AutoML

ML Session 8차시

빅데이터 분석 학회 D&A

CONTENTS.

01. DACON

- 참여자 분석

02. AutoML

- AutoML이란?
- PyCaret
- AutoGluon

DACON

참여자 분석

대구 교통사고 피해 예측 AI 경진대회

2023.11.15 ~ 2023.12.11 (대회 링크)

잠만 [Private 0.42743] 대구교통사고예측 PyCaret

 [public:0.42682/Private:0.42723]AutoML 및 Feature Engineering

HD현대 AI Challenge

2023.09.25 ~ 2023.10.30 (대회 링크)

[Private 4th] AutoML(mljar) + Feature engineering

 [Private 5th] AutoML(autogluon) + Bertime_predictions

참여자 분석

제주 특산물 가격 예측 AI 경진대회

2023.10.26 ~ 2023.11.20 (대회 링크)

[private 4th] catboost + autogluon

[private 8th] Autogluon TabularPredictor (code + PPT)

웹 광고 클릭률 예측 AI 경진대회

2024.05.07 ~ 2024.06.03 (대회 링크)

yo

[Private 10위 0.7851] Under Sampling + AutoGluon

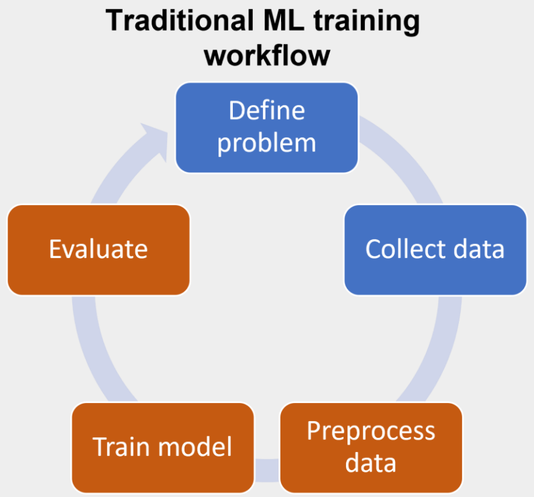
AutoML이란?

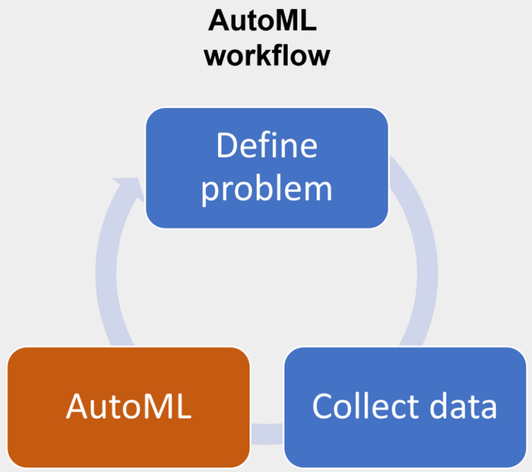
AutoML

- 말 그대로 자동화된 Machine Learning

- 시간 소모적이고 반복적인 기계 학습 모델 개발 작업 (데이터 전처리, 모델링, 하이퍼파라미터 튜닝 등)을 자동화하는 프로세스

- 데이터 과학 전문 지식과 프로그래밍 스킬이 없어도 누구나 쉽게 머신러닝을 활용할 수 있도록 도와줌





AutoML

PyCaret

PyCaret

- Python에서 제공하는 open-source 라이브러리
- 몇 줄의 코드만으로 ML Workflow를 구현할 수 있게 해줌 (low-code 라이브러리)
- 이로 인해 실험을 매우 빠르고 효율적으로 진행할 수 있음
- Classification, Regression, Clustering, Anomaly Detection 등 Task의 모델 지원

