# PLAYDATA 2차 프로젝트 1팀

# Table of Contents

## PLAYDATA 데이터 엔지니어링 30기

김나영

김설아

백승민

장지원

최수빈

## 로또 번호 예측 앱

- 로또 당첨 번호를 통한 모델 설계
- Flask 서버에 모델 배포, 모델 예측
- Android 앱 설계

## 웹 광고 클릭률 예측 AI 경진대회

- 클릭 로그 데이터의 전처리
- 다양한 모델을 통한 학습
- 모델 합성을 통해 성능 향상
- 최종 모델 선택

## 기존 아이디어

### 1.로또 당첨 데이터를 통한 모델 학습

- 회차별 당첨 번호를 통해 모델을 생성
- 버리는 데이터가 많아지며 분포되는 예측값이 일정

CNN

[ 7 13 21 29 36 45]

MLP

[ 7 13 19 27 33 42]

RNN

[ 7 12 19 26 33 40]

RandomForest

[ 7 14 21 27 33 40]

TransFormer

[ 6 12 16 18 21 22]

## 기존 아이디어

## 2. 통계 모델 필터링

- 예측 결과의 예측률을 높이기 위해 이번 회차의 번호들의 통계를 예측하는 모델을 만들어 예측 결과들을 필터링하고자 함

- RNN, LSTM 모델을 통해 학습시키고 여러 회차의 통계(홀수 개수, 소수, 고수 개수)를 예측해보았으나 loss는 적지만 예측값이 계속 동일하여 필터링의 효과가 없다고 판단

| | 회차 | 홀수_개수 | 번호합 | 소수 | 고수 |
|---|---|---|---|---|---|
| 0 | 1118 | 4 | 114 | 2 | 1 |
| 1 | 1117 | 3 | 115 | 2 | 3 |
| 2 | 1116 | 4 | 134 | 2 | 3 |
| 3 | 1115 | 2 | 144 | 2 | 4 |
| 4 | 1114 | 2 | 148 | 1 | 3 |
| ... | ... | ... | ... | ... | ... |
| 1113 | 5 | 2 | 192 | 3 | 5 |

```
1/1 [==============================] - 0s 49ms/step
Predicted number of odd numbers for the next draw: 3
```

## 기존 아이디어

## 3. 번호별 가중치 부여

- 각 번호별 당첨된 횟수를 가중치로 부여하고자 함
- 로또 당첨 번호 각각에 가중치를 부여하여 모델을 학습하기 어렵다고 판단함
- 도출 방안 -> 번호 별 당첨 확률과 회차별 1등 당첨 데이터를 통해 로또 번호를 예측

# 최종 복권 번호 예측 프로세스

로또 당첨 번호별
당첨 확률 계산

회차별 당첨 번호

→ 이번 회차 당첨 예측값들 생성 → 머신러닝, 딥러닝 모델을 통해 예측값들을 학습시켜 최적의 예측값 도출

**한계점 ?**

예측한 번호와 정답 데이터의
정확도를 확인하는 것이 애매함

# 모델 학습

RNN

```
class LOTTO_RNN(nn.Module):
  def __init__(self):
    super(LOTTO_RNN,self).__init__()
    # 시계열 특징 추출
    self.rnn = nn.RNN(input_size = input_size,hidden_size=hidden_size,
                      num_layers=num_layers, batch_first=True)
     # (batch_size, 5,6) --> (batch_size, 5,64)

    # MLP 층 정의(분류기)  5*64 = 320
    self.fc1 = nn.Linear(in_features=5*64,out_features=64)
    self.fc2 = nn.Linear(in_features=64,out_features=6)
    self.relu = nn.ReLU()

  def forward(self,x, h):
    x, hn = self.rnn(x,h)  # x 마지막 RNN 층의 은닉 상태, hn 모든 RNN 층의 은닉 상태
    x = torch.reshape(x, (x.shape[0],-1))  # mlp층에 사용하기 위해서 모양 변경

    x = self.fc1(x)
    x = self.relu(x)
    x = self.fc2(x)
    return x

  def predict(self):
    final_preds = torch.round(pred).type(torch.int)
    predict_RNN = final_preds.cpu().numpy()
    return predict_RNN
```

# 03 로또 번호 예측 앱

## 모델 학습

CNN

```python
class LOTTO_CNN(nn.Module):
    def __init__(self, input_channels=1, num_rows=5, num_cols=6, output_size=6):
        super(LOTTO_CNN, self).__init__()
        # CNN 레이어 구성
        self.conv1 = nn.Conv2d(input_channels, 32, kernel_size=(2, 2), stride=(1, 1))
        self.conv2 = nn.Conv2d(32, 64, kernel_size=(2, 2), stride=(1, 1))
        # Conv2d 출력 크기 계산
        conv1_size = (num_rows - 2 + 1, num_cols - 2 + 1)  # kernel_size = 2, stride = 1, no padding
        conv2_size = (conv1_size[0] - 2 + 1, conv1_size[1] - 2 + 1)  # Second layer same as first
        conv_output_size = 64 * conv2_size[0] * conv2_size[1]  # Num channels * height * width
        # 완전 연결층
        self.fc1 = nn.Linear(conv_output_size, 128)
        self.fc2 = nn.Linear(128, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = x.unsqueeze(1)  # (batch_size, 1, 5, 6)
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = torch.flatten(x, start_dim=1)
        x = self.relu(self.fc1(x))
        return self.fc2(x)

    def predict(self):
      predicted_lotto_numbers = generate_lotto_numbers(cnn_model, initial_sequence, num_predictions=1)
      for idx, numbers in enumerate(predicted_lotto_numbers, 1):
          print(f"CNN : {numbers[0]}")
      return numbers[0]
```

# 모델 학습

MLP

```python
# 다층 퍼셉트론 모델 정의
class LOTTO_MLP(nn.Module):
    def __init__(self, input_size=6, hidden_size=64, output_size=6):
        super(LOTTO_MLP, self).__init__()
        # MLP 레이어 구성
        self.layer1 = nn.Linear(input_size * 5, hidden_size)
        self.layer2 = nn.Linear(hidden_size, output_size)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = torch.flatten(x, start_dim=1)
        x = self.relu(self.layer1(x))
        return self.layer2(x)

    def predict(self):
        predicted_lotto_numbers = generate_lotto_numbers(mlp_model, initial_sequence, num_predictions=1)
        for idx, numbers in enumerate(predicted_lotto_numbers, 1):
            print(f"MLP : {numbers[0]}")
        return numbers[0]
```

