[Week1] 파머완 1~3장

CHAPTER 01 파이썬 기반의 머신러닝과 생태계 이해

01 머신러닝의 개념

머신러닝(Machine Learning)

- 애플리케이션 수정없이도 데이터를 기반으로 패턴을 학습하고 결과를 예측하는 알고리즘 기법의 통칭
- 일정한 패턴을 찾기 어려운 경우(ex. 스팸메일 필터링) 데이터를 기반으로 숨겨진 패턴을 인지
- 데이터 분석 영역 → 머신러닝 기반 예측 분석
- 데이터에 매우 의존적이라는 단점 → 최적의 데이터를 준비하는 능력 중요

머신러닝의 분류

- 지도학습 : 분류, 회귀, 추천시스템, 시각/음성 감지/인지, 텍스트 분석(NLP)
- 비지도학습 : 클러스터링, 차원 축소, 강화학습

02 파이썬 머신러닝 생태계를 구성하는 주요 패키지

머신러닝 패키지 : 사이킷런(Scikit-Learn), 텐서플로, 케라스

행렬/선형대수/통계 패키지 : 넘파이(NumPy) → 행렬 기반 데이터 처리 특화, 사이파이(SciPy)

데이터 핸들링 : 판다스 → 2차원 데이터 처리에 특화

시각화 : 맷플롯립(Matplotlib), 시본(Seaborn)

03 넘파이

넘파이(NumPy, Numerical Python)

- 선형대수 기반 프로그램을 쉽게 만들 수 있도록 지원
- C/C++ 저수준 언어 기반 호환 API 제공
- 배열 기반 연산, 데이터 핸들링

넘파이 ndarray 개요

- array( ) : 다양한 인자를 입력 받아 ndarray로 변환
- ndarray : 넘파이 기반 데이터 타입, 다차원 배열 쉽게 생성 및 연산 수행 가능
- .shape : (행, 열)
- .ndim : 차원

ndarray 차원, 크기 변경

- reshape(r, c) : r행 c열 형태로 변환

ndarray 데이터 타입

- 숫자, 문자열, 불 값 모두 가능하지만 같은 데이터 타입만 포함 가능
- 서로 다른 데이터 타입을 가진 리스트를 ndarray로 변경 → 더 큰 데이터 타입으로 형 변환 일괄 적용
- astype( ) : 데이터 타입 변경

ndarray 편리하게 생성

- arange(n) : 0~n-1 까지의 값을 ndarray 데이터값으로 변환
- zeros((r,c)) : 해당 shape 값을 모두 0으로 가지는 ndarray 반환
- ones((r,c)) : 해당 shape 값을 모두 1로 가지는 ndarray 반환

인덱싱

- 단일 값 추출 : ndarray[i]

- -1 인자 : ndarray와 호환될 수 있는 사이즈로 변환
- reshape(-1, 1) : 여러 개의 행을 가지되 반드시 1개의 열을 가진 2차원 ndarray로 변환됨을 보장
- tolist( ) : ndarray를 리스트 자료형으로 변환

행렬의 정렬

- np.sort( ) : 원 행렬을 그대로 유지한 채 정렬된 행렬 반환
  ** np.sort( )[::-1] : 내림차순 정렬
- ndarray.sort( ) : 원 행렬 자체를 정렬 형태로 변환, 반환 값은 None
- np.argsort( ) : 정렬된 원본 행렬에서 기존 원본 행렬의 원소에 대한 인덱스가 필요할 때 사용

- 슬라이싱 : ndarray[a:b]
- 팬시 인덱싱 : 리스트나 ndarray로 인덱스 집합을 지정하면 해당 위치의 인덱스에 해당하는 ndarray를 반환
- 불린 인덱싱 : 조건 필터링, 검색으로 추출

선형대수연산

- np.dot( ) : 행렬 내적(행렬 곱) 계산
- transpose( ) : 전치 행렬 계산

04 데이터 핸들링 - 판다스

판다스

- 2차원 데이터를 효율적으로 가공/처리할 수 있는 라이브러리
- DataFrame : 2차원 데이터를 담는 데이터 구조체, 여러 개의 Series로 이루어짐
- Index : 개별 데이터를 고유하게 식별하는 Key 값
- Series : 칼럼이 하나인 데이터 구조체

