PET UJ Raport nr 1/2019

Porównanie zastosowania algorytmów XGBoost oraz AdaBoost do klasyfikacji zdarzeń wielofotonowych w tomografie J-PET.

inż. Jan Bielecki

Wydział Fizyki, Politechnika Warszawska,

pod nadzorem:

dr inż. Wojciech Krzemień

Narodowe Centrum Badań Jądrowych, Otwock-Świerk

Spis treści

Streszczenie

Zadaniem klasyfikacyjnym było znalezienie wśród par fotonów zmierzonych w detektorze tych, które pochodzą z tego samego rozpadu para-pozytonium oraz ich pierwszą interakcją z otoczeniem była detekcja w scyntylatorze (te pary fotonów możemy użyć do rekonstrukcji obrazu), tak aby uzyskać jak najlepszą wstępną selekcję par fotonów. Dane użyte w zadaniu pochodzą z symulacji Monte-Carlo, wykonanych za pomocą oprogramowania GATE [12]. Do klasyfikacji użyto dwóch algorytmów: AdaBoost oraz XGBoost opartych na drzewach decyzyjnych.

Wyniki testowe skuteczności obu algorytmów są porównywalne, ze wskazaniem na XGBoost i prezentują się następująco: XGBoost - 85,59%, AdaBoost - 85,49% (wśród danych znajdowało się 73,27% przykładów o klasie 0 - tzw. próbek negatywnych oraz komplementarna ilość przykładów pozytywnych).

1. Wstęp

1.1 Zestaw danych

Zestaw danych został wygenerowany w środowisku GATE i przekształcony do postaci ramki danych z użyciem oprogramowania GOJA [11]. Badanym źródłem prominiowania był fantom IEC wypełniony radioaktywną substancją. Detektor składał się z jednej warstwy ściśle sąsiadujących (bez przerw) scyntylatorów (w liczbie 382) o szerokości 7 mm i długości 50 cm. Warstwa ta tworzyła cylinder o średnicy 85 cm. Użyte do analizy dane znajdują się na klastrze CIŚ [1] w folderze */mnt/opt/groups/jpet/NEMA_Image_Quality/3000s/*. Są to dane nierozmyte (nieuwzględniające niepewności pomiarowych). Więcej informacji o danych można znaleść w publikacji *Estimating the NEMA characteristics of the J-PET tomograph using the GATE package* [7]. Więcej szczegółów na temat działalności badawczej nad tomografem J-PET [2] można znaleść na stronach: `koza.if.uj.edu.pl/pet`[8] oraz `pet.ncbj.gov.pl`[9].

Do analizy wykorzystano część danych (o rozmiarze 10000000 wierszy na 11 kolumn) zawierających następujące atrybuty (kolumny):

- pozycja detekcji pierwszego fotonu (x, y, z) - 3 atrybuty,
- pozycja detekcji drugiego fotonu (x, y, z) - 3 atrybuty,
- energia detekcji pierwszego fotonu,
- energia detekcji drugiego fotonu,
- różnica czasów detekcji fotonów,
- numer scyntylatora, w którym zmierzono pierwszy foton,
- numer scyntylatora, w którym zmierzono drugi foton.

Ostatnia kolumna ramki zawiera oznaczenie klasy, do której zaliczona została dana para (*target*): 1 - dla fotonów pochodzących z tego samego rozpadu para-pozytonium, których pierwszą interakcją z otoczeniem była detekcja w scyntylatorze, 0 - dla pozostałych. Pierwotnie dane były podzielone na cztery klasy:

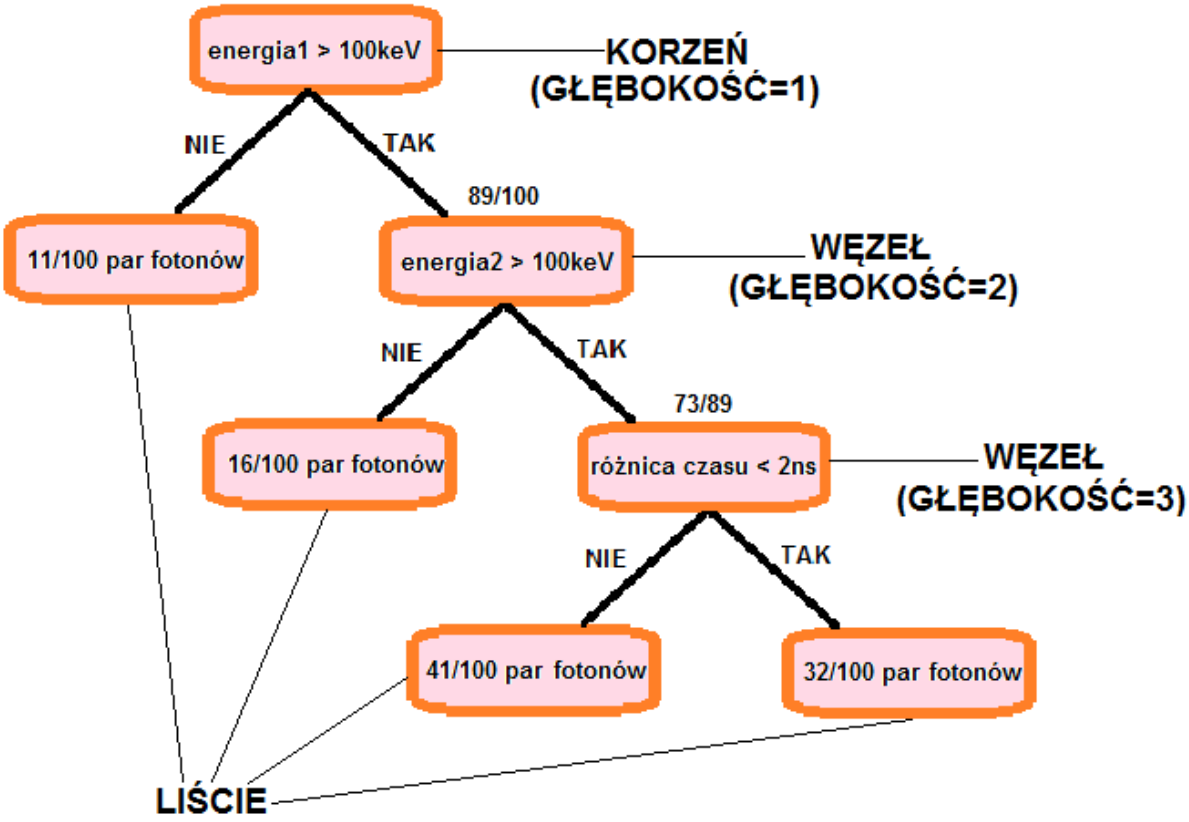
- 1 - oba rozproszenia pochodzą z tej samej anihilacji i są rozproszeniami pierwotnymi (nowa klasa 1),
- 2 - pochodzą z tej samej anihilacji, ale przynajmniej jedno rozproszenie jest wtórne i wcześniej ten foton rozproszył się w fantomie (nowa klasa 0),
- 3 - pochodzą z tej samej anihilacji, ale przynajmniej jedno rozproszenie jest wtórne i wcześniej ten foton rozproszył się w detektorze (nowa klasa 0),
- 4 - rozproszenia pochodzą z różnych anihilacji, mogą być wtórne (nowa klasa 0)

1.2 Drzewo decyzyjne

Drzewo dezycyjne to graficzna metoda wspomagania procesu decyzyjnego, stosowana w teorii decyzji. Algorytm drzew decyzyjnych jest również stosowany w uczeniu maszynowym do pozyskiwania wiedzy na podstawie przykładów [15]. Rysunek 1 przedstawia przykładowe drzewo decyzyjne, które dla analizowanych danych opisanych w poprzednim podrozdziale wykonuję wstępną

[1] CIŚ - Centrum Informatyczne Świerk
[2] JPET - Jagiellonian-PET

selekcję par fotonów dla przykładowych 100 par (gdzie energia1 - energia detekcji pierwszego fotonu, energia2 - energia detekcji drugiego fotonu, różnica czasu - różnica czasów detekcji fotonów).



Rysunek 1: Drzewo decyzyjnego wykonujące wstępną selekcję par fotonów.

1.3 XGBoost

XGBoost[3] [3] jest zoptymalizowaną biblioteką implementującą algorytm wzmacniania gradientowego, zaprojektowaną tak, aby była wysoce wydajna, elastyczna i przenośna. XGBoost zapewnia równoległe wzmacnianie drzew (znane również jako GBDT, GBM), które rozwiązuje wiele problemów związanych z uczeniem maszynowym w szybki i dokładny sposób. XGBoost jest bardzo efektywnym algorytmem (m.in. najczęściej wygrywa konkursy *kaggle*), ale algorytm wzmacniania gradientowego (ang. *gradient boosting*) łatwo ulega efektowi przetrenowania. Aby temu zapobiec XGBoost umożliwia skorzystanie z wielu hiperparametrów regularyzacyjnych.

Poniższa lista zawiera domyślne hiperparametry dla algorytmu XGBoost wraz z ich opisem:

- max_depth = 3 - maksymalna głębokość drzewa
- learning_rate = 0.1 - współczynnik uczenia - dla *XGBoost* jest to udział każdego z drzew w procesie wzmacniania gradientowego. *Boosting* polega na dodawaniu do puli tworzącej skumulowany predyktor, kolejnych klasyfikatorów trenowanych na podstawie błędów resztowych poprzedniego skumulowanego predyktora zdefiniowanych następująco:

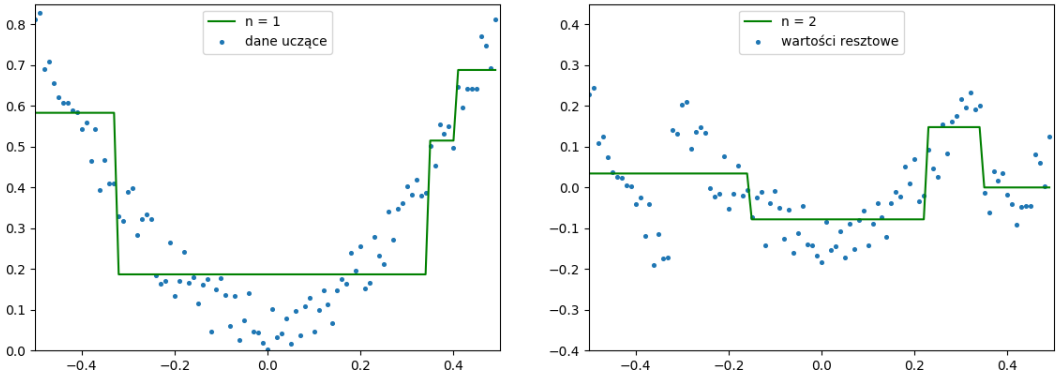
$$error(\hat{y_i}^j) = y_i - \hat{y_i}^{j-1},$$

[3]XGBoost oznacza *Extreme Gradient Boosting*.

gdzie $\hat{y_i}^j$ oznacza wynik regresji dla próbki uczącej i na podstawie predyktora skumulowanego z pierwszych j predyktorów, natomiast y_i oznacza wartość próbki uczącej i. Wynik regresji skumulowanego klasyfikatora opisuje wzór:

$$\hat{y_i}^j = {y_i}^1 + \theta \sum_{k=2}^{n} \hat{err or}(\hat{y_i}^j),$$

gdzie $\hat{err or}$ - wynik regresji dla błędów resztowych, natomiast θ - współczynnik uczenia. Rysunek 2a przedstawia przykładowe drzewo wytrenowane na danych uczących (oś odciętych - atrybut X - zawiera wartości od -0.5 do 0.5 oddalone o wartość 0.01, oś rzędnych - *target* $Y = 3X^2 + 0.15U(0,1)^4$), natomiast rysunek 2b przedstawia przykładowe drzewo wytrenowane na błędach resztowych poprzedniego drzewa[5] (prosta to wynik regresji, punkty - dane uczące).



(a) Pierwsze, wytrenowane drzewo. (b) Estymacja pierwszych błędów resztowych.

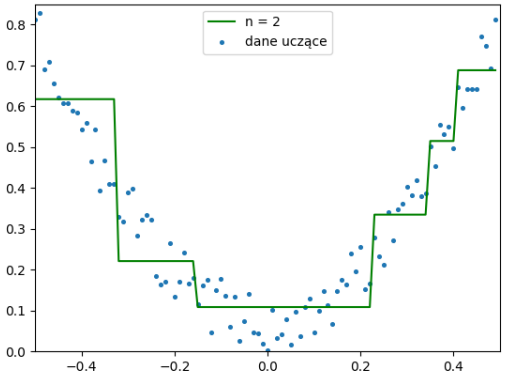
Rysunek 2: Pierwszy krok przykładowego *boosting*'u na drzewach.

Następnie odejmujemy od naszej estymowanej funkcji dla pierwszego drzewa estymowaną funkcję dla pierwszych błędów resztowych przemnożoną przez współczynnik uczenia[6] (Rysunek 3a) oraz liczymy błędy resztowe powstałego drzewa (Rysunek 3b).

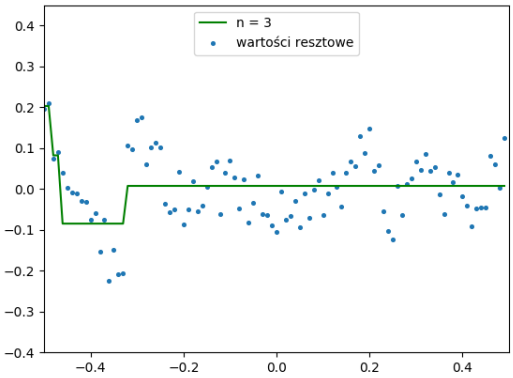
[4] U(0,1) - wartość losowa z rozkładu jednorodnego z zakresu [0, 1]

[5] Przykład na podstawie książki *Uczenia maszynowe z użyciem Scikit-Learn i TensorFlow* autorstwa Auréliena Gérona [5].

[6] W tym przykładzie współczynnik uczenia wynosi 1.



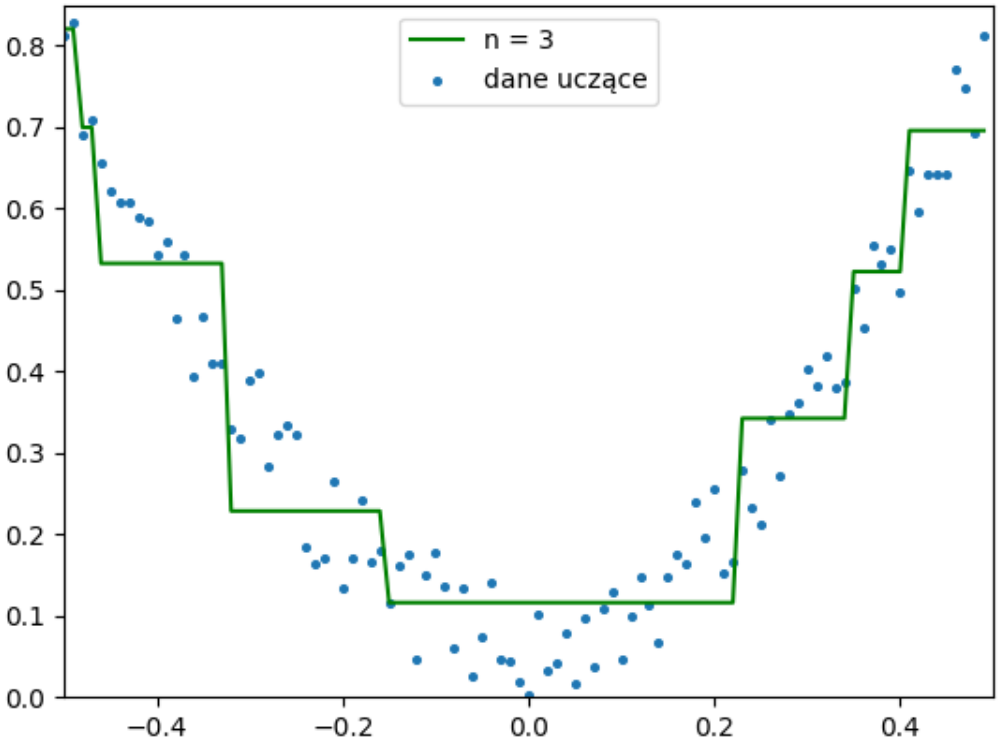
(a) Wynik regresji na podstawie predyktora kumulującego dwa estymatory.



(b) Estymacja kolejnych błędów resztowych.

Rysunek 3: Kolejny krok przykładowego *boosting*'u na drzewach.

Powtarzając procedurę *boosting*'u otrzymujemy coraz lepszy predyktor. Rysunek 4 przedstawia ostateczny predyktor składający się z 3 estymatorów (n_estimators = 3). Skrypt 1 zawiera kod programu wykonującego przedstawioną wizualizację *boosting*'u.

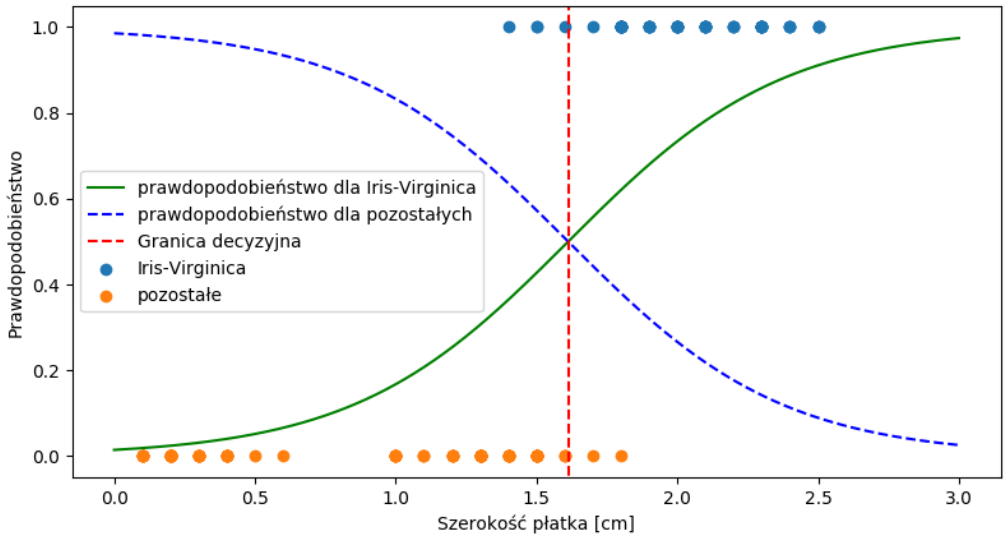


Rysunek 4: Wynik regresji na podstawie predyktora kumulującego trzy estymatory.

