Przeszukiwanie i optymalizacja - projekt

Dokumentacja wstępna

Skład zespołu:

Ignacy Dąbkowski
Ireneusz Okniński

Treść zadania

Optymalizacja hiperparametrów algorytmu xgboost za pomocą algorytmów heurystycznych

Algorytm xgboost jest uważany za jedną z najlepszych metod klasyfikacji. Niestety jego stosowanie jest utrudnione ze względu na bardzo dużą liczbę hiperparametrów takich jak np. liczba produkowanych drzew decyzyjnych, ich głębokość, krok uczenia, wielkość podzbiorów losowanych przy kolejnym kroku etc. Wymienione atrybuty często optymalizowane są ręcznie w ograniczony sposób. Zadanie to można jednak robić w mądrzejszy sposób wykorzystując algorytmy heurystyczne. W ramach tematu tego projektu należy przetestować różne algorytmy heurystyczne/populacyjne w kontekście problemu strojenia hiperparametrów algorytmu xgboost. Jakość działania algorytmów należy przetestować na podstawie dowolnego (nietrywialnego) zbioru. Np: https://www.kaggle.com/c/porto-seguro-safe-driver-prediction.

Analiza problemu

Celem projektu będzie próba optymalizacji parametrów algorytmu xgboost przy użyciu algorytmów heurystycznych. Podzieliliśmy problem na kilka ważnych zagadnień:

1. Wybór parametrów algorytmu xgboost.
2. Wybór algorytmów optymalizacji.
3. Wybór zbiorów danych do testowania rozwiązania.
4. Przeprowadzenie analizy danych.
5. Projekt implementacji
6. Planowanie eksperymentów.

Poniżej opisujemy szczegóły każdego z powyższych kroków:

Wybór parametrów algorytmu xgboost.

Przedstawienie parametrów

Korzystamy z algorytmu zaimplementowanego w bibliotece xgboost dla języka Python (wersja algorytmu 2.0.2). Poniższe parametry pochodzą z dokumentacji biblioteki, podzielone są na trzy główne kategorie:

1. General parameters
    - booster - Którego modelu używamy jako “wzmacniacza”, do wyboru: tree, linear, dart.
    - device - Mówi na jakim urządzeniu ma działać algorytm
    - verbosity - Poziom wyświetlania komunikatów
    - validate_parameters - Czy algorytm ma sprawdzać poprawność (istnienie) parametrów
    - nthread - Ilość wątków używanych przez xgboost
    - disable_default_eval_metric - Flaga wyłączająca domyślną metrykę oceny
2. Booster parameters

    Tree Booster:
    - eta - Rozmiar kroku zmniejszający wagi cech, zapobiegający nadmiernemu dopasowaniu. Im większe eta, tym bardziej zachowawczy proces wzmacniania.
    - gamma - Minimalna redukcja strat wymagana do podziału liścia w drzewie. Im większe gamma, tym bardziej zachowawczy algorytm.
    - max_depth - Maksymalna głębokość drzewa. Zwiększenie tego parametru sprawi, że model stanie się bardziej skomplikowany, ale również bardziej skłonny do nadmiernego dopasowania.
    - min_child_weight - Minimalna suma wagi instancji potrzebna w potomku. Im większe min_child_weight, tym bardziej zachowawczy algorytm.
    - max_delta_step - Maksymalna zmiana kroku, jaką można zastosować do każdego liścia. Pomaga kontrolować aktualizację, szczególnie przy niezrównoważonych klasach.
    - subsample - Stosunek podpróby instancji treningowych. Zapobiega nadmiernemu dopasowaniu, gdy ustawione na wartość mniejszą niż 1.
    - sampling_method - Metoda próbkowania instancji treningowych, np. równomierne lub oparte na gradientach.
    - colsample_bytree, colsample_bylevel, colsample_bynode - Parametry do podpróbkowania kolumn, kontrolujące, ile kolumn jest używanych podczas konstrukcji drzewa na różnych poziomach i węzłach.
    - lambda - Wyraz kary L2 na wagi. Zwiększenie tego parametru sprawi, że model będzie bardziej zachowawczy.
    - alpha - Wyraz kary L1 na wagi. Zwiększenie tego parametru sprawi, że model będzie bardziej zachowawczy.
    - tree_method - Algorytm konstrukcji drzewa, np. auto, exact, approx, hist. Wartość auto oznacza to samo co hist.
    - scale_pos_weight - Kontroluje równowagę wag pozytywnych i negatywnych, przydatne przy niezrównoważonych klasach.
    - updater - Kolejność aktualizatorów drzewa, umożliwiająca elastyczną konstrukcję drzew. Najczęściej ustawiane automatycznie.