DataFrame 변환

- ndarray, 리스트, 딕셔너리 → DataFrame 변환 : 칼럼명 지정
- DataFrame → ndarray 변환 : values 이용
- DataFrame → 리스트 변환 : tolist( ) 호출
- DataFrame → 딕셔너리 변환 : to_dict( ) 호출

데이터 셀렉션 및 필터링

- DataFrame '[ ]' : 칼럼명 문자, 인덱스 변환 가능한 표현식 (칼럼 지정 연산자)
- iloc[ ] : 위치 기반 인덱싱 방식으로 동작 → 행과 열의 위치를 0을 출발점으로 하는 세로축, 가로축 좌표 정숫값으로 지정
- loc[ ] : 명칭 기반 인덱싱 방식으로 동작 → 인덱스 값으로 행 위치, 칼럼 명칭으로 열 위치 지정
- 행과 열을 함께 사용하여 데이터를 추출해야 한다면 iloc[ ] / loc[ ]을 사용해야 함
- 불린 인덱싱 : [ ] & loc[ ]

결손 데이터 처리

판다스 시작

- read_csv(filepath, sep=',') : 해당 경로의 데이터를 dataframe으로 로딩
- info( ) : 행열 수, 칼럼별 데이터 타입, Null 데이터 개수
- describe( ) : 칼럼별 숫자형 데이터 값의 통계량
- value_counts( ) : 칼럼 값 분포 반환 ** dropna=False : Null값 포함하여 건수 계산

DataFrame 데이터 삭제

- drop(labels=None, axis=0, index=None, columns=None, level=None, inplace=False, errors='raise')
  - axis : 특정 행 삭제할 때 axis = 0, 특정 열을 삭제할 때는 axis = 1
  - inplace = False : 자기 자신의 DataFrame 데이터는 삭제X, 삭제된 결과를 반환

    ** inplace = True : 자기 자신 DataFrame 데이터 삭제, None 반환

정렬

- sort_values( )
  - by : 특정 칼럼에서 정렬 수행
  - ascending = True : 오름차순 정렬
  - ascending = False : 내림차순 정렬
- Aggregation : count( ), mean( ), min( ), max( ), sum( ) …
- groupby( ) : agg( )를 이용해 여러 개의 칼럼에 서로 다른 aggregation 함수 적용 가능

apply lambda

- isna( ) : 결손 데이터 여부 확인
- isna( ).sum( ) : 결손데이터 개수 확인
- fillna( ) : 결손 데이터 대체

- lambda 식 : 함수 선언과 함수 내 처리를 한 줄로 쉽게 변환하는 식 → lambda 입력인자 : 계산식
- 입력인자가 여러 개인 경우 map( )으로 결합
- lambda 입력인자 : 반환값 if 조건식 else 반환값

CHAPTER 2 사이킷런으로 시작하는 머신러닝

01 사이킷런 소개와 특징

사이킷런 : 파이썬 머신러닝 라이브러리

- 쉽고 가장 파이썬스러운 API 제공
- 다양한 알고리즘 & 편리한 프레임워크와 API 제공
- 오랜 기간 검증된 성숙한 라이브러리

02 첫 번째 머신러닝 만들어 보기 - 붓꽃 품종 예측하기

분류(Classification)

- 대표적인 지도학습
- 지도학습 : 학습을 위한 다양한 피처와 분류 결정값인 레이블 데이터로 모델을 학습 → 별도의 테스트 데이터 세트에서 미지의 레이블 예측

실습

- sklearn.dataset : 사이킷런에서 자체적으로 제공하는 데이터 세트 생성 모듈 모임
- sklearn.tree : 트리 기반 알고리즘 구현한 클래스 모임
- sklearn.model_selection : 학습, 검증, 예측 데이터로 데이터 분리 / 최적의 하이퍼 파라미터 평가를 위한 모듈 모임

