Algorytmy Uczenia Maszynowego

Klasyfikacja liter- sprawozdanie

Paweł Prucnal 248937

1. **Opis problemu**

Za problem projektu obrano klasyfikację liter przy użyciu kilku algorytmów uczenia maszynowego. Litery opisano szeregiem cech, które potem przekazano na wejście programu. Model nauczony na bazie tychże cech miałby być w stanie zdecydować, jaką literę opisuje zbiór cech, odmienny od tych użytych przy uczeniu modelu.

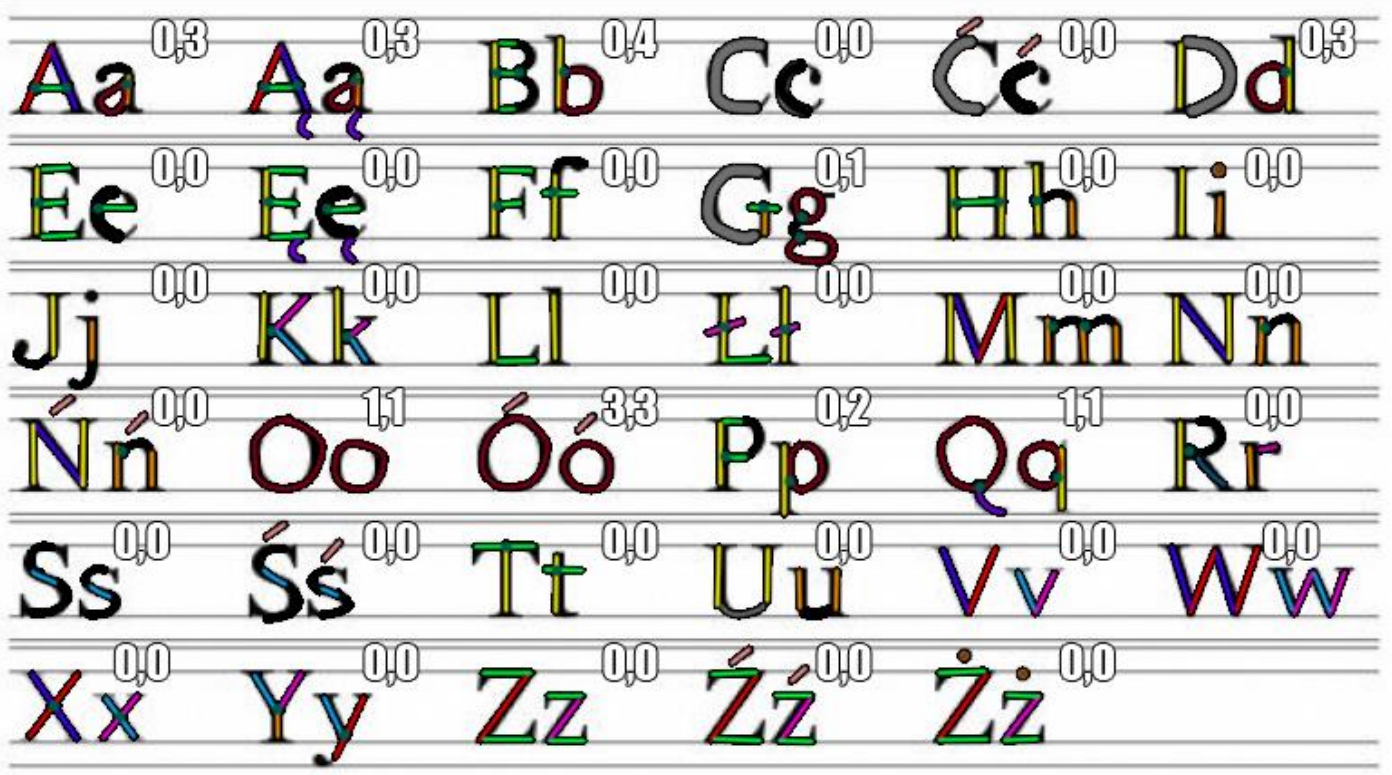
2. **Cechy oraz nauczanie**

Celem utworzenia listy cech, korzystając z programu graficznego zaznaczono na każdej z liter cechy, które mogłyby zostać wykryte w rozsądny sposób przez algorytm. Zdefiniowano tym sposobem następujące cechy:

