

Politechnika Warszawska

W Y D Z I A Ł M A T E M A T Y K I
I N A U K I N F O R M A C Y J N Y C H



Praca dyplomowa inżynierska

na kierunku Inżynieria i Analiza Danych

Narzędzie do analizy podobieństwa rozkładów metryk jakości
algorytmów uczenia maszynowego w zadanej przestrzeni
hiperparametrów

Mateusz Grzyb

Numer albumu 298820

promotor

mgr Katarzyna Woźnica

konsultacje

dr hab. inż. Przemysław Biecek

WARSZAWA 2022

Streszczenie

Narzędzie do analizy podobieństwa rozkładów metryk jakości algorytmów uczenia maszynowego w zadanej przestrzeni hiperparametrów

Wartości hiperparametrów algorytmów uczenia maszynowego mają znaczący wpływ na jakość uzyskiwanych modeli. Dobór optymalnej konfiguracji algorytmu wymaga indywidualnego podejścia i jest zazwyczaj czasochłonny. Celem meta-learningu jest usprawnienie procesu optymalizacji hiperparametrów, wykorzystując wiedzę zgromadzoną podczas wcześniej przeprowadzonych eksperymentów. Jedną z obiecujących technik, opartych na tym założeniu, jest transfer konfiguracji pomiędzy zadaniami predykcyjnymi. Warunki efektywności transferu hiperparametrów, czyli transferowalności, nie są jednak dobrze poznane. Badanie ich stwarza zapotrzebowanie na uporządkowane składowanie wyników licznych eksperymentów obliczeniowych oraz przeprowadzanie zautomatyzowanych analiz. W niniejszej pracy zaproponowane zostało programistyczne narzędzie, które odpowiada na wymienione potrzeby (HyPerf). Przygotowane rozwiązanie ma postać pakietu języka programowania Python, składającego się z dwóch modułów. Moduł składowania oparty jest na rozwiązaniu bazodanowym, dostosowanym do uporządkowanego przechowywania oraz łatwego udostępniania wyników optymalizacji. Moduł analityczny automatyzuje analizy transferowalności hiperparametrów, które były najczęściej stosowane w dotychczasowej literaturze. Częścią pakietu jest również przykładowa baza danych, która została zasilona wynikami głównie z dziedziny medycyny.

Słowa kluczowe: uczenie maszynowe, optymalizacja hiperparametrów, transfer hiperparametrów, portfolio zestawów hiperparametrów, meta-learning

Abstract

Analysis of distributions of quality metrics of machine learning algorithms in a given hyperparameter space

The hyperparameter values of machine learning algorithms have a significant impact on the quality of the obtained models. The selection of the optimal algorithm configuration requires an individual approach and is usually time-consuming. The aim of meta-learning is to improve the process of optimizing hyperparameters, using knowledge accumulated from previous experiments. One promising technique, based on this premise, is configuration transfer between prediction tasks. However, the conditions for the effectiveness of hyperparameter transfer, or so called transferability, are not well understood. Examining them creates a demand for structured storage of the results of numerous computational experiments and for performing automated analyses. In this work, the proposed software tool (HyPerf) addresses the mentioned needs. The package is implemented in Python programming language and consists of two modules. The storage module is based on a database solution, adapted to the structured storage and easy sharing of optimization results. The analysis module automates transferability analyses of the hyperparameters that have been most commonly used in the literature to date. A sample database is also part of the package, which has been populated with results mainly from the medical field.

Keywords: machine learning, hyperparameter optimization, hyperparameter transfer, hyperparameter portfolio, meta-learning

Spis treści

1. Wprowadzenie	9
1.1. Motywacja	10
1.2. Wkład pracy	10
1.3. Powiązane prace	11
2. Transferowalność hiperparametrów	12
2.1. Uczenie maszynowe	12
2.2. Optymalizacja i transfer hiperparametrów	13
2.3. Analizy transferowalności	14
2.4. Metody wyznaczania portfolio	17
3. Opis rozwiązania	19
3.1. Wymagania wstępne	19
3.2. Wybór technologii	20
3.3. Moduł składowania	21
3.3.1. Struktura bazy danych	21
3.3.2. Interfejs programistyczny	28
3.3.3. Przykład użycia	29
3.4. Moduł analityczny	34
3.4.1. Rodzaje analiz	34
3.4.2. Implementacja	37
3.4.3. Przykład użycia	38
4. Testy rozwiązania	42
4.1. Testy jednostkowe	42
4.2. Przykładowa baza danych	43
4.3. Analiza demonstracyjna	44
5. Podsumowanie	47
5.1. Dalsze prace	47
Bibliografia	49
Dodatek A	53
Dodatek B	69

1. Wprowadzenie

Podstawowym celem uczenia maszynowego jest uzyskanie jak najlepszego modelu w rozumieniu trafności jego przewidywań. Liczne algorytmy uczenia maszynowego dobierają parametry tworzonych przez nie modeli w procesie automatycznej optymalizacji, za sprawą ekspozycji na dane treningowe. Na przebieg tego procesu wpływają również parametry samych algorytmów, zwane hiperparametrami, których wartości muszą zostać określone przez użytkownika. Konfiguracja hiperparametrów algorytmu może mieć znaczący wpływ na jakość uzyskiwanych modeli, ale ich optymalne wartości zależą od rozważanego problemu i często są trudne do określenia [22]. Optymalizacja hiperparametrów jest przykładem *meta-optymalizacji*, czyli optymalizacji zmiennych sterujących inną metodą optymalizacji, w celu podniesienia jej skuteczności.

Najstarsze i zarazem najprostsze metody automatycznej optymalizacji hiperparametrów - *grid search* oraz *random search* [1], są metodami typu *brute force*, ponieważ opierają się na prostym przeszukiwaniu wyczerpującym. Nie wykorzystują one wiedzy zdobytej podczas obecnego procesu optymalizacji, ani też pochodzącej z poprzednich. Przyczynia się to do ich dużej złożoności obliczeniowej oraz ograniczonej efektywności, szczególnie w przypadku wielowymiarowych przestrzeni hiperparametrów, co powszechnie określane jest mianem *przekleństwa wymiarowości*.

Algorytmy *Bayesian optimization* [13, 24], *iterated F-racing* [3] oraz *Evolutionary optimization* [2] są przedstawicielami grupy metod automatycznej optymalizacji hiperparametrów wykorzystujących wiedzę o charakterystyce problemu optymalizacyjnego, pozyskaną w trakcie jego rozwiązywania. Podejście to, określane mianem optymalizacji w sposób *online* [8], ma na celu balansowanie pomiędzy testowaniem obiecujących kandydatów na rozwiązanie oraz eksploracją słabo poznanych obszarów przeszukiwanej przestrzeni. Dzięki temu, procedura optymalizacji stopniowo dopasowuje się do rozważanego zbioru danych, co przeważnie owocuje jej wyższą efektywnością. Efekt ten zachodzi dopiero w późniejszym etapie działania algorytmu, dlatego takie algorytmy nie mają właściwości dobrego *anytime performance*, czyli zdolności do podania zadowalającego rozwiązania po zaledwie kilku iteracjach optymalizacji. W przypadku podstawowych wersji tych metod wiedza pochodząca z optymalizacji dla innych problemów nie jest wykorzystywana, a zatem każde uruchomienie optymalizacji wiąże się z akumulacją doświadczeń od zera.

Odmienne podejście do rozważanego problemu reprezentują metody oparte o transfer optymalnych konfiguracji hiperparametrów między zadaniami predykcyjnymi. Polega to na konstrukcji portfolio zestawów hiperparametrów, czyli określeniu kolejności ich sprawdzania, w oparciu o obliczone już dla innych zadań predykcyjnych rezultaty. Jest to przykład techniki z dziedziny zwanej *meta-learning*, określanej też w bardziej wymowny sposób jako *learning to learn* [25]. Pojęcie to dotyczy wszelkich rozwinięć uczenia maszynowego, opartych o doświadczenia związane z samym procesem uczenia. Wistuba, Schilling i Schmidt-Thieme [28] pokazują, że opisywane w niniejszym akapicie procedury mogą być równie efektywne, jak wspomniane algorytmy *online*, przy jednoczesnym zachowaniu niższej złożoności obliczeniowej oraz lepszego *anytime performance*, ale warunkiem jest wykorzystanie bazy doświadczeń adekwatnej do rozważanego problemu.

We wprowadzeniu oraz w Sekcji 1.1 przedstawiona została szersza perspektywa i motywacja stojąca za badaniem zjawiska transferu hiperparametrów. Wkład niniejszej pracy w rozważaną dziedzinę został opisany w Sekcji 1.2. Przegląd powiązanych z rozważanym zagadnieniem prac został zawarty w Sekcji 1.3. W Rozdziale 2 zdefiniowane zostały zagadnienie transferowalności hiperparametrów oraz metody jego badania. Rozdział 3 opisuje opracowane narzędzie, natomiast w Rozdziale 4 przedstawione są testy zaimplementowanego pakietu. Podsumowanie niniejszej pracy znajduje się w Rozdziale 5.

1.1. Motywacja

Zagadnienie transferowalności hiperparametrów jest tematem wspólnie prowadzonych badań [28, 21], ale stanowi również podstawę bardziej praktycznych projektów. Przykładem tego są niektóre z najnowszych narzędzi realizujących ideę *AutoML*, czyli automatyzacji całego procesu przygotowania modeli uczenia maszynowego, które opierają optymalizację hiperparametrów o ich transfer między zadaniami predykcyjnymi [17, 8]. Rozwiązanie tego typu, noszące nazwę *Auto-Sklearn 2.0*, jest zwycięzcą drugiej edycji konkursu *ChaLearn AutoML Challenge* [10]. Fakt aktywnego wykorzystywania tej techniki z zadowalającymi skutkami jest najlepszym argumentem przemawiającym za potrzebami lepszego poznania zjawiska, na którym jest ona oparta, oraz składowania wyników czasochłonnych obliczeń, które okazały się wartościowe. Transfer konfiguracji algorytmów uczenia maszynowego tworzy szczególne możliwości dla podmiotów budujących duże ilości modeli, a także społeczności chętnych do współdzielenia uzyskiwanych wyników.

Według Vanschoren [25], zgodnie z intuicją, występuje korelacja między efektywnością transferu hiperparametrów, a podobieństwem rozważanych zadań predykcyjnych. Nie można oczekiwać korzyści z analogii do problemów w żaden sposób niezwiązanych z rozważanym [9]. Jednakże, bez jednoznacznych odpowiedzi w literaturze pozostają pytania takie, jak "Kiedy transfer zestawów hiperparametrów między zadaniami predykcyjnymi jest efektywny?" oraz "Jakie metody tworzenia portfolio konfiguracji hiperparametrów są skuteczne?"

Badanie zjawiska transferowalności hiperparametrów stawia liczne wyzwania. Koniecznością staje się składowanie wyników licznych eksperymentów obliczeniowych w uporządkowanej formie, która będzie wspierać ich reprodukowalność oraz współdzielenie z innymi badaczami. Potrzebne są również zautomatyzowane procedury analizy tychże wyników pod kątem podobieństwa rozkładów metryk jakości modeli w zadanych przestrzeniach hiperparametrów oraz skuteczności poszczególnych metod tworzenia portfolio zestawów konfiguracji. Istnieje zatem zapotrzebowanie na oprogramowanie realizujące wspomniane zadania.

1.2. Wkład pracy

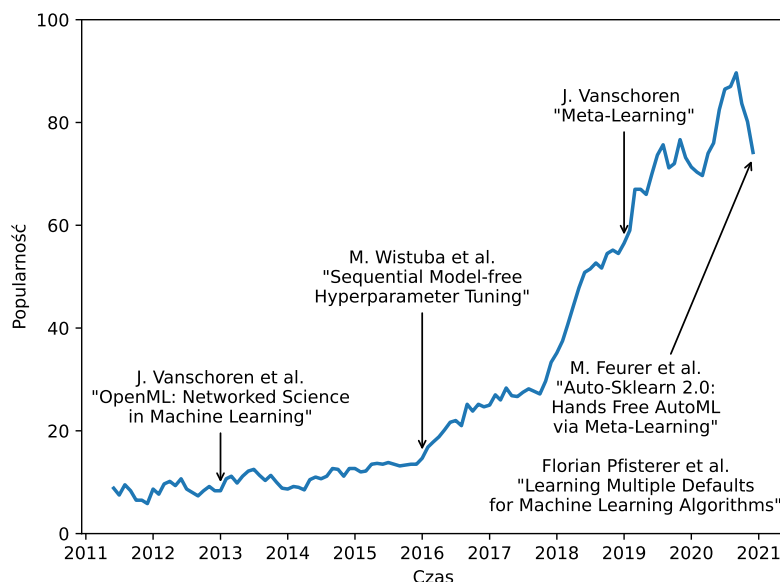
Niniejszą pracę stanowi narzędzie ułatwiające uporządkowane składowanie wyników eksperymentów obliczeniowych, dotyczących optymalizacji hiperparametrów dla różnych zadań predykcyjnych, algorytmów uczenia maszynowego, siatek hiperparametrów (*hyperparameter grids*) i metryk jakości modeli, oraz pozwalające analizować je pod kątem transferowalności konfiguracji. Nie istnieje uznany schemat wspomnianej analizy, dlatego wybrane zostały metody najczęściej stosowane w dotychczasowej literaturze. W trakcie projektowania narzędzia wzięto pod uwagę wymagania wspierania reprodukowalności wyników oraz łatwego ich współdzielenia z innymi badaczami. Przygotowane rozwiązanie ma na celu wsparcie badaczy uczenia maszynowego

w staraniach nad lepszym poznaniem praw rządzących zjawiskiem transferowalności konfiguracji między zadaniami predykcyjnymi. Całość została opracowana jako wygodny w użyciu pakiet kodu popularnego w dziedzinie uczenia maszynowego języka programowania *Python* (*HyPerf*).

Dodatkowo, przykładowa baza danych została uzupełniona wynikami dla zadań predykcyjnych głównie z dziedziny medycyny. Część tych danych pochodzi z projektów *metaMIMIC* [29], współtworzonego przeze mnie w ramach stażu wakacyjnego pod okiem promotora i opiekuna naukowego niniejszej pracy, oraz *MementoML* [16], a pozostałe zostały przygotowane w trakcie realizacji projektu. Wspomniane dane posłużyły do przetestowania przygotowanego rozwiązania.

1.3. Powiązane prace

Zgodnie z moją wiedzą, nie są dostępne inne rozwiązania, próbujące odpowiedzieć na te same potrzeby, co przygotowane narzędzie. Prowadzone są jednak inne badania, które są związane z tematem niniejszej pracy. Przytoczone zostały już projekty z dziedziny *AutoML*, którym wykorzystanie transferu hiperparametrów pozwoliło osiągnąć obiecujące wyniki. Istotny wkład w badanie zjawiska transferowalności mają również prace związane z składowaniem i udostępnianiem różnych zadań predykcyjnych w celach porównawczych i eksperymentalnych. Repozytoria przykładowych problemów [26] oraz benchmarki przeznaczone dla testowania różnych metod optymalizacji hiperparametrów [7] porządkują oparte na nich badania oraz wspierają reprodukowalność publikowanych wyników. Przygotowane rozwiązanie jest utrzymane w duchu obecnie dostępnych, uzupełniając je o podobne możliwości, ale dla samych wyników eksperymentów obliczeniowych. Wpisuje się ono również w narastającą popularność tematu ogólnie rozumianej optymalizacji hiperparametrów (Rysunek 1.1).



Rysunek 1.1: Wzrost popularności hasła "hyperparameter optimization" w wyszukiwarce *Google* na przestrzeni ostatniej dekady. Dane pochodzą z platformy *Google Trends* i zostały wygładzone średnią ruchomą o okresie pół roku. Za pomocą adnotacji zaznaczono istotne, z punktu widzenia badań nad transferowalnością hiperparametrów, publikacje naukowe.

2. Transferowalność hiperparametrów

Prezentowane rozwiązanie ma przeznaczenie badawcze i jest związane z eksploracją stosunkowo nowego zagadnienia, jakim jest transferowalność hiperparametrów. W związku z tym, znaczenie niektórych, istotnych z punktu widzenia opisu narzędzia pojęć może nie być oczywiste. W celu zapewnienia możliwie dobrego zrozumienia niniejszego dokumentu, w tym rozdziale znajdują się objaśnienia określeń i oznaczeń, które używane są w dalszej jego części. Zostały one podzielone na kilka sekcji, które dotyczą kolejno podstaw uczenia maszynowego, optymalizacji i transferu hiperparametrów, analiz transferowalności konfiguracji oraz metod konstrukcji tak zwanych portfolio zestawów hiperparametrów.

2.1. Uczenie maszynowe

Ta sekcja zawiera podstawowe definicje używane, w dziedzinie uczenia maszynowego. Niektóre sformułowania podane są w uproszczonej postaci, ponieważ ich celem jest nadanie intuicji oraz nakreślenie sposobu użycia przy dalszym opisie rozwiązania, a nie zbudowanie sformalizowanej teorii. Podane w definicjach skrótowe oznaczenia są używane w dalszych sekcjach tego rozdziału.

Problemy najczęściej rozwiązywane poprzez zastosowanie uczenia maszynowego nazywane są zadaniami predykcyjnymi (Definicja 2.1). Dzielią się one na różne rodzaje (Definicja 2.2).

Definicja 2.1 (Zadanie predykcyjne). Zadanie polegające na przewidzeniu wartości nieobserwowanej zmiennej, na podstawie wartości innych zmiennych. Na zadanie predykcyjne składają się przykładowe dane, pełniące zarówno rolę uczącą, jak i weryfikacyjną. Zadanie predykcyjne oznaczane jest jako \mathcal{D} .

Definicja 2.2 (Typ zadania predykcyjnego). Typ określany na podstawie rodzaju objaśnianej zmiennej. Przykładami są klasyfikacja (zmienna nominalna) lub regresja (zmienna liczbowa).

W określonym zadaniu predykcyjnym obserwowany jest wyłącznie pewien podzbiór obserwacji, natomiast jego rozwiązanie powinno uogólniać się na nowe dane. Przeważnie określa się pewien schemat estymacji jakości predyktora, np. podział na zbiór treningowy i testowy lub tak zwaną walidację krzyżową (*cross-validation*). Dla uproszczenia, w tej pracy wspomniany schemat traktowany jest jako integralna część zadania predykcyjnego.

Sposobem na rozwiązywanie zadań predykcyjnych są algorytmy uczenia maszynowego (Definicja 2.3). Algorytmy te wymagają konfiguracji poprzez ręczne określenie wartości ich wewnętrznych parametrów, zwanych hiperparametrami (Definicje 2.4 i 2.5).

Definicja 2.3 (Algorytm uczenia maszynowego). Algorytm pozwalający na tworzenie matematycznych modeli problemów predykcyjnych, poprzez automatyczny dobór ich parametrów, dzięki ekspozycji na dane uczące. Algorytm uczenia maszynowego oznaczany jest jako \mathcal{A} .

Definicja 2.4 (Hiperparametr). Parametr algorytmu uczenia maszynowego sterujący procesem uczenia. Skończenie wymiarowa przestrzeń hiperparametrów algorytmu uczenia maszynowego \mathcal{A} oznaczana jest jako $\Lambda_{\mathcal{A}} = \Lambda_1 \times \dots \times \Lambda_n$.

Definicja 2.5 (Zestaw hiperparametrów). Krotka wartości hiperparametrów pozwalająca na skonfigurowanie algorytmu uczenia maszynowego na jej podstawie. Zestaw hiperparametrów przeznaczony dla algorytmu uczenia maszynowego \mathcal{A} oznaczany jest jako $\lambda \in \Lambda_{\mathcal{A}}$.

Efektem działania algorytmów uczenia maszynowego na danych uczących są matematyczne modele problemów predykcyjnych (Definicja 2.6). Sposobem ewaluacji modeli uczenia maszynowego dla danych weryfikacyjnych są liczne metryki jakości (Definicja 2.7).

Definicja 2.6 (Model uczenia maszynowego). Matematyczny model problemu predykcyjnego pozwalający na wykonywanie predykcji, będący efektem działania algorytmu uczenia maszynowego. Model uczenia maszynowego powstały w wyniku użycia algorytmu \mathcal{A} , skonfigurowanego zestawem hiperparametrów $\lambda \in \Lambda_{\mathcal{A}}$, oznaczany jest jako $\mathcal{M} = \mathcal{A}_{\lambda}(\mathcal{D})$.

Definicja 2.7 (Metryka jakości modeli). Funkcja ewaluująca jakość matematycznego modelu problemu predykcyjnego pod kątem trafności jego predykcji. Metryka jakości modeli oznaczana jest jako \mathcal{S} , a jej wartość dla modelu uczenia maszynowego \mathcal{M} i zadania predykcyjnego \mathcal{D} oznaczana jest jako $\mathcal{S}(\mathcal{M}, \mathcal{D})$.

Wpływ na adekwatność uzyskiwanych matematycznych modeli problemów predykcyjnych mają nie tylko jakość danych uczących oraz wybór algorytmu uczenia maszynowego, ale również odpowiednia jego konfiguracja. Proces strojenia konfiguracji algorytmu uczenia maszynowego z rozważanym zadaniem predykcyjnym nazywany jest optymalizacją hiperparametrów.