AutoML

PyCaret

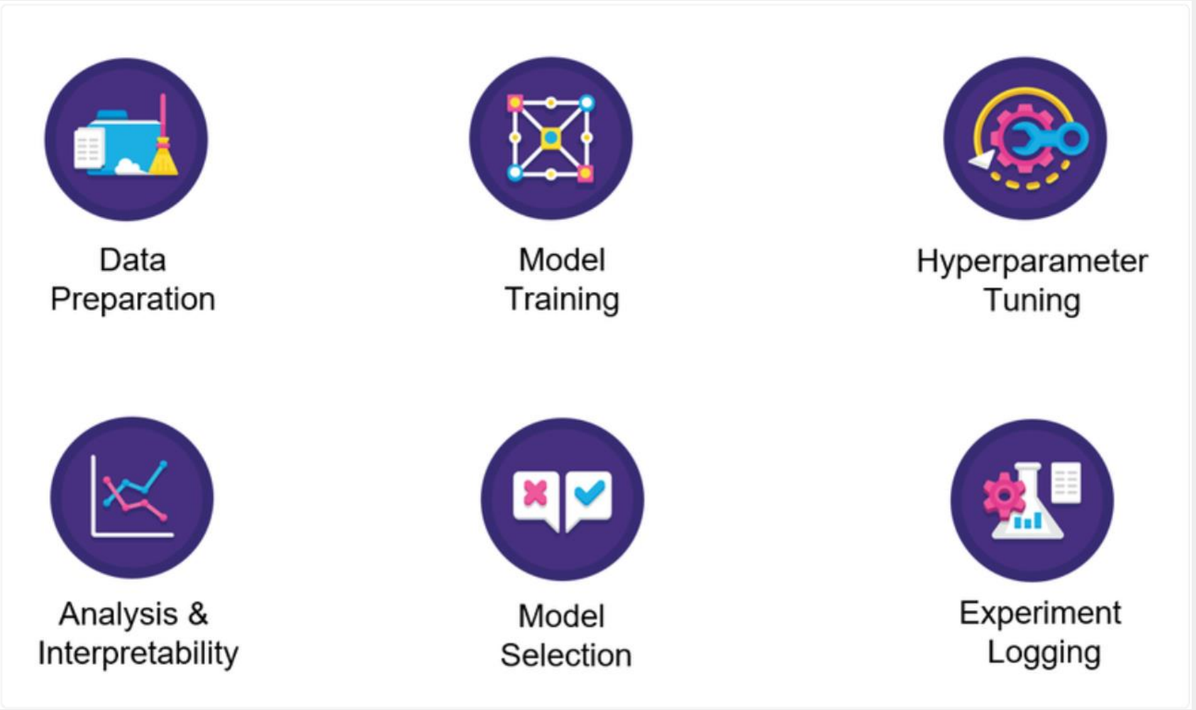
주요 기능

1) Data Preparation

- 데이터 전처리를 자동으로 수행
- ex) 결측값 처리, 범주형 데이터 인코딩, 데이터 스케일링

2) Model Training

- 다양한 모델 지원 (Classification, Regression, Clustering, Anomaly Detection 등)
- 여러 모델을 동시에 자동으로 훈련시킴



AutoML

PyCaret

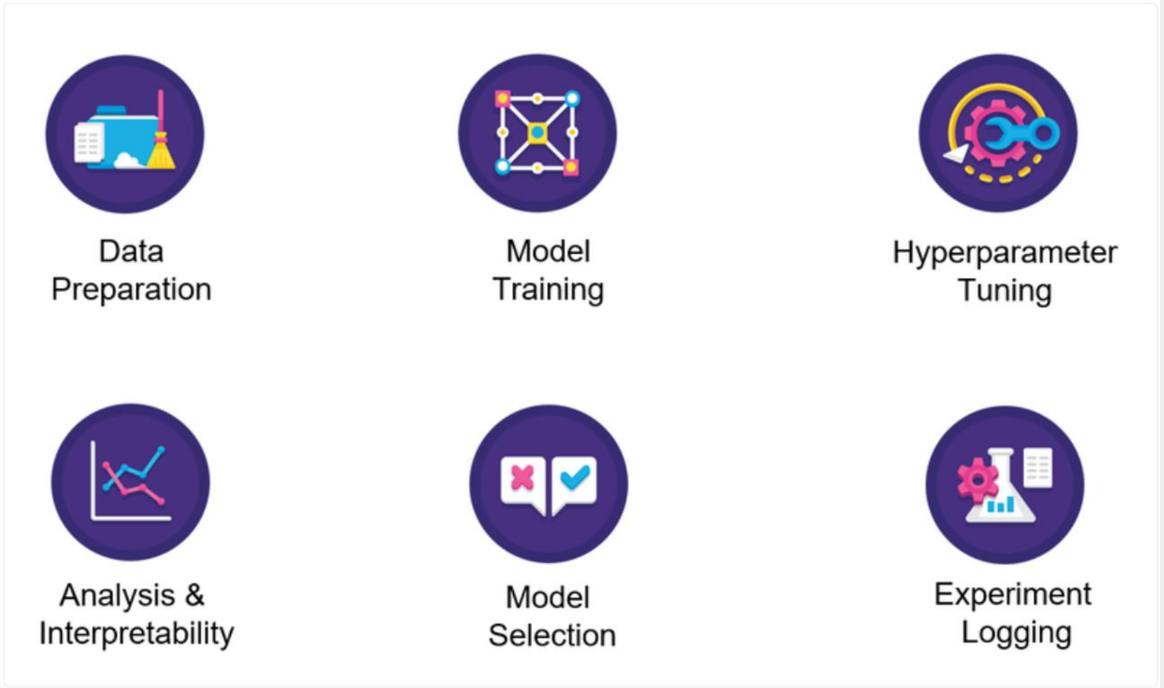
주요 기능

3) Hyperparameter Tuning

- Grid Search, Random Search, Bayesian Optimization 등 다양한 방법을 사용하여 hyperparameter 자동으로 tuning
- tuning을 병렬로 수행하여 tuning 시간을 단축시킴

