Tunowalność algorytmów uczenia maszynowego

Katarzyna Rogalska
Paweł Pozorski

Listopad 2024

1 Cel projektu

Celem projektu jest zbadanie tunowalności trzech algorytmów uczenia maszynowego

- XGBoost
- Logistic Regression
- KNN

Będziemy posługiwać się terminologią, która jest zgodna z definicjami zawartymi w artykule Tunability: Importance of Hyperparameters of Machine Learning Algorithms.

2 Wstęp

Do eksperymentu wybraliśmy 5 ramek danych do klasyfikacji binarnej. Każda z nich zawiera dane z podobnej tematyki jaką są linie lotniczne. Wybrane ramki to:

- Us domestic flights delay prediction
- 2019 Airline delays and cancellations
- Airlines delay
- Flight delay dataset
- Airline quality ratings

Następujące operacje zostały przeporwadzone na każdej z wyżej wspomnianych ramek w ramach prepocessingu, w następującej kolejności:

1. Ograniczenie się do pierwszych 10000 obserwacji, ze sprawdzeniem czy w tym zbiorze zmienna objaśniana jest podobnie zbalansowana co na całym zbiorze.
2. Usunięcie kolumn posiadających $> 50\%$ brakujących wartości, jak i kolumn o odsetku wierszy o tej samej wartości przekraczającym 95% bądź kolumn kategorycznych o więcej niż 5000 klasach.
3. Imputacja kolumn numeryczych medianą oraz kategorycznych najczęściej występującą obserwacją.
4. OrdinalEncoding kolumn kategorycznych.

3 Optymalizacja hiperparametrów

Wykorzystaliśmy dwie metody optymalizujące hiperparametry: Grid Search oraz Bayes Search. Obie metody zostały wywołane dla każdego modelu na wszystkich ramkach danych używając tych samych przestrzeni parametrów podanych poniżej. Z uwagi na czsochłonność metody Bayes Search ograniczyliśmy ją do 100 iteracji, co jak zobaczymy w dalszej części okazało się wystarczające. Parametry oraz ich wartości ustalone były bazując na znalezionych artykułach, które są dołączone w bibliografii.

XGBoost Parameter Grid	
n_estimators:	100, 300, 500, 1000
learning_rate:	0.01, 0.05, 0.1
max_depth:	10, 15, 20
min_child_weight:	1, 2, 5, 10
gamma:	0, 0.15, 0.3
eval_metric:	"logloss", "error", "auc"
random_state:	42

Logistic Regression Parameter Grid	
C:	np.logspace(-4, 4, 10)
max_iter:	50, 100, 200, 300, 500, 1000
tol:	1e-3, 1e-4, 1e-5, 1e-6
fit_intercept:	True, False
class_weight:	None, "balanced"
random_state:	42
max_iter:	500

KNN Parameter Grid	
n_neighbors:	3, 5, 7, 9, 11, 15, 30, 50
weights:	"uniform", "distance"
algorithm:	"auto", "ball tree", "kd tree", "brute"
metric:	"euclidean", "manhattan", "minkowski", "chebyshev"
leaf_size:	20, 30, 40, 50

3.1 Wyznaczenie defaultowych hiperparametrów

Zgodnie z definicją defaultu zawartą w artykule zobrazujmy na czym polegało ich wyznaczenie. Dla uproszczenia przyjmijmy, że pracowaliśmy na 3 ramkach danych i dla wybranego modelu (z 3-wymiarową przestrzenią parametrów θ) wywołaliśmy na nich Grid Search otrzymując wyniki AUC jak na framgencie tabeli poniżej.

Dataframe	Parameter 1	Parametr 2	Parametr 3	AUC
1	0.1	50	6	0.7
2	0.1	50	6	0.78
3	0.1	50	6	0.81
1	0.6	80	7	0.91
2	0.6	80	7	0.87
3	0.6	80	7	0.9

Table 1: Przykładowa tabela wyników AUC dla 2 zestawów parmetrów

Następnie dla każdego zestawu parametrów obliczamy średnie AUC uzyskane na wszystkich ramkach. W naszym przypadku mamy fragment dwóch zestawów : $\theta_1 = (0.1, 50, 6)$ oraz $\theta_2 = (0.6, 80, 7)$. Łatwo zauważyć, że dla drugiego zestawu hiperparametrów średnie AUC jest wyższe niż dla pierwszego. Rozszerzając ten przykład na każdy model i o więcej zestawów, dla każdego modelu zestaw defaultowy θ_d to ten, który daje nam średnie najwyższe AUC.

