**02**

**데이터 핸들링 (1)**

1

**DataFrame 기본**

Pd.read\_csv(path, sep, sheet\_name, index\_col)

Pd.DataFrame(dict)

Pd.DataFrame(list or ndarray, columns=[ ])

Df.copy()

Df.index.values

Df.columns

Df.values

Df.reset\_index(inplace,drop)

2

**행, 열 선택/ 추가/ 삭제 (pandas)**

df[index:]

df[col]

df[‘new\_col’] = df[‘col1’] \* 100 + df[‘col2’]

df[‘new\_col’] = df[‘col1].apply(lambda x : ‘a’ if x > n else ‘b’)

df.drop(labels, inplace, axis)

df.dropna()

3

**조건에 따른 행, 열 선택 (loc,iloc)**

df.loc[:, :]

df.iloc[:, :]

df[df[‘col1’] >= n]

df.fillna(values, inplace=False)

4

**데이터 정렬 (sort\_value)**

df.sort\_values(by, ascending, inplace)

5

**데이터 결합 (concat, merge)**

Pd.concat([df1,df2], axis, join = ‘outer’ or ‘inner’, ignore\_index, keys)

Pd.merge(df1, df2, on=’col’, how=’outer’ or ‘inner’ or ‘right’, or ’left’)

Pd.merge(df1, df2, left\_on = ‘lkey’, right\_on = ‘rkey’)

Df1.join(df2, how)

**03**

**데이터 핸들링 (2)**

1

**데이터 요약 (groupby)**

Df.groupby(by=’col’).agg()

Df.groupby(by=’col1’)[‘col2’].agg([func1,func2])

Df.groupby([‘col1’,’col2’]).agg()

Df.groupby(by=’col1’).agg({‘Age’:’max’,’Fare’:’mean’})

2

**데이터 모양 변경 (pivot\_table)**

pd.pivot\_table(data, index, columns, values, aggfunc, fill\_value, margins, margins\_name)

df.pivot\_table(values, index, columns, aggfunc=’count’)

df.pivot(index, columns, values)

3

**문자열 데이터 변환하기 (str)**

str.split(sep, maxsplit)

‘-‘.join(list)

str.strip(value)

str.find(value)

str.count(value)

str.replace(old\_value, new\_value)

4

**데이터프레임에 함수 적용하기 (apply, map, lambda)**

Func = lambda x : x\*\*2

Map(lambda x : x\*\*2, list)

Df[‘new\_col’] = Df[‘col’].apply(lambda x : x\*\*2)

Df[‘new\_col’] = df[‘col’].apply(lambda x : ‘a’ if x > n else ‘b’)

5

날짜 데이터 변환하기 (datetime)

time= datetime.datetime(year,month,day,hour,minute,second)

time\_diff = day2 – day1

time\_diff.days

datetime.date.today()

datetime.datetime.now()

**04**

**데이터 전처리**

1

**데이터 분할 기법**

단순랜덤(shuffle=True) & 층화추출법(stratify)

From sklearn.model\_selection import train\_test\_split

X = df[df.columns.difference([‘target’])]

Y = df[‘target’]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size, test\_size, shuffle, stratify=df[‘col’])

Df.sample(n, replace)

From sklearn.model\_selection import Kfold

Kf = Kfold(n\_splits=4, shuffle=False)

Kf.get\_n\_splits(X)

For train\_index, test\_index in kf.split(X):

2

**데이터 표준화, 정규화**

From sklearn.preprocessing import MinMaxScaler, StandardScaler

Norm = MinMaxScaler()

Df\_\_norm = norm.fit\_transform(df)

Scaler = StandardScaler()

Df\_scaled = scaler.fit\_transform(df)

3

**결측치 처리하기**

Df.dropna(axis, subset=[‘col1’,’col2’], how = ‘any’ or ‘all’, inplace)

Df.fillna(values, method=[‘ffill’,’bfill’], inplace)

Df.fillna({‘A’:0, “B”:1, “C”:2})

