용해탱크 제조 데이터를 이용한 용해 품질 분류기 모델

데이터사이언스학과 허준봉

CONTENTS

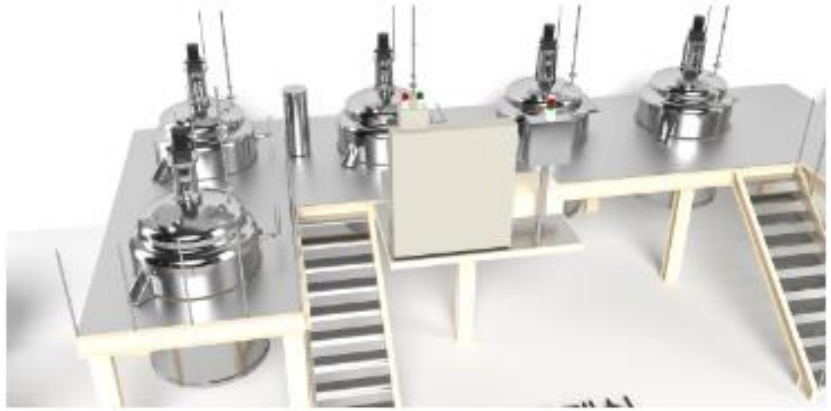
EXPLANATION and EXPLORATION OF MELTING TANK DATA

용해탱크 제조 데이터셋에 대한 설명 및 탐색

1. 용해 제조데이터에 대한 설명 및 탐색

용해 제조 데이터란?

- 용해탱크 제조 데이터를 기반으로 해당 용해 상태가 불량인지 정상인지 식별하기 위해 만들어진 데이터
- 용해탱크 제조 데이터셋은 용해탱크의 PLC를 통해 수집하였으며 총 데이터셋은 총 835200개입니다.



용해탱크

DATASET

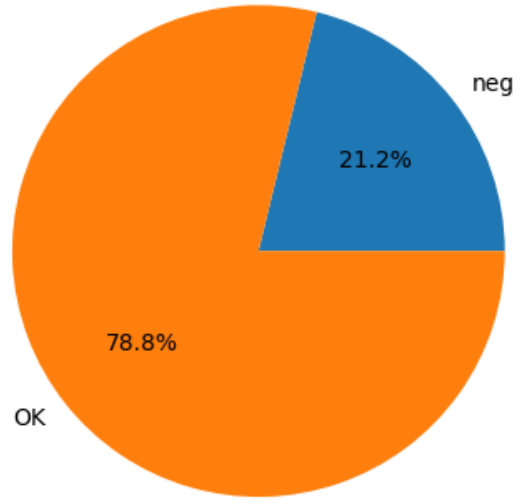
변수명	설명	데이터 타입
STD_DT	날짜, 시간 (YYYY-MM-DD HH:MM:SS)	Object
NUM	인덱스	int64
MELT_TEMP	용해 온도	int64
MOTORSPEED	용해 교반속도	Int64
MELT_WEIGHT	용해탱크 내용량(중량)	Int64
INSP	생산품의 수분함유량(%)	float64
TAG	불량여부	Object

1. 용해 제조데이터에 대한 설명 및 탐색

클래스 비율 확인 및 각 변수의 상관계수 확인

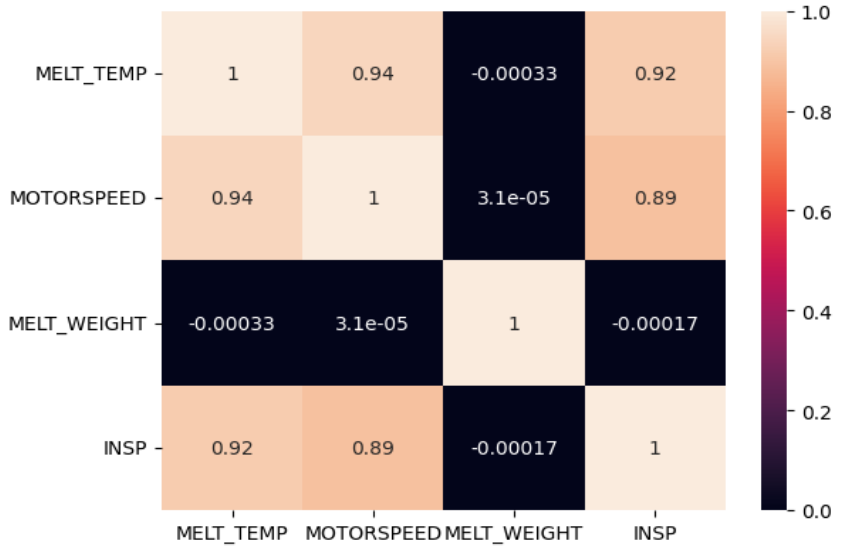
```
# 윤활유의 품질이 불량인 비율
neg_ratio=y_train['TAG'].value_counts()[0]/(y_train['TAG'].value_counts()[0]+y_train['TAG'].value_counts()[1])*100
# 윤활유의 품질이 정상인 비율
ok_ratio=y_train['TAG'].value_counts()[1]/(y_train['TAG'].value_counts()[0]+y_train['TAG'].value_counts()[1])*100
ratio=[neg_ratio, ok_ratio]
labels=['neg','OK']

plt.pie(ratio,labels=labels,autopct='%.1f%%')
plt.show()
```

