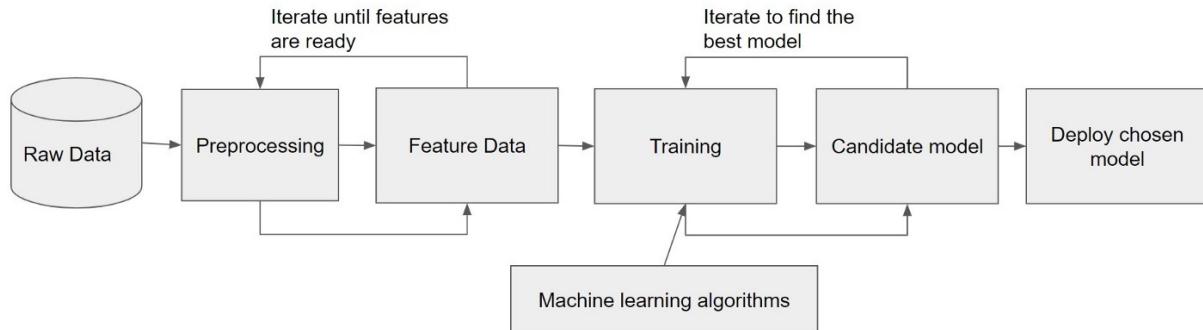
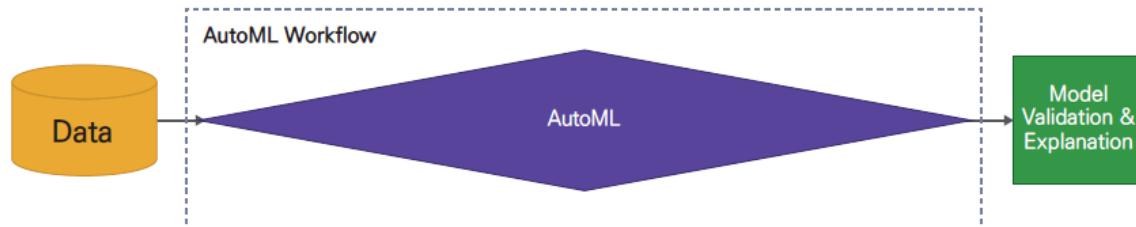
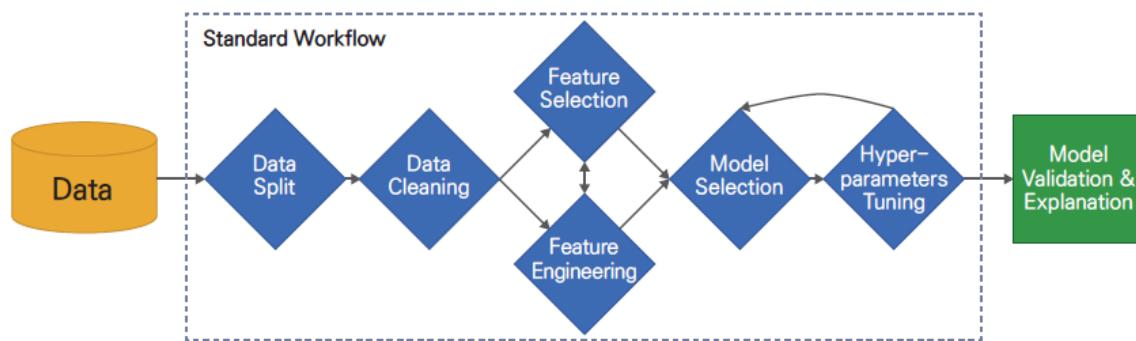


# AutoML



- 머신러닝 모델을 개발하고 실제 운영에 도입하기에는 수많은 과정을 거치게 됨
- 기존의 머신러닝을 수행하기 위해서는 데이터 전처리와 알고리즘 결과 해석까지 모든 단계에서 인간의 노력이 많이 필요
- 전지전능한 인공지능이 존재하여 컴퓨터와 데이터만 주어지면 모든 과정을 알아서 해결해 줄 수 있다면 좋겠지만, 현실의 머신러닝 모델링은 문제 정의 과정에서부터 데이터 수집, 전처리, 모델 학습 및 평가를 거쳐 서비스 적용에 이르기까지에는 여러 분야 전문가들의 많은 시간과 노력이 요구
- AutoML은 머신러닝을 적용 할 때 마다 이러한 과정을 되풀이하면서 발생하는 비효율적인 작업을 최대한 자동화하여 생산성과 효율을 높이기 위하여 등장



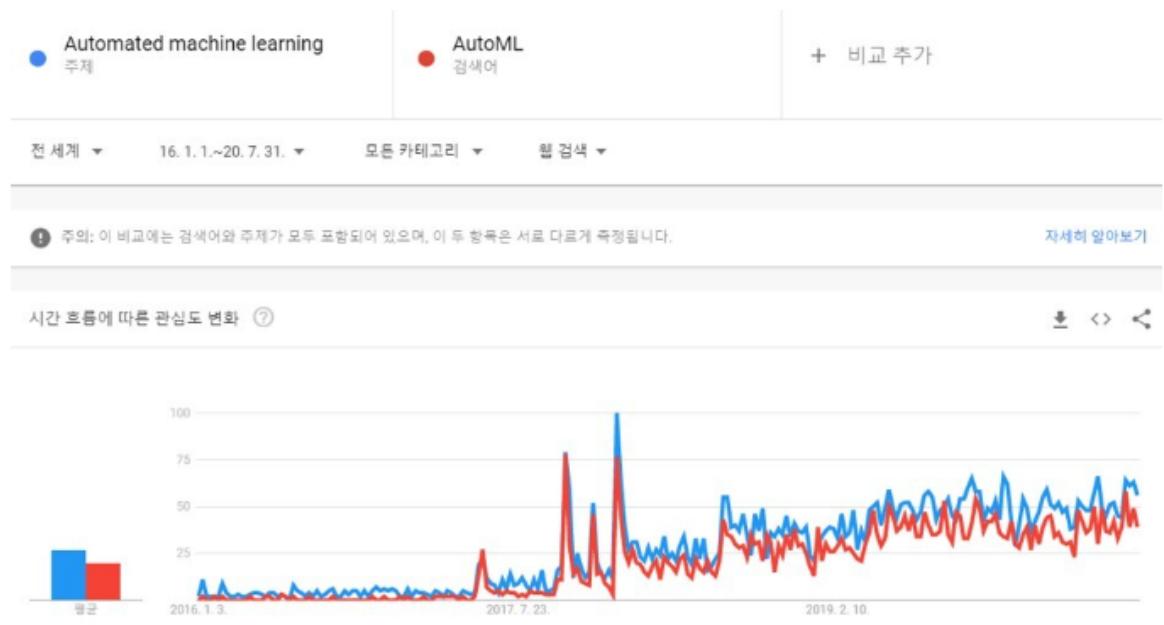
- Automatted Machine Learning은 기계 학습 파이프 라인에서 수작업과 반복되는 작업을 자동화 하는 프로세스
- 반복적으로 수행하게 되는 데이터 분할, 정제, 특징 선택 및 추출, 모델 선택, 하이퍼 파라미터 튜닝 등을 함수화하여 자동화하는 것**
- 각 머신러닝 절차 자동화의 핵심은 여러 대안들의 정확도를 자동으로 테스트하여 비교하는 것에 있음
- 특히, 데이터 전처리 과정에서부터 알고리즘 선택 및 튜닝까지의 과정에서 모델 개발자의 개입을 최소화 하여 품질 좋은 모델을 효과적으로 개발할 수 있는 기술에 대한 연구가 오랜동안 진행되어 옴
- 최근 머신러닝 분야의 발전과 관심에 힘입어 AutoML 기술을 머신러닝 모델 개발시에 손쉽게 적용할 수 있는 package toolkit들이 연구 개발되어 오고 있음