  ** 하이퍼 파라미터 : 최적의 학습을 위히 입력하는 파라미터, 성능 튜닝
- train_test_split(피처 데이터셋, 레이블 데이터셋, test_size, random_state) : 학습 데이터와 테스트 데이터 분리, test_size 입력값 비율로 분할
- 데이터셋 분리 → DecisionTreeClassifier 객체 생성 → fit( )으로 학습 → predict( )로 예측값 반환 → accuracy_score( )로 정확도 평가

03 사이킷런의 기반 프레임워크 익히기

Estimator 이해

- Estimator 클래스 : Classifier 클래스 + Regressor 클래스 → fit( )과 predict( ) 구현
- evaluation함수(ex. cross_val_score( )), 하이퍼파라미터 튜닝 지원 클래스의 경우 Estimator를 인자로 받아 수행

사이킷런 주요 모듈(p.92~93 표)

- sklearn.preprocessing : 데이터 전처리를 위한 가공 기능 제공
- sklearn.feature_selection : 알고리즘에 큰 영향을 미치는 피처를 우선순위대로 셀렉션 작업 수행 기능 제공
- sklearn.feature_extraction : 벡터화된 피처 추출
- sklearn.decomposition : 차원 축소 관련 알고리즘 지원

- sklearn.metrics : 성능 측정 방법 제공
- sklearn.ensemble : 앙상블 알고리즘 제공
- sklearn.linear_model : 회귀 관련 알고리즘 지원
- sklearn.naive_bayes : 나이브 베이즈 알고리즘 제공
- sklearn.neighbors : 최근접 이웃 알고리즘 제공
- sklearn.svm : 서포트 벡터 머신 알고리즘 제공
- sklearn.tree : 의사결정트리 알고리즘 제공
- sklearn.cluster : 비지도 클러스터링 알고리즘 제공
- sklearn.pipleline : 변환, 학습, 예측 등을 함께 묶어서 실행할 수 있는 유틸리티 제공

04 Model Selection 모듈 소개

학습/테스트 데이터 세트 분리 - train_test_split( )

- train_test_split(피처 데이터셋, 레이블 데이터셋) → 튜플 형태로 반환 (학습 데이터 피처, 테스트 데이터 피처, 학습 데이터 레이블, 테스트 데이터 레이블)
  - test_size = 0.25 : 테스트 데이터셋 크기 설정
  - train_size : 학습 데이터셋 크기 설정
  - shuffle = True : 데이터 분리 전 데이터를 섞을지 결정
  - random_state : 호출할 때마다 동일한 데이터셋을 생성하기 위해 설정하는 난수값

교차검증

: 별도의 여러 세트로 구성된 학습 데이터셋과 검증 데이터셋에서 학습과 평가를 수행 → 과적합 개선

▼ K-Fold 교차검증 : K개의 데이터 폴드 세트를 만들어서 K번만큼 각 폴트 세트에 학습과 검증 평가를 반복적으로 수행

```
# K-Fold 교차 검증
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import KFold
import numpy as np

iris = load_iris()
features = iris.data
label = iris.target
dt_clf = DecisionTreeClassifier(random_state=156)

kfold = KFold(n_splits=5)
cv_accuracy = []
print('붓꽃 데이터 세트 크기 : ', features.shape[0])

n_iter = 0

# KFold 객체의 split() 호출 -> 폴드별 학습, 검증 데이터의 행 인덱스를 array로 반환
for train_index, test_index in kfold.split(features):
  X_train, X_test = features[train_index], features[test_index]
  y_train, y_test = label[train_index], label[test_index]

  dt_clf.fit(X_train,y_train)
  pred = dt_clf.predict(X_test)
  n_iter += 1

  # 반복 시마다 정확도 측정
  accuracy = np.round(accuracy_score(y_test, pred), 4)
  train_size = X_train.shape[0]
  test_size = X_test.shape[0]
  print("\n#{0} 교차 검증 정확도 : {1}, 학습 데이터 크기 : {2}, 검증 데이터 크기 {3}".format(n_iter, accuracy, train_size, test_size))
  print("#{0} 검증 세트 인덱스 : {1}".format(n_iter, test_index))
  cv_accuracy.append(accuracy)
```

```
# 개별 iteration별 정확도를 합하여 평균 정확도 계산
print("\n## 평균 검증 정확도 : ", np.mean(cv_accuracy))
```