✓ 0.1s



PieChart

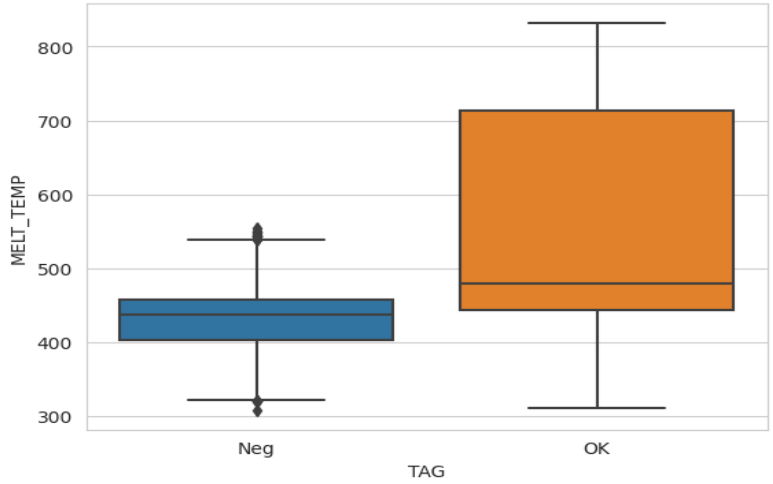
```
# Relationship analysis

import seaborn as sns
correlation=data.corr()
sns.heatmap(correlation, xticklabels=correlation.columns,
            yticklabels=correlation.columns,annot=True)
```

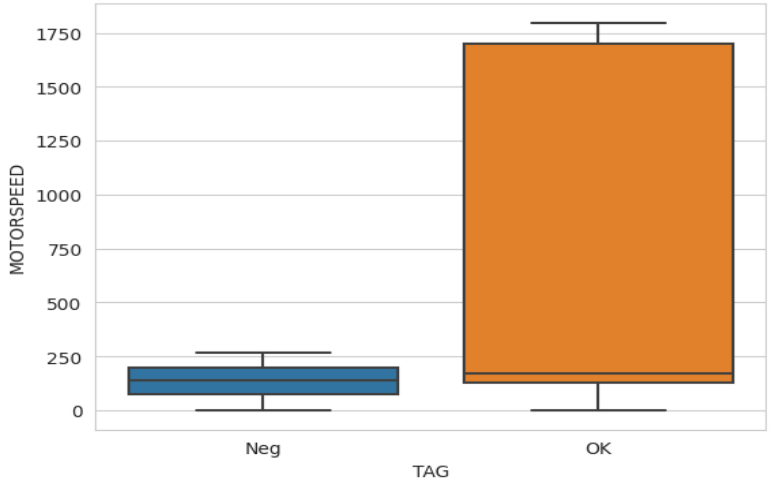


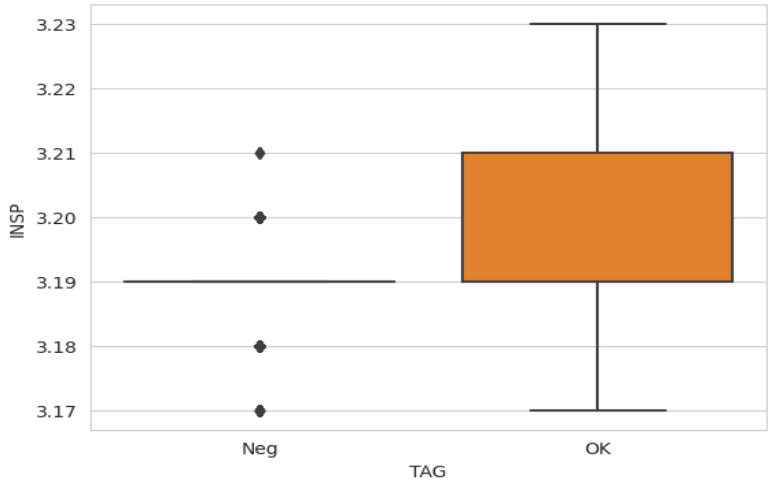
HeatMap

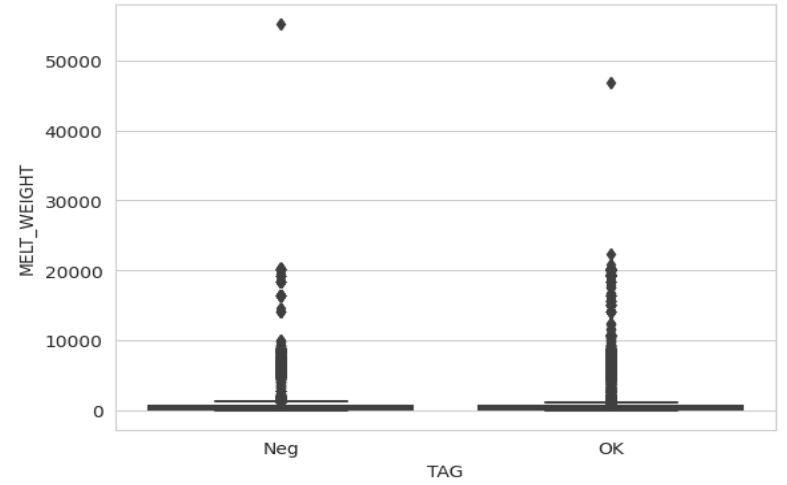
1. 용해 제조데이터에 대한 설명 및 탐색

Boxplot 시각화









DATA PROCESSING

데이터 전처리

2. 데이터 전처리

결측치 확인

```
# Check null data

data.isnull().sum()
```

```
STD_DT          0
NUM             0
MELT_TEMP       0
MOTORSPEED      0
MELT_WEIGHT     0
INSP            0
TAG             0
dtype: int64
```