4 Stabilność metod

Pierwszym krokiem było sprawdzenie po jakim czasie każda z metod samplingu znajdzie najlepszy zestaw hiperparametrów oraz jakie są różnice między wynikami w kolejnych iteracjach Grid Search i Bayes Search. Wykresy 1, 2 i 3 obrazują maksymalne znalezione do tej pory AUC w zależności od liczby iteracji dla każdego modelu i każdej metody samplingu. Oprócz tego sprawdziliśmy bezpośrednio wyniki

AUC w każdej iteracji dla Grid Search i Bayes Search, aby zobaczyć w jaki sposób obie metody zbliżają się do znalezienia najlepszych parametrów. Wykresy 4 i 5 przedstawiają dwie z ciekawszych obserwacji, podczas gdy w większości przypadków nie zauważyliśmy istotnych różnic w działaniu GS i BS.

4.1 Wnioski

Z pierwszych trzech wykresów widzimy, że Bayes Search prawie dla każdego przypadku potrzebuje znacznie mniej iteracji do znalezienia optymalnego zestawu hiperaparametró niż Grid Search. Z wykresów 4 i 5 możemy zauważyć, że rzeczywiście metoda Bayesa spędza mniej czasu na przeszukiwanie parametrów, które dają słabe wyniki, jednak podobne zależności nie zostały zaobserwowane dla każdej z ramek danych. Należy tu jednak wspomnieć, iż we wcześniejszym sformuowaniu "mniej czasu" oznacza mniejszą liczbę potrzebnych iteracji, co niekoniecznie przekłada się na fizyczny czas szukania optimum, z uwagi na ograniczone możliwości wielowątkowości Bayes Searcha. Na podstawie wyników czasowych z tabeli 2 wyciągamy również następujący wniosek: Bayes Search potrzebując ułamka iteracji Grid Searcha jest lepszą opcją do optymalizacji ciężkich modeli o dużej przestrzeni parametrów. Jednak w przypadku modeli lekkich których czas trenowania jest mały (i można je zrównoleglić), Grid Search może okazać się szybszy.

Model	Search Method	Dataset	Iterations	Time	Avg It Time	Coefficient
KNN	Bayes Search	1	100	207.470	2.070	17.250
KNN	Grid Search	1	1024	119.660	0.120	0.058
KNN	Bayes Search	2	100	240.060	2.400	14.118
KNN	Grid Search	2	1024	175.210	0.170	0.071
KNN	Bayes Search	3	100	232.420	2.320	33.143
KNN	Grid Search	3	1024	66.700	0.070	0.030
KNN	Bayes Search	4	100	274.810	2.750	5.851
KNN	Grid Search	4	1024	480.140	0.470	0.171
KNN	Bayes Search	5	100	228.410	2.280	11.400
KNN	Grid Search	5	1024	203.230	0.200	0.088
LR	Bayes Search	1	100	210.940	2.110	105.500
LR	Grid Search	1	160	3.200	0.020	0.009
LR	Bayes Search	2	100	233.240	2.330	233.000
LR	Grid Search	2	160	1.530	0.010	0.004
LR	Bayes Search	3	100	206.840	2.070	207.000
LR	Grid Search	3	160	0.860	0.010	0.005
LR	Bayes Search	4	100	233.010	2.330	38.833
LR	Grid Search	4	160	9.510	0.060	0.026
LR	Bayes Search	5	100	208.490	2.080	208.000
LR	Grid Search	5	160	1.410	0.010	0.005
XGBoost	Bayes Search	1	100	152.680	1.530	25.500
XGBoost	Grid Search	1	1296	76.890	0.060	0.039
XGBoost	Bayes Search	2	100	258.110	2.580	5.733
XGBoost	Grid Search	2	1296	584.910	0.450	0.174
XGBoost	Bayes Search	3	100	269.510	2.700	7.500
XGBoost	Grid Search	3	1296	472.370	0.360	0.133
XGBoost	Bayes Search	4	100	395.650	3.960	2.658
XGBoost	Grid Search	4	1296	1929.640	1.490	0.376
XGBoost	Bayes Search	5	100	242.160	2.420	10.522
XGBoost	Grid Search	5	1296	301.170	0.230	0.095

Table 2: Porównanie całkowitego czasu optymalizacji na platformie z 8 rdzeniami

Szybko można wywnioskować że GridSearch skaluje się dużo lepiej, oraz że kroki pomiędzy kolejnymi iteracjami BayesSearch mogą być czasochłonne. Tu dla Grid Search - Coefficient = Avg It Time dla GridSearch / Avg It Time dla Bayes Search i vice wersa dla każdego modelu, datasetu.