2.2. Optymalizacja i transfer hiperparametrów

Ta sekcja zawiera definicję problemu optymalizacji hiperparametrów (Definicja 2.8), definicję jednego z sposobów usprawnienia jego rozwiązywania, jakim jest transfer konfiguracji między zadaniami predykcyjnymi (Definicje 2.9 i 2.10), oraz definicje innych pojęć, związanych z transferem hiperparametrów. Zagadnienia te są kluczowe z punktu widzenia przedstawianego rozwiązania, ponieważ ma ono na celu usprawnienie pracy badaczy uczenia maszynowego nad lepszym poznaniem zasad nimi rządzących.

Definicja 2.8 (Optymalizacja hiperparametrów). Proces poszukiwania zestawu hiperparametrów, dla którego rozważany algorytm uczenia maszynowego zwraca możliwie najlepszy model, w terminach metryki jakości modeli, dla danego zadania predykcyjnego. Optymalizacja hiperparametrów pod kątem metryki jakości modeli \mathcal{S} może być przedstawiona jako $\arg \max_{\lambda \in \Lambda_{\mathcal{A}}} \mathcal{S}(\mathcal{A}_{\lambda}(\mathcal{D}), \mathcal{D})$ (lub $\arg \min$, w zależności od charakterystyki metryki \mathcal{S}).

Definicja 2.9 (Transfer hiperparametrów). Proces wykorzystania zestawów hiperparametrów, które okazały się być optymalne dla pewnego zbioru zadań predykcyjnych, przy tworzeniu modelu dla nowego zadania predykcyjnego, w celu uzyskania dobrych wyników.

Definicja 2.10 (Transferowalność hiperparametrów). Efektywność procesu transferu hiperparametrów wynikająca z doboru historycznych wyników oraz strategii przenoszenia konfiguracji między zadaniami predykcyjnymi.

Z transferem konfiguracji ściśle związane są pojęcia siatki (Definicja 2.11) oraz portfolio zestawów hiperparametrów (Definicja 2.12). Siatka hiperparametrów jest prostym sposobem dyskretyzacji ciągłej przestrzeni konfiguracji, natomiast zbudowane na jej podstawie portfolio to metoda na wykorzystanie doświadczeń, nabytych w wyniku tworzenia i ewaluacji modeli.

Definicja 2.11 (Siatka hiperparametrów). Zbiór zestawów hiperparametrów przeznaczonych dla danego algorytmu uczenia maszynowego. Siatka hiperparametrów, określona dla algorytmu uczenia maszynowego \mathcal{A} , oznaczana jest jako $\mathcal{G} = (\lambda_i)_{i=1,\dots,p}$, $\forall_{i \in [1,\dots,p]} \lambda_i \in \Lambda_{\mathcal{A}}$.

Definicja 2.12 (Portfolio zestawów hiperparametrów). Uporządkowany podzbiór siatki hiperparametrów wyznaczony na podstawie wyników obliczonych dla pewnego zbioru zadań predykcyjnych oraz określający potencjalnie optymalną kolejność sprawdzania zestawów hiperparametrów. Portfolio zestawów hiperparametrów oznaczane jest jako \mathcal{P} .

Przeniesienie konfiguracji między zadaniami predykcyjnymi jest efektywnym sposobem optymalizacji hiperparametrów algorytmu uczenia maszynowego, ale tylko w przypadku użycia adekwatnej bazy doświadczeń w połączeniu z efektywną metodą konstrukcji portfolio zestawów hiperparametrów. W celu badania czynników wpływających na spełnienie tych warunków, przydatne są różne metody analityczne, zbiorczo nazywane metodami analizy transferowalności, które zostały szczegółowo opisane w kolejnej sekcji. Ponadto, w Sekcji 2.4 opisane zostały dostępne w rozwiązaniu metody konstrukcji portfolio zestawów hiperparametrów.

2.3. Analizy transferowalności

Ta sekcja zawiera definicje metod analizy transferowalności hiperparametrów, które były najczęściej stosowane w dotychczasowej literaturze. Można podzielić je na dwie grupy - miary podobieństwa rankingów zestawów hiperparametrów (Definicje 2.13–2.16) oraz miary związane z symulacją procesu transferu hiperparametrów (Definicje 2.17 i 2.18).

Jednym z najprostszych sposobów analizy transferowalności hiperparametrów jest porównanie podobieństwa rankingów zestawów konfiguracji, należących do tej samej siatki, dla pary zadań predykcyjnych. Użytecznym narzędziem są tutaj wszelkie matematyczne miary podobieństwa rankingów. Definicje 2.13–2.16 opisują cztery miary tego typu, które pochodzą z [4], gdzie zostały podane w kontekście porównywania rankingów ekspresji genów. Wartości wszystkich miar należą do przedziału $[0, 1]$ oraz wartości większe świadczą o większym podobieństwie rankingów.

Dla Definicji 2.13–2.16 przyjmijmy zbiorcze oznaczenia. Niech \mathcal{D} i \mathcal{D}' oznaczają zadania predykcyjne tego samego typu, \mathcal{G} siatkę p zestawów hiperparametrów, określoną dla algorytmu uczenia maszynowego \mathcal{A} , a \mathcal{S} metrykę jakości modeli. Rankingiem zestawów hiperparametrów

$\lambda \in \mathcal{G}$ nazywamy permutację $r = (r_i)_{i=1,\dots,p}$ ciągu $(1, \dots, p)$, gdzie r_i jest rangą zestawu hiperparametrów λ_i w odniesieniu do wartości metryki jakości modeli $\mathcal{S}(\mathcal{A}_\lambda(\mathcal{D}), \mathcal{D})$. Niech r i r' oznaczają rankingi zestawów hiperparametrów dla zadań predykcyjnych odpowiednio \mathcal{D} i \mathcal{D}' .

Pierwszą miarą podobieństwa rankingów jest *Percentage of Overlap* (Definicja 2.13). Metryka ta określa podobieństwo rankingów wyłącznie na podstawie ich górnych części, do głębokości zadanej przez parametr k . Jej wartość określona jest jako proporcja liczności przecięcia rozważanych zbiorów i wspomnianej głębokości (czyli wartości parametru k).

Definicja 2.13 (Percentage of Overlap). Wartość miary podobieństwa rankingów zestawów hiperparametrów *Percentage of Overlap* wyraża się wzorem:

$$s^{\text{PO}}(r, r', k) = \frac{\sum_{i=1}^p \mathbb{1}(r_i \leq k \wedge r'_i \leq k)}{k}, \quad (2.1)$$

gdzie k jest parametrem i $k \in \mathbb{N}$, $k \leq p$.

Miara podobieństwa rankingów *Correspondence at the Top* (Definicja 2.14) stanowi niewielką modyfikację zdefiniowanej powyżej miary *Percentage of Overlap*. W jej przypadku liczność przecięcia rozważanych zbiorów jest dzielona nie przez wartość parametru głębokości k , a przez liczność ich sumy.

Definicja 2.14 (Correspondence at the Top). Wartość miary podobieństwa rankingów zestawów hiperparametrów *Correspondence at the Top* wyraża się wzorem:

$$s^{\text{CT}}(r, r', k) = \frac{\sum_{i=1}^p \mathbb{1}(r_i \leq k \wedge r'_i \leq k)}{2 \cdot k - \sum_{i=1}^p \mathbb{1}(r_i \leq k \wedge r'_i \leq k)}, \quad (2.2)$$

gdzie k jest parametrem i $k \in \mathbb{N}$, $k \leq p$.

Bardziej złożonym rozwinięciem miary podobieństwa rankingów *Percentage of Overlap* jest agregująca wiele jej wartości miara *Overlap Score* (Definicja 2.15). Wspomniana agregacja odbywa się na zasadzie średniej ważonej, gdzie podobieństwo głębszych, potencjalnie bardziej zaszuflonych części rankingów rozważane jest z mniejszą istotnością. Mocą opisanego “wygaszania” steruje parametr α . Definicja tej miary została zmodyfikowana przeze mnie o normalizację.

Definicja 2.15 (Overlap Score). Wartość miary podobieństwa rankingów zestawów hiperparametrów *Overlap Score* wyraża się wzorem:

$$\begin{aligned} s^{\text{OS}}(r, r', \alpha) &= \frac{\sum_{k=1}^p e^{-\alpha \cdot k} \cdot \sum_{i=1}^p \mathbb{1}(r_i \leq k \wedge r'_i \leq k)}{\sum_{k=1}^p e^{-\alpha \cdot k} \cdot k} \\ &= \frac{\sum_{k=1}^p e^{-\alpha \cdot k} \cdot \sum_{i=1}^p \mathbb{1}(r_i \leq k \wedge r'_i \leq k)}{\frac{e^{-\alpha \cdot p} \cdot (e^{\alpha \cdot (p+1)} - e^{\alpha \cdot (p+1)+p})}{(e^\alpha - 1)^2}}, \end{aligned} \quad (2.3)$$

gdzie α jest parametrem i $\alpha \in \mathbb{R}$, $\alpha > 0$.

Ostatnia miara podobieństwa rankingów, zwana *Canberra Distance* (Definicja 2.16), oparta jest na metryce odległości o tej samej nazwie. Ma ona również właściwość rozważania dalszych pozycji z mniejszymi wagami, ale nie jest zależna od wartości żadnych parametrów. Definicja

tej miary została zmodyfikowana przez mnie o normalizację oraz odwrócenie monotoniczności. Modyfikację oparłem o wzór na maksymalną permutację wspomnianej metryki, który został podany w [14].

Definicja 2.16 (Canberra Distance). Wartość miary podobieństwa rankingów zestawów hiperparametrów *Canberra Distance* wyraża się wzorem:

$$s^{\text{CD}}(r, r') = 1 - \frac{\sum_{i=1}^p \frac{|r_i - r'_i|}{r_i + r'_i}}{\sum_{i=1}^p \frac{|t_i - t'_i|}{t_i + t'_i}}, \quad (2.4)$$

gdzie

$$t = \begin{cases} \begin{pmatrix} 1 & 2 & \cdots & \frac{p}{2} & \frac{p}{2} + 1 & \cdots & p \end{pmatrix} & \text{jeśli } p \text{ parzyste,} \\ \begin{pmatrix} 1 & 2 & \cdots & \frac{p-1}{2} & \frac{p-1}{2} + 1 & \cdots & p \end{pmatrix} & \text{jeśli } p \text{ nieparzyste,} \end{cases}$$

$$t' = \begin{cases} \begin{pmatrix} \frac{p}{2} + 1 & \frac{p}{2} + 2 & \cdots & p & 1 & \cdots & \frac{p}{2} \end{pmatrix} & \text{jeśli } p \text{ parzyste,} \\ \begin{pmatrix} \frac{p-1}{2} + 1 & \frac{p-1}{2} + 2 & \cdots & p & 1 & \cdots & \frac{p-1}{2} \end{pmatrix} & \text{jeśli } p \text{ nieparzyste.} \end{cases}$$

W przypadku rozważania optymalizacji hiperparametrów spośród skończonego zbioru możliwości oraz ich transferu dla wielu zadań predykcyjnych jednocześnie, przydatnym narzędziem są miary agregujące, które pozwalają na wyrażenie efektywności wielokrotnego przenoszenia konfiguracji za pomocą skalarnej wartości. Przykładami takich miar są miara *Average Normalized Error* (Definicja 2.17) oraz pochodząca od niej miara *Cumulative Average Normalized Error* (Definicja 2.18), które zostały zaproponowane w [28].

Dla Definicji 2.17 i 2.18 przyjmijmy zbiorcze oznaczenia. Przypomnijmy, niech \mathbf{D} oznacza skończony zbiór zadań predykcyjnych tego samego typu, \mathcal{S} metrykę jakości modeli, a \mathcal{P}_t pierwsze t elementów portfolio zestawów hiperparametrów \mathcal{P} , będącego permutacją siatki p zestawów hiperparametrów \mathcal{G} , przeznaczonej dla algorytmu uczenia maszynowego \mathcal{A} .

Definicja 2.17 (Average Normalized Error). Jeżeli większa wartość metryki jakości modeli oznacza lepszy model, to wartość miary skuteczności transferu hiperparametrów *Average Normalized Error*, po sprawdzeniu pierwszych t spośród p zestawów hiperparametrów, wyraża się wzorem:

$$\text{ANE}(\mathbf{D}, \mathcal{P}_t) = \frac{1}{|\mathbf{D}|} \sum_{\mathcal{D} \in \mathbf{D}} \frac{\mathcal{S}^{\max}(\mathcal{D}) - \max_{\lambda \in \mathcal{P}_t} \mathcal{S}(\mathcal{D})}{\mathcal{S}^{\max}(\mathcal{D}) - \mathcal{S}^{\min}(\mathcal{D})}, \quad (2.5)$$

w przeciwnym przypadku:

$$\text{ANE}(\mathbf{D}, \mathcal{P}_t) = \frac{1}{|\mathbf{D}|} \sum_{\mathcal{D} \in \mathbf{D}} \frac{\min_{\lambda \in \mathcal{P}_t} \mathcal{S}(\mathcal{D}) - \mathcal{S}^{\min}(\mathcal{D})}{\mathcal{S}^{\max}(\mathcal{D}) - \mathcal{S}^{\min}(\mathcal{D})}, \quad (2.6)$$

gdzie

$\mathcal{S}(\mathcal{D}) = \mathcal{S}(\mathcal{A}_\lambda(\mathcal{D}), \mathcal{D})$ - skrócony zapis wartości metryki jakości dla konfiguracji λ ,

$\mathcal{S}^{\max}(\mathcal{D}) = \max_{\lambda \in \mathcal{P}} \mathcal{S}(\mathcal{D})$ - największa wartość metryki jakości na całym portfolio,

$\mathcal{S}^{\min}(\mathcal{D}) = \min_{\lambda \in \mathcal{P}} \mathcal{S}(\mathcal{D})$ - najmniejsza wartość metryki jakości na całym portfolio.

Miara *Cumulative Average Normalized Error* (Definicja 2.18) jest sposobem na akumulację wspomnianej miary *Average Normalized Error* pomiędzy różnymi licznosciami t portfolio zestawów hiperparametrów.

Definicja 2.18 (Cumulative Average Normalized Error). Wartość miary skuteczności transferu hiperparametrów *Cumulative Average Normalized Error* wyraża się wzorem:

$$\text{CANE}(\mathbf{D}, \mathcal{P}) = \sum_{t=1}^p \text{ANE}(\mathbf{D}, \mathcal{P}_t). \quad (2.7)$$

W dalszej części pracy używane są spolszczone nazwy miar zdefiniowanych w 2.17 i 2.18 - (skumulowany) średni znormalizowany błąd strojenia.

2.4. Metody wyznaczania portfolio

Ta sekcja zawiera definicje dostępnych w rozwiązaniu metod wyznaczania portfolio zestawów hiperparametrów. Przypomnijmy, że konstruowanie portfolio konfiguracji polega na stworzeniu uporządkowanego podzbioru rozważanej siatki hiperparametrów, na podstawie wyników obliczonych dla pewnego zbioru zadań predykcyjnych. Ma to na celu zaproponowanie możliwie optymalnej kolejności testowania konfiguracji w trakcie strojenia algorytmu uczenia maszynowego.

Dla Definicji 2.19–2.21 przyjmijmy zbiorcze oznaczenia. Ponownie, niech \mathbf{D} oznacza skończony zbiór zadań predykcyjnych tego samego typu, \mathcal{G} siatkę hiperparametrów, przeznaczoną dla algorytmu \mathcal{A} , \mathcal{S} metrykę jakości modeli, a $r_{\mathcal{D}}(\lambda, \mathcal{G})$ rangę zestawu hiperparametrów λ w siatce \mathcal{G} w odniesieniu do wartości metryki jakości modeli $\mathcal{S}(\mathcal{A}_{\lambda}(\mathcal{D}), \mathcal{D})$.

Metoda *Simple* (Definicja 2.19) została zainspirowana [5] i może zostać określona podejściem naiwnym. Wynika to z faktu, że w pierwszej kolejności wybiera ona zestawy hiperparametrów, które osiągnęły sumarycznie najlepsze wyniki dla rozważanego zbioru zadań predykcyjnych, pełniąc rolę bazy doświadczeń.

Definicja 2.19 (Simple). Metoda konstrukcji portfolio *Simple* polega na uporządkowaniu (rosnąco lub malejąco, w zależności od charakterystyki metryki jakości modeli \mathcal{S}) zestawów hiperparametrów $\lambda \in \mathcal{G}$ w odniesieniu do wartości:

$$\text{perf}(\lambda, \mathbf{D}) = \sum_{\mathcal{D} \in \mathbf{D}} \mathcal{S}(\mathcal{D}), \quad (2.8)$$

lub, z uwzględnieniem opcjonalnego skalowania, w odniesieniu do wartości:

$$\text{perf}(\lambda, \mathbf{D}) = \sum_{\mathcal{D} \in \mathbf{D}} \frac{\mathcal{S}(\mathcal{D})}{\mathcal{S}^{\max}(\mathcal{D}) - \mathcal{S}^{\min}(\mathcal{D})}, \quad (2.9)$$

gdzie

$\mathcal{S}(\mathcal{D}) = \mathcal{S}(\mathcal{A}_{\lambda}(\mathcal{D}), \mathcal{D})$ - skrócony zapis wartości metryki jakości dla konfiguracji λ ,

$\mathcal{S}^{\max}(\mathcal{D}) = \max_{\lambda \in \mathcal{P}} \mathcal{S}(\mathcal{D})$ - największa wartość metryki jakości na całym portfolio,

$\mathcal{S}^{\min}(\mathcal{D}) = \min_{\lambda \in \mathcal{P}} \mathcal{S}(\mathcal{D})$ - najmniejsza wartość metryki jakości na całym portfolio.

Metody *CANE Optimal Sequence* (Definicja 2.20) oraz *Average SMFO* (Definicja 2.21) to bardziej wyszukane algorytmy konstrukcji portfolio zestawów hiperparametrów, które zostały zaproponowane w [28]. Podobnie do metody *Simple*, oba opisywane algorytmy podążają za prostą intuicją wybierania zestawów hiperparametrów, które okazały się być najlepsze dla rozważanych zadań predykcyjnych. Istotną różnicą jest fakt, że metody te starają się jednocześnie maksymalnie różnicować proponowane konfiguracje pod względem wyników, osiągniętych dla poszczególnych zadań predykcyjnych. Możliwe jest to dzięki rozważaniu częściowych wartości metryki jakości modeli, a nie wyłącznie jej agregatów.

Algorytm *CANE Optimal Sequence* zatrzymuje się po wybraniu wszystkich zestawów hiperparametrów, które były najlepsze dla poszczególnych zadań predykcyjnych, dlatego nie gwarantuje on uzyskania pełnej permutacji rozważanej siatki hiperparametrów. O metodzie *Average SMFO* można myśleć jak o wywoływaniu algorytmu *CANE Optimal Sequence* z restartami, aż do momentu uzyskania nowej kolejności dla wszystkich rozważanych konfiguracji.

Definicja 2.20 (CANE Optimal Sequence). Metoda konstrukcji portfolio zestawów hiperparametrów *CANE Optimal Sequence* wyraża się algorytmem:

Algorytm 1 CANE Optimal Sequence

Wejście: siatka hiperparametrów \mathcal{G} , skończony zbiór zadań predykcyjnych \mathbf{D} , limit prób $T \leq |\mathcal{G}|$

Wyjście: sekwencja zestawów hiperparametrów o długości $\leq T$

```

 $\mathcal{P}_0 \leftarrow ()$ 
for  $t = 1$  to  $T$  do
     $\lambda_t \leftarrow \arg \min_{\lambda \in \mathcal{G}} \sum_{\mathcal{D} \in \mathbf{D}} \min_{\lambda' \in \mathcal{P}_{t-1} \cup \{\lambda\}} r_{\mathcal{D}}(\lambda', \mathcal{G})$ 
     $\mathcal{P}_t \leftarrow (\mathcal{P}_{t-1}, \lambda_t)$ 
    if  $\frac{1}{|\mathbf{D}|} \sum_{\mathcal{D} \in \mathbf{D}} \min_{\lambda \in \mathcal{P}_t} r_{\mathcal{D}}(\lambda, \mathcal{G})$  then
        return  $\mathcal{P}_t$ 
    end if
end for
return  $\mathcal{P}_T$ 
    
```

Definicja 2.21 (Average SMFO). Niech Algorytm1 oznacza algorytm przedstawiony w Definicji 2.20. Metoda konstrukcji portfolio zestawów hiperparametrów *Average SMFO* wyraża się algorytmem:

Algorytm 2 Average SMFO

Wejście: siatka hiperparametrów \mathcal{G} , skończony zbiór zadań predykcyjnych \mathbf{D}

Wyjście: portfolio zestawów hiperparametrów \mathcal{P}

```

 $\mathcal{P} \leftarrow ()$ 
 $T \leftarrow |\mathcal{G}|$ 
while  $T > 0$  do
     $\mathcal{P}' \leftarrow \text{Algorytm1}(\mathcal{G} \setminus \mathcal{P}, T, \mathbf{D})$ 
     $T \leftarrow T - |\mathcal{P}'|$ 
     $\mathcal{P} \leftarrow (\mathcal{P}, \mathcal{P}')$ 
end while
return  $\mathcal{P}$ 
    
```

3. Opis rozwiązania

Przygotowane rozwiązanie ma postać pakietu kodu języka programowania `Python` - `HyPerf`. Na pakiet składają się dwa moduły. Moduł składowania ma na celu asystowanie użytkownikowi przy przechowywaniu wyników eksperymentów obliczeniowych w uporządkowanej formie oraz przy ich udostępnianiu. W tym celu zostały wykorzystane system zarządzania relacyjną bazą danych, odpowiednio zaprojektowana, predefiniowana struktura bazy danych oraz implementacja interfejsu programistycznego, który pozwala użytkownikowi rozwiązania na łatwą interakcję z bazą danych oraz weryfikuje poprawność wykonywanych operacji. Moduł analityczny służy automatyzacji analiz transferowalności hiperparametrów na podstawie zgromadzonych wyników. Zadanie to realizowane jest poprzez implementację kilku rodzajów analiz, która pozwala na wygodne obliczanie wartości liczbowych oraz ich wizualizację. Szczegóły dotyczące specyfikacji obu modułów są zawarte w kolejnych sekcjach tego rozdziału.

