Raport Projekt 2

Bartosz Szymański, Aleksandra Uznańska, Igor Lechoszest

25 stycznia 2026

1 Cel projektu

Celem projektu było stworzenie uproszczonego systemu AutoML, który miał umożliwić automatyczne wykonanie klasyfikacji binarnej na dowolnym dostarczonym zbiorze danych do tego przeznaczonym. Należało stworzyć portfolio modeli, z których następnie, na podstawie zbioru testowego, system miał wybrać najlepszy z nich i go wytrenować.

2 Wstępna selekcja modeli

Do procesu wyboru modeli zostały wykorzystane gotowe wyniki dostępne na platformie OpenML, w ramach banchmarku 'OpenML-CC18'. Dostępne zbiory danych zostały przefiltrowane do tych, które dotyczyły klasyfikacji binarnej. Po zastosowaniu ograniczenia otrzymaliśmy ostatecznie 35 zbiorów danych. Z powodu różnych poziomów trudności oraz różnych charakterystyk każdy z nich został potraktowany oddzielnie. W ramach każdego zbioru wyszukaliśmy model z pakietu scikit-learn, który uzyskał najwyższy wynik wybranej przez nas metrki (AUC) oraz dodaliśmy go do naszego portfolio.

W ten sposób otrzymaliśmy 35 modeli. W finalnym pliku models.json znajduje się ich jednak 34, ponieważ jeden trzeba było usunąć ze względu na bardzo długi czas trening (SVM). Każdy z modeli był zatem najlepszy w konkretnym zbiorze danych, a ze względu na różnorodność zbiorów dostępnych w banchmarku, takie działanie zapewnia możliwość wyboru solidnego modelu (lub modeli) w zależności od nowych danych.

Typy modeli jakie zostały zastosowane (z różnymi konfiguracjami hiperparametrów) to:

1. RandomForestClassifier (sklearn.ensemble.RandomForestClassifier),
2. HistGradientBoostingClassifier (sklearn.ensemble.HistGradientBoostingClassifier),
3. SVC (sklearn.svm.SVC).
4. MLPClassifier (sklearn.neural_network.MLPClassifier).

Wśród naszych modeli najwięcej było różnych konfiguracji RandomForestClassifier, bo aż 23. HistGradientBoostingClassifier był reprezentowany przez 8 różnych kombinacji, SVC przez 2, a MLPClassifier tylko 1.

3 Selekcja modeli z portfolio

Kluczowym elementem systemu Mini-AutoML był proces wyboru optymalnego modelu spośród wszystkich konfiguracji zawartych w pliku JSON dla zadanego zbioru danych. Zastosowaliśmy standardowe podejście, gdzie każdy model był opisany przez słownik z kluczami "name", "class" i "params", gdzie wartości pierwszych dwóch kluczy były zmiennymi tekstowymi opisującymi model, a wartość klucza "params" była słownikiem z konfiguracją hiperparametrów danego modelu.

3.1 Preprocessing

Aby zapewnić porównywalność wyników każdy model jest testowany w ramach identycznego pipeline'u generowanego przez metodę *_create_preprocessing_pipeline*.

1. **Zmienne numeryczne:** zastosowaliśmy *SpimpleImputer(strategy = 'mean')* oraz *StandardScaler*, który jest kluczowy np dla modelu SVM oraz KNN.

2. **Zmienne kategoryczne:** braki danych uzupełniane są najczęściej wystepującą wartością, a następnie stosowany jest *OneHotEncoder*.

Jest to podejście minimalistyczne. Ze względu na to, że nie znamy struktury danych jakie będą używane wstrzymaliśmy się od bardziej rozbudowanej inżynierii danych. Dzięki temu modele dostają pełne dane, co wiąże się oczywiście z pewnymi minusami w przypadku występowania szumu lub innych czynników zniekształcających dane. Nie stanowi to jednak większego problemu, ponieważ ogromna większość naszych modeli to modele oparte na drzewach, a jak wykazano w pracach [1] i [2] tego typu modele dobrze sobie radzą z danymi niskiej jakości.

3.2 Strategia Selekcji

Metoda *fit* iteruje po kolei po wszystkich modelach z portfolio oraz tworzy tymczasowy pipeline dla każdego z nich. Niektóre z parametrów w modelach gradientowych oraz drzewiastych okazały się przestarzałe więc stworzyliśmy mechanizm, który je poprawia aby zapewnić kompatybilność z najnowszymi wersjami bibliotek. System wykorzystuje 5-krotną kroswalidację w celu ewaluacji każdego z modeli, a jako kryterium wyboru najlepszego z nich stosowana jest metryka *balanced accuracy*. W przypadku błędu w pojedynczym modelu, jest on odnotowany, a funkcja przechodzi do kolejnego. Gdy wybrany zostanie najlepszy model, to jest ostatecznie trenowany na całym zbiorze danych.

3.3 Ensembling

Jeżeli parametr klasy *voting* ustawiony jest na wartość *true* to system uruchamia budowę komitetu. Wszystkie modele sortowane są ze względu na wartości metryki *balanced accuracy*, a następnie wybierane jest 5 najlepszych z nich. Tworzony jest *VotingClassifier*, który łączy ich predykcje stosując głosowanie miękkie (*voting = 'soft'*). Za pomocą 5-krotnej walidacji krzyżowej porównywane jest średnie *balanced accuracy* z najlepszym otrzymanym dotychczas średnim wynikiem z modeli z portfolio.

4 Podsumowanie

Utworzona przez nas klasa *MiniAutoML* jest w pełni funkcjonalnym pipelinem automatycznego uczenia maszynowego, któremu wystarczy podać konfigurację modeli jaką ma przeszukiwać, wartość True badź False dla parametru *voting* zależnie od tego czy chcemy aby ostateczna klasyfiakcja odbywała się przy użyciu głosowania lub nie, oraz ilość procesów (parametr *n_jobs*). Dwa ostatnie parametry domyślnie ustawione na *voting*=False i *n_jobs*=1.

Klasa *MiniAutoML* implementuje metody *fit*, *predict* i *predict_proba* zgodnie z założeniami projektu.

Literatura

[1] Hubert Ruczyński and Anna Kozak. Do tree-based models need data preprocessing? 2024.

[2] Igor Lechoszest. Wpływ wstępnego przetwarzania danych na jakość modelu *extra-trees*. 2025.