Faculty of Mathematics and Information Sciences

WARSAW UNIVERSITY OF TECHNOLOGY

Automatyczne Uczenie Maszynowe - Projekt 1

Tunowalność Algorytmów ML

Igor Kołodziej
Hubert Kowalski
Julia Przybytniowska

Warszawa, Listopad 2024

1 Wstęp

Celem niniejszego eksperymentu była analiza tunowalności hiperparametrów 3 algorytmów uczenia maszynowego na 4 zbiorach danych, zgodnie z definicjami zawartymi w pracy Tunability: Importance of Hyperparameters of Machine Learning Algorithms [2]. Celem pobocznym było również wyznaczenie domyślnych zestawów hiperparametrów, które średnio na różnych zbiorach danych zapewniają poprawę wyników modelu względem wartości domyślnych biblioteki *scikit-learn*. W ramach eksperymentu zastosowano dwie techniki optymalizacji hiperparametrów: **Random Search** oraz **Tree-Structured Parzen Estimator** [3].

W dokumencie zawarto opis przeprowadzonego doświadczenia oraz analizę uzyskanych wyników. Szczegóły implementacyjne oraz sposób uruchomienia kodu zawarto w pliku *README.md*.

2 Wybrane zbiory danych i ich przygotowanie

W eksperymencie wykorzystano cztery zbiory danych pobrane z platformy OpenML:

- **Credit-g** (ID 31): shape = (1000, 21)
- **Diabetes** (ID 37): shape = (768, 9)
- **Shrutime** (ID 45062): shape = (10000, 11)
- **Wine** (ID 43980): shape = (2554, 12)

Zbiory te nie zawierały brakujących wartości. Zmienne celu zostały przetworzone za pomocą kodowania `LabelEncoder`. Cechy numeryczne zostały przetransformowane poprzez standardyzację przy użyciu `StandardScaler`. Cechy kategoryczne zostały dodatkowo zakodowane metodą `OneHotEncoder` z usunięciem pierwszej kategorii, aby zapobiec kolinearyzmowi. Proces ten zrealizowano za pomocą `ColumnTransformer`, który obsługiwał równocześnie cechy numeryczne i kategoryczne. W celu uniknięcia zjawiska przeuczenia modelu, podczas ewaluacji zastosowano **kroswalidację 5 - krotną**.

3 Wybrane Algorytmy ML i techniki losowania punktów

Do przeprowadzenia analizy wybraliśmy następujące algorytmy:

- **Logistic Regression** (penalizacja *'elasticnet'*)
- **Extra Trees Classifier**
- **XGBoost Classifier**

Na każdym z wybranych algorytmów przetestowaliśmy dwie metody losowania punktów: **Random Search** oraz **Tree-Structured Parzen Estimator (TPE)**, oparty o metody bayesowskie. Dla każdej z metod wykonano różną liczbę iteracji: dla Random Search przeprowadzono 300 prób, a dla TPE po 100 prób zaczynając od 3 punktów startowych, zatem łącznie również 300 iteracji. Wykorzystano implementacje tych algorytmów z biblioteki `Optuna` [1], która umożliwia zdefiniowanie własnej funkcji celu `objective()`, stanowiącej przedmiot optymalizacji. Podczas eksperymentów testowano różne metryki, jednak ostatecznie zdecydowano się wykorzystać funkcję **AUC**.

Dzięki dostarczeniu globalnego ziarna RANDOM_SEED zapewniono stałość siatki hiperparametrów i powtarzalność wywołań.

4 Wybrane siatki hiperparametrów

Poniżej przedstawiono siatki hiperparametrów dla wybranych algorytmów uczenia maszynowego. Wartości zostały dobrane w oparciu o literaturę [2], czynnik czasu obliczeń oraz w oparciu o własne liczne eksperymenty na badanych zbiorach danych, których wyników nie zamieszczono w tym raporcie. Nadrzędnym celem

było dobranie hiperparametrów w taki sposób, aby z dużym prawdopodobieństwem ich zakres pokryły rzeczywiste optymalne wartości dla badanych zbiorów danych. Niektórym hiperparametrom przypisano stałe wartości, aby uniknąć ich wzajemnej zależności (przykładowo parametr `l1_ratio` wymaga użycia penalizacji `elasticnet`).

