Raport 1

Julita Kulesza, Agata Osmałek

21 listopada 2023

1 Wprowadzenie

Celem naszego projektu było przeanalizowanie tunowalności — wpływu doboru hiperparametrów na wynik algorytmu — dla trzech wybranych algorytmów uczenia maszynowego, tj.

- `GradientBoostingClassifier` (GBC),
- `RandomForestClassifier` (RF),
- `Support Vector Classification` (SVC).

Każdy z tych algorytmów został wykorzystany w problemie klasyfikacji binarnej na zbiorach z platformy `openml`, *credit-g*, *Phoneme*, *diabetes* oraz *kc2*. Wszystkie cztery zbiory były podobnej wielkości, około 200 000 obserwacji, a także były zupełne, tzn. nie zawierały brakujących obserwacji.

2 Wyniki

Wyniki dotyczące tunowalności omówione w ostatniej sekcji są ściśle powiązane z zakresem hiperparametrów które rozważałyśmy, ale też sposobem ich losowania. W poniższych podpunktach opisujemy szczegóły obu aspektów naszych symulacji.

2.1 Zakres hiperparametrów

Dobór naszych modeli bazował na heurstyce, tzn. założeniu, że tunowalność algorytmu jest w jakimś stopniu skorelowana z liczbą jego hiperparametrów, które jako twórczynie modelu możemy modyfikować. Niemniej, wkład tych parametrów w wynik końcowy oczywiście nie musi być równomierny.

Powodem dla którego wybrałyśmy `GradientBoostingClassifier` oraz `RandomForest` jako nasze klasyfikatory była stosunkowo duża liczba hiperparametrów. Zacznijmy więc od omówienia siatek hiperparametrów, które rozważałyśmy w naszych symulacjach. Poniżej znajduje się siatka dla klasyfikatora `RandomForest`.

Tabela 1: Siatka hiperparametrów dla klasyfikatora `Random Forest`

Parametr	**Wartości**
`n_estimators`	100, 1000, 2000
`max_features`	'auto', 'sqrt'
`max_depth`	10, 100, None
`min_samples_split`	2, 10
`min_samples_leaf`	1, 4

Siatka dopuszcza różne liczby drzew w modelu, ograniczenie na głębokość drzewa, a także ogólną liczbę cech braną pod uwagę przy tworzeniu podziału, minimalną liczbę próbek w liściu i próbkę, jakiej potrzeba aby w liściu nastąpił wewnętrzny podział. Nie są to wszystkie hiperparametry dla powyższego

modelu, jednak są one łatwo interpretowalne, przez co da się do pewnego stopnia przewidzieć jak model zachowa się przy zmianie wartości jednego z tych parametrów przy jednoczesnym ustaleniu pozostałych. Przy doborze wartości starałyśmy się uwzględnić wartości domyślne, ale także takie, które są istotnie różne od domyślnych wynikających z implementacji klasyfikatorów.

Jako że `GradientBoostingClassifier` bazuje na drzewach losowych, część z wymienionych hiperprametrów uwzględniona jest też w siatce dla klasyfikatora `RandomForest`. Zasadniczą różnicą jest pojawienie się hiperparametru `learning_rate`, który odpowiada za tempo uczenia, karząc za błąd który dotychczas wystąpił.

Tabela 2: Siatka hiperparametrów dla klasyfikatora `GradientBoostingClassifier`

Parametr	**Wartości**
`n_estimators`	100, 200, 300, 500, 1000
`learning_rate`	0.01, 0.1, 0.25, 0.5, 1
`max_depth`	3, 4, 5, 6, 7, 8
`min_samples_split`	2, 5, 10, 15
`min_samples_leaf`	1, 2, 4, 10

Ostatnim z rozważanych przez nas klasyfikatorów jest `SVC`, który ma na celu wyznaczenie hiperpłaszczyzny rozdzielającej z maksymalnym marginesem pomiędzy klasami. Liczba dostępnych hiperparametrów dla tego klasyfikatora nie jest tak duża jak dla poprzednich dwóch, niemniej nie oznacza to, że taki algorytm z góry uznać trzeba za nietunowalny. Hiperarametry te dotyczą też przede wszystkim jądra, którego zadaniem jest przekształcenie danych. W naszych symulacjach rozważałyśmy poniższą siatkę.