#### ▼ 주요 AutoML package toolkits

- [Auto-WEKA \(2013\)](#)
- [auto-sklearn \(2015\)](#)
- [TPOT \(2016\)](#)
- [AutoKeras \(2019\)](#)
- [Neural Network Intelligence \(2019\)](#)
- [AutoGluon \(2020\)](#)

Step1. 학습 하이퍼파라미터 최적화	Step2. 의미 있는 특성 추출	Step3. 모델링 프로세스 설계
<p><b>Hyperparameter Optimization</b></p>  <p>학습률(learning rate), 배치 크기(mini-batch size) 등 학습에 큰 영향을 주는 hyperparameter들을 학습을 통해 추정하는 것을 의미합니다.</p>	<p><b>Feature Learning</b></p>  <p>Extraction 혹은 Feature Engineering을 의미하는 것으로, 학습 모델에 입력을 그대로 사용하지 않고, 학습을 통하여 유익미한 feature(특징)를 추출해서 입력으로 사용하는 방법입니다.</p>	<p><b>Architecture Search</b></p>  <p>모델링 전체 프로세스를 사람이 직접 하나하나 설계하는 대신에 학습을 통해 최적의 아키텍처를 설계하는 방법을 의미합니다.</p>

## Auto ML 관심도 증가



- Gartner는 2020년 10대 전략 기술 트렌드를 발표하면서 “데이터 과학 직업의 40% 이상이 2020년까지 자동화 될 것이다.”라고 예측함
- 실제로 인공지능과 머신러닝 기반의 데이터 과학 직업에 대한 요구는 증가하고 있으며, AutoML 솔루션과 서비스는 점점 더 자동화되고 대중화 되고 있음

## Auto ML이 왜 필요한가?

- 새로운 비즈니스 가치를 찾고 경쟁 우위를 확보하려면, 비즈니스 활동을 통해 축적한 많은 양의 데이터를 신속하고 정확하게 분석하여 필요하는 일은 이제 선택이 아닌 필수가 됨
- 즉, 데이터를 통한 기업의 혁신과 성장은 필연이라고 할 수 있음
- Auto ML은 프로세스를 자동화를 통하여 전문 지식 없이도 기업 구성원 누구나 몇 번 마우스클릭만으로도 손쉽게 머신러닝을 활용할 수 있기 때문에 전문 인력이 부족한 기업에게는 좋은 선택
- 또한 데이터 사이언티스트들도 더 짧은 시간에 더 많은 모델을 구축함으로써 모델 품질과 정확도를 개선하고, 보다 생산적인 일에 몰두하며 전문성을 강화할 수 있음

## 1) 머신러닝의 기회비용

- 기존 머신러닝에 투입되는 기회비용이 상당함

- 하나의 머신러닝 작업을 수행하는 데는 정말 많은 시간과 노력이 요구됨, 여기서 중요한 점은 들이는 노력의 많은 부분이 여러 알고리즘 간 성능 비교와 같은 단순 작업의 반복
- 결국 머신러닝을 활용하기 위해서는 그 절차와 시스템을 정확하게 이해하고 있는 고급 인력이 단순 반복 작업을 수행해야 한다는 딜레마에 빠지게됨
- 이때 AutoML을 적용하게 되면, 데이터 분석가가 훨씬 더 생산적이고 발전적인 일을 할 수 있음

## Auto ML의 장점

- AutoML의 초기 목적대로 원활하게 모든 작업을 자동화한다면 시간 측면에서 이점을 얻을 수 있음
- AutoML을 적용하면 데이터 분석가 입장에서 직접 모든 코드를 구성하고, 각 머신러닝 알고리즘의 성능을 비교할 필요가 없음
- 이 모든 과정을 컴퓨터가 자동으로 진행해주기 때문에 상대적으로 간단한 코드 몇 줄만 적어내면, 모든 머신러닝 작업을 끝낼 수 있음
- 더 이상 데이터 분석가가 모델 간 비교 방법을 구상하고 그에 상응하는 코드를 만들 필요가 없다는 것을 의미
- AutoML을 적용할 경우 기존의 머신러닝보다 더 높은 정확도를 지닌 결과물을 기대할 수 있음
- 인간이 머신러닝 작업을 수행한다면 현실적으로 모든 알고리즘과 방법론을 직접 비교하기엔 무리가 있음
- 시간적 여유가 부족하고, 모든 방법론을 일일이 비교하기에는 양이 너무 방대하기 때문
- 간이 선택한 몇몇 방법론 vs 존재하는 모든 방법론이라는 잣대로 비교한다면, 자연스레 AutoML을 적용했을 때 더욱 정밀한 머신러닝 결과물을 얻을 가능성이 높음

## Auto ML

- 현재 AutoML 기술로 해결하고자 하는 문제는 크게 두가지가 있음

**1) 모델을 학습하고 평가할 때 다양한 알고리즘들과 연관된 하이퍼 파라미터들을 실험하고 성능을 비교하여 최상의 성능을 갖는 모델을 찾는 과정을 자동화 하는 문제 (Combined Algorithm Selection and Hyper-parameter optimization, so called **CASH**)**

**2) 인공 신경망 기술을 활용함에 있어서 문제에 적합한 architecture를 찾는 과정을 자동화 하는 문제 (Neural Architecture Search, so called **NAS**)**

## Combined Algorithm Selection and Hyper-parameter (CASH) optimization

- 모델 개발자는 학습을 위한 준비를 마친 후 주어진 컴퓨팅 자원과 시간을 최대한 활용하여 최상의 모델을 만들어야함
- 문제에 적합한 알고리즘을 찾고, 나아가 그 알고리즘의 성능을 극대화 할 수 있는 하이퍼파라미터를 찾는 작업이 수반

→ 일종의 최적화(optimization) 문제로써 소위 줄여서 CASH 문제라고 부름(Chris et al., 2012).

- 해당 문제를 풀기위한 방법으로 다양한 방법들이 연구되어 옴
  - 전통적인 최적화 알고리즘을 적용하는 방법
  - 강화학습의 문제로 접근하는 방법
  - 최적화 과정에서 관찰되는 탐색 결과를 학습하여 (i.e. learning response function/surrogate model) 다음 탐색 방향을 결정하는데 이용하는 Bayesian optimization (BO) 방법

→ 최근의 연구들은 BO의 framework 내에서 다양한 surrogate model 을 적용하거나, meta-learning, hyperband의 아이디어를 혼합한 방법들이 연구

### ▼ CASH 문제를 풀기 위한 주요 방법들

- Conventional optimization:** grid search, random search, genetic algorithm, simulated annealing
- Reinforcement learning:** Successive-halving, Hyperband
- Bayesian optimization:** Gaussian Processes (GP), Sequential Model-based Algorithm Configuration (SMAC), FLASH: Fast Bayesian Optimization for Data Analytic Pipelines, FABOLAS
- Hybrid:** BOHB, Maxtrix Factorization

## Neural Architecture Search (NAS)

- Neural-net 기반의 머신 러닝 알고리즘을 사용하기 위해서는 neural architecture 를 풀고자 하는 문제에 적합하도록 잘 설계해야 함
- NAS는 이 과정을 사람이 직접하지 않고 자동으로 하는 것을 목표로 연구되고 있으며, 몇가지 문제 (e.g. image classification, object detection, or segmentation)에서는 NAS 가 기존 사람에 의해 설계된 모델의 성능이 뛰어날 수 있음이 보고되고 있음 (Elsken et al., 2019).