Algorytm	**Hyperparametr**	**Dolny zakres**	**Górny zakres**
ElasticNet	C	1e-4	1e4
	penalty	elasticnet	
	l1_ratio	1e-4	1.0
	class_weight	balanced	
	max_iter	1500	
	solver	saga	
Extra Trees	n_estimators	10	1000
	criterion	gini, entropy, log_loss	
	bootstrap	True	
	max_samples	0.5	1.0
	max_features	0.1	0.9
	min_samples_leaf	0.05	0.25
XGBoost	n_estimators	10	2000
	learning_rate	1e-4	0.4
	subsample	0.25	1.0
	booster	gbtree	
	max_depth	1	15
	min_child_weight	1	128
	colsample_bytree	0.2	1.0
	colsample_bylevel	0.2	1.0
	reg_alpha	1e-4	512.0
	reg_lambda	1e-3	1e3

Tabela 1: Siatki hiperparametrów dla wybranych algorytmów

5 Analiza wyników

5.1 Optymalne wartości domyślne

Optymalne wartości domyślne hiperparametrów wyznaczono θ^* wzorując się na definicji z artykułu [2]. Niech $AUC(\theta, \mathcal{D}_i)$ oznacza wartość AUC dla zbioru $\mathcal{D}_i$ i hiperparametrów θ.

$$\theta^* = \arg\max_{\theta} \frac{1}{n} \sum_{i=1}^{n} AUC(\theta, \mathcal{D}_i), \quad i = 1 \dots 4$$

Dzięki zastosowaniu pakietu Optuna, możliwa była optymalizacja średniej wartości AUC na wielu zbiorach na raz, zamiast optymalizacji brzegowej i późniejszego uśredniania. Zatem poszukiwania optymalnych wartości domyślnych dokonano zarówno metodą Random Search jak i TPE i otrzymano zbliżone wyniki. Do dalszej analizy wykorzystano θ^* obliczone za pomocą Random Search, w celu zachowania spójności z artykułem [2]. Dzięki wykonanej optymalizacji otrzymano wartości domyślne, które są lepsze od wartości domyślnych w pakietach *scikit-learn*. Tabela 2 przedstawia porównanie dla algorytmu Logistic Regression. Porównania dla pozostałych algorytmów umieszczono w sekcji 6.

Params	**C**	**l1_ratio**	**penalty**	**mean_AUC**
Package defaults	1.0	-	l2	0.8155
Our best params	0.1869	0.00049	elasticnet	0.8160

Tabela 2: Porównanie wartości domyślnych pakietowych i θ^* dla Logistic Regression

5.2 Wyniki optymalizacji i ich stabilność

W tabeli 3 porównano najlepsze wyniki uzyskane po wykonaniu optymalizacji metodami **RandomSearch** i **TPE**. Można zauważyć, że algorytm bayesowski zapewnił lepsze wyniki niezależnie od modelu.

Na wykresie 4 zawarto informację o stabilności optymalizacji dla różnych metod samplingu. Pomimo dużej liczby iteracji dla **Random Search**, można zaobserwować brak zbieżności. Liczba iteracji nie ma wpływu na moment znalezienia najlepszych hiperparametrów. Na wykresach dla optymalizacji bayesowskiej można zaobserwować wyraźnie tendencję do znajdowania coraz lepszych punktów hiperprzestrzeni w kolejnych iteracjach. Efekt ten widoczny jest również na konturowym wykresie aproksymującym funkcję celu przedstawionym na rysunku 6. Dzięki temu metoda **TPE** oparta o poszukiwanie bayesowskie jest bardziej efektywna.