▼ Stratified K-Fold : 불균형한 분포도를 가진 레이블 데이터 집합을 위한 K-Fold 방식, 원본 데이터의 레이블 분포를 고려하여 데이터셋 분배

```
# Stratified K 폴드
iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_df['label'] = iris.target
iris_df['label'].value_counts()

from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=3)
n_iter = 0

for train_index, test_index in skf.split(iris_df, iris_df['label']):
  n_iter += 1
  label_train = iris_df['label'].iloc[train_index]
  label_test = iris_df['label'].iloc[test_index]
  print('## 교차검증 : {0}'.format(n_iter))
  print("학습 레이블 데이터 분포 : \n", label_train.value_counts())
  print("검증 레이블 데이터 분포 : \n", label_test.value_counts())
```

```
dt_clf = DecisionTreeClassifier(random_state=156)

skfold = StratifiedKFold(n_splits=3)
n_iter = 0
cv_accuracy = []

for train_index, test_index in skfold.split(features, label):
  X_train, X_test = features[train_index], features[test_index]
  y_train, y_test = label[train_index], label[test_index]

  dt_clf.fit(X_train, y_train)
  pred = dt_clf.predict(X_test)

  n_iter =+ 1
  accuracy = np.round(accuracy_score(y_test, pred),4)
  train_size = X_train.shape[0]
  test_size = X_test.shape[0]
  print("\n#{0} 교차 검증 정확도 : {1}, 학습 데이터 크기 : {2}, 검증 데이터 크기 {3}".format(n_iter, accuracy, train_size, test_size))
  print("#{0} 검증 세트 인덱스 : {1}".format(n_iter, test_index))
  cv_accuracy.append(accuracy)

  print("\n## 교차 검증별 정확도 : ", np.round(cv_accuracy, 4))
  print("## 평균 검증 정확도 : ", np.round(np.mean(cv_accuracy),4))
```

- 분류 → Stratified KFold / 회귀 → 결정값이 연속된 숫자값이기 때문에 Stratified KFold 지원 X

▼ cross_val_score(estimator, X, y=None, scoring=None, cv=None, n_jobs=1, verbose=0, fit_params=None, pre_dispatch='2*n_jobs') : 교차검증의 일련의 과정을 한번에 수행해주는 API

```
# cross_val_score()
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score , cross_validate
from sklearn.datasets import load_iris

iris_data = load_iris()
dt_clf = DecisionTreeClassifier(random_state=156)

data = iris_data.data
label = iris_data.target

# 성능 지표(scoring)는 정확도, 교차검증 세트 3개
scores = cross_val_score(dt_clf, data, label, scoring = 'accuracy', cv=3)
print('교차 검증별 정확도 : ', np.round(scores, 4))
print('평균 검증 정확도 : ', np.round(np.mean(scores), 4))
```

** estimator : Classifier 또는 Regressor / X : 피처 데이터 세트, y : 레이블 데이터 세트 / scoring : 예측 성능 평가 지표 기술 / cv : 교차 검증 폴드 수

GridSearchCV

- 하이퍼파라미터 : 값을 조정하여 알고리즘의 예측 성능 개선
- GridSearchCV : 교차 검증을 기반으로 최적의 파라미터 테스트
- estimator : classifier / regressor / pipeline
- param_grid : key + 리스트 딕셔너리 주어짐
- scoring : 예측 성능 측정 평가 방법 지정
- cv : 교차검증을 위해 분할되는 데이터셋의 개수 지정
- refit = True : 가장 최적의 하이퍼 파라미터를 찾아 해당 하이퍼 파라미터로 재학습 → 재학습 결과는 best_estimator_에 기록
- 파라미터 딕셔너리 형태로 설정 → GridSearchCV 파라미터 입력 → fit( ) → 결과 cv_results_로 기록 → 최고 성능 파람 값과 평과 결과 값 best_params_, best_score_에 기록