# 03 로또 번호 예측 앱

## 모델 학습

RandomForest

```
lass LOTTO_RANDOMFOREST:
    def __init__(self, max_depth=10, max_features='sqrt', min_samples_leaf=4, min_samples_split=2, n_estimators=300):
        self.rf_model = RandomForestRegressor(max_depth=max_depth, max_features=max_features,
                                              min_samples_leaf=min_samples_leaf, min_samples_split=min_samples_split,
                                              n_estimators=n_estimators)

    def fit(self, X, y, test_size=0.2, random_state=42):
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_split(X, y, test_size=test_size, random_state=random_state)

        # Reshape data for training
        X_train_2d = self.X_train.reshape(self.X_train.shape[0], -1)
        X_test_2d = self.X_test.reshape(self.X_test.shape[0], -1)

        # Train the model
        self.rf_model.fit(X_train_2d, self.y_train)

        # Evaluate the model
        self.accuracy = self.rf_model.score(X_test_2d, self.y_test)

    def predict(self):
        # Reshape test data for prediction
        X_test_2d = self.X_test.reshape(self.X_test.shape[0], -1)

        # Make predictions
        predictions = self.rf_model.predict(X_test_2d)
        predicted_numbers = np.round(predictions).astype(int)

        # Cap numbers to max lotto value (e.g., 45)
        predicted_numbers = np.where(predicted_numbers > 45, 45, predicted_numbers)

        return predicted_numbers[0]
```

# 03 로또 번호 예측 앱

## 모델 학습

TransFormer

```python
class LottoTransformer(nn.Module):
    def __init__(self, input_size=6, d_model=64, num_heads=8, num_layers=4, output_size=6):
        super(LottoTransformer, self).__init__()
        self.embedding = nn.Linear(input_size, d_model)
        encoder_layer = nn.TransformerEncoderLayer(d_model=d_model, nhead=num_heads, batch_first=True)
        self.transformer = nn.TransformerEncoder(encoder_layer, num_layers=num_layers)
        self.fc = nn.Linear(d_model, output_size)

    def forward(self, x):
        x = self.embedding(x)
        x = self.transformer(x)
        x = x[:, -1, :]  # Use only the last timestep's output for prediction
        return self.fc(x)

    def predict(self):
      # Prediction example
      initial_sequence = torch.tensor([[3, 15, 21, 30, 33, 42], [4, 9, 14, 18, 27, 31], [6, 8, 10, 19, 23, 25]
      predicted_numbers = trans_model(initial_sequence)
      predicted_numbers = torch.round(predicted_numbers).type(torch.int)
      print("Predicted Lotto Numbers:", predicted_numbers.cpu().numpy())
      return predicted_numbers.cpu().numpy()
```

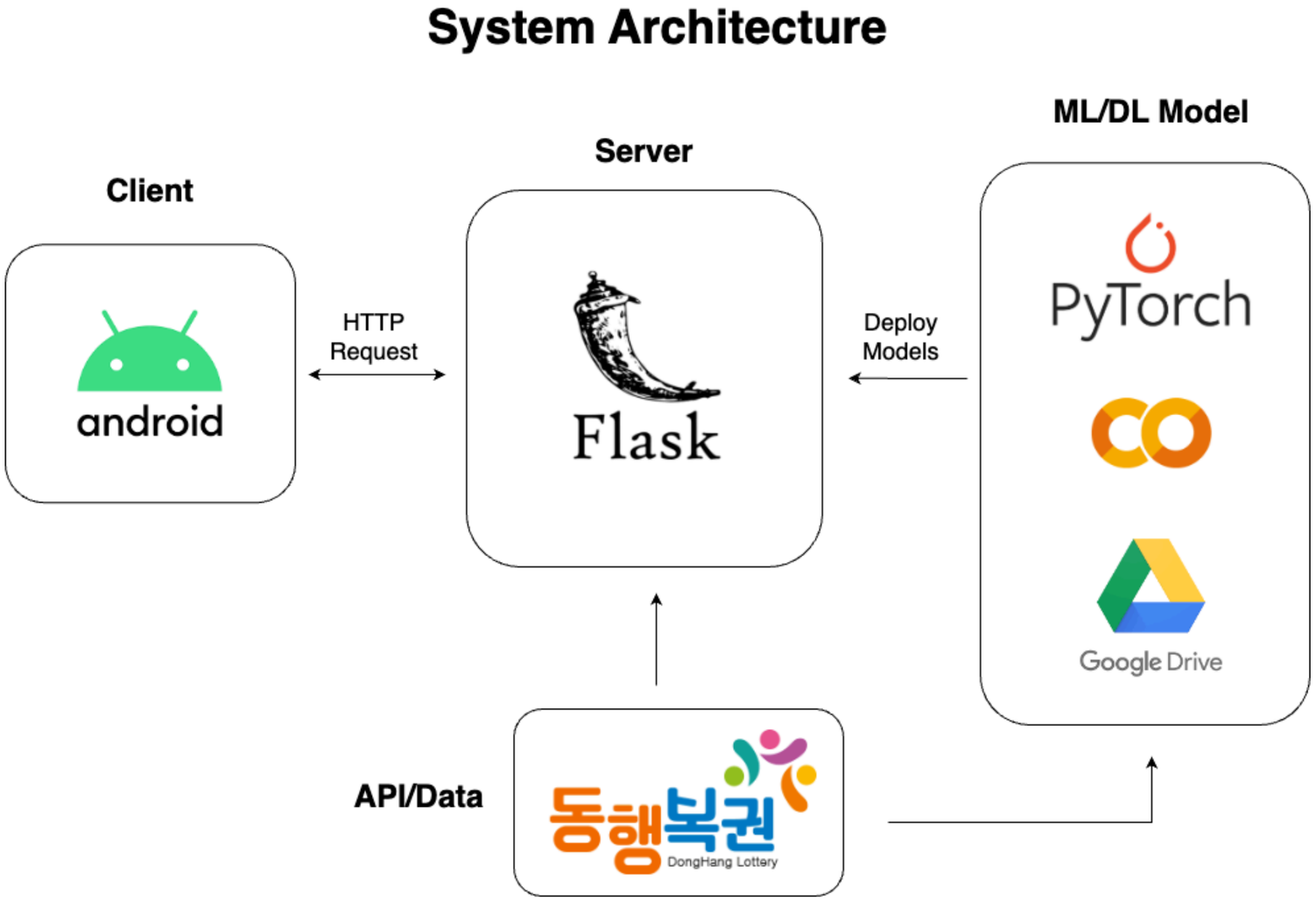
# 모델 학습

학습코드

```
# 학습
device = "cuda" if torch.cuda.is_available() else 'cpu'
rnn_model = LOTTO_RNN().to(device)
optim = Adam(rnn_model.parameters(),lr=1e-4)

iterator = tqdm.tqdm(range(50))
for epoch in iterator:
  for data, label in loader:
    optim.zero_grad()
    # 초기 은닉상태  (은닉층개수, 배치크기, 출력차원) -->0
    h0 = torch.zeros(5,data.shape[0],64).to(device)
    # 모델 forward - 예측
    pred = rnn_model(data.type(torch.FloatTensor).to(device), h0)
    # 손실
    loss = nn.MSELoss()(pred,label.type(torch.FloatTensor).to(device) )
    # 역전파
    loss.backward()
    # 가중치 업데이트(최적화)
    optim.step()
    iterator.set_description(f"epoch{epoch+1} loss:{loss.item()}")
```