▼ NAS 문제를 풀기 위한 방법은 크게 search space를 어떻게 정의하고, 어떤 search 알고리즘을 적용하느냐에 따라 나누어 질 수 있음

- search 알고리즘:

탐색 알고리즘을 알아봅시다!!:)

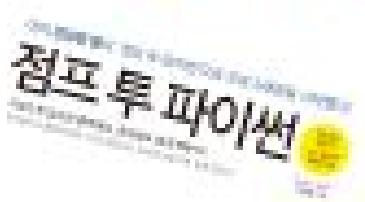
탐색 알고리즘 (Search algorithm) 알고리즘의 종류중에서도 탐색 및 정렬은 가장 많이 사용되기 때문에 초기에 배워두어야 합니다. 검색 엔진은 탐색 알고리즘을 사용한다. 검색의 다른 명칭이 탐색입니다. 검색 엔진에는 엔진이라는 이름이 붙어

 <https://velog.io/@dlrmsghks7/%ED%83%90%EC%83%89-%EC%95%8C%EA%B3%A0%EB%A6%AC%EC%A6%98%EC%9D%84-%EC%95%8C%EC%95%84%EB%B4%85%EC%8B%9C%EB%8B%A4>

▼ CASH 문제와 유사하게, random search, Bayesian optimization, evolutionary algorithm, reinforcement learning 등의 search 알고리즘을 적용하는 방법들이 연구되어 왔으며, 근래에는 search space를 continuous variable로 relaxation 하여 gradient-based optimization 방법을 적용하여 탐색을 더 효율화하는 연구도 진행

점프 투 파이썬

점프 투 파이썬 오프라인 책(개정판) 출간 !! (2019.06) \*\*\* [책 구입 안내]  
(<https://wikidocs.net/4321>) 이 책은 파이썬 ...

 <https://wikidocs.net/22149>

▼ NAS 문제를 풀기 위한 주요 방법들

- **Evolutionary algorithm:** [NEAT](#), [AmeobaNet](#), [Hierarchical NAS](#), [JASQNet](#)
- **Reinforcement learning:** [NAS with RL](#), [NASNet](#), [MnasNet](#), [ENAS](#)
- **Relaxation:** [One-Shot NAS](#), [DARTS](#), [ProxylessNAS](#)

## Auto ML 적용을 위해 필요한 것



## 1) 머신러닝에 대한 이해

- AutoML이 각 머신러닝 절차 속에서 작업을 수행한 이후에 내놓는 결과물은 어떤 것이며, 그 과정에 어떤 비교 실험을 진행했는지 이해해야 하기 때문
- 모든 AutoML은 각 작업이 모듈화 된 기능으로 존재
- 클릭 몇 번 혹은 코드 한 줄로 처음부터 끝까지 모든 단계를 완벽히 자동화할 수는 없음
- 머신러닝 각 절차에 따라 데이터 전처리에 필요한 기능, 알고리즘 선택에 필요한 기능이 따로 존재하여 데이터 분석가가 이를 조합하여 활용

## 2) 시스템과 자원

- AutoML을 현실에 적용하기 위해서는 시스템이나 자원의 뒷받침이 필요
- 이는 AutoML의 종류에 따라 다르지만, 기본적으로 굉장히 많은 알고리즘을 모두 수행하며 그 결과를 비교해 단 하나의 알고리즘을 선정하는 것이 기본 작동 원리
- 온 알고리즘을 수행하기 위해서는 높은 컴퓨터 자원이 준비되어야 함
- 특히 AutoML을 적용하려면 비교도 할 수 없이 높은 사양의 컴퓨터가 필요

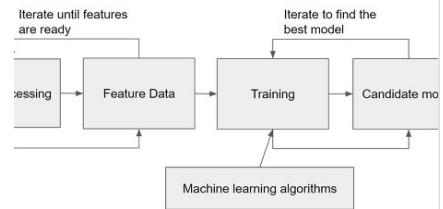
## Auto ML 종류와 현황

- 현재 적용되고 있는 AutoML은 크게 세 가지로 분류가 가능

### AutoML(Automated Machine Learning)

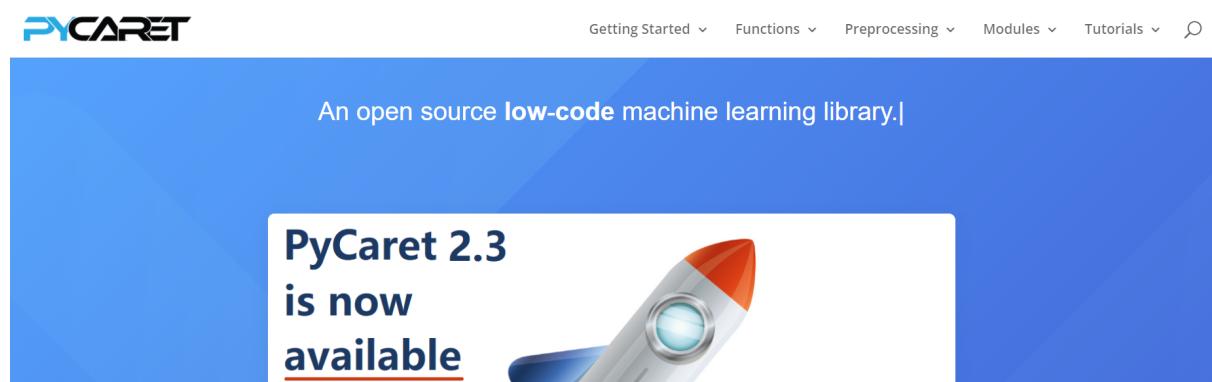
모델을 선정하고 학습하는데 소모적인 부분과 반복적인 기계학습 모델 개발 단계를 자동화하는 프로세스를 말한다. ML 모델 개발에 많은 리소스와 많은 모델을 생성하고 각 도메인 영역의 지식과 시간이 필요한데 준비된 ML 모델을 매우 쉽고 효과적으로 수행

⇒ <https://dbrang.tistory.com/1533>



## 1) OSS(Open Source Software)

- 첫 번째는 OSS로 무료로 제공되는 소프트웨어를 통해 AutoML을 적용하는 방법
- OSS 방식은 파이썬, R 등 각 프로그래밍 환경에 라이브러리 형태로 AutoML 기능이 내장되어, 이를 사용자가 그대로 적용할 수 있는 방식
- 쉽게 이해하자면 파이썬과 같이 머신러닝에 자주 이용되는 컴퓨터 언어 속에 AutoML 기능을 삽입한 다음 이용하는 방식



- OSS 방식을 통해 AutoML을 적용한다면 무료로 이용할 수 있다는 큰 장점이 존재
- 한 다른 AutoML 종류와는 다르게 직접 코드를 불러와 수행하기 때문에, 개인화(Customization)하여 AutoML을 수행하기에 가장 최적화되어 있음
- 코딩 작업에 인간의 관여가 상대적으로 많은 편이기에 완전한 자동화를 진행하거나, 쉽게 AutoML을 바로 적용하기에는 한계가 있음

## 2) Cloud Provider solutions

- 두 번째 방법은 클라우드 환경을 통해 AutoML 기능을 이용하는 Cloud Provider Solutions
- 클라우드란 개인 혹은 기업 각자의 컴퓨터 장치를 이용하는 것이 아닌, 클라우드라는 환경 속에서 제공되는 컴퓨터 자원을 이용해 작업을 수행하는 방식을 의미

- 평소 해당 클라우드 시스템을 활용해 작업을 수행하고 있다면, 가장 적용하기 쉬운 방법은 클라우드에서 직접 제공하는 솔루션을 이용



## Google Cloud Platform

- 클라우드 시스템 기반의 솔루션들을 이용한다면, AutoML을 위한 준비물 중 시스템과 자원에 대한 부분은 자연스럽게 해결
- 클라우드 시스템 특성상, 직접 소유하고 있는 컴퓨터가 아닌 웹 상에 존재하는 거대 기업의 컴퓨터 자원을 활용하기 때문
- 지만 OSS 방식과는 반대로 클라우드 시스템에서 제공되는 AutoML을 그대로 이용하여 사용은 쉽지만, 작업 과정을 투명하게 관찰하거나 AutoML을 개인화 하기에는 단점이 있음