- n_estimators = 100 - liczba estymatorów wchodząca w skład ostatecznego - skumulowanego

predyktora

- objective = 'binary:logistic' - funkcja kosztu. Algorytm liczy koszt na podstawie regresji logistycznej dla klasyfikacji binarnej. Rysunek 5 wizualizuje przykład regresji logistycznej dla przypadku klasyfikacji binarnej (klasyfikator znajduje wśród wszystkich kwiatów popularnego zestawu *IRIS* te, które należą do gatunku *Iris-Virginica* na podstawie szerokości płatka). Skrypt 2 zawiera kod programu wykonującego przedstawioną wizualizację.



Rysunek 5: Wizualizacja przykładu regresji logistycznej dla klasyfikacji binarnej.

- booster = 'gbtree' - *Gradient boosting tree* - wykorzystanie drzew jako estymatorów do przeprowadzenia wzmacnianiania gradientowego (przykład działania pokazany przy opisie hiperparametru learning_rate, alternatywą są 'gblinear' - bazujący na liniowej estymacji oraz 'dart' - również bazujący na drzewach decyzyjnych [10]).
- gamma = 0 - minimalne zmniejszenie funkcji kosztu do jakiego musi dojść podczas powstawania nowego węzła - im większa *gamma* tym większa regularyzacja algorytmu.
- min_child_weight = 1 - minimalna suma wag wystąpień wymagana w węźle - im większa wartość tego hiperparametru tym większa regularyzacja algorytmu.
- max_delta_step = 0 - maksymalny krok zmiany umożliwiający oszacowanie wag drzewa. Wartość 0 oznacza brak ograniczenia. Wartość dodatnia powoduje, że krok aktualizacji może być realizowany bardziej konserwatywnie. Zwykle ten parametr nie jest potrzebny, ale może pomóc w przypadku regresji logistycznej, gdy klasa jest skrajnie niezrównoważona.
- subsample = 1 - ułamek próbek treningowych włączonych do procesu uczenia kolejnego drzewa wchodzącego w skład ostatecznego predyktora skumulowanego.
- colsample_bytree = 1 - ułamek atrybutów (kolumn) wykorzystywanych do stworzenia kolejnego drzewa wchodzącego w skład ostatecznego predyktora skumulowanego.
- colsample_bylevel = 1 - ułamek atrybutów (kolumn) wykorzystywanych do stworzenia kolejnego węzła drzewa.

- reg_alpha = 0 - regularyzacja wag liści - L1. Funkcję kosztu definiuje wzór:

$$L = \sum_{i=1}^{n} loss(\hat{y}_i(\mathbf{x}_j) - y_i) + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2 + \alpha \sum_{j=1}^{T} |w_j|,$$

  gdzie α - reg_alpha, λ - reg_lambda, w_j waga liścia j.

- reg_lambda = 1 - regularyzacja wag - L2.

1.4 AdaBoost

Algorytm AdaBoost polega na sekwencyjnym uczeniu predyktarów, w taki sposób, że każdy następny koryguje swojego poprzednika, poprzez zwracanie większej uwagi na przykłady uczące, dla których poprzedni estymator pozostał niedotrenowany. W ten sposób kolejne predyktory przykładają większą uwagę do najtrudniejszych przykładów. Algorytm AdaBoost korzysta z metody wzmacniania (ang. *boosting*). W przypadku użycia drzew decyzyjnych jako estymatora w algorytmie AdaBoost przykładanie większej uwagi do przykładów uczących dla których poprzedni estymator pozostał niedotrenowany (przykłady te dostają wyższe wagi), polega na ponownym próbkowaniu (ze zwracaniem) danych uczących, gdzie prawdopodobieństwo wylosowania przykładu jest większe dla przykładów o większej wadze zgodnie ze wzorem:

$$p_i = \frac{w_i}{\sum_{j=1}^{n} w_j},$$

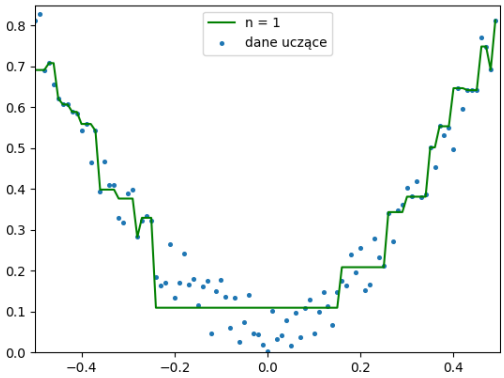
gdzie p_i - prawdopodobieństwo wylosowania przykładu i, n - liczba przykładów uczących, w_i - waga przykładu i jest aktualizowana za każdym dodanym estymatorem zgodnie z poniższym wzorem (na początku wszystkie próbki otrzymują równe wagi $p_i = \frac{1}{n}$ dla każdego $i \in [1, n]$):

$$w_i(t) = w_i(t-1)\beta^{1-L_i},$$

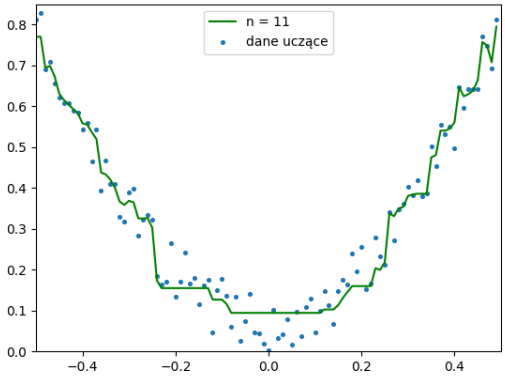
gdzie L_i - koszt próbki i, $\beta = \frac{\overline{L}}{1-\overline{L}}$, $\overline{L} = \sum_{i=1}^{n} L_i p_i$. Szczegółowe informacje dotyczące metody wzmacniania można znaleść w artykule *Improving Regressors using Boosting Techniques* [4].

Skrypt 3 służy do wizualizacji algorytmu AdaBoost na bazie drzew decyzyjnych o głebokości maksymalnej równej 5 (zastosowano liniową funkcję kosztu oraz wsółczynnik uczenia równy 0.2). Rysunki 6a, 6b, 6c, 6d, 6e oraz 6f przedstawiają regresję na danych uczących (oś odciętych - atrybut X - zawiera wartości od -0.5 do 0.5 oddalone o wartość 0.01, oś rzędnych - *target* $Y = 3X^2 + 0.15U(0,1)^7$) dla algorytmu AdaBoost opartego na drzewach decyzyjnych z ilością estymatorów równą kolejno: 1, 11, 21, 31, 41 oraz 50.

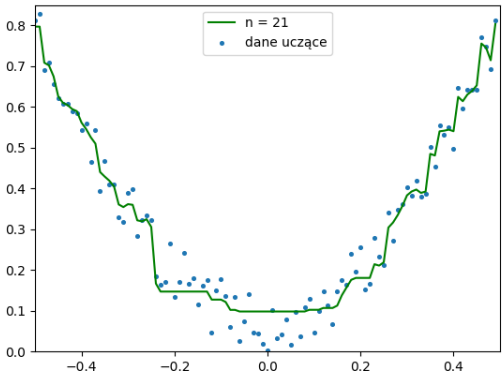
[7] U(0,1) - rozkład jednorodny z zakresu [0, 1]



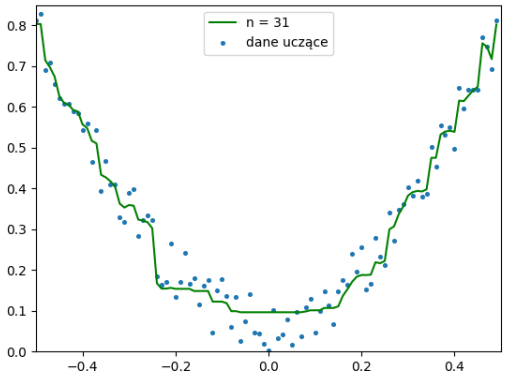
(a) Wynik regresji dla jednego estymatora).



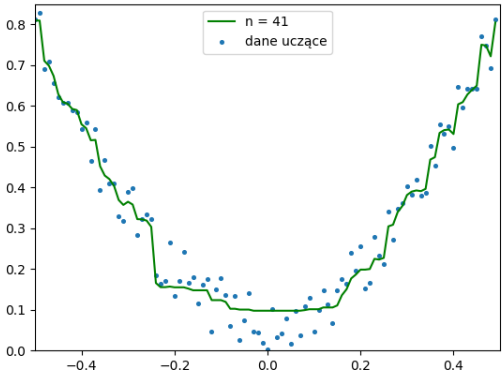
(b) Wynik regresji na podstawie predyktora kumulującego 11 estymatorów.



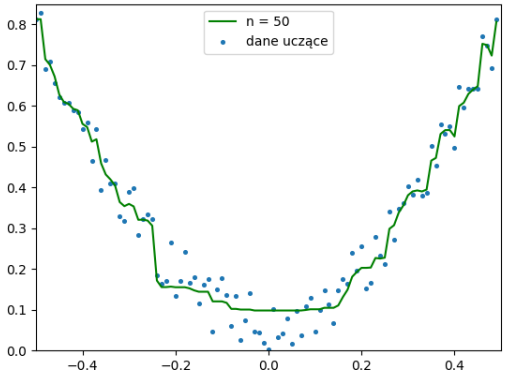
(c) Wynik regresji na podstawie predyktora kumulującego 21 estymatorów.



(d) Wynik regresji na podstawie predyktora kumulującego 31 estymatorów.



(e) Wynik regresji na podstawie predyktora kumulującego 41 estymatorów.



(f) Wynik regresji na podstawie predyktora kumulującego 50 estymatorów.

Rysunek 6: Wizualizacja działania algorytmu AdaBoost na drzewach decyzyjnych.

2. Wyniki

Dostępne dane wejściowe podzielono na dwa podzbiory (z zachowaniem proporcji występowania przykładów o danej klasie): 80% (8 000 000 próbek) - dane treningowe. 20% (2 000 000 próbek) - dane testowe. Skrypty użyte do przeprowadzania algorytmów oraz wizualizacji wyników znajdują się w pod adresem:
https://github.com/K4liber/MultiphotonClassification/tree/master/Classification/NemaSource

2.1 XGBoost

Pierwszym etapem poszukiwania odpowiednich hiperparametrów modelu XGBoost, dla których klasyfikator osiągałby najlepsze rezulaty było przeszukanie siatki hiperparametrów wyglądającej następująco[8] [9]:

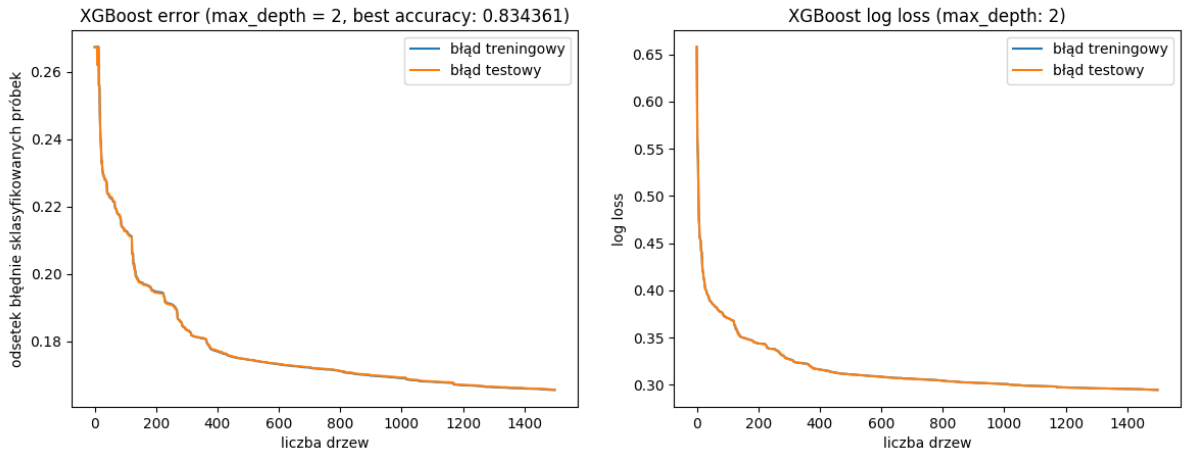
```
param_dist = {
  'n_estimators': stats.randint(50, 200),
  'learning_rate': stats.uniform(0.15, 0.05),
  'max_depth': [5, 6, 7, 8, 9, 10, 11],
  'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1],
  'min_child_weight': [1, 2, 3, 4]
}
```

Z powyższej siatki wylosowano 5 zestawów hiperparametrów (resztę hiperparametrów zostawiono domyślnych) i metodą sprawdzianu krzyżowego (ang. *cross-validation*) [14] z podziałem na 3 podzbiory sprawdzono rezultaty modeli z każdym z zestawów hiperparametrów. Pośród nich wybrano ten, który osiągnął najlepszą wartość AUC[10]. Jego parametry to:

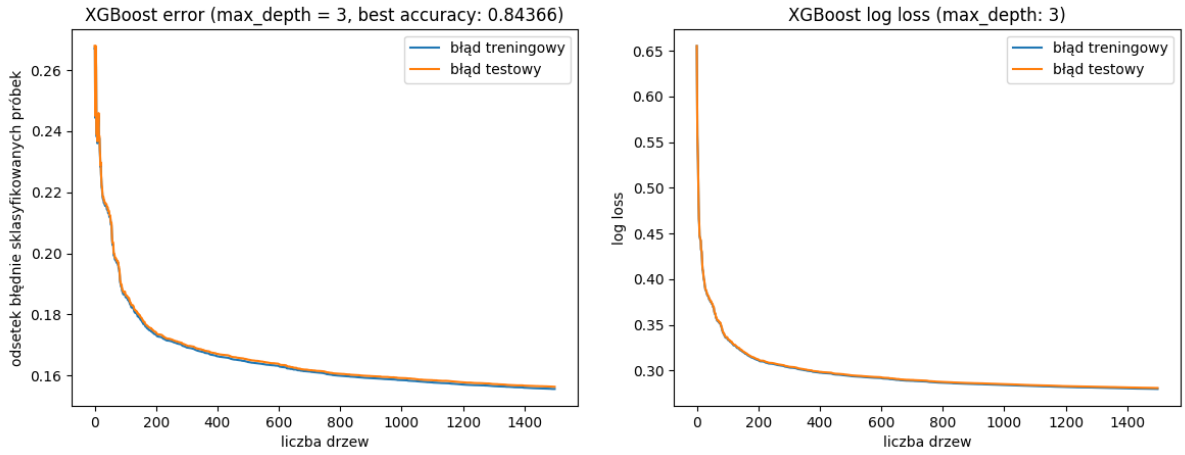
```
bestHiperparams = {
  'n_estimators': 188,
  'learning_rate': 0.1717,
  'max_depth': 7,
  'colsample_bytree': 0.6,
  'min_child_weight': 2
}
```

Następnie w zestawie najlepszych hiperparametrów zmieniano wartość maksymalnej głębokości drzewa (*max_ depth*) i sprawdzono wartości błędów klasyfikatora oraz funkcji straty (dla zestawu treningowego oraz zestawu testowego) w funkcji liczby drzew (*n_ estimators*). Wyniki przedstawiają rysunki 7-18.

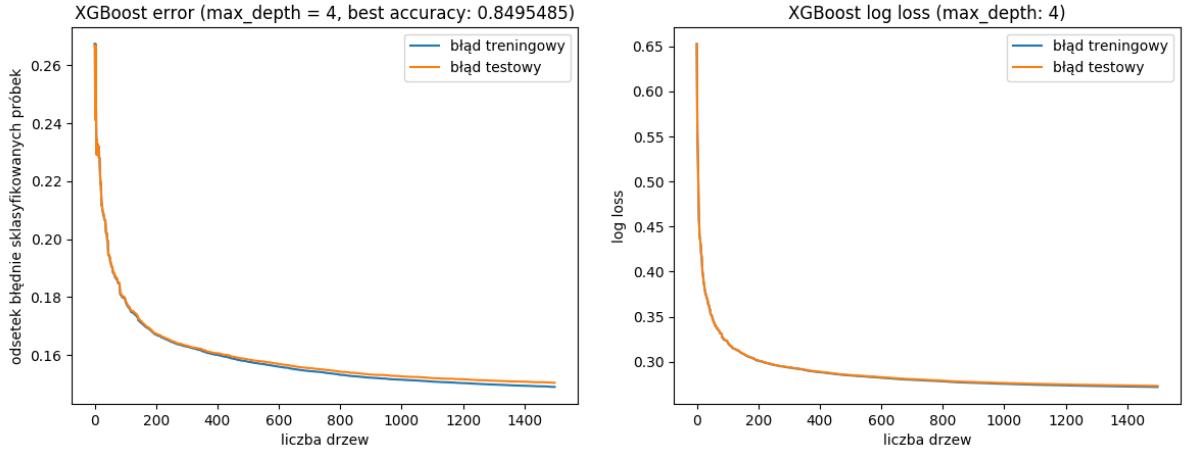
[8] stats.randint(50, 200) - losowa wartość całkowita z przedziału [50, 200]
[9] stats.randint(50, 200) - losowa wartość z rozkładu jednorodnego z przedziału [0.15, 0.2]
[10] AUC - pole pod krzywą charakterystyki roboczej odbiornika (ang. *receiver operating characteristic - ROC* [16])



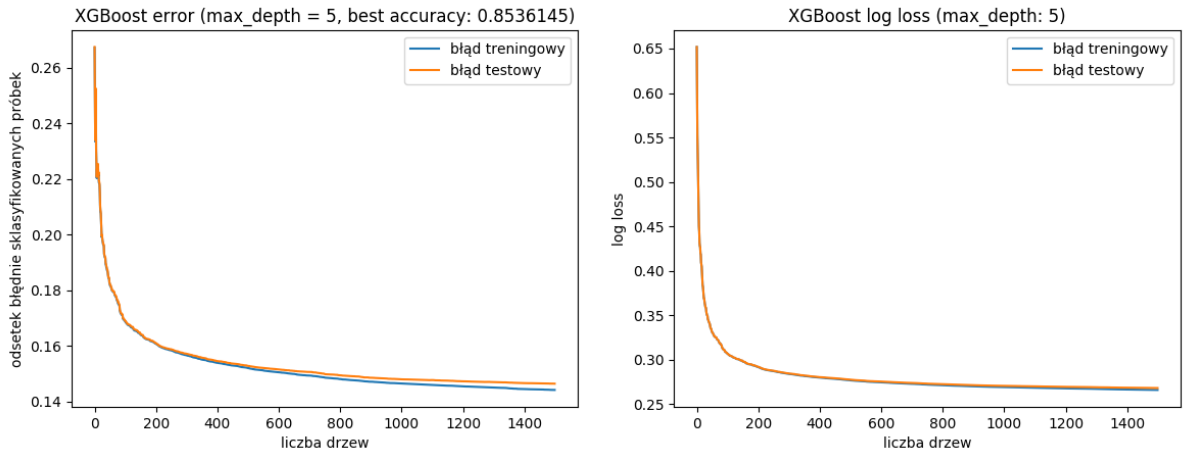
Rysunek 7: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 2$).