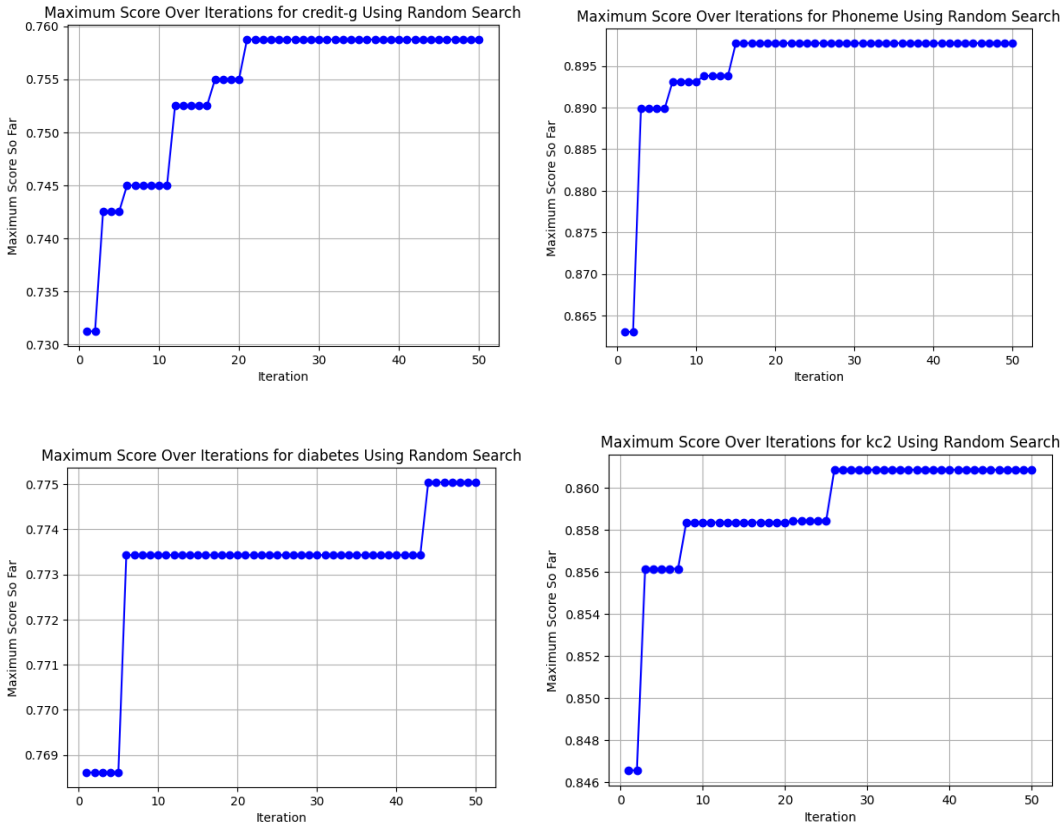
Tabela 3: Siatka hiperparametrów dla klasyfikatora `SVC`

Parametr	**Wartości**				
`C`	0.1	1	10	100	1000
`gamma`	1	0.1	0.01	0.001	0.0001
`kernel`	rbf	linear		poly	