### 3) AutoML Platforms

- 마지막은 AutoML만을 위해 개발된 플랫폼을 이용하는 방식인 AutoML Platform
- AutoML만을 위해 하나의 프로그램을 새로 개발하여, 이를 활용하는 방식
- 해당 방식은 일반적으로 기업에서 AutoML을 활용할 때 가장 많이 이용
- 직접 AutoML을 위해 개발된 솔루션 혹은 프로그램의 비용을 지불하고, 하나의 일관된 분석 환경으로 사용

- 앞선 두 방식과는 다르게 AutoML만을 위해 새롭게 솔루션을 구매해 사용하는 방식으로, 비용 측면에서 단점이 될 수 있음
- 또한 프로그램 자체를 이용하는 것으로 직접 코드를 조작해, 개인화된 AutoML을 구성하기에는 OSS 방식에 비해 한계가 있음
- 하지만 AutoML만을 위해 개발된 솔루션이어서, 가장 쉽고 간단하게 AutoML을 구현할 수 있게 설계

## AutoML의 한계점

### 1) 비용적 한계

- AutoML의 종류 중 OSS 방식을 제외한다면 각각 클라우드 사용 비용, 솔루션 구매 비용 등을 지불
- 개인이 지불하기에는 당연히 부담스럽고, 기업 역시 지속적으로 AutoML을 위해 비용을 쓰는 것은 부담이 될 수 있음
- AutoML은 기본적으로 사람이 할 수 있는 일을 자동화해주는 작업이기에 비용을 지불하려면 심리적인 부담도 따름
- 자동화 솔루션을 거꾸로 말하면 '사람이 할 수 있는 일을 굳이 비용을 지불하며 처리하는 것'
- 또한 앞서 언급한 OSS에 해당하는 AutoML 구현 방식도 비용적 한계에서 완전히 자유로운 것은 아님
- 비록 프로그래밍에 구현되어 있는 코드를 그대로 활용하는 것뿐이지만, 이 경우는 온전히 개인이 소유하고 있는 컴퓨터 자원을 활용하고 있기 때문
- AutoML은 기본적으로 높은 수준의 컴퓨터 자원을 필요로 하며, OSS 방식의 AutoML을 사용한다는 것은 컴퓨터 자원 구축은 순전히 개인이 온전히 부담한다는 의미

### 2) 기능적 한계

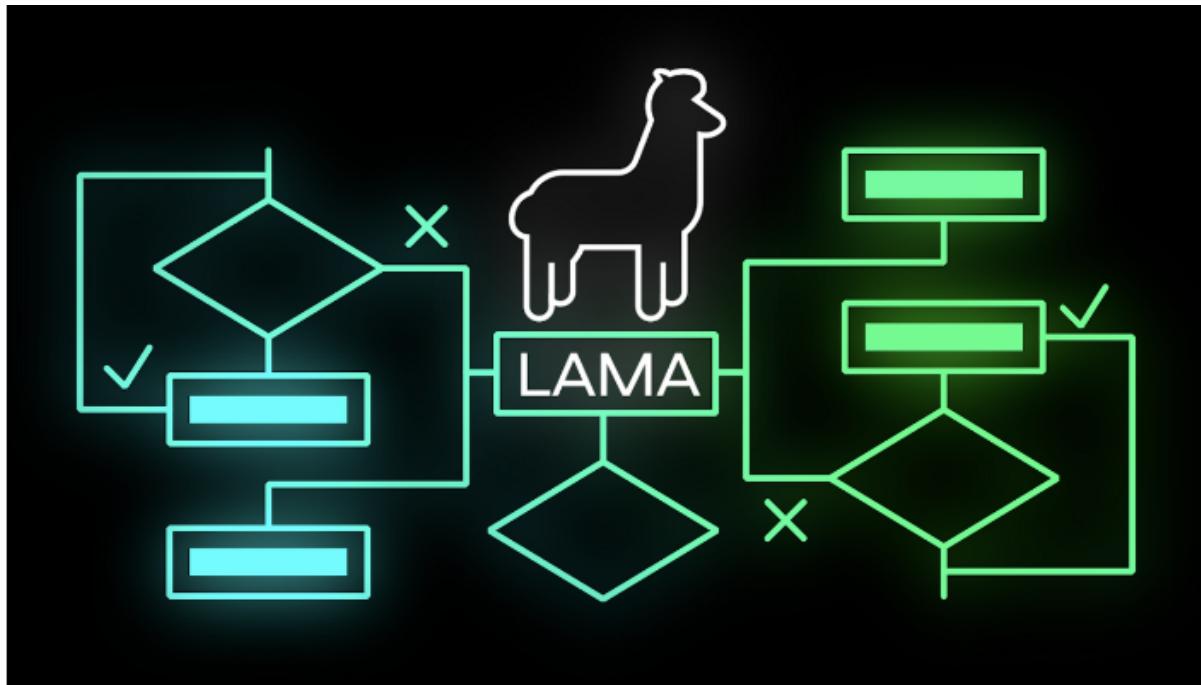


- 현재까지의 AutoML은 정확도 측면에서 사람이 만드는 머신러닝보다 품질이 떨어지는 경향이 있고, 모델 완성에 걸리는 시간도 더 많음
- 정확도가 상대적으로 낮은 현상은 보통 인간의 도메인 지식 때문이라고 해석
- 인간은 도메인 지식을 활용하여 결과 예측에 유효하게 작용하는 변수를 컴퓨터에게 선별하여 입력해주는 반면, AutoML에서는 단순히 처음 주어진 데이터만을 활용하여 머신러닝을 진행함으로써 정확도에 한계를 보임
- 따라서 단 1%라도 정확도가 높은 작업이 필요하다면, AutoML보단 사람이 직접 수행을 하는 것이 더 타당하다는 의견이 많음
  
- 또한 AutoML은 다양한 경우의 수를 직접 비교하여 알고리즘을 선정하기 때문에, 매우 많은 시간이 소요
- 컴퓨터가 작업을 수행하는 동안 인간이 다른 일을 할 수 있다는 점에선 효율적일 수도 있지만, 소요되는 시간은 더 많기 때문에 결국은 효율적이지는 않다는 역설적인 상황

## Auto ML OSS

### LightAutoML(=LAMA)

- LightAutoML(LAMA)는 Code에서 가장 많이 사용되는 AutoML
- 현재 풀 수 있는 문제는 이진 분류 및 다중클래스 분류, 회귀 문제를 풀 수 있음



▼ Code

```

!python3 -m pip install -q lightautoml

## import packages
from lightautoml.automl.presets.tabular_presets import TabularAutoML
from lightautoml.tasks import Task

## train model
train['target'] = dt.Frame(target)

model_laml = TabularAutoML(
    task=Task('reg'),
    timeout=MAX_MODEL_RUNTIME_SECS
)

model_laml.fit_predict(train_data=train.to_pandas(), roles={'target': 'target'})

del train['target']

## generate predictions
preds_laml = model_laml.predict(test.to_pandas()).data.ravel()

## create submission
submission = dt.Frame(
    id=test_ids,
    loss=preds_laml
)

submission.head()

## save submission
submission.to_csv(PATH_LAML_SUBMISSION)

