MiniAutoML

333148, 333140, 333295

Ludwik Madej, Karol Kacprzak, Mikołaj Bójski

1 Wprowadzenie

Chcielibyśmy zaprezentować nasz pakiet MiniAutoML będący systemem automatycznego uczenia maszynowego do zadania klasyfikacji binarnej. Stworzyliśmy kompleksowe rozwiązanie, które, ze względu na prostotę użycia, jest adekwatne dla osób nieposiadających zaawansowanych umiejętności programistycznych, ale jednocześnie oferuje możliwości szerokiej konfiguracji pod indywidualne wymagania, co odpowiada na potrzeby doświadczonych użytkowników mierzących się z nietypowymi problemami. Zdecydowaliśmy się ograniczyć do czołowych modeli z rodziny GBM, tj. CatBoost, LightGBM i XGBoost - wspomaganych przez las losowy, regresje logistyczną i algorytm najbliższych sąsiadów - ze względu na "lekkość pakietu". Użycie sieci neuronowych czy foundation models wiązałoby się z koniecznością dodanie rozległych zależności, które znacznie utrudniłyby procedurę instalacji pakietów będącą i tak traumatycznym doświadczeniem dla wielu użytkowników.

Każdy etap przetwarzania ma jasno określony cel zapewniający wartość dodaną do całego pakietu. Preprocessing i selekcja cech zapewniają jakościowe dane, portfolio oparte na uznanych benchmarkach solidną konfigurację startową, trzyetapowy sposób wyboru modeli optymalny ich dobór dla danego zadania do komitetu, natomiast sam komitet - oferujący kilka różnych podejść, w tym multi-layer stacking - jak najdokładniejsze predykcje zwracane użytkownikowi.

W poniższych sekcjach każdy z tych etapów zostanie szczegółowo opisany, zarówno z perspektywy technicznej, jak i praktycznej dla potencjalnych użytkowników.

2 Wstępna selekcja modeli - budowa portfolio

Skorzystaliśmy z podejścia opierającego się na screeningu na podstawie wyników zewnętrznych dostępnych na stronach MementoML i TabArena. Konfiguracje modeli LogisticRegression i KNN (w obu przypadkach użyto implementacji z sklearn) pochodzą z MementoML. Dla każdego modelu z tego źródła osobno najpierw przeskalowano accuracy i ROC AUC do przedziału [0, 1] (w celu zapewnienia równego ich wpływu na wynik). Następnie wykonano zgrupowanie po konfiguracji hiperparametrów i jako metrykę oceny konfiguracji wybrano średnią z sumy (po wszystkich zbiorach danych) przeskalowanych accuracy i ROC AUC.

Natomiast konfigurację pozostałych użytych modeli - tj. CatBoost, LightGBM, XGBoost oraz Random Forest - wydobyto po skomplikowanym przetwarzaniu z TabAreny. Skorzystanie w tym przypadku z MementoML byłoby trudne z powodu różnic między pakietami w R i python, a dokładniej hiperparametrami modeli w różnych implementacjach.

W przypadku TabAreny proces rozpoczęto od przeglądu dostępnych wyników eksperymentów, zagregowanych dla zadań klasyfikacji binarnej. Zidentyfikowano, że w benchmarku tym jako miarę błędu dla poszczególnych modeli przyjęto metrykę $1 - ROCAUC$. W celu wyłonienia najlepszych kandydatów do portfolio, dla każdej z 200 losowych konfiguracji (zdefiniowanych przez autorów w ramach strategii Random Search) obliczono średnią wartość tej metryki po wszystkich uwzględnionych w eksperymentach zbiorach danych. Pozwoliło to wskazać indeksy konfiguracji, reprezentowane w danych przez suffixy, cechujące się najniższym uśrednionym błędem. Kluczowym etapem było następnie odnalezienie w repozytorium projektu definicji przestrzeni poszukiwań oraz użytego ziarna losowości. Dzięki tym informacjom możliwe było wierne odwzorowanie wartości hiperparametrów dla interesujących nas modeli, które w wynikach widniały jedynie jako identyfikatory.

Ostatecznie wybrano najlepsze - według przyjętych metryk - konfiguracje modeli i uzyskano portfolio składające się z 50 modeli: 13 Catboost, 13 LightGBM, 11 XGBoost, 8 LogisticRegression, 4 RandomForest i 1 KNN. Na proporcje wpłynęły zarówno wyniki modeli z benchmarków - porównywalne wyniki osiągane przez CatBoost i LightGBM, trochę słabsze przez XGBoost oraz dużo gorsze przez LogisticRegression, RandomForest i KNN - jak i chęć stworzenia na późniejszym etapie różnorodnego komitetu - stąd spora liczba LogisticRegression oraz uwzględnienie w portfolio RandomForest i KNN.