Df.interpolate(method = ‘linear’)

imputer = sklearn.impute.SimpleImputer( np.nan ,'mean')

imputer.fit\_transform(df)

imputer = KNNImputer(n\_neighbots=5)

imputer.fit\_transform(df)

최빈값 : df[‘col’].mode()[0]

4

**범주형 변수 변환하기**

X1 = df[‘col1’].replace([1,2],[‘male’,’female’])

X1\_dum = pd.get\_dummies(x1)

Data = pd.concat([xy,x1\_dum],axis=1)

X\_ohe = pd.get\_dummies(df, columns=[ ])

5

**변수 축소 (PCA)**

From sklearn.decomposition import PCA

Pca = PCA(n\_components = 2)

Pca\_df = Pca.fit\_transform(X)

Pca.explained\_variance\_ration\_

6

**이상치 처리**

Sns.boxplot(df\_Data)

Q25, q75 = np.quantile(date[column], 0.25), np.quantile(data[column], 0.75)

Iqr = q75-q25

Cut\_off = 1.5 \* iqr

Lower\_line, upper\_line = q25 – cut\_off, q75 + cut\_off

Df[‘col’]\_clip = df[‘col’].clip(lower\_line, upper\_line)

7

**변수 선택, 특성 선택**

From sklearn.feature\_selection import f\_regression, SelectKBest

Selector = SelectKBest(score\_func = ‘f\_classif’ or ‘f\_regression’, k=20)

Selector.fit\_transform(X\_train, y\_train)

8

**데이터 불균형 해결**

from imblearn.over\_sampling import SMOTE

sm = SMOTE(sampling\_strategy = ‘auto’, random\_state = 0)

x\_smote, y\_smote = sm.fit\_resample(x\_train, y\_train)

from imblearn.over\_sampling import RandomOverSampler

ros = RandomOverSampler()

X\_ros, y\_ros = ros.fit\_resample(X,y)

**05**

**EDA (Exploratory Data Analysis) & 시각화**

1

**산점도**

Plt.scatter(x,y,s,c,marker,alpha, cmap) : 군집화 결과

2

**점, 선 그래프**

Plt.plot(x, y, fmt=’[color][linestyle][marker]’) : 시계열 변화

3

**막대그래프, 히스토그램**

Plt.bar(x,height,width,color,label, align={‘center’,’edge’}, tick\_label) : 비포 애프터 비교

Plt.barh(x,height,width,color,label)

Plt.hist(list, bins) : 어떤 항목의 개수를 시각화 할 때

Plt.pie(x, labels, autopct=’%.1f%%’, startangle, counterclock, shadow, explode) : 전체 분포 고려

4

**히트맵**

Import seaborn as 눈

Corr = data.corr()

sns.heatmap(data.corr(), cmap, vmin, vmax, annot, annot\_kws, linewidths, linecolor)

cmap = [‘PuBu’, ‘plasma’]

plt.show()

sns.pairplot(data, hue, diag\_kind, kind, markers, ,height, x\_vars, y\_vars, corner=True,

plot\_kws, diag\_kws)

5

**profiling**

Boxplot : sns.boxplot(x, y, hue, data, palette= ‘Set3’ or ‘vlag’)

Subplot : Fig, ax = plt.subplot(nrow, ncol, figsize=())

ax[0,0].plot()

**01**

**통계분석 - (1) 통계분석이란?**

1

**통계 분석의 개념**

데이터 분석을 수행할 때, 전체 데이터를 모두 활용하는 것은 수많은 시간과 비용이 들 수 있음으로, 모집단에서 일부 표본을 추출하여 분석에 활용한다.

통계란 특정집단을 대상으로 수행한 조사나 실험을 통해 나온 결과에 대한 요약된 형태의 표현이며, 통계분석은 이 통계를 가지고 분석을 수행하는 작업이다.