# 시스템 아키텍처


System Architecture
ML/DL Model
Server
Client
HTTP
Request
Deploy
Models
android
Flask
PyTorch
Google Drive
API/Data
동행복권
DongHang Lottery

# Flask 서버 모델 배포

- 학습된 모델의 가중치 정보를 pickle 라이브러리를 사용해 저장해와 Flask 서버에 배포
- Flask 서버에서 HTTP API 통신을 통해 안드로이드에서 POST 요청이 들어오면 모델의 예측값들을 json 형태로 전달

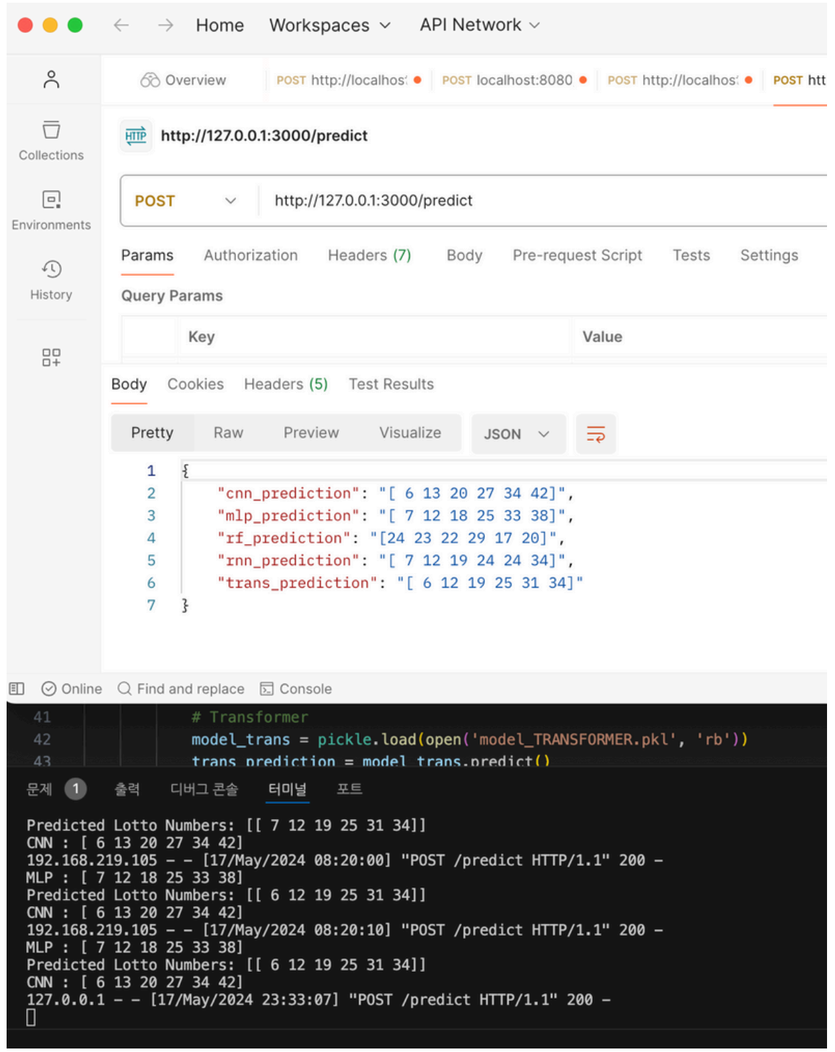
```python
@app.route('/predict', methods=['POST'])
def predict():
    if request.method == 'POST':
        try:
            # RNN
            # 모델 파일 로드
            model_rnn = pickle.load(open('model_RNN.pkl', 'rb'))
            # Make prediction
            rnn_prediction = model_rnn.predict()
```

```
├── app/
│   ├── __init__.py
│   ├── routes.py
│   ├── scheduler.py
│   ├── models/
│   │   ├── 로또번호.csv
│   │   ├── cnn.py
│   │   ├── mlp.py
│   │   ├── rf.py
│   │   ├── rnn.py
│   │   ├── transformer.py
│   │   └── pkl/
│   │       ├── model_CNN.pkl
│   │       ├── model_MLP.pkl
│   │       ├── model_RANDOMFOREST.pkl
│   │       ├── model_RNN.pkl
│   │       └── model_TRANSFORMER.pkl
├── config.py
└── run.py
```

▲ Flask 서버 폴더 구조

# Android와 Flask 서버의 통신 과정

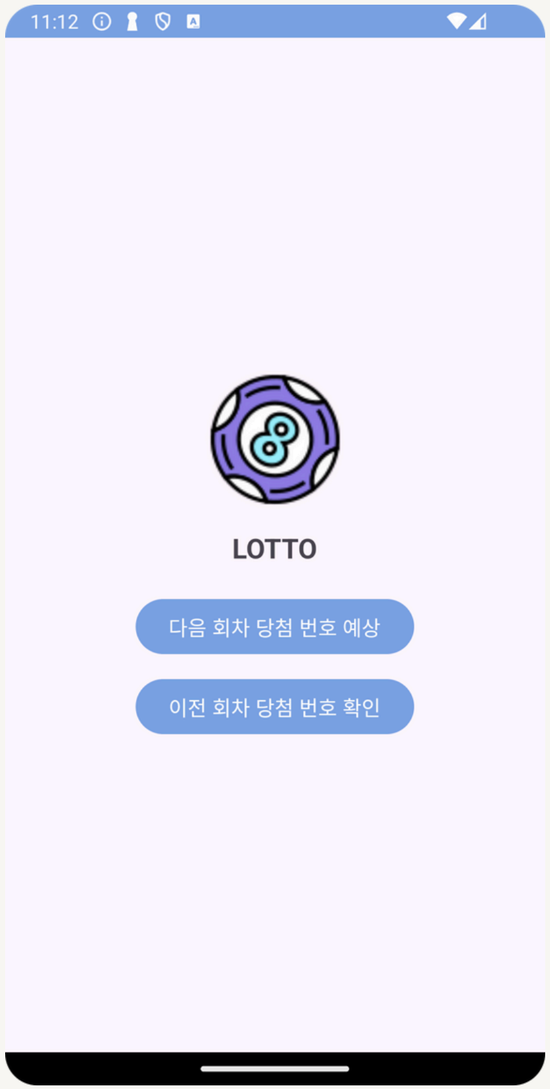
- 안드로이드에서는 Retrofit을 사용하여 서버와 통신
- Flask 서버에서 동행복권 api 연결
- APScheduler 이용하여 매 회차가 끝났을 때마다 당첨 번호들 받아오도록 설계
  => 새로운 회차마다 예측한 정보를 전달하도록 구현 예정