3.1. Wymagania wstępne

Projektowanie i implementację `HyPerf` poprzedziła skrupulatna analiza wymagań, które powinno ono spełniać, aby praca z nim była efektywna i mogło być ono używane przez szeroko zakrojoną społeczność. Zidentyfikowane wymagania pomogły w wyznaczeniu kierunku prac oraz miały znaczący wpływ na podjęte decyzje projektowe, np. wykorzystane technologie, rodzaj przechowywanych danych, sposób interakcji z użytkownikiem, czy wybór dostępnych typów analiz. Zidentyfikowane wymagania zostały opisane poniżej.

Lista zidentyfikowanych wymagań:

- rozwiązanie powinno być użyteczne na średniej klasy komputerze osobistym,
- wdrożenie rozwiązania nie powinno wiązać się z złożoną instalacją oprogramowania bazodanowego,
- rozwiązanie powinno zawierać częściowo wypełnioną danymi przykładową bazę danych,
- praca z rozwiązaniem nie powinna wymagać użycia języka zapytań *SQL*,
- na wykonanie za pomocą rozwiązania podstawowych operacji na danych (pozyskiwanie, dodawanie, usuwanie, modyfikowanie) nie powinno składać się więcej niż kilka linii kodu,
- wykorzystywana przez rozwiązanie forma reprezentacji danych powinna być czytelna dla użytkownika,
- postać i zawartość danych przechowywanych przez rozwiązanie powinny wspierać reprodukowalność stojących za nimi eksperymentów,
- udostępnianie uzyskanej za pomocą rozwiązania bazy danych powinno być łatwe,

- oferowane przez rozwiązanie opcje analizy danych nie powinny wymagać ich transformacji w stosunku do postaci, w jakiej są one przez nie zwracane,
- zaimplementowane w rozwiązaniu algorytmy analityczne powinny być deterministyczne, zastosowane na identycznych danych powinny dawać powtarzalne rezultaty,
- dostępne w rozwiązaniu wizualizacje analiz powinny być czytelne i estetyczne.

3.2. Wybór technologii

Zgodnie z przedstawioną już informacją, HyPerf ma postać pakietu kodu języka programowania `Python`, lecz jego działanie jest możliwe dzięki licznym rozszerzeniom, dostępnym w ekosystemie tego języka. Wybór wykorzystanych technologii oraz stojąca za nim argumentacja przedstawione są w niniejszej sekcji.

Język programowania Pakiet HyPerf został zaimplementowany za pomocą języka programowania `Python` [23]. Jest to język programowania powszechnie używany w dziedzinie uczenia maszynowego oraz wykorzystywany w wielu dotyczących jej pracach badawczych. Ponadto, zaimplementowane są w nim liczne narzędzia do *AutoML*. Za wyborem tym przemawiają również zwięzła składnia, rozwinięty ekosystem rozszerzeń oraz nacisk, kładziony przez twórców języka, na czytelność oraz łatwość dystrybucji kodu źródłowego.

System zarządzania relacyjną bazą danych (*RDBMS*) U podstaw rozwiązania stoi również lekki system zarządzania relacyjną bazą danych typu *serverless* - `SQLite` [11]. Moduł integrujący technologię `SQLite` z językiem programowania `Python` jest częścią jego bogatej biblioteki standardowej, a więc nie wymaga dodatkowej instalacji. Kolejnymi atutami są wysoka wydajność przy niskiej liczbie użytkowników oraz łatwość udostępniania baz danych, które reprezentowane są przez pojedyncze pliki binarne.

Programistyczna reprezentacja danych Do reprezentacji danych w kodzie służy implementacja konceptu ramki danych o nazwie `Pandas` [18]. Udostępnia ona rozbudowane możliwości wydajnego przetwarzania danych tabelarycznych za pomocą interfejsu popartego szczegółową dokumentacją. Dużym atutem jest popularność i szeroka znajomość tej technologii wśród użytkowników języka programowania `Python`, ponieważ jej zastosowanie częściowo kształtuje sposób interakcji z opisywanym rozwiązaniem.

Szybkie obliczenia numeryczne Przy implementacji wymagających obliczeniowo części metod analitycznych wykorzystany został pakiet do obliczeń numerycznych `NumPy` [19]. Pozwala on na przeprowadzanie wydajnych obliczeń na wektorach oraz macierzach, dzięki implementacji za pomocą wysoce zoptymalizowanego kodu kompilowanego języka programowania `C`. Narzędzie to skraca również czas samego programowania, ponieważ udostępnia liczne funkcje pomocnicze, realizujące typowe operacje algebraiczne.

Wizualizacja wyników Wyniki analiz wizualizowane są w oparciu o połączenie pakietów `Matplotlib` [12] oraz bazującego na nim `Seaborn` [27]. Zastosowanie tej kombinacji pozwala na zarówno ingerencję w najmniejsze detale wizualizacji (za sprawą pakietu `Matplotlib`), jak i dostęp do predefiniowanych szablonów bardziej złożonych typów wykresów (dzięki pakietowi `Seaborn`).

Testy jednostkowe Biblioteka standardowa języka programowania `Python` zawiera moduł przeznaczony do testów jednostkowych (`unittest`), jednak przy testach rozwiązania użyte zostało bardziej rozbudowane narzędzie o nazwie `Pytest` [15]. Jego istotnymi przewagami, nad wbudowaną opcją, są bardziej informatywne komunikaty o przyczynach błędów, automatyczna detekcja zdefiniowanych testów oraz możliwość generowania estetycznych raportów.

3.3. Moduł składowania

Część pakietu `HyPerf` odpowiedzialna za składowanie wyników eksperymentów obliczeniowych oparta jest o system zarządzania relacyjną bazą danych `SQLite`, który nie wymaga uruchamiania dodatkowego procesu - w zamian tego jest on zintegrowany z interfejsem pośredniczącym między użytkownikiem, a bazą danych. Sama baza danych reprezentowana jest przez pojedynczy plik binarny, dzięki czemu jej udostępnianie jest łatwe, ponieważ wiąże się tylko z przesłaniem owego pliku. Struktura bazy danych jest predefiniowana - każda nowo utworzona za pomocą rozwiązania baza danych inicjalizowana jest szeregiem poleceń *DDL* z odpowiednio przygotowanego skryptu. Niemniej ważnym elementem jest programistyczny interfejs do bazy danych. Pozwala on na wykonywanie operacji *DQL* i *DML* bez użycia języka zapytań *SQL*, a także dba o to, aby operacje te nie naruszały integralności danych. Interakcja z użytkownikiem odbywa się poprzez kod języka programowania `Python`, a dokładniej poprzez metody obiektu obudowującego połączenie z bazą danych. Za reprezentację danych w kodzie odpowiadają tak zwane ramki danych, których implementację zapewnia pakiet `Pandas`.

3.3.1. Struktura bazy danych

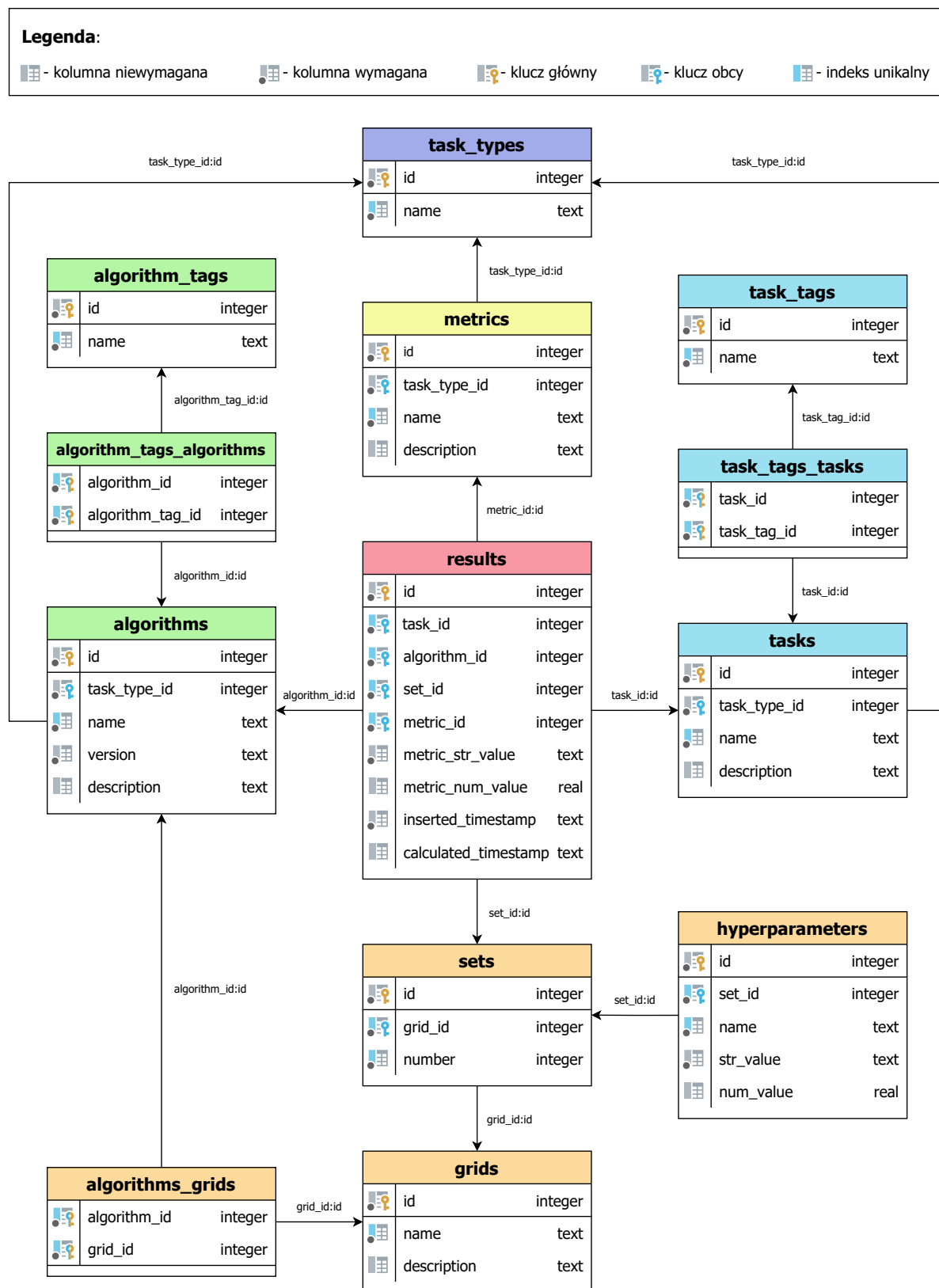
Jedną z pierwszych i jednocześnie najważniejszych decyzji projektowych była ta, dotycząca struktury bazy danych, używanej na potrzeby modułu składowania. Wynika to z faktu, że wartość i postać przechowywanych danych mają wpływ na niemal wszystkie pozostałe obszary rozwiązania - zaczynając od sposobu interakcji z użytkownikiem, a na możliwych do przeprowadzenia rodzajach analiz kończąc. Dane są fundamentem prezentowanego narzędzia, a zatem jego jakość idzie w parze z ich jakością. Ta z kolei jest w dużej mierze kształtowana przez, opisaną w niniejszej sekcji, strukturę relacji i ich wzajemne zależności.

Opracowany model relacyjny w pewnym stopniu przypomina małej skali hurtownię danych. Tabele, odpowiadające za definicje zadań predykcyjnych, algorytmów uczenia maszynowego, zestawów hiperparametrów, czy metryk jakości modeli pełnią tutaj rolę tabel wymiarów, a tabela, przechowująca wyniki eksperymentów obliczeniowych dla kombinacji wspomnianych definicji - tabeli faktów. Kolejnym podobieństwem do wspomnianego podejścia z obszaru *Business Intelligence* jest fakt, że dane mogą pochodzić z potencjalnie wielu różnych źródeł. Jednak w odróżnieniu od wielu rozwiązań określanych mianem hurtowni danych, które zorientowane są przede

wszystkim na sprawny odczyt (*DQL*), zaproponowany model charakteryzuje wysoki stopień znormalizowania, co nie dyskryminuje wydajności operacji *DML*.

Zaproponowana struktura wspiera także reprodukowalność składowanych w bazie danych wyników. Przyczyniają się do tego wymagalność kluczowych atrybutów oraz liczne klucze obce, zmuszające do jawnego określania zależności, a także przechowywanie danych w formie numerycznej, gdy ich postać na to pozwala, czy opcjonalna możliwość zapisywania opisów poszczególnych definicji. Technicznym kluczem głównym prawie każdej relacji jest całkowity indeks *id*, ale w przypadku wszystkich tabel, których pojedyncze wiersze mogą interesować użytkownika, wymuszenie unikalności atrybutu *name* pozwala na używanie go w roli klucza potencjalnego. Pozwala na uniknięcie konieczności posługiwania się przez użytkownika wartościami indeksów w trakcie pracy z interfejsem do bazy danych.

Wszystkie szczegóły zaprojektowanego modelu przedstawione są w formie graficznej na Rysunku 3.1. Ponadto, w dalszej części niniejszej podsekcji zawarta jest lista tabel i kolumn wraz z ich opisami. Zarówno w przypadku diagramu, jak i listy, kolorowe oznaczenia nazw relacji odpowiadają podziałowi na podstawie tematyki składowanych danych.



Rysunek 3.1: Diagram struktury bazy danych modułu składowania. Graficzne oznaczenia cech kolumn opisane są w legendzie, a strzałki, podpisane rozdzielonymi dwukropkiem nazwami par atrybutów, wizualizują zależności, które występują między relacjami.

task_types

Tabela wymiarów przechowująca definicje typów zadań predykcyjnych. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *name*
Wymagana kolumna tekstowa, nazwa typu zadania predykcyjnego

metrics

Tabela wymiarów przechowująca definicje metryk jakości modeli. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *task_type_id*
Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *task_types.id*, typ zadania predykcyjnego, dla którego zdefiniowana jest dana metryka jakości modeli.
- *name*
Wymagana kolumna tekstowa, nazwa metryki jakości modeli.
- *description*
Opcjonalna kolumna tekstowa, opis metryki jakości modeli.

tasks

Tabela wymiarów przechowująca definicje zadań predykcyjnych. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *task_type_id*
Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *task_types.id*, typ zdefiniowanego zadania predykcyjnego.
- *name*
Wymagana kolumna tekstowa, nazwa zadania predykcyjnego.
- *description*
Opcjonalna kolumna tekstowa, opis zadania predykcyjnego.

task_tags

Tabela wymiarów przechowująca tagi zadań predykcyjnych. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *name*
Wymagana kolumna tekstowa, nazwa tagu zadań predykcyjnych.

task_tags_tasks

Tabela łącznikowa przechowująca powiązania tagów z zadaniami predykcyjnymi. Zawiera indeks unikalny dla kolumn *task_id* i *task_tag_id* (powiązanie danej pary tag - zadanie predykcyjne nie powinno występować więcej niż raz).

- *task_id*
Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *tasks.id*, zadanie predykcyjne, z którym powiązany jest dany tag.
- *task_tag_id*
Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *task_tags.id*, tag, z którym powiązane jest dane zadanie predykcyjne.

algorithms

Tabela wymiarów przechowująca definicje algorytmów uczenia maszynowego. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *task_type_id*
Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *task_types.id*, typ zadania predykcyjnego, dla którego zdefiniowany jest dany algorytm uczenia maszynowego.
- *name*
Wymagana kolumna tekstowa, nazwa algorytmu uczenia maszynowego.
- *version*
Wymagana kolumna tekstowa, wersja algorytmu uczenia maszynowego.
- *description*
Opcjonalna kolumna tekstowa, opis algorytmu uczenia maszynowego.

algorithm_tags

Tabela wymiarów przechowująca tagi algorytmów uczenia maszynowego. Zawiera indeks unikalny dla kolumny *name*.

- *id*
Wymagana kolumna numeryczna całkowita, klucz główny.
- *name*
Wymagana kolumna tekstowa, nazwa tagu algorytmów uczenia maszynowego.

task_tags_tasks

Tabela łącznikowa przechowująca powiązania tagów z algorytmami uczenia maszynowego. Zawiera indeks unikalny dla kolumn *algorithm_id* i *algorithm_tag_id* (powiązanie danej pary tag - algorytm uczenia maszynowego nie powinno występować więcej niż raz).

- *task_id*
Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *algorithms.id*, algorytm uczenia maszynowego, z którym powiązany jest dany tag.

- *task_tag_id*

Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *algorithm_tags.id*, tag, z którym powiązany jest dany algorytm uczenia maszynowego.

grids

Tabela wymiarów przechowująca definicje siatek hiperparametrów. Zawiera indeks unikalny dla kolumny *name*.

- *id*

Wymagana kolumna numeryczna całkowita, klucz główny.

- *name*

Wymagana kolumna tekstowa, nazwa siatki hiperparametrów.

- *description*

Opcjonalna kolumna tekstowa, opis siatki hiperparametrów.

sets

Tabela wymiarów przechowująca definicje zestawów hiperparametrów. Zawiera indeks unikalny dla kolumn *grid_id* i *number* (w danej siatce hiperparametrów zestaw hiperparametrów o danym numerze porządkowym nie powinien występować więcej niż raz).

- *id*

Wymagana kolumna numeryczna całkowita, klucz główny.

- *grid_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *grids.id*, siatka hiperparametrów, do której należy dany zestaw hiperparametrów.

- *number*

Wymagana kolumna numeryczna całkowita, numer porządkowy zestawu hiperparametrów w obrębie danej siatki hiperparametrów.

hyperparameters

Tabela wymiarów przechowująca definicje hiperparametrów wraz z ich wartościami. Zawiera indeks unikalny dla kolumn *set_id* i *number* (w danym zestawie hiperparametrów hiperparametr o danej nazwie nie powinien występować więcej niż raz).

- *id*

Wymagana kolumna numeryczna całkowita, klucz główny.

- *set_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *sets.id*, zestaw hiperparametrów, do którego należy dany hiperparametr.

- *name*

Wymagana kolumna tekstowa, nazwa hiperparametru.

- *str_value*

Wymagana kolumna tekstowa, wartość hiperparametru w postaci tekstowej.

- *num_value*

Opcjonalna kolumna numeryczna rzeczywista, wartość hiperparametru w postaci numerycznej (wartości niektórych hiperparametrów mogą nie dawać przedstawić się w postaci numerycznej).

algorithms_grids

Tabela łącznikowa przechowująca powiązania algorytmów uczenia maszynowego z siatkami hiperparametrów. Zawiera indeks unikalny dla kolumn *algorithm_id* i *grid_id* (powiązanie danej pary algorytm uczenia maszynowego - siatka hiperparametrów nie powinno występować więcej niż raz).

- *algorithm_id*

Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *algorithms.id*, algorytm uczenia maszynowego, z którym powiązana jest dana siatka hiperparametrów.

- *grid_id*

Wymagana kolumna numeryczna całkowita, część klucza głównego, klucz obcy do kolumny *grids.id*, siatka hiperparametrów, z którym powiązany jest dany algorytm uczenia maszynowego.

results

Tabela faktów przechowująca wyniki, czyli wartości metryk jakości modeli obliczone dla poszczególnych kombinacji zadanie predykcyjne - algorytm uczenia maszynowego - zestaw hiperparametrów - metryka jakości modeli. Zawiera indeks unikalny dla kolumn *task_id*, *algorithm_id*, *set_id* i *metric_id* (wynik dla danej kombinacji nie powinien występować więcej niż raz).

- *id*

Wymagana kolumna numeryczna całkowita, klucz główny.

- *task_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *tasks.id*, zadanie predykcyjne, dla którego obliczony jest dany wyniki.

- *algorithm_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *algorithms.id*, algorytm uczenia maszynowego, dla którego obliczony jest dany wyniki.

- *set_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *sets.id*, zestaw hiperparametrów, dla którego obliczony jest dany wyniki.

- *metric_id*

Wymagana kolumna numeryczna całkowita, klucz obcy do kolumny *metrics.id*, metryka jakości modeli, dla której obliczony jest dany wyniki.