Model	Random Search mean best AUC	Bayesian Search mean best AUC	Default
LogisticRegression	0.8165	0.8166	0.8159
ExtraTreesClassifier	0.7979	0.8047	0.7946
XGBClassifier	0.8433	0.8527	0.8403

Tabela 3: Porównanie najlepszych wyników AUC po przeprowadzeniu optymalizacji, uśrednionych ze względu na zbiory danych. Uwzględniono wyniki po optymalizacji z wykorzystaniem metody Random Search, TPE (Bayes Search) oraz dla wartości domyślnych.

5.3 Tunowalność algorytmów i bias sampling

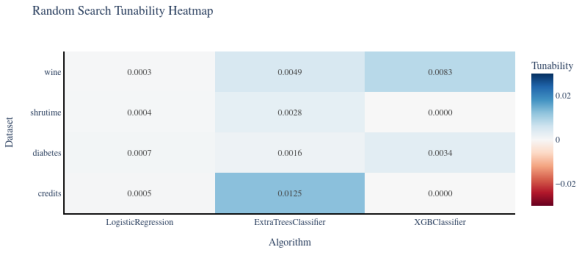
Tunowalność dla każdego modelu została obliczona wzorując się na definicjach z artykułu [2].

$$d_{\mathcal{S}}^{(j)} = AUC(\theta^{(j)*}, \mathcal{D}_j) - AUC(\theta^*, \mathcal{D}_j).$$

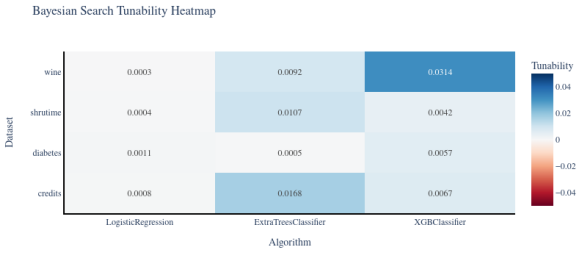
gdzie $j = 1 \dots 4$ zaś $\mathcal{S}$ oznacza wybrany optymalizator - RandomSearch lub TPE.
Wartość $d_{\mathcal{S}}^{(j)}$ wskazuje, jak bardzo można poprawić wydajność modelu dzięki dostrojeniu hiperparametrów w porównaniu do domyślnych wartości. Wyniki przedstawione w tabeli 4 wskazują, że każdy z algorytmów można dostroić przeprowadzając optymalizację hiperparemtrów, jednak lepsze efekty daje optymalizacja bayesowska. Dodatkowo niektóre algorytmy są bardziej podatne na optymalizację, w szczególności **XGBoost** okazał się najbardziej tunowalnym algorytmem, a **regresja logistyczna** najmniej.

Model	Random Search Tunability	Bayesian Search Tunability
LogisticRegression	0.0005 ± 0.0002	0.0007 ± 0.0003
ExtraTreesClassifier	0.0054 ± 0.0043	0.0093 ± 0.0058
XGBClassifier	0.0029 ± 0.0034	0.0120 ± 0.0112

Tabela 4: Średnia tunowalność dla RandomSearch i TPESearch z uwzględnieniem odchylenia standardowego. Jest to zatem średnia wartości Tunability dla danego algorytmu po wszystkich zbiorach danych.



Rysunek 1: Heatmapa przedstawiająca wartości tunability w zależności od algorytmu i zbioru danych dla wyników uzyskanych metodą Random Search



Rysunek 2: Heatmapa przedstawiająca wartości tunability w zależności od algorytmu i zbioru danych dla wyników uzyskanych metodą Random Search