2

**모집단과 표본집단의 개념**

모집단 : 조사하고자 하는 대상 집단 전체

표본집단 : 조사하기 위해 추출한 모집단의 일부 원소

**02**

**통계분석 - (2) T-test**

1

**차이와 분산에 대한 이해**

Deviation = x – np.mean(x

Variation = sum(deviation\*\*2) # 편차 제곱의 평균

from scipy import stats

stats.mode(x) | np.var(x) | np.std(x) | np.quantitle(x,0.25) | stats.skew(x) | stats.kutosis()

2

**통계 검정의 개념 이해**

모집단의 특성에 대한 통계적 가설을 모집단으로부터 추출한 표본을 통해 검토하는 추론.모수적 방법은 모집단의 평균 혹은 분산을 추정하는 방법이고, 비모수적 방법은 정상성을 만족하지 않을 때 사용하는 방법이다.(중위값 추정.)

3

**t-test란**

두 집단의 평균을 통계적으로 비교하기 위해 사용되는 검정방법으로, 귀무가설은 ‘두 집단의 평균이 같다’이다.

4

**t-test 실습**

from scipy import stats

**정상성 & 등분산성 검정**

stats.shipiro(x)

stats.kstest(x, alternative = ‘two-sided’, cdf =’norm’)

stats.normaltest(x)

stats.levene(a, b)

**일표본 t-검정**

stats.ttest\_1samp(x, popmean, alternative=’two-sided’)

stats.wilcoxon(x = data – popmean, y=None, zero\_method = ‘wilcox’, correction=False)

**대응표본 t-검정**

stats.ttest\_rel(a, b, alternative=’two-sided’)

stats.wilcoxon(x, y, zero\_method = ‘wilcox’, correction=False )

**독립표본 t-검정**

stats.ttest\_ind(a, b, alternative=’two-sided’, equal\_var = True)

stats.ttest\_ind(a, b, alternative=’two-sided’, equal\_var = False)

stats.mannwhitneyu(a, b, alternative=’two-sided’)

**03**

**통계분석 - (3) 분산분석**

1

**ANOVA test란?**

두 개 이상의 다수 집단 간 평균을 비교하는 통계분석 방법.

2

**F-value에 대한 이해**

F검정통계량은 두 그룹의 분산에 대한 비율이다.

3

**일원 배치 분산 분석 개념**

분산분석은 두 개 이상의 집단에서 그룹 평균 간 차이를 그룹 내 변동에 비교하여 살펴보는 분석방법으로 두 개 이상 집단들의 평균 간 차이에 대한 통계적 유의성을 검증하는 분석이다.

4

**일원 배치 분산분석 실습**

Stats.shapiro(a)

Stats.levene(a,b,c)

Stats.chi2\_contingency(crosstab)

Stats.f\_oneway(a,b,c)

Stats.kruskal(a,b,c)

import statsmodels.formula.api as smf

import statsmodels.api as sm

reg = Smf.ols(formula = ‘x ~ y’, data=df\_1way)

reg\_fit = reg.fit()

anova\_table = sm.stats.anova\_lm(reg\_fit, typ=2)

print(anova\_table)

from statsmodel.stats.multicomp import pairwise\_tukeyhsd

tukey = pairwise\_tukeyhsd(endog = x, groups = y)

print(tukey)

print(tukey.plot\_simultaneous())

5

**이원 배치 분산 분석 개념**

분산분석에서 반응값에 대해 두 개의 범주형 변수 A, B의 영향을 알아보기 위해 사용되는 검증 방법이다.

6

**이원 배치 분산분석 실습**

Stats.shapiro(x)

Stats.levene(a,b,c,d)

import statsmodels.formula.api as smf

import statsmodels.api as sm

Reg = smf.ols(formula = ‘y ~ x1\*x2’, data=df\_2way)

Reg\_fit = reg.fit()

Anova\_table = sm.stats.anova\_lm(reg\_fit, typ=2)

Print(anova\_table)

**04**

**통계분석 - (4) 교차분석**

1

**교차분석 개념**

교차분석이란 명목척도 혹은 순서척도 등 한 두 개의 범주형 자료들 간의 상호 연관성을 알아볼 때 주로 사용되는 방법이다.

