

Optymalizacja hiperparametrów xgboost

Dokumentacja wstępna

Przemysław Stawczyk, Piotr Zmyślony

15 kwietnia 2020

Spis treści

| | | |
|----------|---|----------|
| 1 | Treść zadania | 2 |
| 2 | Dane testowe | 2 |
| 2.1 | Analiza danych | 2 |
| 3 | Algorytmy | 2 |
| 3.1 | Przestrzeń poszukiwań | 3 |
| 3.2 | Funkcja celu | 3 |
| 3.3 | Algorytm brutalny | 3 |
| 3.4 | Algorytm wspinaczkowy | 3 |
| 4 | Sposób mierzenia jakości rozwiązania | 4 |
| 5 | Wyniki pomiarów | 4 |
| 6 | Wnioski i rekomendacje | 4 |

1 Treść zadania

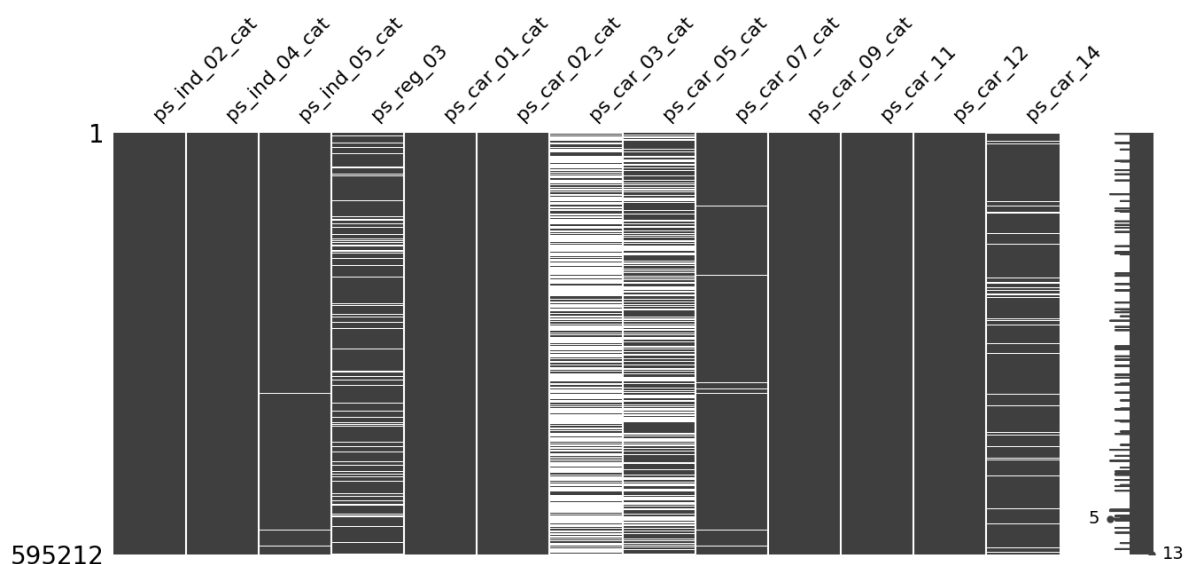
Naszym zadaniem jest przetestowanie różnych algorytmów heurystycznych/populacyjnych w kontekście problemu strojenia hiperparametrów algorytmu xgboost. Problem wyboru hiperparametrów wynika z ich bardzo dużej ilości, co często rozwiązane jest poprzez manualny dobór parametrów klasyfikatora.

Projekt zostanie zrealizowany w języku Python 3+.

2 Dane testowe

Jako dane na których będziemy trenować i testować klasyfikatory przyjęliśmy proponowany zestaw danych <https://www.kaggle.com/c/porto-seguro-safe-driver-prediction>. Zawiera on 57 atrybutów opisujących klientów firmy ubezpieczeniowej i jeden atrybut binarny sygnalizujący, czy w ciągu roku od zawarcia umowy, klient skorzystał z ubezpieczenia.

Rysunek 1: Brakujące atrybuty



2.1 Analiza danych

Po wstępnej analizie danych odkryliśmy, że w zbiorze danych posiadamy około 79% niekompletnych wierszy. Rysunek 1 przedstawia pokrycie niekompletnych atrybutów - jest ich jedynie 13, z czego większość jest wybrakowana w bardzo niewielkim stopniu.