- **Duże linie ukośne (w prawo)**- zajmujące zauważalnie więcej niż połowę wysokości litery oraz skierowane między południowym zachodem a północnym wschodem
- **Małe linie ukośne (w prawo)**- pozostałe linie skierowane między południowym zachodem a północnym wschodem
- **Duże linie ukośne (w lewo)**- zajmujące zauważalnie więcej niż połowę wysokości litery oraz skierowane z północnego zachodu na południowy wschód
- **Małe linie ukośne (w lewo)**- pozostałe linie skierowane z północnego zachodu na południowy wschód
- **Linie poziome**- dowolnego rozmiaru linie poziome
- **Duże linie pionowe-** zajmujące zauważalnie więcej niż połowę wysokości linie pionowe
- **Małe linie pionowe-** pozostałe linie pionowe
- **Ogonek-** obecność ogonka
- **Kreska-** obecność kreski
- **Kropka-** obecność kropki
- **Duże łuki-** zajmujące zauważalnie więcej niż połowę wysokości łuki o dowolnej orientacji
- **Małe łuki-** pozostałe łuki
- **Okręgi-** liczba okręgów
- **Pozycja okręgu-** 0 gdy brak okręgu, 1 gdy okrąg znajduje się w lewej górnej części litery, 2 gdy okrąg w prawej górnej, 3 gdy w lewej dolnej, 4 gdy w prawej dolnej.
- **Przecięcia-** liczba przecięć linii (punktów, z których wychodzą przynajmniej 3 linie)