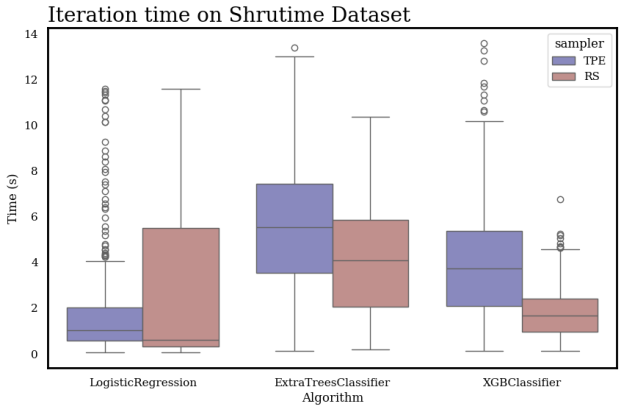
Samplowanie Bayesowskie osiąga wyższą medianę tunowalności dla wszystkich modeli, szczególnie w przypadku bardziej złożonych algorytmów, takich jak ExtraTreesClassifier i XGBClassifier. Różnice dla prostszego modelu Logistic Regression są minimalne, co sugeruje, że w takich przypadkach przeszukiwanie losowe może być wystarczające. Bezpośrednie porównanie dla obu metod samplowania punktów przedstawia wykres 5.

5.4 Analiza czasu przeszukiwania

Na podstawie wyników przedstawionych w Tabeli 5, można zauważyć istotne różnice w czasie strojenia pomiędzy metodami samplingu. Dla większości z testowanych algorytmów klasyfikacji, metoda RS charakteryzowała się krótszym czasem wykonania w porównaniu do metody TPE. Rysunek 3 przedstawiono wykres skrzynkowy, który ilustruje rozkład czasów pojedyńczej iteracji dla obu metod samplingu. Wykres ten potwierdza zróżnicowanie w czasach pomiędzy RS a TPE, co może być istotnym czynnikiem przy wyborze strategii w zależności od ograniczeń czasowych w projekcie.

Sampler	Algorytm	Czas strojenia (min)	AUC
RS	ExtraTreesClassifier	20.422524	0.8099
	LogisticRegression	15.317961	0.8328
	XGBClassifier	9.063305	0.8645
TPE	ExtraTreesClassifier	28.442875	0.8194
	LogisticRegression	9.626361	0.8328
	XGBClassifier	20.624290	0.8684

Tabela 5: Tabela przedstawiająca tade-off między czasem optymalizacji hiperparametrów a osiągniętym wynikiem AUC na zbiorze danych Shrutime.