- refresh_leaf - Flaga odświeżająca liście drzewa. Gdy ustawione na 1, odświeża zarówno liście, jak i statystyki węzłów.
- process_type - Typ procesu wzmacniania: domyślny (default) lub aktualizacja (update).
- grow_policy - Kontroluje sposób dodawania nowych węzłów do drzewa: według głębokości (depthwise) lub przewodnika straty (lossguide).
- max_leaves - Maksymalna liczba węzłów do dodania.
- max_bin - Maksymalna liczba dyskretnych koszów do kategoryzacji cech ciągłych.
- num_parallel_tree - Liczba równoległych drzew konstruowanych podczas każdej iteracji, wspierająca lasy losowe.
- monotone_constraints - Ograniczenia monotoniczności zmiennych.
- interaction_constraints - Ograniczenia interakcji między cechami.
- multi_strategy - Strategia treningu modeli wielokierunkowych.
- max_cached_hist_node - Maksymalna liczba buforowanych węzłów dla histogramu CPU.

Dart Booster

- sample_type - Typ algorytmu próbkowania dla dropoutu. "uniform" oznacza, że odrzucane drzewa są wybierane równomiernie, a "weighted" oznacza, że są wybierane proporcjonalnie do wagi.
- normalize_type - Typ algorytmu normalizacji dla dropoutu. "tree" oznacza, że nowe drzewa mają tę samą wagę co odrzucone drzewa, a "forest" oznacza, że mają wagę równą sumie odrzuconych drzew.
- rate_drop - Współczynnik dropoutu określający frakcję wcześniejszych drzew, które mają być odrzucone podczas dropoutu. Wartości mieści się w zakresie od 0.0 do 1.0.
- one_drop - Gdy ta flaga jest włączona, zawsze przynajmniej jedno drzewo zostanie odrzucone podczas dropoutu, co pozwala na "Binomial-plus-one" lub "epsilon-dropout" z oryginalnej pracy nad DART.
- skip_drop - Prawdopodobieństwo pominięcia procedury dropoutu podczas iteracji wzmacniania. Jeśli dropout jest pominięty, nowe drzewa są dodawane w ten sam sposób, co dla gbtree. Wartości mieści się w zakresie od 0.0 do 1.0. Ważne jest, że wartość niezerowa skip_drop ma wyższy priorytet niż rate_drop lub one_drop.

Linear Booster

- lambda - Wyraz kary L2 na wagi. Zwiększenie tego parametru sprawi, że model będzie bardziej zachowawczy. Jest znormalizowany do liczby przykładów treningowych.
- alpha - Wyraz kary L1 na wagi. Zwiększenie tego parametru sprawi, że model będzie bardziej zachowawczy. Jest znormalizowany do liczby przykładów treningowych.
- updater - Wybór algorytmu do dopasowania modelu liniowego. "shotgun" to algorytm równoległego zstępowania współrzędnych, używający parallelizmu "hogwild". "coord_descent" to zwykły algorytm zstępowania współrzędnych. Przy ustawieniu parametru device na cuda lub gpu, używana jest wersja GPU.
- feature_selector - Metoda wyboru i porządkowania cech. "cyclic" oznacza deterministyczny wybór przez cykliczne przechodzenie przez cechy. "shuffle"

to podobne do cyklicznego, ale z losowym zamieszaniem cech przed każdą aktualizacją. "random" to losowy wybór współrzędnych. "greedy" wybiera współrzędną o największym gradientcie. "thrifty" to przybliżony, oszczędny selektor cech.
    - top_k - Liczba najlepszych cech do wyboru w algorytmach "greedy" i "thrifty". Wartość 0 oznacza użycie wszystkich cech.
3. Learning task parameters
    - objective - Określa cel uczenia i odpowiadającą mu funkcję celu. Na przykład, "reg:squarederror" to regresja z kwadratowym błędem, a "binary:logistic" to regresja logistyczna dla klasyfikacji binarnej.
    - base_score - Początkowy wynik predykcji dla wszystkich instancji, czyli globalna skłonność modelu. Automatycznie estymowany dla niektórych funkcji celu przed treningiem.
    - eval_metric - Metryka oceny dla danych walidacyjnych. Domyślna metryka zostanie przypisana automatycznie w zależności od celu (np. rmse dla regresji, logloss dla klasyfikacji). Można dodawać wiele metryk oceny, takich jak "rmse", "logloss", "auc", "map", itp.
    - seed - Ziarno, pozwalające na powtórzenie takiego samego przebiegu losowań.
    - seed_per_iteration - Ziarno ustawione deterministycznie przez numer iteratora. Domyślnie ustawione na false.