4) Model Selection

- 동시에 훈련시킨 모델들의 주요 성능 지표를 한눈에 확인할 수 있음
- 자동으로 가장 성능이 우수한 모델을 선택하여 예측을 수행하거나 배포할 수 있음
- 앙상블 기법도 지원 (Bagging, Boosting, Stacking 등)



AutoML

PyCaret

코드 실습

실습 환경 (Google Colab)

1) PyCaret Install

```
!pip install pycaret
```

2) 데이터셋 로드

```
# 당뇨병 데이터셋 로드
from pycaret.datasets import get_data
data = get_data('diabetes')
```

*Binary Classification

AutoML

PyCaret

코드 실습

3) Setup

- 모델을 훈련하고 평가하기 전에, 필요한 전처리 작업을 한 번에 처리해주는 함수
- 데이터와 target 변수를 필수로 받으며, 나머지 매개변수는 선택사항
- help(setup)을 통해 사용 가능한 파라미터 확인 가능

```
[16]  from pycaret.classification import *
      s = setup(data, target = 'Class variable', normalize = True, normalize_method = 'minmax', session_id = 777)
```

*pycaret.regression, pycaret.clustering, pycaret.anomaly, pycaret.time_series

	Description	Value
0	Session id	777
1	Target	Class variable
2	Target type	Binary
3	Original data shape	(768, 9)
4	Transformed data shape	(768, 9)
5	Transformed train set shape	(537, 9)
6	Transformed test set shape	(231, 9)
7	Numeric features	8
8	Preprocess	True
9	Imputation type	simple
10	Numeric imputation	mean
11	Categorical imputation	mode
12	Normalize	True
13	Normalize method	minmax
14	Fold Generator	StratifiedKFold
15	Fold Number	10
16	CPU Jobs	-1
17	Use GPU	False
18	Log Experiment	False
19	Experiment Name	clf-default-name
20	USI	dbcd

AutoML

PyCaret

코드 실습

4) create_model()

- 단일 ML 모델 생성, 학습, 성능까지 한번에 실행
- help(create_model)을 통해 사용 가능한 파라미터 확인 가능

```
rf = create_model('rf')
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.7820	0.5263	0.6250	0.5714	0.3682	0.3711
1	0.7778	0.8647	0.3684	1.0000	0.5385	0.4306	0.5238
2	0.7593	0.7504	0.5789	0.6875	0.6286	0.4524	0.4561
3	0.7037	0.7812	0.4211	0.6154	0.5000	0.2998	0.3107
4	0.7222	0.7301	0.5263	0.6250	0.5714	0.3682	0.3711
5	0.7963	0.8451	0.5789	0.7857	0.6667	0.5248	0.5375
6	0.7407	0.7895	0.6316	0.6316	0.6316	0.4316	0.4316
7	0.8113	0.9063	0.7222	0.7222	0.7222	0.5794	0.5794
8	0.7925	0.8754	0.7222	0.6842	0.7027	0.5435	0.5439
9	0.7170	0.7452	0.5556	0.5882	0.5714	0.3604	0.3607
Mean	0.7543	0.8070	0.5632	0.6965	0.6104	0.4359	0.4486
Std	0.0364	0.0582	0.1080	0.1156	0.0685	0.0859	0.0888

AutoML

PyCaret

코드 실습

4) create_model()

- 모델 각각마다 ID 존재
- models()를 통해 사용 가능한 model의 ID 확인

```
models()
```

ID	Name	Reference	Turbo
lr	Logistic Regression	sklearn.linear_model._logistic.LogisticRegression	True
knn	K Neighbors Classifier	sklearn.neighbors._classification.KNeighborsCl...	True
nb	Naive Bayes	sklearn.naive_bayes.GaussianNB	True
dt	Decision Tree Classifier	sklearn.tree._classes.DecisionTreeClassifier	True
svm	SVM - Linear Kernel	sklearn.linear_model._stochastic_gradient.SGDC...	True
rbfsvm	SVM - Radial Kernel	sklearn.svm._classes.SVC	False
gpc	Gaussian Process Classifier	sklearn.gaussian_process._gpc.GaussianProcessC...	False
mlp	MLP Classifier	sklearn.neural_network._multilayer_perceptron....	False
ridge	Ridge Classifier	sklearn.linear_model._ridge.RidgeClassifier	True
rf	Random Forest Classifier	sklearn.ensemble._forest.RandomForestClassifier	True
qda	Quadratic Discriminant Analysis	sklearn.discriminant_analysis.QuadraticDiscrim...	True
ada	Ada Boost Classifier	sklearn.ensemble._weight_boosting.AdaBoostClas...	True
gbc	Gradient Boosting Classifier	sklearn.ensemble._gb.GradientBoostingClassifier	True
lda	Linear Discriminant Analysis	sklearn.discriminant_analysis.LinearDiscrimina...	True
et	Extra Trees Classifier	sklearn.ensemble._forest.ExtraTreesClassifier	True
xgboost	Extreme Gradient Boosting	xgboost.sklearn.XGBClassifier	True
lightgbm	Light Gradient Boosting Machine	lightgbm.sklearn.LGBMClassifier	True
dummy	Dummy Classifier	sklearn.dummy.DummyClassifier	True

PyCaret

코드 실습

5) compare_models()

- 다양한 모델 성능 비교
- sort : 모델을 정렬할 평가 지표
- n_select : 상위 몇개의 모델을 선택할지 지정