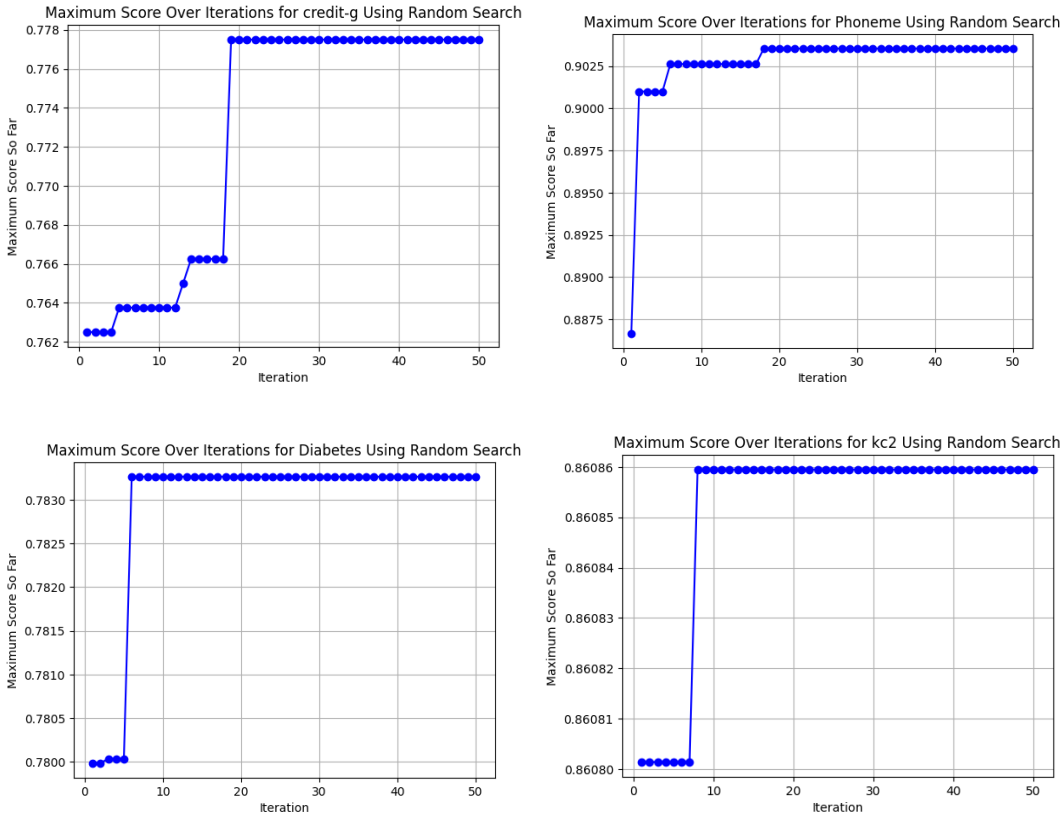
2.2 Sampling i optymalizacja

W momencie utworzenia siatki hiperparametrów kolejną istotną decyzją w projektowaniu symulacji jest wybór metody losowania z tejże siatki. Istnieje więcej niż jedna metoda bazująca na wyborze punktów z rozkładu jednostajnego, my jednak zdecydowałyśmy się na wybór metody `RandomizedSearchCV`.

W przypadku `RandomizedSearchCV`, hiperparametry są losowo wybierane z określonych zakresów wartości, a następnie oceniane jest ich wpływ na wydajność modelu. Proces ten jest powtarzany przez określoną liczbę iteracji lub do momentu osiągnięcia określonego kryterium zatrzymania. W naszym przypadku liczba tych iteracji dla metod `Random Forest` oraz `GradientBoostingClassifier` była zawsze taka sama, tj. 50, co wynikało z ograniczeń obliczeniowych. Ustaliłyśmy też ziarno generatora liczb pseudolosowych, aby mieć pewność, że modele trenowane będą na tych samych zestawach hiperparametrów w przypadku każdego zbioru. Jak widać na wykresach 1 oraz 2, ta stosunkowo niska liczba iteracji pozwalała otrzymać stabilne wyniki.



Rysunek 1: Wykresy pokazujące ile iteracji potrzebujemy żeby uzyskać stabilne wyniki optymalizacji przy użyciu `RandomizedSearchCV` dla klasyfikatora `GradientBoostingClassifier`



Rysunek 2: Wykresy pokazujące ile iteracji potrzebujemy żeby uzyskać stabilne wyniki optymalizacji przy użyciu `RandomizedSearchCV` dla klasyfikatora `RanomForest`

Celem tych działań było wyznaczenie doświadczalnie najlepszej kombinacji hiperparmetrów θ^* dla każdego z rozważanych algorytmów. Każdy z nich on potem wykorzystywany jako punkt odniesienia w podsumowaniu wyników tunowalności dla obu rozważanych metod samplingu.