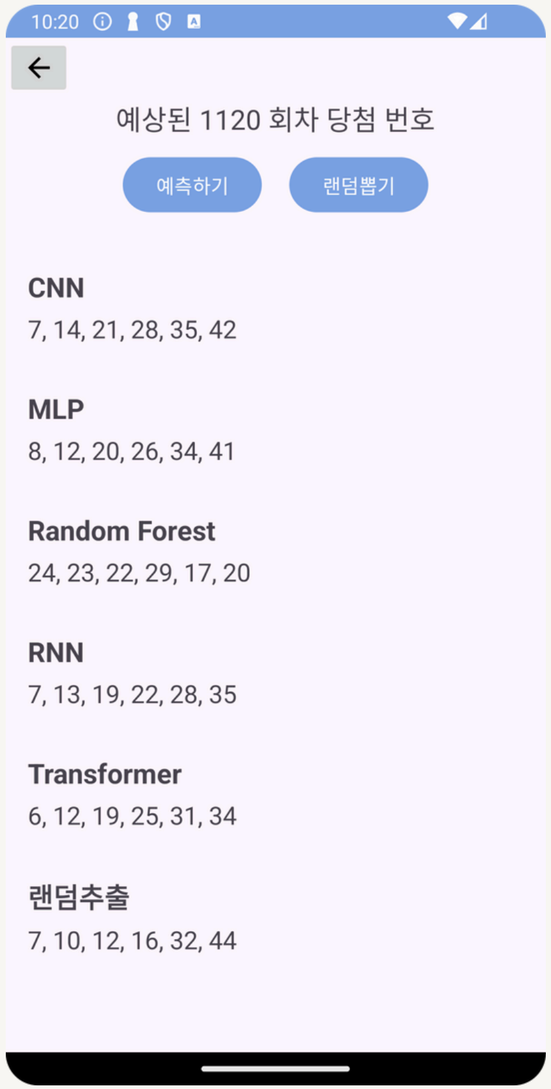


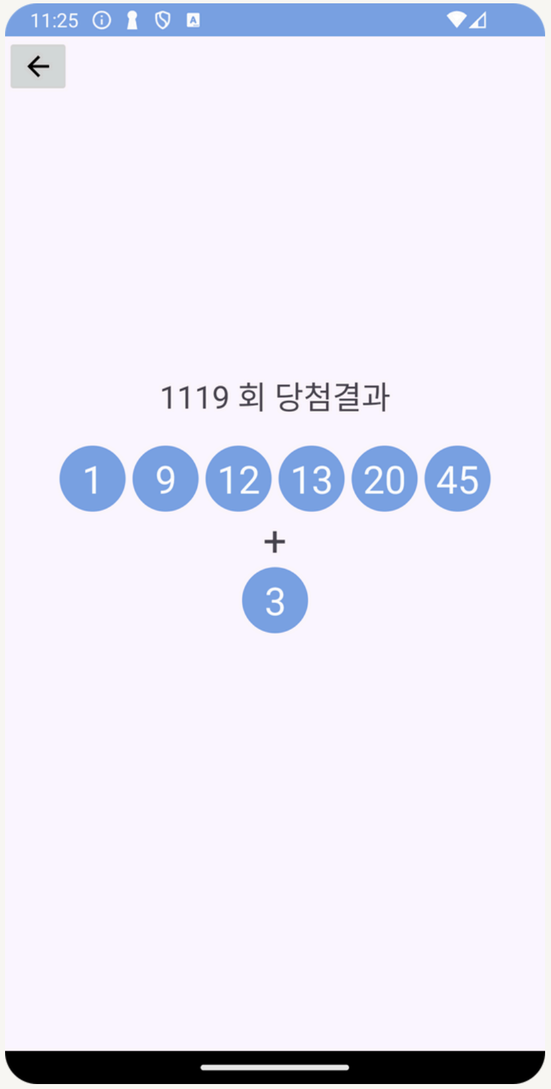

▲ Postman을 통한 Flask api 테스트

# 03 로또 번호 예측 앱

## 앱 화면



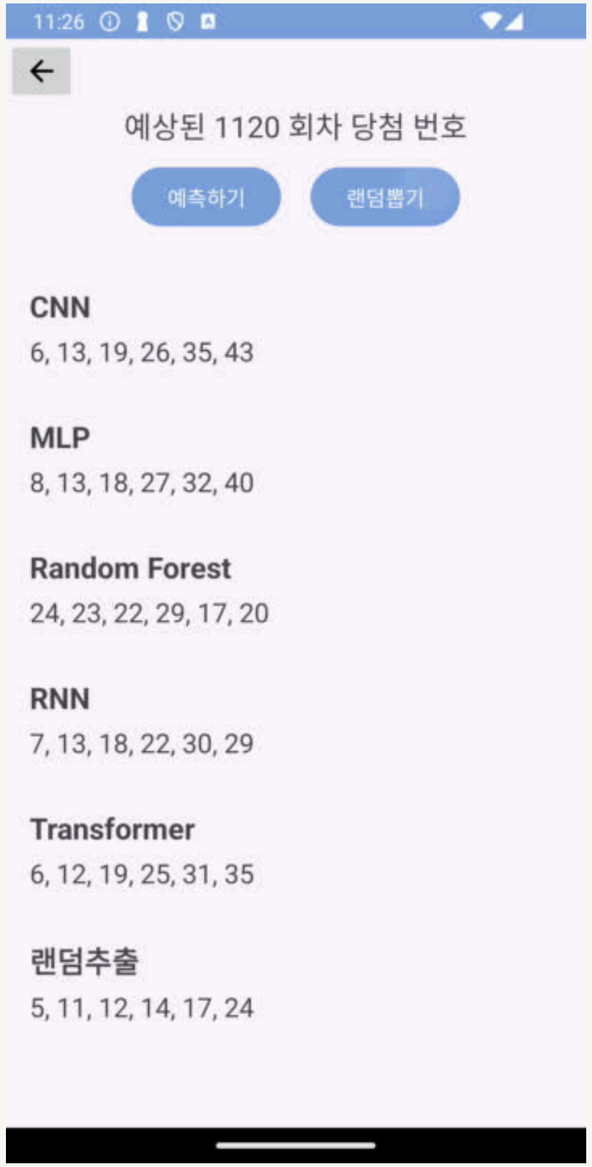



# 03 로또 번호 예측 앱

## 앱 시연영상

영상 링크



## 로또 번호 예측 앱 개선 방향 

- 새로운 회차가 시작될 때마다 새롭게 예측할 수 있도록 파이프라인 설계
- 기존의 UI/UX : 개발자 맞춤 -> 사용자 맞춤으로 수정
- 기능 추가
- aws에 서버 배포해서 직접 사용해보기 ?

# 04 웹 광고 클릭률 예측 AI 경진대회

## 대회 정보

DACON

**웹 광고 클릭률 예측 AI 경진대회**

알고리즘 | 정형 | 시계열 | 분류 | 웹 로그 | AUC

₩ 상금 : 인증서 + 데이스쿨

2024.05.07 ~ 2024.06.03 09:59 + Google Calendar

489명 D-17

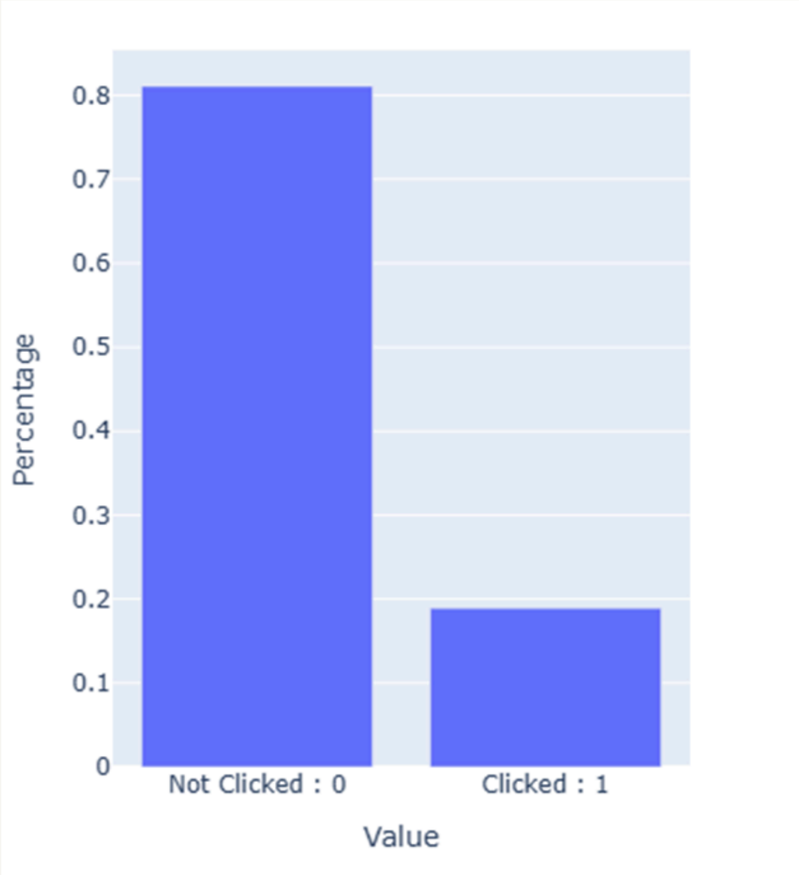
## 데이터 설명

- 시간 순으로 나열된 7일 동안의 웹 광고 클릭 로그
- ID: train 데이터 샘플 고유 ID
- Click: 예측 목표인 클릭 여부
- 0: 클릭하지 않음, 1: 클릭
- F01 ~ F39 : 각 클릭 로그와 연관된 Feature

| | ID | Click | F01 | F02 | F03 | F04 | F05 | F06 | F07 | F08 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | TRAIN_17401228 | 0 | MKVQKAU | IUXRXFC | None | 1.0 | UPCCJTA | 30 | PQZBVMG | SODVHOK |
| 1 | TRAIN_06201228 | 0 | JCDXFYU | PILDDJU | IAGJDOH | 34.0 | LFPUEOV | 10 | QVBETNR | VAWXMCR |
| 2 | TRAIN_11997039 | 0 | SZDXGJR | TUNNKIH | UKXCUGQ | 93.0 | RLXJLNR | 1 | IYNIXVX | FTPHMPQ |
| 3 | TRAIN_06139739 | 0 | YVKEYUW | SMQEJGC | None | NaN | HRDJASO | 1 | CGRJNMF | OFKQGTY |
| 4 | TRAIN_13195189 | 0 | CPDLJFM | PEDSSEZ | None | NaN | OSDVYBJ | -1 | PQZBVMG | FTPHMPQ |