```
best3 = compare_models(sort='Accuracy', n_select=3, fold=5 )
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.7730	0.8305	0.5518	0.7334	0.6255	0.4687	0.4805	0.0460
lr	Logistic Regression	0.7729	0.8296	0.5088	0.7606	0.6064	0.4570	0.4766	0.8160
lda	Linear Discriminant Analysis	0.7711	0.8286	0.5680	0.7176	0.6286	0.4683	0.4778	0.0340
nb	Naive Bayes	0.7617	0.8106	0.6051	0.6772	0.6366	0.4610	0.4643	0.0500
qda	Quadratic Discriminant Analysis	0.7599	0.8051	0.5947	0.6769	0.6296	0.4542	0.4586	0.0560
rf	Random Forest Classifier	0.7524	0.7987	0.5679	0.6780	0.6107	0.4332	0.4411	0.3820
ada	Ada Boost Classifier	0.7524	0.7856	0.5999	0.6630	0.6251	0.4422	0.4469	0.1280
svm	SVM - Linear Kernel	0.7430	0.8250	0.5728	0.6933	0.5963	0.4172	0.4440	0.0500
et	Extra Trees Classifier	0.7393	0.7996	0.5302	0.6638	0.5846	0.3991	0.4074	0.1720
gbc	Gradient Boosting Classifier	0.7356	0.8099	0.5623	0.6391	0.5937	0.4004	0.4049	0.1580
knn	K Neighbors Classifier	0.7319	0.7572	0.5195	0.6484	0.5726	0.3817	0.3893	0.0660
xgboost	Extreme Gradient Boosting	0.7282	0.7722	0.5997	0.6182	0.6050	0.3988	0.4017	0.0880
lightgbm	Light Gradient Boosting Machine	0.7244	0.7905	0.5835	0.6134	0.5921	0.3859	0.3900	0.2460
dt	Decision Tree Classifier	0.6610	0.6306	0.5297	0.5147	0.5204	0.2591	0.2602	0.0520
dummy	Dummy Classifier	0.6518	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0300

AutoML

PyCaret

코드 실습

6) tune_model()

- hyperparameter tuning
- (기본값) scikit-learn의 random search,
  optimize = 'Accuracy' , fold=10

```
tuned_top3 = [tune_model(i) for i in best3]
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.7474	0.4737	0.6429	0.5455	0.3520	0.3605
1	0.7778	0.8391	0.3684	1.0000	0.5385	0.4306	0.5238
2	0.7593	0.7925	0.5263	0.7143	0.6061	0.4384	0.4490
3	0.7593	0.8556	0.4737	0.7500	0.5806	0.4236	0.4456
4	0.7963	0.8165	0.5263	0.8333	0.6452	0.5123	0.5389
5	0.7593	0.8421	0.5263	0.7143	0.6061	0.4384	0.4490
6	0.7778	0.8707	0.5263	0.7692	0.6250	0.4749	0.4921
7	0.8113	0.9032	0.6111	0.7857	0.6875	0.5554	0.5644
8	0.7925	0.8429	0.6111	0.7333	0.6667	0.5178	0.5223
9	0.7547	0.7460	0.4444	0.7273	0.5517	0.3961	0.4189
Mean	0.7710	0.8256	0.5088	0.7670	0.6053	0.4540	0.4765
Std	0.0242	0.0483	0.0696	0.0911	0.0490	0.0582	0.0595

Fitting 10 folds for each of 10 candidates, totalling 100 fits

AutoML

PyCaret

코드 실습

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.7534	0.4737	0.6429	0.5455	0.3520	0.3605
1	0.7407	0.8211	0.3158	0.8571	0.4615	0.3357	0.4083
2	0.7222	0.7820	0.5263	0.6250	0.5714	0.3682	0.3711
3	0.7407	0.8632	0.5789	0.6471	0.6111	0.4176	0.4190
4	0.7963	0.8165	0.5789	0.7857	0.6667	0.5248	0.5375
5	0.7963	0.8692	0.6316	0.7500	0.6857	0.5367	0.5410
6	0.7963	0.8662	0.6316	0.7500	0.6857	0.5367	0.5410
7	0.8491	0.9063	0.7222	0.8125	0.7647	0.6542	0.6566
8	0.7925	0.8397	0.6667	0.7059	0.6857	0.5310	0.5315
9	0.7925	0.7524	0.5556	0.7692	0.6452	0.5038	0.5172
Mean	0.7749	0.8270	0.5681	0.7345	0.6323	0.4761	0.4884
Std	0.0392	0.0492	0.1078	0.0737	0.0826	0.0977	0.0896

Fitting 10 folds for each of 10 candidates, totalling 100 fits

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7037	0.7504	0.4737	0.6000	0.5294	0.3175	0.3223
1	0.7593	0.8421	0.3684	0.8750	0.5185	0.3917	0.4569
2	0.7222	0.7880	0.5263	0.6250	0.5714	0.3682	0.3711
3	0.7222	0.8571	0.5789	0.6111	0.5946	0.3836	0.3839
4	0.7778	0.8150	0.5789	0.7333	0.6471	0.4882	0.4954
5	0.7963	0.8511	0.6842	0.7222	0.7027	0.5479	0.5484
6	0.7963	0.8707	0.6316	0.7500	0.6857	0.5367	0.5410
7	0.8302	0.9032	0.6667	0.8000	0.7273	0.6055	0.6108
8	0.7736	0.8413	0.6667	0.6667	0.6667	0.4952	0.4952
9	0.8113	0.7508	0.6111	0.7857	0.6875	0.5554	0.5644
Mean	0.7693	0.8270	0.5787	0.7169	0.6331	0.4690	0.4789
Std	0.0398	0.0479	0.0940	0.0859	0.0707	0.0918	0.0892

Fitting 10 folds for each of 10 candidates, totalling 100 fits

PyCaret

코드 실습

7) blend_models()

- 이전 단계에서 선택한 모델(들)을 조합하여 더욱 강력한 앙상블 모델을 만듦
- 기본적으로 10번 학습
- method : voting 방식을 지정 (soft / hard)