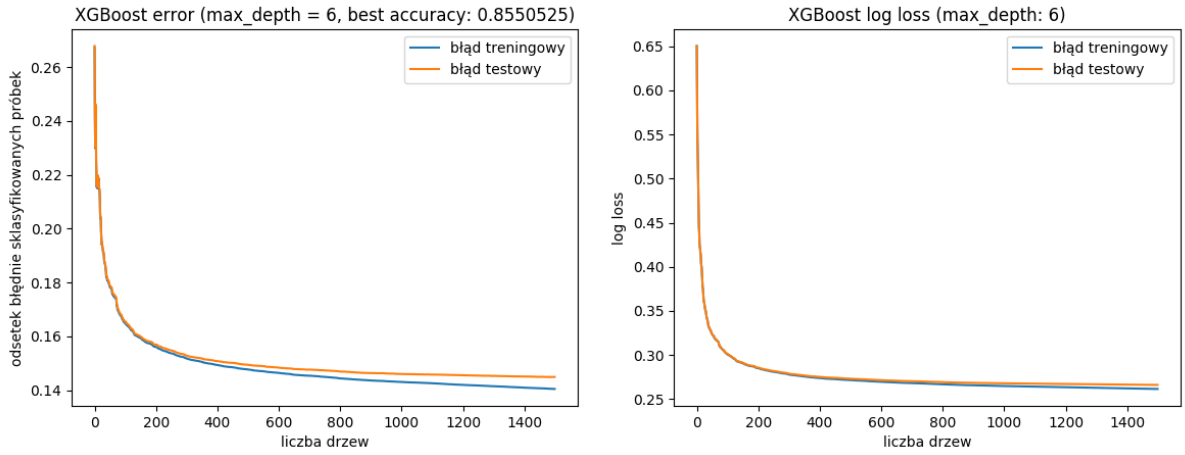
Rysunek 8: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 3$).



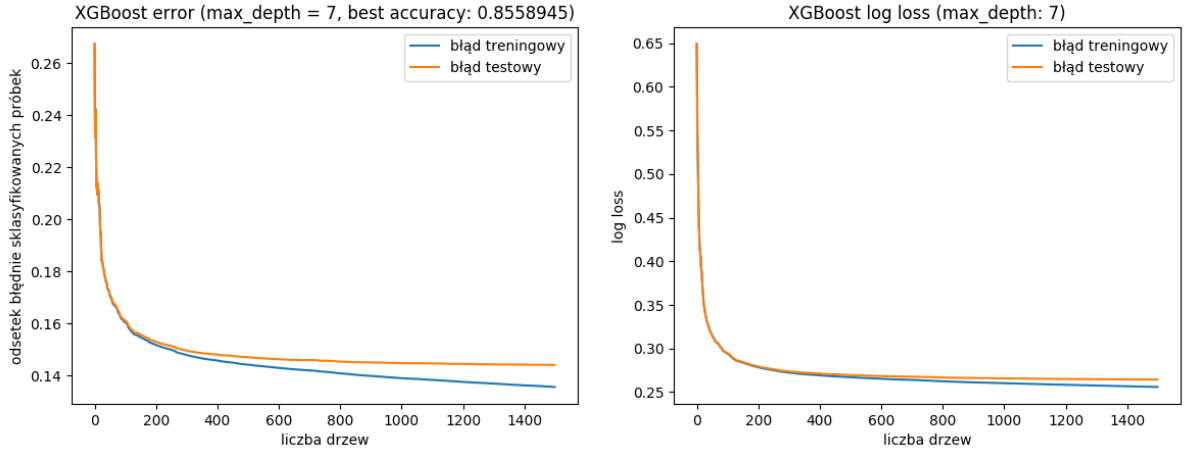
Rysunek 9: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 4$).



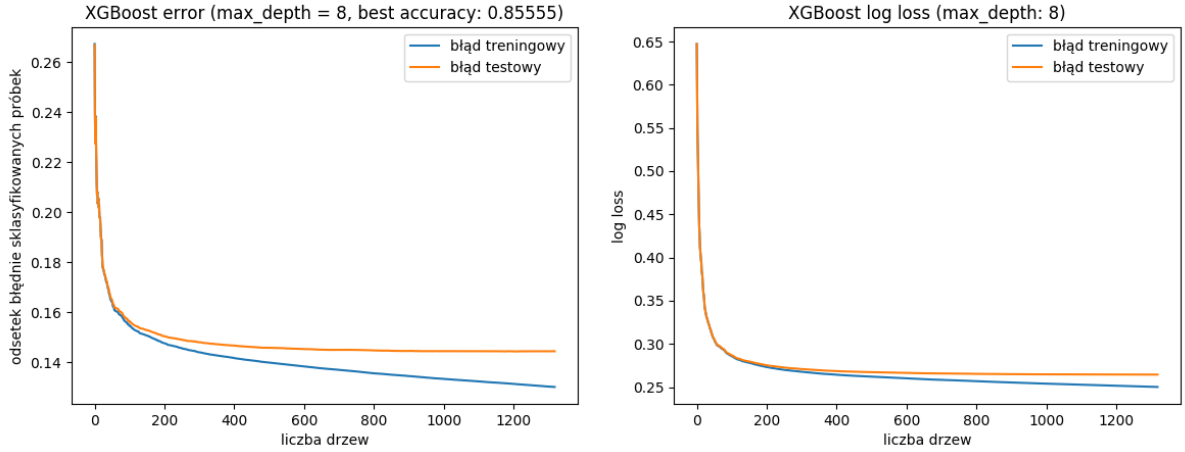
Rysunek 10: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 5$).



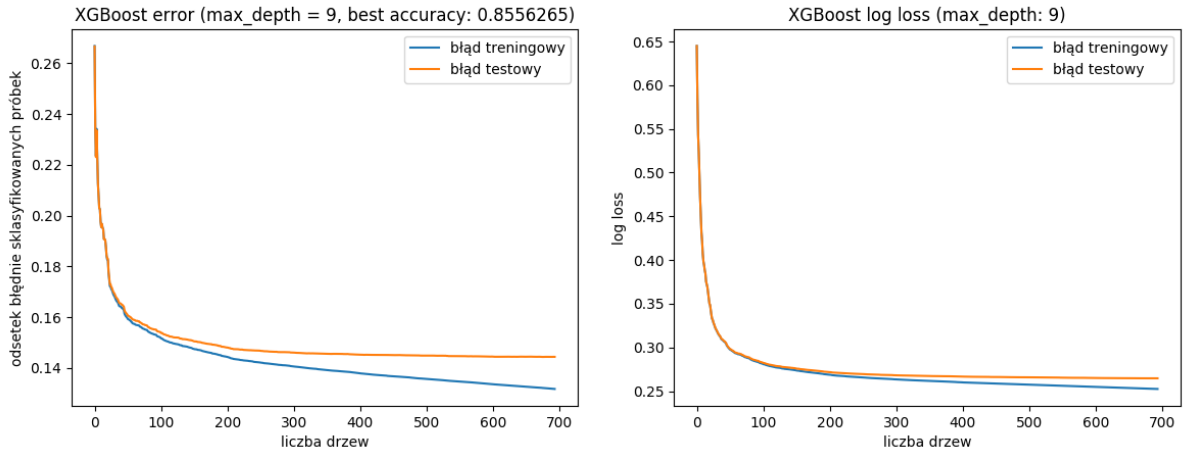
Rysunek 11: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 6$).



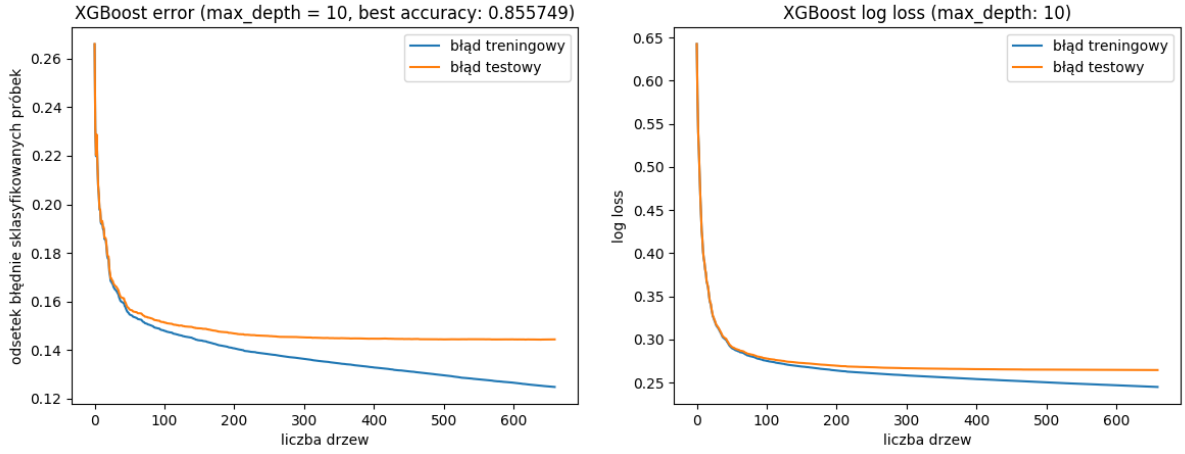
Rysunek 12: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 7$).



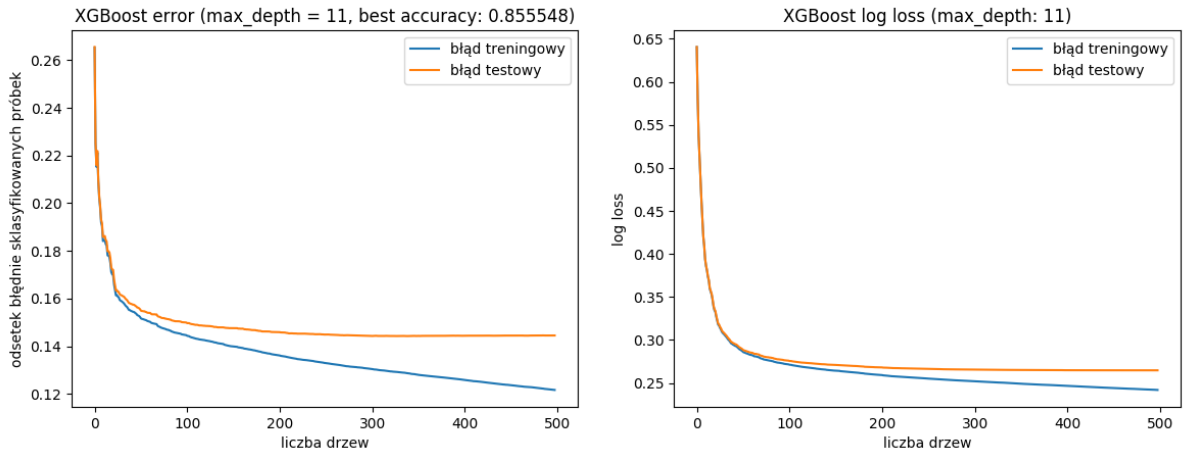
Rysunek 13: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 8$).



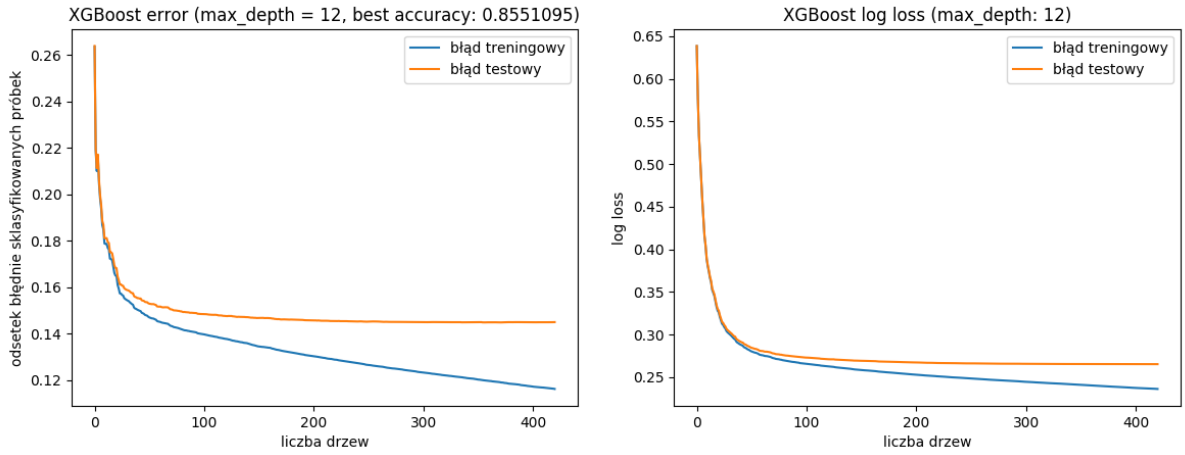
Rysunek 14: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 9$).



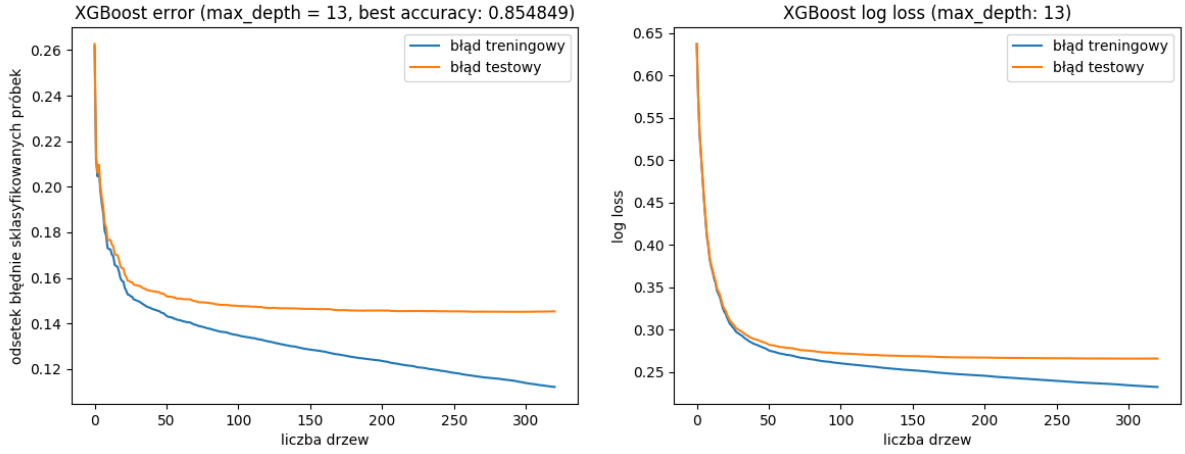
Rysunek 15: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 10$).



Rysunek 16: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 11$).



Rysunek 17: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 12$).



Rysunek 18: Błąd klasyfikatora opartego na XGBoost w funkcji liczby drzew ($max\_depth = 13$).

Następnie, działanie modelu o najlepszych hiperparametrach (na podstawie powyższych wyników wybrano $max\_depth = 7$ oraz $n\_estimators = 1500$) sprawdzono na zestawie danych testowych. Rysunki 19a oraz 19b przedstawiają macierze pomyłek odpowiednio dla danych treningowych oraz testowych, natomiast rysunki 20a oraz 20b przestawiają krzywe ROC odpowiednio dla danych treningowych i testowych, gdzie:

$ACC = \frac{TP+TN}{TP+FP+TN+FN}$ - skuteczność (ang. *accuracy*),

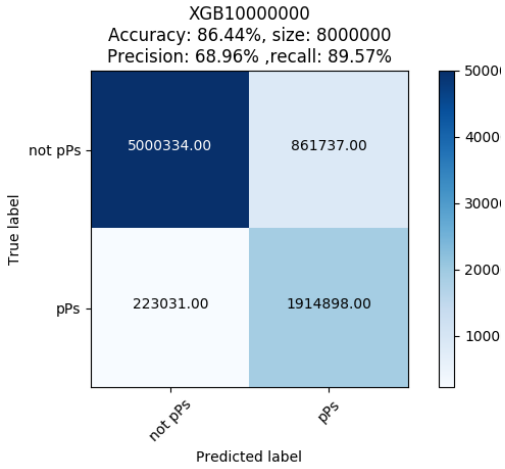
$TPR = \frac{TP}{TP+FN}$ - czułość (ang. *sensitivity*) bądź *recall*,

$TNR = \frac{TN}{TN+FP}$ - selektywność (ang. *selectivity*),

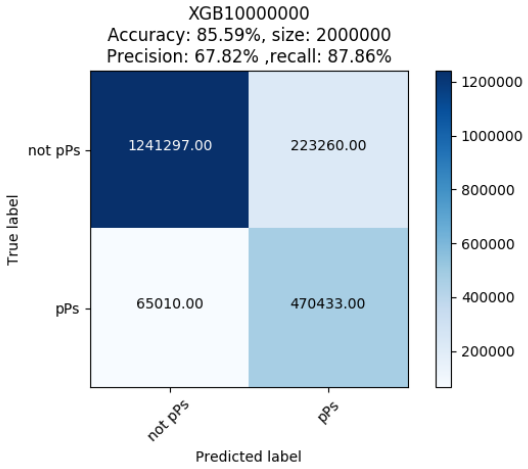
$PPV = \frac{TP}{TP+FP}$ - czystośc sygnału, precyzja (ang. *precision*),

$FPR = \frac{FP}{FP+TN}$ - akceptacja tła (ang. background acceptance).

gdzie *TP - true positive*, *TN - true negative*, *FP - false positive*, *FN - false negative*.

