# Projekt 2 - AutoML

Franciszek Filipek, Julia Girtler, Jan Skalski

Styczeń 2026

## 1 Cel projektu

Celem projektu było stworzenie systemu **Mini-AutoML**, umożliwiającego **klasyfikacje binarną** dla danych w formie **tabelarycznej**. Istotą systemu miało być skonstruowanie i odpowiednie wykorzystanie **portfolio modeli**, na którym miał się opierać cały proces uczenia. System miał spełniać następujące wymagania:

- Posiadać **portfolio modeli** składające się z co najwyżej **50** konfiguracji
- Mieć możliwość wyboru najlepszego modelu w procesie **selekcji** na podstawie dostarczonych danych treningowych
- Umożliwiać stworzenie **ensemblingu** z wykorzystaniem do **5** modeli
- Być w pełni **powtarzalny** i przystosowany do przyjęcia dowolnego zbioru danych tabelarycznych przeznaczonych do klasyfikacji binarnej

## 2 Portfolio modeli

### 2.1 Zewnętrzny screening

Selekcja modeli składających się na portfolio została przeprowadzona w oparciu o **screening modeli** na podstawie zewnętrznych wyników zawartych w zbiorze **MementoML**. Przedstawia on zestawienie wyników uczenia w postaci miar **accuracy** i **AUC** dla 9 wybranych algorytmów:

- **agtboost** - Adaptive and Automatic Gradient Boosting
- **catboost** - Categorical Boosting
- **gbm** - Gradient Boosting
- **xgboost** - Extreme Gradient Boosting
- **glmnet** - Generalised Linear Model
- **kknn** - Weighted K-nearest neighbors
- **rfc** - Random Forest Classifier
- **ranger** - Fast Random Forest C++ Implementation for high-dimensional data
- **svm** - Support Vector Machines

Jak przedstawia opis **MementoML**, każdy algorytm był uczony na wybranych zbiorach po **20** razy dla każdej konfiguracji hiperparametrów dla danego zbioru (na różnych train-test splitach). Wyznaczone zostały miary **accuracy** i **AUC** na zbiorach testowych ze splitów. Łącznie użyto **39** różnych zbiorów danych i wykorzystano z góry ustaloną siatkę hiperparametrów dla każdego algorytmu. Zbiory pochodziły z **OpenML** i miały od **500** do **100000** rekordów, do **5000** zmiennych i proporcję klas powyżej **0.05**. Wszystkie wyliczenia zostały przeprowadzone w **R**, a dokładne siatki badanych hiperparametrów dostępne są tutaj: [Kretowicz and Biecek, 2020].

## 2.2 Utworzenie rankingu

Aby stworzyć portfolio, utworzono ranking konfiguracji modeli w następujący sposób:

- Wyznaczono **średnie accuracy** dla każdej konfiguracji hiperparametrów dla każdego algorytmu i dla każdego zbioru, na którym był uczony (na podstawie **20** wartości na splitach).
- Na podstawie tych średnich wyznaczono dalej uśrednione wyniki po zbiorach (dla każdego algorytmu i każdej konfiguracji odpowiadających mu hiperparametrów: średnie accuracy, medianę i odchylenie standardowe) oraz liczbę tych zbiorów (*n_ datasets*), na których uczony był dany algorytm przy danej konfiguracji.
- Utworzono nową kolumnę z miarą jakości konfiguracji **score**, która miała za zadanie balansować wyniki wyliczone w poprzednim punkcie. Konstrukcja miary oparta została na pewnej heurystyce wiążącej **średnie accuracy** z **medianą** oraz dodatkowej karze za niestabilność $\lambda$ (zbyt wysokie odchylenie) i bonusowi $\gamma$ za większą liczbę zbiorów użytych do uczenia dla danej konfiguracji danego modelu:

$$\textbf{score} \;=\; 0.6\,\textbf{mean\_acc} + 0.4\,\textbf{median\_acc} - \lambda\,\textbf{std\_acc} + \gamma\,log(\textbf{n\_datasets})$$