2

**교차분석 실습**

From scipy import stats

Pd.crosstab(index, columns, values, aggfunc)

Stats.chisquare(x, f\_exp=[0.8,0.2])

Stat, p\_value, \_, \_ = Stats.chi2\_contingency(crosstab)

**05**

**통계분석 - (5) 선형 회귀분석**

1

**선형 회귀 모형의 개념**

하나 혹은 그 이상의 독립변수가 종속변수에 미치는 영향을 추정하여 식으로 표현하는 통계기법

2

**단순선형 회귀분석**

하나의 독립변수가 종속변수에 미치는 영향을 추정하는 통계기법.

결측치, 이상치 처리 -> 상관성검증 -> 회귀분석시행 -> 회귀분석결과 -> 회귀진단

Import statsmodels.formula.api.smf

Lm = smf.ols(formula = ‘y ~ x’, data)

Result = lm.fit()

Print(Result.summary())

3

**다중 선형 회귀분석**

두 개 이상의 독립변수가 종속변수에 미치는 영향을 추정하는 통계기법.

결측치,이상치 처리 -> 상관성검증 -> 회귀분석 -> 회귀분석결과도출 -> 회귀진단

4

**다중 선형 회귀분석 실습**

Import statsmodels.formula.api.smf

Lm = smf.ols(formula = ‘y ~ ‘ + ‘+’.join(df.columns.difference([‘y’])), data=df)

Result = lm.fit()

Print(result.summary())

5

**다중 공선성 진단 방법**

# 분산팽창요인(VIF) 출력 함수 정의

def VIF(model):

for i,col in enumerate(model.exog\_names):

r = variance\_inflation\_factor(model.exog, i)

print(col,r)

VIF(lm) :

10이상일 경우, 다중공선성이 심한 특성.

6

**변수 선택법**

단계적 선택법 : 전진선택법, 후진제거법, 단계적선택법

**06**

**통계분석 - (6) 로지스틱 회귀분석**

1

**로지스틱 회귀분석 개념**

로지스틱 회귀분석은 반응변수가 범주형인 경우에 적용되는 회귀모형으로, 이 방법은 새로운 설명 변수의 값이 주어질 때 반응변수의 각 범주에 속할 확률을 추정하여 기준치에 따라 분류를 수행하는 선형모델의 일종이니다.

2

**로지스틱 회귀분석 해석 방법**

로지스틱 회귀가 선형 회귀와 다른 점은 학습을 통해 선형 함수의 최적선을 찾는 것이 아니라 시그모이드(Sigmoid) 함수 최적선을 찾고, 해당 함수의 반환값으로 확률을 구해 분류를 결정한다는 것이다.

3

**로지스틱 회귀분석 실습**

From sklearn.linear\_model import Logistic Regression

From sklearn.preprocessing import StandardScaler

Scaler = StandardScaler()

X\_scaled = Scaler.fit\_transform()

Train\_test\_split(X, y, test\_size, shuffle, stratify, random\_state)

lr\_clf = LogisticRegression(penalty = {‘l1’, ‘l2’, ‘elasticnet’}, C=[0.01,0.1,1,5,10,20], solver=[‘lbfgs’, ‘saga’], multi\_class=’auto’ )

lr\_clf.fit(X\_train, y\_train)

lr\_clf.coef\_, lr\_clf.intercept\_

4

**다항 로지스틱 회귀분석 실습**

lr\_clf = LogisticRegression(penalty = {‘l1’, ‘l2’, ‘elasticnet’}, C=[0.01,0.1,1,5,10,20], solver=[‘lbfgs’, ‘saga’], multi\_class=’auto’ )

lr\_clf.fit(X\_train, y\_train)

**07**

**통계분석 - (7) 군집분석**

1

**군집분석 개념**

군집 분석(Clustering Analysis)은 각 개체의 유사성을 측정하여 유사성이 높은 대상 집단끼리 분류하고, 군집에 속한 객체들의 유사성과 서로 다른 군집에 속한 개체간의 상이성을 규명하는 다변량 분석 기법이다. 군집 분석에 이용되는 자료는 **별도의 반응변수가 요구되지 않으며**, 오로지 개체간의 유사성(Similarity)에만 기초하여 군집을 형성한다.