- *metric_str_value*

Wymagana kolumna tekstowa, wartość metryki jakości modeli w postaci tekstowej.

- *metric_num_value*

Opcjonalna kolumna numeryczna rzeczywista, wartość metryki jakości modeli w postaci

numerycznej (wartości niektórych metryk jakości modeli mogą nie dawać przedstawić się w postaci numerycznej).

- *inserted_timestamp*

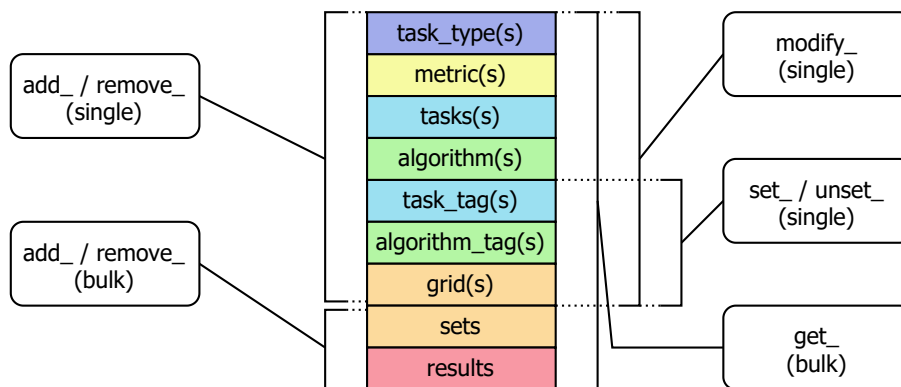
Wymagana kolumna tekstowa, czas umieszczenia wyniku w bazie danych (w technologii *SQLite* nie występuje dedykowany typ *datetime*, dlatego czas przechowywany jest w postaci tekstowej zgodnie z standardem ISO8601).

- *calculated_value*

Opcjonalna kolumna tekstowa, czas obliczenia wyniku (format tak jak w przypadku kolumny *inserted_timestamp*).

3.3.2. Interfejs programistyczny

W pakiecie *HyPerf*, pracę z bazą danych bez użycia języka zapytań *SQL* umożliwia implementacja interfejsu w języku programowania *Python*. Ma ona postać klasy *Database*, której obiekt reprezentuje połączenie z instancją bazy danych, a jego metody odpowiadają poszczególnym operacjom *DQL* i *DML*. Nazwa każdej metody składa się z jednego z sześciu przedrostków (*add* - dodaj, *remove* - usuń, *modify* - zmodyfikuj, *set* - dodaj przypisanie, *unset* - usuń przypisanie, *get* - pozyskaj), który symbolizuje rodzaj operacji, oraz określenia typu danych, których operacja dotyczy. Zbiór tych metod przedstawiony jest w sposób wizualny na Rysunku 3.2. Dostępne jest również kilka metod pomocniczych - *available_results* pozwala na sprawdzenie, dla których dozwolonych kombinacji obecnych w bazie danych definicji dostępne są wyniki, *summary* wyświetla podsumowanie wielkości bazy danych pod względem ilości wierszy w poszczególnych tabelach i rozmiaru pliku, a *close* służy zamknięciu połączenia. Szczegółowa specyfikacja wszystkich metod dostępna jest w Dodatku A.



Rysunek 3.2: Diagram metod dostępnych w programistycznym interfejsie do bazy danych, odpowiadających operacjom *DQL* i *DML*. Etykiety *single* oraz *bulk* rozróżniają grupy metod, które dotyczą pojedynczych oraz wielu wierszy.

Wywołanie żadnej z metod nie wymaga znajomości nadawanych automatycznie przez *RDBMS* indeksów, ponieważ unikalność atrybutu *name* jest gwarantowana we wszystkich zawierających go relacjach. Ułatwia to pracę z interfejsem, ponieważ literały te są prostsze do zapamiętania oraz, co ważniejsze, to użytkownik decyduje o ich wartości. Wspomniane indeksy są natomiast używane do indeksacji zwracanych przez interfejs ramek danych, które służą do reprezentacji danych w kodzie. Implementację ramek danych zapewnia pakiet *Pandas*. Wykorzystanie tej technologii pozwala na wygodne i niemal dowolne przetwarzanie danych pozyskiwanych za pomocą

interfejsu. Ramki danych o odpowiedniej strukturze służą także przekazywaniu danych do bardziej złożonych metod interfejsu, jak `add_sets`, czy `add_results`, które przetwarzają wiele wierszy jednocześnie.

Oprócz pośredniczenia między użytkownikiem, a bazą danych, istotną rolę interfejsu jest także zapewnienie, aby owa interakcja nie miała destruktywnego wpływu na integralność składowanych danych. Odbywa się to w bardziej rozbudowany sposób, niż podstawowe sprawdzenia wykonywane przez system zarządzania bazą danych (np. zapewnienie wartości dla wymaganych kolumn, zachowanie unikalności kolumn z indeksem unikalnym, czy spełnienie wymagań kluczy obcych). Mowa tutaj o następujących zabezpieczeniach:

- wszystkie definicje, dla których dodawane są wyniki, muszą odnosić się do tego samego typu zadania predykcyjnego,
- usunięcie definicji, dla której składowane są wyniki, musi wiązać się usunięciem tych wyników lub nie może zostać wykonane,
- algorytm uczenia maszynowego i siatka hiperparametrów, dla których dodawana są wyniki, muszą posiadać wzajemne przypisanie, informujące o ich kompatybilności
- usunięcie powiązania algorytmu uczenia maszynowego i siatki hiperparametrów, dla których składowane są wyniki, musi wiązać się z usunięciem tych wyników lub nie może zostać wykonane,
- modyfikacja lub usunięcie tagu definicji, dla której składowane są wyniki, wiąże się z odpowiednim ostrzeżeniem.

3.3.3. Przykład użycia

Celem lepszego przybliżenia sposobu interakcji użytkownika rozwiązania z funkcjonalnościami modułu składowania w HyPerf, w niniejszej podsekcji zawarty jest zwięzły przykład jego użycia. Demonstracja obejmuje inicjalizację nowej bazy danych, utworzenie w niej kilku definicji oraz składowanie wyników prostego eksperymentu obliczeniowego. Wyniki oparte są na zadaniu predykcyjnym nazwie *heart*, pochodzącym z przytoczonego już repozytorium OpenML, oraz na implementacji maszyny wektorów nośnych (*SVM*), pochodzącej z pakietu Scikit-learn [20]. Obecne w przykładzie błędy zostały popelnione z premedytacją, aby lepiej zaprezentować możliwości interfejsu oraz zaimplementowane w nim zabezpieczenia.

Pierwszymi krokami są zaimportowanie klasy *Database* oraz pakietu *Pandas* [18], a następnie nawiązanie połączenia z bazą danych. Przekazana do konstruktora klasy *Database* ścieżka nie prowadzi do istniejącego pliku, w wyniku czego w jego miejsce utworzona zostaje nowa baza danych, która inicjalizowana jest opisaną wcześniej strukturą.

```
from HyPerf.storage import Database
import pandas as pd

database = Database(path="./database.sqlite")
```

```
Provided file does not exist, but its directory does. Creating a new database.
```

Następnym etapem jest utworzenie w bazie danych szeregu definicji, do których odwoływać się będą wyniki eksperymentu, przeznaczone do składowania. W pierwszej kolejności zdefiniowany zostaje typ zadania predykcyjnego. Jest to szczególny typ klasyfikacji, zwany binarną (wyłącznie dwie etykiety klas). Efekt operacji dodawania danych sprawdzany jest za pomocą odpowiedniej metody typu *get*.

```
database.add_task_type(name="binary classification")
print(database.get_task_types())
```

id	name
1	binary classification

Po utworzeniu w bazie danych definicji typu zadania predykcyjnego, samo zadanie predykcyjne może zostać określone poprzez odpowiednie odwołanie. W tym przypadku jest to przeskalowana wersja zadania predykcyjnego o nazwie *heart*, o czym informują jego nazwa i opcjonalny opis. Dodatkowo, utworzony i przypisany do zdefiniowanego zadania predykcyjnego tag informuje o jego pochodzeniu.

```
database.add_task(
    task_type="binary classification",
    name="heart-scaled",
    description="all features scaled to [0, 1]"
)
database.add_task_tag(name="OpenML")
database.set_task_tag(task_tag="OpenML", task="heart-scaled")
print(database.get_tasks())
```

id	task type	name	description	tags
1	binary classification	heart-scaled	all features scaled to [0, 1]	(OpenML,)

Kontynuując proces tworzenia definicji, poniższy kod odpowiada za określenie algorytmu uczenia maszynowego. Po raz kolejny występuje tutaj odwołanie do klasyfikacji binarnej, a sam algorytm to implementacja maszyny wektorów nośnych (*SVM*), pochodząca z pakietu *Scikit-learn*. Podanie opisu, który w tym przypadku informuje o genezie implementacji, również tutaj jest opcjonalne, ale określenie wersji algorytmu jest konieczne, nad czym czuwa interfejs.

```
database.add_algorithm(
    task_type="binary classification",
    name="sklearn.svm.SVC",
    version="0.24.2",
    description="SVM implementation based on LIBSVM library"
)
```

Oznaczenie "0.24.2" odnosi się właściwie do wersji pakietu, który zawiera implementacje również innych algorytmów. Zawarcie tej informacji w definicji i dodanie jej do bazy danych pozytywnie wpłynie na reprodukowalność składowanych wyników, dlatego następne polecenie modyfikuje utworzoną już definicję algorytmu uczenia maszynowego, a dokładniej dodaje przedrostek "sklearn" do oznaczenia wersji.

```
database.modify_algorithm(name="sklearn.svm.SVC", new_version="sklearn 0.24.2")
```

Some data may depend on this algorithm. Modify with caution.

```
print(database.get_algorithms())
```

id	task type	name	version	description	tags	grids
1	binary classification	sklearn.svm.SVC	sklearn 0.24.2	SVM implementation based on LIBSVM library	NaN	NaN

Kolejnym krokiem jest zdefiniowanie siatki hiperparametrów i oznaczenie jej jako kompatybilnej z dopiero co określonym algorytmem uczenia maszynowego. Odbywa się to w podobny sposób, co zaprezentowane już utworzenie i przypisanie tagu.

```
database.add_grid(name="svm-simple", description="it is only an example")
database.set_grid(grid="svm-simple", algorithm="sklearn.svm.SVC")
print(database.get_grids())
```

id	name	description	algorithms
1	svm-simple	it is only an example	(sklearn.svm.SVC,)

Siatka hiperparametrów jest już zdefiniowana, ale nie zawiera jeszcze żadnych zestawów hiperparametrów. W tym przykładzie rozważana jest minimalistyczna siatka, składająca się z 10 zestawów, które definiują wartość wyłącznie jednego hiperparametru - "gamma". Ramka danych, zawierająca wspomnianą siatkę hiperparametrów, wczytywana jest z pliku.

```
grid = pd.read_csv("simple_svm_grid.csv")
print(grid)
```

number	gamma
1	1
2	0.1
3	0.01
4	0.001
5	0.0001
6	0.00001
7	0.000001
8	0.0000001
9	0.00000001
10	0.000000001

Dodanie zestawów hiperparametrów do siatki odbywa się poprzez przekazanie do interfejsu ramki danych o odpowiedniej strukturze. Metoda `add_sets` automatycznie wykrywa zdefiniowane hiperparametry na podstawie występujących w ramce kolumn, a także decyduje, czy ich wartości mogą być składowane w postaci numerycznej, co może w przyszłości znacząco ułatwić pracę z siatką po jej wtórnym odzyskaniu z bazy danych. Wartość parametru `expand_grid` określa, czy dozwolone jest poszerzenie zbioru hiperparametrów definiowanych w obrębie rozważanej siatki, który rozważanym przypadkiem jest w oczywisty sposób zbiorem pustym.

```
database.add_sets(df=grid, grid="svm-simple", expand_grid=True)
print(database.get_sets(grid="svm-simple"))
```

id	number	gamma	
		str value	num value
1	1	1	1
2	2	0.1	0.1
3	3	0.01	0.01
4	4	0.001	0.001
5	5	0.0001	0.0001
6	6	0.00001	0.00001
7	7	0.000001	0.000001
8	8	0.0000001	0.0000001
9	9	0.00000001	0.00000001
10	10	0.000000001	0.000000001

Ostatnią definicją, potrzebną do składowania rozważanych wyników, jest ta dotycząca metryki jakości modeli. Zgodnie z przedstawionym poniżej kodem, jest to powszechnie używana metryka *Area Under the Receiver Operating Characteristic Curve (ROC AUC)*. Z programistycznego punktu widzenia, jej dodanie do bazy danych odbywa się w sposób analogiczny, jak dla zadania predykcyjnego. Tym razem jednak popełniony został błąd w nazwie typu zadania predykcyjnego, w wyniku czego interfejs nie pozwolił na wykonanie operacji i zgłosił odpowiedni wyjątek. Integralność bazy danych nie została naruszona, a dopiero poprawne odwołanie do zadania predykcyjnego skutkuje udanym zakończeniem operacji.

```
database.add_metric(
    task_type="binary clasification",
    name="ROC AUC",
    description="receiver operating characteristic area under the curve"
)
```

```
ProgrammingError: There is no task type named 'binary clasification' in the
database.
```

```
database.add_metric(
    task_type="binary classification",
    name="ROC AUC",
    description="receiver operating characteristic area under the curve"
)
print(database.get_metrics())
```

id	task type	name	description
1	binary classification	ROC AUC	receiver operating characteristic area under the curve

Po poprawnym utworzeniu w bazie danych wszystkich potrzebnych definicji, możliwe jest umieszczenie w niej wyników eksperymentu obliczeniowego. Podobnie, jak w przypadku siatki hiperparametrów, wyniki te wczytywane są z wcześniej przygotowanego pliku.

```
results = pd.read_csv("results.csv")
print(results)
```


number	ROC AUC
1	0.550
2	0.633
3	0.679
4	0.746
5	0.751
6	0.748
7	0.739
8	0.736
9	0.732
10	0.726

Umieszczenie wyników w bazie danych wiąże się z przekazaniem do metody `add_results` ramki danych o odpowiedniej strukturze oraz odwołaniu do stworzonych uprzednio definicji. Definicje metryk jakości modeli dopasowywane są na podstawie nazw kolumn wspomnianej ramki danych, dzięki czemu możliwe jest dodanie wyników dla kilku metryk jednocześnie. Również w tym przypadku interfejs decyduje, czy poszczególne wartości mogą być reprezentowane w postaci numerycznej.

```
database.add_results(
    df=results,
    task="heart-scaled",
    algorithm="sklearn.svm.SVC",
    grid="svm-simple"
)
```

Poniżej przedstawione są dwa sposoby sprawdzenia, czy dodanie wyników zakończyło się sukcesem. W pierwszym przypadku używana jest metoda, która informuje o liczbie składowanych rezultatów dla wszystkich dozwolonych kombinacji definicji. Drugi wariant to po prostu pobranie wyników z bazy danych. Podana przy wywołaniu metody `get_results` wartość argumentu `timestamps` sprawia, że zwracane są również znaczniki czasowe składowania i obliczenia rezultatów (pierwsze uzupełniane są automatycznie, a drugie nie zostały w tym przykładzie podane przez użytkownika).

```
print(database.available_results())
```

task type	task	algorithm	grid	metric	available results	possible results
binary classification	heart-scaled	sklearn.svm.SVC	svm-simple	ROC AUC	10	10

```
print(database.get_results(
    task="heart-scaled",
    algorithm="sklearn.svm.SVC",
    grid="svm-simple",
    metric="ROC AUC",
    timestamps=True
))
```

id	task	algorithm	grid	number	metric	str value	num value	inserted	calculated
1	heart-scaled	sklearn.svm.SVC	svm-simple	1	ROC AUC	0.550	0.550	2021-01-01 21:03:18	NaT
2	heart-scaled	sklearn.svm.SVC	svm-simple	2	ROC AUC	0.633	0.633	2021-01-01 21:03:18	NaT
3	heart-scaled	sklearn.svm.SVC	svm-simple	3	ROC AUC	0.679	0.679	2021-01-01 21:03:18	NaT
4	heart-scaled	sklearn.svm.SVC	svm-simple	4	ROC AUC	0.746	0.746	2021-01-01 21:03:18	NaT
5	heart-scaled	sklearn.svm.SVC	svm-simple	5	ROC AUC	0.751	0.751	2021-01-01 21:03:18	NaT
6	heart-scaled	sklearn.svm.SVC	svm-simple	6	ROC AUC	0.748	0.748	2021-01-01 21:03:18	NaT
7	heart-scaled	sklearn.svm.SVC	svm-simple	7	ROC AUC	0.739	0.739	2021-01-01 21:03:18	NaT
8	heart-scaled	sklearn.svm.SVC	svm-simple	8	ROC AUC	0.736	0.736	2021-01-01 21:03:18	NaT
9	heart-scaled	sklearn.svm.SVC	svm-simple	9	ROC AUC	0.732	0.732	2021-01-01 21:03:18	NaT
10	heart-scaled	sklearn.svm.SVC	svm-simple	10	ROC AUC	0.726	0.726	2021-01-01 21:03:18	NaT

Dobrym zwyczajem jest jawne zamknięcie połączenia z bazą danych po zakończonej pracy. Odbyna się to za pomocą nazwanej zgodnie z powszechną konwencją metody *close*.

```
database.close()
```

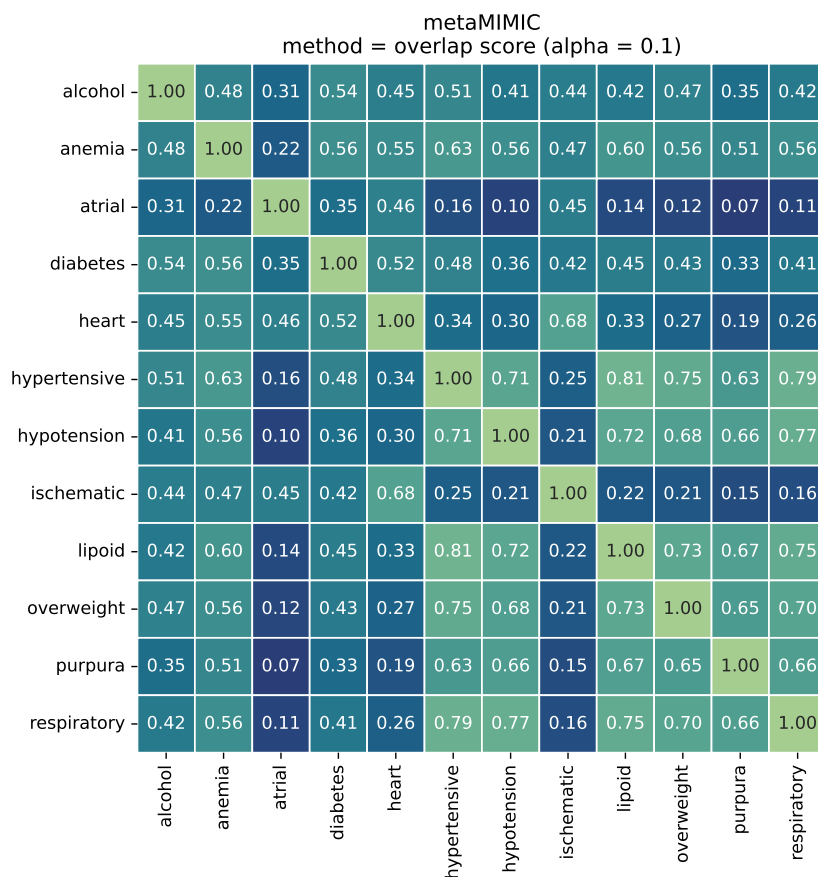
3.4. Moduł analityczny

Drugi komponent pakietu HyPerf odpowiedzialny jest za automatyzację metod analizy transferowalności hiperparametrów, które były najczęściej stosowane w dotychczasowej literaturze. Zrealizowana za pomocą języka programowania Python implementacja współpracuje z formatem danych, przyjętym na potrzeby interfejsu modułu składowania, sprawdza poprawność przekazanych danych pod kątem wymagań rozważanych analiz oraz pozwala nie tylko na łatwe obliczanie, ale również wizualizację uzyskanych wyników. Dostępne są trzy rodzaje analiz, które dodatkowo uwzględniają kilka strategii wykonania. Wydajność metod wspiera implementacja z użyciem pakietu do obliczeń numerycznych Numpy, a estetyczne wizualizacje zapewnia połączenie pakietów Matplotlib i Seaborn.