```

## H2O AutoML



- H2O AutoML도 LAMA 이후로 두번째로 많이 보이는 AutoML코드,
  - AutoML 외에도 머신러닝 전반의 파이프라인 제공
- ▼ Code

```

## import packages
import h2o
from h2o.automl import H2OAutoML

## prepare data
h2o.init()

h2o_train = h2o.H2OFrame(train.to_pandas())
h2o_test = h2o.H2OFrame(test.to_pandas())

h2o_train['target'] = h2o.H2OFrame(target)

## train model
features = [x for x in h2o_train.columns if x not in ['id', 'target']]

model_h2oaml = H2OAutoML(
    max_runtime_secs=MAX_MODEL_RUNTIME_SECS,
    stopping_metric='RMSE',
    sort_metric='RMSE'
)

model_h2oaml.train(x=features, y='target', training_frame=h2o_train)

## check leaderboard
model_h2oaml.leaderboard

## generate predictions
preds_h2oaml = dt.Frame(model_h2oaml.leader.predict(h2o_test).as_data_frame().predict)

## create submission
submission = dt.Frame(
    id=test_ids,
    loss=preds_h2oaml
)

submission.head()

## save submission
submission.to_csv(PATH_H2OAML_SUBMISSION)

```

## PyCaret

# **PyCaret 2.3**

## **is now**

### **available**



- PyCaret는 데이콘 등 대회에서 AutoML로 자주 등장하는 라이브러리
  - 다양한 모델 예측 및 하이퍼파라미터 튜닝, SHAP등을 이용한 시각화 제공
- ▼ Code

```

! pip install pycaret

from pycaret.regression import setup, compare_models, blend_models, finalize_model, predict_model

def pycaret_model(train, target,test, n_select, fold, opt):
    print('Setup Your Data....')
    setup(data=train,
          target=target,
          silent= True,
          use_gpu = True,
          feature_selection = True,
          normalize = True)

    print('Comparing Models....')
    best = compare_models(sort=opt, n_select=n_select, fold = fold,include=['xgboost','lightgbm','catboost'])

    print('Blending Models....')
    blended = blend_models(estimator_list= best, fold=fold, optimize=opt)
    pred = predict_model(blended)

    print('Finalizing Models....')
    final_model = finalize_model(blended)
    print('Done...!!!!')

    pred_test = predict_model(final_model, test)
    re = pred_test['Label']

    return re

submission['loss'] = pycaret_model(train,'loss',test, 3, 10, 'RMSE')
submission.to_csv('submission.csv',index=False)

```

## AutoGluon



### ▼ Code

```

## import packages
from autogluon.tabular import TabularPredictor

## train model
train['target'] = dt.Frame(target)

model_autogluon = TabularPredictor(
    problem_type='regression',
    label='target'
)

model_autogluon.fit(train_data=train.to_pandas(), time_limit=MAX_MODEL_RUNTIME_SECS)

del train['target']

## check leaderboard
model_autogluon.leaderboard()

## generate predictions
preds_autogluon = dt.Frame(model_autogluon.predict(test.to_pandas()))

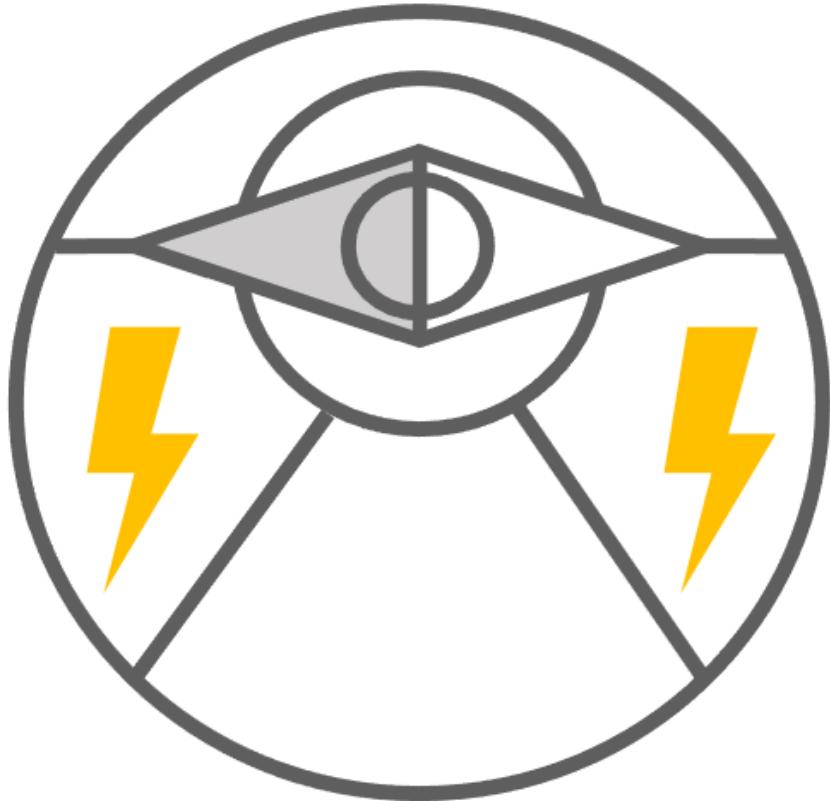
## create submission
submission = dt.Frame(
    id=test_ids,
    loss=preds_autogluon
)

submission.head()

## save submission
submission.to_csv(PATH_AUTOGLUON_SUBMISSION)

```

## FLAML



# FLAML

▼ Code

```
## import packages
from flaml import AutoML

## train model
model_flaml = AutoML()

model_flaml.fit(X_train=train.to_pandas(), y_train=target,
                 time_budget=MAX_MODEL_RUNTIME_SECS, task='regression', metric='mse')

## generate predictions
preds_flaml = dt.Frame(model_flaml.predict(test.to_pandas()))

## create submission
submission = dt.Frame(
    id=test_ids,
    loss=preds_flaml
)

submission.head()

## save submission
submission.to_csv(PATH_FLAML_SUBMISSION)
```

## MLJAR



- MLJAR은 표 형식의 데이터를 위한 자동화된 기계 학습 도구
- ▼ Code

```

## import packages
from supervised import AutoML

## train model
model_mljar = AutoML(
    total_time_limit=MAX_MODEL_RUNTIME_SECS,
    eval_metric='rmse',
    results_path='./mljar'
)

model_mljar.fit(X=train.to_pandas(), y=target)

## check leaderboard
model_mljar.get_leaderboard()

## generate predictions
preds_mljar = dt.Frame(model_mljar.predict(test.to_pandas()))

## create submission
submission = dt.Frame(
    id=test_ids,
    loss=preds_mljar
)

submission.head()

## save submission
submission.to_csv(PATH_MLJAR_SUBMISSION)

```

## AutoKeras



- AutoKeras는 Keras를 기반으로 하는 AutoML 시스템
  - AutoKeras의 목표는 모든 사람이 기계 학습을 이용할 수 있도록 하는 것
- ▼ Code

```
%pip install autokeras

import autokeras as ak

search = ak.StructuredDataRegressor(max_trials=5, loss='mean_squared_error', objective="val_loss")
search.fit(X_train, y_train, verbose=1, validation_data=(X_valid, y_valid), use_multiprocessing=True, epochs=10,)

lm=search.predict(test_data)
final_pred=pd.read_csv("../input/tabular-playground-series-aug-2021/sample_submission.csv")
final_pred["loss"] = lm
final_pred.to_csv('submission_basicnn.csv', index=False)
```

## PyCaret 사용하기

- pycaret을 활용하면 여러 모델의 성능 비교 뿐만아니라 하이퍼파라미터 tunning, 여러 모델을 blending한 모델을 만들 수 있음
- PyCaret은 기존에 있던 Scikit-learn, XGBoost, LightGBM, spaCy 등 여러가지 머신러닝 라이브러리를 ML High-Level API로 제작한 라이브러리
- 단 몇 줄만에 데이터 분석 및 머신러닝 모델 성능 비교까지 가능하고, Log를 생성하여 이력을 남겨줌

## 머신러닝 기법 종류

- [Classification](#)
- [Regression](#)
- [Clustering](#)
- [Anomaly Detection](#)
- [Natural Language Processing](#)

## Pycaret 적용 순서

### 1) 데이터 준비

- pd.dataframe 으로 로드되어진 데이터를 머신러닝에 사용할 수 있도록 로드 및 전처리하는 기능
- 데이터 전처리 단계에서 적용하는 기법들에 대해서 파라미터로 간단하게 사용할 수 있도록 구현

```
from pycaret.classification import setup
clf1 = setup(data, target='Purchase', session_id=123, log_experiment=True, experiment_name='exam_juice')
```

- pycaret에 구현된 머신러닝 기법들을 사용하려면 무조건 런타임에 먼저 setup이 로드되어야 함
- Classification 예제이며 다른 모듈도 사용법이 크게 다르진 않음 (전처리 파라미터는 머신러닝 기법 별로 상이할 수 있다.)

