AutoML

Tunowalność algorytmów i hiperparametrów

Bartosz Kaczorowski

Damian Kąkol

Jakub Kępka

Listopad 2024

Spis treści

1. Wstęp

Celem niniejszego raportu jest analiza tunowalności hiperparametrów trzech wybranych algorytmów uczenia maszynowego (Elastic Net, XGBoost, Random Forest) na czterech zbiorach danych. Analiza obejmie wykorzystanie dwóch różnych metod doboru punktów (tzw. *samplingu*) w procesie optymalizacji hiperparametrów:

- przeszukiwanie losowe - ang. *Random Search*[4],
- optymalizację bayesowską - ang. *Bayes Optimization*[2].

W pracy zostanie przeanalizowana liczba iteracji każdej metody potrzebna, aby uzyskać stabilne wyniki, różnica w tunowalności między różnymi algorytmami oraz to, czy i jak dana technika doboru punktów wpływa na wyniki i wnioski dotyczące tunowalności.

Bieżący raport bazowany jest na pracy *"Tunability: Importance of Hyperparameters of Machine Learning Algorithms"*[1], gdzie została opisana między innymi tunowalność algorytmów i ich hiperparametrów.

2. Dane

Optymalizaja algorytmów i ich hiperparametrów została wykonana na czterech zbiorach danych dostępnych na platformie *OpenML*, każdy z nich dotyczy problemu klasyfikacji binarnej:

- **credit-g** (1000 instancji, 21 cech) - klasyfikacja osób na te o dobrej lub złej zdolności kredytowej;
- **diabetes** (768 instancji, 9 cech) - ocena czy dany pacjent wykazuje objawy cukrzycy zgodnie z kryteriami WHO. Badana populacja obejmuje osoby zamieszkałe w Phoenix, Arizona, USA;
- **spambase** (4601 instancji, 58 cech) - klasyfikacja e-maili na bazie treści, określająca czy są spamem. Kolekcja e-maili oznaczonych jako spam pochodziła od administratora poczty oraz osób zgłaszających spam, natomiast wiadomości niespamowe były zbierane z poczty służbowej i osobistej;
- **yeast** (2417 instancji, 117 cech) - zbiór danych dotyczący drożdży (opisany przez Elisseeffa i Westona, 2002), zawiera dane z mikro-macierzy ekspresji genów oraz profile filogenetyczne drożdży. W sumie gen może mieć przypisanych 14 różnych etykiet ze względu na 7 różnych, binarnych klas - na potrzeby naszej analizy wykorzystano tylko klasę o nazwie `Class1`.

3. Tunowalność algorytmów i hiperparametrów

Przed przejściem do analizy tunowalności, należy przytoczyć kilka definicji z artykułu[1] wspomnianym na wstępie.

Najlepszą konfiguracją hiperparametrów dla zbioru danych j nazywać będziemy wektor:

$$\theta^{(j)*} := \arg\min_{\theta \in \Theta} R^{(j)}(\theta). \tag{1}$$

Do oceny tunowalności algorytmów i ich hiperparametrów niezbędny będzie zestaw optymalnych hiperparametrów (zwanych też *defaultowymi*) dla m zbiorów danych, zdefiniowany w następujący sposób:

$$\theta^* := \arg\min_{\theta \in \Theta} g\left(R^{(1)}(\theta), \ldots, R^{(m)}(\theta)\right). \tag{2}$$

W definicjach (1) i (2) $R^{(j)}(\theta)$ oznacza metrykę mierzącą poprawność alogorytmu (dobrą miarą będzie tu -AUC [5]), zaś g oznacza funkcję agregującą, np. średnią. To czy miara $R^{(j)}(\theta)$ powinna być znormalizowana przed zagregowaniem, będzie przedmiotem analizy zawartej w dalszych rozdziałach.

Do mierzenia tunowalności algorytmu na zbiorze danych j wykorzystana zostanie miara określona w sposób następujący:

$$d^{(j)} := R^{(j)}(\theta^*) - R^{(j)}(\theta^{(j)*}), \quad \text{dla } j = 1, \ldots, m. \tag{3}$$

Podobnie jak w (3), by móc zmierzyć tunowalności pojedynczego i-tego hiperparametru definiujemy:

$$d_i^{(j)} := R^{(j)}(\theta^*) - R^{(j)}(\theta_i^{(j)*}), \quad \text{dla } j = 1, \ldots, m, \tag{4}$$

gdzie $\theta_i^{(j)*}$ oznacza najlepszą wartości hiperparametru na zbiorze danych j (podczas gdy pozostałe parametry pochodzą z optymalnego zestawu θ^*), zgodnie ze wzorem:

$$\theta_i^{(j)*} := \arg \min_{\theta \in \Theta, \theta_l = \theta_l^* \, \forall l \neq i} R^{(j)}(\theta)$$

Metryki (3) i (4) można następnie zagregować pomiędzy zbiorami poprzez wyliczenie średniej lub mediany opisanej różnicy, w celu uzyskania szerszego obrazu przeprowadzonych optymalizacji.

Ostatnią przydatną miarą może okazać się jeszcze względna tunowalność i-tego hiperparametru względem algorytmu na zbiorze j :

$$d_i^{(j),\text{rel}} = \frac{d_i^{(j)}}{d^{(j)}} \tag{5}$$

3.1. Elastic Net

Do zaimplementownania modelu *Elastic Net*[3] wykorzystano klasę `LogisticRegression` z pakietu *scikit-learn* z parametrem `penalty` ustawionym na `elasticnet` oraz `solver` na `saga`. Z innych istotnych hiperparametrów, przyjęto liczbę 200 jako maksymalną liczbę iteracji, a także ustawiono `class_weight` na `balanced` w celu wyrównania nierówności związanych z dysproporcjami klas.

Do przeszukiwania losowego zastosowano klasę `RandomizedSearchCV` z tego samego pakietu, zaś do optymalizacji bayesowskiej klasę `BayesSearchCV` z pakietu *scikit-optimize*. Obie metody wykorzystywały do pomiaru metrykę `roc_auc`, a także kroswalidację (ustawioną na 5). Co istotne, w obu przypadkach ustawiono ten sam `random_state`, dzięki temu metody korzystały z tej samej siatki hiperparametrów. Przeszukiwanie losowe w trakcie optymalizacji wykonywało 180 iteracji - optymalizacja bayesowska zaś 3 razy mniej.