3 Preprocessing

Obecność mechanizmów do wstępnego przetwarzania danych w wiodących pakietach służących do automatycznego uczenia maszynowego nie jest sprawą tak oczywistą jak mogłoby się wydawać. Nawet w tak uznanych i popularnych pakietach jak AutoGluon zaimplementowano, z różnych, niewyjaśnionych w dokumentacji powodów tylko bardzo podstawowy preprocessing służący uniknięciu przekazaniu modelom nieobsługiwanych przez nie typów kolumn.

W tym projekcie postawiliśmy sobie za cel stworzenie kompleksowego rozwiązania, które nie będzie miało takich ograniczeń. Wszak niewiele jest tak irytujących rzeczy dla użytkownika jak konieczność ręcznego usuwania brakujących wartości i ustawiania typów kolumn, czego doświadczyliśmy przy AutoGluon. Nasze narzędzie do wstępnego przetwarzania danych składa się z 4 głównych komponentów: wykrywania typów zmiennych, kodowania, skalowania oraz zarządzania brakami danych. Poprawne wykrycie typów oraz przetworzenie surowych danych nie tylko ułatwia zadanie modelom, ale też upraszcza wewnętrzne procesy implementacyjne. Dzięki udostępnieniu wielu hiperparametrów oraz wykorzystaniu API sklearn użytkownik może decydować o prawie każdym kroku i przekazywać własne instancje m.in. obiektów Encoder, Scaler czy Imputer, co pozwala na rozbudowaną konfigurację dostosowaną do bardzo specyficznych zadań. Na szczególną uwagę zasługuje komponent służący do wykrywania typów zmiennych. Inspirując się podejściem twórców pakietu lightautoml nasze rozwiązanie nie tylko bazuje na różnych, możliwych do spersonalizowania regułach wyboru - na przykład minimalnej zmienności wartości w kolumnie, aby została ona uznana za istotną - ale też oferuje zaawansowane automatyczne typowanie. Przy zastosowaniu zbliżonych reguł decyzyjnych do tych przedstawionych w artykule omawiającym pakiet lightautoml nasze narzędzie może zaoferować realne korzyści takie jak te opisane, na podstawie przeprowadzonych przez autorów eksperymentów, w powyższej publikacji. Ponadto, nasz mechanizm do wstępnego przetwarzania danych zwraca ramki danych na różnych, sterowanych przez hiperparametry, poziomach przetworzenia - przykładowo z wykonanym lub nie skalowaniem zmiennych - co pozwoliło nam przekazywać do poszczególnych modeli dane w formacie najbardziej im pasującym. W połączeniu z opisanymi niżej metodami selekcji cech nasz rozwiązanie zapewnia - przy braku wykorzystywania ręcznego etykietowania i oceny istotności danych - możliwie czyste oraz niosące informacje zbiory, które będą łatwiejsze do przewidywania dla modeli, a zatem potencjalnie zwiększające istotnie dokładność predykcji, która jest często decydującym kryterium dla użytkowników końcowych.

4 Metody selekcji cech

Proces selekcji cech został zrealizowany w postaci modularnego transformera `FeatureSelector`, w pełni zgodnego z interfejsem `scikit-learn`. Projekt zakłada, że każda ramka danych — potencjalnie poddana innemu etapowi wstępnego przetwarzania — może być selekcjonowana *niezależnie*, z użyciem metody najlepiej dopasowanej do jej charakterystyki. Takie podejście naturalnie wspiera architektury wielowidokowe oraz zespołowe, w których różne modele operują na odmiennych reprezentacjach cech.

Kluczowym elementem implementacji jest parametryzacja liczby wybieranych cech poprzez frakcję początkowego zbioru. Użytkownik określa parametr

$$k = \lceil \alpha \cdot d \rceil ,$$

gdzie $\alpha \in (0, 1]$ oznacza zadaną frakcję, a d liczbę cech wejściowych. Mechanizm ten zapewnia spójną kontrolę złożoności modelu niezależnie od wymiarowości danych oraz ułatwia porównywanie różnych strategii selekcji.

Zaimplementowane metody selekcji obejmują zarówno podejścia filtrujące, jak i metody nadzorowane oraz oparte na modelach uczących się. Tabela 1 przedstawia ich syntetyczne zestawienie.

Metoda	**Opis działania**
`variance`	Usuwanie cech stałych i quasi-stałych na podstawie progu wariancji.
`mutual_info`	Ranking cech na podstawie estymacji wzajemnej informacji z wykorzystaniem sąsiedztwa punktów (KNN).
`chi2`	Test niezależności badający zależność cech od zmiennej objaśnianej (dla danych nieujemnych).
`random_forest`	Selekcja cech na podstawie miar istotności generowanych przez las losowy (MDI).
`xgboost`	Ranking cech w oparciu o statystyki ważności (np. *gain*, *weight*) modelu gradientowego.
`boruta`	Iteracyjna metoda typu *all-relevant*, porównująca cechy rzeczywiste z losowymi cechami cieniami.
`permutation`	Pomiar spadku jakości predykcji po losowej permutacji pojedynczej cechy.