3.4.1. Rodzaje analiz

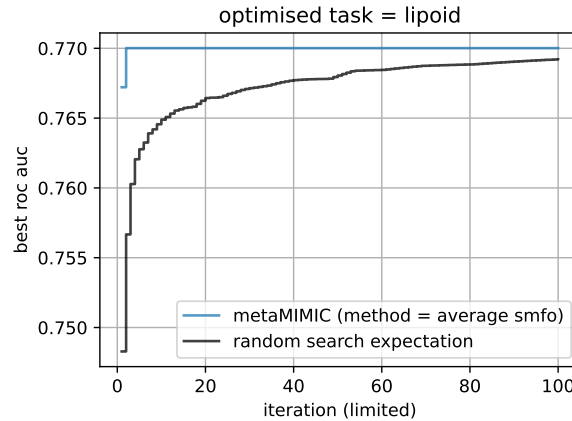
Podobieństwo rankingów zestawów hiperparametrów Pierwszy dostępny rodzaj analizy to porównanie, między parami zadań predykcyjnych, podobieństwa rankingów zestawów hiperparametrów, należących do tej samej siatki. Polega to na uszeregowaniu zestawów hiperparametrów (dla wszystkich rozważanych zadań predykcyjnych niezależnie) na podstawie wcześniej obliczonych wartości wybranej metryki jakości modeli, a następnie porównaniu wszystkich par rankingów za pomocą jednej z czterech miar podobieństwa (*Percentage of Overlap* - Definicja 2.13, *Correspondence at the Top* - Definicja 2.14, *Overlap Score* - Definicja 2.15 lub *Canberra Distance* - Definicja 2.16). Wartości wszystkich miar należą do przedziału $[0, 1]$, dzięki czemu można je ze sobą zestawiać, oraz wartości większe świadczą o większym podobieństwie rankingów. Podejście to pozwala na ocenę podobieństwa rozkładów metryk jakości dla zarówno pojedynczych

zadań predykcyjnych, jak i całych ich grup, potencjalnie pochodzących z różnych eksperymentów. W implementacji, za ten rodzaj analizy odpowiada klasa *RankingSimilarityAnalyser*, a jej wyniki są wizualizowane za pomocą mapy ciepła, czego przykład widoczny jest na Rysunku 3.3. Przedstawiona macierz jest symetryczna, ponieważ rozważane są wyniki pochodzące z wyłącznie jednego eksperymentu, ale nie zawsze musi tak być.



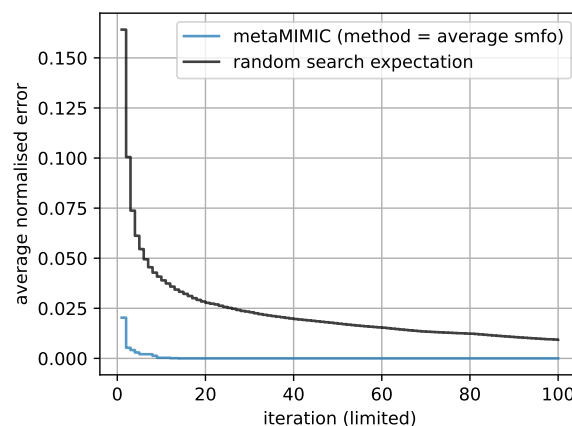
Rysunek 3.3: Przykład wizualizacji analizy podobieństwa rankingów zestawów hiperparametrów dla wyników pochodzących z eksperymentu *metaMIMIC*. Zastosowaną miarą podobieństwa rankingów jest *Overlap Score* z wartością parametru $\alpha = 0.1$.

Szybkość transferu hiperparametrów Drugi dostępny rodzaj analizy to określenie szybkości transferu hiperparametrów pomiędzy pewnym zbiorem zadań predykcyjnych, a wybranym pojedynczym zadaniem. Polega to na skonstruowaniu portfolio zestawów hiperparametrów (Definicja 2.12), za pomocą jednej z dwóch metod (*Simple* - Definicja 2.19 lub *Average SMFO* - Definicja 2.21), a następnie wyznaczeniu najlepszej wartości metryki jakości modeli dla rozważanego zadania predykcyjnego, po sprawdzeniu danej liczby konfiguracji, w określonej wcześniej kolejności. Możliwe jest również rozważanie wartości oczekiwanej losowego przeglądania zestawów hiperparametrów, jako punktu odniesienia. W implementacji, za ten rodzaj analizy odpowiada klasa *TransferSpeedAnalyser*, a jej wyniki są wizualizowane za pomocą wykresu liniowego, czego przykład widoczny jest na Rysunku 3.4.



Rysunek 3.4: Przykład wizualizacji analizy szybkości transferu hiperparametrów metodą *Average SMFO* pomiędzy zadaniem predykcyjnym o skrótowej nazwie *lipoid*, a innymi zadaniami predykcyjnymi z eksperymentu *metaMIMIC*. Czarna linia reprezentuje wartość oczekiwaną dla losowego przeglądania konfiguracji.

Średni znormalizowany błąd strojenia Trzeci dostępny rodzaj analizy to wyznaczenie zmienności miary *Average Normalized Error*, która została szczegółowo opisana w Definicji 2.17. Ta metoda również opiera się na symulacji procesu transferu hiperparametrów oraz udostępnia te same metody wyznaczania portfolio zestawów hiperparametrów, ale, w przeciwieństwie do opisanej w poprzednim akapicie analizy, średni znormalizowany błąd strojenia jest zagregowaną metodą oceny transferowalności hiperparametrów, ponieważ brany jest pod uwagę transfer zestawów hiperparametrów dla wielu zadań predykcyjnych równocześnie. Dzięki temu możliwe jest sprawdzenie, jak dana baza doświadczeń sprawdza się dla całego zbioru zadań predykcyjnych, a nie wyłącznie pojedynczego problemu. Tutaj również możliwe jest rozważanie wartości oczekiwanej losowego przeglądania zestawów hiperparametrów, jako punktu odniesienia. W implementacji, za ten rodzaj analizy odpowiada klasa *AverageNormalizedErrorAnalyser*, a jej wyniki są wizualizowane za pomocą wykresu liniowego, czego przykład widoczny jest na Rysunku 3.5.



Rysunek 3.5: Przykład wizualizacji analizy średniego znormalizowanego błędu strojenia metodą *Average SMFO* dla wyników pochodzących z eksperymentu *metaMIMIC*. Czarna linia reprezentuje wartość oczekiwaną dla losowego przeglądania konfiguracji.

3.4.2. Implementacja

Praca z modułem analitycznym pakietu HyPerf, podobnie jak w przypadku modułu składowania, odbywa się poprzez kod języka programowania Python. W ramach rozwiązania przygotowane są trzy klasy (*RankingSimilarityAnalyser*, *TransferSpeedAnalyser* oraz *AverageNormalizedErrorAnalyser*), z których każda odpowiada za jeden, z wymienionych w poprzedniej podsekcji, rodzaj analizy. Implementacja wszystkich klas oparta jest na bazowej klasie *Analyser*, dzięki czemu możliwe jest łatwe dodanie implementacji kolejnych typów analiz w przyszłości.

Celem wykonania analizy, obiekt odpowiedniej klasy (analizator) musi zostać zainicjalizowany danymi. Rozważane wyniki są przekazywane do jego konstruktora w postaci dwóch ramek danych *Pandas* o strukturze zgodnej z tą, która używana jest do zwracania wyników w interfejsie modułu składowania. Wspomniane ramki danych oznaczane są jako lewa i prawa. W przypadku klasy *RankingSimilarityAnalyser* rozróżnienie między nimi nie ma znaczenia, natomiast, w przypadku klas symulujących proces transferu hiperparametrów (*TransferSpeedAnalyser* oraz *AverageNormalizedErrorAnalyser*), lewa ramka danych pełni rolę ramki docelowej, a prawa źródłowej.

Operacja inicjalizacji analizatora wynikami uwzględnia szereg sprawdzeń poprawności przekazanych danych. Dotyczą one zarówno ich struktury, jak i zawartości. Każdy rodzaj analizy wymaga wyników obliczonych dla tego samego algorytmu (z dokładnością do wersji, w przypadku rozbieżności uwzględnione jest tutaj specjalne ostrzeżenie), siatki hiperparametrów oraz metryki jakości modeli. Niemal zawsze możliwe jest jednoczesne przekazanie wyników obliczonych dla wielu zadań predykcyjnych. Jedynie w przypadku analizatora szybkości transferu hiperparametrów konieczne jest, aby docelowa ramka zawierała wyniki, dotyczące wyłącznie jednego zadania predykcyjnego. Dodatkowo, analizatory symulujące proces transferu hiperparametrów ignorują wyniki w źródłowej ramce danych, które dotyczą zadań predykcyjnych występujących w docelowej ramce. Ma to na celu uniknięcie wycieku informacji.

Oprócz ramek danych, każda klasa definiuje również trzy atrybuty, których wartości są muszą zostać określone przez użytkownika:

- atrybut *ascending* określa monotoniczność rozważanej metryki jakości modeli,
- atrybut *method* decyduje o wyborze strategii analitycznej (miara podobieństwa rankingów lub metoda konstrukcji portfolio),
- atrybut *label* stanowi etykietę instancji analizatora, która używana jest głównie przy wizualizacji wyników jego analizy.

Interakcja z poprawnie zainicjalizowanym analizatorem odbywa się poprzez użycie jednej z dwóch metod:

- metoda *calculate* pozwala na przeprowadzenie obliczeń i uzyskanie wyników w postaci ramki danych *Pandas*, dzięki czemu możliwe jest ich dalsze przetwarzanie lub zapis do pliku,
- metoda *plot* służy do przygotowania estetycznej wizualizacji wyników analiz z użyciem pakietów *Matplotlib* i *Seaborn*.

Dodatkowo, w przypadku klas *TransferSpeedAnalyser* oraz *AverageNormalizedErrorAnalyser* możliwa jest jednoczesna wizualizacja kilku analizatorów na jednym wykresie. Również tutaj zostały przewidziane zabezpieczenia, których celem jest niedopuszczenie do wspólnej wizualizacji niekompatybilnych analizatorów.

Szczegółowa specyfikacja wszystkich metod dostępna jest w Dodatku B.

3.4.3. Przykład użycia

W niniejszej podsekcji znajduje się krótka demonstracja sposobu pracy z modułem analitycznym. Przedstawiony scenariusz uwzględnia połączenie z przykładową bazą danych, pobranie z niej wyników eksperymentów *metaMIMIC* i *MementoML*, które dotyczą algorytmu *XGBoost* [6] i tej samej siatki hiperparametrów, oraz wykonanie dla nich analiz szybkości transferu konfiguracji. Obecne w przykładzie błędy zostały popełnione celowo, aby lepiej przedstawić możliwości rozwiązania oraz zaimplementowane w nim zabezpieczenia.

W pierwszej kolejności, wszystkie potrzebne moduły są importowane i nawiązywane jest połączenie z przykładową bazą danych. Komunikat informuje użytkownika, że przekazana do konstruktora klasy *Database* ścieżka istnieje, a kryjąca się za nią baza danych powinna być utworzona i modyfikowana wyłącznie przez opisywane narzędzie.

```
from HyPerf.storage import Database
from HyPerf.analysis import TransferSpeedAnalyser
import pandas as pd

database = Database(path="./database_example/database.sqlite")
```

```
Provided file exists. Connecting to an existing database. Make sure it has been
created and altered with this tool only.
```

Następnie, z bazy danych pozyskiwana jest lista tagów, zawierająca również nazwy zadań predykcyjnych, do których są one przypisane. W tym przypadku są to dwa tagi, odpowiadające wspomnianym eksperymentom *metaMIMIC* oraz *MementoML*.

```
task_tags = database.get_task_tags()
print(task_tags)
```

id	name	tasks
1	metamimic	(alcohol, anemia, atrial, diabetes, heart, ...)
2	mementoml	(37-diabetes, 44-spambase, 1043-ada, 1046-mozilla, ...)

Wyniki są pobierane z bazy danych poprzez nieskomplikowane pętle po pozyskanych nazwach zadań predykcyjnych. Rozważaną w tym scenariuszu metryką jakości modeli jest *Area Under the Receiver Operating Characteristic Curve (ROC AUC)*. Warto zaznaczyć, że wyniki eksperymentu *MementoML* opierają się na starszej wersji algorytmu *XGBoost*. Komunikaty informują, że zostały pozyskane kolejno 12 i 22 ramki danych. Za pomocą funkcji *concat* z pakietu *Pandas* ramki danych, odpowiadające poszczególnym eksperymentom, są łączone w pionie.

```
metamimic_dfs = []
for task in task_tags[task_tags["name"]=="metamimic"]["tasks"].values[0]:
    df = database.get_results(
        task=task,
        algorithm="xgboost-1.3.3",
        grid="mementoml",
        metric="roc_auc"
    )
    metamimic_dfs.append(df)
print("Retrived dataframes: " + len(metamimic_dfs))
metamimic_df=pd.concat(metamimic_dfs)
```

```
Retrived dataframes: 12
```

```
mementoml_dfs = []
for task in task_tags[task_tags["name"]=="mementoml"]["tasks"].values[0]:
    df = database.get_results(
        task=task,
        algorithm="xgboost-0.90.0.2",
        grid="mementoml",
        metric="roc auc"
    )
    mementoml_dfs.append(df)
print("Retrieved dataframes: " + len(mementoml_dfs))
mementoml_df = pd.concat(mementoml_dfs)
```

```
Retrieved dataframes: 22
```

Połączenie z bazą danych zostaje zamknięte, ponieważ nie jest już potrzebne.

```
database.close()
```

Kolejnym krokiem jest inicjalizacja danymi obiektu klasy *TransferSpeedAnalyser*. W pierwszym przypadku został popełniony błąd - docelowa ramka danych zawiera wyniki dla więcej niż jednego zadania predykcyjnego, dlatego zgłaszany jest adekwatny wyjątek. W drugim przypadku docelowa ramka danych zawiera wyniki dla wyłącznie jednego zadania predykcyjnego o skrótowej nazwie *hypotension* i operacja kończy się sukcesem. Wartość `False` argumentu *ascending* oznacza, że im większa wartość rozważanej metryki jakości modeli, tym lepszy model, a wartość `simple` argumentu *method* odpowiada za wybór metody konstrukcji portfolio zestawów hiperparametrów o tej samej nazwie.

```
analyser_1 = TransferSpeedAnalyser(
    left=metamimic_df,
    right=metamimic_df,
    ascending=False,
    method="simple",
    label="metaMIMIC"
)
```

```
ValueError: 'left' contains results for more than one task ('alcohol', 'anemia',
'atrial', 'diabetes', 'heart', ...).
```

```
analyser_1 = TransferSpeedAnalyser(
    left=metamimic_df[metamimic_df["task"]=="hypotension"],
    right=metamimic_df,
    ascending=False,
    method="simple",
    label="metaMIMIC"
)
```

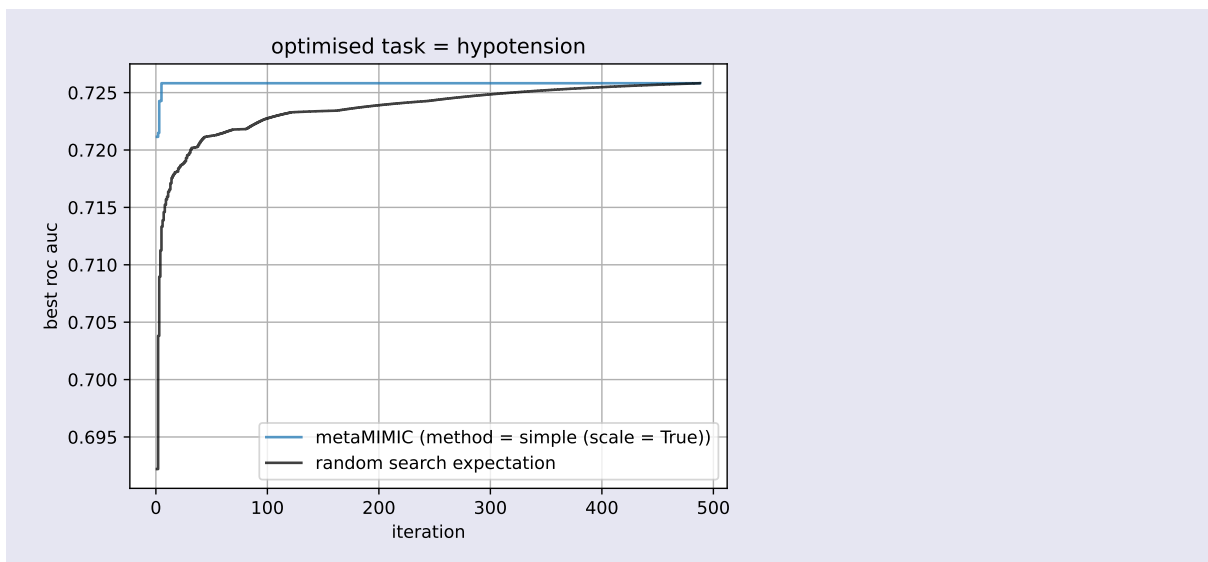
Po udanym utworzeniu instancji analizatora możliwe jest przeprowadzanie za jego pomocą analiz. Pierwszą, z dwóch dostępnych opcji, jest wywołanie metody *calculate*, która służy do pozyskiwania wyników rozważanej analizy w postaci ramki danych. Argumenty metody stanowią parametry rozważanej strategii analitycznej - w tym przypadku jest to parametr *Scale* metody konstrukcji portfolio *Simple*. Zwrócona ramka danych zawiera informacje o numerze iteracji procesu optymalizacji, sprawdzanym na jego etapie zestawie hiperparametrów, a także aktualnej oraz dotychczasowo najlepszej wartości metryki jakości modeli.

```
analyser_1.calculate(scale=True)
```

iteration	set_number	value	best_value
1	425	0.721146	0.721146
2	469	0.721493	0.721493
3	325	0.724270	0.724270
...
488	118	0.606467	0.725818

Wyniki analizy mogą również zostać od razu zwizualizowane. Służy do tego metoda *plot*, która także przyjmuje parametry strategii analitycznej jako argumenty. Dodatkowo, wartość argumentu *random_expectation* decyduje, czy na wykresie narysowana zostanie linia, odpowiadająca wartości oczekiwanej dla losowego przeglądania zestawów hiperparametrów. Etykieta analizatora (*label*) posłużyła do oznaczenia go na wykresie, a tytuł wykresu informuje o rozważanym zadaniu predykcyjnym.

```
analyser_1.plot(scale=True, random_expectation=True)
```



Dalsza część przedstawionego scenariusza ma na celu demonstrację możliwości wizualizacji wielu analizatorów na jednym wykresie, a także związanych z tą operacją zabezpieczeń. Tworzona jest druga instancja klasy *TransferSpeedAnalyser*, tym razem uwzględniająca transfer konfiguracji między zadaniami predykcyjnymi pochodzącymi z eksperymentu *MementoML*, a zadaniem z eksperymentu *metaMIMIC* o skrótowej nazwie *hypertensive*. Obiekt zostaje utworzony, ale odpowiedni komunikat ostrzega użytkownika, że zawarte w lewej i prawej ramce danych wyniki określone są dla różnych wersji algorytmu uczenia maszynowego. Fakt ten może, ale nie musi (tak, jak w rozważanym przypadku) mieć wpływ na jakość analiz.

```
analyser_2 = TransferSpeedAnalyser(
    left=metamimic_df[metamimic_df["task"]=="hypertensive"],
    right=mementoml_df,
    ascending=False,
    method="simple",
    label="MementoML"
)
```

'left' and 'right' contain results for different algorithms ({'xgboost-1.3.3'} vs {'xgboost-0.90.0.2'}). Make sure this is intended.

Po raz kolejny wywoływana jest metoda `plot` obiektu `analyser_1`. Tym razem jednak, drugi analizator zostaje przekazany do wspólnej wizualizacji za pomocą argumentu `analysers`. Zgłaszany jest wyjątek, ponieważ rozważane analizatory nie symulują optymalizacji tego samego zadania predykcyjnego (*hypotension* i *hypertensive*).

```
analyser_1.plot(scale=True, analysers={analyser_2: {"scale": True}},
               random_expectation=True)
```

Analysers 'MementoML' does not optimize the same task ({'hypotension'} vs {'hypertensive'}).

Obiekt `analyser_2` jest tworzony ponownie, tym razem dla zadania predykcyjnego zgodnego z pierwszym analizatorem (*hypotension*). Przyczyna wywołania komunikatu jest taka sama, jak przy poprzedniej inicjalizacji.