Algorytm posiada więcej hiperparametrów opisujących jego działanie gdy zostanie wybrany inny hiperparametr np. eval_metric. Zdecydowaliśmy, że wykorzystamy domyślną metrykę oceny, dlatego nie bierzemy tych parametrów pod uwagę w naszym problemie.

Wybór parametrów

Odrzucenie parametrów nie mających wpływu na wyniki oraz własnych metryk:

Na początku zdecydowaliśmy, że odrzucimy parametry nie mające żadnego wpływu na wyniki procesu uczenia się. Na tym etapie pozostały nam:

1. General parameters
    - booster
2. Booster parameters

    Tree Booster:
    - eta
    - gamma
    - max_depth
    - min_child_weight
    - max_delta_step
    - subsample
    - sampling_method
    - colsample_bytree, colsample_bylevel, colsample_bynode
    - lambda
    - alpha
    - tree_method
    - scale_pos_weight
    - updater

- refresh_leaf
- process_type
- grow_policy
- max_leaves
- max_bin
- num_parallel_tree

Dart Booster
- sample_type
- normalize_type
- rate_drop
- one_drop
- skip_drop

Linear Booster
- lambda
- alpha
- updater
- feature_selector
- top_k

3. Learning task parameters
   odrzucono wszystkie parametry

Wybór najbardziej znaczących parametrów

Po wstępnym przeanalizowaniu parametrów algorytmu, oraz konsultacji z prowadzącym zdecydowaliśmy, że wartym przeanalizowania będzie parametr “booster”, ponieważ od niego zależy z jakich elementów będziemy budować algorytm. Porównamy domyślny model “tree” oraz “linear”.

Model “linear”

Model liniowy ma jedynie 5 hiperparametrów. Zdecydowaliśmy się, że wybierzemy wszystkie.

Model “tree”

Model drzewiasty posiada aż 21 hiperparametrów. Trudno powiedzieć, które mają największy wpływ na działanie modelu. Planujemy początkowo użyć wszystkich. W przypadku, gdy model będzie się uczył wolno, lub jego wyniki będą bardzo zróżnicowane spróbujemy zbadać które z nich możemy odrzucić, żeby uprościć model.

Typy atrybutów

Atrybuty mają różne typy:
- ciągłe
- kategoryczne
- binarne

W algorytmach heurystycznych będziemy musieli uwzględnić ograniczenia na dany typ atrybutu.

Wybór algorytmów optymalizacji.

Zdecydowaliśmy się na trzy poniższe algorytmy optymalizacji:

- Algorytm genetyczny
- Symulowane wyżarzanie
- Przeszukiwanie losowe

Powyższe algorytmy mają różny poziom skomplikowania. Spodziewamy się, że rezultaty będą zróżnicowane. Aby porównać te algorytmy wymagane będzie wielokrotne uruchomienie trenowania i walidacji modelu tak, aby dało się wyciągnąć sensowne wnioski na podstawie listy wyników tych uruchomień.

Wybór zbiorów danych do testowania rozwiązania.

Wykorzystamy polecany w zadaniu zbiór danych: “porto-seguro-safe-driver-prediction”. Podczas konsultacji otrzymaliśmy sugestię, że warto porównać optymalizację parametrów xgboost na więcej niż jednym zbiorze danych. Wybraliśmy: “Binary Prediction of Smoker Status using Bio-Signals”. Drugi zbiór jest mniejszy niż pierwszy, pozwoli to sprawdzić czy algorytmy optymalizacji lepiej poradzą sobie z uczeniem modelu na prostszym zbiorze danych.

Źródła:
https://www.kaggle.com/c/porto-seguro-safe-driver-prediction
https://www.kaggle.com/competitions/playground-series-s3e24

Przeprowadzenie analizy danych.