Określenie przestrzeni hiperparametrów i optymalnego zestawu

Model *Elastic Net* posiada dwa główne hiperparametry, ich zakresy zaczerpnięto z artykułu[1]:

- **C** - rozkład logarytmiczno-jednostajny od 2^{-10} do 2^{10},
- **l1_ratio** - rozkład jednostajny od 0 do 1.

Do określenia optymalnego zestawu hiperparametrów wykorzystano przeszukiwanie losowe. Za funkcję g z definicji (2) przyjęto średnią, zaś $R^{(j)}(\theta)$ rozpatrzono na trzy sposoby - przyjmując zwykłe *-AUC*, znormalizowane i ustandaryzowane, co można zobaczyć na rysunku 1. Pierwsze dwa podejścia

wskazały na ten sam zestaw hiperparametrów, który w toku dalszego postępowania okazał się być lepszy (wykazywał mniejsze tunability według (3) i (4)).

Wyniki

Uzyskane wyniki tunowalności algorytmu oraz jego hiperparametrów zawarte w tabeli 1 pokazują, że zysk nie jest duży, zwłaszcza w kontekście tunowalności parametru `C`, co dobrze widać patrząc na względną tunowalności w tabeli 2. Dużo większą (nawet o kilka rzędów wielkości) tunowalność zaobserwować można dla parametru `l1`, czy wogóle całego zestawu. Niemniej, zarówno średnia jak i mediana obliczanej różnicy różni się dopiero na trzecim miejscu po przecinku.

Jeśli chodzi o różnice między samymi metodami *samplingu*, to trudno określić tu jednoznacznie zwycięzcę. To co napewno można stwierdzić, to znacznie dłuższy czas iteracji optymalizacji bayesowskiej, która w skrajnym przypadku trwała średnio nawet 11 razy dłużej niż iteracja przeszukiwania losowego. To jak liczba iteracji przekłada się na wynik zostało przeanalizowane w sekcji 4.

3.2. XGBoost

Model *XGBoost* był uruchamiany na 250 iteracjach dla metody *Random Search* oraz 20 dla *Bayes Optimization*.

Siatka parametrów

Zakres hiperparametrów do treningu był inspirowany artykułem do projektu. Odrzucone zostały parametry `eta`, `lambda` i `alpha` w celu przyśpieszenia procesowania, odrzucony został również dyskretny parametr `booster`.

Tunowalność algorytmu

Ogólnie rzecz biorąc *Random Search* częściej oferuje poprawe wydajności modelu niż optymalizacja bayesowska. Należy jednak zauważyć, że najwyższy wynik tunowalności osiągnęła optymalizacja bayesowska - około 0.0234.

Tunowalność hiperparametrów

Tunowalnośc poszczególnych hiperparametrów (tabela 7) wskazuje, że największe znaczenie mają parametry `subsample` i `min_child_weight`, które osiągają największe przyrosty efektywności na przestrzeni różnych zbiorów danych.

Optymalne zakresy hiperparametrów

Analiza kwantyli dla siatki hiperparametrów (tabela 8) sugeruje, że siatka startowa mogła zostać lepiej przygotowana, ponieważ niektóre kwantyle (np. `model__n_estimators`) wskazują (swoją niesymetrycznością), że górne ograniczenie mogłoby być większe.

3.3. Random Forest

W pracy wykorzystano algorytm `RandomForestClassifier`, który został zoptymalizowany przy użyciu technik *Random Search* oraz *Bayesian Optimization*. W obu metodach przyjęto określony zakres

parametrów, który pozwolił na dokładne dostrojenie modelu. W `RandomizedSearchCV` ustawiono liczbę iteracji na 50 oraz w `BayesSearchCV` ustawiono liczbę iteracji na 60.

Zakres liczby drzew w lesie `n_estimators` ustalono od 200 do 1740, co pozwalało na balans między dokładnością modelu a czasem jego trenowania. Głębokość każdego drzewa `max_depth` ograniczono w przedziale od 5 do 30, aby zapobiec przetrenowaniu modelu, a jednocześnie umożliwić mu uchwycenie istotnych wzorców w danych. Minimalna liczba próbek wymagana do podziału w węźle `min_samples_split` została określona na przedział od 2 do 20, aby zapewnić elastyczność w tworzeniu nowych podziałów. Minimalną liczbę próbek w liściu drzewa `min_samples_leaf` przyjęto od 1 do 10, co chroniło przed zbyt szczegółowymi podziałami.

Dodatkowo testowano różne podejścia do wyboru liczby cech branych pod uwagę przy każdym podziale `max_features`, obejmujące opcje `sqrt`, `log2` oraz rozważanie wszystkich cech. Sprawdzono także, czy lepsze wyniki przynosi próbkowanie bootstrapowe, włączając opcję `bootstrap` jako parametr, który mógł przyjmować wartości `True` lub `False`. Wreszcie, jako kryterium oceny jakości podziałów stosowano zarówno miarę `gini`, jak i `entropy`.

4. Zbieżności metod i charakterystyka przeszukiwania

Patrząc na wykresy 2 i 3 zbieżności metod przeszukiwania dotyczące eksperymentu 3.1 z modelem *Elastic Net*, można wskazać nie tylko optymalną liczbę iteracji, ale także pewną różnicę między samymi metodami. *Random Search* wymaga zdecydowanie więcej iteracji (około 100), niż *Bayes Optimization* (około 30), co jest jednak rekompensowane przez jego znaczenie szybsze działanie (patrz tabela 1). Warto jednak zauważyć też, że optymalizacja bayesowska potrafi bardzo szybko osiągnąć dobry wynik, który zostanie nieznacznie (lub wogóle) poprawiony w kolejnych iteracjach. Tę różnicę przedstawia też niejako wykres 4, który pokazuje, że przeszukiwanie losowe testuje więcej zestawów o niższej tunowalności od tej najlepszej (chodzi o szerokość wszystkich obserwacji). Wartości przeszukiwanych parametrów zależą jednak głównie od ustalnych zakresów, a nie metod, co pokazuje rysunek 5.

Model *XGBoost* potrzebował około 200 iteracji dla *Random Search* aby uzyskać stabilne wyniki, a około 50 iteracji wystarczyło aby osiągalny był błąd rzędu 0.01 (8 oraz 11). Optymalizacja bayesowska była uruchamiana na okło 20 iteracjach z powodu angażownia dużych zasobów, ale należy zauważyć że już kilkanaście iteracji pozwalało osiągąć błąd rzędu 0.01.