Największym winowajcą jest atrybut binarny `ps_car_03_cat`, którego brakuje aż w 70% wierszy, oraz atrybut `ps_car_05_cat` (brakuje go w 44% przypadków). W końcowej wersji zdecydowaliśmy się usunąć oba z tych parametrów.

Dodatkowo, występuje znaczna dysproporcja między klasami rekordów - tylko 3% wierszy opisuje klientów, którzy skorzystali z ubezpieczenia. Stąd niezbędna będzie interpolacja danych, tak aby ilość rekordów obu klas była równa.

3 Algorytmy

Zaimplementowaliśmy następujące algorytmy:

- algorytm wspinaczkowy z tabu.
W 2 wariantach:
 - mutacyjny z prawdopodobieństwem P mutacji jednego (losowego) z parametrów
 - z przeglądem sąsiedztwa i powracaniem
- przegląd wyczerpujący hipersiatki jako metoda bazowa

W przypadku mutacji połączonej z tabu, problematycznym okazało się tworzenie mutantów z wykorzystaniem rozkładu normalnego, jak i mutacji w ten sam sposób wielu parametrów - tworzone były wielokrotnie już sprawdzone zestawy parametrów.

Przyjeliśmy metodę w której tworzone są wszystkie (jeszcze niezbadane) możliwe zestawy różniące się jednym parametrem w stosunku do rodzica i spośród nich losowany jest nowy mutant.

3.1 Przestrzeń poszukiwań

Trenowane modele posiadały parametry z hipersiatki, czyli iloczyn zbiorów każdego z parametrów, co w sumie daje nam 52272 dopuszczalnych rozwiązań.

| nazwa parametu | zakres |
|-----------------------------|---------------------|
| liczba słabych modeli | 50, 75, 100 ... 300 |
| eta | 0.1, 0.2, 0.3, 0.4 |
| min_split_loss <i>gamma</i> | 0, 1, 2, 3 |
| max_depth | 4, 5, 6 ... 14 |
| max_delta_step | 0, 1, 2 |
| subsample | 0.6, 0.8, 1 |
| colsample_bytree | 0.6, 0.8, 1 |

Dalej, *sąsiadem* zestawu A będziemy nazywali takie zestawy parametrów, które od zestawu A różnią się tylko jednym parametrem, większym bądź mniejszym o jeden "kwant".

3.2 Funkcja celu

Jako funkcję celu przybraliśmy *Average Precision Recall* obliczając wartość funkcji celu jako średnią arytmetyczną skuteczności przypisania predykcji. Zastosowaliśmy implementację z pakietu *scikit-learn*.

Ta sama funkcja zostanie wykorzystana do oceny jakości finalnych wytrenowanych modeli na zbiorach testowych, przy 3-krotnej walidacji krzyżowej, gdzie nasz końcowy jest średnią osiągniętych wyników AUC ROC.

3.3 Algorytm brutalny

Algorytm brutalny generuje zbiór wszystkich możliwych zestawów parametrów, które przegląda krok po kroku. Wynikiem działania jest zestaw parametrów osiągający najwyższy wynik AUC ROC.

3.4 Algorytm wspinaczkowy

Zaczyna swoje działanie od wylosowania punktu startowego z dostępnych zbiorów parametrów (na początku przypisujemy go do P_b i P_c). Oblicza AUC ROC dalej w następujący sposób:

1. wygeneruj wszystkich sąsiadów P_c

TODO PISAC CZY NIE PISAC TE PSEUDOKODY

4 Sposób mierzenia jakości rozwiązania

Jako że trenowanie znacznej liczby modeli przy użyciu adekwatnych ilości danych wejściowych jest bardzo czasochłonne, postanowiliśmy zaimplementować opcję testowania wszystkich trzech algorytmów w określonych ramach czasowych. Dzięki podaniu odpowiedniego parametru, program uruchamia każdy z nich tylko na jakiś czas, po którym algorytm musi zwrócić najlepszy dotychczasowy wynik.

5 Wyniki pomiarów

6 Wnioski i rekomendacje

Algorytmy zbyt nie były efektywne *imho* należałoby wałnąć se parę paramsów i okolice obiecujących zbadać przegłędem wyczerpującym Xd