05 데이터 전처리

데이터 전처리

- 결손값(NaN, Null) 처리 → 대체, 드롭
- 문자열값 처리 → 숫자형으로 변환

레이블 인코딩 : 카테고리 피처 → 숫자값으로 변환

- LabelEncoder 클래스
- classes_ : 문자열값이 어떤 숫자로 인코딩됐는지 확인
- inverse_transform( ) : 디코딩
- 숫자의 크고 작음이 알고리즘에 적용되어 예측 성능이 떨어지는 문제 → 선형회귀 적용 X, 트리 계열 OK

원-핫 인코딩

- 피처 값의 유형에 따라 새로운 피처 추가 → 고유값에 해당하는 칼럼에만 1, 나머지는 0을 표시
- OneHotEncoder 클래스 → 2차원 데이터 입력 필요, 인코딩 결과 희소 행렬 형태 → toarray( ) → 밀집 행렬 변환 필요
- pd.get_dummies( )

피처 스케일링

- 서로 다른 변수의 값 범위를 일정 수준으로 맞추는 작업
- 표준화 : 피처 각각을 평균 0, 분산 1인 가우시안 정규 분포를 가진 값으로 변환 $x_i new = \frac{x_i - mean(x)}{stdev(x)}$
- 정규화 : 서로 다른 피처 크기 통일을 위해 크기를 변환 $x_i new = \frac{x_i - min(x)}{max(x) - min(x)}$
- 사이킷런 정규화 : 개별 벡터의 크기를 맞추기 위한 변환 $x_i new = \frac{x_i}{\sqrt{x_i^2 + y_i^2 + z_i^2}}$

StandardScaler

- 표준화 지원 클래스
- 서포트 벡터 머신, 선형 회귀, 로지스틱 회귀에서 중요

MinMaxScaler

- 데이터값을 0과 1사이의 범위값으로 변환 ** 음수가 있으면 -1~1로 변환
- 가우시안 분포가 아닌 경우 적용

❗ 스케일링 유의점 → 테스트 데이터셋으로는 다시 fit( )을 수행하지 않고 학습 데이터셋으로 fit( )을 수행한 결과로 transform( ) 변환을 적용해야 함 → 데이터셋 분리 전 먼저 전체 데이터셋에 스케일링 적용 후 분리하는 것이 바람직

07 정리

1. 사이킷런
2. 머신러닝 애플리케이션 : 전처리 → 데이터셋 분리 → 학습 데이터 기반 머신러닝 알고리즘 모델 학습 → 테스트 데이터 예측 → 모델 평가 수행
3. 데이터 전처리 : 데이터 클렌징(오류 보정, 결손값 처리), 인코딩(레이블, 원-핫), 스케일링/정규화 작업
4. 머신러닝 모델 : 학습 데이터셋 학습 → 테스트 데이터셋으로 평가
5. 교차검증 : KFold, Stratified KFold, cross_val_score( ), GridSearchCV

CHAPTER 03 평가

회귀 성능 평가 지표 → 실제값과 예측값의 오차 평균값에 기반

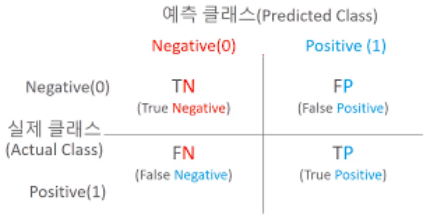
분류 성능 평가 지표 → 정확도, 오차행렬, 정밀도, 재현율, F1 스코어, ROC AUC ⇒ 이진 분류(긍정/부정 2개의 결과값)에서 더욱 중요한 지표

01 정확도(Accuracy)

- 실제 데이터에서 예측 데이터가 얼마나 같은지 판단하는 직관적 평가 지표
- 데이터 구성에 따라 성능 왜곡 가능성 → 정확도 수치 하나만 가지고 성능 평가 X
- 정확도 = 예측 결과가 동일한 데이터 건수 / 전체 예측 데이터 건수
- 불균형한 레이블 값 분포에서 ML 모델 성능을 판단할 때 부적합

02 오차행렬(Confusion matrix)