Tabele

Szukane	mean(d_{RS})	med(d_{RS})	mean(d_{BO})	med(d_{BO})	mean(t_{it})	med(t_{it})
`C`	0.000018	0.000041	-0.000875	0.000041	8.903920	6.951483
`l1`	0.001472	0.002087	0.001693	0.002268	9.496289	8.589719
`C` i `l1`	0.001721	0.001759	0.002947	0.001435	11.054320	8.693008

Tabela 1: Średnie oraz mediany tunowalności hiperparametrów (4) i algorytmu (3) (szukane) oraz stosunku czasu iteracji metod, tj. $t_{it} = \frac{t_{BO}}{t_{RS}} \cdot \frac{n_{RS}}{n_{BO}}$, gdzie t to czas trwania optymalizacji, zaś n liczba iteracji.

Hiperparametr	mean(d^{rel}_{RS})	mean(d^{rel}_{BO})	med(d^{rel}_{RS})	med(d^{rel}_{BO})
`C`	0.010407	-0.296815	0.023350	0.028827
`l1`	0.855382	0.574536	1.186648	1.579783

Tabela 2: Średnie oraz mediany względnej tunowalności hiperparametrów (5).

Tabela 3: Siatka hiperparametrów modelu.

Hiperparametr	**Zakres wartości**
`model__n_estimators`	`randint(100, 1000)`
`model__learning_rate`	`uniform(0.01, 0.2)`
`model__max_depth`	`randint(1, 10)`
`model__subsample`	`uniform(0.1, 0.9)`
`model__min_child_weight`	`uniform(0, 7)`
`model__colsample_bytree`	`uniform(0.1, 0.9)`
`model__colsample_bylevel`	`uniform(0.1, 0.9)`

Dataset	Metoda Optymalizacji	
	Random Search	Bayes Search
diabeties (37)	0.01	0.023
creditg (31)	0.0071	-0.0083
spambase (44)	0.0016	0.0014
yeast (40597)	0.0013	-0.0074

Tabela 4: Tabela tunowalności dla *XGBoost*.

Dataset	Metoda Optymalizacji	
	Random Search	Bayes Search
diabeties (37)	0.00219	0.0021
creditg (31)	0.0006	0.00094
spambase (44)	0.0014	0.0010
yeast (40597)	0.0059	-0.0214

Tabela 5: Tabela tunowalności dla *RandomForest*.

Dataset	n_estimators	learning_rate	max_depth	subsample	min_child_weight	colsample_bytree	colsample_bylevel
diabetes (37)	0.0071	0.0017	0.0076	0.0101	0.0084	0.0028	0.0021
creditg (31)	-0.0013	0.0015	0.0022	0.0062	0.0065	0.0017	0.0008
spambase (44)	0.0010	0.0007	0.0007	0.0012	0.0017	0.0007	0.0007
yeast (40597)	-0.0237	-0.0276	-0.0227	-0.0217	-0.0195	-0.0188	-0.0187

Tabela 6: Tunowalność hiperparametrów dla metody *Random Search* dla *XGBoost*.

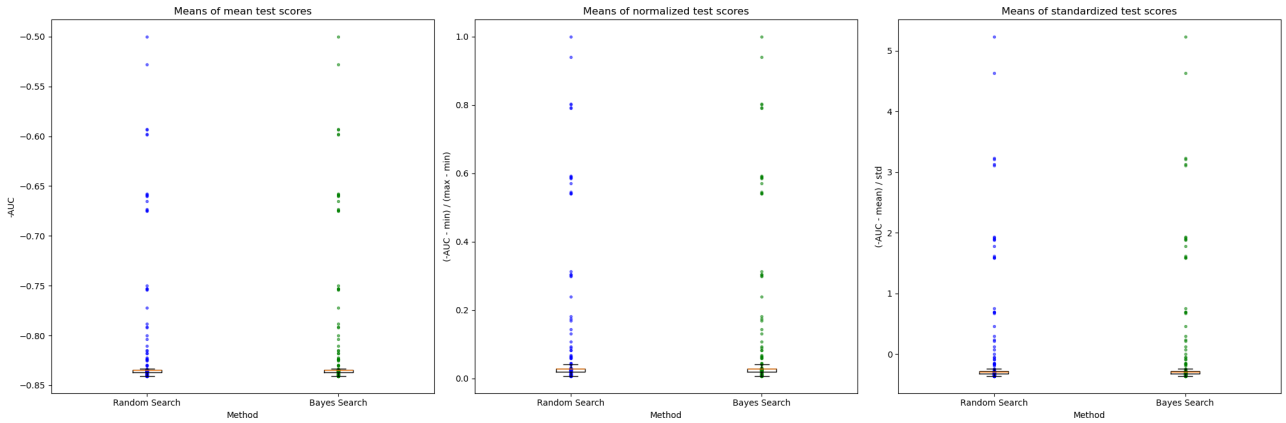
Dataset	n_estimators	learning_rate	max_depth	subsample	min_child_weight	colsample_bytree	colsample_bylevel
diabetes (37)	-0.0003	-0.0043	0.0019	0.0049	0.0037	-0.0068	-0.0070
creditg (31)	-0.0267	-0.0305	-0.0222	-0.0261	-0.0187	-0.0194	-0.0224
spambase (44)	0.0016	0.0016	0.0015	0.0021	0.0028	0.0017	0.0015
yeast (40597)	-0.0238	-0.0243	-0.0027	-0.0233	-0.0249	-0.0220	-0.0231

Tabela 7: Tunowalność hiperparametrów dla metody *Bayes Search* dla *XGBoost*.