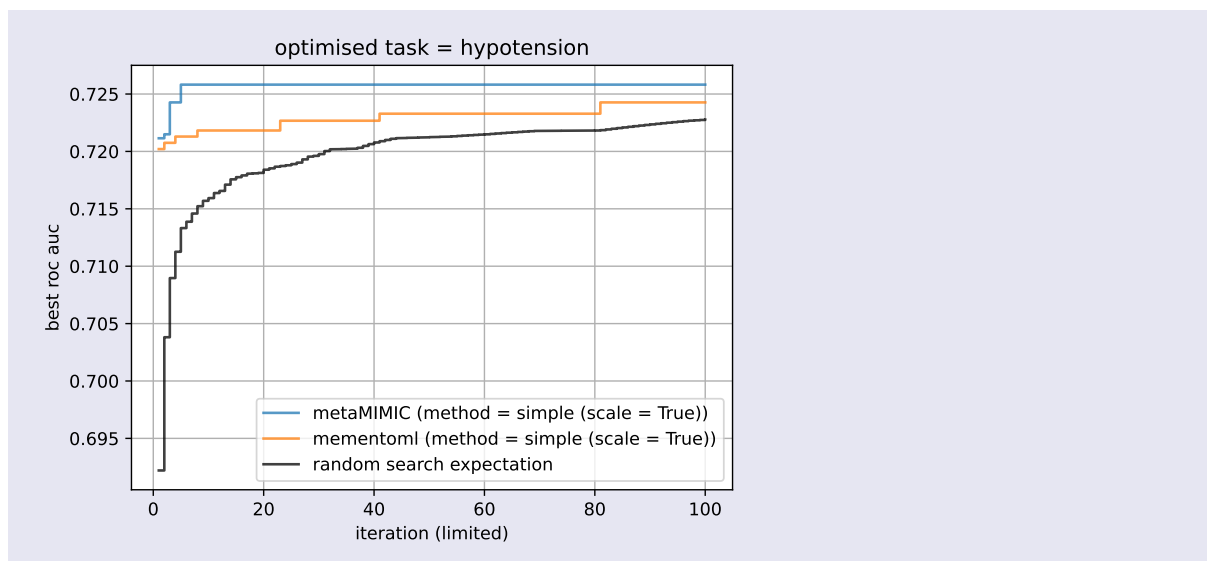
```
analyser_2 = TransferSpeedAnalyser(
    left=metamimic_df[metamimic_df["task"]=="hypotension"],
    right=mementoml_df,
    ascending=False,
    method="simple",
    label="MementoML"
)
```

'left' and 'right' contain results for different algorithms ({'xgboost-1.3.3'} vs {'xgboost-0.90.0.2'}). Make sure this is intended.

Rozważane analizatory symulują optymalizację tego samego zadania predykcyjnego (choć na podstawie innych baz doświadczeń), a zatem mogą zostać przedstawione na wspólnym wykresie. Odbyna się to w ten sam, opisany już wcześniej sposób. Dodatkowo, za pomocą argumentu `iteration_limit` określona zostaje maksymalna liczba iteracji do uwzględnienia. Ma to wpływ nie tylko na czytelność wykresu, ale również na czas samych obliczeń. Komunikat ostrzega, że tym razem poszczególne analizatory rozważają różne wersje algorytmu uczenia maszynowego. W tym przypadku nie stanowi to problemu, ale decyzja ta musi zostać podjęta przez użytkownika.

```
analyser_1.plot(scale=True, analysers={analyser_2: {"scale": True}},
               random_expectation=True, iteration_limit=100)
```

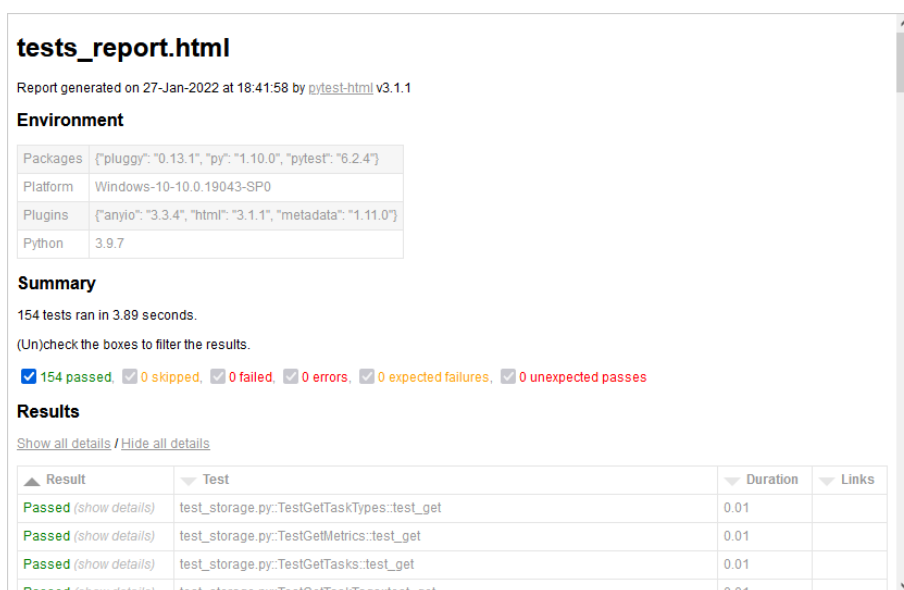
Analysers 'MementoML' is not defined for the same algorithms ({'xgboost-1.3.3'} + {'xgboost-1.3.3'} vs {'xgboost-1.3.3'} + {'xgboost-0.90.0.2'}). Make sure this is intended.



4. Testy rozwiązania

Pakiet HyPerf został przetestowane pod kątem poprawności działania i wygody użycia za pomocą dwóch rodzajów testów. Dla implementacji modułu składowania przygotowane są automatyczne testy jednostkowe, które obejmują sprawdzenia prawidłowości wykonywanych na bazie danych operacji oraz skuteczności przewidzianych zabezpieczeń. Ponadto, przeprowadzone zostały dwa scenariusze testów manualnych. W przypadku modułu składowania jest to stworzenie i wypełnienie wynikami przykładowej bazy danych (która zgodnie z wymaganiami wstępnymi stanowi część rozwiązania) wyłącznie przy użyciu przygotowanego narzędzia. Dla modułu analitycznego testy manualne przyjęły postać wykonania analizy demonstracyjnej, opartej o wyniki zawarte we wspomnianej przykładowej bazie danych. Wszystkie testy zakończyły się powodzeniem, co potwierdza użyteczność przygotowanego rozwiązania.

4.1. Testy jednostkowe




Rysunek 4.1: Zrzut ekranu przedstawiający raport z wykonania automatycznych testów jednostkowych dla modułu składowania.

Moduł składowania został przetestowany za pomocą automatycznych testów jednostkowych. Przy użyciu technologii PyTest przygotowane zostały 154 testy jednostkowe, podzielone na 40 grup. Testy obejmują po kilka scenariuszy dla niemal wszystkich metod dostępnych w implementacji, testując zarówno poprawne, jak i niepoprawne warunki ich wywoływania. W drugim przypadku mowa o sytuacjach tego samego typu, co popełnione z premedytacją i opisane w podsekcjach 3.3.3 i 3.4.3 błędy. Zgodnie z zalecaną praktyką, przykładowe dane, które są używane

przez niektóre z testów, zostały opracowane przed ich implementacją. Ponadto, kolejność testów zakłada testowanie najbardziej niezależnych funkcjonalności jako pierwszych. Dzięki rozszerzeniu `PyTest-html` możliwe jest uzyskanie estetycznego raportu z wykonania testów, czego przykład widoczny jest na Rysunku 4.1. Przeprowadzenie wszystkich testów zajmuje około czterech sekund oraz wszystkie testy kończą się sukcesem.

4.2. Przykładowa baza danych



```
database.summary()
This database contains:
* 4 metrics
* 36 tasks
* 2 algorithms
* 1 grid
* 488 sets
* 20496 results
Its file size is 0.19 MB.
```

Rysunek 4.2: Zrzut ekranu przedstawiający wygenerowane za pomocą narzędzia podsumowanie przykładowej bazy danych.

Jednym z zidentyfikowanych wymagań wstępnych, postawionych przed HyPerf (Sekcja 3.1), jest zawarcie przykładowej bazy danych, uzupełnionej wynikami, pochodzącymi z kilku eksperymentów obliczeniowych. Spełnienie tego wymagania zostało potraktowane jako dodatkowa okazja do przetestowania rozwiązania w sposób manualny. Przykładowa baza danych została utworzona i wypełniona wynikami dla zadań predykcyjnych, pochodzących głównie z dziedziny medycyny. Zasilenie bazy wynikami odbyło się wyłącznie poprzez użycie funkcjonalności udostępnianych przez rozwiązanie. Proces ten dobrze odzwierciedla typową pracę użytkownika z narzędziem i nie wykazał problemów w trakcie jego wykonywania.

Przykładowa baza danych została uzupełniona wynikami, pochodzącymi z kilku źródeł:

- eksperyment *metaMIMIC* [29], współtworzony przeze mnie w trakcie stażu wakacyjnego pod okiem promotora i opiekuna naukowego pracy,
- eksperyment *MementoML* [16], który został zrealizowany w grupie badawczej *MI2* pod kierownictwem opiekuna naukowego pracy,
- obliczone przeze mnie wyniki dla zadania predykcji śmiertelności nowotworów płuc, do którego dane otrzymałem od promotora pracy (*lung cancer*),
- obliczone przeze mnie wyniki dla zadania kwalifikacji do przeszczepu wątroby, do którego pochodzą od organizacji *United Network for Organ Sharing (UNOS)*¹ (*liver transplant*).

Wszystkie wyniki dotyczą popularnego algorytmu uczenia maszynowego *XGBoost* [6] oraz zdefiniowanej w [16] siatki hiperparametrów. Łącznie, przykładowa baza danych zawiera 20496 indywidualne wyniki dla 36 zadań predykcyjnych. Wygenerowane dla niej przez opisywane narzędzie podsumowanie widoczne jest na Rysunku 4.2.

¹<https://unos.org>

4.3. Analiza demonstracyjna

W celu przetestowania ergonomii pracy z modulem analitycznym *HyPerf*, przeprowadzona została analiza demonstracyjna, która bazuje na wynikach pochodzących z przykładowej bazy danych. Scenariusz ten dobrze odwzorowuje typowe zastosowanie narzędzia oraz stanowi przykład użyteczności zaimplementowanych metod. Praca nad nim dowiodła słuszności wyboru technologii *Pandas* do celów reprezentacji danych w kodzie, ponieważ dzięki temu wykonanie prostej, ale niestandardowej operacji na uzyskanych wynikach okazało się niemal natychmiastowe.

Poniżej przedstawione są najważniejsze części przygotowanego scenariusza. Wykonanie analiz poprzedziło pobranie z przykładowej bazy danych wyników, pochodzących z eksperymentów *metaMIMIC* i *MementoML* oraz obliczonych dla zadań dotyczących predykcji śmiertelności nowotworów płuc (*lungs cancer*) oraz kwalifikacji do przeszczepu wątroby (*liver transplant*). Odbyło się to w sposób analogiczny do przedstawionego w przykładzie użycia modułu analitycznego, który znajduje się w Podsekcji 3.4.3.

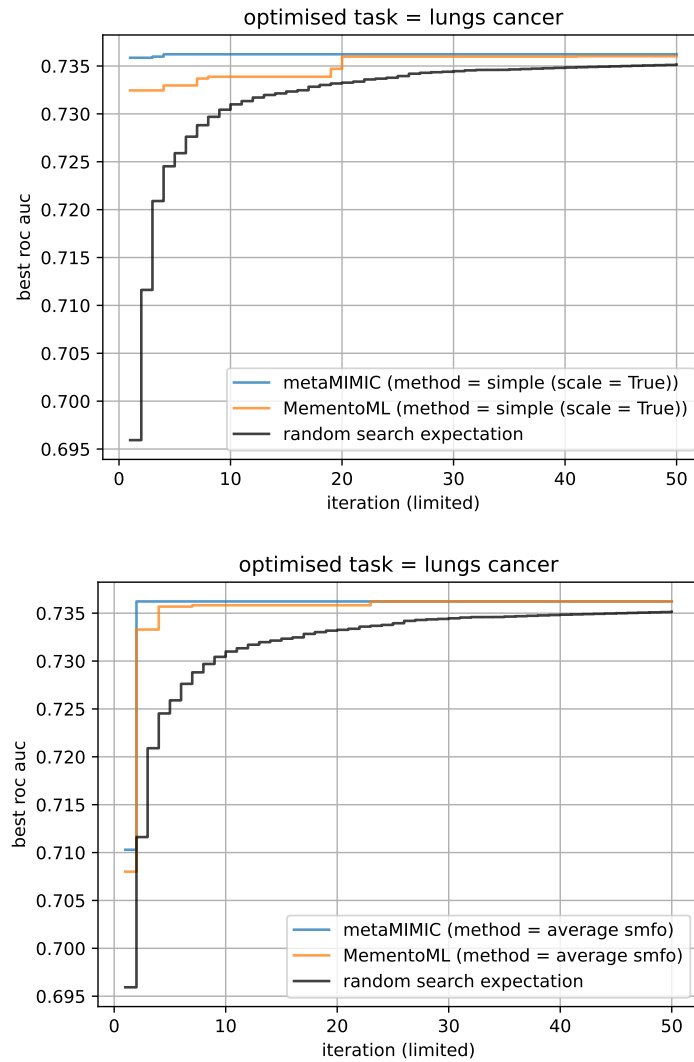
Następnie, wykonane zostały dwie analizy podobieństwa rankingów zestawów hiperparametrów (Rysunek 4.3). Pierwsza analiza porównuje zadanie *lungs cancer* (lewy panel) do zadań pochodzących z eksperymentu *metaMIMIC*, a druga do zadań pochodzących z eksperymentu *MementoML* (prawy panel). Zastosowaną miarą podobieństwa rankingów jest *Overlap Score* z domyślną wartością parametru $\alpha = 0.1$. Metoda *calculate* zwraca macierz wartości miary podobieństwa rankingów dla każdej pary problemów predykcyjnych, znajdujących się w przekazanych ramkach danych. Aby ułatwić porównanie wyników analiz, poszczególne wartości rozważanej miary zostały w obu przypadkach uśrednione. Dzięki przemyślanej implementacji operacja ta wymagała jedynie zastosowania metody *mean* na ramkach danych zwracanych przez metodę *calculate*.

<pre> analyser = RankingSimilarityAnalyser(left=metamimic_df, right=lungs_df, ascending=False, method="os", label="Lungs cancer vs metaMIMIC") print(analyser.calculate(alpha=0.1).mean()) lungs cancer 0.297136 dtype: float64 </pre>	<pre> analyser = RankingSimilarityAnalyser(left=mementoml_df, right=lungs_df, ascending=False, method="os", label="Lungs cancer vs mementoML") print(analyser.calculate(alpha=0.1).mean()) lungs cancer 0.164681 dtype: float64 </pre>
---	---

Rysunek 4.3: Zrzut ekranu przedstawiający analizy podobieństwa rankingów zestawów hiperparametrów dla zadania predykcyjnego nazwie *lungs cancer* oraz zadań predykcyjnych pochodzących z eksperymentu *metaMIMIC* (lewy panel) i eksperymentu *MementoML* (prawy panel). W przypadku eksperymentu *metaMIMIC* średnia z 12 wartości miary podobieństwa rankingów wyniosła około 0.30, a w przypadku eksperymentu *MementoML* średnia z 22 wartości wyniosła około 0.16.

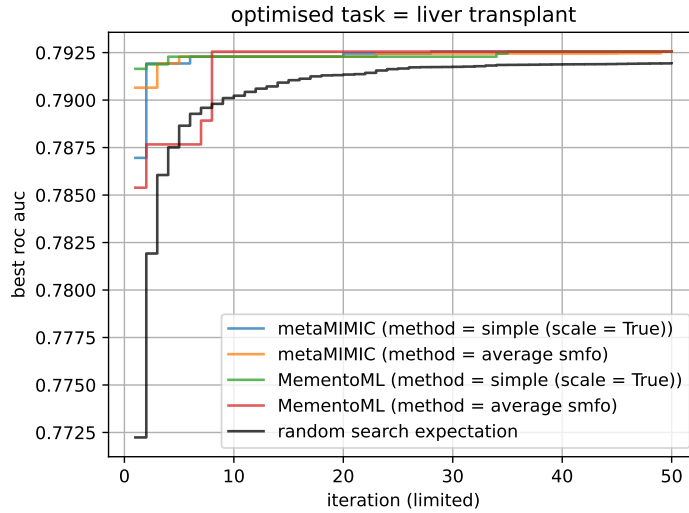
Uzyskane wyniki pokazują, że średnia wartość podobieństwa jest niemal dwukrotnie większa w przypadku porównania do zadań predykcyjnych, które pochodzą z eksperymentu *metaMIMIC*. Nie jest to zaskoczeniem, ponieważ eksperyment ten obejmuje zadania predykcyjne wyłącznie z dziedziny medycyny, a eksperyment *MementoML* bazuje na zadaniach predykcyjnych o różnorodnej tematyce. Powstaje jednak pytanie, czy fakt ten ma odwzorowanie w szybkości transferu hiperparametrów.

Rysunki 4.4 przedstawia wizualizację analiz szybkości transferu hiperparametrów pomiędzy zadaniem *lungs cancer*, a zadaniami pochodzącymi z eksperymentów *metaMIMIC* oraz *MementoML*. Zastosowane zostały obie dostępne metody konstrukcji portfolio zestawów hiperparametrów (metoda *Simple* - górny panel, metoda *Average SMFO* - dolny panel). Wykonane analizy pokazują, że zgodnie z oczekiwaniami transfer hiperparametrów jest szybszy w przypadku bazy doświadczeń opartej o eksperyment *metaMIMIC*. Ponadto, w rozważanym przypadku metoda *Average SMFO* zapewnia szybszą zbieżność, choć charakteryzuje ją gorszy start.



Rysunek 4.4: Wizualizacja analizy szybkości transferu hiperparametrów dla docelowego zadania predykcyjnego nazwie *liver transplant*, metodą konstrukcji portfolio *Simple* (górny panel) oraz *Average SMFO* (dolny panel). Zadania źródłowe pochodzą z eksperymentu *metaMIMIC* (niebieska linia) oraz z eksperymentu *MementoML* (pomarańczowa linia).

Cały przedstawiony proces został powtórzony dla zadania predykcyjnego nazwie *liver transplant*. W tym przypadku średnie podobieństwo rankingów zestawów hiperparametrów wynosi około 0.25, przy rozważaniu wyników z eksperymentu *metaMIMIC*, oraz około 0.10, przy rozważaniu zadań z eksperymentu *MementoML*. Rysunek 4.5 przedstawia analogiczną wizualizację analizy szybkości transferu hiperparametrów. Tym razem obie metody konstrukcji portfolio zestawów hiperparametrów zostały uwzględnione na jednym wykresie. Opisywany wykres pokazuje, że transfer hiperparametrów jest efektywny w niemal każdym przypadku i jedynie zastosowanie połączenia bazy doświadczeń, opartej o eksperyment *MementoML*, z metodą konstrukcji portfolio *Average SMFO* skutkuje nieco wolniejszą zbieżnością.



Rysunek 4.5: Wizualizacja analizy szybkości transferu hiperparametrów pomiędzy zadaniem predykcyjnym nazwie *liver transplant*, a zadaniami predykcyjnymi pochodzącymi z eksperymentów *metaMIMIC* (niebieska linia - metoda konstrukcji portfolio *Simple*, pomarańczowa linia - metoda konstrukcji portfolio *Average SMFO*) i *MementoML* (zielona linia - metoda konstrukcji portfolio *Simple*, czerwona linia - metoda konstrukcji portfolio *Average SMFO*).

5. Podsumowanie

W niniejszej pracy przedstawione zostało programistyczne narzędzie **HyPerf**. Zostało ono zaimplementowane w języku programowania Python. Głównym przeznaczeniem pakietu jest wspieranie badaczy, zajmujących się *meta-learningiem*, a w szczególności transferem hiperparametrów.

Wyzwanie, jakim jest badanie transferowalności hiperparametrów, wymaga analizowania wyników licznych eksperymentów obliczeniowych. Ich duży wolumen oraz wysoki stopień złożoności są przeciwwskazaniem do przechowywania w postaci nieustrukturyzowanych plików płaskich. Moduł składowania wspiera uporządkowane przechowywanie wyników eksperymentów obliczeniowych oraz współdzielenie ich z innymi badaczami, dzięki przeznaczonemu ku temu rozwiązaniu bazodanowemu. Dodatkowo, przykładowa baza danych została zasilona wynikami eksperymentów, przeprowadzonych wcześniej w grupie badawczej *MI2*.

Moduł analityczny automatyzuje analizowanie transferowalności konfiguracji, dzięki implementacji metod, które były najczęściej stosowane w dotychczasowej literaturze. Oprócz oczywistych udogodnień, według mojej wiedzy jest to pierwszy pakiet, który proponuje zestaw konkretnych metod analiz do badania zjawiska transferowalności hiperparametrów. Dzięki jego stosowaniu, zwiększona zostanie porównywalność prac badawczych, dotyczących tego zagadnienia. Jest to istotne, ponieważ dotychczas większość prac stosowała różne podejścia do oceny metod optymalizacji.

Zrealizowana implementacja została przetestowana pod kątem poprawności działania i wydoby pracy. W tym celu wykonane zostały zarówno automatyczne, jak i manualne testy. Ich pozytywne rezultaty świadczą o użyteczności opracowanego rozwiązania. Ponadto, przeprowadzone analizy potwierdzają potencjał transferu hiperparametrów jako metody ich optymalizacji. Usystematyzowanie zarówno składowania, jak i analizowania wyników eksperymentów, powinno znacząco wpłynąć na badania prowadzone przez grupę *MI2*. W przypadku upowszechnienia pakietu wśród innych badaczy *AutoML* i *meta-learningu*, **HyPerf** wzmocni wymianę informacji i udostępnianie wyników eksperymentów. Mam nadzieję, że opisane narzędzie znajdzie zastosowanie i przyczyni się do rozwoju lepszych metod, automatyzujących żmudne zadanie optymalizacji hiperparametrów algorytmów uczenia maszynowego.