5 Tunowalność algorytmu

Ostatnim krokiem jest wyznaczenie tunowalności algorytmów. Zgodnie z definicją z artykułu będziemy mierzyć różnicę między wynikiem AUC modeli trenowanych na najlepszych znalezionych parametrach, a wynikiem AUC modeli trenowanych na parametrach defaultowych dla każdego modelu. Wykres 6 przedstawia tunowalność modeli wyznaczoną za pomocą obu metod samplingu.

5.1 Wnioski

Mamy na uwadze, że 5 ramek danych to zdecydowanie za mało, aby wysnuwać konkretne praktyczne wnioski. Jednak z analizy wyników naszego eksperymentu wynika, że obie metody samplingu dały bardzo podobne wyniki, z których widzimy, że algorytm KNN jest najmniej podatny na tunowanie hiperaparametrów, natomiast wyniki dla XGBoost i Logistic Regression mogą poprawić się nawet o ok 2 %.

6 Dodatkowe analizy

Biorąc pod uwagę niewielką ilość ramek danych przeprowadziliśmy analizę różnicy AUC między modelami trenowanymi na każdym zestawie hiperparametrów, a AUC na zestawie defaultowym, otrzymane wyniki są pokazane na wykresie 7.

Łatwo zauważyć, że w znacznej większości przypadków modele osiągają lepsze wyniki na parametrach defaultowych niż na losowo dobranych, stąd można podejrzewać, że metoda wyznaczenia defaultowych hiperparametrów użyta w tym eksperymencie rzeczywiście może poprawić performance modeli w stosunku do losowego ustawienia parametrów.