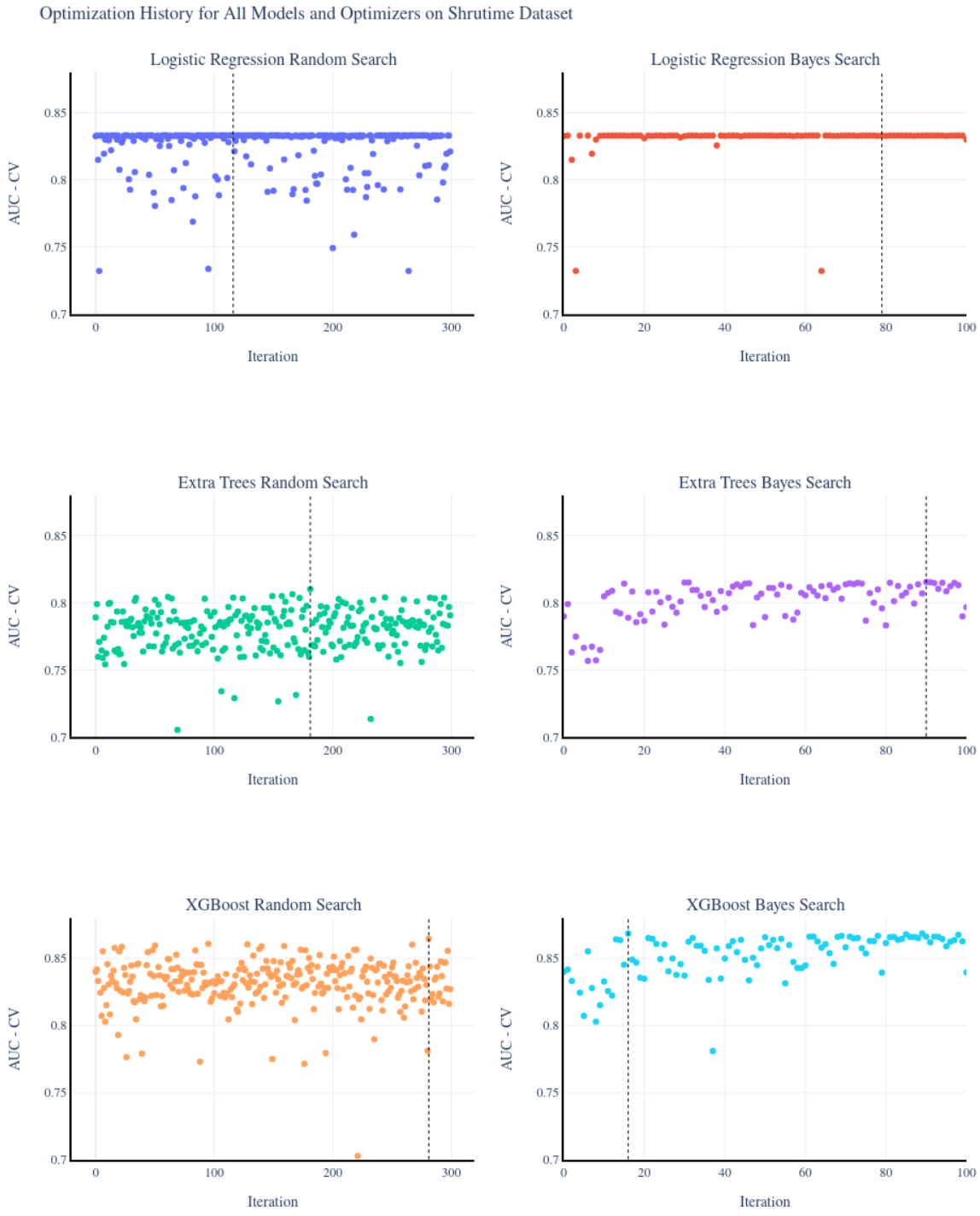


Rysunek 3: Czas iteracji na wybranym zbiorze danych.

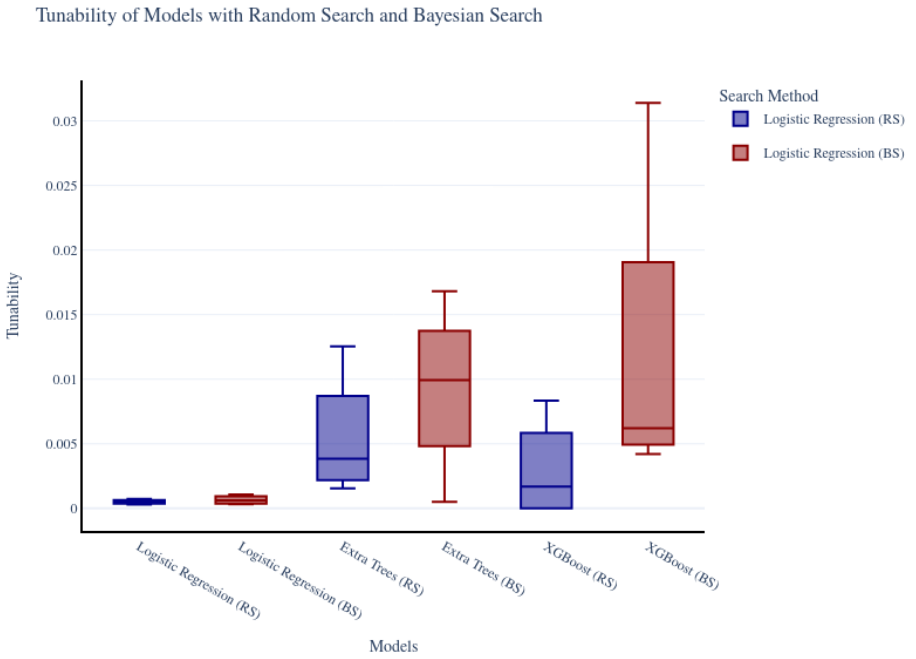
6 Dodatki

Wykresy przedstawione na rysunku 7 pokazują, że w przypadku **Logistic Regression (ElasticNet)** najistotniejszym parametrem jest `C`, który reguluje siłę regularyzacji. Dla **Extra Trees** kluczowy wpływ ma `min_samples_leaf`, kontrolujący minimalną liczbę obserwacji w liściu drzewa, podczas gdy `max_samples` ma mniejszy wpływ jednakże dalej wyróżniający się wśród reszty. W **XGBoost** natomiast najistotniejszym parametrem okazał się `reg_alpha`, który odpowiada za regularyzację L1.