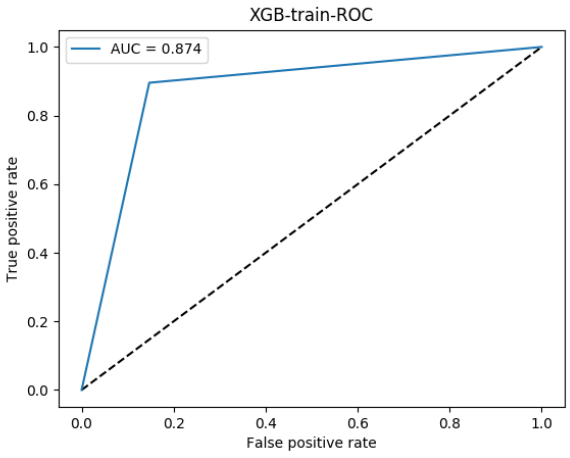


(a) Macierz pomyłek dla danych treningowych.

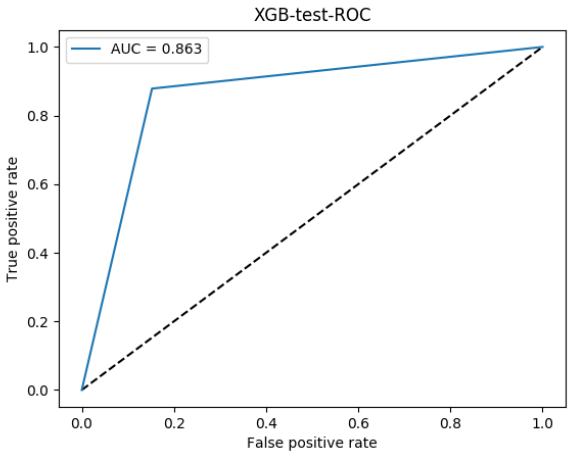


(b) Macierz pomyłek dla danych testowych.

Rysunek 19: Macierze pomyłek dla algorytmu XGBoost o najlepszych hiperparametrach.



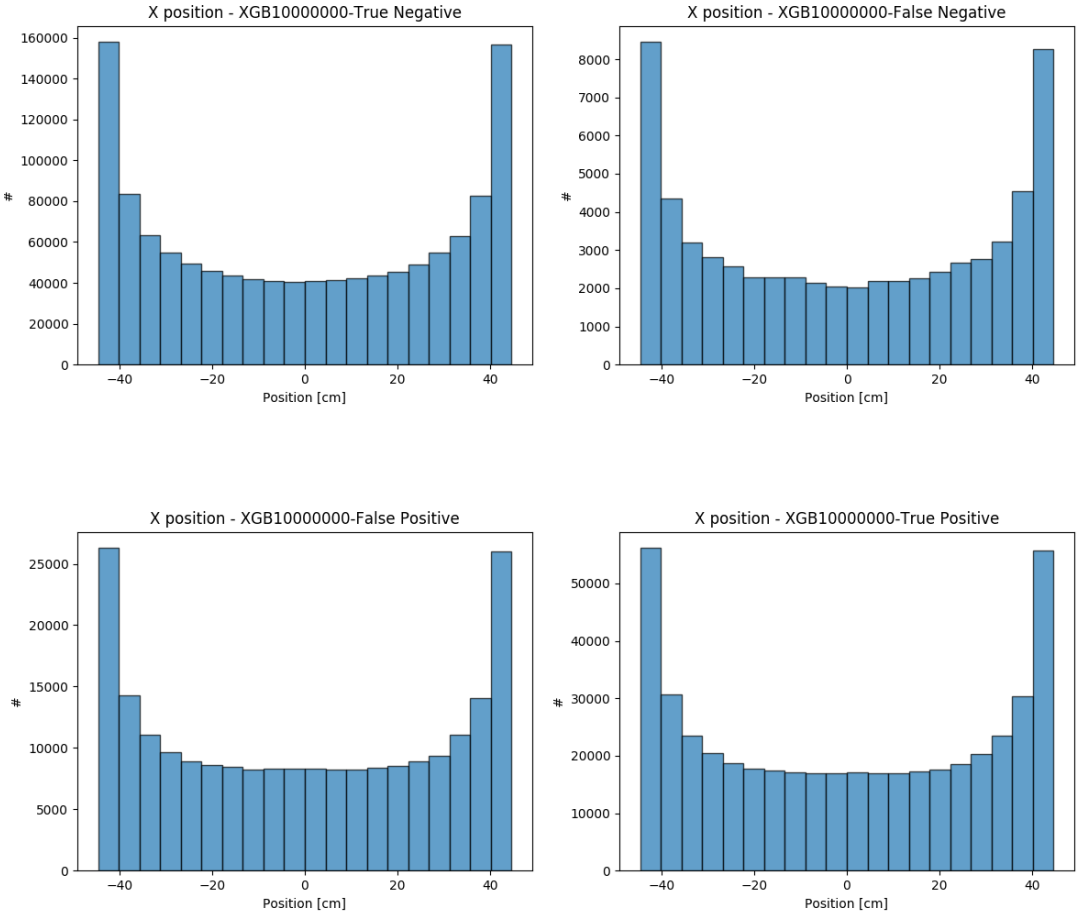
(a) ROC dla danych treningowych.



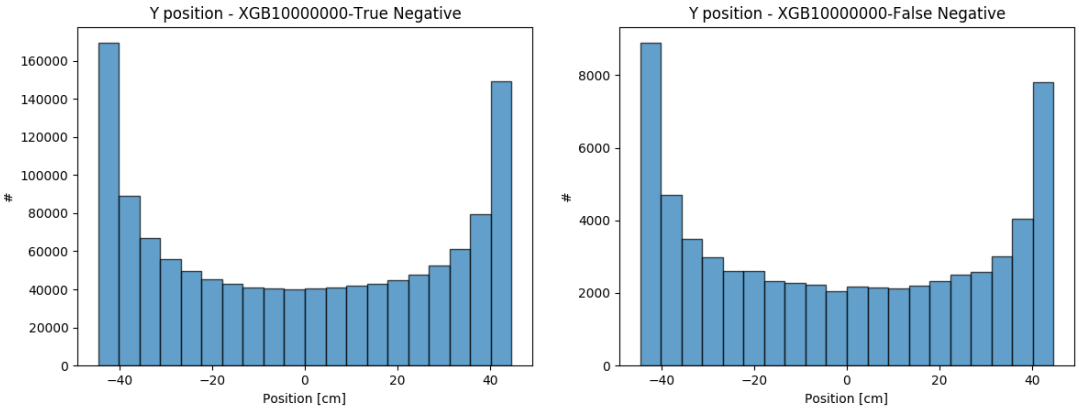
(b) ROC dla danych testowych.

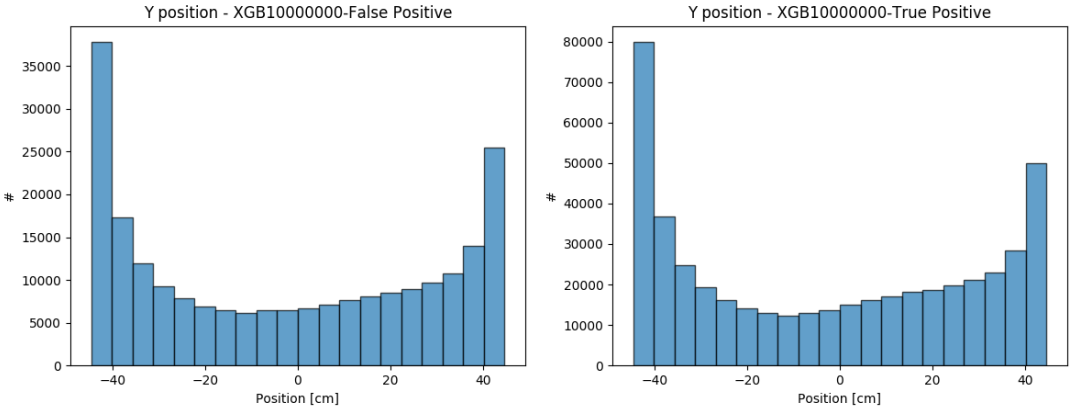
Rysunek 20: Krzywe ROC dla algorytmu XGBoost o najlepszych hiperparametrach.

Rysunki 21, 22, 23, 24 oraz 25 przestawiają statystyki poszczególnych grup klasyfikacyjnych (*True negative*, *False negative*, *False positive*, *True positive*) algorytmu XGBoost o najlepszych hiperparametrach dla atrybutów miejsca detekcji, różnicy czasów detekcji oraz energii.

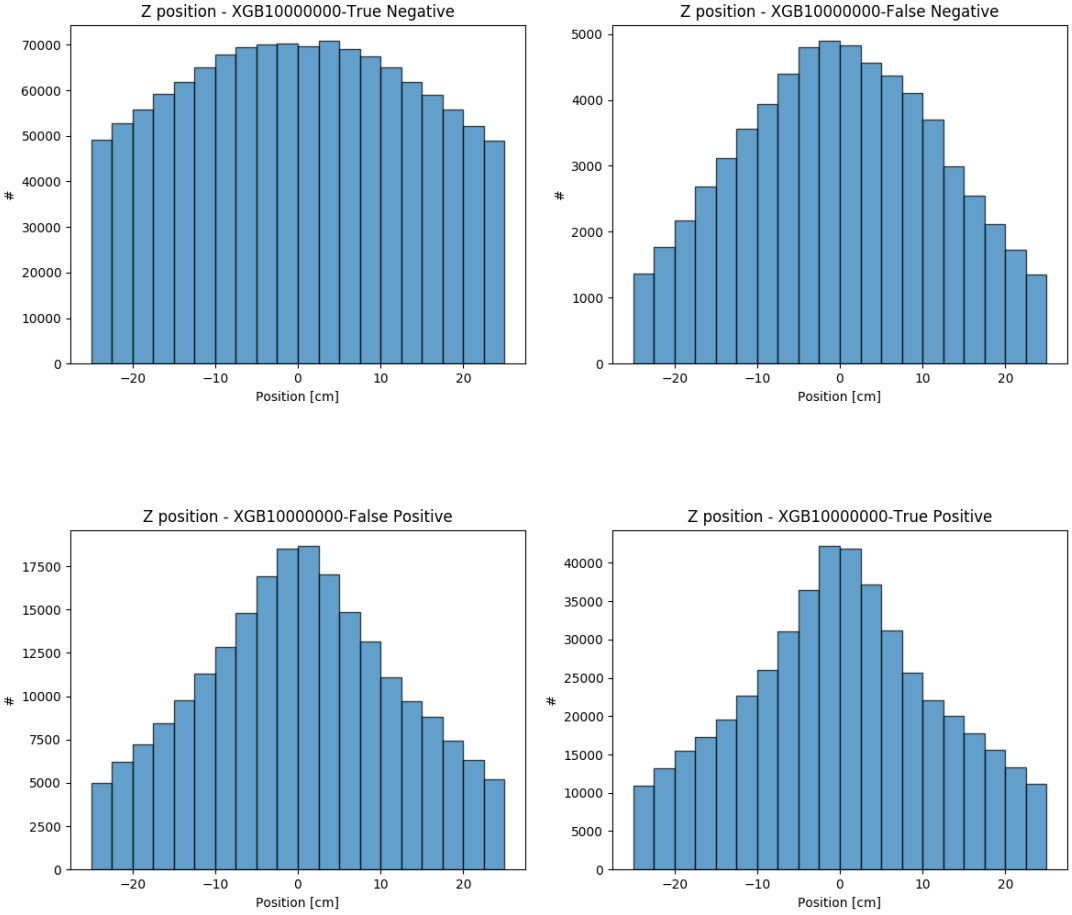


Rysunek 21: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej X detekcji.

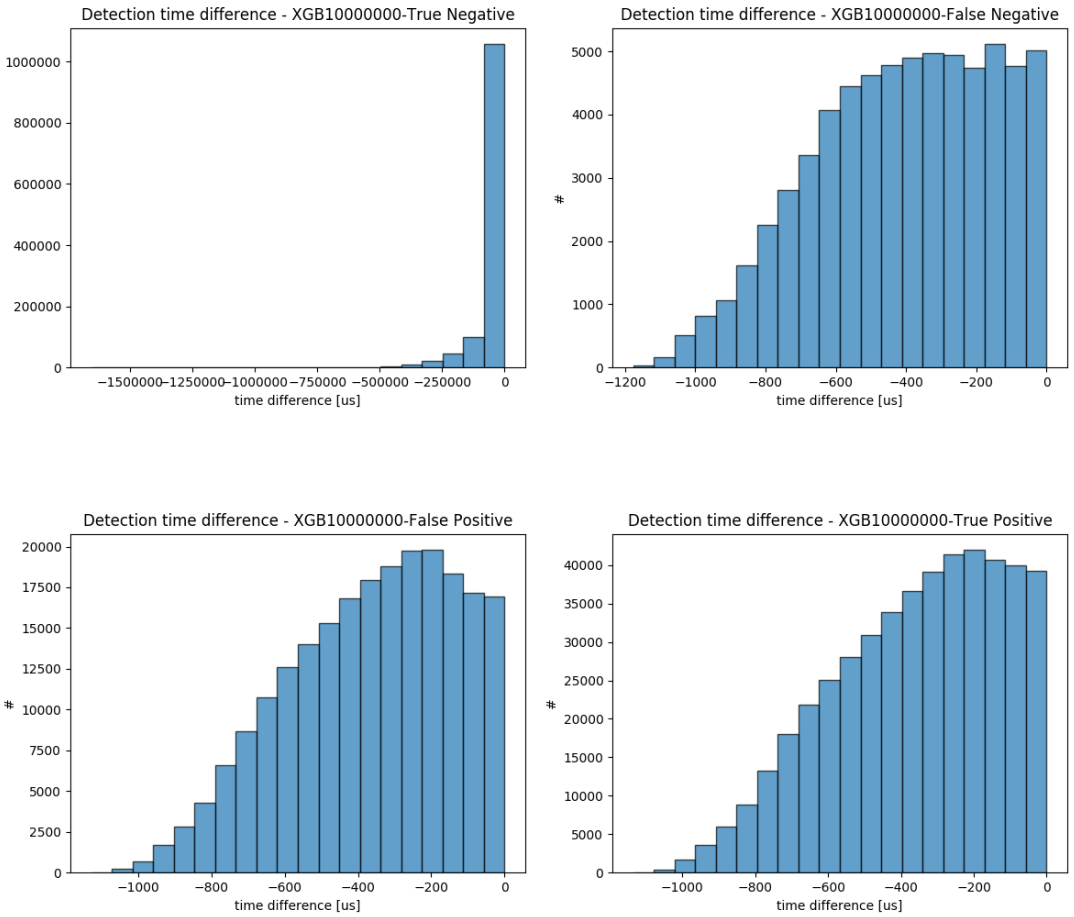




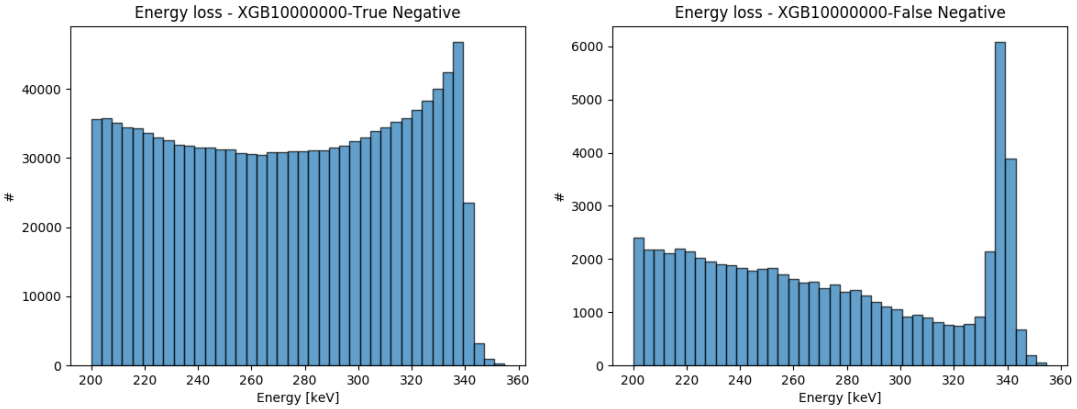
Rysunek 22: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej Y detekcji.

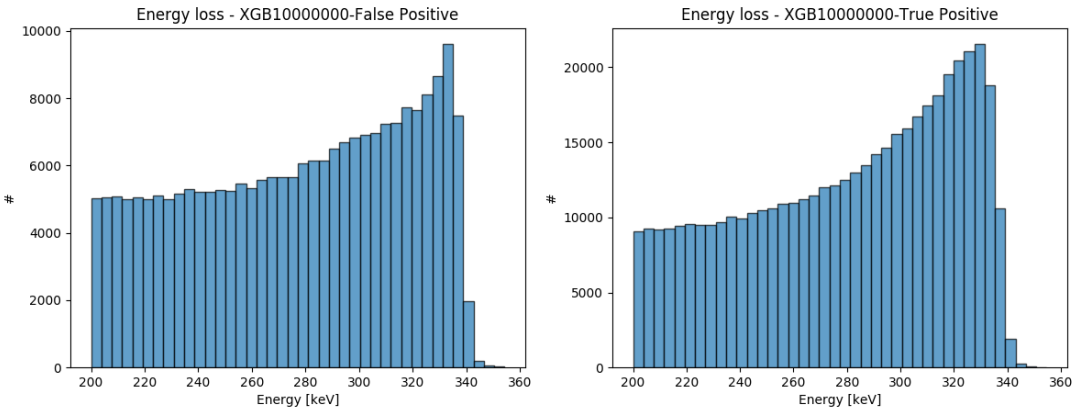


Rysunek 23: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej Z detekcji.



Rysunek 24: Statystyki poszczególnych grup klasyfikacyjnych dla różnic czasów detekcji.