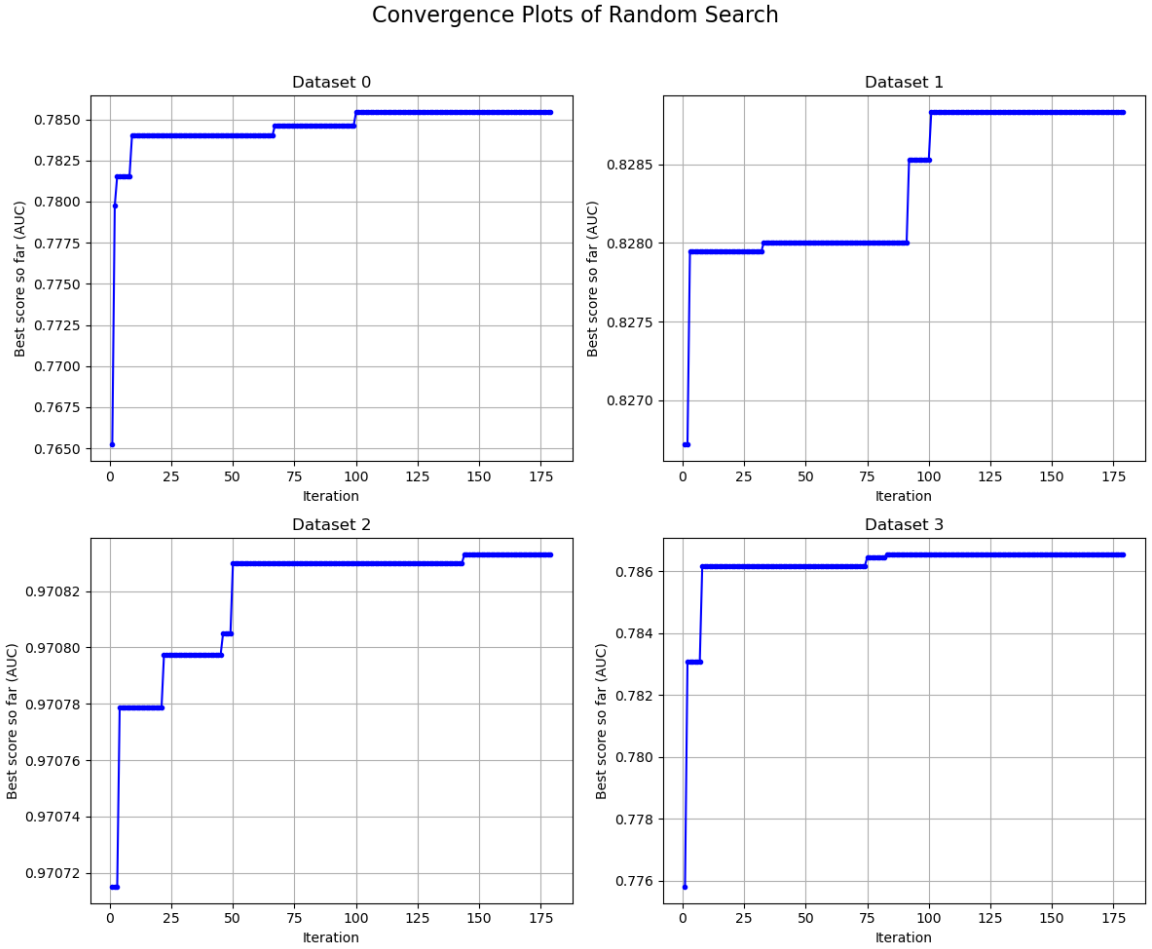
Hiperparametr	Zakres wartości	Kwantyl 5%	Kwantyl 95%	Optymalna wartość
`model__n_estimators`	`randint(100, 1000)`	103	960	809
`model__learning_rate`	`uniform(0.01, 0.2)`	0.010007	0.191541	0.010104
`model__max_depth`	`randint(1, 10)`	1.450000	10.0	9
`model__subsample`	`uniform(0.1, 0.9)`	0.100127	0.899934	0.559673
`model__min_child_weight`	`uniform(0, 7)`	0.025412	6.998327	2.133469
`model__colsample_bytree`	`uniform(0.1, 0.9)`	0.102904	0.875444	0.682921
`model__colsample_bylevel`	`uniform(0.1, 0.9)`	0.102904	0.864382	0.344919

Tabela 8: Siatka hiperparametrów modelu z zakresami kwantyli i optymalnymi wartościami.