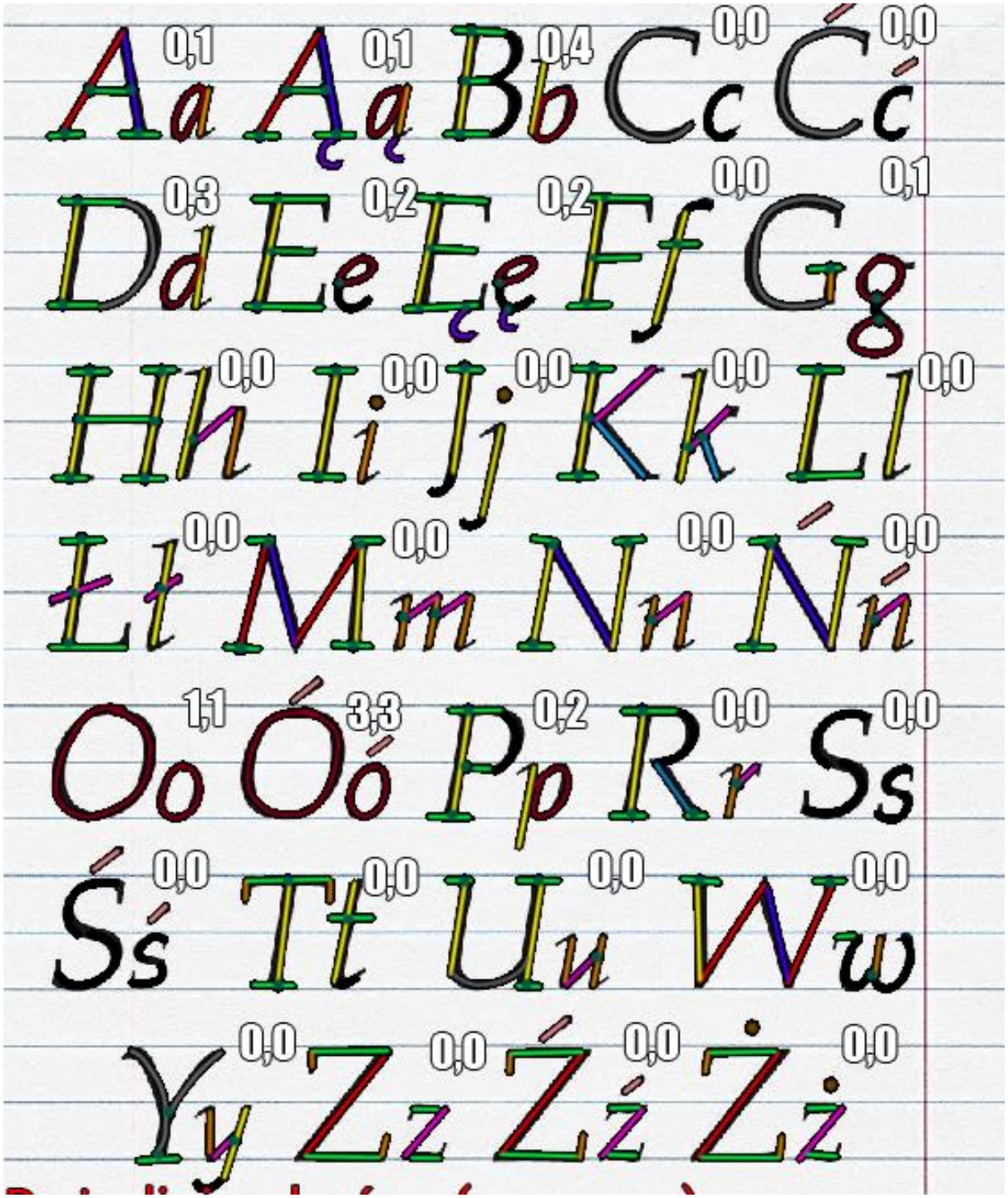
Lista 1- lista cech

Cechy spisane w Lista 1 naniesiono graficznie na każdą z liter alfabetu, a następnie podliczono ilość ich wystąpień dla każdej z nich:



Rysunek 1: Alfabet z naniesionymi cechami

Operację z Rysunek 1 powtórzono dla innego zestawu liter, spisanego w innej czcionce:



Rysunek 2- cechy naniesione na litery z drugiego zestawu

Jednak by właściwie nauczyć model rozpoznawać cały alfabet, należałoby wygenerować takie zbiory dla bardzo wielu fontów, a ręczne utworzenie takiego zbioru dla jednej czcionki to kwestia godzin, toteż zdecydowano się skupić jedynie na rozpoznawaniu litery A oraz B. W efekcie wyeliminowano także ogonek, kreskę i kropkę ze zbioru cech, gdyż te litery ich zwyczajnie nie posiadają w żadnej z badanych czcionek.

Utworzono listę, na której wypisano znaki *Aa* oraz *Bb* w 33 czcionkach, możliwie różnych od siebie. Każdej ze 132 wypisanych liter nadano adekwatny zbiór cech (Rysunek 3).