# 데이터 확인

## Target 데이터 클래스 분포

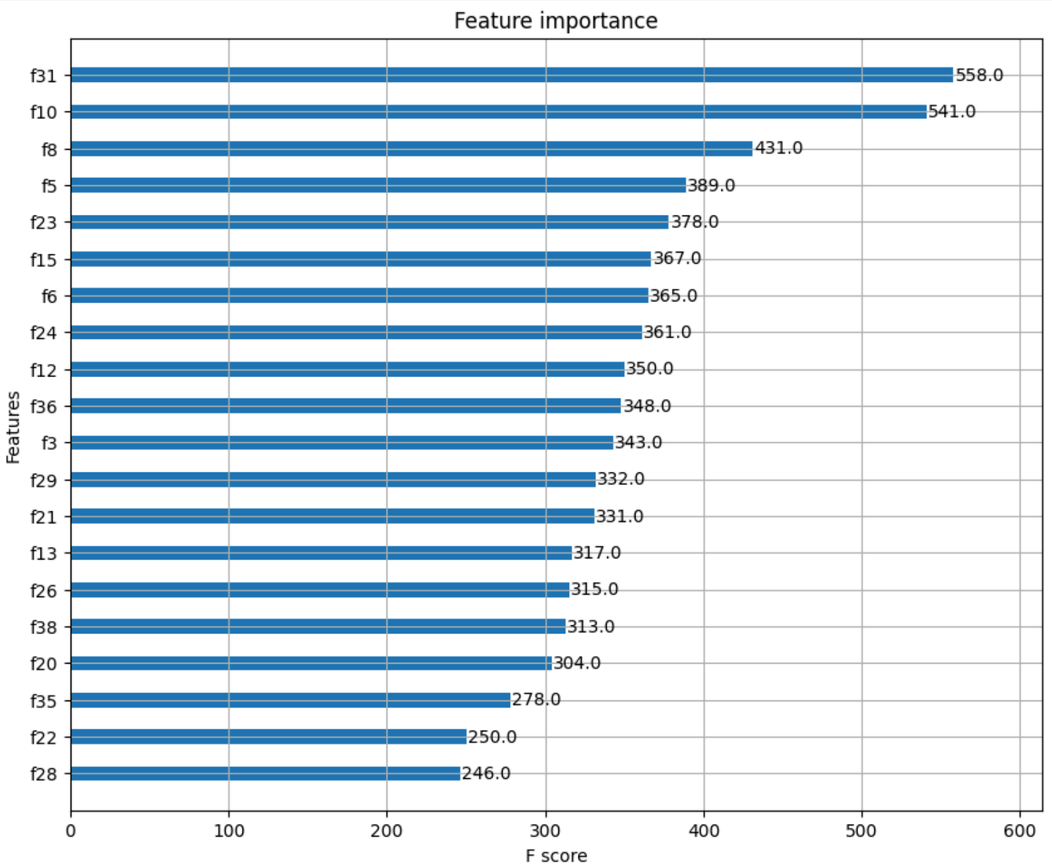



## 결측치 확인 및 고유값 확인 - feature 요약본

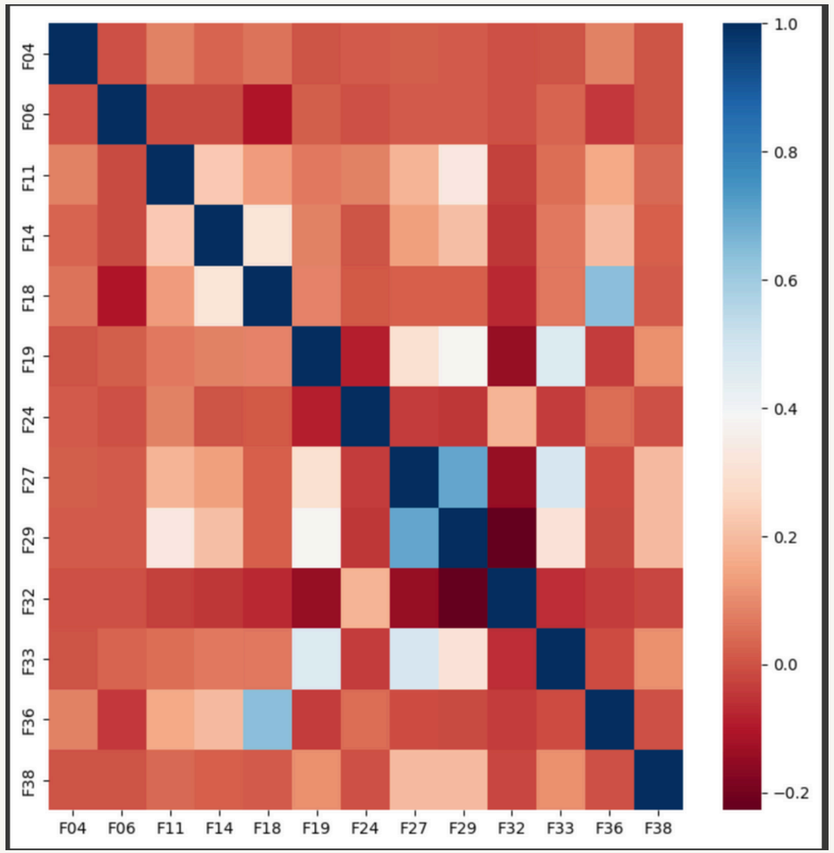
| | 피처 | 데이터 타입 | 결측값 개수 | 고윳값 개수 |
|---|---|---|---|---|
| 0 | ID | object | 0 | 28605391 |
| 1 | Click | int64 | 0 | 2 |
| 2 | F01 | object | 1234711 | 4760930 |
| 3 | F02 | object | 1234711 | 304404 |
| 4 | F03 | object | 10543986 | 63 |
| 5 | F04 | float64 | 5742331 | 9978 |
| 6 | F05 | object | 1234711 | 5343556 |
| 7 | F06 | int64 | 0 | 8562 |
| 8 | F07 | object | 0 | 151200 |
| 9 | F08 | object | 0 | 79 |
| 10 | F09 | object | 0 | 27551 |
| 11 | F10 | object | 1234711 | 1404254 |
| 12 | F11 | float64 | 2955564 | 8000 |
| 13 | F12 | object | 1234711 | 4174063 |
| 14 | F13 | object | 0 | 1307 |
| 15 | F14 | int64 | 0 | 526 |
| 16 | F15 | object | 10543986 | 3 |
| 17 | F16 | object | 0 | 15467 |
| 18 | F17 | object | 0 | 10 |
| 19 | F18 | float64 | 7324999 | 564 |
| 20 | F19 | float64 | 2588853 | 14 |
| 21 | F20 | object | 10543986 | 178602 |
| 22 | F21 | object | 0 | 33 |
| 23 | F22 | object | 0 | 7187 |
| 24 | F23 | object | 0 | 950 |
| 25 | F24 | float64 | 8994270 | 15229 |
| 26 | F25 | object | 0 | 10700 |
| 27 | F26 | object | 10543986 | 2204 |
| 28 | F27 | float64 | 11063877 | 3820 |
| 29 | F28 | object | 0 | 55 |
| 30 | F29 | float64 | 11063877 | 196 |
| 31 | F30 | object | 0 | 19444 |
| 32 | F31 | object | 0 | 14 |
| 33 | F32 | float64 | 251142 | 424278 |
| 34 | F33 | float64 | 2588853 | 600 |
| 35 | F34 | object | 1234711 | 3165580 |
| 36 | F35 | object | 0 | 3 |
| 37 | F36 | float64 | 7324999 | 1152 |
| 38 | F37 | object | 0 | 9423 |
| 39 | F38 | float64 | 800058 | 342 |
| 40 | F39 | object | 0 | 6800 |

# 데이터 확인

## Feature Importance




## 수치형 데이터 상관관계 Heatmap



## 데이터 전처리 아이디어

범주형

수치형

Fillna(0)

KNN 결측치

Robust Scaler

IQR

SMOTE 기법 적용

'ffill' 'bfill'