## 모델 생성 및 비교

- 데이터 준비(setup) 이후, 머신러닝 모델을 선언해서 사용하거나, 전처리한 데이터셋에 맞는 모델을 비교해볼 수 있음
- `model()`
  - 각각 머신러닝 기법(Classification, Regression 등)에 따라 구현된 모델들을 나열
- `compare_models()`
  - Setup 된 데이터를 각각 머신러닝 모델에 적용 후 비교
- `create_model()`
  - `model()`에 적힌 머신러닝 모델을 선택해서 생성한

## 모델 최적화

- 2번째에서 compare\_models 나 create\_model을 사용해서 각각의 모델들을 생성하거나 비교
- 결과를 바탕으로 성능이 좋은 모델들을 조합하여 실험해볼 수 있는 모듈을 제공
- `tune_model()`
  - 모델의 하이퍼파라미터를 최적화하는 모듈
  - 하이퍼파라미터 반복 횟수나 최적화할 매트릭을 선택할 수 있음
- `ensemble_model()`
  - ensemble 기법을 구현한 모듈
  - bagging과 boosting을 파라미터에서 선택할 수 있음
- `blend_models()`
  - Voting 알고리즘을 구현한 모듈
  - compare\_models()에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용(n\_select)시켜서 사용할 수 있음
- `stack_models()`

- stacking ensemble 방법을 구현한 모듈
- compare\_models()에서 성능이 잘 나온 모델들을 선택하는 파라미터를 적용(n\_select)시켜서 사용할 수 있음

## 학습된 모델 분석

- pycaret은 위에서 소개된 방법을 사용하여 학습한 이후에 데이터셋이 잘 학습되었는지 검증을 도와주는 모듈을 제공
- plot\_model()
  - 학습한 모델에 대한 각종 지표들을 시각화한 플롯을 그려주는 모듈
  - auc, threshold, confusion matrix 등 약 15가지 이상의 다양한 플롯들을 지원
- interpret\_model()
  - 모델이 예측한 결과에 대해서 각 파라미터들이 얼마나 영향을 줬는지 시각화해서 보여줌
  - SHAP(SHapley Additive exPlanations) 를 사용하여 plot을 그려줌
- assign\_model()
  - unsupervised 류의 머신러닝 기법(anomaly detection, clustering, NLP)에 대한 머신러닝 결과값(라벨)을 데이터셋에 부쳐줌
- calibrate\_model()
- evaluate\_model()
  - 모델 분석 후 각 플롯을 볼 수 있도록 사용자 인터페이스를 제공
  - 한방에 플롯을 전부 띄우고 싶다면 plot\_model()보다는 evaluate\_model()을 사용
- get\_leaderboard()
  - setup 이후 훈련된 모든 모델을 출력
- dashboard()
  - explainerdashboard 를 사용하여 모델 분석에 대한 사용자 인터페이스를 제공
  - evaluate\_model() 과 비교해서 사용
- eda()
  - auto\_viz 를 사용하여 데이터셋 분석 결과를 제공
- check\_fairness()
  - 데이터셋에 각 feature들에 대해 measure를 확인해볼 수 있도록 모듈을 제공
  - 예를 들어 성별이 feature로 있을 때 check\_fairness를 적용시키면, 성별 별로 얼마만큼의 데이터셋이 존재하고, 결과가 어떻게 되는지 제공

# PyCaret 사용

- pycaret을 활용하면 여러 모델의 성능 비교 뿐만 아니라 하이퍼파라미터 tunning, 여러 모델을 blending한 모델을 만들 수 있

## pycaret 설치

- pycaret을 설치

```
!pip install pycaret
```

- 모델 비교시 catboost가 없다면 다음과 같이 설치

```
!pip install pycaret[full]
```

## 데이터 로드

- 데이터는 pycaret에서 제공하는 'credit' 데이터를 사용
- 해당 데이터를 불러와줌

```
from pycaret.datasets import get_data
dataset = get_data('credit')
```

- train/test 데이터로 데이터를 분리

```
train = dataset.sample(frac=0.95, random_state=786)
test = dataset.drop(train.index)
train.reset_index(inplace=True, drop=True)
test.reset_index(inplace=True, drop=True)
```

## 데이터 설정

- pycaret을 사용하기 전에 pycaret에 맞게 데이터를 설정
- set\_up() 함수를 사용하며, 기본적으로 data와 target을 입력
- 입력 후 column에 대한 자료형이 출력되며 enter를 치면 data가 설정

```
from pycaret.classification import *
exp_clf = setup(data = train, target = 'default', session_id=123)
```

#### **set\_up():** pycaret을 사용하기 위한 data setting

- session\_id: random\_state와 같은 개념으로 같은 결과가 나올 수 있게 seed를 고정
- data: train 데이터를 입력
- target = target 변수 이름을 입력
- data 설정값이 다음과 같이 출력

	Description	Value
0	session_id	123
1	Target	default
2	Target Type	Binary
3	Label Encoded	None
4	Original Data	(22800, 24)
5	Missing Values	False
6	Numeric Features	14
7	Categorical Features	9
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None
11	Transformed Train Set	(15959, 88)
12	Transformed Test Set	(6841, 88)
13	Shuffle Train-Test	True
14	Stratify Train-Test	False
15	Fold Generator	StratifiedKFold

## 모델 비교

- 여러 모델을 적합하여 성능을 비교하는 단계

```
best_model = compare_models()
```

- 총 14개 모델의 적합 결과가 다음과 같이 출력

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
ridge	Ridge Classifier	0.8254	0.0000	0.3637	0.6913	0.4764	0.3836	0.4122	0.045
lda	Linear Discriminant Analysis	0.8247	0.7634	0.3755	0.6794	0.4835	0.3884	0.4132	0.280
gbc	Gradient Boosting Classifier	0.8226	0.7789	0.3551	0.6806	0.4664	0.3725	0.4010	5.379
ada	Ada Boost Classifier	0.8221	0.7697	0.3505	0.6811	0.4626	0.3690	0.3983	1.250
lightgbm	Light Gradient Boosting Machine	0.8210	0.7750	0.3609	0.6679	0.4683	0.3721	0.3977	0.391
rf	Random Forest Classifier	0.8199	0.7598	0.3663	0.6601	0.4707	0.3727	0.3965	2.880
et	Extra Trees Classifier	0.8092	0.7377	0.3677	0.6047	0.4571	0.3497	0.3657	2.155
lr	Logistic Regression	0.7814	0.6410	0.0003	0.1000	0.0006	0.0003	0.0034	0.987
dummy	Dummy Classifier	0.7814	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.029
knn	K Neighbors Classifier	0.7547	0.5939	0.1763	0.3719	0.2388	0.1145	0.1259	0.812
dt	Decision Tree Classifier	0.7293	0.6147	0.4104	0.3878	0.3986	0.2242	0.2245	0.355
svm	SVM - Linear Kernel	0.7277	0.0000	0.1017	0.1671	0.0984	0.0067	0.0075	0.407
qda	Quadratic Discriminant Analysis	0.5098	0.5473	0.6141	0.2472	0.3488	0.0600	0.0805	0.152
nb	Naive Bayes	0.3760	0.6442	0.8845	0.2441	0.3826	0.0608	0.1207	0.045