2

**계층적 군집분석**

계층적 군집분석은 n 개의 군집으로 시작해 점차 군집의 개수를 줄여 나가는 방법이다. 계층적 군집을 형성하는 방법에는 **합병형 방법(Agglomerative)과 분리형 방법(Divisive)**이 있다.

from scipy.cluster.hierarchy import dendrogram, linkage

from sklearn.cluster import AgglomerativeClustering

linked = linkage(data, method=[‘simple’, ‘complete’, ‘average’, ‘ward’])

dendrogram(linked, orientation = ‘top‘ , distance\_sort=’descending’, show\_leaf\_counts=True)

cluster = AgglomerativeClustering(n\_clusters=5, affinity=’euclidean’, linkage=’ward’)

labels = cluster.fit\_predict(data)

3

**비계층적 군집분석**

비계층적 군집분석은 n개의 개체를 k개의 군집으로 나눌 수 있는 모든 가능한 방법을 점검해 최적화한 군집을 형성하는 것이다. 대표적으로 K-Means Clustering 군집분석이 있다.

from sklearn.cluster import KMeans

kmeans = KMeans(n\_clusters=3, init='k-means++', max\_iter=300, random\_state=0, )

kmeans.fit(X\_scaled)

print(kmeans.labels\_)

print(kmeans.cluster\_centers\_)

print(kmeans.inertia\_)

4

**혼합분포 군집분석**

혼합 분포 군집은 모형 기반의 군집 방법이며, 데이터가 k개의 모수적 모형의 가중합으로 표현되는 모집단 모형으로부터 나왔다는 가정하에서 모수와 함께 가중치를 자료로부터 추정하는 방법을 사용한다

from sklearn.cluster import DBSCAN

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=3, random\_state=42, n\_init=10)

gmm.fit(X\_scaled)

dbscan = DBSCAN(eps = 0.6, min\_samples=8, metric='euclidean')

dbscan\_labels = dbscan.fit\_predict(X\_scaled)

5

**SOM**

From sklearn\_som.som import SOM

Iris\_som = SOM(m=3, n=1, dim=2)

Iris\_som.fit(iris.data)

Iris\_som.predict(iris.data)

**08**

**통계분석 - (8) 연관분석**

1

**연관분석 Run-test**

모집단으로부터 추출된 표본집단이 무작위로 추출되었는지를 검정하는 작업

귀무가설 : 추출된 표본집단이 무작위로 추출되었다.

Import statsmodel.sandbox.stats.runs import Runs

Runs(x).runs\_test()

2

**연관분석 개념**

연관분석은 기업의 데이터베이스에서 상품의 구매, 서비스 등 일련의 거래 또는 사건들 간의 규칙을 발견하여 IF-THEN의 구조로 분석 결과의 연관성을 파악하는 데이터마이닝 방법론이다.

3

**연관분석 실습**

From mlxtend.preprocessing import TransactionEncoder

From mlxtend.frequent\_patterns import apriori, association\_rules

Te\_ary = Pd.get\_dummies(x)

Or

Te = TransactionEncoder()

Te\_ary = Te.fit(x).transform(x)

Runs(x).runs\_test()

Itemset = apriori(df, min\_support, max\_len, use\_colnames = True)

Association\_rules(itemset, metric = ‘confidence’ or ‘lift’, min\_threshold=0.5)

**09**

**통계분석 - (9) 시계열분석**

1

**시계열분석 개념**

과거 데이터의 패턴을 분석하여 미래의 값을 예측하는 방법으로, 과거의 패턴이 미래에도 지속된다는 데이터의 안정성이 기본적인 가정으로 필요하다.

2

**시계열 분해**

시계열 분해법이란 시계열 데이터를 추세/순환/계절/불규칙 요소로 분해하는 기법이다.

from statsmodels.tsa.seasonal import seasonal\_decompose

result = seasonal\_decompose(ts['MeanTemp'], model='additive',freq=7)

3

**정상성 확인 및 검정**

ARIMA는 AR과 MA 모형을 합친 모형으로 시계열 데이터의 정상성을 가정하고 있다.