  Wartości parametrów $\lambda$ i $\gamma$ zostały wyznaczone deterministycznie tak aby wyższa stabilność miała większe znaczenie niż duża liczba zbiorów oraz aby kary i bonusy miały sensowną skalę względem różnic w **średniej accuracy** i w **medianie** pomiędzy konfiguracjami.
- W celu zestawienia rankingu, ostatecznie utworzono klasyfikację ze średnimi wynikami dla każdego modelu (*mean_ score*, *mean_ accuracy*, *median_ accuracy*, *best_ accuracy*, *mean_ std_ accuracy*) agregując po różnych konfiguracjach hiperparametrów, wraz z liczbą wszystkich testowanych konfiguracji(*n_ configs*) oraz średnią liczbą zbiorów użytych do uczenia dla każdego modelu (*mean_ n_ datasets*).
- Posortowano wyniki ze względu na wartość średniej miary *mean_ score*.

| | **score** | **Accuracy** | | | | **n_datasets** | |
|---|---|---|---|---|---|---|---|
| **model** | **mean** | **mean** | **median** | **best** | **mean_std** | **mean** | **n_configs** |
| ranger | 0.9324 | 0.8948 | 0.8947 | 0.9010 | 0.1025 | 35.0880 | 1000 |
| rfc | 0.9248 | 0.8934 | 0.8936 | 0.8977 | 0.1002 | 32.8940 | 1000 |
| xgboost | 0.8980 | 0.9828 | 0.9927 | 0.9952 | 0.1170 | 2.1487 | 195 |
| agtboost | 0.8908 | 0.9504 | 0.9454 | 0.9840 | 0.0922 | 1.7146 | 473 |
| gbm | 0.8885 | 0.8633 | 0.8706 | 0.8999 | 0.1141 | 33.7540 | 1000 |
| kknn | 0.8880 | 0.8574 | 0.8615 | 0.8671 | 0.1119 | 37.9310 | 1000 |
| svm | 0.8041 | 0.7972 | 0.8209 | 0.8526 | 0.1415 | 39.0000 | 11 |
| glmnet | 0.7453 | 0.7519 | 0.6907 | 0.8475 | 0.1429 | 38.0340 | 1000 |
| catboost | 0.6348 | 0.7358 | 0.7984 | 0.8480 | 0.1787 | 4.4360 | 1000 |

Tabela 1: Ranking modeli na podstawie *mean_ score*

## 2.3 Selekcja najlepszych modeli do portfolio

Wybór **35** konfiguracji do portfolio opierał się w pełni na mierze **score** i na powyższym rankingu. Zdecydowano, że ze względu na niski **score** (spowodowany słabą stabilnością i niewielką liczbą wykorzystanych zbiorów), portfolio nie będzie wcale zawierać konfiguracji dla algorytmu **catboost**. Z kolei z powodu braku dostępnego mapowania nazw hiperparametrów wykluczono także algorytmy **agtboost** i **svm**. Konfiguracje dla **RandomForest** zostały wybrane na podstawie innego zewnętrznego źródła, ponieważ konfiguracje dla tych modeli w zbiorze **MementoML** zawierały zbyt duże ilości drzew (np. 8000), co uniemożliwiło wykonywanie metody **fit** systemu w rozsądnym czasie. Algorytm **ranger** został odrzucony z powodu braku odpowiednika w pakiecie **skicit-learn** innego niż **RandomForest**. Z pozostałych **5 algorytmów** utworzono portfolio **35 konfiguracji**, biorąc po ustalonej liczbie konfiguracji z najlepszym wynikiem **score** dla każdego z tych algorytmów. Alokacja była proporcjonalna do **mean_score**:

- **xgboost** - **8** najlepszych konfiguracji
- **gbm** - **8** najlepszych konfiguracji
- **kknn** - **8** najlepszych konfiguracji
- **glmnet** - **6** najlepszych konfiguracji
- **rfc** - **5** konfiguracji inspirowanych artykułem [van Rijn and Hutter, 2018]

Model **glmnet** otrzymał więcej konfiguracji (mimo stosunkowo słabszego wyniku), aby zapewnić różnorodność w portfolio oraz lepsze działanie systemu dla danych o zależnościach w przybliżeniu liniowych. Nazwy hiperparametrów zostały odpowiednio zmapowane i przekonwertowane na składnie języka **Python**. Ostatecznie portfolio **35** konfiguracji (nazwa modelu, klasa modelu, zestaw hiperparametrów) zapisano do pliku **models.json**.