- 이진 분류의 예측 오류가 얼마인지 & 어떠한 유형의 예측 오류인지



- True : 실제값 = 예측값 / False : 실제값 ≠ 예측값
- Positive : 예측값 긍정(1) / Negative : 예측값 부정(0)
- confusion_matrix(실제 결과(y_test), 예측 결과(pred)) : ndarray 형태로 TN,FP,FN,FP 분포 보여줌
- 오차 행렬상에서 정확도 = (TN + TP) / (TN + FP + FN + TP) → Negative에 대한 예측 정확도만으로 분류 정확도가 높아지는 수치적 오류 발생

03 정밀도와 재현율

- 정밀도(양성 예측도) : 예측을 Positive로 한 대상 중 예측과 실제 값이 Positive로 일치한 데이터 비율

  → 실제 음성 데이터를 양성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우
- 재현율(민감도, TPR) : 실제 값이 Positive인 대상 중 예측과 실제 값이 Positive로 일치한 데이터 비율

  → 실제 양성 데이터를 음성으로 잘못 판단하게 되면 업무상 큰 영향이 발생하는 경우 중요 지표
- 정밀도 = TP / (FP + TP) → TP는 높이고, FP는 줄이고
- 재현율 = TP / (FN + TP) → TP는 높이고, FN은 줄이고

  ⇒ 정밀도와 재현율 모두 높은 수치를 얻는 것이 가장 좋은 평가
- precision_score( ) : 정밀도

정밀도/재현율 트레이드오프

- 정밀도와 재현율 중 어느 한 쪽을 강제로 높이면 다른 하나의 수치는 떨어지는 것
- 분류 알고리즘 : 예측 확률이 큰 레이블 값으로 예측
- predict_proba(테스트 데이터셋) : 예측 확률 결과를 반환 → 첫번째 칼럼은 Negative(0) 확률, 두번째 칼럼은 Positive(1) 확률
- Binarizer(threshold =) : threshold보다 같거나 작으면 0, 크면 1로 변환해 반환 → 임계값 조절하여 트레이드오프
- 임계값↓ → Positive 예측값↑ → 재현율↑, 정밀도↓
- precision_recall_curve(y_test, probas_pred) : 임계값에 따른 정밀도와 재현율 반환

❗ 임계값 변경은 두 수치를 상호 보완할 수 있는 수준에서 적용해야 함

- recall_score( ) : 재현율

04 F1 스코어

- 정밀도와 재현율을 결합한 지표
- 어느 한쪽으로 치우치지 않을 때 높은 값
- $F1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = 2 * \frac{precision * recall}{precision + recall}$
- f1_score( )

05 ROC 곡선과 AUC

- ROC 곡선 : FPR이 변할 때 TPR이 어떻게 변하는지를 나타내는 곡선
  - TPR(재현율, 민감도) : 실제값 양성이 정확히 예측돼야 하는 수준 → TPR = TP / (FN + TP
  - TNR(특이성) : 실제값 음성이 정확히 예측돼야 하는 수준 → TNR = TN / (FP + TN)
  - FPR = FP / (FP + TN) = 1 - TNR → 임계값이 1이면 FPR = 0, 임계값이 0이면 FPR = 1
- roc_curve( )
- AUC : ROC 곡선 밑의 면적을 구한 값, 1에 가까울수록 좋은 수치 → FPR이 작은 상태에서 얼마나 큰 TPR을 얻을 수 있는지

07 정리

1. 불균형 분포의 이진 분류에서는 정확도만으로 예측 성능 평가 불가
2. 오차행렬 : Negative와 Positive 값을 가지는 실제값과 예측값이 True, False에 따라 TN, FP, FN, TP로 매핑되는 4분면 행렬을 기반으로 예측 성능 평가
3. 정밀도와 재현율 : Positive 데이터셋의 예측 성능에 좀 더 초점을 맞춘 평가 지표 → 임계값 조정으로 정밀도/재현율 수치 향상 가능
4. F1 스코어 : 정밀도와 재현율 결합, 균형을 이룰 때 높은 지표값
5. ROC-AUC : 이진 분류 성능 평가를 위해 가장 많이 사용되는 지표