데이터 표준화

```
#Scale data

from sklearn.preprocessing import StandardScaler
scale = StandardScaler()
X_transform = scale.fit_transform(X)
```

```
X_transform
```

```
array([[-0.43119289,  1.15778156,  0.99250729],
       [-1.08260274,  1.06078885, -0.36134605],
       [ 0.22021697,  1.05638009,  0.38810848],
       ...,
       [ 0.11736278, -0.23097769,  0.41228443],
```

- 트리계열 모델 구축 때는 변수 표준화 적용하지 않음
- 로지스틱 회귀모델 구축할 때는 변수 표준화 적용함

목적 데이터를 범주형 데이터로 변환

```
from sklearn.preprocessing import LabelEncoder

le=LabelEncoder()
y=le.fit_transform(y)
print(y)
```

```
[1 1 1 ... 1 1 1]
```

2. 데이터 전처리

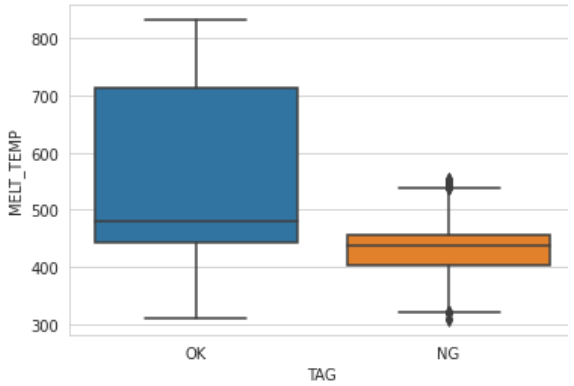
데이터에 음수값이 있는지 확인하기

```
data.describe()
```

	NUM	MELT_TEMP	MOTORSPEED	MELT_WEIGHT	INSP
count	835200.000000	835200.000000	835200.000000	835200.000000	835200.000000
mean	417599.500000	509.200623	459.782865	582.962125	3.194853
std	241101.616751	128.277519	639.436413	1217.604433	0.011822
min	0.000000	308.000000	0.000000	0.000000	3.170000
25%	208799.750000	430.000000	119.000000	186.000000	3.190000
50%	417599.500000	469.000000	168.000000	383.000000	3.190000
75%	626399.250000	502.000000	218.000000	583.000000	3.200000
max	835199.000000	832.000000	1804.000000	55252.000000	3.230000

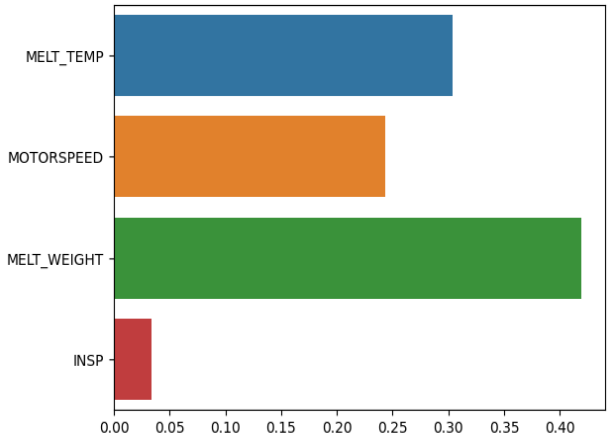
요약통계량

이상치 제거하기



Boxplot

Feature Importance



INSP 변수 제외하고
변수중요도가 높은 위
세 개의 변수만 선택

2. 데이터 전처리

교차 검증은 Overfitting을 줄이기 위한 작업

전체 데이터를 Training data와 Test data로 나눈 다음
Training data를 Training data와 Validation data로
나누었습니다.

5개의 데이터 폴드 세트를 만들어서
학습과 검증을 위한 데이터 세트로 변경하면서
5번 평가를 수행한 뒤, 이 5개의 평가를 평균한 결과를
가지고 예측 성능을 평가함.