Table 1: Zestawienie zaimplementowanych mechanizmów selekcji cech.

Zastosowana architektura umożliwia spójne włączenie selekcji cech w złożone potoki uczenia maszynowego, w tym w systemy ensemble i wielowarstwowy stacking, przy jednoczesnym zachowaniu pełnej kontroli nad kompromisem pomiędzy złożonością a jakością predykcji.

5 Selekcja modeli

Proces doboru ostatecznego zestawu predyktorów z szerokiego portfolio został zrealizowany w formie hierarchicznej procedury selekcji. System `ModelSelectorV3` implementuje zaawansowany mechanizm zarządzania czasem, który traktuje dostępny budżet obliczeniowy jako ścisłe ograniczenie, dynamicznie alokowane pomiędzy poszczególne fazy procesu.

Zarządzanie czasem odbywa się w sposób adaptacyjny:

- Pierwsza faza (eksploracja) otrzymuje wydzieloną część całkowitego budżetu, co wymusza szybkie odrzucanie słabych rozwiązań.
- Druga faza (weryfikacja) dysponuje czasem pozostałym po zakończeniu pierwszej, co pozwala na głębszą analizę, jeśli wstępna selekcja przebiegła sprawnie.
- Trzecia faza (agregacja) posiada sztywny limit czasowy, który przerywa proces dołączania kolejnych modeli do komitetu w momencie wyczerpania zasobów, gwarantując zwrócenie wyniku przed upływem limitu.

5.1 Etap 1: Szybka filtracja algorytmem Hyperband

Pierwszym etapem jest szeroka eksploracja przestrzeni rozwiązań z wykorzystaniem algorytmu Hyperband z pakietu Optuna. Modele trenowane są na zredukowanych zasobach, a algorytm w sposób ciągły monitoruje ich postępy. Konfiguracje, które nie rokują nadziei na poprawę wyników, są natychmiast przerywane (*pruning*), co pozwala zaoszczędzić cenny czas obliczeniowy dla bardziej obiecujących kandydatów.

Jako funkcję celu dla zadań klasyfikacji przyjęto *Weighted Log Loss*, co pozwala na poprawną obsługę zbiorów niezbalansowanych. Istotnym elementem naszej implementacji jest modyfikacja funkcji oceny końcowej. Zamiast polegać wyłącznie na średnim wyniku, system promuje stabilność, odejmując karę za zmienność wyników między foldami:

$$Score = \mu_{score} - \lambda_{stability} \cdot \sigma_{score}$$

gdzie σ_{score} to odchylenie standardowe wyników, a $\lambda_{stability}$ to konfigurowalny współczynnik kary. Mechanizm ten zapobiega promowaniu modeli, które osiągnęły wysoki wynik przypadkowo w wyniku specyficznego podziału danych.

5.2 Etap 2: Weryfikacja i ocena odporna (Robust Scoring)

Do drugiego etapu przechodzi ściśle wyselekcjonowana grupa kandydatów. Modele poddawane są pełnej walidacji krzyżowej (*Full CV*) bez mechanizmu wczesnego zatrzymywania. Aby zmieścić się w pozostałym budżecie czasowym, ograniczenie nakładane jest nie na czas treningu pojedynczego modelu, lecz na liczebność grupy kandydatów dopuszczonych do tego etapu.

W tym kroku następuje zmiana metryki decyzyjnej na *F1-weighted*, co premiuje modele lepiej separujące klasy decyzyjne. Aby uodpornić proces wyboru na wartości odstające (np. jeden nieudany fold walidacji), zastosowano *Robust Scoring* oparty na medianie i odchyleniu bezwzględnym:

$$RobustScore = Mediana(scores) - \lambda_{mad} \cdot MAD(scores)$$

gdzie MAD to *Median Absolute Deviation*. Modele są szeregowane wewnątrz swoich rodzin, a do następnego przechodzą tylko najlepsi reprezentanci każdej z grup, co zapewnia różnorodność algorytmiczną w kolejnym etapie.

5.3 Etap 3: Budowa komitetu (Greedy Ensemble z dywersyfikacją)

Ostatnim etapem jest konstrukcja finalnego predyktora metodą zachłannej selekcji (*Greedy Forward Selection*). Algorytm operuje na predykcjach typu *Out-Of-Fold* (OOF) wygenerowanych w poprzedniej fazie, co eliminuje konieczność ponownego trenowania modeli i znacząco oszczędza czas.