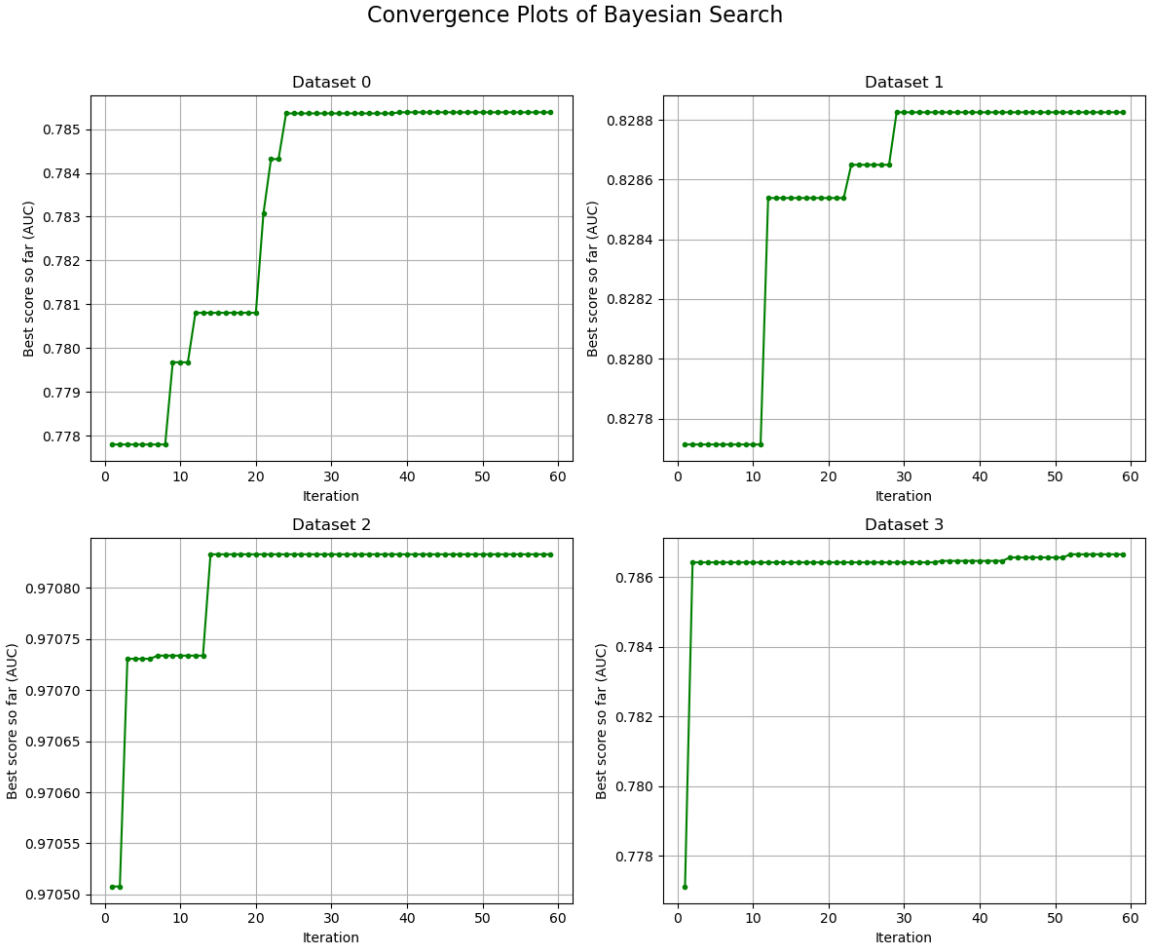
Rysunki



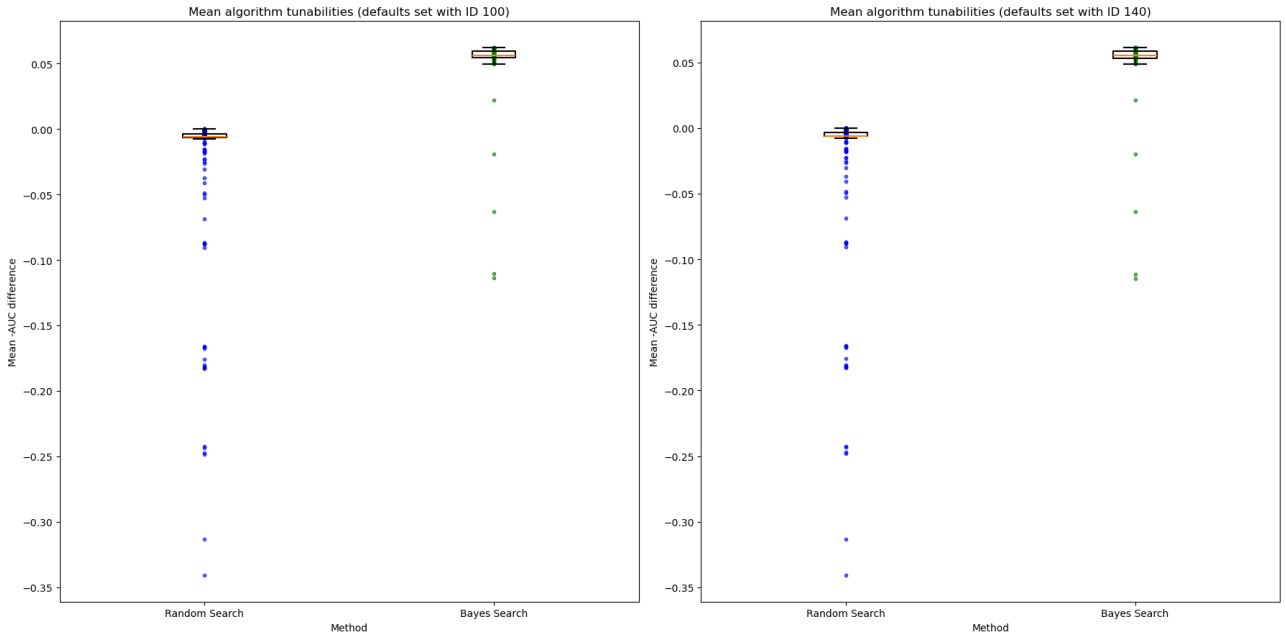
Rysunek 1: Uśrednione po zbiorach danych wartości $R^{(j)}(\theta)$ dla zestawów hiperparametrów. W pierwszym przypadku zastosowanano $-AUC$, w drugim miara została znormalizowana do przedziału [0,1], zaś w trzecim ustandaryzowana. Jak widać różnice w rozkładach nie są duże, zwłaszcza między pierwszym a drugim podejściem.



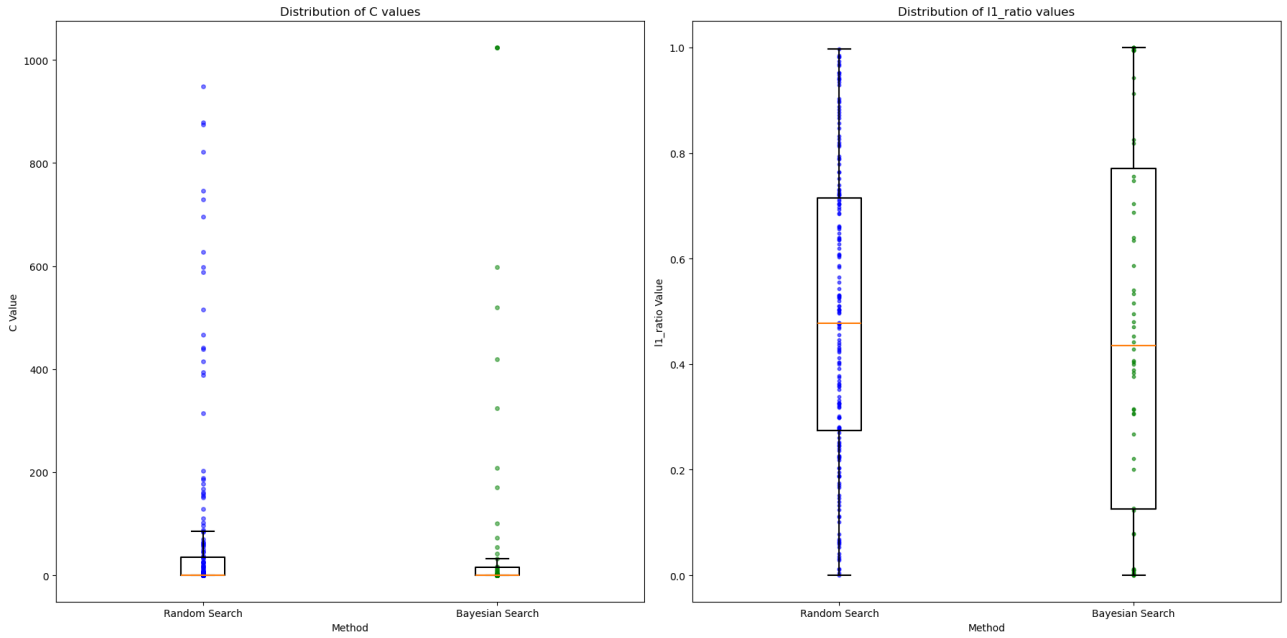
Rysunek 2: Wykresy zbieżności metody *Random Search* podczas tunowania algorytmu dla zbiorów danych opisanych w rozdziale 2. Jak widać już dla 100 iteracji w większości przypadków osiągana jest zbieżność, co jest niewielką liczbą biorąc pod uwagę szybkość metody.