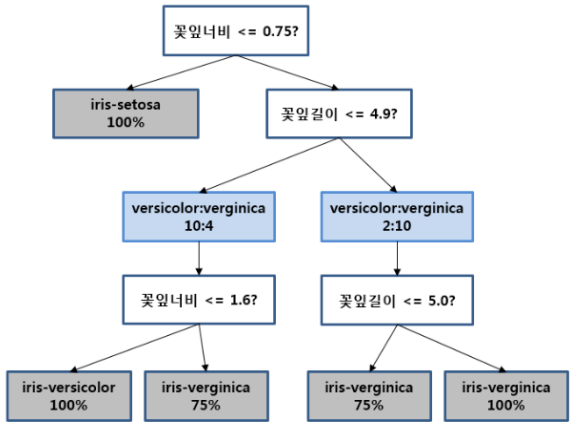
INTRODUCTION DATA ANALYSIS TECHNIQUES

데이터 분석기법 소개

3. 데이터 분석기법 소개

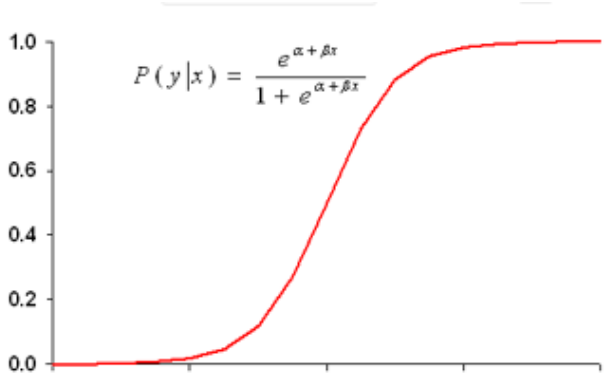
Decision Tree

- 데이터를 분석하여 이들 사이에 존재하는 패턴을 예측 가능한 규칙들의 조합으로 나타내며, 그 모양이 '나무'와 같다.



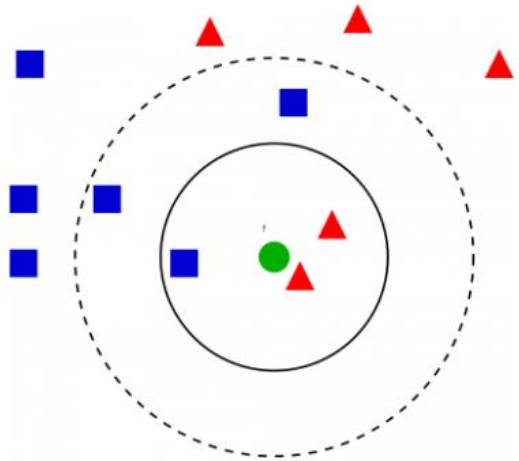
Logistic Regression

- 종속변수가 범주형이면서 0 또는 1인 경우 사용하는 회귀분석



KNN

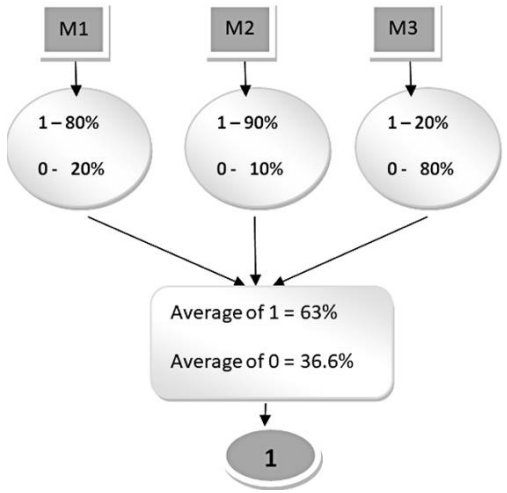
- k개의 최근접 이웃 사이에서 가장 공통적인 항목에 할당되는 객체로 과반수 의결에 의해 분류된다



