# Auto ML HW 2

Paweł Gelar, Sabina Sidarovich

## 1 Wstęp

Celem pracy było zaproponowanie metody klasyfikacji, która pozwoli zbudować model o jak największej mocy predykcyjnej. Model należało przygotować w dwóch wariantach:

1. ręcznie, czyli wybierając rodzaj modelu, hiperparametry,
2. wykorzystując poznane frameworki AutoMLowe.

Oba modele miały być zoptymalizowane pod kątem skutecznego rozwiązania zadania klasyfikacji binarnej na sztucznie wygenerowanym zbiorze danych artificial. Zbiór treningowy zawierał 500 zmiennych objaśniających i obejmował 2000 obserwacji, z dołączoną zmienną celu. Dodatkowo otrzymaliśmy zbiór testowy, który nie zawierał zmiennej celu i składał się z 600 obserwacji. Naszym głównym celem było osiągnięcie maksymalnej dokładności predykcji dla obu modeli na zbiorze testowym, mierzonej za pomocą balanced accuracy.

## 2 Model klasyczny

Po krótkiej eksploracji danych przysąpiliśmy do transformacji danych. Ponieważ dane zawierały wyłącznie zmienne numeryczne, nie posiadały brakujących wartości oraz były zróżnicowane, jedynym etapem przygotowania danych było feature selection. Podzieliliśmy oryginalny zbiór *artificial_train* zbiór na treningowy i testowy w proporcji 3:1, żeby móc oszacować wyniki dokładność modeli. Do wyboru kolumn użyliśmy algorytmu Boruta. Wybraliśmy 21 kolumn, które otrzymały najlepszy wynik (miały ranking 1). Przetestowaliśmy również metody:

1. SelectKBest,
2. Mutual Information,
3. Recursive Feature Elimination,
4. Variance Threshold,
5. VIF (Variance Inflation Factor),
6. Outlier Detection,

lecz otrzymały one gorsze wyniki, niż Boruta, więc zdecydowaliśmy się je porzucić. Przetestowaliśmy nastęlujące rodzaje modeli:

1. XGBoost
2. LightGBM
3. Catboost

Zoptymalizowaliśmy hiperparametry modeli za pomocą biblioteki optuna z użyciem kroswalidacji. Ze względu na duże ograniczenia sprzętowe optymalizacja została uruchomiona tylko kilka razy dla każdego z modeli. Przestrzeń przeszukiwań hiperparametrów dla każdego z modeli została wybrana na podstawie dokumetacji oraz artykułow naukowych.

| **Hiperparametr** | **Przestrzeń** |
|---|---|
| Booster | [booster, gblinear, dart] |
| Learning rate | [0.01, 1.0] |
| Max depth | 1 - 10 |
| Subsample | [0.1, 1.0] |
| Colsample by tree | [0.5, 1.0] |
| Gamma | $[10^{-9}, 1.0]$ |
| Reg alpha | $[10^{-9}, 1.0]$ |
| Reg lambda | $[10^{-9}, 1]$ |
| Scale pos weight | [1, 10] |

Tabela 1: Przestrzeń przeszukiwań dla algorytmu XGBoost

| **Hiperparametr** | **Przestrzeń** |
|---|---|
| Learning rate | [0.01, 1.0] |
| N estimators | 10 - 1000 |
| Max depth | 1 - 15 |
| Min child samples | 1 - 20 |
| Min child weight | $[10^{-5}, 10^{2}]$ |
| Subsample | [0.01, 1.0] |
| Colsample by tree | [0.01, 1.0] |
| Reg alpha | $[10^{-9}, e]$ |
| Reg lambda | $[10^{-9}, e]$ |
| Num leaves | 2 - 200 |

Tabela 2: Przestrzeń przeszukiwań dla algorytmu LightGBM

| **Hiperparametr** | **Przestrzeń** |
|---|---|
| Learning rate | [0.001, 1.0] |
| Iterations | 100 - 2000 |
| Depth | 4 - 10 |
| L2 leaf reg | $[10^{-8}, 10^{2}]$ |
| Border count | 32 - 255 |
| Random strength | $[10^{-8}, 10.0]$ |
| Bagging temperature | [0.0, 10.0] |
| Od type | [IncToDec, Iter] |
| Od wait | 10 - 50 |

Tabela 3: Przestrzeń przeszukiwań dla algorytmu Catboost

Warto zaznaczyć, że nawet w przypadku biblioteki Optuna, która jest zoptymalizowanym narzędziem do doboru hiperparametrów, proces optymalizacji trwał od 10 minut do godziny przy przeprowadzeniu 500 iteracji. Natomiast w kontekście metody BayesSearchCV, czas ten wynosił kilka godzin. Poniżej przedstawione są wyniki na zbiorze testowym przed i po optymalizacji:

| Metoda | Balanced Accuracy (base) | Balanced Accuracy (tuned) |
|---|---|---|
| XGBoost | 86.0% | 88.6% |
| LightGBM | 85.2% | 88.8% |
| Catboost | 86.2% | 87.0% |

Tabela 4: Balanced accuracy dla algorytmów

# 3 Model automatycznego uczenia maszynowego

Przetestowaliśmy frameworki

1. Auto scikit-learn 2
2. Ml Jar
3. Tab PFN

Tab PFN nie zadziałał, gdyż maksymalna liczba kolumn dla niego wynosi 100. Zgodnie z duchem Auto ML postanowiliśmy nie ingerować w proces transformacji danych i uczenia, więc eksperymenty przeprowadziliśmy na pozostałych dwóch pakietach. W Ml Jar ustawiliśmy tryb *Compete*, a następnie uruchomiliśmy proces trenowania na dwie godziny. Modelem końcowym dla auto scikit-learn był model złożony z czterech zespołów extra trees i jednego lasu losowego. W przypadku Ml Jar był to Catboost.

# 4 Wyniki

Miarą do porównania modeli była dokładność (accuracy), gdyż klasy były idealnie zrównoważone. W przypadku tradycyjnych modeli liczona była ona na zbiorze testowym (hold-out), a auto-ml korzysta z kroswalidacji.

| Metoda | Dokładność |
|---|---|
| LightGBM (domyślne parametry) | 85% |
| LightGBM (parametry zoptymalizowane) | 89% |
| Auto scikit-learn | 85% |
| Ml Jar | 87% |

Tabela 5: Wyniki dla automatycznych i manualnych modeli

Jak można zauważyć, model wygenerowany przy użyciu frameworka uczenia automatycznego okazał się mniej efektywny, lecz jego tworzenie było łatwiejsze w porównaniu do tradycyjnego modelu. Warto jednak zaznaczyć, że charakterystyka zbioru danych, z którym pracowaliśmy, mogła wpłynąć na wyniki algorytmów, zwłaszcza ze względu na jego prostotę.