## 데이터 전처리 결측치 처리 - 데이터 스케일링

범주형 → fill na = 0 → CountEncoding

수치형 → fill na = 0

model_training

train_test_split

standardscaler

## 모델 선택

| BaseLine Model | ROC AUC | BaseLine Model | ROC AUC |
|---|---|---|---|
| LogisticRegression | 0.6702 | SGDClassifier | 0.6628 |
| KNN | 0.5740 | XGBoost Classifier | **0.7282** |
| RandomForest Classifier | **0.7137** | LightGBM Classifier | **0.7291** |
| DecisionTree | 0.6721 | CatBoost Classifier | **0.7069** |
| AdaBoost Classifier | 0.7067 | wide & deep | 0.5001 |

# 모델링 전략 - 아이디어

## 목표 : 선정한 베이스 모델을 이용해 최종 모델의 성능 높이기

**Stacking Ensemble**

ROC AUC 가장 높은 모델 3개 선택 + LogisticRegression (선형)
ROC AUC 가장 높은 모델 3개 선택 + RandomForestClassifier (트리형)
다양한 모델 특성을 반영하기
- RandomForestClassifier + SGDClassifier + CatBoostClassifier + KNeighborsClassifier

**Voting**

소프트 보팅 - ROC AUC 가장 높은 모델 3개 선택
ROC AUC 가장 높은 모델 4개 선택
하드 보팅 - ROC AUC 가장 높은 모델 3개 선택
ROC AUC 가장 높은 모델 4개 선택

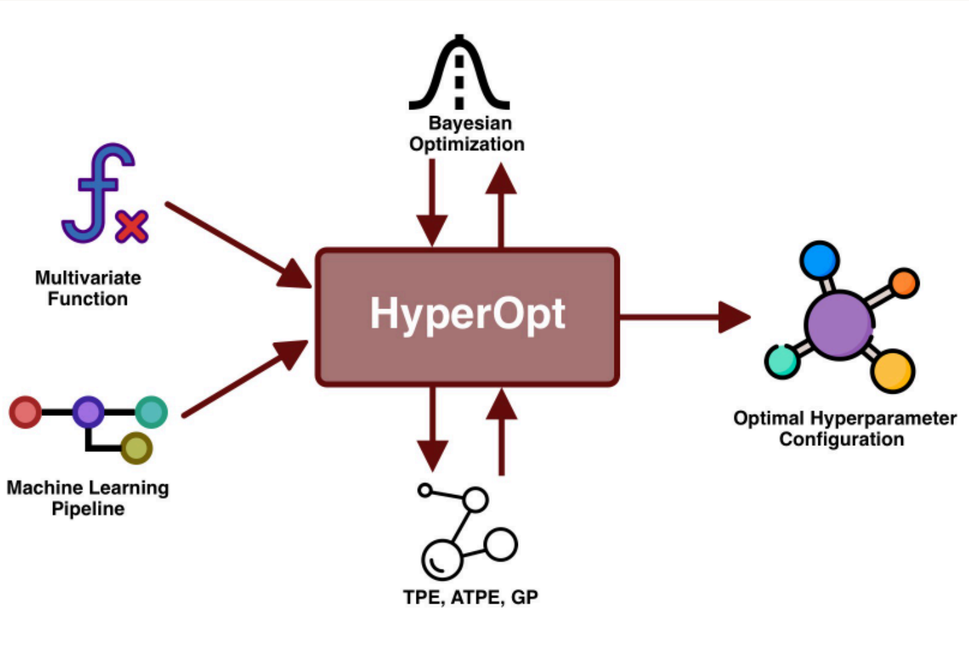
**Blending**

ROC AUC 가장 높은 모델 3개 선택 + LogisticRegression
ROC AUC 가장 높은 모델 3개 선택 + RandomForestClassifier

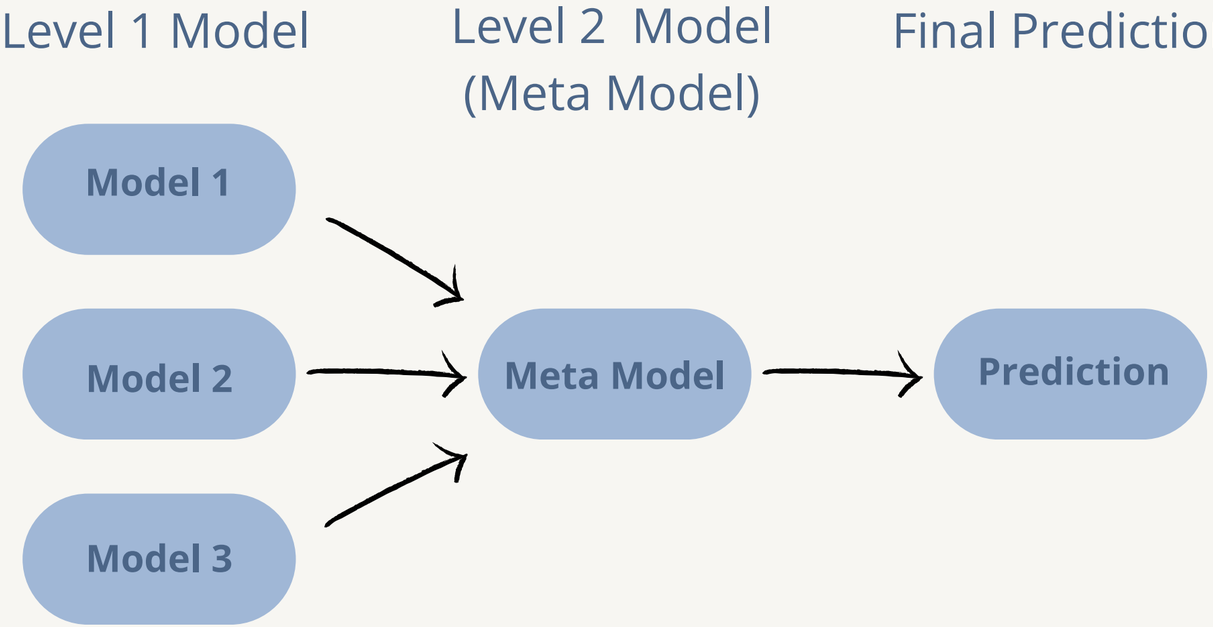
# 모델링 전략

## HyperOpt를 이용한 하이퍼파라미터 최적화 + Stacking Ensemble

### HyperOpt




### Stacking Ensemble



# 모델 학습 및 최적화 -HyperOPT 이용한 하이퍼파라미터 최적화

## 1. Search space 정의

```
xgb_search_space = {
    'max_depth':hp.quniform('max_depth', 5, 20, 1),
    'min_child_weight':hp.quniform('min_child_weight', 2, 15, 1),
    'learning_rate':hp.uniform('learning_rate', 0.01, 0.8),
    'colsample_bytree':hp.uniform('colsample_bytree', 0.5, 1),
    'colsample_bylevel': hp.uniform('colsample_bylevel', 0.5, 1.0),
    'colsample_bynode': hp.uniform('colsample_bynode', 0.5, 1.0),
    'reg_lambda': hp.loguniform('reg_lambda', np.log(0.01), np.log(1)),
    'reg_alpha': hp.loguniform('reg_alpha', np.log(0.01), np.log(1)),
    'subsample': hp.uniform('subsample', 0.6, 1.0),
    'gamma': hp.loguniform('gamma', np.log(0.1), np.log(1.0)),
}
```