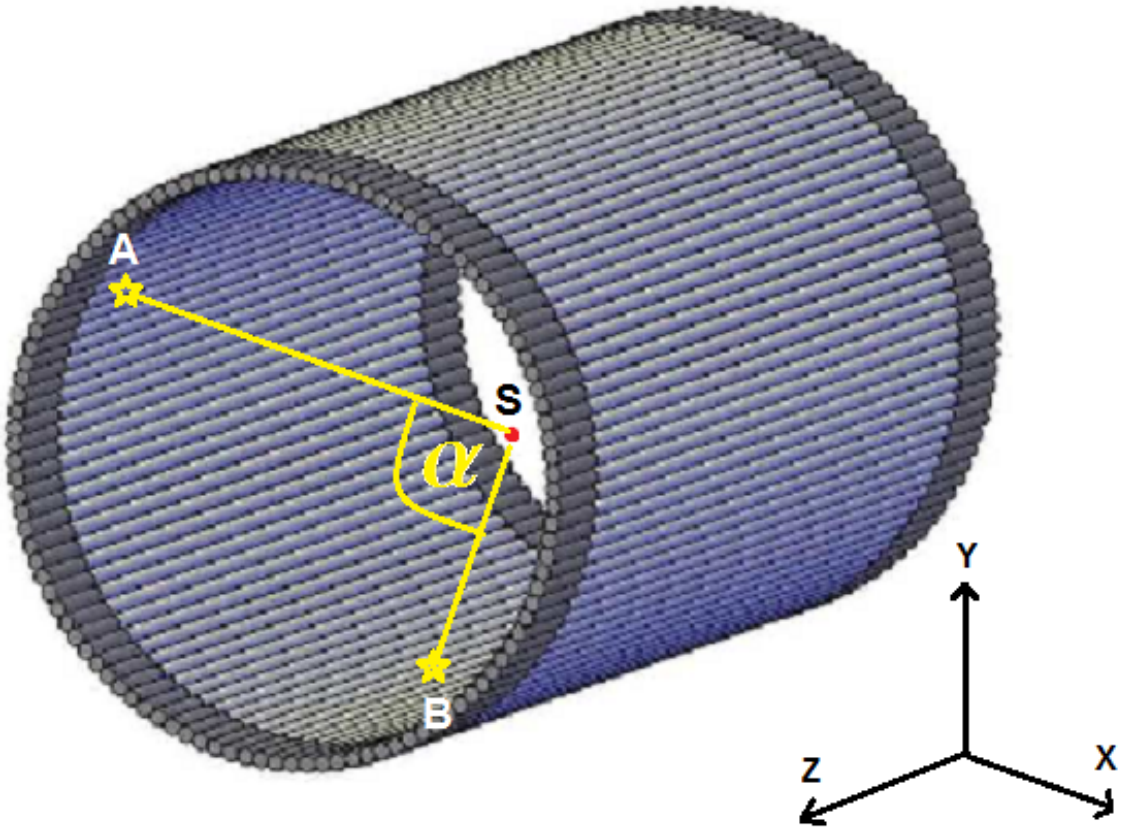




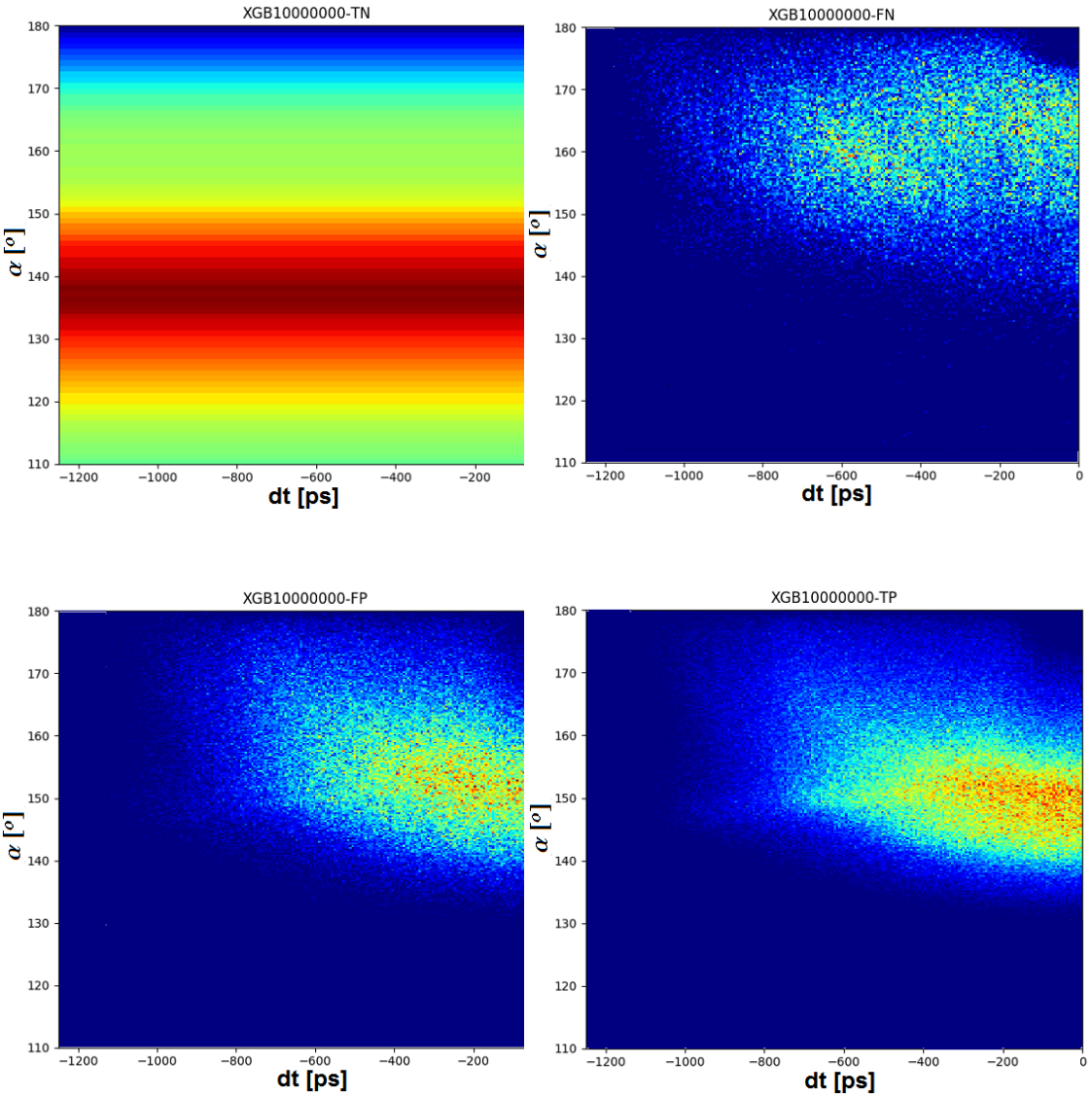
Rysunek 25: Statystyki poszczególnych grup klasyfikacyjnych dla energii detekcji.

Rysunek 27 przedstawia kąt (w stopniach) pomiędzy prostymi łączącymi punkty detekcji ze środkiem detektora dla analizowanych par fotonów w funkcji różnicy czasu detekcji (również dla poszczególnych grup klasyfikacyjnych) opisany wzorem (zgodnie z rysunkiem 26):

$$\alpha = \arccos\left(\frac{A_x * B_x + A_y * B_y + A_z * B_z}{|AS| * |BS|}\right)$$

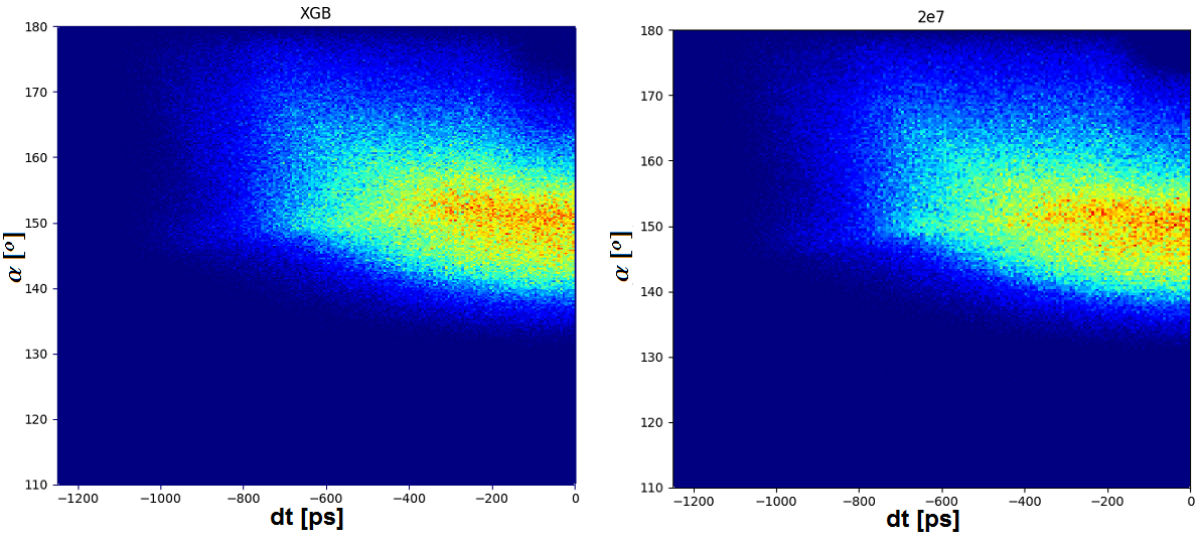


Rysunek 26: Kąt pomiędzy prostymi łączącymi punkty detekcji fotonów z przykładowej pary ze środkiem detektora [6].



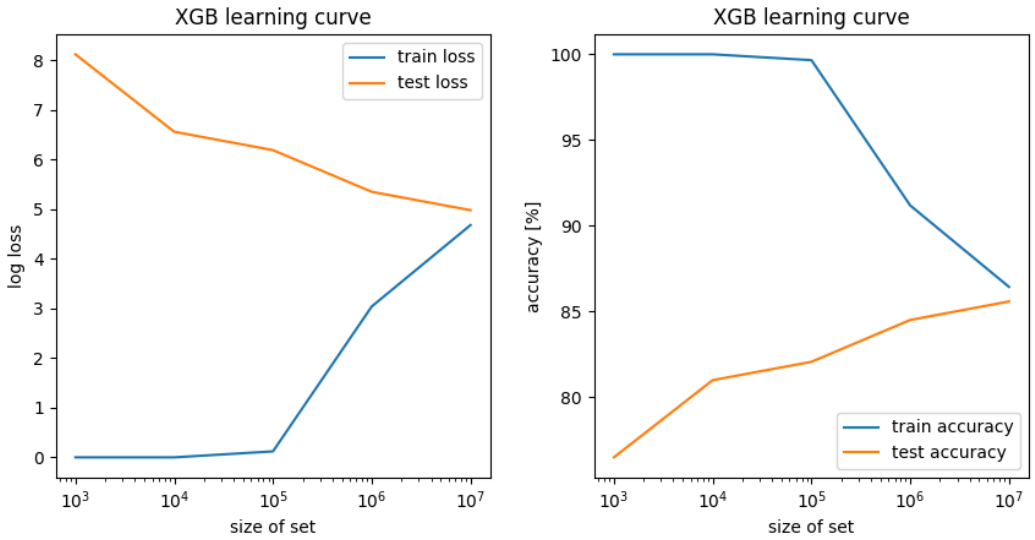
Rysunek 27: Statystyki poszczególnych grup klasyfikacyjnych dla kąta w funkcji różnic czasu.

Po prawej stronie Rysunku 28 widnieje kąt w funkcji różnic czasu detekcji dla wszystkich analizowanych zdarzeń ze zbioru testowego o klasie 1 (klasy opisano na początku rozdziału *Wyniki*), zaś po lewej stronie rysunku widnieje kąt w funkcji różnic czasu detekcji dla wszystkich zdarzeń ze zbioru testowego, które to algorytm XGBoost zakwalifikował jako pozytywne.



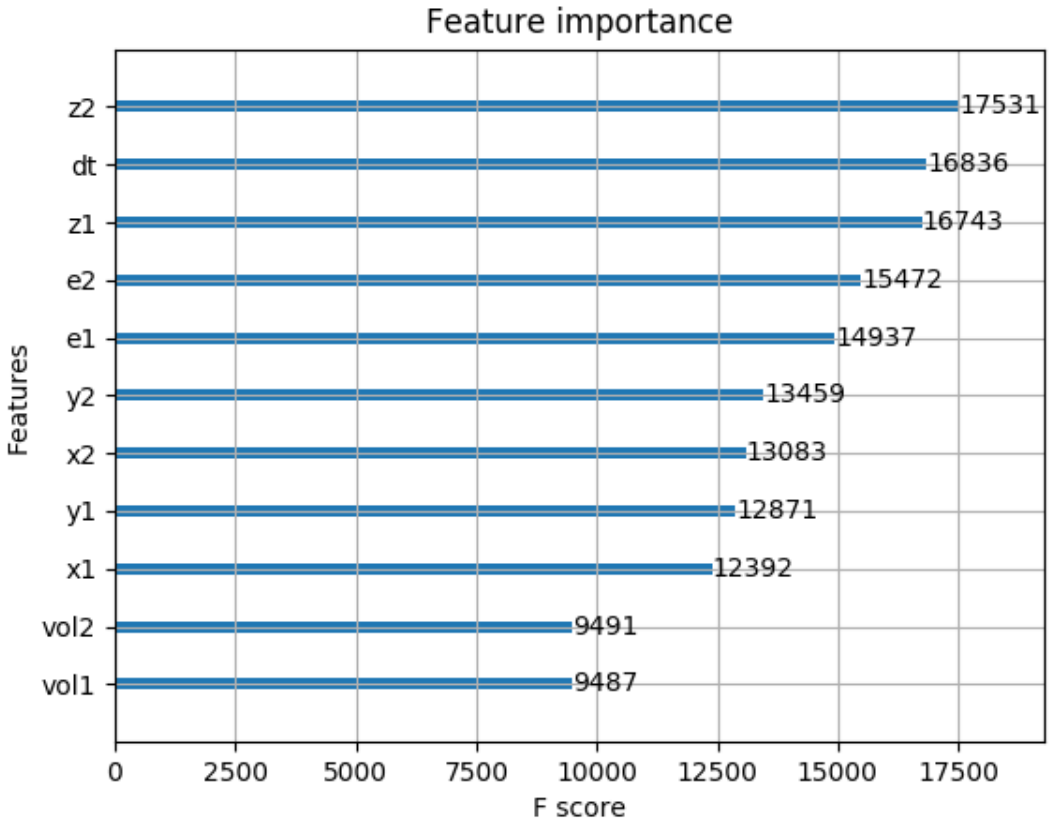
Rysunek 28: Wizualizacja skuteczności klasyfikacji zdarzeń pozytywnych dla kąta w funkcji różnic czasu.

Rysunek 29 przedstawia wartość funkcji kosztu oraz skuteczność klasyfikatora w funkcji rozmiaru zbioru danych (dane testowe + dane treningowe).



Rysunek 29: Krzywa uczenia się (ang. *learning curve*) dla algorytmu XGBoost o najlepszych hiperparametrach.

Rysunek 30 przedstawia wpływ poszczególnych atrybutów na proces uczenia (ang *feature importance* [2]) algorytmem XGBoost o najlepszych hiperparametrach. Ważność atrybutu jest to ilość użyć danego atrybutu przy podziale węzła dla wszystkich drzew w zespole.



Rysunek 30: Wpływ poszczególnych atrybutów na proces uczenia algorytmem XGBoost o najlepszych hiperparametrach.

Po prawej stronie Rysunku 31 widnieje rekonstrukcja źródła promieniowania na podstawie wszystkich analizowanych zdarzeń ze zbioru testowego o klasie 1, zaś po lewej stronie rysunku widnieje rekonstukcja źródła promieniowania na podstawie wszystkich zdarzeń ze zbioru testowego, które to algorytm XGBoost zakwalifikował jako pozytywne.

Składowa x wektora położenia punktu rekonstrukcji obliczana jest dla analizowanych par według wzoru (pozostałe składowe obliczane są analogicznie):

$$x = \frac{x_1 + x_2}{2} - \frac{c\Delta t(x_1 - x_2)}{2\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (y_1 - y_2)^2}},$$

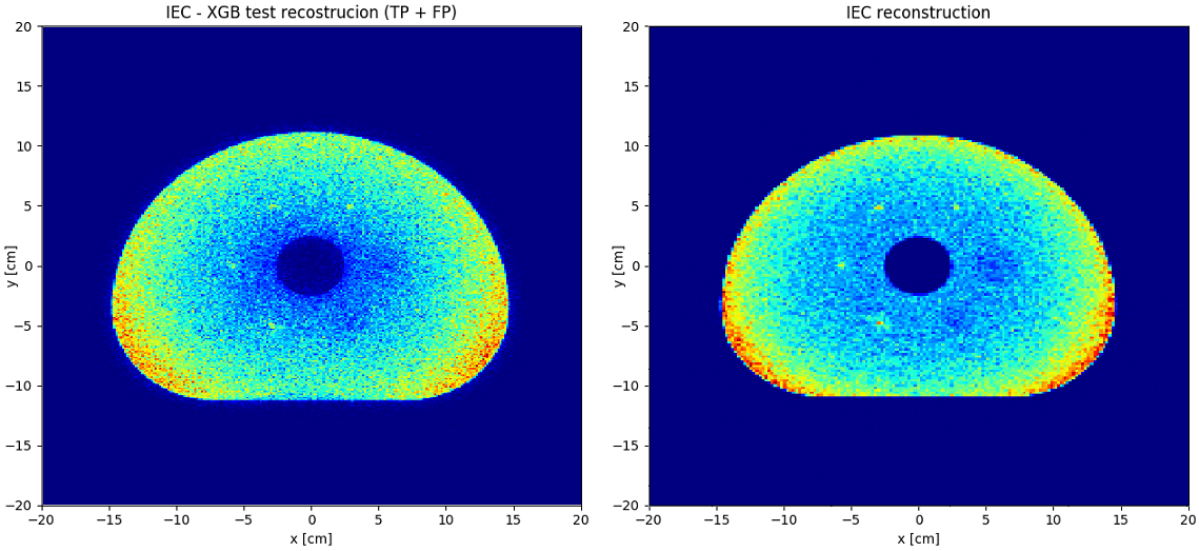
gdzie:

x_1 - współrzędna x detekcji pierwszego fotonu (analagicznie y_1, z_1),

x_2 - współrzędna x detekcji drugiego fotonu (analagicznie y_2, z_2),

c - prędkość światła (przyjęto 0.03 cm/ps),

Δt - różnica pomiędzy czasem detekcji pierwszego oraz drugiego fotonu.