Równoległym podejściem do problemu jest optymalizacja bayesowska (OB). OB to zaawansowana technika optymalizacji hiperparametrów, która wykorzystuje metody bayesowskie do skutecznego przeszukiwania przestrzeni hiperparametrów w celu znalezienia optymalnych ustawień dla danego modelu lub algorytmu. W tym celu wykorzystałyśmy funkcję `BayesSearchCV`, która tak jak `RandomizedSearchCV` nie wypróbowuje wszystkich kombinacji hiperparametrów, a z góry zadaną liczbę ich kombinacji. W przeciwieństwie do `RandomizedSearchCV`, hiperparametry są zadane w sposób ciągły i losowane są z góry zadanych rozkładów.

Tabela 4: Przestrzeń parametrów w OB dla klasyfikatora `RandomForest`

Parameter	**Range**
`max_depth`	(1, 100)
`n_estimators`	(1, 2000)
`max_features`	('auto', 'sqrt')
`min_samples_leaf`	(0.0 + a, 1 - a)
`min_samples_split`	(0.0 + a, 1 - a)

Tabela 5: Przestrzeń parametrów w OB dla klasyfikatora `GBC`

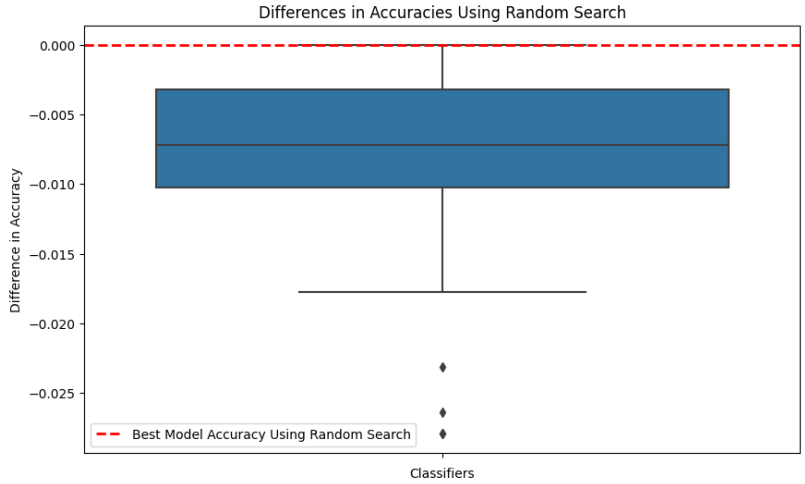
Parameter	**Range**	**Distribution**
`learning_rate`	(0.01, 0.1)	log-uniform
`n_estimators`	(50, 200)	-
`max_depth`	(3, 10)	-
`subsample`	(0.5, 1.0)	-
`min_samples_split`	(2, 10)	-
`min_samples_leaf`	(1, 10)	-

Rozkłady parametrów dla klasyfikatora `SVC` pomijamy jako że, pomimo znacznego zmniejszania liczby iteracj, nie udało nam się otrzymać wyników optymalizacji dla wszystkich zbiorów.