여기서 정상성이란 평균, 분산이 시간에 따라 일정한 성질을 말하는데, 예로 추세나 계절성이 있는 시계열은 정상 시계열이 아니다.

따라서 정상성을 나타내지 않는 데이터를 정상 시계열로 변환하는 방법은 아래와 같다.

1. 변동폭이 일정하지 않은 경우 : 로그변환 np.log1p(x)
2. 추세, 계절성이 존재하는 경우 : 차분 data.diff(n)

from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf

from statsmodels.tsa.stattools import adfuller

fig = plot\_acf(ts, lags=20)

result = adfuller(ts)

print('ADF Statistic : %f' % result[0])

print('p-value : %f' % result[1])

ts\_diff1 = ts- ts.shift() # ts.diff()

4

**ARIMA**

ARIMA(p,d,q) 모형은 d차 차분한 데이터에 AR(p), MA(q) 이 두 모형을 합친 것이다.

from statsmodels.tsa.arima\_model import ARIMA

model = ARIMA(ts, order=(2,1,2))

model\_fit = model.fit(disp=0)

forecast = model\_fit.predict(start=start\_index, end=end\_index, typ='levels')

5

SARIMA 실습

6

Auto-ARIMA 실습

**11**

**머신러닝 - (2) 선형회귀분석**

2

**LinearRegression 실습**

From sklearn.linear\_model import LinearRegression

From sklearn.preprocessing import StandardScaler, MinMaxScaler

Scaler = StandardScaler()

Scaler.fit\_transform(X)

Minmax = MinMaxScaler()

Minmax.fit\_transform(X)

Np.log1p(X)

Lr = LinearRegression()

Lr\_.coef\_, lr.intercept\_

imp\_series = pd.Series(index = feature\_names, data = lr.coef\_)

top5\_imp = np.abs(imp\_series).sort\_values(ascending=False).head()

top5\_imp

sns.barplot(x = top5\_imp.values, y = top5\_imp.index)

3

**Lidge, Rasso, ElasticNet 실습**

From sklearn.linear\_model import Ridge, Lasso, ElasticNet

ridge = Ridge()

lasso = Lasso()

elnet = ElasticNet()

**12**

**머신러닝 - (3) 의사결정나무**

1

**의사결정나무 개념**

의사결정나무는 분류함수를 의사결정 규칙으로 이뤄진 나무 모양으로 그리는 방법입니다. 계산 결과가 의사결정나무에 직접 나타나기 떄문에 해석이 간편하며, if/else 기반으로 최대한 많은 데이터 세트가 해당 분류에 속할 수 있도록 결정 노드의 규칙이 정해지는 방법입니다.

2

**의사결정나무 (분류)**

From sklearn.tree import DecisionTreeClassifier, plot\_tree, export\_text

Train\_test\_split(X, y, test\_size, shuffle, stratify, random\_state)

Dt\_clf = DecisonTreeClassifier(max\_depth, min\_samples\_leaf, criterion = [‘gini’,’entropy’], max\_features )

Dt\_clf.fit(X\_train, y\_train)

Dt\_clf.get\_params()

Dt\_clf.feature\_importances\_

Export\_text(dt\_clf, feature\_names)

3

**의사결정나무 (회귀)**

from sklearn.tree import DecisionTreeRegressor

Train\_test\_split(X, y, test\_size, shuffle, stratify, random\_state)

Dt\_reg = DecisionTreeRegressor(max\_depth, min\_samples\_leaf, max\_features , criterion = [‘mse’, ‘mae’])

Dt\_reg.fit(X\_train, y\_train)

Dt\_reg.get\_params()

Dt\_reg.feature\_importances\_

Export\_text(dt\_reg, feature\_names)