3. 데이터 분석기법 소개

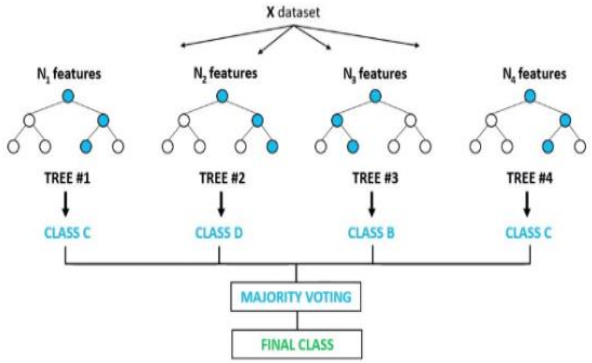
Soft Voting Classifier

- 여러 분류기들의 예측 결과 값에 대한 확률을 평균하여 최종 결과를 내리는 모델



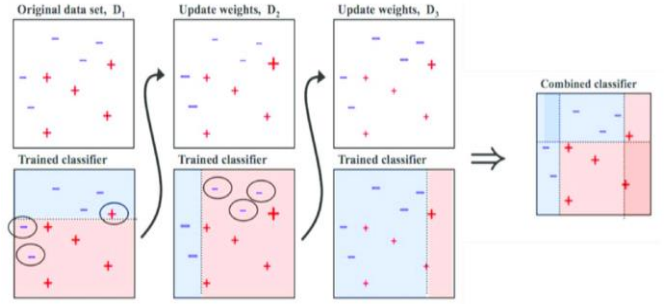
Random Forest

- Bagging을 사용하여 의사결정나무 여러 개를 훈련시켜 결과를 예측하는 모형



Boosting

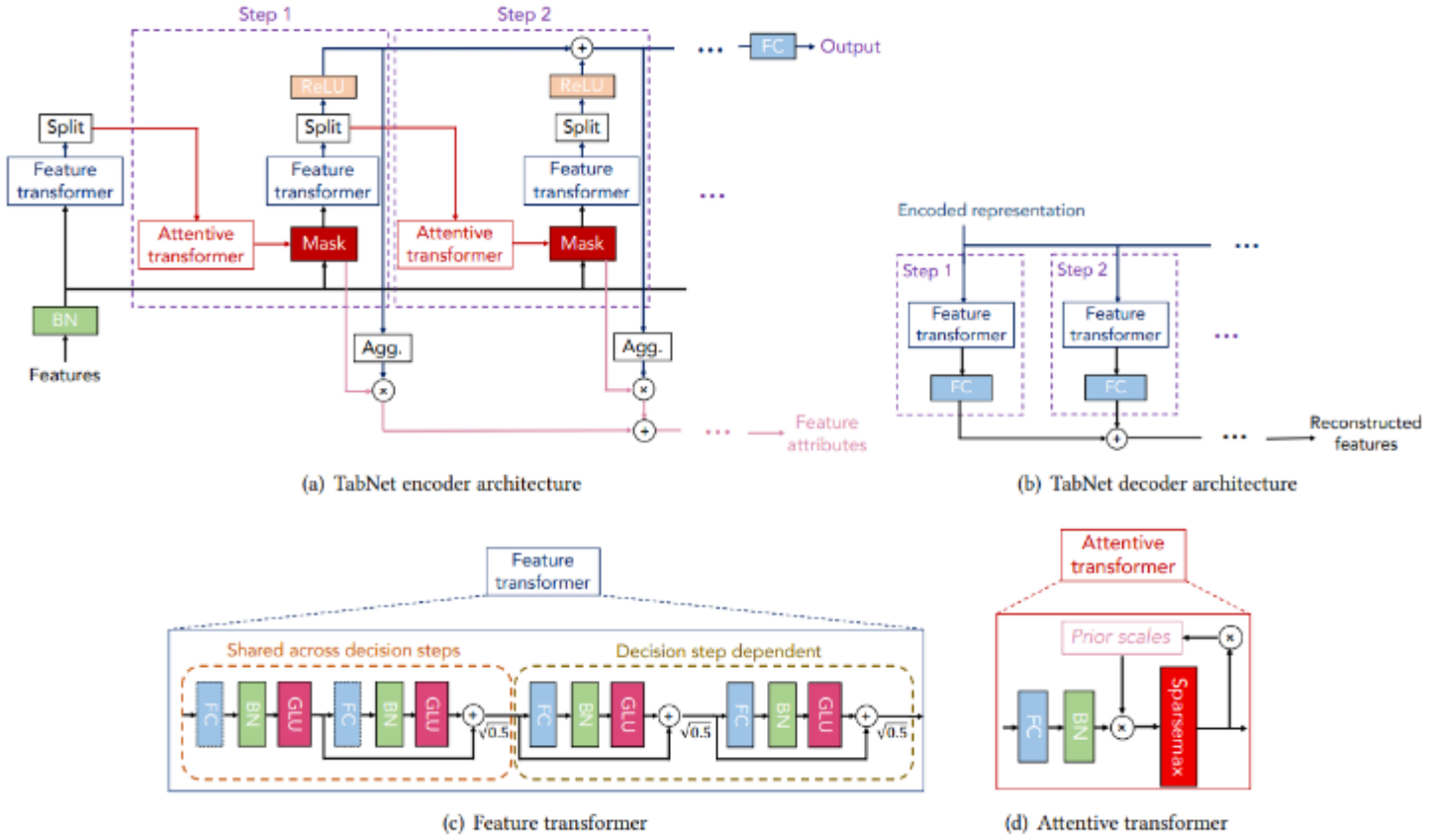
- 가중치를 활용하여 약분류기를 강분류기로 만드는 모형



3. 데이터 분석기법 소개

TabNet

- 변수 선택을 위한 마스크가 네트워크에 추가된 모델 해석이 용이한 딥러닝 모델
- 변수를 선택하는 attentive transformer 블록을 반복 통과할 때 마다 이미 선택한 변수가 중복 선택되지 않도록 sparkmax 함수와 prior scales를 활용함



(a) TabNet encoder architecture

(b) TabNet decoder architecture

(c) Feature transformer

(d) Attentive transformer

Basic 모델

기본 모형 적합

4. 기본 모형 적합

의사결정나무모형

```
parameters={'max_depth':[3,5,8], 'min_samples_split': [5,10,15]}
```

```
# GridSearchCV 결과를 추출해 DataFrame으로 변환

scores_df=pd.DataFrame(grid_dtree.cv_results_)
scores_df[['params','mean_test_score','rank_test_score',
           'split0_test_score','split1_test_score','split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 3, 'min_samples_split': 5}	0.781966	7	0.782163	0.781187	0.780641
1	{'max_depth': 3, 'min_samples_split': 10}	0.781966	7	0.782163	0.781187	0.780641
2	{'max_depth': 3, 'min_samples_split': 15}	0.781966	7	0.782163	0.781187	0.780641
3	{'max_depth': 5, 'min_samples_split': 5}	0.796417	4	0.797177	0.796039	0.794497
4	{'max_depth': 5, 'min_samples_split': 10}	0.796417	4	0.797177	0.796039	0.794497
5	{'max_depth': 5, 'min_samples_split': 15}	0.796417	4	0.797177	0.796039	0.794497
6	{'max_depth': 8, 'min_samples_split': 5}	0.800427	2	0.801262	0.799699	0.797962
7	{'max_depth': 8, 'min_samples_split': 10}	0.800439	1	0.801262	0.799718	0.797976
8	{'max_depth': 8, 'min_samples_split': 15}	0.800423	3	0.801179	0.799718	0.797976

```
print('GridSearchCV 최적 파라미터:', grid_dtree.best_params_)
print('GridSearchCV 최고 정확도:{0:.4f}'.format(grid_dtree.best_score_))
```