```
blender_top3 = blend_models(estimator_list=tuned_top3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.0000	0.4737	0.6429	0.5455	0.3520	0.3605
1	0.7593	0.0000	0.3684	0.8750	0.5185	0.3917	0.4569
2	0.7222	0.0000	0.5263	0.6250	0.5714	0.3682	0.3711
3	0.7407	0.0000	0.5789	0.6471	0.6111	0.4176	0.4190
4	0.7963	0.0000	0.5789	0.7857	0.6667	0.5248	0.5375
5	0.7963	0.0000	0.6316	0.7500	0.6857	0.5367	0.5410
6	0.7963	0.0000	0.6316	0.7500	0.6857	0.5367	0.5410
7	0.8302	0.0000	0.6667	0.8000	0.7273	0.6055	0.6108
8	0.7925	0.0000	0.6667	0.7059	0.6857	0.5310	0.5315
9	0.7925	0.0000	0.5556	0.7692	0.6452	0.5038	0.5172
Mean	0.7748	0.0000	0.5678	0.7351	0.6343	0.4768	0.4886
Std	0.0347	0.0000	0.0886	0.0756	0.0660	0.0824	0.0784

AutoML

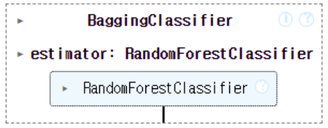
PyCaret

코드 실습

Bagging

```
ensemble_model(rf, method= 'Bagging')
```

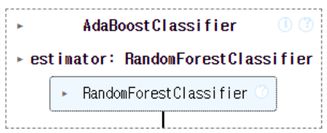
Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7407	0.7744	0.5263	0.6667	0.5882	0.4028	0.4088
1	0.7593	0.8466	0.3684	0.8750	0.5185	0.3917	0.4569
2	0.7407	0.7278	0.5263	0.6667	0.5882	0.4028	0.4088
3	0.7037	0.8090	0.4737	0.6000	0.5294	0.3175	0.3223
4	0.6852	0.7361	0.4737	0.5625	0.5143	0.2839	0.2862
5	0.7407	0.8346	0.5263	0.6667	0.5882	0.4028	0.4088
6	0.7222	0.8105	0.5789	0.6111	0.5946	0.3836	0.3839
7	0.8113	0.9151	0.7222	0.7222	0.7222	0.5794	0.5794
8	0.7736	0.8651	0.7222	0.6500	0.6842	0.5085	0.5102
9	0.7170	0.7778	0.5000	0.6000	0.5455	0.3424	0.3454
Mean	0.7394	0.8097	0.5418	0.6621	0.5873	0.4016	0.4111
Std	0.0343	0.0553	0.1041	0.0833	0.0654	0.0824	0.0829