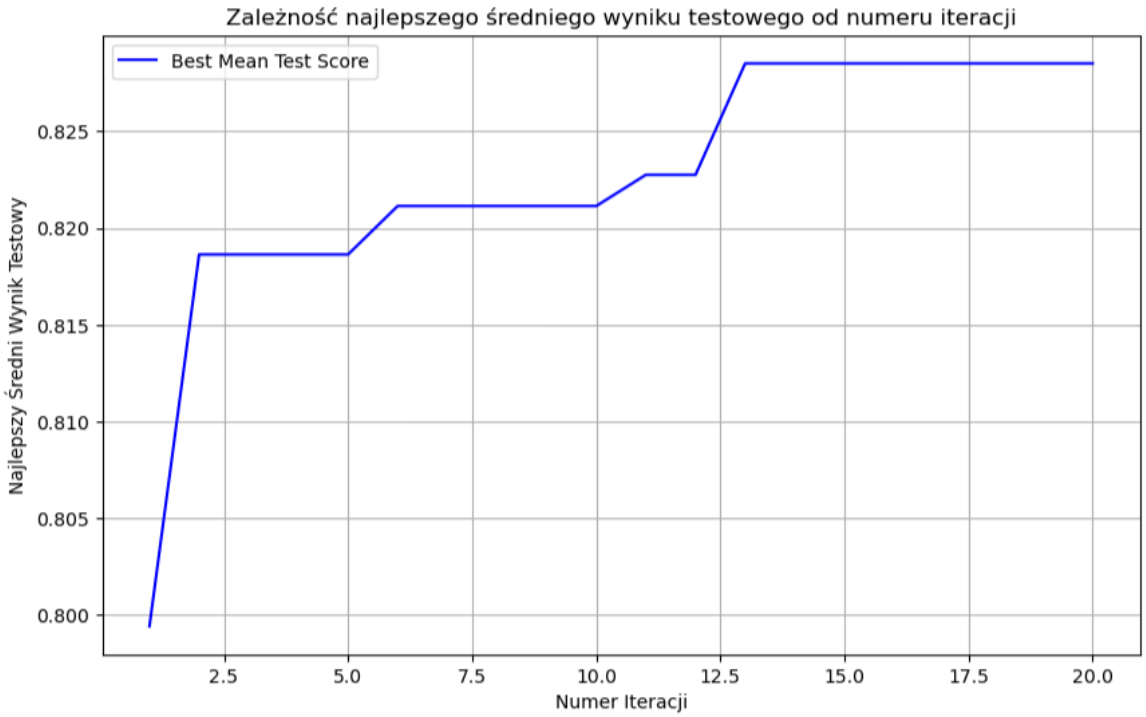
Rysunek 3: Wykresy zbieżności metody *Bayes Optimization* podczas tunowania algorytmu dla zbiorów danych opisanych w rozdziale 2. Jak widać metoda potrafi bardzo szybko osiągnąć zbieżność jeżeli wcześnie napotka ekstremum, jednak bezpieczną liczbą iteracji wydaje się być liczba 30.



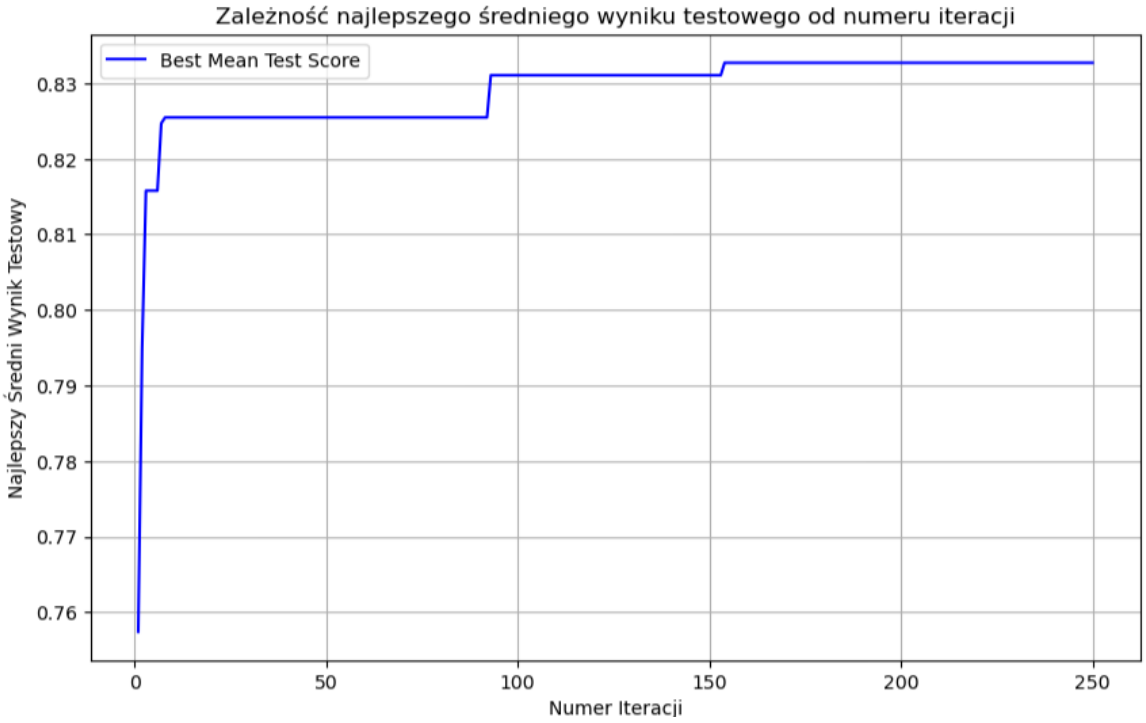
Rysunek 4: Średnia tunowalność algorytmu (3) obliczona dla zestawów hiperparametrów przeszukiwanych podczas jego tunowania. Niezależnie od zestawu optymalnego (którego wpływ widać jedynie poprzez nieznaczne przesunięcie wykresów na osi Y) można dostrzec istotną różnicę pomiędzy rozkładami dla metod *samplingu*.