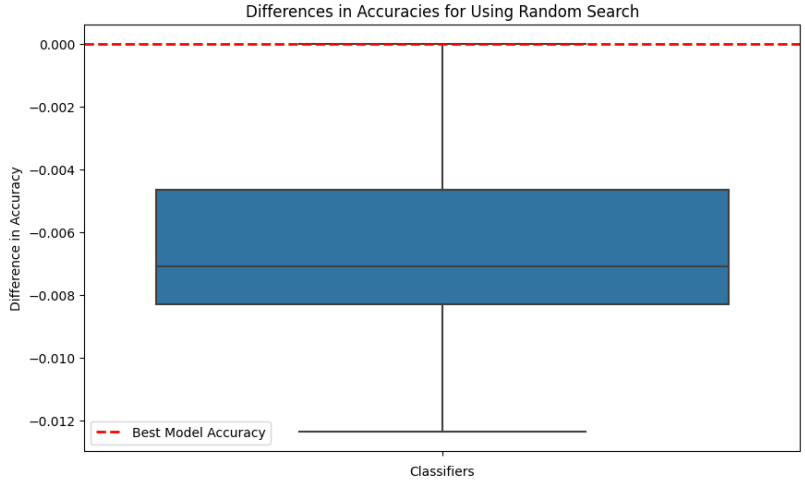
2.3 Tunowalność algorytmów

2.3.1 Przeszukiwania przy pomocy `RandomizedSearchCV`

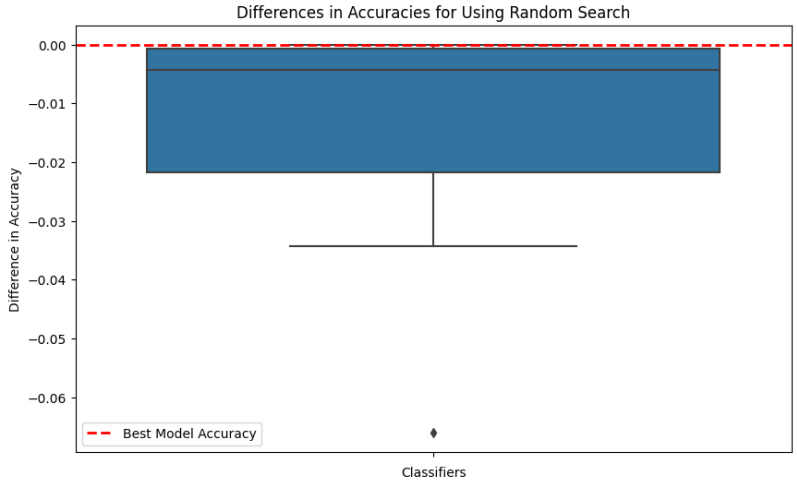
Wykresy 3, 4, oraz 5 służą porównaniu wyników dopasowania modeli dla czterech rozważanych zbiorów dla metody `RandomizedSearchCV` z wynikiem dla optymalnej kombinacji hiperparametrów θ^*. Powstały one poprzez uśrednienie dopasowania po czterech rozważanych zbiorach dla każdej ustalonej kombinacji hiperparametrów (które były takie same z racji ustalenia ziarna). Następnie od każdego w tych wyników został odjęty wynik dopasowania modelu dla θ^*. *Innymi słowy, dodatnie wartości na wykresie oznaczały, że otrzymany wynik był lepszy niż ten dla* θ^*.
Wykresy 3 oraz 4 przedstawiają wyniki dla $n_{iter} = 50$ interacji. W przypadku naszej zaproponowanej siatki niemożliwym okazało się uzyskanie wyników dla 50 iteracji dla klasyfikatora `SVC`. Prezentowany wykres powstał dla zaledwie $n_{iter} = 10$ interacji, niemniej samo to przeszukiwanie było i tak czasochłonne.



Rysunek 3: Różnice w średniej dokładności dla θ^* wybranej przy pomocy `RandomizedSearchCV` oraz pozostałych średnich dokładności dla algorytmu `GBC`