4

**의사결정나무 실습(시각화)**

plt.title('feature imporances')

sns.barplot(x = best.feature\_importances\_, y =feature\_columns)

plt.show()

fig = plt.figure(figsize=(20,20))

plot\_tree(best, filled=True, fontsize=15)

plt.show()

sns.regplot(x=y\_train, y=y\_valid\_predict)

plt.show()

sns.jointplot(x=y\_train, y=y\_valid\_predict, kind='hex')

plt.show()

sns.residplot(x=y\_train, y=y\_valid\_predict)

plt.show()

sns.displot(df\_y, kde=True, height=2, aspect=5)

plt.show()

**13**

**머신러닝 - (4) 앙상블기법**

1

**앙상블 개념**

앙상블 학습(Ensemble)을 통한 분류는 여러 개의 분류기(Classifier)를 생성하고 그 예측을 결합함으로써 보다 정확한 최종 예측을 도출하는 기법을 말합니다. 대부분의 정형 데이터 분류 시 앙상블 기반 알고리즘이 뛰어난 성능을 나타내고 있습니다. **(RandomForest, Gradient Boost, XGBoost)**

앙상블 학습의 유형은 전통적으로 보팅(Voting), 배깅(Bagging), 부스팅(Boosting)의 세 가지로 나눌 수 있으며, 이외에도 스태킹을 포함한 다양한 앙상블 방법이 있습니다.

2

**배깅,보팅, 부스팅 개념**

배깅 방법(Bagging Classifier)은 동일한 분류기를 데이터 샘플링을 서로 다르게 가져가면서 학습을 수행하고, 해당 분류기들의 결괏값 투표를 통해 최종 예측 결과를 결정하는 방법입니다.

보팅 방법(Voting Classifier)은 서로 다른 여러 개의 분류기를 학습한 후 투표를 통해 최종 예측결과를 결정하는 방식입니다. 보팅 방법에는 하드 보팅 방법과 소프트 보팅 방법 이렇게 두 가지가 있습니다. 하드 보팅은 다수의 분류기가 예측한 결괏값 중 다수의 분류기가 결정한 결괏값으로 최종 결과를 선정하는 방식이고, 소프트 보팅은 분류기들의 레이블 당 결정 확률의 평균을 구하여 가장 높은 확률의 레이블을 선택하는 방식입니다. 일반적으로 소프트 보팅 방식이 많이 사용됩니다.

3

**배깅, 보팅(회귀)**

From sklearn.ensemble import BagginRegressor, VotingRegressor

4

**배깅, 보팅(분류)**

From sklearn.ensemble import BaggingClassifier, VotingClassifier

From sklearn.neighbors import KNeighborsClassifier

From sklearn.linear\_model import LogisticRegression

Base\_estimator = KNeigborsClassifier(n\_neighbors=8)

Bagging\_clf = BaggingClassifier(base\_estimator = base\_estimator, n\_estimators = 10, oob\_score=True, random\_state=42)

Bagging\_clf.fit(X\_train, y\_train)

Bagging\_clf.oob\_score\_

Lr\_Clf = LogisticRegression()

Knn\_clf = KNeigborsClassifier(n\_neighbors=8)

Voting\_Clf = VotingClassifier(estimators = [ (‘LR’,lr\_clf), (‘KNN’,knn\_clf) ], voting = ‘soft’)

Voting\_clf.fit(X\_train, y\_train)

Voting\_clf.estimators\_

5

**랜덤포래스트(회귀)**

From sklearn.ensemble import RandomForestRegressor

6

**랜덤포래스트(분류)**

Rf\_clf = RandomForestClassifier(n\_estimators = 100, max\_depth, min\_samples\_leaf, min\_samples\_split)

7

**부스팅(회귀)**

from sklearn.ensemble import GradientBoostingClassifier

from xgboost import XGBRegressor

from lightgbm import LGBMRegressor

8

**부스팅(분류)**

From sklearn.ensemble import GradientBoostingClassifier

From xgboost import XGBClassifier, plot\_importance

From lightgbm import LGBMClassifier, plot\_importance

gbm\_clf = GradientBoostingClassifier(max\_depth = 6, min\_samples\_leaf = 8, min\_samples\_split = 8,

learning\_rate=0.05, n\_estimators=100)

