# Optymalizacja hiperparametrów xgboost

Dokumentacja wstępna

Przemysław Stawczyk, Piotr Zmyślony

15 kwietnia 2020

## Spis treści

# 1 Treść zadania

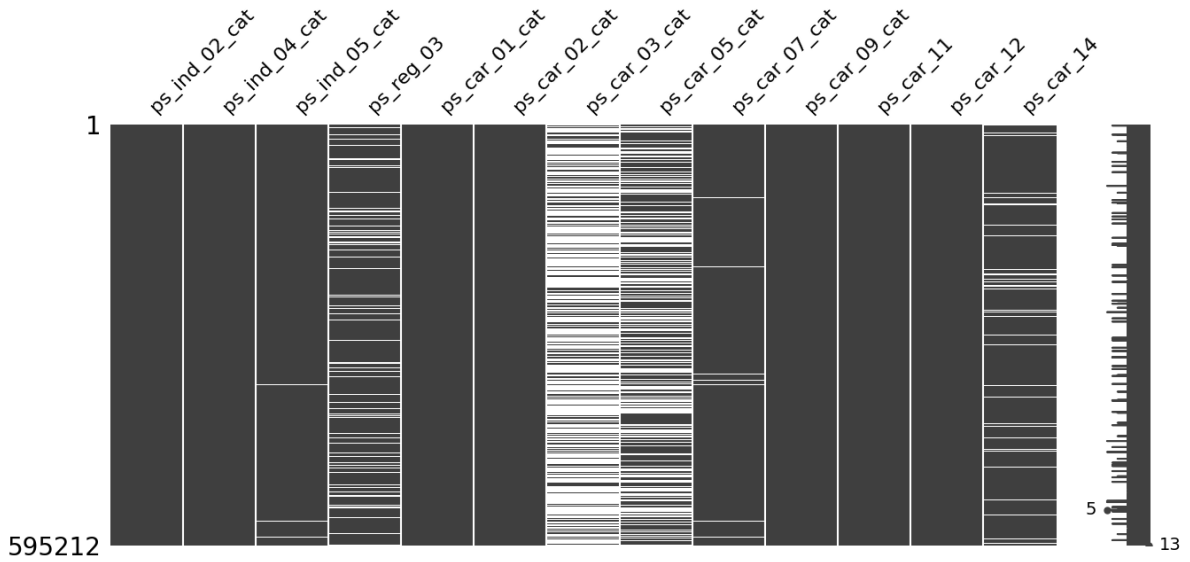
Naszym zadaniem jest przetestowanie różnych algorytmów heurystycznych/populacyjnych w kontekście problemu strojenia hiperparametrów algorytmu xgboost. Problem wyboru hiperparametrów wynika z ich bardzo dużej ilości, co często rozwiązane jest poprzez manualny dobór parametrów klasyfikatora.

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).

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 tablu problematycznym okazało się tworzenie mutantów z wykoorzystaniem 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 mozliwe zmiany pojedyńczego parametru [jeszcze niesprawdzone] i z pośród nich losowany jest nowy mutant.

## 3.1 Przestrzeń poszukiwań

Trenowane parametry - hipersiatka, iloczyn zbiorów każdego z parametrów.

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

# 4 Funkcja celu

Jako funkcję celu przybraliśmy *Average Precision Recall* obliczając wartość funkcji celu jako średnią arytmetyczną skuteczności przypisania predykcji. Planujemy wykorzystać implementację z pakietu *scikit-learn*.
Ta sama funkcja zostanie wykorzystana do oceny jakości finalnych wytrenowanych modeli na zbiorze testowym.

## 5 Sposób mierzenia jakości rozwiązania

## 6 Wyniki pomiarów

## 7 Wnioski i rekomendacje

Algorytmy zbytnio nie były efektywne *imho* należałoby walnąc se parę paramsów i okolice obiecujących zbadać przegłądem wyczerpującym Xd