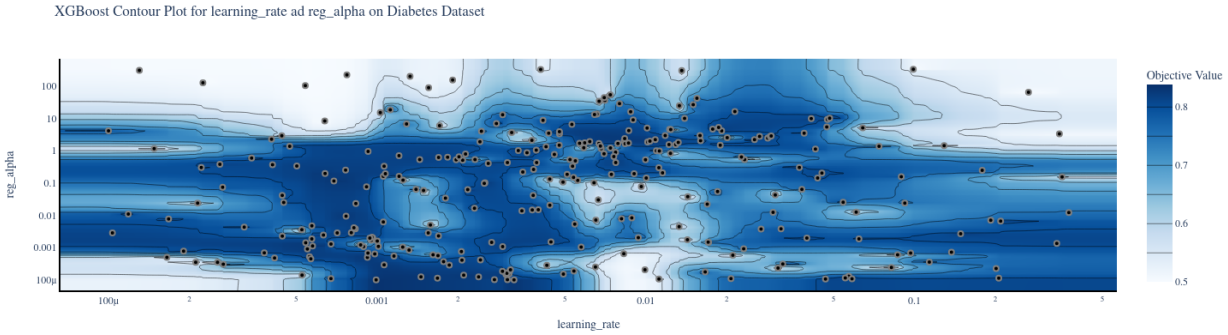
Wykres 6 przedstawia **aproksymację** funkcji celu (**AUC**) w zależności od 2 wybranych parametrów modelu XGBoost. Aproksymacja ta oparta jest o punkty w hiperprzestrzeni obliczane podczas optymalizacji bayesowskiej. Można zauważyć, że wyraźnie istnieją takie obszary, w których zagęszczenie punktów jest wyraźnie większe. W tych obszarach funkcja celu z dużym prawdopodobieństwem osiąga maksimum.



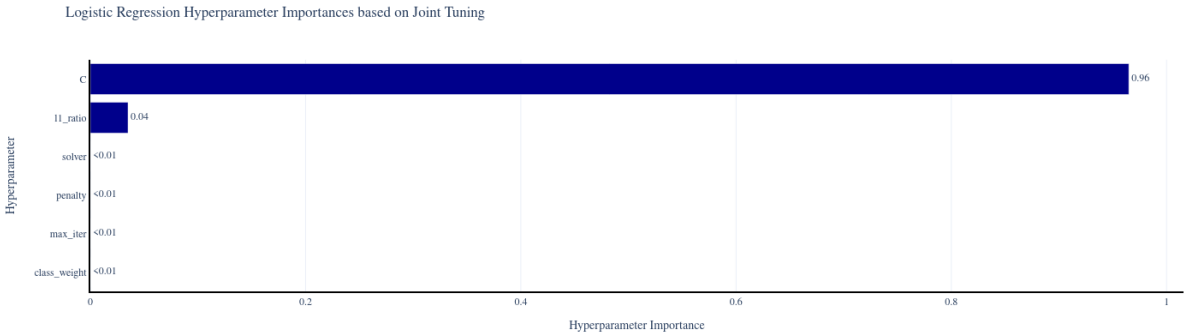
Rysunek 4: Porównanie osiągniętej wartości AUC w zależności od liczby iteracji przedstawione dla metody Random Search (kolumna lewa) oraz TPE (kolumna prawa). Porównanie to przedstawiono dla wybranego zbioru danych Shrutime. Pionowa przerywana linia na wykresie oznacza iterację, w której osiągnięto wartość maksymalną AUC.