5.1. Dalsze prace

Zarówno w trakcie, jak i po zakończeniu prac nad przedstawionym rozwiązaniem, zidentyfikowane zostały jego obszary, które warto poświęcić dodatkowej uwagi oraz dalszej rozbudowy. Głównym powodem ich niewdrożenia jest krótki czas realizacji projektu wyłącznie przez jedną osobę. Poniżej znajduje się lista określonych kierunków rozwoju:

- reimplementacja modułu składowania zgodnie z wzorcem mapowania obiektowo-relacyjnego (*ORM*) (na przykład, przy użyciu pakietu *SQLAlchemy*),

- rozwój struktury danych i metod analitycznych o podobieństwo zadań predykcyjnych,
- rozbudowa modułu analitycznego o kolejne typy analiz transferowalności hiperparametrów
- przygotowanie modułu, automatyzującego wykonywanie eksperymentów obliczeniowych,
- przeprowadzenie kolejnych eksperymentów obliczeniowych i uzupełnienie nimi przykładowej bazy danych.

Bibliografia

- [1] James Bergstra i Yoshua Bengio. „Random Search for Hyper-Parameter Optimization”. W: *Journal of Machine Learning Research* 13.10 (2012), s. 281–305.
- [2] James Bergstra i in. „Algorithms for Hyper-Parameter Optimization”. W: *Advances in Neural Information Processing Systems*. T. 24. 2011.
- [3] Mauro Birattari i in. „F-Race and Iterated F-Race: An Overview”. W: *Experimental Methods for the Analysis of Optimization Algorithms*. 2010, s. 311–336.
- [4] Anne-Laure Boulesteix i Martin Slawski. „Stability and aggregation of ranked gene lists”. W: *Briefings in Bioinformatics* 10.5 (2009), s. 556–568.
- [5] Pavel B. Brazdil i Carlos Soares. *A Comparison of Ranking Methods for Classification Algorithm Selection*. 2000.
- [6] Tianqi Chen i Carlos Guestrin. „XGBoost: A Scalable Tree Boosting System”. W: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD '16. 2016, s. 785–794.
- [7] Katharina Eggenberger i in. *HPOBench: A Collection of Reproducible Multi-Fidelity Benchmark Problems for HPO*. 2021.
- [8] Matthias Feurer i in. *Auto-Sklearn 2.0: Hands-free AutoML via Meta-Learning*. 2021.
- [9] Christophe Giraud-Carrier i Foster Provost. „Toward a justification of meta-learning: Is the no free lunch theorem a show-stopper?” W: *Proceedings of the ICML-2005 Workshop on Meta-learning*. 2005, s. 12–19.
- [10] Isabelle Guyon i in. „Analysis of the AutoML Challenge Series 2015-2018”. W: *Automated Machine Learning: Methods, Systems, Challenges*. 2019, s. 177–219.
- [11] Richard D. Hipp. *SQLite*. Wer. 3.36.0. 2021. URL: <https://www.sqlite.org/index.html>.
- [12] John D. Hunter. *Matplotlib*. Wer. 3.4.2. 2021. URL: <https://matplotlib.org/>.
- [13] Frank Hutter, Holger H. Hoos i Kevin Leyton-Brown. „Sequential Model-Based Optimization for General Algorithm Configuration”. W: *Learning and Intelligent Optimization*. 2011, s. 507–523.
- [14] Giuseppe Jurman i in. „Canberra Distance on Ranked Lists”. W: 2009.
- [15] Holger Krekel. *PyTest*. Wer. 6.2.4. 2021. URL: <https://docs.pytest.org/>.
- [16] Wojciech Kretowicz i Przemysław Biecek. *MementoML: Performance of selected machine learning algorithm configurations on OpenML100 datasets*. 2020. arXiv: 2008.13162 [cs.LG].
- [17] Mohamed Maher i Sherif Sakr. *SmartML: A Meta Learning-Based Framework for Automated Selection and Hyperparameter Tuning for Machine Learning Algorithms*. 2019.

- [18] Wes McKinney. *Pandas*. Wer. 1.3.3. 2021. URL: <https://pandas.pydata.org/>.
- [19] Travis Oliphant. *NumPy*. Wer. 1.20.3. 2021. URL: <https://numpy.org/>.
- [20] F. Pedregosa i in. „Scikit-learn: Machine Learning in Python”. W: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830.
- [21] Florian Pfisterer i in. „Learning Multiple Defaults for Machine Learning Algorithms”. W: *Proceedings of the Genetic and Evolutionary Computation Conference Companion*. 2021, s. 241–242.
- [22] Philipp Probst, Anne-Laure Boulesteix i Bernd Bischl. „Tunability: Importance of Hyperparameters of Machine Learning Algorithms”. W: *Journal of Machine Learning Research* 20.53 (2019), s. 1–32.
- [23] Guido Van Rossum i Fred L. Drake. *Python 3*. Wer. 3.9.7. 2021. URL: <https://www.python.org/>.
- [24] Jasper Snoek, Hugo Larochelle i Ryan P. Adams. „Practical Bayesian Optimization of Machine Learning Algorithms”. W: *Advances in Neural Information Processing Systems*. T. 25. 2012, s. 2951–2959.
- [25] Joaquin Vanschoren. „Meta-Learning”. W: *Automated Machine Learning: Methods, Systems, Challenges*. 2019, s. 35–61.
- [26] Joaquin Vanschoren i in. „OpenML: Networked Science in Machine Learning”. W: *SIGKDD Explorations* 15.2 (2013), s. 49–60.
- [27] Michael Waskom. *Seaborn*. Wer. 0.11.2. 2021. URL: <https://seaborn.pydata.org/>.
- [28] Martin Wistuba, Nicolas Schilling i Lars Schmidt-Thieme. „Sequential Model-Free Hyperparameter Tuning”. W: *IEEE International Conference on Data Mining*. T. 2016-January. 2016, s. 1033–1038.
- [29] Katarzyna Woźnica i in. *Consolidated learning – a domain-specific model-free optimization strategy with examples for XGBoost and MIMIC-IV*. 2022. arXiv: 2201.11815.

Wykaz skrótów

ANE	Average Normalized Error
API	Application Programming Interface
AUC	Area Under the Curve
CANE	Cumulative Average Normalized Error
DDL	Data Definition Language
DML	Data Manipulation Language
DQL	Data Query Language
ORM	Object Relational Mapping
RDBMS	Relational Database Management System
ROC	Receiver Operating Characteristic
SMFO	Sequential Model Free Optimization
SQL	Structured Query Language
SVM	Support Vector Machine

Spis rysunków

1.1	Wzrost popularności wyszukiwania hasła "hyperparameter optimization"	11
3.1	Diagram struktury bazy danych modułu składowania	23
3.2	Diagram metod dostępnych w programistycznym interfejsie do bazy danych . . .	28
3.3	Przykład wizualizacji analizy podobieństwa rankingów zestawów hiperparametrów	35
3.4	Przykład wizualizacji analizy szybkości transferu hiperparametrów	36
3.5	Przykład wizualizacji analizy średniego znormalizowanego błędu strojenia	36
4.1	Raport z automatycznych testów jednostkowych	42
4.2	Podsumowanie przykładowej bazy danych	43
4.3	Demonstracyjna analiza podobieństwa rankingów zestawów hiperparametrów . .	44
4.4	Demonstracyjna analiza szybkości transferu hiperparametrów 1	45
4.5	Demonstracyjna analiza szybkości transferu hiperparametrów 2	46

Dodatek A - specyfikacja API modułu składowania

`Database.__init__`

konstruktor klasy

Jeżeli podany w argumencie `path` plik istnieje, to utworzone zostaje połączenie z istniejącą bazą danych, a jeżeli nie, ale jego ścieżka jest poprawna, to utworzona zostaje nowa baza danych, która następnie inicjalizowana jest odpowiednią strukturą i również utworzone zostaje połączenie.

Przyjmowane argumenty:

`path (str)`

ścieżka do pliku o rozszerzeniu `.sqlite`

Plik lub przynajmniej ścieżka jego położenia muszą istnieć w systemie plików.

Zwracane wartości:

obiekt klasy `Database`

reprezentacja połączenia z bazą danych

`Database.available_results`

metoda służąca do pozyskania podsumowania dostępnych w bazie danych wyników

Przyjmowane argumenty:

`ignore_empty (bool) = True`

pominięcie wierszy reprezentujących brak wyników

Zwracane wartości:

obiekt klasy `pandas.DataFrame`

ramka danych zawierająca podsumowanie dostępnych w bazie danych wyników

Zwracana ramka danych zawiera wiersze odpowiadające wszystkim dopuszczalnym kombinacjom definicji. Kolumny `available_results` oraz `possible_results` informują odpowiednio, ile wyników jest oraz ile wyników mogłoby być dostępnych dla danej kombinacji definicji. Kolumna `ratio` stanowi o wypełnieniu wynikami.

`Database.summary`

metoda służąca do podsumowania zawartości bazy danych

Wywołanie metody powoduje wyświetlenie podsumowania zawartości bazy danych w postaci liczby poszczególnych definicji i wyników na `stdout`.

Przyjmowane argumenty: -

Zwracane wartości: -

Database.close

metoda służąca do zamknięcia połączenia z bazą danych

Przyjmowane argumenty: -

Zwracane wartości: -

Database.vacuum

metoda służąca do wykonania kompaktacji na bazie danych

W technologii SQLite usuwane wiersze są wyłącznie oznaczane jako możliwe do nadpisania. Zmniejszenie pliku reprezentującego bazę danych po usunięciu części wierszy wymaga wykonania kompaktacji.

Przyjmowane argumenty: -

Zwracane wartości: -

Database.get_task_types

metoda służąca do pozyskania definicji typów zadań predykcyjnych

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje typów zadań predykcyjnych

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumnę odpowiadającą nazwom typów zadań predykcyjnych.

Database.get_metrics

metoda służąca do pozyskania definicji metryk jakości modeli

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje metryk jakości modeli

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom metryk jakości modeli oraz nazwom typów zadań predykcyjnych, do których się one odwołują.

Database.get_tasks

metoda służąca do pozyskania definicji zadań predykcyjnych

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje zadań predykcyjnych

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom i opisom zadań predykcyjnych, nazwom typów zadań predykcyjnych, do których się one odwołują oraz nazwom tagów, które są do nich przypisane.

Database.get_task_tags

metoda służąca do pozyskania tagów zadań predykcyjnych

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca tagi zadań predykcyjnych

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom tagów oraz nazwom zadań predykcyjnych, do których są one przypisane.

Database.get_algorithms

metoda służąca do pozyskania definicji algorytmów uczenia maszynowego

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje algorytmów uczenia maszynowego

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom, opisom i wersjom algorytmów uczenia maszynowego, nazwom typów zadań predykcyjnych, do których się one odwołują oraz nazwom tagów, które są do nich przypisane.

Database.get_algorithm_tags

metoda służąca do pozyskania tagów algorytmów uczenia maszynowego

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca tagi algorytmów uczenia maszynowego

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom tagów oraz nazwom algorytmów uczenia maszynowego, do których są one przypisane.

Database.get_grids

metoda służąca do pozyskania definicji siatek hiperparametrów

Przyjmowane argumenty: -

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje siatek hiperparametrów

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające nazwom i opisom siatek hiperparametrów oraz nazwom algorytmów uczenia maszynowego, do których są one przypisane.

Database.get_sets

metoda służąca do pozyskania definicji zestawów hiperparametrów, pochodzących z określonej siatki hiperparametrów

Przyjmowane argumenty:

grid (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odwoływać się do istniejącej definicji.

sanitize (str) = False

czyszczenie zwracanej ramki danych

Jeżeli dla niektórych hiperparametrów postać numeryczna nie jest wystarczająca, to zwracana ramka danych nie będzie jej zawierać, a jeżeli jest, to zwracana ramka nie będzie zawierać postaci tekstowej. Gwarantuje to przełożenie jednego hiperparametru na jedną, możliwie odpowiednią kolumnę zwracanej ramki danych.

Zwracane wartości:

obiekt klasy pandas.DataFrame

ramka danych zawierająca definicje zestawów hiperparametrów

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające numerom porządkowym zestawów hiperparametrów oraz wartościom składających się na nie hiperparametrów w postaci zarówno tekstowej, jak i numerycznej.

Database.get_results

metoda służąca do pozyskania wyników, odpowiadających określonej kombinacji definicji

Przyjmowane argumenty:

task (str)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odwoływać się do istniejącej definicji.

algorithm (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odwoływać się do istniejącej definicji.

grid (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odwoływać się do istniejącej definicji.

metric (str)

nazwa metryki jakości modeli

Nazwa metryki jakości modeli musi odwoływać się do istniejącej definicji.

concat_ready (bool) = True

postać zwracanej ramki danych przystosowana do łączenia w pionie

W przypadku wartości prawdziwej, zwracana ramka danych zawiera dodatkowe kolumny (**task**, **algorithm**, **grid**, **metric**), które wypełnione są wyłącznie pojedynczymi unikalnymi wartościami, ale pozwalają na łatwe łączenie w pionie z innymi ramkami danych tego samego typu.

`timestamps (bool) = False`

zawarcie kolumn odpowiadających znacznikom czasowym

W przypadku wartości prawdziwej, zwracana ramka danych zawiera dodatkowe kolumny (`calculated_timestamp`, `inserted_timestamp`), które informują o odpowiednio czasie obliczenia i składowania wyników.

Zwracane wartości:

obiekt klasy `pandas.DataFrame`

ramka danych zawierająca wyniki

Ramka danych indeksowana jest tak samo, jak wiersze w bazie danych i zawiera kolumny odpowiadające numerom porządkowym zestawów hiperparametrów, wartościom metryki jakości modeli w postaci zarówno tekstowej, jak i numerycznej oraz potencjalnie dodatkowe kolumny, wynikające z wartości argumentów `concat_ready` i `timestamps`.

`Database.add_task_type`

metoda służąca do dodania definicji typu zadań predykcyjnych

Przyjmowane argumenty:

`name (str)`

nazwa typu zadań predykcyjnych

Nazwa typu zadań predykcyjnych musi być unikalna.

Zwracane wartości: -

`Database.add_metric`

metoda służąca do dodania definicji metryki jakości modeli

Przyjmowane argumenty:

`task_type (str)`

nazwa typu zadań predykcyjnych

Nazwa typu zadań predykcyjnych musi odwoływać się do istniejącej definicji.

`name (str)`

nazwa metryki jakości modeli

Nazwa metryki jakości modeli musi być unikalna.

`description (str, NoneType) = None`

opcjonalny opis metryki jakości modeli

Zwracane wartości: -

`Database.add_task`

metoda służąca do dodania definicji zadania predykcyjnego

Przyjmowane argumenty:

`task_type (str)`

nazwa typu zadań predykcyjnych

Nazwa typu zadania predykcyjnego musi odwoływać się do istniejącej definicji.

name (str)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi być unikalna.

description (str, NoneType) = None

opcjonalny opis zadania predykcyjnego

Zwracane wartości: -

Database.add_task_tag

metoda służąca do dodania tagu zadań predykcyjnych

Przyjmowane argumenty:

name (str)

nazwa tagu zadań predykcyjnych

Nazwa tagu zadań predykcyjnych musi być unikalna.

Zwracane wartości: -

Database.add_algorithm

metoda służąca do dodania definicji algorytmu uczenia maszynowego

Przyjmowane argumenty:

task_type (str)

nazwa typu zadań predykcyjnych

Nazwa typu zadań predykcyjnych musi odwoływać się do istniejącej definicji.

name (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi być unikalna.

version (str)

wersja algorytmu uczenia maszynowego

description (str, NoneType) = None

opcjonalny opis algorytmu uczenia maszynowego

Zwracane wartości: -

Database.add_algorithm_tag

metoda służąca do dodania tagu algorytmów uczenia maszynowego

Przyjmowane argumenty:

name (str)

nazwa tagu algorytmów uczenia maszynowego

Nazwa tagu algorytmów uczenia maszynowego musi być unikalna.

Zwracane wartości: -

Database.add_grid

metoda służąca do dodania definicji siatki hiperparametrów

Przyjmowane argumenty:

name (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi być unikalna.

description (str, NoneType) = None

opcjonalny opis siatki hiperparametrów

Zwracane wartości: -

Database.add_sets

metoda służąca do dodania definicji zestawów hiperparametrów do określonej siatki hiperparametrów

Definicje zestawów hiperparametrów, a także samych hiperparametrów, są tworzone automatycznie, na podstawie przekazanej do metody ramki danych o odpowiedniej strukturze. Wartości hiperparametrów zawsze składowane są w postaci tekstowej, a decyzja, dotycząca przechowywania w postaci numerycznej, podejmowana jest na podstawie typów odpowiednich kolumn, zawartych w przekazanej do metody ramce danych.

Przyjmowane argumenty:

df (pandas.DataFrame)

ramka danych zawierająca zestawy hiperparametrów

Ramka danych musi zawierać kolumnę typu `int` o nazwie zgodnej z wartością argumentu `number_col`, odpowiadającą unikalnym numerom porządkowym zestawów hiperparametrów, a także kolumny odpowiadające poszczególnym hiperparametrom i ich wartościom. Nazwy hiperparametrów pozyskiwane są z nazw odpowiednich kolumn. Ramka danych może zawierać brakujące wartości, ale musi zawierać kolumny dla wszystkich hiperparametrów definiowanych przez już istniejące w siatce zestawy.

grid (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odwoływać się do istniejącej definicji.

number_col (str) = 'number'

nazwa kolumny ramki danych `df` zawierającej numery porządkowe zestawów hiperparametrów

expand_grid (bool) = False

pozwolenie na rozszerzenie siatki o nowe hiperparametry względem zestawów już istniejących w siatce

Zwracane wartości: -

Database.add_results

metoda służąca do dodania wyników dla określonej dopuszczalnej kombinacji definicji

Wyniki są przekazywane w postaci ramki danych o odpowiedniej strukturze. Możliwe jest jednoczesne przekazanie wyników dla wielu metryk jakości modeli (jest to typowa forma, używana w wyniku eksperymentów obliczeniowych). Wartości metryk jakości modeli zawsze składowane są w postaci tekstowej, a decyzja, dotycząca przechowywania w postaci numerycznej, podejmowana jest na podstawie typów odpowiednich kolumn, zawartych w przekazanej do metody ramce danych.

Przyjmowane argumenty:**df** (`pandas.DataFrame`)

ramka danych zawierająca wyniki

Ramka danych musi zawierać kolumnę typu `int` o nazwie zgodnej z wartością argumentu `number_col`, odpowiadającą wszystkim numerom porządkowym zestawów hiperparametrów zdefiniowanych w siatce, określonej przez argument `grid`. Pozostałe kolumny ramki danych (z pominięciem opcjonalnej kolumny, określonej przez argument `calculated_col`), muszą odpowiadać metrykom jakości modeli, które są zdefiniowane w bazie danych i tyczą się tego samego typu zadań predykcyjnych.

task (`str`)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odwoływać się do istniejącej definicji.

algorithm (`str`)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odwoływać się do istniejącej definicji. Algorytm uczenia maszynowego musi odnosić się do tego samego typu zadań predykcyjnych, co zadanie predykcyjne, określone przez argument `task`.

grid (`str`)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odwoływać się do istniejącej definicji. Siatka hiperparametrów musi być powiązana z algorytmem uczenia maszynowego, określonym przez argument `algorithm`.

number_col (`str`) = 'number'

nazwa kolumny ramki danych `df` zawierającej numery porządkowe zestawów hiperparametrów

calculated_col (`str, NoneType`) = `None`nazwa kolumny ramki danych `df` zawierającej czas obliczenia wyników

Kolumna ramki danych `df`, określona przez argument `calculated_col`, musi być typu `datetime`.

Zwracane wartości: -**Database.modify_task_type**

metoda służąca do modyfikacji definicji typu zadań predykcyjnych

Przyjmowane argumenty:**name** (`str`)

nazwa typu zadań predykcyjnych

Nazwa typu zadań predykcyjnych musi odnosić się do istniejącej definicji.

new_name (`str`)

opcjonalna nowa nazwa typu zadań predykcyjnych

Nowa nazwa typu zadań predykcyjnych musi być unikalna.

Zwracane wartości: -

Database.modify_metric

metoda służąca do modyfikacji definicji metryki jakości modeli

Przyjmowane argumenty:

name (str)

nazwa metryki jakości modeli

Nazwa metryki jakości modeli musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa metryki jakości modeli

Nowa nazwa metryki jakości modeli musi być unikalna.

new_description (str, NoneType)

opcjonalny nowy opis metryki jakości modeli

Nowy opis metryki jakości modeli może być typu **NoneType**, co równoważne jest z jego usunięciem.

Zwracane wartości: -

Database.modify_task

metoda służąca do modyfikacji definicji zadania predykcyjnego

Przyjmowane argumenty:

name (str)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa zadania predykcyjnego

Nowa nazwa zadania predykcyjnego musi być unikalna.

new_description (str, NoneType)

opcjonalny nowy opis zadania predykcyjnego

Nowy opis zadania predykcyjnego modeli może być typu **NoneType**, co równoważne jest z jego usunięciem.

Zwracane wartości: -

Database.modify_task_tag

metoda służąca do modyfikacji definicji tagu zadań predykcyjnych

Przyjmowane argumenty:

name (str)

nazwa tagu zadań predykcyjnych

Nazwa tagu zadań predykcyjnych musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa tagu zadań predykcyjnych

Nowa nazwa tagu zadań predykcyjnych musi być unikalna.