```
GridSearchCV 최적 파라미터: {'max_depth': 8, 'min_samples_split': 10}
GridSearchCV 최고 정확도:0.8004
```

```
pred_dtree=best_dtree.predict(X_test)
prob_dtree=best_dtree.predict_proba(X_test)[:,1]

print('Test set accuracy: ', accuracy_score(y_test, pred_dtree))
print('Test set precision: ',precision_score(y_test,pred_dtree))
print('Test set recall: ', recall_score(y_test,pred_dtree))
print('Test set F1 score: ', f1_score(y_test, pred_dtree))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_dtree))
```

```
Test set accuracy:  0.7941570881226053
Test set precision:  0.7969447249275268
Test set recall:  0.9913695299837926
Test set F1 score:  0.8835882342319569
Test set AUC Score:  0.7992216849697209
```

4. 기본 모형 적합

로지스틱 회귀모형

```
param_grid = {'C': [ 0.01, 0.1, 1, 10, 100]}
```

C: 비용함수

```
scores_df=pd.DataFrame(grid_lg.cv_results_)
scores_df[['params','mean_test_score','rank_test_score',
           'split0_test_score','split1_test_score','split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'C': 0.01}	0.760185	5	0.762202	0.759428	0.757892
1	{'C': 0.1}	0.761106	4	0.763031	0.760222	0.758837
2	{'C': 1}	0.761195	3	0.763108	0.760296	0.758927
3	{'C': 10}	0.761203	2	0.763116	0.760303	0.758936
4	{'C': 100}	0.761204	1	0.763117	0.760304	0.758937

```
print('GridSearchCV 최적 파라미터:', grid_lg.best_params_)
print('GridSearchCV 최고 정확도:{0:.4f}'.format(grid_lg.best_score_))
```

```
GridSearchCV 최적 파라미터: {'C': 100}
GridSearchCV 최고 정확도:0.7612
```

```
pred_lg=best_lg.predict(X_test_transform)
prob_lg=best_lg.predict_proba(X_test_transform)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_lg))
print('Test set precision: ',precision_score(y_test,pred_lg))
print('Test set recall: ', recall_score(y_test,pred_lg))
print('Test set F1 score: ', f1_score(y_test, pred_lg))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_lg))
```

```
Test accuracy:  0.7826947637292465
Test set precision:  0.7927380523436732
Test set recall:  0.9806118314424636
Test set F1 score:  0.8767229980619102
Test set AUC Score:  0.7608525751894583
```

4. 기본 모형 적합

KNN

```
grid_params = {
    'n_neighbors' : list(range(1,10)),
    'metric' : ['euclidean', 'manhattan']
}
```

```
scores_df=pd.DataFrame(grid_knn.cv_results_)
scores_df[['params','mean_test_score','rank_test_score',
           'split0_test_score','split1_test_score','split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'metric': 'euclidean', 'n_neighbors': 1}	0.600098	18	0.600603	0.600177	0.598121
1	{'metric': 'euclidean', 'n_neighbors': 2}	0.656225	16	0.656728	0.656227	0.654932
2	{'metric': 'euclidean', 'n_neighbors': 3}	0.690611	13	0.691799	0.691161	0.688205
3	{'metric': 'euclidean', 'n_neighbors': 4}	0.712614	12	0.713815	0.713825	0.710074
4	{'metric': 'euclidean', 'n_neighbors': 5}	0.728446	9	0.729568	0.729534	0.726014
5	{'metric': 'euclidean', 'n_neighbors': 6}	0.739388	8	0.740244	0.740277	0.737429
6	{'metric': 'euclidean', 'n_neighbors': 7}	0.748084	5	0.748568	0.748370	0.746657
7	{'metric': 'euclidean', 'n_neighbors': 8}	0.754548	3	0.754943	0.754444	0.753308
8	{'metric': 'euclidean', 'n_neighbors': 9}	0.759088	2	0.759583	0.758808	0.757500
9	{'metric': 'manhattan', 'n_neighbors': 1}	0.600169	17	0.600241	0.600280	0.597937
10	{'metric': 'manhattan', 'n_neighbors': 2}	0.656305	15	0.657659	0.655252	0.654797
11	{'metric': 'manhattan', 'n_neighbors': 3}	0.690405	14	0.692849	0.690406	0.688909
12	{'metric': 'manhattan', 'n_neighbors': 4}	0.713150	11	0.714240	0.713809	0.711846
13	{'metric': 'manhattan', 'n_neighbors': 5}	0.728339	10	0.729485	0.728138	0.726780
14	{'metric': 'manhattan', 'n_neighbors': 6}	0.739520	7	0.740656	0.739546	0.737926
15	{'metric': 'manhattan', 'n_neighbors': 7}	0.748084	6	0.749312	0.748223	0.746238
16	{'metric': 'manhattan', 'n_neighbors': 8}	0.754287	4	0.755703	0.754355	0.751946
17	{'metric': 'manhattan', 'n_neighbors': 9}	0.759350	1	0.760620	0.758887	0.757151

```
print('GridSearchCV 최적 파라미터:', grid_knn.best_params_)
print('GridSearchCV 최고 정확도:{0:.4f}'.format(grid_knn.best_score_))
```

```
GridSearchCV 최적 파라미터: {'metric': 'manhattan', 'n_neighbors': 9}
GridSearchCV 최고 정확도:0.7594
```