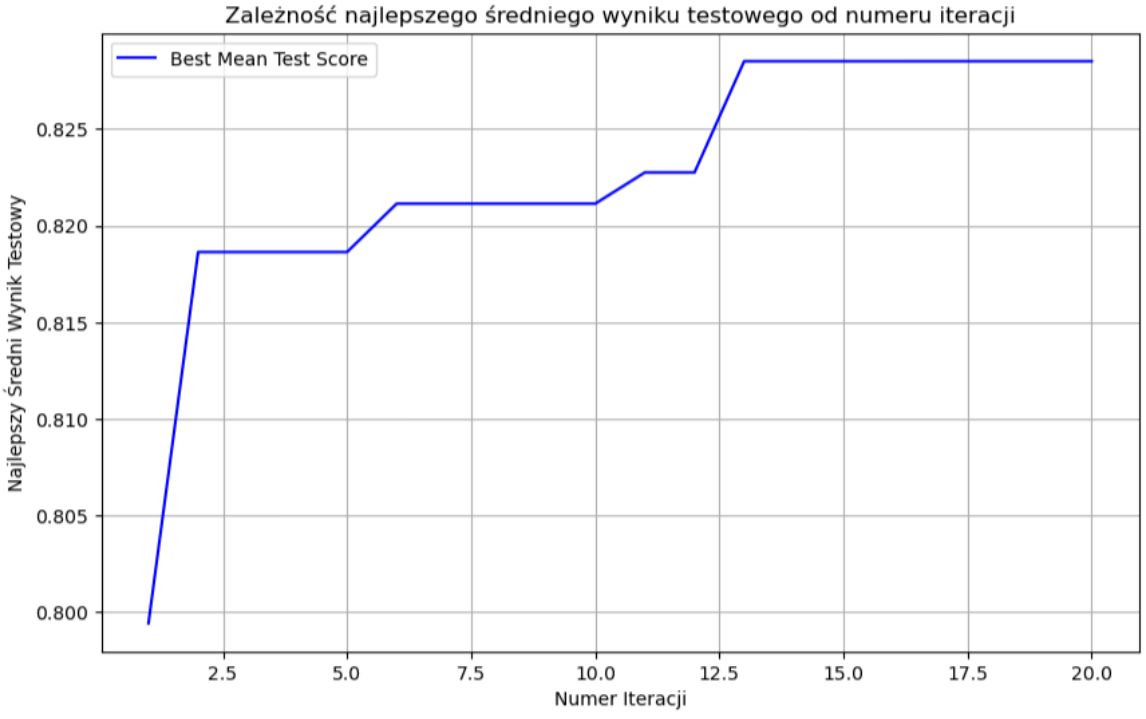
Rysunek 5: Dystrybucja przeszukiwanych wartości hiperparametrów podczas tunowania algorytmu dla każdej z metod *samplingu*. Jak widać różnice pomiędzy rozkładami według metod są nieduże, jednak same rozkłady hiperparametrów mocno różnią się między sobą, co najpradopodobniej wynika z ich zakresów przyjętych w rozdziale 3.1.



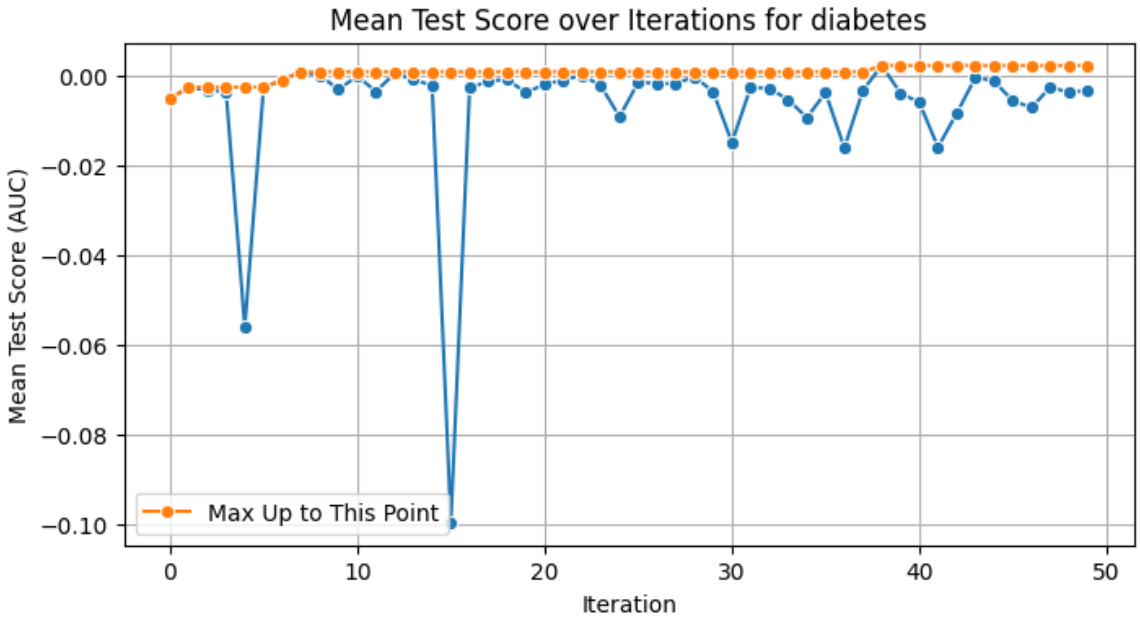
Rysunek 6: Zbieżność dla optymalizaji bayesowskiej dla *XGBoost*, dane **diabetes**.