XGB\_clf = XGBClassifier(n\_estimators=400, learning\_rate=0.1, max\_depth = 3, subsample=0.8)

evals = [(X\_test,y\_test)]

XGB\_clf.fit(X\_train, y\_train, early\_stopping\_rounds=100, eval\_metric='logloss', eval\_set=evals,

verbose=True)

lgbm\_clf = LGBMClassifier(n\_estimators=400, learning\_rate=0.1, subsample=0.8, max\_depth=15,

min\_child\_samples=25)

evals = [(X\_test, y\_test)]

lgbm\_clf.fit(X\_train, y\_train, early\_stopping\_rounds=50, eval\_metric='log\_loss', eval\_set = evals)

**14**

**머신러닝 - (5) 나이브베이즈 & KNN**

1

**나이브 베이즈 개념**

나이브 베이즈 분류는 데이터에서 변수들에 대한 조건부 독립을 가정하는 알고리즘으로 클래스에 대한 사전 정보와 데이터로보터 추출된 정보를 결합하고, **베이즈 정리**를 이용하여 어떤 데이터가 특정 클래스에 속하는지를 분류하는 알고리즘이다.

2

**나이브 베이즈 분류기 실습**

From sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

3

**KNN 분류기 개념 & 실습**

K-NN은 어떤 범주로 나누어져 있는 데이터 셋이 있을 때, 새로운 데이터가 추가된다면 이를 어떤 범주로 분류할 것인지를 결정할 때 사용할 수 있는 분류 알고리즘이다. K-NN은 새로운 데이터의 클래스를 해당 데이터와 **거리 상 가장 가까운 k개 데이터들의 클래스(범주)**로 결정한다.

param\_grid = {

'n\_neighbors':[3,5,7,9,12,15],

'weights':['uniform','distance']

}

knn = KNeighborsClassifier()

**15**

**머신러닝 - (6) SVM & 인공신경망**

1

**Support Vector Machine 개념**

서포트 벡터 머신 알고리즘은 주어진 데이터 집합을 바탕으로 하여 새로운 데이터가 어떤 범주에 속할 것인지를 판단하는 비확률적 이진 선형 분류 모델을 생성한다. Svm의 분류 모델은 데이터가 사상된 공간에서 경계로 표현되며, 공간상에 존재하는 여러 경계 중 가장 큰폭을 가진 경계를 찾습니다. 반대로 svm 회귀 모델은 일정한 마진 오류 안에서 두 클래스 간의 도로 폭이 가능한 한 최대가 되도록 하는 대신, 제한된 마진 오류 안에서 도로 안에 가능한 많은 샘플이 들어가도록 학습합니다.

2

**SVC 실습**

From sklearn.svm import SVC

Scaler = StandardScaler()

Scaler.fit\_transform(X)

Svm\_clf = svc(kernel=[‘linear’,’poly’,’rbf’], C=1, gamma = 5)

3

**SVR 실습**

From sklearn.sv import SVR

Svm\_reg=svr(C=100, epsilon=0.1, kernel='poly', degree=2)

4

**인공신경망(분류)**

From sklearn.neural\_network import MLPClassifier

clf = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)

5

**인공신경망(회귀)**

From sklearn.neural\_network import MLPRegressor

Reg = MLPRegressor(max\_iter =500)

**16**

**머신러닝 - (7) 불균형 데이터 처리**

**1**

**언더 샘플링 기법**

From imblearn.under\_sampling import RandomUnderSampler, EditedNearestNeighbours

rus = RandomUnderSampler()

X\_rus, y\_rus = rus.fit\_resample(X,y)

**2**

**오버 샘플링 기법**

from imblearn.over\_sampling import SMOTE, RandomOverSampler

sm = SMOTE(sampling\_strategy = ‘auto’, random\_state = 0)

x\_smote, y\_smote = sm.fit\_resample(x\_train, y\_train)

ros = RandomOverSampler()

X\_ros, y\_ros = ros.fit\_resample(X,y)