```
pred_knn=best_knn.predict(X_test_transform)
prob_knn=best_knn.predict_proba(X_test_transform)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_knn))
print('Test set precision: ',precision_score(y_test,pred_knn))
print('Test set recall: ', recall_score(y_test,pred_knn))
print('Test set F1 score: ', f1_score(y_test, pred_knn))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_knn))
```

```
Test accuracy:  0.7738625478927204
Test set precision:  0.8205014957449812
Test set recall:  0.9126873987034035
Test set F1 score:  0.8641428273426317
Test set AUC Score:  0.7578260266806692
```

Ensemble Model and Deep Learning Model

앙상블 모형과 딥러닝 모형

5. 앙상블 모형과 딥러닝 모형

RandomForest

```
params={
    'n_estimators': [100,200],
    'max_depth':[5,7,9],
    'min_samples_split': [2,4,8]
}
```

```
print('GridSearchCV 최적 하이퍼 파라미터: ',grid_rf.best_params_)
print('GridSearchCV  최고 예측 정확도: {0:.4f}'.format(grid_rf.best_score_))
```

```
GridSearchCV 최적 하이퍼 파라미터:  {'max_depth': 9, 'min_samples_split': 8, 'n_estimators': 200}
GridSearchCV  최고 예측 정확도: 0.8009
```

```
scores_df=pd.DataFrame(grid_rf.cv_results_)
scores_df[['params','mean_test_score','rank_test_score',
           'split0_test_score','split1_test_score','split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 5, 'min_samples_split': 2, 'n_es...	0.794402	14	0.794292	0.793511	0.793159
1	{'max_depth': 5, 'min_samples_split': 2, 'n_es...	0.794248	17	0.794559	0.793392	0.792615
2	{'max_depth': 5, 'min_samples_split': 4, 'n_es...	0.794404	13	0.794293	0.793511	0.793163
3	{'max_depth': 5, 'min_samples_split': 4, 'n_es...	0.794259	16	0.794529	0.793380	0.792564
4	{'max_depth': 5, 'min_samples_split': 8, 'n_es...	0.794387	15	0.794297	0.793483	0.793163
5	{'max_depth': 5, 'min_samples_split': 8, 'n_es...	0.794244	18	0.794490	0.793371	0.792510
6	{'max_depth': 7, 'min_samples_split': 2, 'n_es...	0.798781	9	0.799301	0.798156	0.797162
7	{'max_depth': 7, 'min_samples_split': 2, 'n_es...	0.798737	11	0.799270	0.798014	0.797030
8	{'max_depth': 7, 'min_samples_split': 4, 'n_es...	0.798814	7	0.799287	0.798196	0.796854
9	{'max_depth': 7, 'min_samples_split': 4, 'n_es...	0.798775	10	0.799218	0.798027	0.796991
10	{'max_depth': 7, 'min_samples_split': 8, 'n_es...	0.798787	8	0.799363	0.798045	0.796839
11	{'max_depth': 7, 'min_samples_split': 8, 'n_es...	0.798682	12	0.799198	0.797862	0.796816
12	{'max_depth': 9, 'min_samples_split': 2, 'n_es...	0.800623	5	0.801437	0.799598	0.798677
13	{'max_depth': 9, 'min_samples_split': 2, 'n_es...	0.800594	6	0.801195	0.799534	0.798724
14	{'max_depth': 9, 'min_samples_split': 4, 'n_es...	0.800763	3	0.801403	0.800109	0.798890
15	{'max_depth': 9, 'min_samples_split': 4, 'n_es...	0.800680	4	0.801325	0.799831	0.798859
16	{'max_depth': 9, 'min_samples_split': 8, 'n_es...	0.800828	2	0.801366	0.799988	0.798967
17	{'max_depth': 9, 'min_samples_split': 8, 'n_es...	0.800850	1	0.801389	0.799946	0.799087

```
# GridSearchCV의 best_estimator_는 이미 최적 학습이 됐으므로 별도 학습이 필요 없음

pred_rf=grid_rf.predict(X_test)
prob_rf=best_rf.predict_proba(X_test)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_rf))
print('Test set precision: ',precision_score(y_test,pred_rf))
print('Test set recall: ', recall_score(y_test,pred_rf))
print('Test set F1 score: ', f1_score(y_test, pred_rf))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_rf))
```

```
Test accuracy:  0.7937060983397191
Test set precision:  0.7948002249163643
Test set recall:  0.9951225688816856
Test set F1 score:  0.883751683935202
Test set AUC Score:  0.7988611837411181
```

5. 앙상블 모형과 딥러닝 모형

XGBoost

```
params={'n_estimators':[100, 200],'max_depth':[5,7,9,11]}
```

```
scores_df=pd.DataFrame(grid_xg.cv_results_)
scores_df[['params','mean_test_score','rank_test_score',
           'split0_test_score','split1_test_score','split2_test_score']]
```