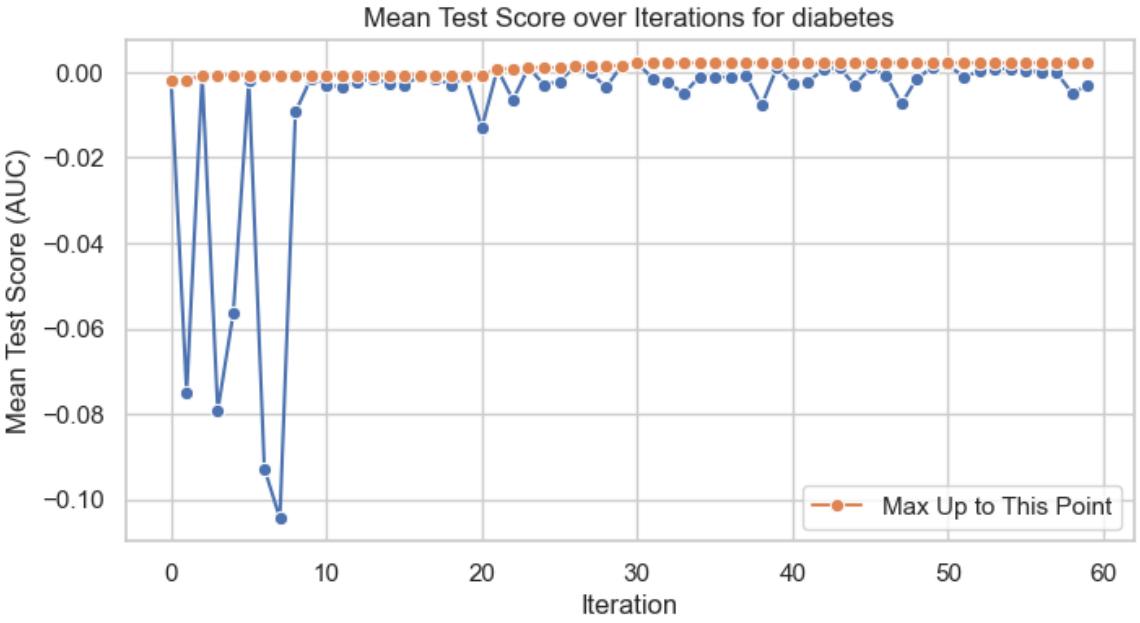
Rysunek 7: Zbieżność dla przeszukiwania losowego dla *XGBoost*, dane **diabetes**.



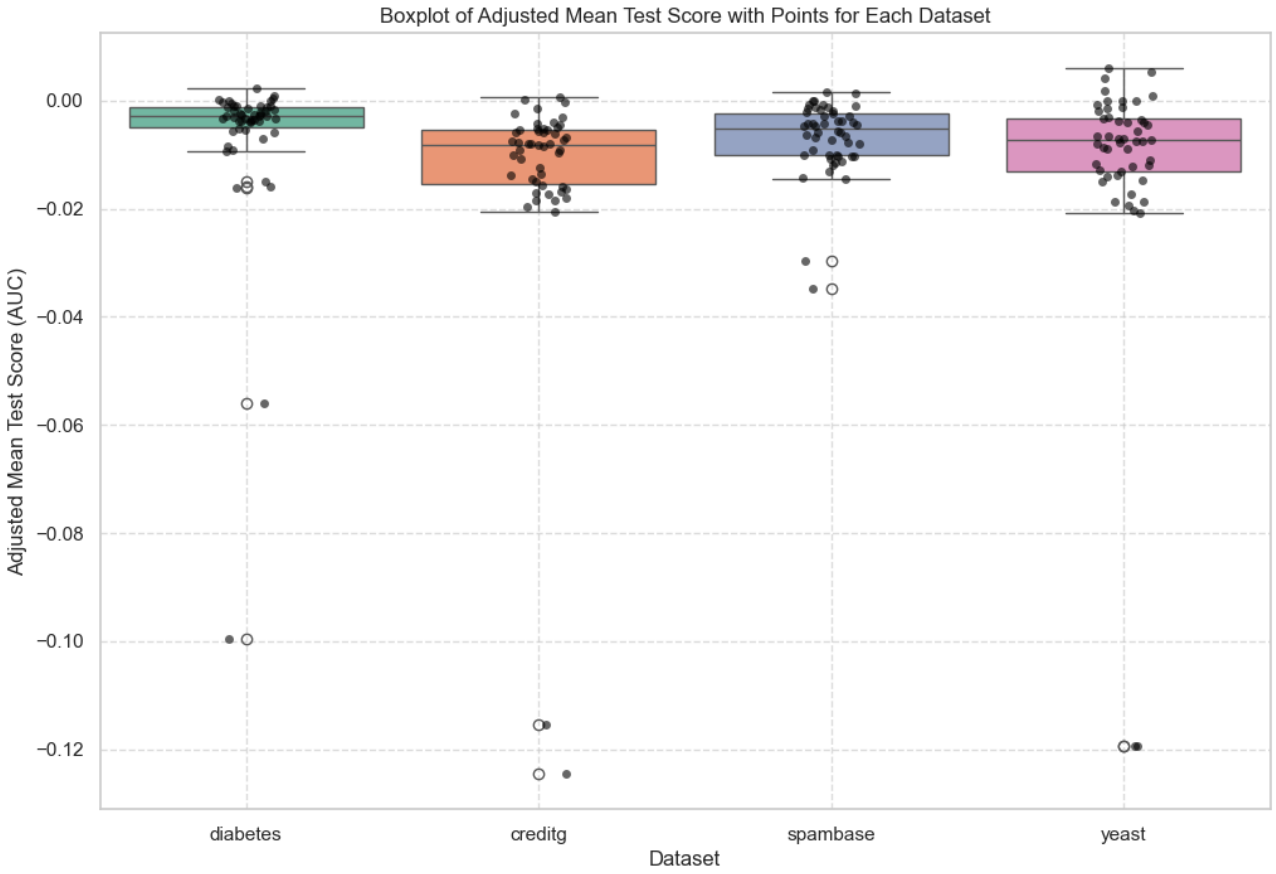
Rysunek 8: Zbieżność dla optymalizacji bayesowskiej na *XGBoost*, dane **diabetes**.



Rysunek 9: Zbieżność dla metody *Random Search* przy użyciu `RandomForestClassifier`, dane **diabetes**.



Rysunek 10: Zbieżność dla metody *Bayes Search* przy użyciu `RandomForestClassifier`, dane **diabetes**.



Rysunek 11: Boxplot dla różnych datasetów, metoda *Random Search* dla algorytmu `RandomForestClassifier`.

Literatura

[1] Philipp Probst, Bernd Bischl, and Anne-Laure Boulesteix. Tunability: Importance of hyperparameters of machine learning algorithms, 2018.

[2] Wikipedia contributors. Bayesian optimization. https://en.wikipedia.org/w/index.php?title=Bayesian_optimization&oldid=1250232772, 2024. [Online; accessed 15-November-2024].

[3] Wikipedia contributors. Elastic net regularization. https://en.wikipedia.org/w/index.php?title=Elastic_net_regularization&oldid=1251555251, 2024. [Online; accessed 15-November-2024].

[4] Wikipedia contributors. Random search. https://en.wikipedia.org/w/index.php?title=Random_search&oldid=1222133590, 2024. [Online; accessed 15-November-2024].

[5] Wikipedia contributors. Receiver operating characteristic. https://en.wikipedia.org/w/index.php?title=Receiver_operating_characteristic&oldid=1251138529, 2024. [Online; accessed 15-November-2024].