**compare\_models():** 다양한 모델 적합 후 성능 비교

- fold: cross\_validation의 fold를 지정 (default = 10)
- sort: 정렬기준 지표 설정
- n\_select: 상위 n개의 모델 결과만 출력

## 모델 적합

- 다양한 모델의 적합 결과를 확인하였는데, 이번에는 하나의 모델의 적합 결과를 보는 방법

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

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8133	0.7673	0.3610	0.6269	0.4582	0.3551	0.3749
1	0.8239	0.7615	0.3782	0.6735	0.4844	0.3882	0.4117
2	0.8258	0.7708	0.3467	0.7076	0.4654	0.3756	0.4098
3	0.8177	0.7605	0.3725	0.6436	0.4719	0.3710	0.3913
4	0.8208	0.7642	0.3725	0.6599	0.4762	0.3780	0.4006
5	0.8283	0.7638	0.3954	0.6866	0.5018	0.4070	0.4297
6	0.8127	0.7647	0.3582	0.6250	0.4554	0.3522	0.3721
7	0.8283	0.7390	0.3553	0.7168	0.4751	0.3861	0.4202
8	0.8108	0.7496	0.3610	0.6146	0.4549	0.3496	0.3678
9	0.8176	0.7565	0.3621	0.6462	0.4641	0.3645	0.3867
Mean	0.8199	0.7598	0.3663	0.6601	0.4707	0.3727	0.3965
SD	0.0062	0.0089	0.0131	0.0335	0.0139	0.0172	0.0202

**create\_model()**: 하나의 모델 적합

- fold: cross\_validation의 fold 지정 (default = 10)

## Tunning

- tune\_model() 함수를 사용해서 모델의 하이퍼파라미터 튜닝을 진행

```
tuned_rf = tune_model(rf)
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8158	0.7508	0.3181	0.6647	0.4302	0.3363	0.3689
1	0.8283	0.7675	0.3295	0.7419	0.4563	0.3719	0.4152
2	0.8139	0.7337	0.3181	0.6529	0.4277	0.3321	0.3628
3	0.8246	0.7588	0.3095	0.7347	0.4355	0.3514	0.3976
4	0.8170	0.7567	0.3438	0.6557	0.4511	0.3539	0.3805
5	0.8258	0.7506	0.3324	0.7205	0.4549	0.3676	0.4067
6	0.8170	0.7530	0.3324	0.6629	0.4427	0.3474	0.3771
7	0.8221	0.7507	0.3381	0.6901	0.4538	0.3621	0.3951
8	0.8177	0.7201	0.2980	0.6933	0.4168	0.3286	0.3699
9	0.8207	0.7484	0.3132	0.6987	0.4325	0.3439	0.3831
Mean	0.8203	0.7490	0.3233	0.6915	0.4402	0.3495	0.3857
SD	0.0045	0.0126	0.0135	0.0310	0.0129	0.0140	0.0165

- tuned\_rf를 호출하면 튜닝 결과를 확인할 수 있

```
RandomForestClassifier(bootstrap=False, ccp_alpha=0.0, class_weight={},
                      criterion='entropy', max_depth=5, max_features=1.0,
                      max_leaf_nodes=None, max_samples=None,
                      min_impurity_decrease=0.0002, min_impurity_split=None,
                      min_samples_leaf=5, min_samples_split=10,
                      min_weight_fraction_leaf=0.0, n_estimators=150,
                      n_jobs=-1, oob_score=False, random_state=123, verbose=0,
                      warm_start=False)
```

### tune\_model(model): 모델의 하이퍼파라미터 튜닝

- optimize: 평가 metric 지정

## Blending

- blend\_models() 함수를 사용하면 여러 모델들을 혼합하여 새로운 모델을 생성
- 모델을 하나씩 생성해서 blend해도 되고 compare\_model을 사용하여 생성한 모델을 사용해서 blend 할 수 있음

```
# 방법 1
dt = create_model('dt')
```

```

rf = create_model('rf')

blender_2 = blend_models(estimator_list = [dt, rf])

# 방법 2
best_model_5 = compare_models(n_select=5)
blender_5 = blend_models(best_model_5)

```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
<b>0</b>	0.8170	0.0	0.3582	0.6477	0.4613	0.3619	0.3849
<b>1</b>	0.8308	0.0	0.3954	0.7005	0.5055	0.4128	0.4374
<b>2</b>	0.8227	0.0	0.3438	0.6897	0.4589	0.3668	0.3986
<b>3</b>	0.8277	0.0	0.3668	0.7033	0.4821	0.3908	0.4206
<b>4</b>	0.8208	0.0	0.3582	0.6684	0.4664	0.3703	0.3964
<b>5</b>	0.8333	0.0	0.4011	0.7107	0.5128	0.4215	0.4466
<b>6</b>	0.8227	0.0	0.3696	0.6719	0.4769	0.3808	0.4055
<b>7</b>	0.8283	0.0	0.3524	0.7193	0.4731	0.3846	0.4196
<b>8</b>	0.8183	0.0	0.3438	0.6630	0.4528	0.3568	0.3844
<b>9</b>	0.8257	0.0	0.3793	0.6804	0.4871	0.3921	0.4165
<b>Mean</b>	0.8247	0.0	0.3669	0.6855	0.4777	0.3838	0.4111
<b>SD</b>	0.0051	0.0	0.0189	0.0219	0.0187	0.0202	0.0199

- **blend\_models(models):** 여러 모델들을 혼합한 새로운 모델을 생성

## 예측

- finalize\_model() 함수로 모델을 설정하면, cross\_validation을 사용하여 적합한 모델을 전체 데이터로 마지막으로 학습
- 마지막 모델을 설정한 후에 predict\_model()을 통해 예측

```

final_model = finalize_model(blender_5)
prediction = predict_model(final_model, data = test)

```

- 예측 결과는 'Label'변수에 다음과 같이 저장

```

0      0
1      0
2      0
3      1
4      0
..
1195   0
1196   0
1197   0
1198   0
1199   0
Name: Label, Length: 1200, dtype: int64

```

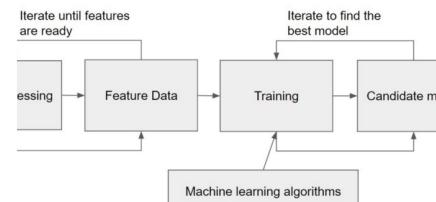
- **finalize\_model()**: 최종 모델로 설정 후 마지막 학습 진행
- **predict\_model()**: 예측 결과를 'Label' 변수에 저장

## 참고자료

- 블로그

**AutoML 이란 무엇일까?**

머신러닝 모델을 개발하고 실제 운영에 도입하기에는 수많은 과정을 거치게 됩니다. 전지전능한 인공지능이 존재하여 컴퓨터와 데이터만 주어지면 모든 과정을 알아서 해결해 줄 수 있다면 좋겠지만, 현실의 머신러닝 모델링은 문제 정의 과정에서부터 데이터  
 <https://medium.com/daria-blog/automl-%EC%9D%B4%EB%9E%80-%EB%AC%B4%EC%97%87%EC%9D%BC%EA%B9%8C-1af227af2075>