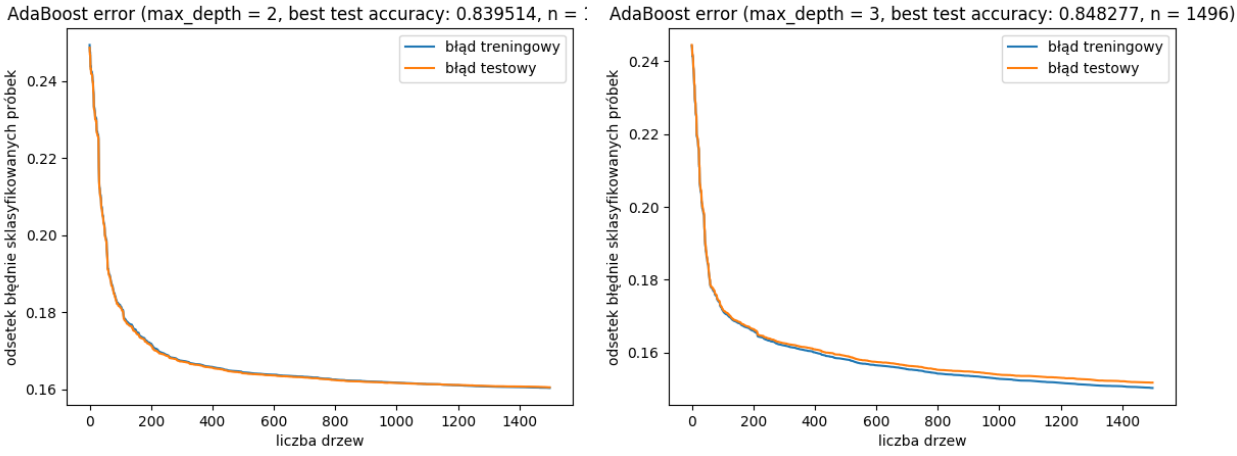
Rysunek 31: Wizualizacja skuteczności klasyfikacji zdarzeń pozytywnych na podstawie rekonstrukcji źródła promieniowania.

2.2 AdaBoost

Do przeprowadzenia klasyfikacji wykorzystano algorytm *AdaBoostClassifier* z pakietu *Python*'a - *sklearn.ensemble* [13] o następujących parametrach[11]:

```
model = AdaBoostClassifier(
  base_estimator = DecisionTreeClassifier(max_depth = max_depth),
  n_estimators = 1500,
  learning_rate = 0.2
)
```

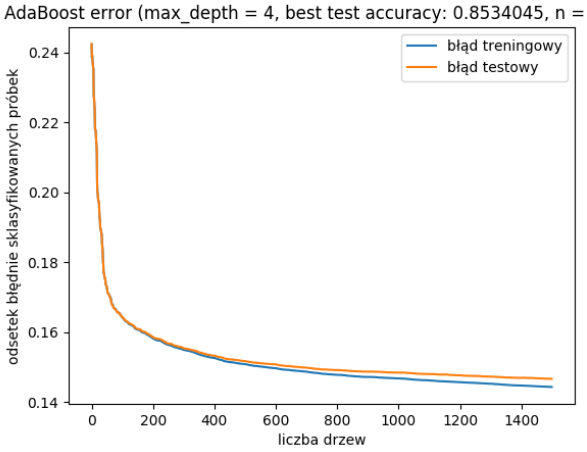
Następnie zmieniano wartość maksymalnej głębokości drzewa i sprawdzono wartości błędów klasyfikatora (dla zestawu treningowego oraz zestawu testowego) w funkcji liczby drzew. Wyniki przedstawiają rysunki 32a-35b.



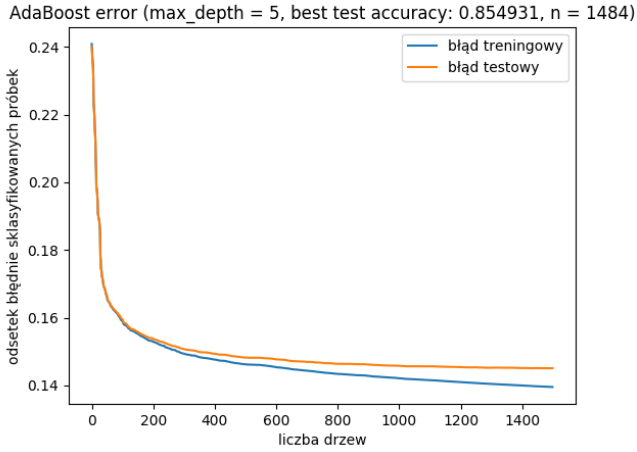
(a) AdaBoost *error* (*max_ depth* = 2).

(b) AdaBoost *error* (*max_ depth* = 3).

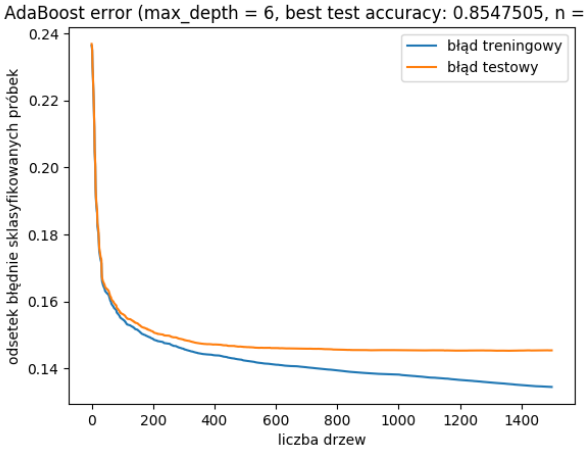
[11] *DecisionTreeClassifier* - klasyfikator na podstawie drzewa decyzyjnego z pakietu *sklearn.tree*.



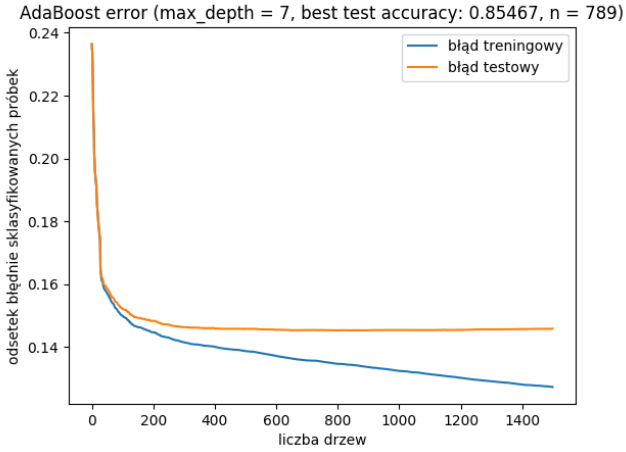
(a) AdaBoost *error* ($max\_depth = 4$).



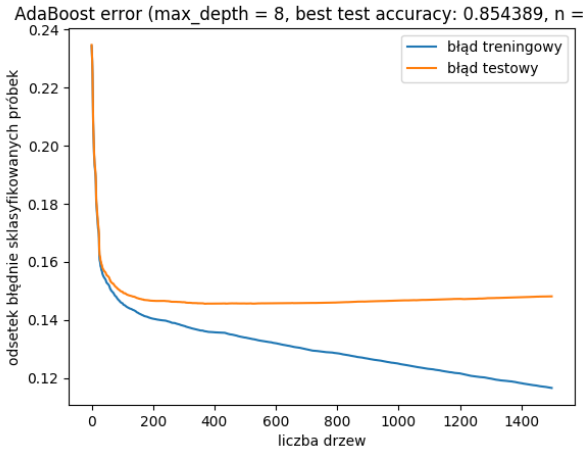
(b) AdaBoost *error* ($max\_depth = 5$).



(a) AdaBoost *error* ($max\_depth = 6$).



(b) AdaBoost *error* ($max\_depth = 7$).

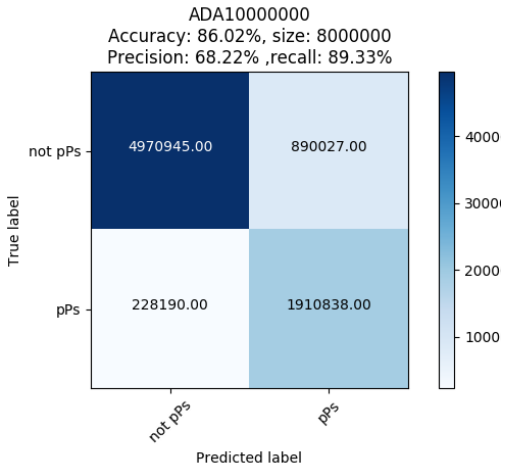


(a) AdaBoost *error* ($max\_depth = 8$).

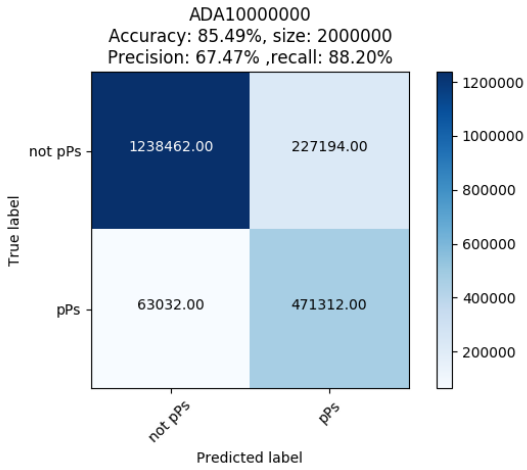
(b) AdaBoost *error* ($max\_depth = 9$).

Następnie, działanie modelu o najlepszych hiperparametrach (na podstawie powyższych wy-

ników wybrano $max\_depth = 5$ oraz $n\_estimators = 1484$) sprawdzono na zestawie danych testowych. Rysunki 36a oraz 36b przedstawiają macierze pomyłek odpowiednio dla danych treningowych oraz testowych, natomiast rysunki 37a oraz 37b przestawiają krzywe ROC odpowiednio dla danych treningowych i testowych.

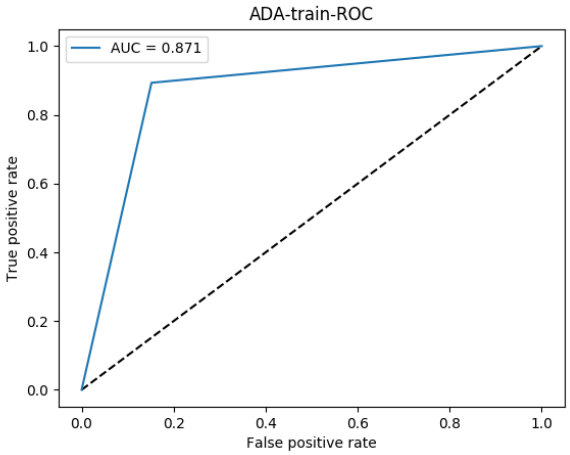


(a) Macierz pomyłek dla danych treningowych.

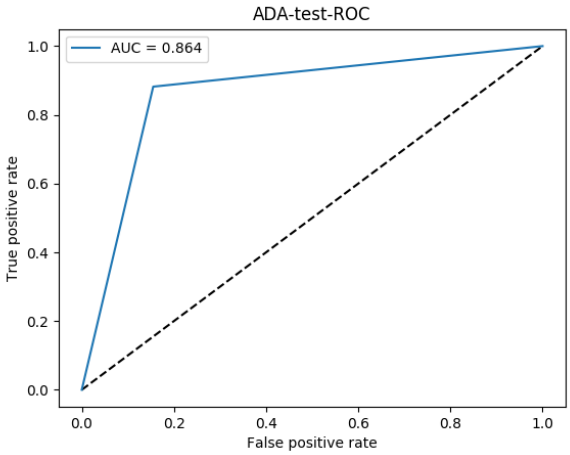


(b) Macierz pomyłek dla danych testowych.

Rysunek 36: Macierze pomyłek dla algorytmu AdaBoost o najlepszych hiperparametrach.



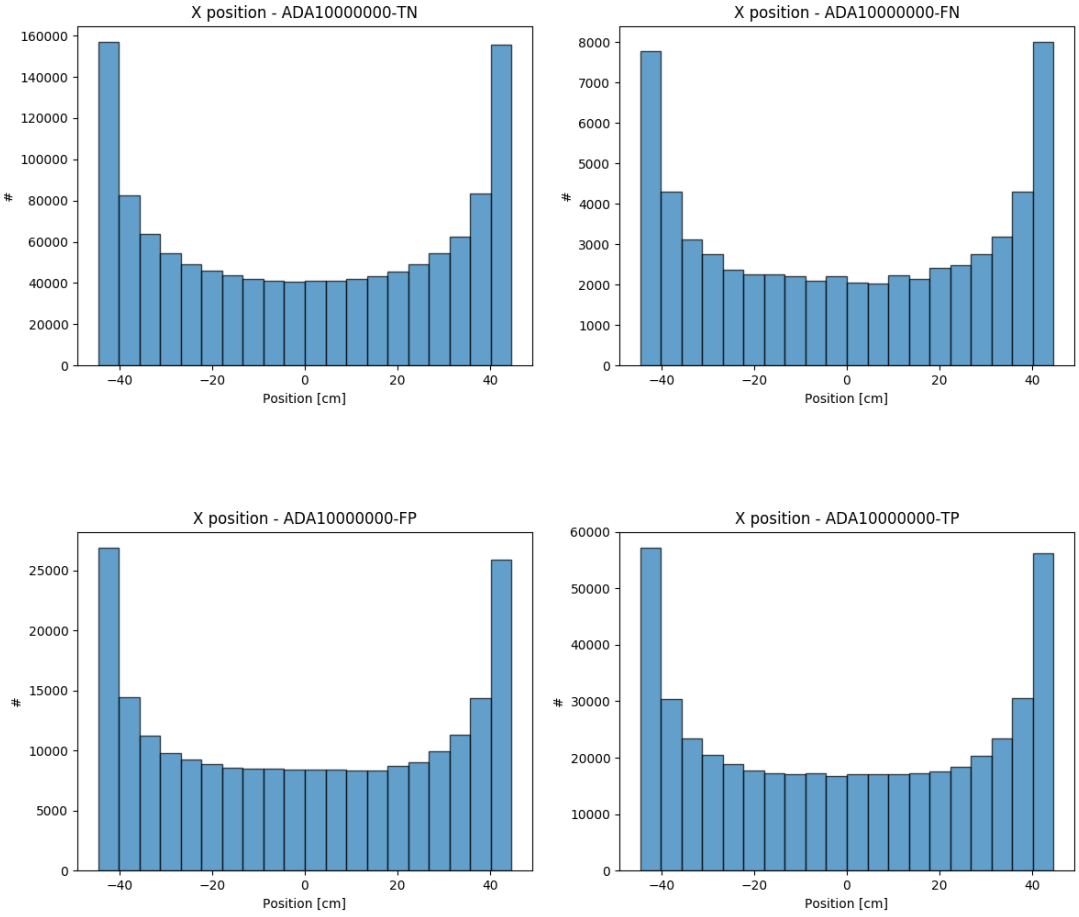
(a) ROC dla danych treningowych.



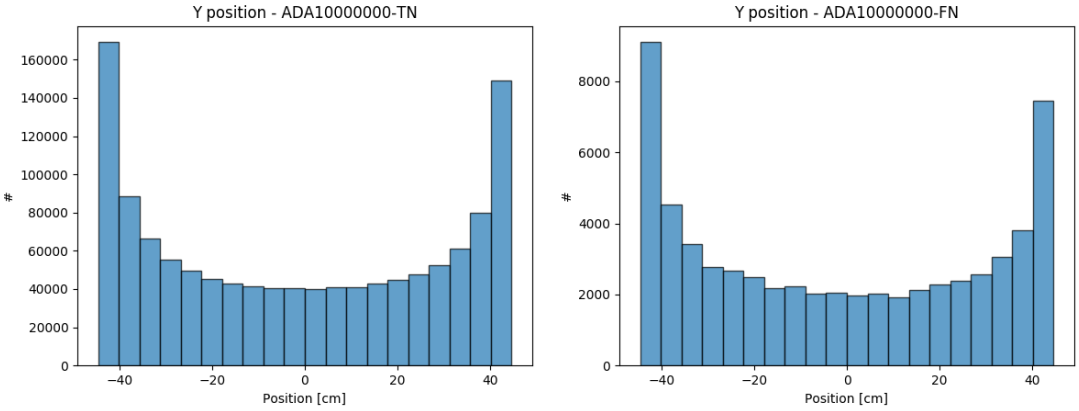
(b) ROC dla danych testowych.

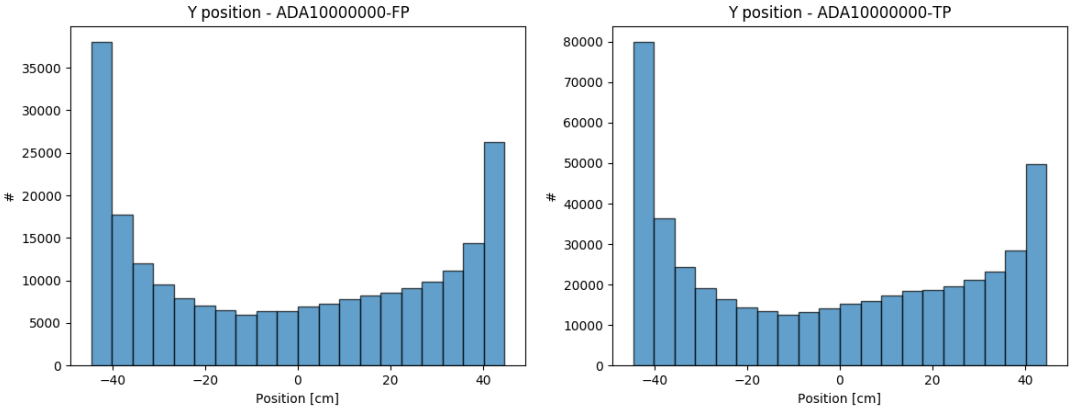
Rysunek 37: Krzywe ROC dla algorytmu AdaBoost o najlepszych hiperparametrach.