Boosting

```
ensemble_model(rf, method= 'Boosting')
```

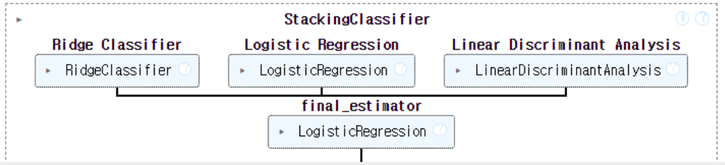
Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.7767	0.5263	0.6250	0.5714	0.3682	0.3711
1	0.7593	0.8293	0.3684	0.8750	0.5185	0.3917	0.4569
2	0.6852	0.7271	0.4211	0.5714	0.4848	0.2656	0.2720
3	0.7222	0.8128	0.4737	0.6429	0.5455	0.3520	0.3605
4	0.7222	0.7278	0.5263	0.6250	0.5714	0.3682	0.3711
5	0.7593	0.8586	0.5263	0.7143	0.6061	0.4384	0.4490
6	0.7222	0.7970	0.5789	0.6111	0.5946	0.3836	0.3839
7	0.7925	0.9079	0.6667	0.7059	0.6857	0.5310	0.5315
8	0.7925	0.8754	0.7778	0.6667	0.7179	0.5553	0.5594
9	0.7170	0.7548	0.5000	0.6000	0.5455	0.3424	0.3454
Mean	0.7394	0.8067	0.5365	0.6637	0.5841	0.3996	0.4101
Std	0.0332	0.0587	0.1114	0.0823	0.0680	0.0829	0.0838



Stacking

```
stack_models(tuned_top3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7037	0.7534	0.4737	0.6000	0.5294	0.3175	0.3223
1	0.7593	0.8376	0.3684	0.8750	0.5185	0.3917	0.4569
2	0.7222	0.7865	0.5263	0.6250	0.5714	0.3682	0.3711
3	0.7222	0.8586	0.5789	0.6111	0.5946	0.3836	0.3839
4	0.7778	0.8165	0.5789	0.7333	0.6471	0.4882	0.4954
5	0.7778	0.8571	0.6316	0.7059	0.6667	0.5008	0.5025
6	0.7963	0.8677	0.6316	0.7500	0.6857	0.5367	0.5410
7	0.8302	0.9063	0.6667	0.8000	0.7273	0.6055	0.6108
8	0.7925	0.8413	0.6667	0.7059	0.6857	0.5310	0.5315
9	0.7925	0.7508	0.6111	0.7333	0.6667	0.5178	0.5223
Mean	0.7674	0.8276	0.5734	0.7140	0.6293	0.4641	0.4738
Std	0.0380	0.0481	0.0893	0.0819	0.0677	0.0877	0.0849



PyCaret

코드 실습

8) 다시 모델 tuning

```
blender_tune3= tune_model(blender_top3)
```

Fold	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.7222	0.0000	0.4737	0.6429	0.5455	0.3520	0.3605
1	0.7407	0.0000	0.3158	0.8571	0.4615	0.3357	0.4083
2	0.7222	0.0000	0.5263	0.6250	0.5714	0.3682	0.3711
3	0.7407	0.0000	0.5789	0.6471	0.6111	0.4176	0.4190
4	0.7963	0.0000	0.5789	0.7857	0.6667	0.5248	0.5375
5	0.7963	0.0000	0.6316	0.7500	0.6857	0.5367	0.5410
6	0.7963	0.0000	0.6316	0.7500	0.6857	0.5367	0.5410
7	0.8491	0.0000	0.7222	0.8125	0.7647	0.6542	0.6566
8	0.7925	0.0000	0.6667	0.7059	0.6857	0.5310	0.5315
9	0.7925	0.0000	0.5556	0.7692	0.6452	0.5038	0.5172
Mean	0.7749	0.0000	0.5681	0.7345	0.6323	0.4761	0.4884
Std	0.0392	0.0000	0.1078	0.0737	0.0826	0.0977	0.0896

Fitting 10 folds for each of 10 candidates, totalling 100 fits

9) prediction

- finalize_model()를 통해 전체 데이터로 마지막으로 학습 진행
- predict_model()을 통해 예측 수행

```
final_model = finalize_model(blender_tune3)
prediction = predict_model(final_model)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Voting Classifier	0.7576	0.7367	0.6667	0.6506	0.6585	0.4707	0.4707