Rysunek 3- Litery A, a, B oraz b w 33 czcionkach z naniesionymi cechami

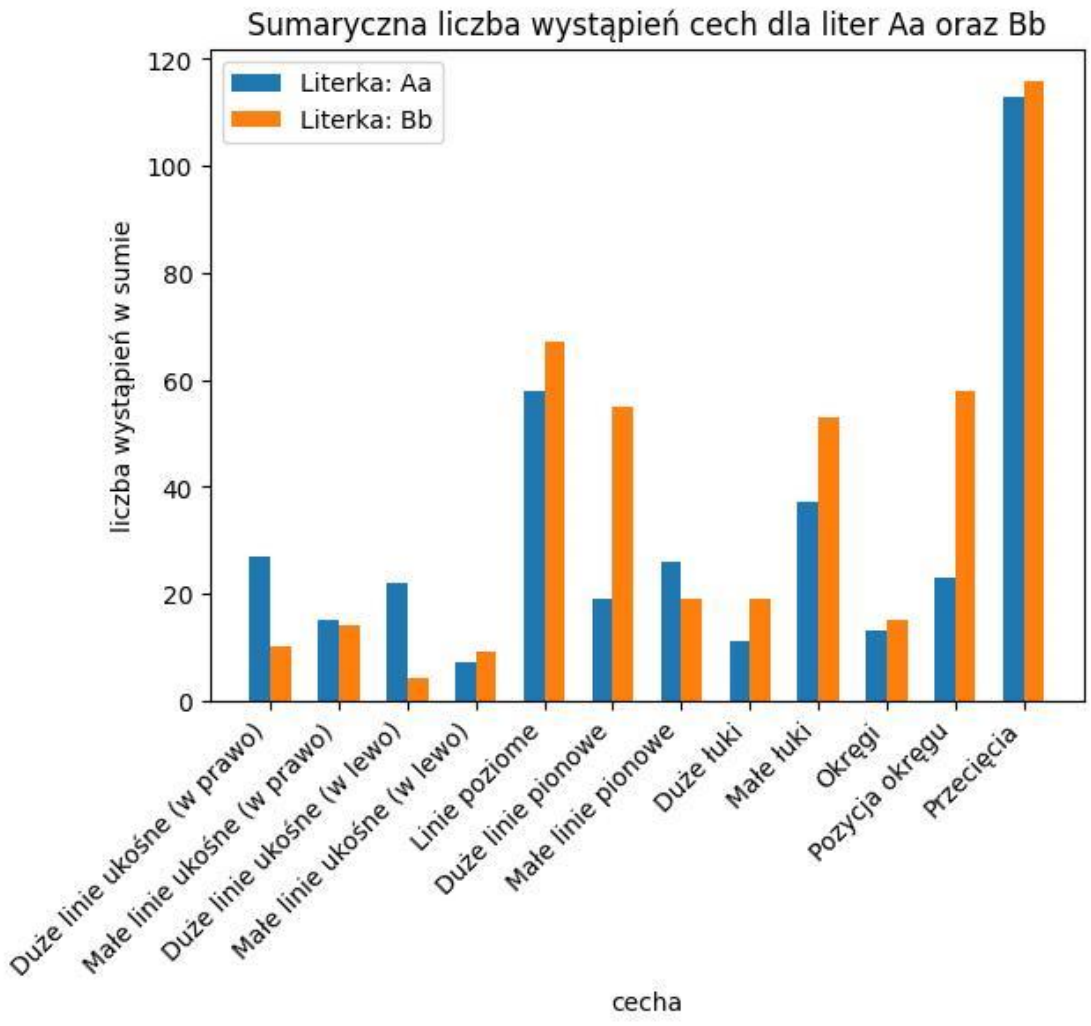
Na bazie Rysunek 3 utworzono dwa pliki *.csv* z danymi: jeden przypisujący małym literom takie same wartości, co wielkim, oraz drugi, odróżniający każdą z badanych liter (A, a, B, b) jako odmienne.

Przykładowy fragment pliku .csv:

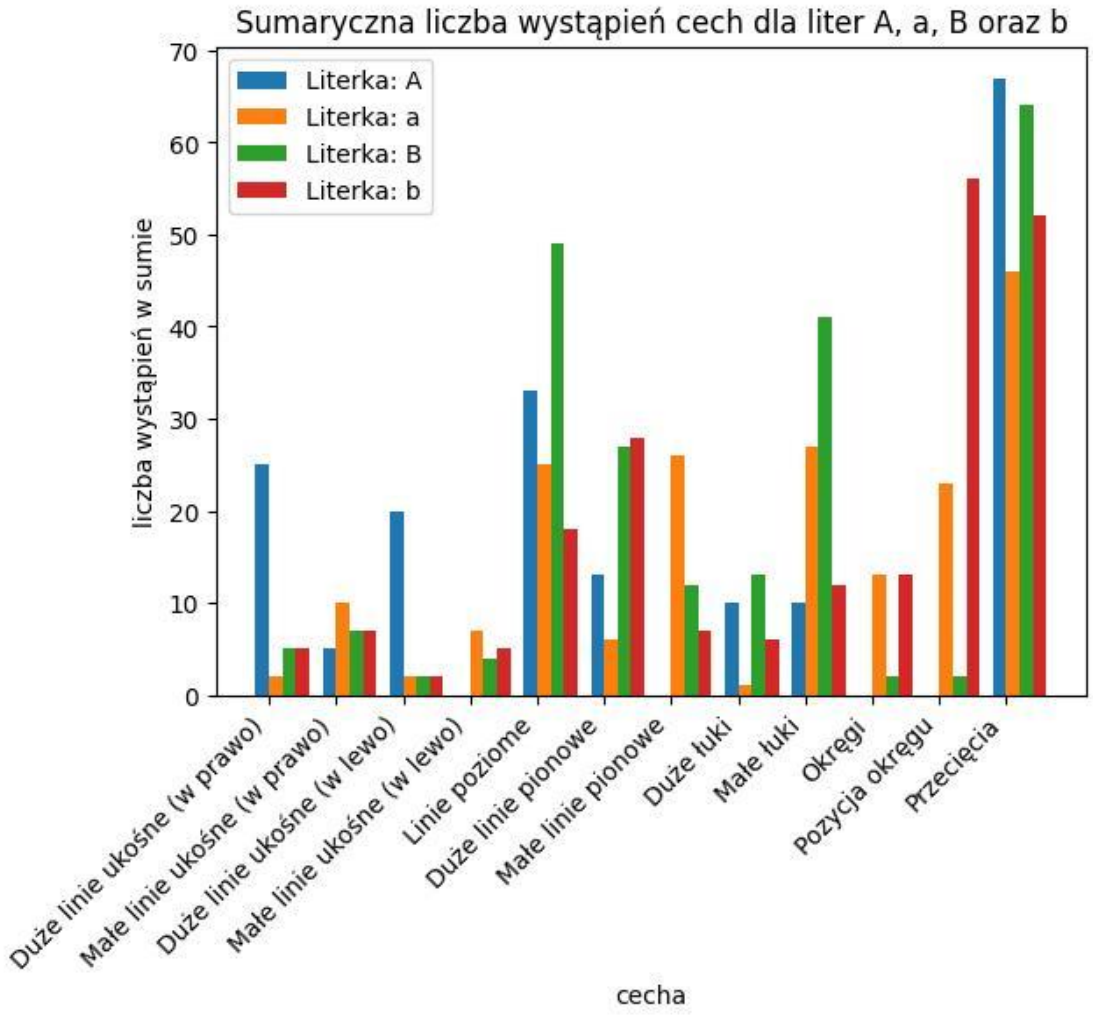
```
EternalUIa;0;0;0;3;3;0;2;0;0;0;0;1;0
EternalUIB;0;1;0;1;3;1;2;0;0;0;0;2;1
EternalUIb;0;1;0;1;2;1;1;0;0;0;0;1;1
AlluraA;1;1;0;0;1;0;0;1;3;0;0;2;0
Alluraa;0;1;0;0;0;0;0;0;2;0;0;0;0
AlluraB;1;0;0;0;0;0;0;2;1;0;0;0;1
Allurab;2;0;0;0;0;0;0;0;2;0;0;1;1
ComicSansA;1;1;1;0;0;0;0;0;0;0;0;2;0
ComicSansa;0;0;0;0;0;0;1;0;0;1;1;1;0
```

Pierwsza kolumna danych spisanych w takim pliku zawiera nazwę czcionki oraz literę, ostatnia wartość litery (0 gdy A lub a, 1 gdy B lub b), a pozostałe- kolejne cechy według Lista 1.

Zależność liczby wystąpień danej cechy od badanej litery pokazano na wykresach (Rysunek 4, Rysunek 5), celem obserwacji różnic w ich natężeniu dla każdej z liter.