```

graph LR
    A[Data Processing] --> B[Feature Data]
    B --> C[Training]
    C --> D[Candidate model]
    D --> E[Machine learning algorithms]
    E --> B
    B --> F[Feature Engineering]
    F --> G[Model Selection]
    G --> H[Hyper-parameters Tuning]
    H --> C
    C --> I[Iterate until features are ready]
    I --> B
    I --> J[Iterate to find the best model]
    J --> D
  
```

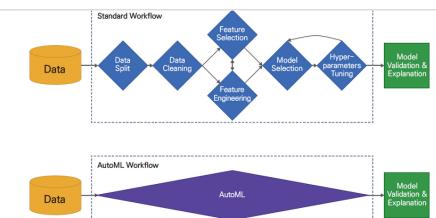
**[BIG DATA] AutoML특집 (1부)- AutoML이란 무엇인가**

전 세계 기업 및 조직들의 가장 중요한 화두는 데이터입니다. 데이터는 이미 폭발적으로 증가하고 있으며, AI, 빅데이터, IoT, 5G 등에 대한 투자가 더욱 가속화되면서 데이터 증가 속도는 더욱 탄력을 받게 될 것입니다. 새로운 비즈니스 가치를 찾고 경쟁 우위  
 <https://kolonbenits-time.com/78>



**[AutoML] AutoML이란 무엇인가?**

이번에 회사 업무상 AutoML에 대해 정리해야 할 일이 있어, 온갖 자료를 다 끌어다가 정리해보고자 합니다. 특히, 제가 직접 AutoML 시스템을 개발해야 할 수도 있어서 상용화된 AutoML을 분석해보고, 구현을 위해 어떤 지식이 필요한지 여러 포스팅을 통해 좀  
 <https://gils-lab.tistory.com/m/65>



```

graph TD
    subgraph Standard_Workflow [Standard Workflow]
        A[Data] --> B[Data Split]
        B --> C[Data Cleaning]
        C --> D{Feature Selection}
        D --> E{Feature Engineering}
        E --> F{Model Selection}
        F --> G[Hyper-parameters Tuning]
        G --> H[Training]
        H --> I[Model Validation & Explanation]
    end
    subgraph AutoML_Workflow [AutoML Workflow]
        A[Data] --> J{AutoML}
        J --> K[Model Validation & Explanation]
    end
    D -.-> E
    F -.-> G
    J -.-> K
  
```

### 스마트하게 머신러닝 적용하는 법: ①AutoML이란? | 요즘IT

빅데이터를 활용하는 다양한 방법 중 단연코 가장 인기가 많은 것은 머신러닝입니다. 머신러닝은 빅데이터의 활용성을 비약적으로 발전하게 함과 동시에 AI의 근간이 되고 있습니다. 이에 따라 최근 머신러닝에 대한 교육이나 강의도 많이 개설되고 있습니다.

🔗 <https://yozm.wishket.com/magazine/detail/1267/>



### 스마트하게 머신러닝 적용하는 법: ②AutoML의 한계와 미래 | 요즘IT

1부에서는 AutoML이란 무엇인지, AutoML이 가진 장점과 주목받는 이유를 살펴봤습니다. 하지만 AutoML의 목적과 정의를 이해하는 것과 현실에서 AutoML 적용을 이야기하는 것은 다른 차원의 문제입니다. 이번 글에서는 AutoML이 어디까지 발전해왔고,

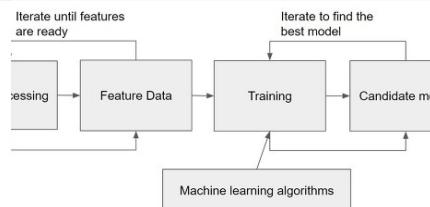
🔗 <https://yozm.wishket.com/magazine/detail/1282/>



### AutoML(Automated Machine Learning)

모델을 선정하고 학습하는데 소모적인 부분과 반복적인 기계학습 모델 개발 단계를 자동화하는 프로세스를 말한다. ML 모델 개발에 많은 리소스와 많은 모델을 생성하고 각 도메인 영역의 지식과 시간이 필요한데 준비된 ML모델을 매우 쉽고 효과적으로 수행

👉 <https://dbrang.tistory.com/1533>



### AutoML Tutorial : TPS (August 2021)

Explore and run machine learning code with Kaggle Notebooks | Using data from Tabular Playground Series - Aug 2021

🔗 <https://www.kaggle.com/code/rohanrao/automl-tutorial-tps-august-2021/notebook>



### 캐글, AutoML로 결과 제출

Tabular Playground Series - Aug 2021 | 월간 데이터 분석은? 매월 진행 중 데이터 분석 대회 및 특징에 대해 알아봅니다. 대회에 참여하여 베이스라인 코드를 작성하고 탐색적 데이터 분석(EDA)를 통해 데이터에 대한 이해도를 높입니다. 모델 학습과 예측

✍ <https://brunch.co.kr/@hansungdev/44>

	f93	f94	f95	f96	f97	f98
0	1.45751	0.696161	0.941764	1.828470	0.924090	0.700000
1	1.70644	-0.494699	-2.058300	0.819184	0.439152	0.700000
2	3.25339	0.337934	0.615037	2.216760	0.745268	0.700000
3	1.49425	0.517513	-10.222100	2.627310	0.617270	0.700000
4	1.73557	-0.476668	1.390190	2.195740	0.826987	0.700000

### pycaret을 사용한 데이터 분석, 머신러닝

pycaret은 기존에 있던 Scikit-learn, XGBoost, LightGBM, spaCy 등 여러가지 머신러닝 라이브러리를 ML High-Level API로 제작한 라이브러리다. 단 몇 줄만에 데이터 분석 및 머신러닝 모델 성능 비교까지 가능하고, Log를 생성하여 이력을 남겨준다.

🔗 <https://velog.io/@devseunggwan/Machine-Learning-pycaret%EC%9D%84-%EC%82%AC%EC%9A%99%ED%95%9C-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EB%B6%84%EC%84%9D>

velog

### [Python] pycaret을 사용하여 모델 선정하기

오늘은 여러 모델을 한번에 학습하여 비교해주는 라이브러리인 pycaret에 대해 알아보도록 하겠습니다. pycaret을 활용하면 여러 모델의 성능 비교 뿐만아니라 하이퍼파라미터 tuning, 여러 모델을 blending한 모델을 만들 수 있습니다. pycaret Binary

🔗 <https://velog.io/@ezoo0422/Python-pycaret%EC%9D%84-%EC%82%AC%EC%9A%99%ED%95%98%EC%97%AC-%EB%AA%A8%EB%8D%B8-%EC%84%A0%EC%A0%95%ED%95%98%EA%B8%BO>

4	Original Data	(22800, 24)
5	Missing Values	False
6	Numeric Features	14
7	Categorical Features	9
8	Ordinal Features	False
9	High Cardinality Features	False
10	High Cardinality Method	None

### [ML] Pycaret으로 ML모델 쉽게 만들기

ML Pycaret으로 ML모델 쉽게 만들기 pycaret이란 AutoML을 하게 해주는 파이썬 라이브러리입니다. scikit-learn 패키지를 기반으로 하고 있으며 Classification, Regression, Clustering, Anomaly Detection 등등 다양한 모델을 지원합니다. 공식 링크: <https://minimin2.tistory.com/137>

#	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	Spec
7	1	1.75	1.99	0.00	0.0	
9	1	1.75	1.99	0.00	0.3	
5	1	1.86	2.09	0.17	0.0	
7	1	1.69	1.69	0.00	0.0	
3	7	1.69	1.69	0.00	0.0	

### • 유튜브

#### [ICML 2018] AutoML

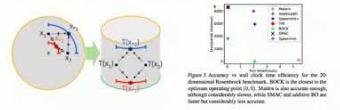
발표자: 김민규 (NAVER Search&Clova)<https://tv.naver.com/naverd2> 더욱 다양한 영상을 보시려면 NAVER Engineering TV를 참고하세요. 발표일: 2018.9. 이번 ICML에서의 경험을 공유하고자 합니다. 특히, AutoML에 관련된 ...

▶ <https://www.youtube.com/watch?v=F4K1Qc2F8fU>

#### Papers

##### Bayesian Optimization (BO)

###### - BOCK: Bayesian Optimization with Cylindrical Kernel



### • 깃허브