Rysunki 38, 39, 40, 41 oraz 42 przestawiają statystyki poszczególnych grup klasyfikacyjnych algorytmu AdaBoost o najlepszych hiperparametrach dla atrybutów miejsca detekcji, różnicy czasów detekcji oraz energii.

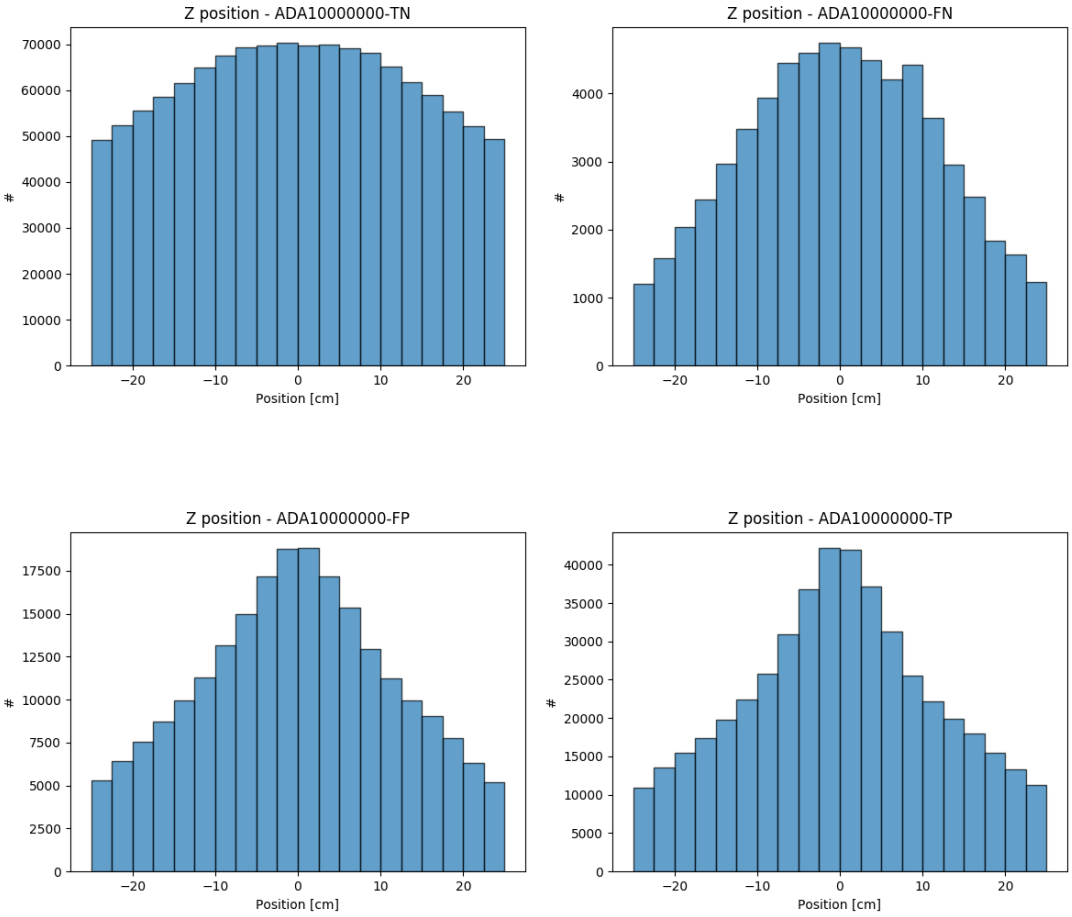


Rysunek 38: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej X detekcji.

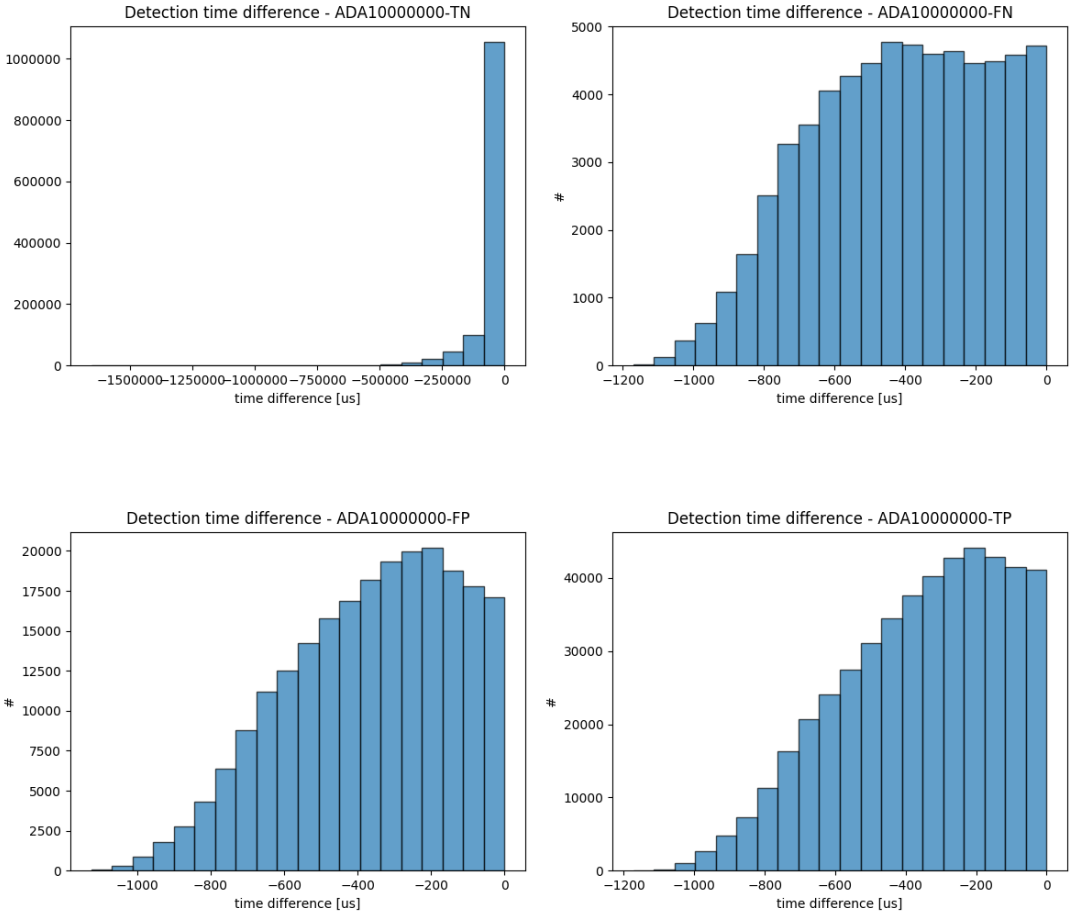




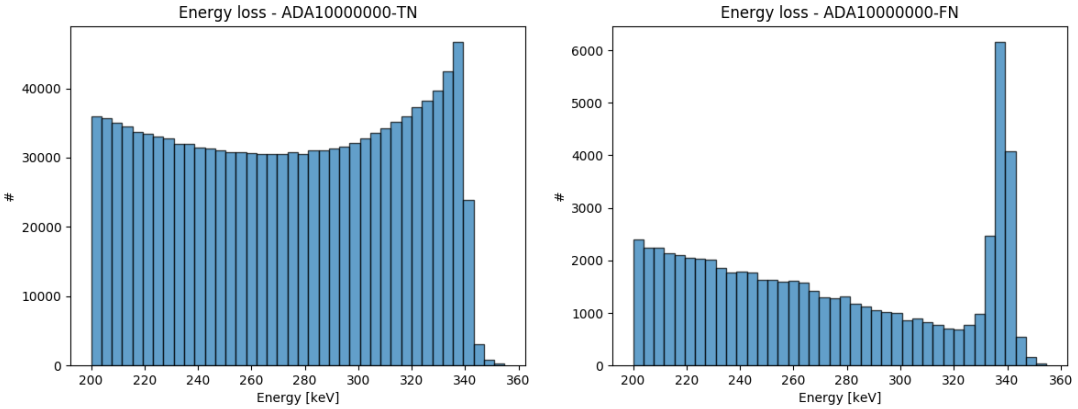
Rysunek 39: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej Y detekcji.

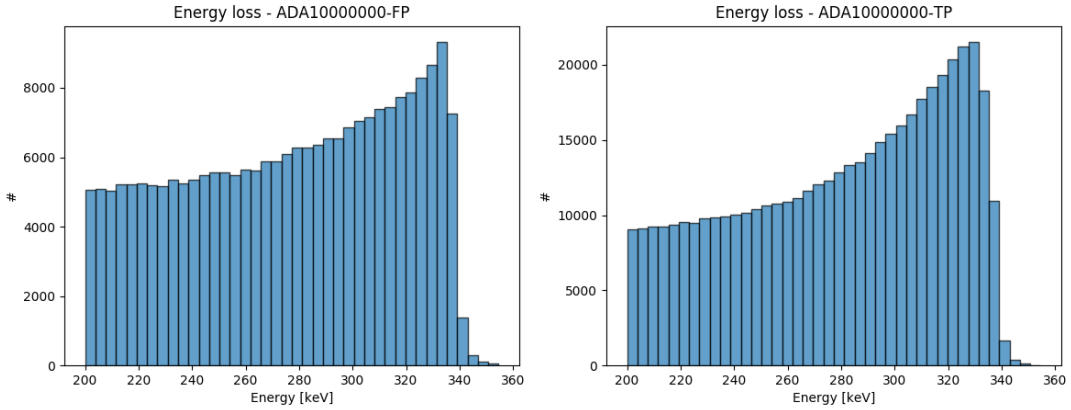


Rysunek 40: Statystyki poszczególnych grup klasyfikacyjnych dla współrzędnej Z detekcji.

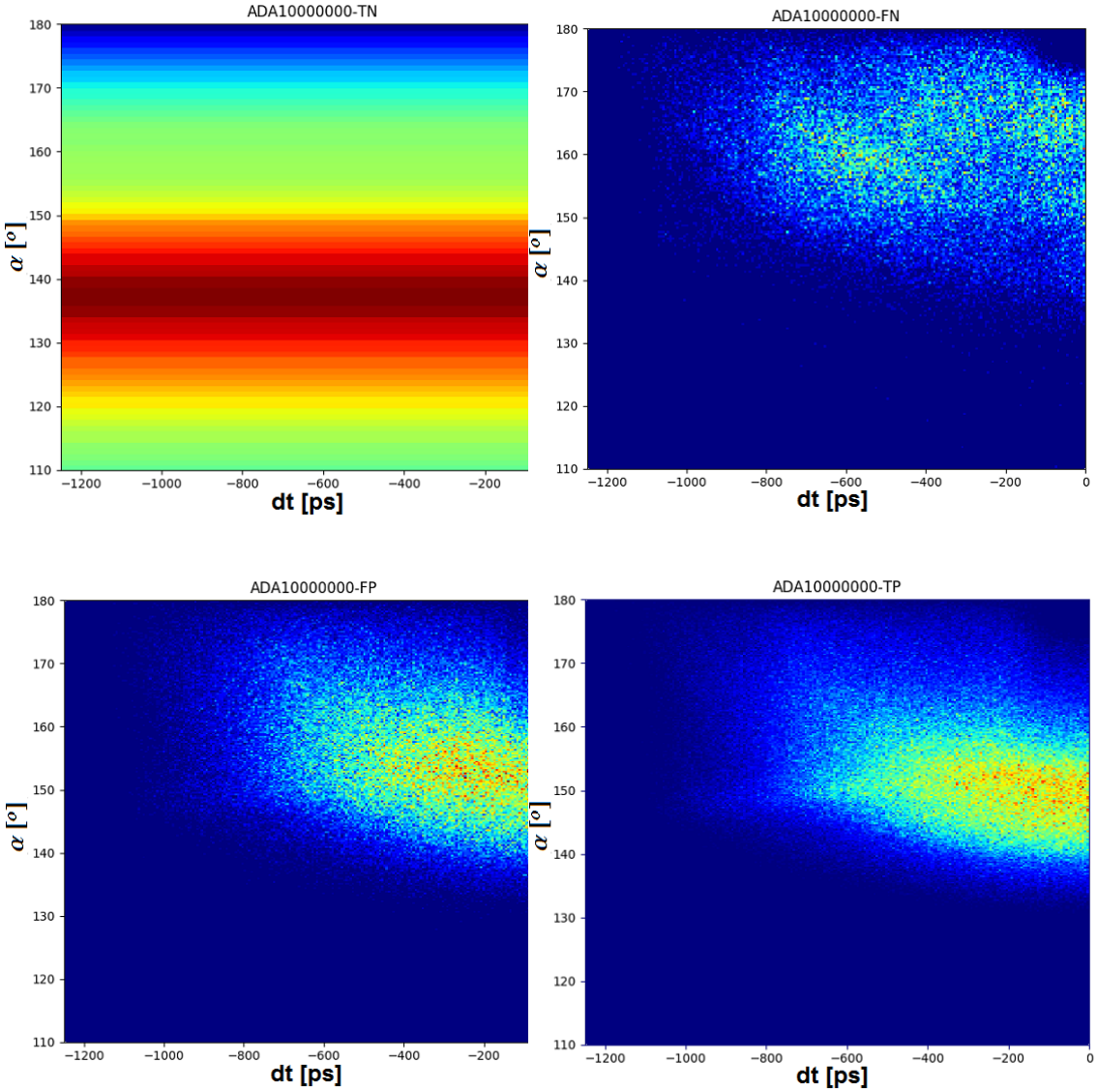


Rysunek 41: Statystyki poszczególnych grup klasyfikacyjnych dla różnic czasów detekcji.





Rysunek 42: Statystyki poszczególnych grup klasyfikacyjnych dla energii detekcji.

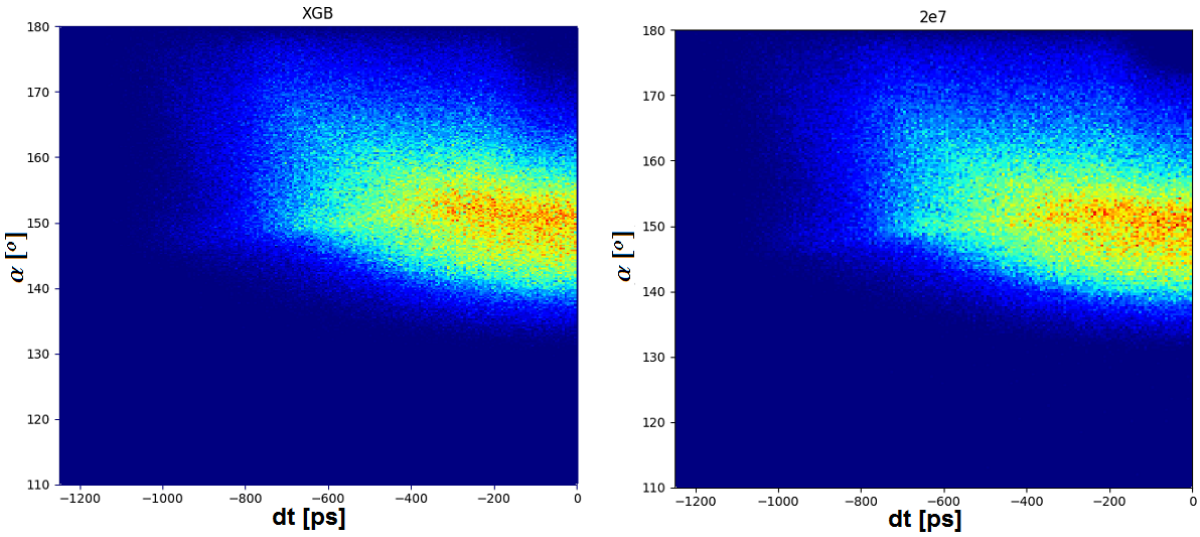


Rysunek 43: Statystyki poszczególnych grup klasyfikacyjnych dla kąta w funkcji różnic czasu.

Rysunek 43 przedstawia kąt (w stopniach) pomiędzy prostymi łączącymi punkty detekcji ze

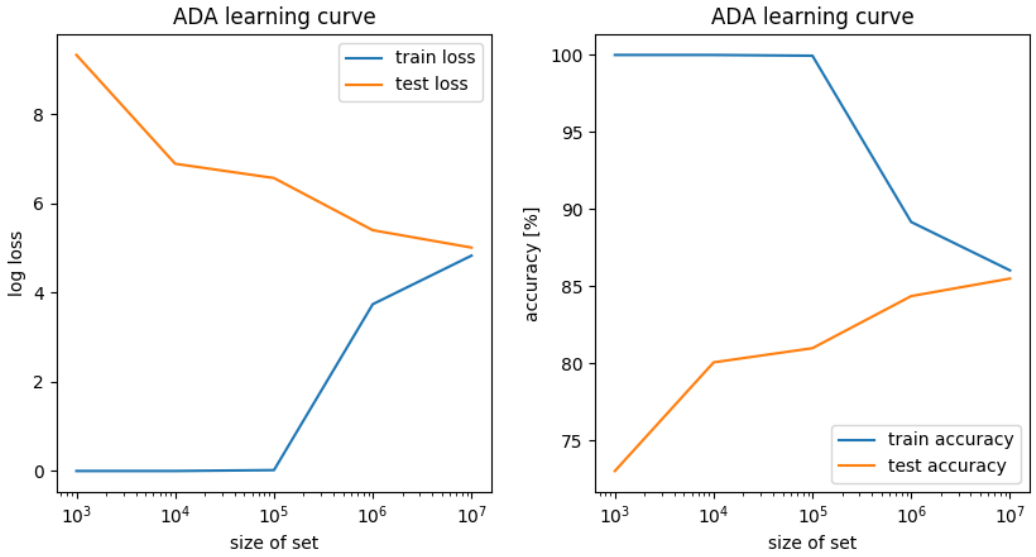
środkiem detektora dla analizowanych par fotonów w funkcji różnicy czasu detekcji (również dla poszczególnych grup klasyfikacyjnych).