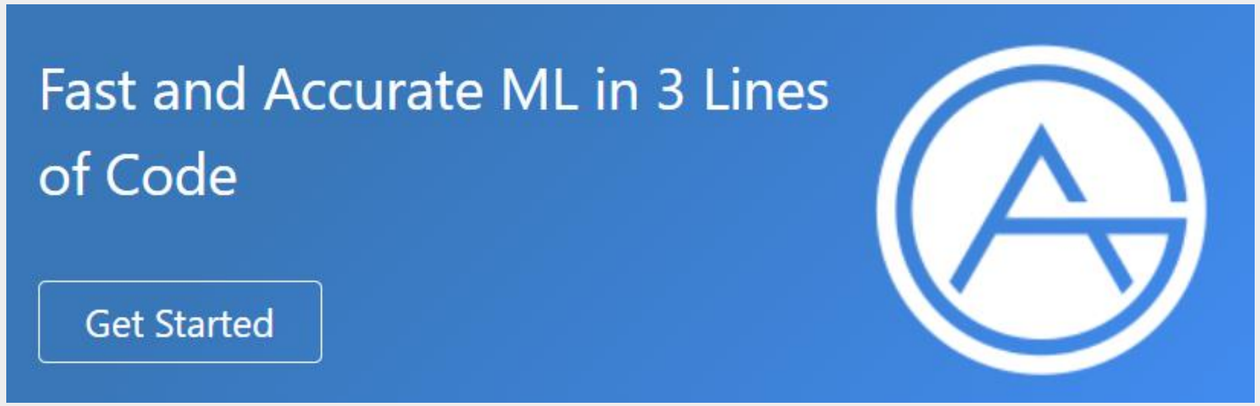
https://pycaret.gitbook.io/docs

AutoGluon

AutoGluon

- AWS (Amazon Web Services)에서 개발한 AutoML 오픈 소스 라이브러리

- 단 3줄의 코드로도 이미지, 텍스트, 시계열, Tabular 데이터를 다루는 high-accuracy 머신러닝 및 딥러닝 모델을 학습하고 배포할 수 있다고 강조



AutoML

AutoGluon

코드 실습

1) AutoGluon 라이브러리 설치

```
!python -m pip install --upgrade pip
!python -m pip install autogluon
```

2) 필요한 클래스 불러오기

TabularDataset
- Tabular 데이터를 처리하는 클래스 (AutoGluon에서 사용할 수 있는 형식으로 변환)
- pandas DataFrame의 서브클래스이며, pandas의 모든 기능을 사용할 수 있음

TabularPredictor
- Tabular 데이터를 기반으로 예측 모델을 생성하고, 학습시켜며, 예측을 수행하는 데 사용

```
from autogluon.tabular import TabularDataset, TabularPredictor
```

AutoGluon

코드 실습

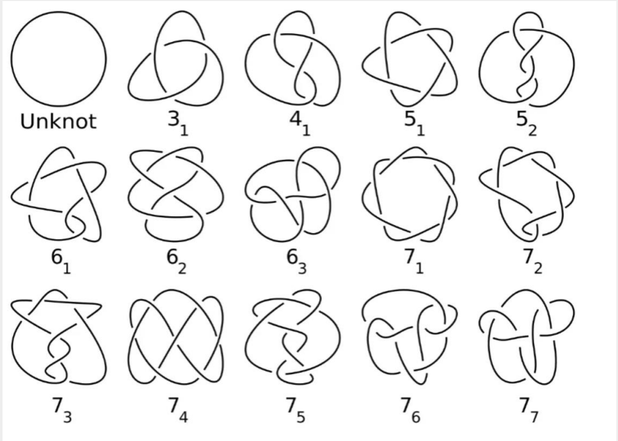
3) 데이터 불러오기

- 매듭(knot)의 다양한 특성(ex. 매듭의 모양, 꼬임의 수 등)에 기반하여 매듭의 signature를 예측하는 데이터
- Multiclass Classification

```
data_url = 'https://raw.githubusercontent.com/mli/ag-docs/main/knot_theory/'
train_data = TabularDataset(f'{data_url}train.csv')
train_data.head()
```

	Unnamed: 0	chern_simons	cusp_volume	hyperbolic_adjoint_torsion_degree	hyperbolic_torsion_degree	injectivity_radius	longitudinal_translation	meridinal_translation_imag
0	70746	0.090530	12.226322	0	10	0.507756	10.685555	1.144192
1	240827	0.232453	13.800773	0	14	0.413645	10.453156	1.320249
2	155659	-0.144099	14.761030	0	14	0.436928	13.405199	1.101142
3	239963	-0.171668	13.738019	0	22	0.249481	27.819496	0.493827
4	90504	0.235188	15.896359	0	10	0.389329	15.330971	1.036879

signature
-2
0
2
-8
4



test 데이터는 train을 test로만 바꿔주면 됨

AutoGluon

코드 실습

4) Training

- label를 지정해주고 TabularPredictor.fit()을 사용해 데이터셋 학습 진행
- 다른 매개변수를 지정할 필요 X
- AutoGluon은 label 열의 값을 바탕으로 Multiclass Classification Task임을 자동으로 인식
- 또한, 데이터셋의 각 특성을 분석하여 자동으로 feature engineering 수행
- 그리고 여러 모델을 학습시킨 후 이들을 앙상블하여 최종 예측기 실행

```
predictor = TabularPredictor(label='signature').fit(train_data)
```

```
Fitting model: LightGBM ...
        0.956    = Validation score   (accuracy)
        7.28s    = Training   runtime
        0.13s    = Validation runtime
Fitting model: RandomForestGini ...
        0.9449   = Validation score   (accuracy)
        8.5s     = Training   runtime
        0.16s    = Validation runtime
Fitting model: RandomForestEntr ...
        0.9499   = Validation score   (accuracy)
        10.34s   = Training   runtime
        0.15s    = Validation runtime
Fitting model: CatBoost ...
        0.956    = Validation score   (accuracy)
        76.12s   = Training   runtime
        0.01s    = Validation runtime
Fitting model: ExtraTreesGini ...
        0.9469   = Validation score   (accuracy)
        3.79s    = Training   runtime
        0.24s    = Validation runtime
Fitting model: ExtraTreesEntr ...
        0.9429   = Validation score   (accuracy)
        2.88s    = Training   runtime
        0.15s    = Validation runtime
Fitting model: XGBoost ...
        0.957    = Validation score   (accuracy)
        15.23s   = Training   runtime
        0.37s    = Validation runtime
```