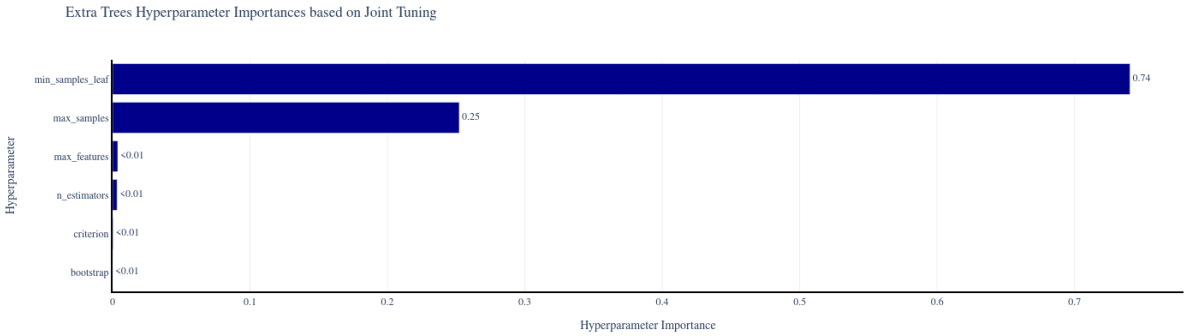
Rysunek 5: Wykres skrzynkowy przedstawiający bezpośrednie porównanie wyników Tunability uzyskanych za pomocą Random Search i Bayes Search.



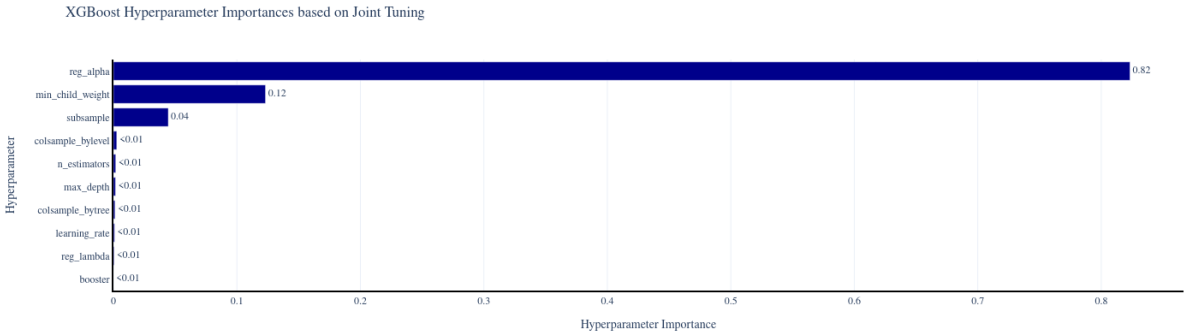
Rysunek 6: Wykres poziomicowy przeszukiwania wybranych hiperparametrów xgb. Przedstawia on aproksymację funkcji celu (AUC) w zależności od poziomu wartości parametrów learning_rate i reg_alpha. Wykres został utworzony na podstawie historii optymalizacji dla zbioru danych Diabetes.



(a) Istotność parametrów przy strojeniu regresji logistycznej mierzona wewnętrznym algorytmem biblioteki Optuna.



(b) Istotność parametrów przy strojeniu Extra Trees mierzona wewnętrznym algorytmem biblioteki Optuna.



(c) Istotność parametrów przy strojeniu XGBoost mierzona wewnętrznym algorytmem biblioteki Optuna.

Rysunek 7: Istotność parametrów mierzona wewnętrznym algorytmem biblioteki Optuna dla różnych modeli.

Literatura

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019.

[2] Philipp Probst, Anne-Laure Boulesteix, and Bernd Bischl. Tunability: Importance of hyperparameters of machine learning algorithms. *Journal of Machine Learning Research*, 20(53):1–32, 2019.

[3] Shuhei Watanabe. Tree-structured parzen estimator: Understanding its algorithm components and their roles for better empirical performance, 2023.