7 Wykresy

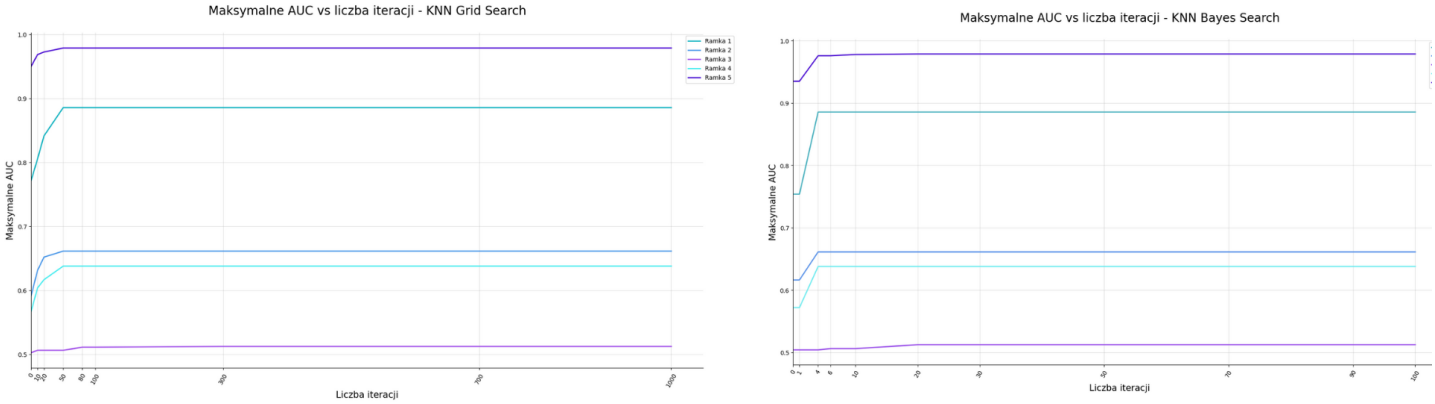

Figure 1: Ilość iteracji potrzebnych do uzyskania najlepszego AUC - KNN

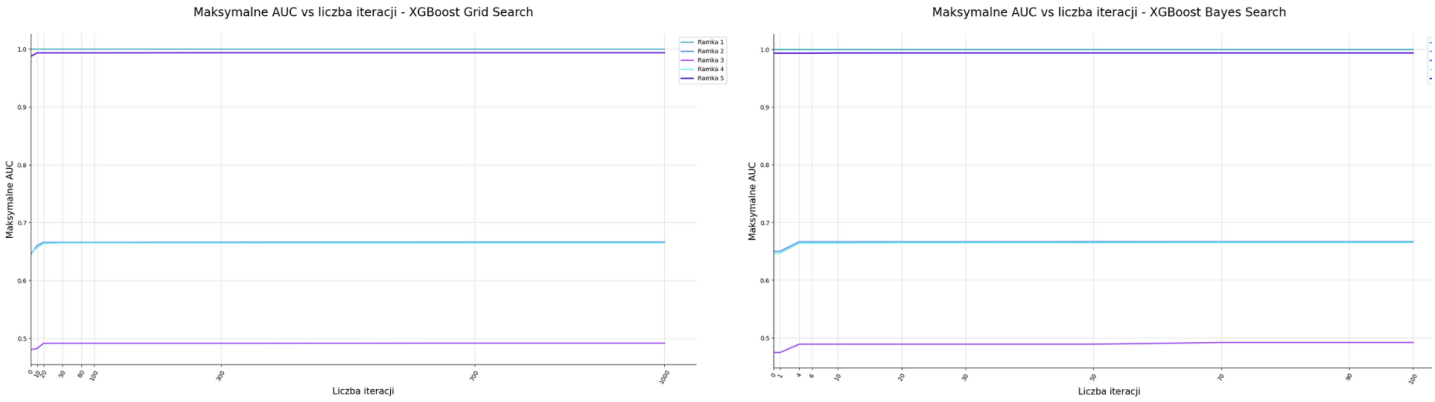

Figure 2: Ilość iteracji potrzebnych do uzyskania najlepszego AUC - XGBoost

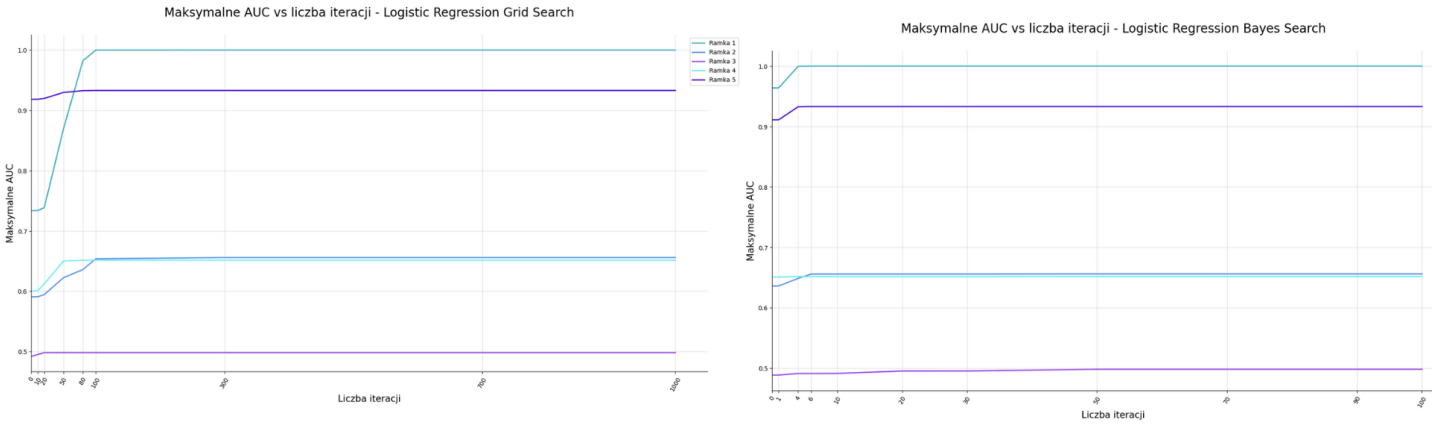

Figure 3: Ilość iteracji potrzebnych do uzyskania najlepszego AUC - Logistic Regression