W tej fazie system działa w ścisłej pętli czasowej – po każdej iteracji sprawdzane jest, czy nie przekroczono globalnego limitu. Procedura dodaje do zespołu model, który najbardziej poprawia wynik, uwzględniając przy tym dywersyfikację:

$$EnsembleScore = (1 - w) \cdot Performance + w \cdot Diversity$$

gdzie *Diversity* definiowane jest poprzez średnią korelację predykcji. Takie podejście sprawia, że system preferuje dołączenie modelu popełniającego błędy na innych przykładach niż obecny komitet, nawet jeśli jego indywidualna skuteczność jest nieco niższa.

6 Generowanie ostatecznych predykcji modeli

W niniejszym rozdziale przedstawiono autorskie implementacje struktur typu *ensemble*, które zaprojektowano w celu zwiększenia zdolności generalizacyjnych systemu. Opisano trzy klasy realizujące różne strategie agregacji: od prostych komitetów głosujących, po zaawansowany stacking wielowarstwowy.

6.1 Komitety modeli z obsługą wielu widoków danych (klasa `Ensemble`)

Najprostszą, lecz jednocześnie elastyczną formą agregacji jest komitet modeli, w którym każdy estimator może być trenowany na innym zbiorze cech. Takie podejście umożliwia implementację uczenia wielowidokowego (*multi-view learning*), w którym poszczególne modele specjalizują się w różnych reprezentacjach danych.

Zaprojektowana klasa `Ensemble` obsługuje zarówno zadania klasyfikacji, jak i regresji, a także dwa tryby agregacji:

- **Hard voting** - decyzja podejmowana na podstawie głosowania większościowego predykcji klas.
- **Soft voting** - agregacja oparta na uśrednianiu rozkładów prawdopodobieństwa, co pozwala zachować więcej informacji o niepewności modeli.

Rozwiązanie to umożliwia łatwe eksperymentowanie z heterogenicznymi modelami oraz stanowi bazowy punkt odniesienia dla bardziej zaawansowanych architektur.

6.2 Dwupoziomowy stacking z predykcjami probabilistycznymi (klasa `Stacker`)

Kolejnym etapem rozwoju architektury ensemble jest klasyczny stacking, w którym predykcje modeli bazowych są wykorzystywane jako cechy wejściowe dla modelu meta. W zaimplementowanej klasie `Stacker` szczególny nacisk położono na:

- automatyczne wykorzystywanie predykcji probabilistycznych tam, gdzie są dostępne,
- spójne traktowanie klasyfikacji binarnej i wieloklasowej,
- możliwość trenowania modeli bazowych na różnych zbiorach danych.

Tak skonstruowany meta-model uczy się nie tylko średniej opinii zespołu, lecz także relatywnej wiarygodności poszczególnych estimatorów, co pozwala na bardziej precyzyjne łączenie ich predykcji.

6.3 Wielowarstwowy stacking inspirowany AutoGluon (klasa `PseudoAutoGluon`)

Najbardziej zaawansowanym rozwiązaniem zastosowanym w projekcie jest autorska implementacja wielowarstwowego stackingu, inspirowana architekturą odnoszącego sukcesu systemu AutoGluon. Zaprojektowany mechanizm umożliwia budowę głębokich struktur ensemble, w których kolejne warstwy modeli korzystają zarówno z oryginalnych cech wejściowych, jak i z predykcji wygenerowanych przez wcześniejsze warstwy. Kluczowe elementy tej architektury obejmują:

- **Hierarchiczny stacking warstwowy** — predykcje typu *out-of-fold* z każdej warstwy są iteracyjnie dołączane do przestrzeni cech, co pozwala modelom w głębszych warstwach uchwycić złożone zależności między danymi a predykcjami wcześniejszych modeli.
- **Ścisłą kontrolę przeuczenia** — wykorzystanie walidacji krzyżowej oraz predykcji OOF gwarantuje, że meta-modele nie uczą się na predykcjach wygenerowanych na tych samych próbkach, na których były trenowane.
- **Obsługę różnych typów zadań** — architektura automatycznie dostosowuje sposób propagacji predykcji do regresji, klasyfikacji binarnej oraz wieloklasowej.

Zaimplementowana klasa `PseudoAutoGluon` stanowi w pełni funkcjonalny, rozszerzalny system ensemble, który łączy zalety klasycznych algorytmów uczenia maszynowego z głęboką, wieloetapową agregacją predykcji.

7 Podsumowanie

W fazie testowania zaobserwowano, że budowa komitetu pozwala osiągnąć wyniki zauważalnie lepsze niż osiągane przez pojedynczy model. Wykorzystanie starannie wyselekcjonowanego portfolio pozwala na osiąganie dobrych wyników mniejszym nakładem czasowym w porównaniu do tego jakie byłoby potrzebne przy przeszukiwaniu przestrzeni możliwych hiperparametrów - szczególnie w przypadku komitetu modeli, dla którego to zadanie wymagałoby dużo większych zasobów.