AutoGluon

코드 실습

5) Evaluation

- Balanced_Accuracy : 클래스 불균형이 있는 데이터셋에서 각 클래스의 정확도를 평균내어 계산한 값

- MCC (Matthews Correlation Coefficient) : 예측의 정확성과 균형성을 나타내는 지표 (-1~1값)

```
predictor.evaluate(test_data, silent=True)
```

```
{'accuracy': 0.9502,
 'balanced_accuracy': 0.7632564373347 31,
 'mcc': 0.938981847944 77}
```

AutoGluon

코드 실습

5) Evaluation

- leaderboard() : 학습된 모든 모델의 성능을 보여줍
- stack_level : level 1은 개별 모델, level 2는 앙상블 모델
- predictor.info() : predictor에 대한 정보 확인

```
predictor.leaderboard(test_data)
```

	model	score_test	score_val	eval_metric	pred_time_test	pred_time_val	fit_time	pred_time_test_marginal	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer	fit_order
0	WeightedEnsemble_L2	0.9502	0.964965	accuracy	3.507623	0.572177	43.693294	0.011337	0.001249	0.268902	2	True	14
1	LightGBM	0.9456	0.955956	accuracy	0.960481	0.134465	7.284395	0.960481	0.134465	7.284395	1	True	5
2	XGBoost	0.9448	0.956957	accuracy	2.494196	0.374554	15.225580	2.494196	0.374554	15.225580	1	True	11
3	LightGBMLarge	0.9444	0.949950	accuracy	2.509739	0.459657	18.820689	2.509739	0.459657	18.820689	1	True	13
4	CatBoost	0.9432	0.955956	accuracy	0.135866	0.012252	76.119367	0.135866	0.012252	76.119367	1	True	8
5	RandomForestEntr	0.9384	0.949950	accuracy	0.590418	0.150284	10.344932	0.590418	0.150284	10.344932	1	True	7
6	NeuralNetFastAI	0.9364	0.940941	accuracy	0.350277	0.030031	17.823136	0.350277	0.030031	17.823136	1	True	3
7	ExtraTreesGini	0.9360	0.946947	accuracy	0.731223	0.239290	3.793901	0.731223	0.239290	3.793901	1	True	9
8	ExtraTreesEntr	0.9358	0.942943	accuracy	0.761350	0.147754	2.875609	0.761350	0.147754	2.875609	1	True	10
9	RandomForestGini	0.9352	0.944945	accuracy	0.356117	0.162882	8.498667	0.356117	0.162882	8.498667	1	True	6
10	LightGBMXT	0.9320	0.945946	accuracy	2.171277	0.218824	10.289097	2.171277	0.218824	10.289097	1	True	4
11	NeuralNetTorch	0.9236	0.940941	accuracy	0.035255	0.017374	59.507827	0.035255	0.017374	59.507827	1	True	12
12	KNeighborsDist	0.2210	0.213213	accuracy	0.061395	0.016058	0.030744	0.061395	0.016058	0.030744	1	True	2
13	KNeighborsUnif	0.2180	0.223223	accuracy	0.069601	0.023459	6.945935	0.069601	0.023459	6.945935	1	True	1

AutoGluon

코드 실습

6) Prediction

- predict() : 학습된 모델을 사용해 test data의 label를 예측

```
y_pred = predictor.predict(test_data.drop(columns=[label]))
y_pred.head()
```

	signature
0	-4
1	-2
2	0
3	4
4	2
...	...
4995	-2
4996	-2
4997	0
4998	2
4999	0

5000 rows × 1 columns

https://auto.gluon.ai/stable/index.html

ML Competiton

평가지표

평가 기준

총합 (75)

- **Private 등수** : 예측 성능이 얼마나 뛰어난가? / **35**
- **코드 품질** : 코드의 오류 없이 깔끔하게 코드를 작성하였는가? / **5**
- **모델링** : 적절한 모델을 선택하고 이를 효과적으로 적용했는가? / **10**
- **피처 엔지니어링** : 창의적이면서 의미있는 피처를 생성하고, 피처 생성 과정에서 적절한 시각화를 진행하였는가? / **10**
- **피처 셀렉션** : 최종적으로 반영한 피쳐들이 타당하고, 피쳐 선택 과정이 논리적이었는가? / **5**
- **상호평가** : 타당하고 독창적인 분석을 수행했는지 상호평가 / **10**

REFERENCE

https://pycaret.gitbook.io/docs
https://mz-moonzoo.tistory.com/5
https://snowwhite1106.tistory.com/166
https://auto.gluon.ai/stable/index.html

THANK YOU

ML Session 8차시

빅데이터 분석 학회 D&A