Rysunek 4- rozkład cech dla wariantu AB



Rysunek 5- rozkład cech dla wariantu AaBb

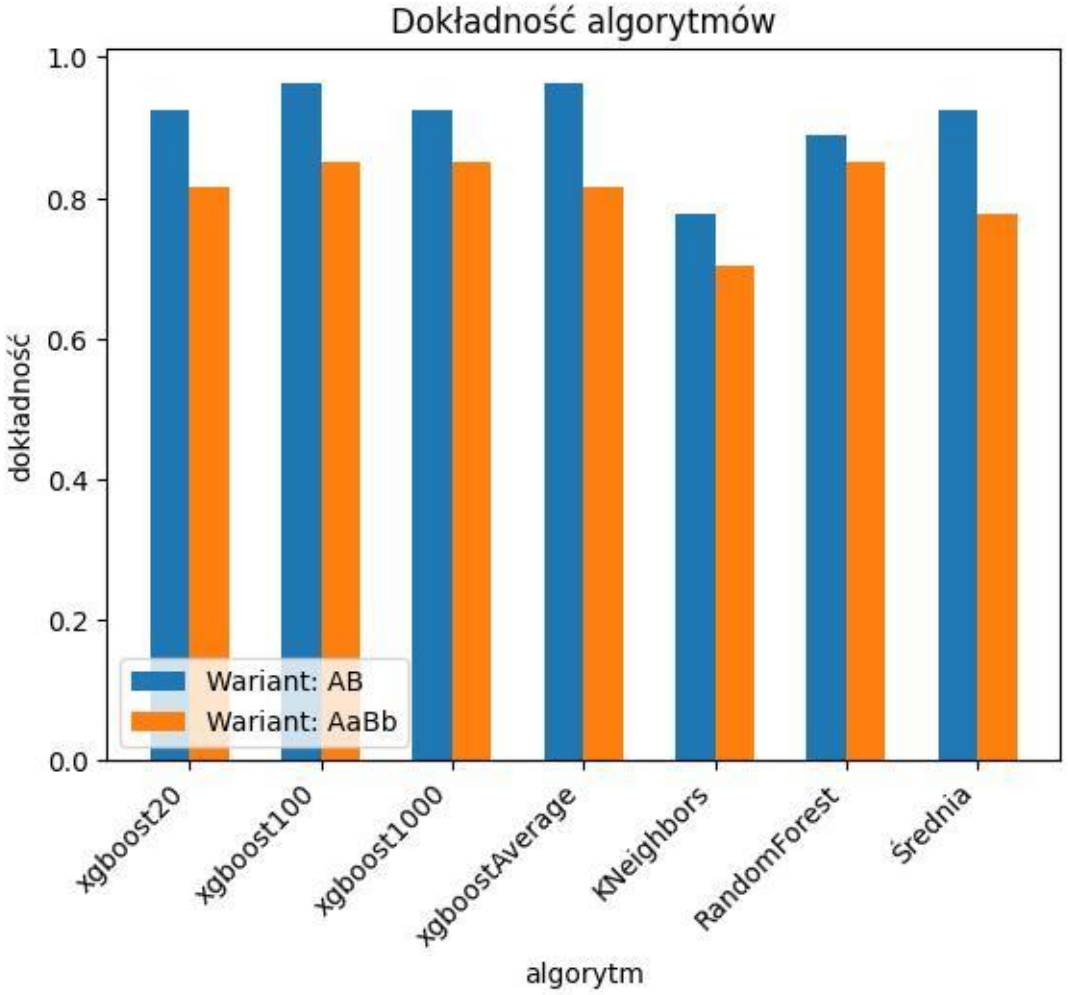
Tak spreparowane dane wczytywane były do programu jako DataFrame przy użyciu biblioteki *pandas.* Następnie część rzędów (70-80%) losowo wybierano z całego zestawu i przekazywano do algorytmu celem wyuczenia modelu. Pozostałe dane używane były do przetestowania utworzonego modelu. Model klasyfikował literę na bazie cech, co następnie porównywane było z faktyczną wartością.

3. Użyte algorytmy

Modele tworzono przy użyciu trzech algorytmów, implementowanych przy użyciu odpowiednich bibliotek:

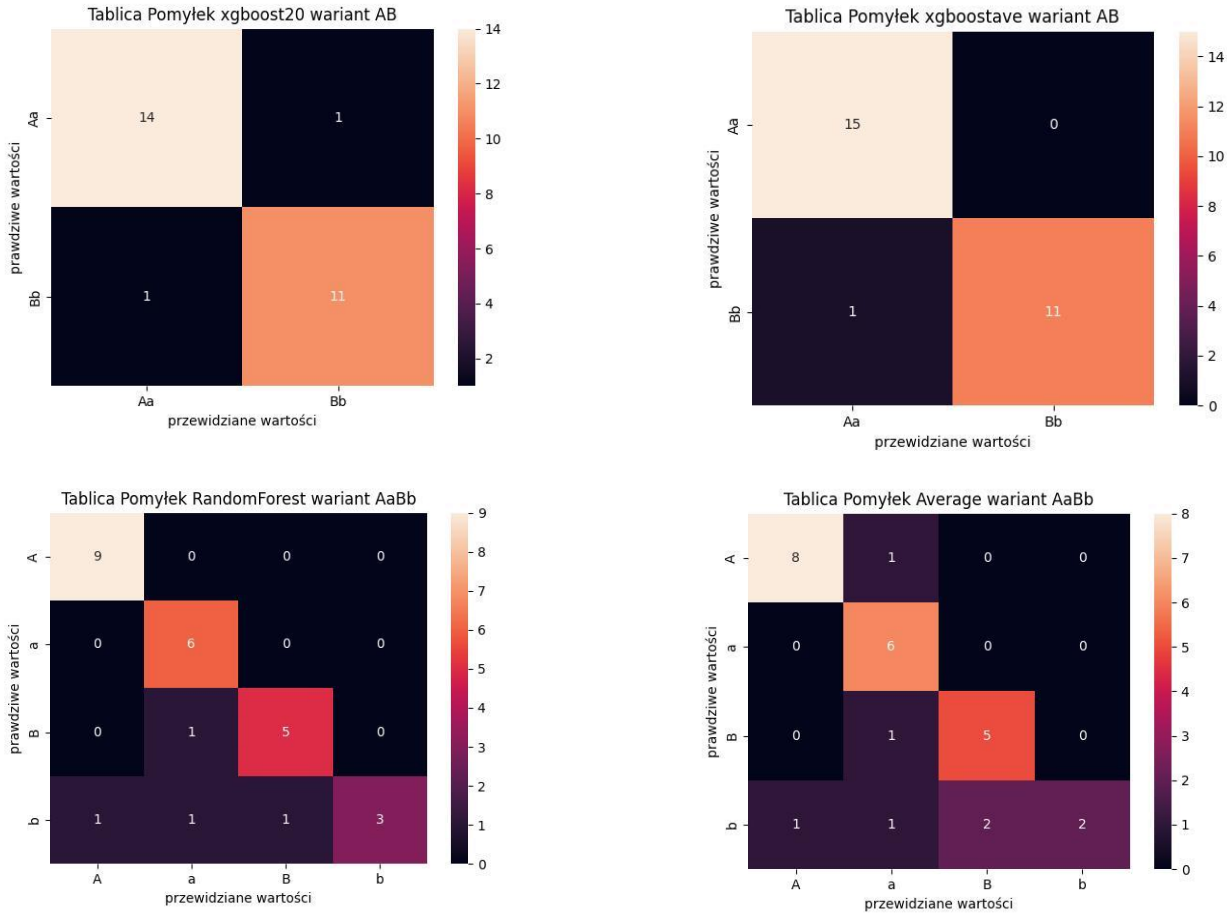
- **XGBoost** (eXtreme Gradient Boosting)- framework oparty na metodzie *gradient boosting*,
- **sklearn.neighbors.KNeighborsClassifier**- oparty na metodzie *K-Nearest Neighbour*,
- **sklearn.ensemble.RandomForestClassifier**- oparty na metodzie *Random Forest*

Każdy z tych algorytmów na wejście przyjmował zestaw danych zawierających wektory treningowe X i Y oraz wektory testowe X i Y. Algorytm *XGBoost* dodatkowo przyjmował wariant metody oraz liczbę powtórzeń, jaką ma wykonać. Algorytmy zwracały wektor przewidzianych wartości oraz ich jakość, mierzona jako liczba poprawnych predykcji podzielona przez wszystkie predykcje. Wartości te umieszczono na wykresie (Rysunek 6), celem porównania.