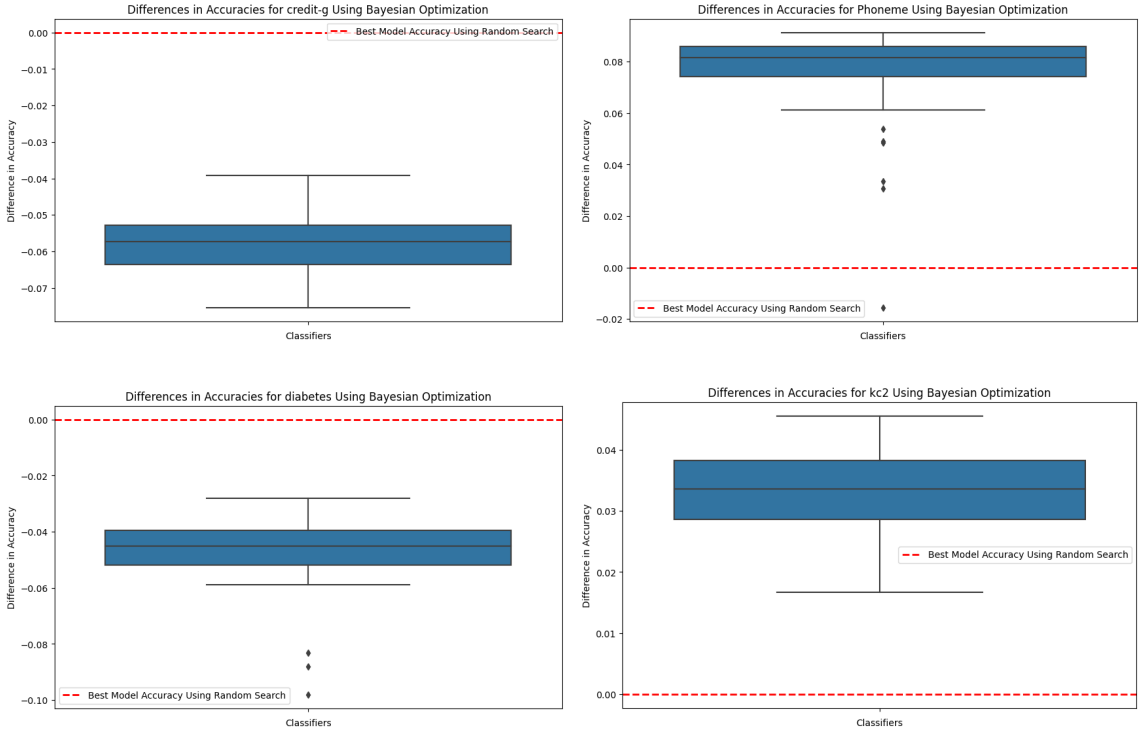
Rysunek 4: Różnice w średniej dokładności dla θ^* wybranej przy pomocy `RandomizedSearchCV` oraz pozostałych średnich dokładności dla algorytmu `Random Forest`



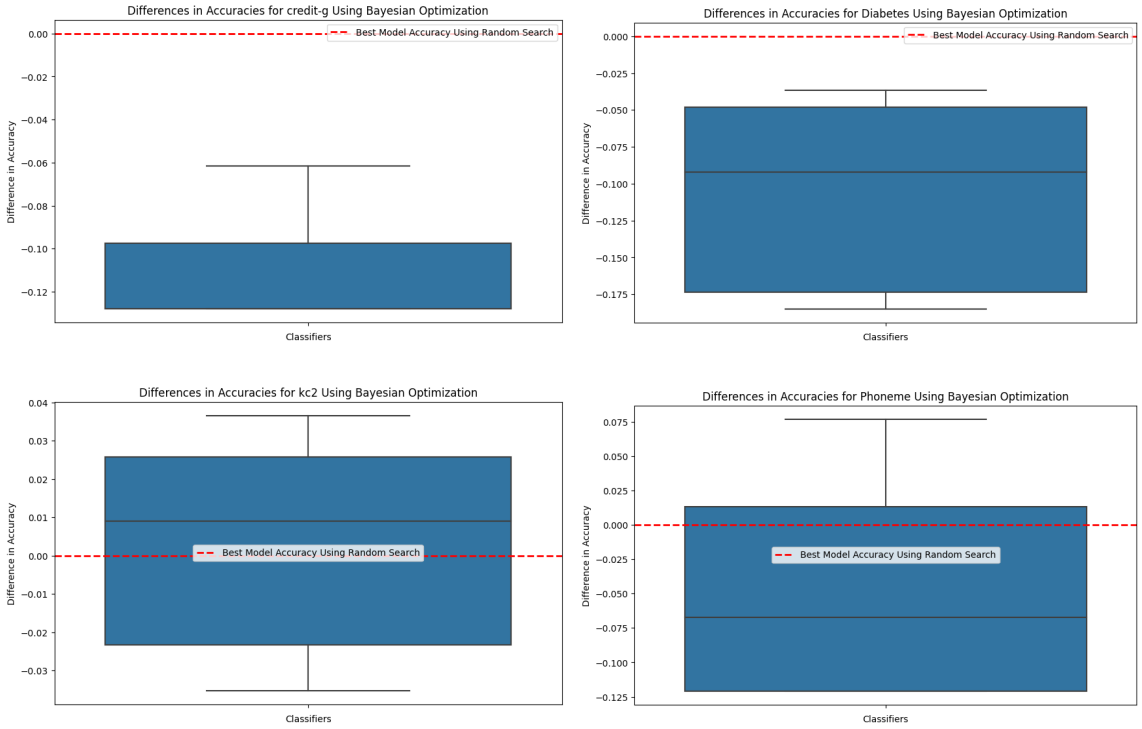
Rysunek 5: Różnice w średniej dokładności dla θ^* wybranej przy pomocy `RandomizedSearchCV` oraz pozostałych średnich dokładności dla algorytmu `SVC`