# 3 Implementacja systemu

## 3.1 Funkcje pomocnicze

Na początku, w celu zapewnienia poprawności działania systemu oraz replikowalności wyników, utworzono dwie funkcje wykorzystywane następnie w metodzie **fit**:

- **create_estimator** – Tworzy obiekt estymatora na podstawie klasy modelu oraz zestawu hiperparametrów; dopuszcza jedynie modele zgodne ze **scikit-learn** oraz implementacje **xgboost** i **catboost**.
- **add_random_state_if_supported** – Dodaje parametr **random_state** do estymatora, jeśli jest on wspierany przez dany model (tj. jeżeli jego działanie ma charakter stochastyczny); parametr nie jest nadpisywany, jeśli został już zdefiniowany w hiperparametrach.

## 3.2 Klasa MiniAutoML

Następnie utworzono klasę **MiniAutoML**, która implementuje uproszczony system **AutoML**, którego celem jest automatyczna selekcja najlepszych konfiguracji modeli oraz zbudowanie końcowego klasyfikatora. System przyjmuje zbiór kandydatów **models_config** w postaci listy konfiguracji (nazwa modelu, klasa modelu, zestaw hiperparametrów) z pliku **models.json**, a następnie ocenia je w sposób porównywalny dzięki ujednoliconemu przetwarzaniu danych oraz walidacji krzyżowej. Klasa **MiniAutoML** posiada następujące metody:

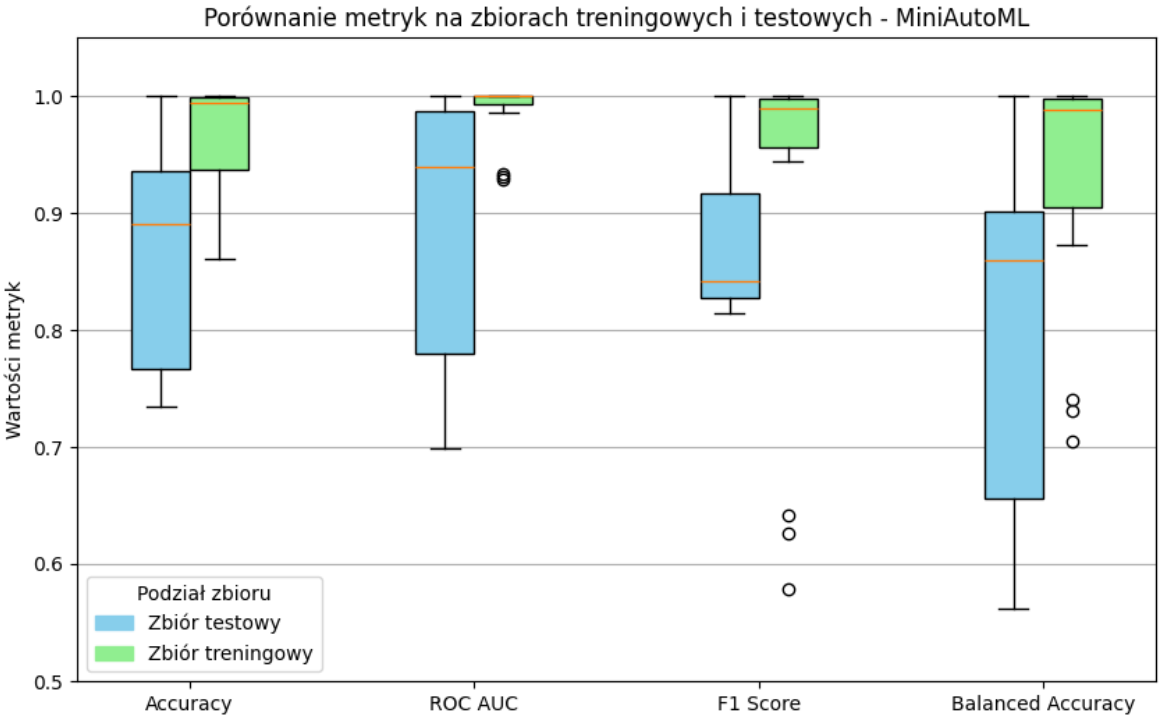
- **__init__(n_folds, models_config, random_state)** – Konstruktor odpowiada za przygotowanie listy kandydatów oraz **pipeline'u preprocessingu**. Konfiguracje modeli są najpierw grupowane według klasy estymatora (**models_by_class**), co umożliwia późniejszy wybór najlepszego przedstawiciela każdej klasy modeli. Następnie tworzony jest wspólny transformator danych (**ColumnTransformer**), składający się z dwóch gałęzi: dla cech **numerycznych** (imputacja medianą i standaryzacja) oraz dla cech **kategorycznych** (imputacja najczęstszą wartością i one-hot-encoding). W ten sposób wszystkie modele są oceniane na identycznie przetworzonych danych, co zwiększa **porównywalność** wyników.
- **fit(X_train, y_train)** – Realizuje pełną procedurę selekcji modeli. W pierwszym kroku, jeśli **n_folds** ustawiono na **"auto"**, liczba foldów dobierana jest heurystycznie na podstawie liczebności próby (im mniejsze zbiory tym mniej foldów). Następnie wszystkie konfiguracje modeli są oceniane przy użyciu walidacji krzyżowej **StratifiedKFold**, z metryką **ROC AUC**. Dla każdej konfiguracji tworzony jest pipeline z preprocessingiem, a dla estymatorów wspierających losowość dodawany jest parametr **random_state** w celu zapewnienia replikowalności. Wyniki ewaluacji zapisywane są w tabeli **leaderboard**, która stanowi ranking konfiguracji na podstawie średniego **AUC**. Kolejno wybierany jest najlepszy model z każdej klasy, po czym spośród nich wybierane jest maksymalnie **5** najlepszych konfiguracji tworzących finalny ranking **selected_models_leaderboard**. W celu utworzenia **final_model**, na końcu wykonywany jest **ensebling** z najlepszych konfiguracji za pomocą **VotingClassifier** z głosowaniem miękkim (**soft voting**), w którym każdy składnik jest pełnym pipeline'em zawierającym preprocessing oraz wybrany estymator.
- **predict(X_test)** – Zwraca predykcje klas, korzystając z wytrenowanego ensemble'a **final_model**.
- **predict_proba(X_test)** – Zwraca prawdopodobieństwa przynależności do klasy pozytywnej.