Analiza danych znajduje się w notatnikach:
porto-seguro-safe-driver-prediction - porto-seguro_alalysis.ipynb
Binary Prediction of Smoker Status using Bio-Signals - smoker-status_analysis.ipynb

Projekt implementacji

Narzędzia

Algorytmy zostaną zaimplementowane w języku Python. Wykorzystamy biblioteki: numpy, pandas, scikit-learn, xgboost, matplotlib. Szczegóły użytych bibliotek znajdą się w pliku requirements.txt. Do analizy danych i przeprowadzenia eksperymentów użyjemy notatników Jupyter. Pozwoli to zaprezentować rezultaty w bardzo przejrzysty sposób.

Reprezentacja parametrów

Jednoczesne uczenie dwóch rodzajów elementów algorytmu xgboost będzie wymagało od nas wprowadzenia ograniczeń w algorytmach heurystycznych, tak aby w danym kroku zajmował się wyłącznie parametrami obecnego modelu. Wstępny plan zakłada, że każdy zestaw parametrów będzie reprezentowany jako słownik:
{
“tree”: [*lista parametrów modelu drzewiastego*],
“linear”: [*lista parametrów modelu liniowego*]
}
W podejściu tym w każdej iteracji będzie sprawdzane jakiego typu jest w danym momencie osobnik oraz modyfikowana będzie jedynie lista parametrów dla jego obecnego typu.

Obecny typ modelu będzie losowany z rozkładu Bernoulliego, tak aby zapewnić równomierną naukę dla obu typów parametrów.

Drugie podejście zakłada utworzenie jednej listy parametrów. Pierwszy parametr tj. “booster” będzie mówił na jakim wycinku listy obecnie wykonujemy algorytm. Pierwszy parametr będzie podlegał przekształceniom jak każdy pozostały parametr np. krzyżowanie.

Ostateczny wybór sposobu reprezentacji zostanie wybrany podczas implementacji algorytmów.

Zarys algorytmu

1. Przygotuj dane
2. Zainicjuj początkowy stan algorytmu optymalizacji
3. Dopóki n < n_max:
    3.1. Dla każdego zestawu parametrów (populacji)
        3.1.1. Oceń dany zestaw parametrów:
            3.1.1.1. Wytrenuj algorytm xboost dla danego zestawu parametrów, przy pomocy zbioru treningowego
            3.1.1.2. Oceń model przy pomocy zbioru walidacyjnego
    3.2. Zmień parametry zestawu zgodnie z przyjętym algorytmem
4. Zwróć najlepszy zestaw parametrów ze wszystkich iteracji.

Szczegółowy przebieg algorytmu będzie zależał od jego konkretnej implementacji.

Funkcja celu

Funkcją celu, którą będziemy optymalizować będzie jakość modelu xgboost dla danego zestawu parametrów obliczona na zbiorze walidacyjnym. Przy każdym wywołaniu model zostanie wytrenowany przy użyciu zbioru treningowego i oceniony przy użyciu zbioru walidacyjnego. Zgodnie z wymaganiem zadania przedstawionym na stronie kaggle.com, do oceny modelu użyjemy znormalizowanego współczynnika Giniego: https://en.wikipedia.org/wiki/Gini_coefficient

Mierzenie jakości

Po zakończeniu optymalizacji hiperparametrów, utworzymy przy ich użyciu model xgboost i zbadamy jego jakość na zbiorze treningowym. Wyniki z jednego uruchomienia algorytmu mogą być niedokładne z kilku powodów: losowość algorytmu heurystycznego, losowość uczenia modelu xgboost, inny podział zbioru danych na uczący, walidacyjny i treningowy. Aby wyniki były bardziej obiektywne porównamy rezultaty z wielu uruchomień.

Planowanie eksperymentów.

1. Porównanie przebiegu optymalizacji przy użyciu każdego z algorytmów heurystycznych
    a. ilość iteracji wymagana do znalezienia dobrego zestawu parametrów
    b. jakość najlepszego rozwiązania
    c. jak deterministyczny jest algorytm, czy w każdej iteracji znajdowane są zbliżone parametry
2. Porównanie optymalizacji na różnych zbiorach danych

a. czy znalezione parametry są różne dla różnych zestawów danych
b. czy dla mniejszego, prostszego zbioru danych osiągniemy lepsze rezultaty