2.3.2 Przeszukiwania przy pomocy `BayesSearchCV`

Wykresy 6 oraz 7 przedstawiają porównanie wyników w kolejnych iteracjach optymalizacji bayesowskiej od których odjęłyśmy wynik dopasowania dla optymalnej θ^* otrzymanej dla `RandomizedSearchCV`. Z powodów ograniczeń obliczeniowych, nie udało nam się przeprowadzić kompletnej optymalizacji bayesowskiej dla klasyfikatora `SVC`.



Rysunek 6: Różnice w dokładności dla θ^* wybranej przy pomocy `RandomizedSearchCV` oraz dokładności w każdej iteracji `BayesSearchCV` dla klasyfikatora `GradientBoostingClassifier`



Rysunek 7: Różnice w dokładności dla θ^* wybranej przy pomocy `RandomizedSearchCV` oraz dokładności w każdej iteracji `BayesSearchCV` dla klasyfikatora `RandomForest`

3 Wnioski

Na podstawie wykresów 3, 4, oraz 5 można stwierdzić, że wynik predykcji na zbiorze testowym dla θ^* jest lepszy niż którykolwiek z wyników otrzymany w procesie przeszukiwania siatki z użyciem metody `RandomizedSearchCV`. Utwierdza nas to więc w przekonaniu o optymalności wybranych zestawów hiperparametrów.

Jednocześnie można odczytać, że różnice między miarą dopasowania dla θ^* a minimalną wartością ze wszystkich dopasowań to odpowiednio 0.03, 0.012 oraz 0.06 dla algorytmów `GBC`, `RF` oraz `SVC`. W zależności od punktu widzenia, największa z tych rozpiętości, 0.06, może zarówno świadczyć o tunowalności algorytmu `SVC`, jak i nieoptymalności pewnych kombinacji hiperparametrów w siatce.

Wykresy 6 i 7 pokazują, że optymalizacja bayesowska nie zawsze poprawia wynik przeszukania losowego. W przypadku 6 jedynie dwóch z czterech z rozważanych zbiorów, *Phoeneme* i *kc2*, wyniki dopasowania w kolejnych iteracjach były jednoznacznie lepsze niż dla θ^*. Żeby jednak móc jednoznacznie stwierdzić że ta optymalizacja słusznym krokiem, należałoby przeprowadzić odpowiedni test statystyczny sprawdzający, czy uzyskane wyniki są, statystycznie rzecz biorąc, istotnie różne.

W przypadku klasyfikatora `RandomForest` sytuacja kształtuje się inaczej, co widzimy dla 7. W przypadku dwóch pierwszych zbiorów możemy stwierdzić, że polepszenia wyników w wyniku optymalizacji nie zaobserwowałyśmy. Dla pozostałych dwóch zbiorów możemy zaobserwować, że wyniki dla rozważanych kombinacji były zarówno lepsze jak i gorsze.

Aby oddać sprawiedliwość metodzie `BayesSearchCV`, musimy pokreślić, że są to obserwacje dotyczące konkretnych czterech zbiorów. Aby wnioski były pełniejsze, symulacje należałoby przeprowadzić na setkach, jeśli nie tysiącach zbiorów, nie ograniczając parametrów jedynie do zakresów podanych przez nas. Oznaczałoby to oczywywiście dużo większe nakłady obliczeniowe, których niestety nie miałyśmy. Niemniej, w ramach nawet tak małych siatek jak te rozważane w naszych symulacjach, da się zaobserwować pewne rozproszenie, sięgające nawet 0.2, w wartościach dopasowania. Świadczy to więc na

rzecz tunowalności algorytmów.