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	split2_test_score
0	{'max_depth': 5, 'n_estimators': 100}	0.803744	1	0.804595	0.802360	0.802092
1	{'max_depth': 5, 'n_estimators': 200}	0.803125	2	0.804186	0.801708	0.801449
2	{'max_depth': 7, 'n_estimators': 100}	0.802300	3	0.803099	0.801069	0.800334
3	{'max_depth': 7, 'n_estimators': 200}	0.800492	4	0.801668	0.798741	0.798622
4	{'max_depth': 9, 'n_estimators': 100}	0.800035	5	0.800462	0.799310	0.798297
5	{'max_depth': 9, 'n_estimators': 200}	0.796707	6	0.797707	0.795552	0.795423
6	{'max_depth': 11, 'n_estimators': 100}	0.796636	7	0.797181	0.795770	0.795335
7	{'max_depth': 11, 'n_estimators': 200}	0.792062	8	0.792971	0.790569	0.791733

```
print('최적 하이퍼 파라미터:\n',grid_xg.best_params_)
print('최고 예측 정확도: {0:.4f}'.format(grid_xg.best_score_))
```

```
최적 하이퍼 파라미터:
 {'max_depth': 5, 'n_estimators': 100}
최고 예측 정확도: 0.8037
```

```
# GridSearchCV의 best_estimator_는 이미 최적 학습이 됐으므로 별도 학습이 필요 없음

pred_xg=best_xg.predict(X_test)
prob_xg=best_xg.predict_proba(X_test)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_xg))
print('Test set precision: ',precision_score(y_test,pred_xg))
print('Test set recall: ', recall_score(y_test,pred_xg))
print('Test set F1 score: ', f1_score(y_test, pred_xg))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_xg))
```

```
Test accuracy:  0.7953983077905492
Test set precision:  0.7988860399209923
Test set recall:  0.9894347649918963
Test set F1 score:  0.8840087516658258
Test set AUC Score:  0.8020046028643291
```

5. 앙상블 모형과 딥러닝 모형

Soft Voting Classifier

```
# 개별 모델을 소프트 보팅 기반의 앙상블 모델로 구현한 분류기

from sklearn.ensemble import VotingClassifier
vo_clf=VotingClassifier(estimators=[('LR',best_lg),('KNN',best_knn),('Dtree',best_dtree),('XGB',best_xg),('Rf',best_rf)],voting='soft')
```

```
# VotingClassifier 학습/예측/평가.

vo_clf.fit(X_train,y_train)
```

```
pred_vo=vo_clf.predict(X_test)
prob_vo=vo_clf.predict_proba(X_test)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_vo))
print('Test set precision: ',precision_score(y_test,pred_vo))
print('Test set recall: ', recall_score(y_test,pred_vo))
print('Test set F1 score: ', f1_score(y_test, pred_vo))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_vo))
```

```
Test accuracy:  0.7942608556832694
Test set precision:  0.7966329144809077
Test set recall:  0.9922001620745543
Test set F1 score:  0.8837261757076802
Test set AUC Score:  0.797722502350493
```

- 이전에 구축한 각각의 모델들(로지스틱 회귀, 의사결정나무 모델, KNN, RandomForest와 Xgboost)을 앙상블한 모델임
- Voting 방식은 'soft'로 설정하여 각각의 클래스들의 예측확률값을 평균을 냄.

5. 앙상블 모형과 딥러닝 모형

TabNet

- Tabnet 딥러닝 모델을 구축할 때는 Train data의 일부를 validation set으로 이용함

```
tab_classifier=TabNetClassifier(verbose=0, optimizer_fn=torch.optim.Adam,seed=42)
tab_classifier.fit(X_train=X_train.values, y_train=y_train.values.ravel(), eval_set=[(X_val.values, y_val.values.ravel())], batch_size=500, eval_metric=['auc'], patience=5, max_epochs=30)
```

```
Early stopping occurred at epoch 6 with best_epoch = 1 and best_val_0_auc = 0.79389
```

```
pred_tab=tab_classifier.predict(X_test.values)
prob_tab=tab_classifier.predict_proba(X_test.values)[:,1]

print('Test accuracy: ', accuracy_score(y_test, pred_tab))
print('Test set precision: ',precision_score(y_test,pred_tab))
print('Test set recall: ', recall_score(y_test,pred_tab))
print('Test set F1 score: ', f1_score(y_test, pred_tab))
print('Test set AUC Score: ', roc_auc_score(y_test,prob_tab))
```

```
Test accuracy:  0.7880946679438059
Test set precision:  0.7883981410812257
Test set recall:  0.9992858589951378
Test set F1 score:  0.8814031043594607
Test set AUC Score:  0.7901565271735748
```

- Batch_size: 연산 한 번에 들어가는 데이터의 크기
- Patience: 개선이 안된다고 종료시키지 않고 몇 번의 에포크를 기다릴지 설정함

CONCLUSION

최종결론

6. 최종결론

- Xgboost 모델이 가장 최적의 용해 제조 품질 분류모델이라고 할 수 있음
  - 용해 품질이 불량인 제품들을 잘 걸러내야하므로 다른 metric들보다 F-1 Score와 AUC score값이 높은 모델을 가장 최적의 모델로 봐야됨

Model	Accuracy	Precision	Recall	F1-Score	AUC
Decision Tree	0.7942	0.7970	0.9914	0.8836	0.7992
Logistic Regression	0.7827	0.7928	0.9806	0.8767	0.7609
KNN	0.7739	0.8205	0.9127	0.8641	0.7578
RF	0.7937	0.7948	0.9951	0.8838	0.7989
XGBoost	0.7954	0.7989	0.9894	0.8840	0.8020
Voting	0.7943	0.7966	0.9922	0.8837	0.7977
TabNet	0.7881	0.7884	0.9993	0.8814	0.7901

감사합니다