트 세트에서 새로운 모델과 이전 모델을 평가하는 스크립트를 하나 더 작성한다. 성능이 감소하지 않으면 새로운 모델을 제품에 배포한다.
    - 입력 데이터 품질을 평가해야 한다.
3. 모델 백업
    - 새로운 모델이 어떤 이유로 올바르지 않게 작동하는 경우, 이전 모델로 빠르게 롤백하기 위한 절차와 도구를 준비해야 한다. 백업을 가지고 있으면 새로운 모델과 이전 모델을 쉽게 비교할 수 있다.