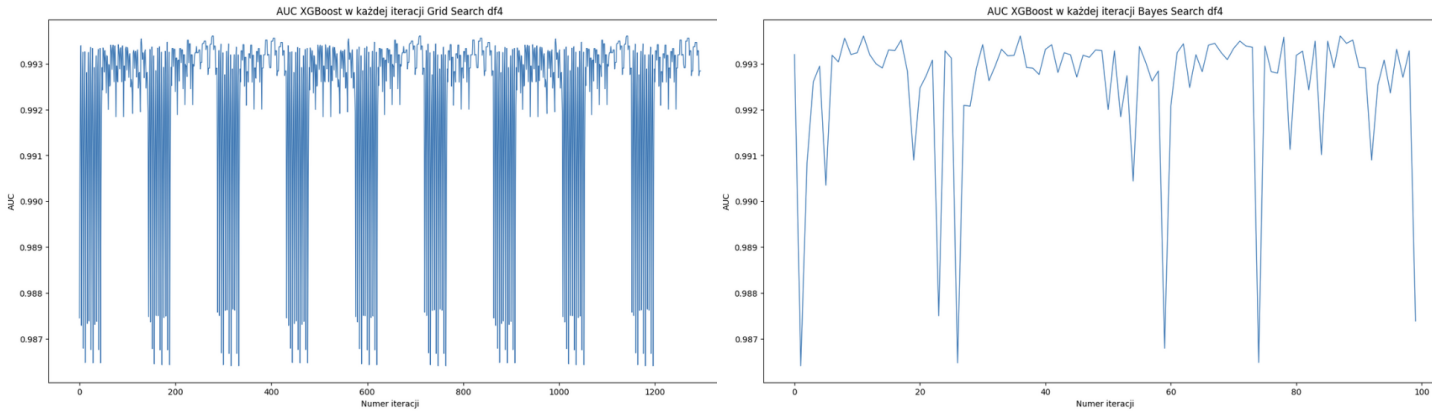

Figure 4: AUC w każdej iteracji poszukiwań dla XGBoost

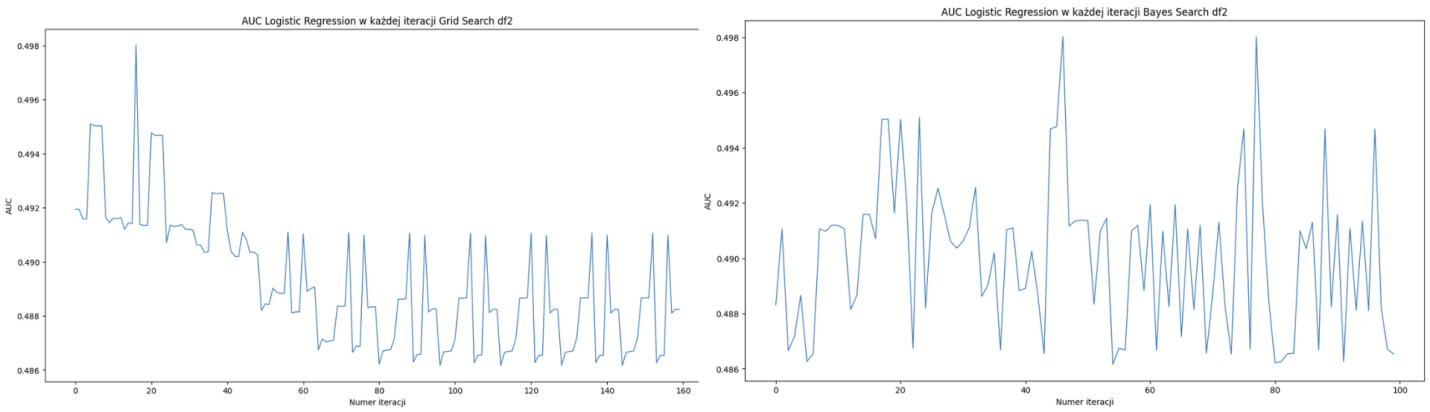

Figure 5: AUC w każdej iteracji poszukiwań dla Logistic Regression

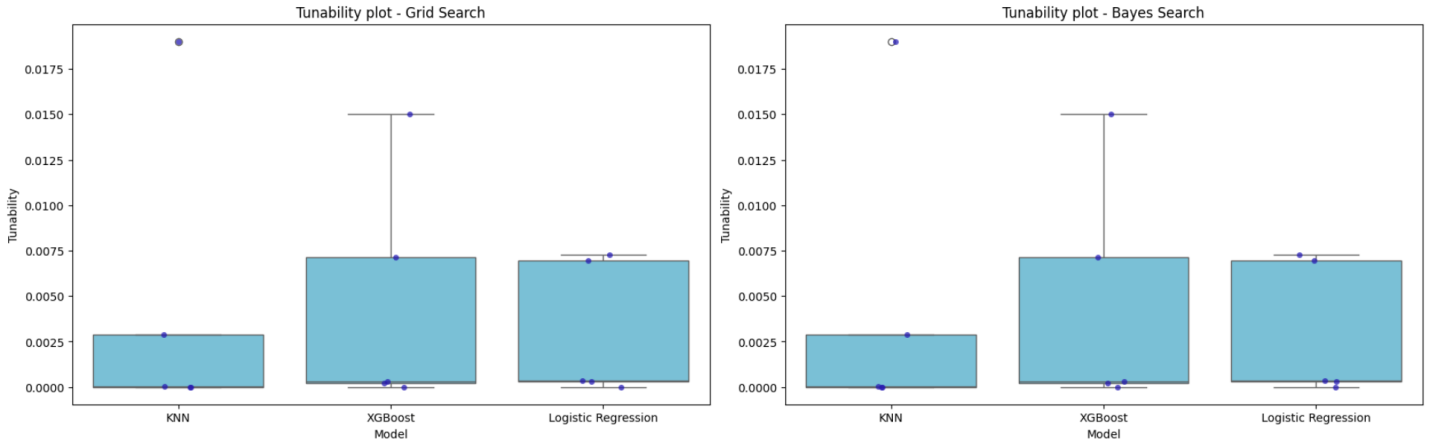

Figure 6: Tunowalność algorytmów

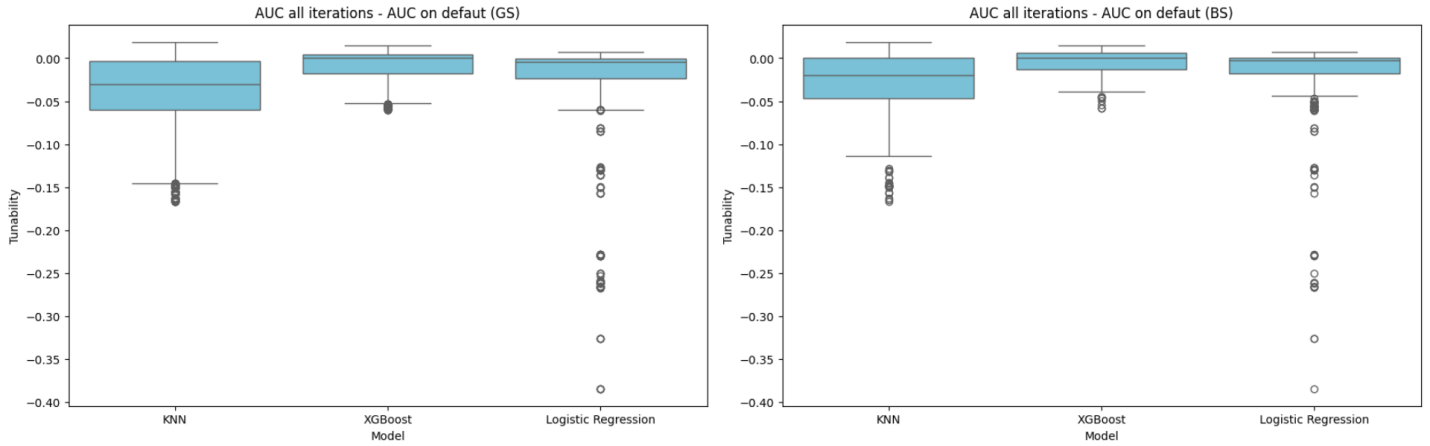

Figure 7: Różnica AUC w każdej iteracji metod od AUC na zestawie defaultowym

Literatura

Poniżej przedstawiamy artykuły, którymi kierowaliśmy się przy doborze przestrzeni hiperparametrów.

XGBoost

- Xgboost official documentation
- Medium publication
- More detailed grid
- Excellent toward data science article on optimization
- XGBoost hyperparameters explained
- Many XGBoost optimization techniques

Logistic Regression

- Official Documentation
- Medium Paper
- Is it even worth tuning?

KNN

- Official documentation
- Medium
- Some intuition