## 2. Objective function 정의

```
def objective_func(search_space):
  xgb = XGBClassifier(n_estimators=100, max_depth=int(search_space['max_depth']),
                      min_child_weight=int(search_space['min_child_weight']),
                      learning_rate=search_space['learning_rate'],
                      colsample_bytree=search_space['colsample_bytree'],
                      colsample_bylevel=search_space['colsample_bylevel'],
                      colsample_bynode=search_space['colsample_bynode'],
                      reg_lambda=search_space['reg_lambda'],
                      reg_alpha=search_space['reg_alpha'],
                      subsample=search_space['subsample'],
                      gamma=search_space['gamma'],
                      eval_metric='auc')
  accuracy=cross_val_score(xgb, X_train_scaled, y_train, scoring='accuracy', cv=3)
  return {'loss':-1*np.mean(accuracy), 'status':STATUS_OK}
```

## 3. fmin으로 최적화값 리턴

```
trial_val = Trials()
best_xgb = fmin(fn=objective_func,
            space=xgb_search_space,
            algo=tpe.suggest,
            max_evals=50,
            trials=trial_val, rstate=np.random.default_rng(seed=9))
print("Best_xgb : ", best_xgb)
```

## 4. 최적 결과 출력 예시

```
# Best_xgb :  {'colsample_bylevel': 0.6670511042672311,
# 'colsample_bynode': 0.6374177225777424, 'colsample_bytree': 0.63597808749109
# 'gamma': 0.24886841697353138, 'learning_rate': 0.6853609308587266,
# 'max_depth': 19.0, 'min_child_weight': 2.0, 'reg_alpha': 0.07335157324491604
# 'reg_lambda': 0.0278188184352277, 'subsample': 0.8226333119871067}
```

# 모델 학습 및 최적화 - Level 1 모델 객체 생성 및 모델별 예측

## XGBoostClassifier

```
best_xgb_params = {
    'colsample_bytree': 0.6359780874910903,
    'learning_rate': 0.6853609308587266,
    'max_depth': 19,
    'min_child_weight': 2.0,
    'colsample_bynode': 0.6374177225777424,
    'colsample_bylevel': 0.6670511042672311,
    'gamma': 0.24886841697353138,
    'subsample': 0.8226333119871067,
    'reg_alpha': 0.07335157324491604,
    'reg_lambda': 0.027818818435227 7,

}

xgb = XGBClassifier(**best_xgb_params)
xgb.fit(X_train_scaled, y_train,
        early_stopping_rounds=100,
        eval_metric='auc',
        eval_set=[(X_test_scaled, y_test)])
xgb_pred = xgb.predict(X_test_scaled)
```

## CatBoostClassifier

```
best_cat_params = {
    'learning_rate': 0.040965744554766 62,
    'depth': 8,
    'iterations': 770,
    'l2_leaf_reg': 9.63140473917977,
    'border_count': 167,
    'random_strength': 0.007988045757175943
}

cat = CatBoostClassifier(**best_cat_params)
cat.fit(X_train_scaled, y_train,
        early_stopping_rounds=100,
        eval_set=[(X_test_scaled, y_test)])
cat_pred = cat.predict(X_test_scaled)
```

## LightGBMClassifier

```
best_lgbm_params = {
        'learning_rate': 0.19634012586729802,
        'max_depth': 121,
        'n_estimators': 931.0,
        'colsample_bytree': 0.6942343011406127,
        'num_leaves': 126,
        'min_child_samples': 94.0,
        'reg_alpha': 0.9845531684633071,
        'reg_lambda': 0.04747281468893378,
        'subsample': 0.7064988181106986}

lgbm = LGBMClassifier(**best_lgbm_params)
lgbm.fit(X_train_scaled, y_train,
         early_stopping_rounds=100,
         eval_metric='auc',
         eval_set=[(X_test_scaled, y_test)])
lgbm_pred = lgbm.predict(X_test_scaled)
```

# 모델 학습 및 최적화 - Stacking Ensemble

```python
from joblib import Memory
import tempfile
from sklearn.ensemble import StackingClassifier

# 임시 디렉토리 생성
cachedir = tempfile.mkdtemp()
memory = Memory(cachedir, verbose=1)

@memory.cache
def train_model(model, X, y):
    return model.fit(X, y)

estimators = [('cb', cat), ('lgbm', lgbm), ('xgb', xgb)]
stack = StackingClassifier(estimators=estimators,
                           final_estimator=LogisticRegression(),
                           verbose=1)

stack.fit(X_train_scaled, y_train)
pred = stack.predict(X_test_scaled)
pred_proba = stack.predict_proba(X_test_scaled)[:, 1]
roc_auc = roc_auc_score(y_test, pred_proba)
print(f'ROC AUC Score: {roc_auc}')
```

데이터 용량 ↑
모델을 디스크에 캐시하는 방식으로 학습 진행

## 최종 예측

### 단일 모델 하이퍼파라미터 튜닝 전 후 비교

| | | |
|---|---|---|
| XGBoostClassifier | 0.7282 | 0.7512 |
| LightGBMClassifier | 0.7291 | 0.7310 |
| CatBoostClassifier | 0.7069 | 0.7654 |

### 스태킹 앙상블 모델 하이퍼파라미터 튜닝 전 후 비교

0.7554 -> 0.7698

## 웹 광고 클릭률 예측 AI 경진대회 회고 및 개선 방향

- 대용량 데이터의 전처리
- 모델들의 결합으로 높이는 AUC
- 다양한 모델 활용, 모델 과적합 해결 스터디

# 감사합니다

PLAYDATA 데이터 엔지니어링 30기
김나영, 김설아, 백승민, 장지원, 최수빈

2024.05.20