Po prawej stronie Rysunku 44 widnieje kąt w funkcji różnic czasu detekcji dla wszystkich analizowanych zdarzeń ze zbioru testowego o klasie 1, zaś po lewej stronie rysunku widnieje kąt w funkcji różnic czasu detekcji dla wszystkich zdarzeń ze zbioru testowego, które to algorytm AdaBoost zakwalifikował jako pozytywne.



Rysunek 44: Wizualizacja skuteczności klasyfikacji zdarzeń pozytywnych dla kąta w funkcji różnic czasu.

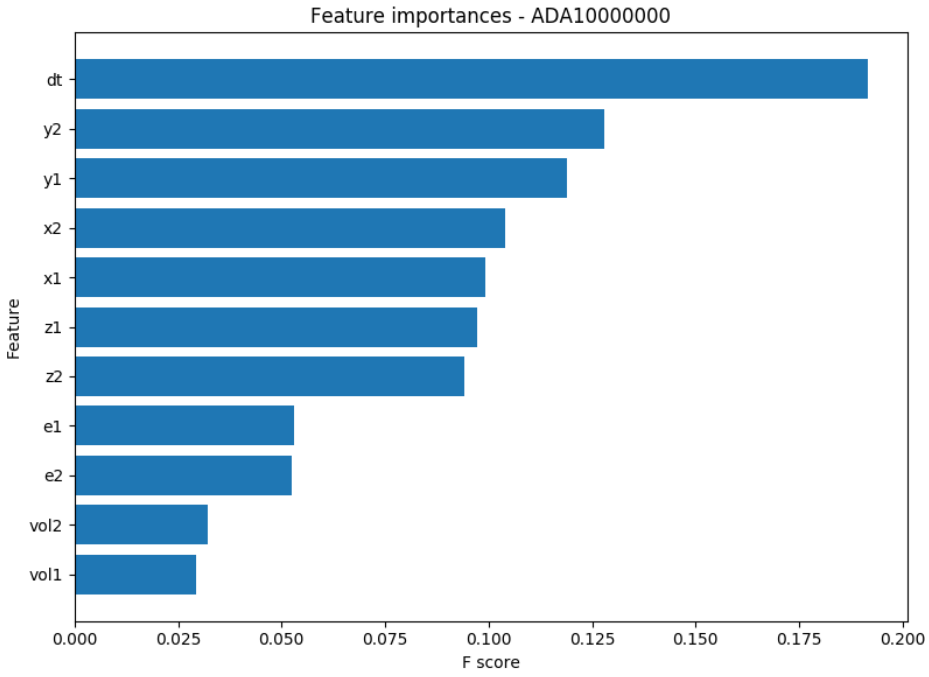
Rysunek 45 przedstawia wartość funkcji kosztu oraz skuteczność klasyfikatora w funkcji rozmiaru zbioru danych (dane testowe + dane treningowe).



Rysunek 45: Krzywa uczenia się (ang. *learning curve*) dla algorytmu AdaBoost o najlepszych hiperparametrach.

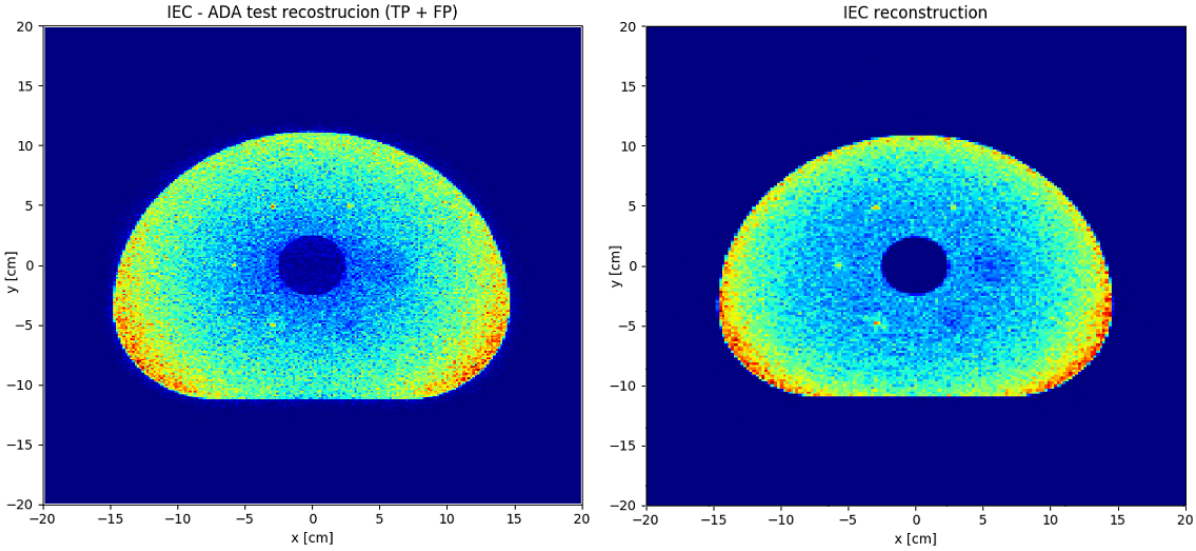
Rysunek 46 przedstawia wpływ poszczególnych atrybutów na proces uczenia algorytmem

AdaBoost o najlepszych hiperparametrach. Ważność danego atrybutu jest liczona jako spadek zanieczyszczenia węzła przy dzieleniu wobec danego atrybutu, ważony przez prawdopodobieństwo dotarcia do tego węzła (odsetek próbek docierających do tego węzła), uśredniony dla wszystkich drzew zespołu [1].



Rysunek 46: Wpływ poszczególnych atrybutów na proces uczenia algorytmem AdaBoost o najlepszych hiperparametrach.

Po prawej stronie Rysunku 47 widnieje rekonstrukcja źródła promieniowania na podstawie wszystkich analizowanych zdarzeń ze zbioru testowego o klasie 1, zaś po lewej stronie rysunku widnieje rekonstrukcja źródła promieniowania na podstawie wszystkich zdarzeń ze zbioru testowego, które to algorytm AdaBoost zakwalifikował jako pozytywne.



Rysunek 47: Wizualizacja skuteczności klasyfikacji zdarzeń pozytywnych na podstawie rekonstrukcji źródła promieniowania.

3. Wnioski

Tabela 1 przedstawia porównanie działania algorytmów XGBoost oraz AdaBoost o najlepszych hiperparametrach dla danych treningowych, a Tabela 2 przedstawia to porównanie dla danych testowych.

Tabela 1. Porównanie działania algorytmów XGBoost oraz AdaBoost o najlepszych hiperparametrach dla danych treningowych.

	TP	FP	TN	FN	ACC	TPR	TNR	PPV	FPR
XGB	1914898	861737	5000334	223031	86,44%	89,57%	85,30%	68,96%	14,70%
ADA	1910838	890027	4970945	228190	86,02%	89,33%	84,81%	68,22%	15,19%

Tabela 2. Porównanie działania algorytmów XGBoost oraz AdaBoost o najlepszych hiperparametrach dla danych testowych.

	TP	FP	TN	FN	ACC	TPR	TNR	PPV	FPR
XGB	470433	223260	1241297	65010	85,59%	87,86%	84,76%	67,82%	15,24%
ADA	471312	227194	1238462	63032	85,49%	88,20%	84,50%	67,47%	15,50%

Oba algorytmy osiągnęły podobne rezultaty, ze wskazaniem na algorytm XGBoost. Warto podkreślić, że dane, których użyto do analizy, pochodzą z symulacji i nie uwzględniają niepewności pomiarowych - są to dane bez tzw. rozmycia (ang. *smear*).

Na podstawie wykresów skuteczności klasyfikatorów w funkcji liczby drzew dla różnych, maksymalnych głębokości drzewa można zauważyć, że wariancja modelu na podstawie algorytmu AdaBoost wyraźnie wzrasta przy zwiększaniu maksymalnej głębokości drzewa. Mocniej regularyzowany model oparty o algorytm XGBoost nie doznaje, aż tak dużego wzrostu wariancji i cechuje się mniejszym przetrenowaniem.

4. Skrypty

Skrypt 1: Wizualizacja przebiegu wzmacniania gradientowego.

```
#!/usr/bin/env python3.6

import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

size = 100
np.random.seed(1)
X = np.arange(-0.5, 0.5, 0.01)[:, np.newaxis]
Y = []

for i in range(size):
Y.append(3*X[i]**2 + 0.15*np.random.rand())

tree = DecisionTreeRegressor(max_depth = 2)
tree.fit(X, Y)

X_test = np.arange(-0.5, 0.5, 0.01)[:, np.newaxis]
Y_test = tree.predict(X_test)

plt.scatter(X, Y, s = 7, label = "dane uczace")
plt.plot(X_test, Y_test, color = "green", label = "n = 1")
plt.xlim([-0.5, 0.5])
plt.ylim([0, 0.85])
plt.legend(loc = "upper center")
plt.savefig("n1.png")
plt.clf()

Y2 = []

for i in range(size):
Y2.append(Y[i] - Y_test[i])

tree.fit(X, Y2)
Y2_test = tree.predict(X_test)

plt.scatter(X, Y2, s = 7, label = "wartosci resztowe")
plt.plot(X_test, Y2_test, color = "green", label = "n = 2")
plt.xlim([-0.5, 0.5])
plt.ylim([-0.4, 0.45])
plt.legend(loc = "upper center")
plt.savefig("r1.png")
plt.clf()

Y12 = []

for i in range(size):
```

```
Y12.append(Y_test[i] + Y2_test[i])

plt.scatter(X, Y, s = 7, label = "dane uczace")
plt.plot(X_test, Y12, color = "green", label = "n = 2")
plt.xlim([-0.5, 0.5])
plt.ylim([0, 0.85])
plt.legend(loc = "upper center")
plt.savefig("n12.png")
plt.clf()

Y3 = []

for i in range(size):
Y3.append(Y2[i] - Y2_test[i])

tree.fit(X, Y3)
Y3_test = tree.predict(X_test)

plt.scatter(X, Y3, s = 7, label = "wartosci resztowe")
plt.plot(X_test, Y3_test, color = "green", label = "n = 3")
plt.xlim([-0.5, 0.5])
plt.ylim([-0.4, 0.45])
plt.legend(loc = "upper center")
plt.savefig("r2.png")
plt.clf()

Y123 = []

for i in range(size):
Y123.append(Y_test[i] + Y2_test[i] + Y3_test[i])

plt.scatter(X, Y, s = 7, label = "dane uczace")
plt.plot(X_test, Y123, color = "green", label = "n = 3")
plt.xlim([-0.5, 0.5])
plt.ylim([0, 0.85])
plt.legend(loc = "upper center")
plt.savefig("n123.png")
plt.clf()
```

Skrypt 2: Wizualizacja regresji logistycznej dla klasyfikacji binernej.

```
#!/usr/bin/env python3.6

from sklearn import datasets
import numpy as np
from sklearn.linear_model import LogisticRegression
import matplotlib.pyplot as plt

iris = datasets.load_iris()
X = iris['data'][:, 3:]
y = (iris['target'] == 2).astype(np.int)
```

```
  log_reg = LogisticRegression()
  log_reg.fit(X, y)
  X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
  y_proba = log_reg.predict_proba(X_new)
  plt.plot(X_new, y_proba[:,1], "g-",
  label = "prawdopodobienstwo dla Iris-Virginica")
  plt.plot(X_new, y_proba[:,0], "b--",
  label = "prawdopodobienstwo dla pozostalych")
  IV = (iris['target'] == 2) # Iris-virginica
  plt.scatter(X[IV], np.full((X[IV].size), 1),
  label = "Iris-Virginica")
  plt.scatter(X[~IV], np.full((X[~IV].size), 0),
  label = "pozostale")
  plt.plot([1.615, 1.615], [-1, 2], "b--", c = "red",
  label = "Granica decyzyjna")
  plt.ylim([-0.05, 1.05])
  plt.xlabel("Szerokosc platka [cm]")
  plt.ylabel("Prawdopodobienstwo")
  plt.legend()
  plt.show()
```

Skrypt 3: Wizualizacja algorytmu AdaBoost.

```
#!/usr/bin/env python3.6

import math
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import DecisionTreeRegressor

def probabilitiesSections(weights):
  weight = 0
  prop = np.full((weights.size, 1), 1.0)

  for i in range(weights.size):
    prop[i] = weight

  weight += weights[i]

  return (prop/weight)

def getSampleIndex(probabilities):
  rand = np.random.rand()

  for i in range(probabilities.size):
    if rand < probabilities[i]:
      return i

  return (probabilities.size - 1)

def reSample(X, Y, probabilities):
```

```
  size = X.size
  XreSampled = np.full((size, 1), 0.0)
  YreSampled = np.full((size, 1), 0.0)
  indexReSampled = np.full((size, 1), 0)

  for i in range(size):
    index = getSampleIndex(probabilities)
    indexReSampled[i] = index
    XreSampled[i] = X[index]
    YreSampled[i] = Y[index]

  return XreSampled, YreSampled, indexReSampled

def calcLoss(Y, Y_pred, P_sections):
  size = Y.size
  L = np.full((size, 1), 0.0)
  Lmean = 0.0

  for i in range(size):
    L[i] = abs(Y[i] - Y_pred[i])

  L = L/max(L)

  for i in range(size):
    if i > size - 2:
      Lmean += (1.0 - P_sections[i])*L[i]
    else:
      Lmean += (P_sections[i+1] - P_sections[i])*L[i]

  return L, Lmean

def updateWeights(W, B, L_res, index_res, learningRate):
  size = W.size
  Wupdated = np.copy(W)

  for i in range(size):
    index = index_res[i]
    Wupdated[index] = W[index]*B**(learningRate*L_res[i])

  return Wupdated/np.sum(Wupdated)

def predict(Y_steps_pred, beta, step, size):
  Y_pred = np.full((size, 1), 0.0)
  betaNorm = 0.0

  for i in range(step):
    betaNorm += math.log(1/beta[i])

  for i in range(size):
    for j in range(step):
```

```
      Y_pred[i] += math.log(1/beta[j])*Y_steps_pred[i,j]/betaNorm

  return Y_pred


def adaBoost(X, Y, estimators, maxDepth, learningRate = 1.0, printRate = 0):
  size = X.size
  W = np.full((size, 1), 1.0)
  P_sections = probabilitiesSections(W)
  B = np.full((estimators, 1), 1.0)
  Y_steps_pred = np.full((size, estimators), 1.0)

  for i in range(estimators):
    X_res, Y_res, index_res = reSample(X, Y, P_sections)

  tree = DecisionTreeRegressor(max_depth = maxDepth)
  tree.fit(X_res, Y_res)
  Y_steps_pred[:,i] = tree.predict(X)
  L_res, Lmean = calcLoss(Y, Y_steps_pred[:,i], P_sections)
  B[i] = Lmean/(1 - Lmean)
  Y_pred = predict(Y_steps_pred, beta = B, step = i+1, size = size)

  if printRate != 0 and (i%printRate == 0 or i == estimators - 1):
    plt.scatter(X, Y, s = 7, label = "dane uczace")
    plt.plot(X, Y_pred, color = "green", label = "n = " + str(i+1))
    plt.xlim([-0.5, 0.5])
    plt.ylim([0, 0.85])
    plt.legend(loc = "upper center")
    plt.savefig("adaBoostN" + str(i+1) + ".png")
    plt.clf()

  W = updateWeights(W, B[i], L_res, index_res, learningRate)
  P_sections = probabilitiesSections(W)

size = 100
np.random.seed(1)
X = np.arange(-0.5, 0.5, 0.01)[:, np.newaxis]
Y = np.full((size, 1), 0.0)

for i in range(size):
  Y[i] = 3*X[i]**2 + 0.15*np.random.rand()

adaBoost(
  X, Y,
  estimators = 50,
  maxDepth = 5,
  learningRate = 0.2,
  printRate = 10
)
```

Bibliografia

[1] Leo Breiman. *Classification and Regression Trees*. 1984. URL: https://doi.org/10.1201/9781315139470.

[2] xgboost developers. *Feature importance*. URL: https://xgboost.readthedocs.io/en/latest/R-package/discoverYourData.html#feature-importance.

[3] xgboost developers. *XGBoost Documentation*. URL: https://xgboost.readthedocs.io/en/latest/index.html.

[4] Harris Drucker. *Improving Regressors using Boosting Techniques*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.314&rep=rep1&type=pdf.

[5] Aurélien Géron. *Uczenia maszynowe z użyciem Scikit-Learn i TensorFlow*. HELION, 2018. ISBN: 9788328343733.

[6] K. Dulski Anna Wieczorek i in. *Novel scintillating material 2-(4-styrylphenyl)benzoxazole for the fully digital and MRI compatible J-PET tomograph based on plastic scintillators*. URL: https://www.ncbi.nlm.nih.gov/pubmed/29176834.

[7] Paweł Kowalski i in. *Estimating the NEMA characteristics of the J-PET tomograph using the GATE package*. URL: https://www.ncbi.nlm.nih.gov/pubmed/29992906.

[8] Paweł Moskal i in. *J-PET (Jagiellonian-PET TOMOGRAPHY)*. URL: http://koza.if.uj.edu.pl/pet/.

[9] Wojciech Wiślicki i in. *J-PET NCBJ*. URL: http://pet.ncbj.gov.pl.

[10] Ran Gilad-Bachrach K. V. Rashmi. *DART: Dropouts meet Multiple Additive Regression Trees*. URL: http://proceedings.mlr.press/v38/korlakaivinayak15.pdf.

[11] Paweł Kowalski. *GOJA*. URL: https://github.com/JPETTomography/j-pet-gate-tools/tree/master/goja.

[12] OpenGATE. *GATE*. URL: http://www.opengatecollaboration.org/.

[13] F. Pedregosa i in. "Scikit-learn: Machine Learning in Python". W: *Journal of Machine Learning Research* 12 (2011), s. 2825–2830.

[14] wikipedia.org. *Cross-validation*. URL: https://en.wikipedia.org/wiki/Cross-validation.

[15] wikipedia.org. *Drzewo decyzyjne*. URL: https://pl.wikipedia.org/wiki/Drzewo_decyzyjne.

[16] wikipedia.org. *Receiver operating characteristic*. URL: https://en.wikipedia.org/wiki/Receiver_operating_characteristic.