# 4 Eksperymenty

Przeprowadzono eksperymenty w celu sprawdzenia jakości nowego frameworku. W tym celu wzięto następujące, raczej mniejsze co do wielkości zbiory danych z OpenML:

| Zbiór danych | Wiersze | Kolumny | Proporcja klasy "1" |
|---|---|---|---|
| phoneme | 5404 | 5 | 0.29 |
| credit-g | 1000 | 20 | 0.7 |
| blood-transfusion | 748 | 4 | 0.24 |
| qsar-biodeg | 1055 | 41 | 0.66 |
| banknote-authentication | 1372 | 4 | 0.44 |
| socmob | 1156 | 5 | 0.22 |

Na każdym zbiorze danych, trzykrotnie dokonywano podziału na część treningową i testową (z różnymi ziarnami losowości) i dopasowywano framework na części treningowej (około 80% zbioru). Poniżej znajdują się boxploty różnych miar klasyfikacji wyliczane zarówno na części treningowej jak i testowej. Finalnie więc, każdy boxplot jest zbudowany z 18 obserwacji.




Framework radzi sobie bardzo dobrze na tych zbiorach, natomiast czasami ma tendencję do przeuczania się (być może wina dużych ensembli). Widać, że wariancja wyników na zbiorze testowym jest istotnie większa, co zgadza się z intuicją.

# Literatura

[Kretowicz and Biecek, 2020] Kretowicz, W. and Biecek, P. (2020). Mementoml: Performance of selected machine learning algorithm configurations on openml100 datasets.

[van Rijn and Hutter, 2018] van Rijn, J. N. and Hutter, F. (2018). Hyperparameter importance across datasets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2018)*, pages 2367–2376. ACM.