Zwracane wartości: -

Database.modify_algorithm

metoda służąca do modyfikacji definicji algorytmu uczenia maszynowego

Przyjmowane argumenty:

name (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa algorytmu uczenia maszynowego

Nowa nazwa algorytmu uczenia maszynowego musi być unikalna.

new_version (str)

opcjonalna nowa wersja algorytmu uczenia maszynowego

new_description (str, NoneType)

opcjonalny nowy opis algorytmu uczenia maszynowego

Nowy opis algorytmu uczenia maszynowego może być typu `NoneType`, co równoważne jest z jego usunięciem.

Zwracane wartości: -

Database.modify_algorithm_tag

metoda służąca do modyfikacji definicji tagu algorytmów uczenia maszynowego

Przyjmowane argumenty:

name (str)

nazwa tagu algorytmów uczenia maszynowego

Nazwa tagu algorytmów uczenia maszynowego musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa tagu algorytmów uczenia maszynowego

Nowa nazwa tagu algorytmów uczenia maszynowego musi być unikalna.

Zwracane wartości: -

Database.modify_grid

metoda służąca do modyfikacji definicji siatki hiperparametrów

Przyjmowane argumenty:

name (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

new_name (str)

opcjonalna nowa nazwa siatki hiperparametrów

Nowa nazwa siatki hiperparametrów musi być unikalna.

new_description (str, NoneType)

opcjonalny nowy opis siatki hiperparametrów

Nowy opis siatki hiperparametrów może być typu `NoneType`, co równoważne jest z jego usunięciem.

Zwracane wartości: -

`Database.remove_task_type`

metoda służąca do usunięcia definicji typu zadań predykcyjnych

Jeżeli w bazie danych znajdują się metryki jakości modeli, zadania predykcyjne lub algorytmy uczenia maszynowego, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie wykonana. Usuwanie odbywa się w sposób kaskadowy i może pociągnąć za sobą usunięcie innych, pośrednio zależnych rekordów (wyników).

Przyjmowane argumenty:

`name (str)`

nazwa typu zadań predykcyjnych

Nazwa typu zadań predykcyjnych musi odnosić się do istniejącej definicji.

`cascade (bool) = False`

usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_metric`

metoda służąca do usunięcia definicji metryki jakości modeli

Jeżeli w bazie danych znajdują się wyniki, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`name (str)`

nazwa metryki jakości modeli

Nazwa metryki jakości modeli musi odnosić się do istniejącej definicji.

`cascade (bool) = False`

usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_task`

metoda służąca do usunięcia definicji zadania predykcyjnego

Jeżeli w bazie danych znajdują się tagi powiązane z usuwanym rekordem, a wartość argumentu `ignore` jest prawdziwa, to zostanie on usunięty. W przeciwnym wypadku operacja nie zostanie ukończona.

Jeżeli w bazie danych znajdują się wyniki, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`name (str)`

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odnosić się do istniejącej definicji.

`ignore (bool) = False`
ignorowanie powiązań
`cascade (bool) = False`
usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_task_tag`

metoda służąca do usunięcia tagu zadań predykcyjnych

Jeżeli w bazie danych znajdują się zadania predykcyjne powiązane z usuwanym rekordem, a wartość argumentu `ignore` jest prawdziwa, to zostanie on usunięty. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`name (str)`
nazwa tagu zadań predykcyjnych
Nazwa tagu zadań predykcyjnych musi odnosić się do istniejącej definicji.
`ignore (bool) = False`
ignorowanie powiązań

Zwracane wartości: -

`Database.remove_algorithm`

metoda służąca do usunięcia definicji algorytmu uczenia maszynowego

Jeżeli w bazie danych znajdują się tagi lub siatki hiperparametrów powiązane z usuwanym rekordem, a wartość argumentu `ignore` jest prawdziwa, to zostanie on usunięty. W przeciwnym wypadku operacja nie zostanie ukończona.

Jeżeli w bazie danych znajdują się wyniki, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`name (str)`
nazwa algorytmu uczenia maszynowego
Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.
`ignore (bool) = False`
ignorowanie powiązań
`cascade (bool) = False`
usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_algorithm_tag`

metoda służąca do usunięcia tagu algorytmów uczenia maszynowego

Jeżeli w bazie danych znajdują się algorytmy uczenia maszynowego powiązane z usuwanym rekordem, a wartość argumentu `ignore` jest prawdziwa, to zostanie on usunięty. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`name (str)`

nazwa tagu algorytmów uczenia maszynowego

Nazwa tagu algorytmów uczenia maszynowego musi odnosić się do istniejącej definicji.

`ignore (bool) = False`

ignorowanie powiązań

Zwracane wartości: -

`Database.remove_grid`

metoda służąca do usunięcia definicji siatki hiperparametrów

Jeżeli w bazie danych znajdują się algorytmy uczenia maszynowego powiązane z usuwanym rekordem, a wartość argumentu `ignore` jest prawdziwa, to zostanie on usunięty. W przeciwnym wypadku operacja nie zostanie ukończona.

Jeżeli w bazie danych znajdują się zestawy hiperparametrów, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona. Usuwanie odbywa się w sposób kaskadowy i może pociągnąć za sobą usunięcie innych, pośrednio zależnych rekordów (wyników).

Przyjmowane argumenty:

`name (str)`

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

`ignore (bool) = False`

ignorowanie powiązań

`cascade (bool) = False`

usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_set`

metoda służąca do usunięcia definicji zestawów hiperparametrów

Jeżeli w bazie danych znajdują się wyniki, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`grid (str)`

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

`numbers (int, array(int), NoneType)`

numery porządkowe zestawów hiperparametrów

Wartość typu `NoneType` oznacza usunięcie wszystkich zestawów hiperparametrów z siatki określonej przez argument `grid`.

`cascade (bool) = False`

usuwanie kaskadowe

Zwracane wartości: -

`Database.remove_results`

metoda służąca do usunięcia wyników

Jeżeli w bazie danych znajdują się wyniki, odnoszące się do usuwanego rekordu, a wartość argumentu `cascade` jest prawdziwa, to zostaną one również usunięte. W przeciwnym wypadku operacja nie zostanie ukończona.

Przyjmowane argumenty:

`task (str)`

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odnosić się do istniejącej definicji.

`algorithm (str)`

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

`grid (str)`

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

`metric (str)`

nazwa metryki jakości modeli

Nazwa metryki jakości modeli musi odnosić się do istniejącej definicji.

`numbers (int, array(int), NoneType)`

numery porządkowe zestawów hiperparametrów

Wartość typu `NoneType` oznacza usunięcie wyników dla wszystkich zestawów hiperparametrów z siatki określonej przez argument `grid`.

`cascade (bool) = False`

usuwanie kaskadowe

Zwracane wartości: -

`Database.set_task_tag`

metoda służąca do dodania przypisania tagu zadań predykcyjnych

Przypisanie danego tagu zadań predykcyjnych i zadania predykcyjnego może występować w bazie danych tylko raz.

Przyjmowane argumenty:

task_tag (str)

nazwa tagu zadań predykcyjnych

Nazwa tagu zadań predykcyjnych musi odnosić się do istniejącej definicji.

task (str)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Database.set_algorithm_tag

metoda służąca do dodania przypisania tagu algorytmów uczenia maszynowego

Przypisanie danego tagu algorytmów uczenia maszynowego i algorytmu uczenia maszynowego może występować w bazie danych tylko raz.

Przyjmowane argumenty:

algorithm_tag (str)

nazwa tagu algorytmów uczenia maszynowego

Nazwa tagu algorytmów uczenia maszynowego musi odnosić się do istniejącej definicji.

algorithm (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Database.set_grid

metoda służąca do dodania przypisania siatki hiperparametrów

Przypisanie danej siatki hiperparametrów i algorytmu uczenia maszynowego może występować w bazie danych tylko raz.

Przyjmowane argumenty:

grid (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

algorithm (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Database.unset_task_tag

metoda służąca do usunięcia przypisania tagu zadań predykcyjnych

Przypisanie tagu zadań predykcyjnych, określonego przez argument **task_tag**, i zadania predykcyjnego, określonego przez argument **task**, musi istnieć w bazie danych.

Przyjmowane argumenty:

task_tag (str)

nazwa tagu zadań predykcyjnych

Nazwa tagu zadań predykcyjnych musi odnosić się do istniejącej definicji.

task (str)

nazwa zadania predykcyjnego

Nazwa zadania predykcyjnego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Database.unset_algorithm_tag

metoda służąca do usunięcia przypisania tagu algorytmów uczenia maszynowego

Przypisanie tagu algorytmów uczenia maszynowego, określonego przez argument **algorithm_tag**, i algorytmu uczenia maszynowego, określonego przez argument **algorithm**, musi istnieć w bazie danych.

Przyjmowane argumenty:

algorithm_tag (str)

nazwa tagu algorytmów uczenia maszynowego

Nazwa tagu algorytmów uczenia maszynowego musi odnosić się do istniejącej definicji.

algorithm (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Database.unset_grid

metoda służąca do usunięcia przypisania siatki hiperparametrów

Przypisanie siatki hiperparametrów, określonej przez argument **grid**, i algorytmu uczenia maszynowego, określonego przez argument **algorithm**, musi istnieć w bazie danych.

Przyjmowane argumenty:

grid (str)

nazwa siatki hiperparametrów

Nazwa siatki hiperparametrów musi odnosić się do istniejącej definicji.

algorithm (str)

nazwa algorytmu uczenia maszynowego

Nazwa algorytmu uczenia maszynowego musi odnosić się do istniejącej definicji.

Zwracane wartości: -

Dodatek B - specyfikacja API modułu analitycznego

`RankingSimilarityAnalyser.__init__`

konstruktor klasy

Klasa `RankingSimilarityAnalyser` odpowiada za analizę podobieństwa rankingów zestawów hiperparametrów. Porównania wykonywane są dla wszystkich kombinacji zadań predykcyjnych określonych w lewej i prawej ramce danych. Obie ramki danych muszą zawierać wyniki odnoszące się do tego samego algorytmu uczenia maszynowego (dopuszczalne są różne jego wersje), tej samej siatki hiperparametrów oraz tej samej numerycznej metryki jakości modeli.

Przyjmowane argumenty:

`left (pandas.DataFrame)`

lewa ramka danych z wynikami

Struktura ramki danych musi być zgodna z używaną przez moduł składowania.

`right (pandas.DataFrame)`

prawa ramka danych z wynikami

Struktura ramki danych musi być zgodna z używaną przez moduł składowania.

`ascending (bool)`

charakterystyka metryki jakości modeli

Wartość fałszywa oznacza, że większa wartość metryki świadczy o lepszym modelu, a wartość prawdziwa przeciwnie.

`method (str)`

strategia analityczna - miara podobieństwa rankingów

Możliwe wartości to `'po'` (*Percentage of Overlap*), `'ct'` (*Correspondence at the Top*), `'os'` (*Overlap Score*) oraz `'cd'` (*Canberra Distance*).

`label (str)`

etykieta analizatora

W przypadku wizualizacji analizatora jego etykieta używana jest jako tytuł wykresu.

Zwracane wartości:

obiekt klasy `RankingSimilarityAnalyser`

instancja analizatora zainicjalizowana danymi

`RankingSimilarityAnalyser.calculate`

metoda służąca do obliczenia wyników analizy

Przyjmowane argumenty:

****kwargs**

opcjonalne argumenty zależne od wybranej miary podobieństwa rankingów

Miary podobieństwa rankingów *Percentage of Overlap* i *Correspondence at the Top* wymagają podania wartości parametru `k` (`int`) ($k > 0$, $k < p$, gdzie p to liczność siatki hiperparametrów). Miara podobieństwa rankingów *Overlap Score* wymaga podania wartości parametru `alpha` (`float`) ($\alpha > 0$).

Zwracane wartości:**obiekt klasy `pandas.DataFrame`**

ramka danych zawierająca wyniki analizy

Indeks ramki danych stanowią zadania predykcyjne z lewej ramki danych, a kolumny zadania predykcyjne z prawej ramki danych. Wartości w poszczególnych komórkach to wartości miary podobieństwa rankingów dla danej pary zadań predykcyjnych.

`RankingSimilarityAnalyser.plot`

metoda służąca do wizualizacji wyników analizy

Przyjmowane argumenty:

`ax` (`matplotlib.axes.Axes`, `NoneType`) = `None`

opcjonalna kanwa przeznaczona do narysowania wykresu

w przypadku wartości typu `NoneType` utworzona zostanie nowa kanwa, na której narysowany zostanie wykres.

****kwargs**

opcjonalne argumenty zależne od wybranej miary podobieństwa rankingów

Miary podobieństwa rankingów *Percentage of Overlap* i *Correspondence at the Top* wymagają podania wartości parametru `k` (`int`) ($k > 0$, $k < p$, gdzie p to liczność siatki hiperparametrów). Miara podobieństwa rankingów *Overlap Score* wymaga podania wartości parametru `alpha` (`float`) ($\alpha > 0$).

Zwracane wartości:**obiekt klasy `matplotlib.axes.Axes`**

kanwa zawierająca narysowany wykres

`TransferSpeedAnalyser.__init__`

konstruktor klasy

Klasa `TransferSpeedAnalyser` odpowiada za analizę szybkości transferu hiperparametrów. Symulowana jest optymalizacja dla zadania predykcyjnego, określonego w lewej ramce danych, na podstawie bazy doświadczeń składającej się z zadań predykcyjnych, określonych w prawej ramce danych. Obie ramki danych muszą zawierać wyniki odnoszące się do tego samego algorytmu uczenia maszynowego (dopuszczalne są różne jego wersje), tej samej siatki hiperparametrów oraz tej samej numerycznej metryki jakości modeli.

Przyjmowane argumenty:

`left` (`pandas.DataFrame`)

lewa ramka danych z wynikami, ramka docelowa

Struktura ramki danych musi być zgodna z używaną przez moduł składowania. Docelowa ramka danych musi zawierać wyniki dla wyłącznie jednego zadania predykcyjnego.

right (`pandas.DataFrame`)

prawa ramka danych z wynikami, ramka źródłowa

Struktura ramki danych musi być zgodna z używaną przez moduł składowania. Źródłowa ramka danych może zawierać wyniki dla wielu zadań predykcyjnych.

ascending (`bool`)

charakterystyka metryki jakości modeli

Wartość fałszywa oznacza, że większa wartość metryki świadczy o lepszym modelu, a wartość prawdziwa przeciwnie.

method (`str`)

strategia analityczna - metoda konstrukcji portfolio

Możliwe wartości to `'simple'` (*Simple*) oraz `'asmfo'` (*Average SMFO*).

label (`str`)

etykieta analizatora

W przypadku wizualizacji analizatora jego etykieta używana jest w legendzie wykresu.

Zwracane wartości:

obiekt klasy `TransferSpeedAnalyser`

instancja analizatora zainicjalizowana danymi

`TransferSpeedAnalyser.calculate`

metoda służąca do obliczenia wyników analizy

Przyjmowane argumenty:

iteration_limit (`int`, `NoneType`) = `None`

limit liczby iteracji

Wartość typu `NoneType` oznacza symulację optymalizacji dla całego portfolio zestawów hiperparametrów.

****kwargs**

opcjonalne argumenty zależne od wybranej metody konstrukcji portfolio

Metoda konstrukcji portfolio (*Simple*) wymaga podania wartości parametru `scale` (`bool`).

Zwracane wartości:

obiekt klasy `pandas.DataFrame`

Indeks ramki danych stanowią kolejne iteracje symulacji procesu optymalizacji. Kolumna `set_number` zawiera numer porządkowy aktualnie rozważanego zestawu hiperparametrów, kolumna `value` odpowiadającą mu wartość metryki jakości modeli dla rozważanego w optymalizacji zadania predykcyjnego, a kolumna `best_value` dotychczas najlepszą wartość metryki.

`TransferSpeedAnalyser.plot`

metoda służąca do wizualizacji wyników analizy

Przyjmowane argumenty:

analysers

opcjonalne analizatory do wspólnej wizualizacji

Możliwa jest wizualizacja kilku analizatorów tego samego typu na jednym wykresie. Wszystkie analizatory muszą symulować optymalizację tego samego zadania predykcyjnego oraz bazować na tych samych definicjach (algorytm uczenia maszynowego, siatka hiperparametrów, metryka jakości modeli). W tym celu należy przekazać słownik postaci `{analizator: {argument: wartość}}` (argumenty zostaną przekazane do metody `calculate` odpowiednich analizatorów).

ax (`matplotlib.axes.Axes`, `NoneType`) = `None`

opcjonalna kanwa przeznaczona do narysowania wykresu

w przypadku wartości typu `NoneType` utworzona zostanie nowa kanwa, na której narysowany zostanie wykres.

iteration_limit (`int`, `NoneType`) = `None`

limit liczby iteracji

Wartość typu `NoneType` oznacza symulację optymalizacji dla całego portfolio zestawów hiperparametrów.

random_expectation (`bool`) = `False`

rysowanie wartości oczekiwanej losowego przeglądania konfiguracji

****kwargs**

opcjonalne argumenty zależne od wybranej metody konstrukcji portfolio

Metoda konstrukcji portfolio (*Simple*) wymaga podania wartości parametru `scale` (`bool`).

Zwracane wartości:

obiekt klasy `matplotlib.axes.Axes`

kanwa zawierająca narysowany wykres

AverageNormalizedErrorAnalyser.__init__

konstruktor klasy

Klasa `AverageNormalizedErrorAnalyser` odpowiada za analizę średniego znormalizowanego błędu strojenia. Symulowana jest optymalizacja dla zadań predykcyjnych, określonych w lewej ramce danych, na podstawie bazy doświadczeń składającej się z zadań predykcyjnych, określonych w prawej ramce danych. Obie ramki danych muszą zawierać wyniki odnoszące się do tego samego algorytmu uczenia maszynowego (dopuszczalne są różne jego wersje), tej samej siatki hiperparametrów oraz tej samej numerycznej metryki jakości modeli.

Przyjmowane argumenty:

left (`pandas.DataFrame`)

lewa ramka danych z wynikami, ramka docelowa

Struktura ramki danych musi być zgodna z używaną przez moduł składowania.

right (`pandas.DataFrame`)

prawa ramka danych z wynikami, ramka źródłowa

Struktura ramki danych musi być zgodna z używaną przez moduł składowania.

ascending (bool)

charakterystyka metryki jakości modeli

Wartość fałszywa oznacza, że większa wartość metryki świadczy o lepszym modelu, a wartość prawdziwa przeciwnie.

method (str)

strategia analityczna - metoda konstrukcji portfolio

Możliwe wartości to **'simple'** (*Simple*) oraz **'asmfo'** (*Average SMFO*).

label (str)

etykieta analizatora

W przypadku wizualizacji analizatora jego etykieta używana jest w legendzie wykresu.

Zwracane wartości:

obiekt klasy `AverageNormalizedErrorAnalyser`

instancja analizatora zainicjalizowana danymi

`AverageNormalizedErrorAnalyser.calculate`

Przyjmowane argumenty:

iteration_limit (int, NoneType) = None

limit liczby iteracji

Wartość typu `NoneType` oznacza symulację optymalizacji dla całego portfolio zestawów hiperparametrów.

****kwargs**

opcjonalne argumenty zależne od wybranej metody konstrukcji portfolio

Metoda konstrukcji portfolio (*Simple*) wymaga podania wartości parametru **scale** (bool).

Zwracane wartości:

obiekt klasy `pandas.DataFrame`

Indeks ramki danych stanowią kolejne iteracje symulacji procesu optymalizacji, a kolumna **ane** zawiera odpowiadające im wartości średniego znormalizowanego błędu strojenia.

`AverageNormalizedErrorAnalyser.plot`

Przyjmowane argumenty:

analysers

opcjonalne analizatory do wspólnej wizualizacji

Możliwa jest wizualizacja kilku analizatorów tego samego typu na jednym wykresie. Wszystkie analizatory muszą symulować optymalizację tego samego zadania predykcyjnego oraz bazować na tych samych definicjach (algorytm uczenia maszynowego, siatka hiperparametrów, metryka jakości modeli). W tym celu należy przekazać słownik postaci `{analizator: {argument: wartość}}` (argumenty zostaną przekazane do metody `calculate` odpowiednich analizatorów).

`ax (matplotlib.axes.Axes, NoneType) = None`

opcjonalna kanwa przeznaczona do narysowania wykresu

w przypadku wartości typu `NoneType` utworzona zostanie nowa kanwa, na której narysowany zostanie wykres.

`iteration_limit (int, NoneType) = None`

limit liczby iteracji

Wartość typu `NoneType` oznacza symulację optymalizacji dla całego portfolio zestawów hiperparametrów.

`random_expectation (bool) = False`

rysowanie wartości oczekiwanej losowego przeglądania konfiguracji

****kwargs**

opcjonalne argumenty zależne od wybranej metody konstrukcji portfolio

Metoda konstrukcji portfolio (*Simple*) wymaga podania wartości parametru `scale (bool)`.

Zwracane wartości:

obiekt klasy `matplotlib.axes.Axes`

kanwa zawierająca narysowany wykres