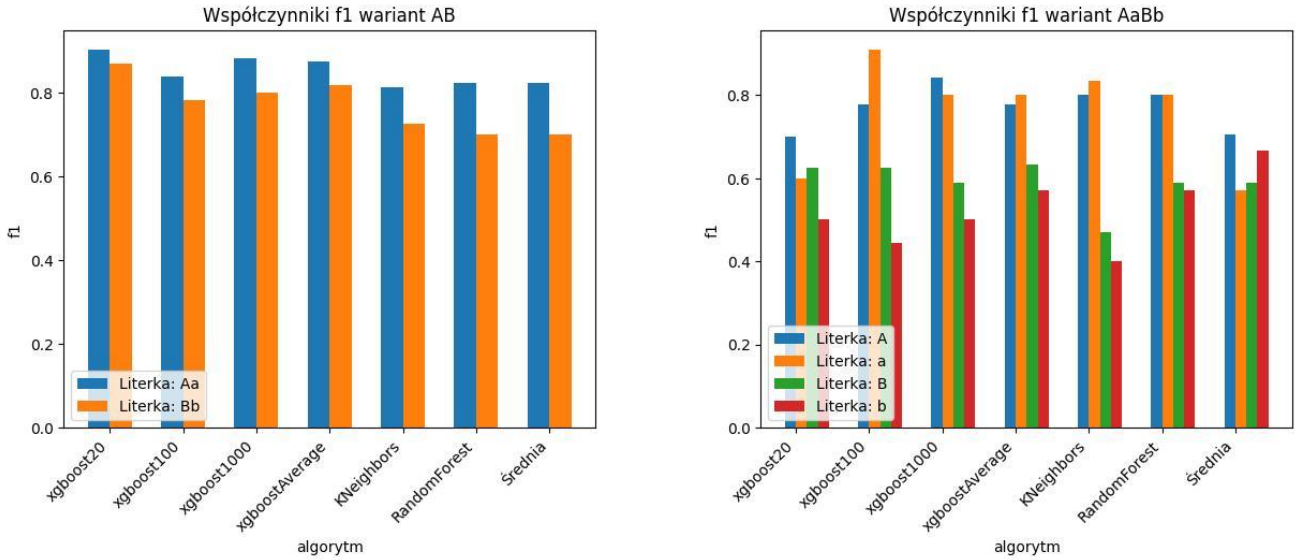
Rysunek 6- Dokładność algorytmów

Dla każdego z wykonanych modeli wygenerowano również tablicę pomyłek (*confusion matrix*) (Rysunek 7), jako kolejną miarę jakości.



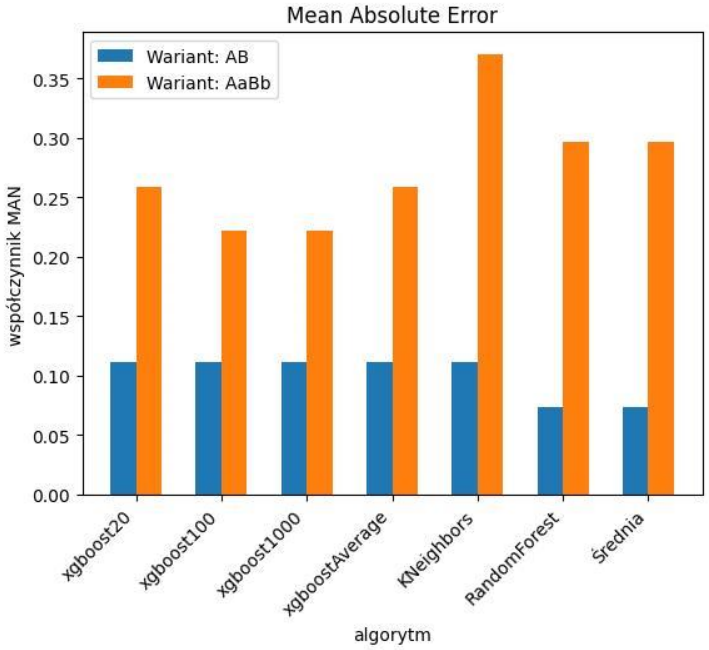
Rysunek 7- przykładowe tablice pomyłek

Na bazie powyższych tablicy pomyłek wyznaczono wartości współczynników f1, czyli miary jakości opartej na dwóch współczynnikach: precyzji (*precision*) i pamięci (*recall*). Pierwszy z nich określa, jak wiele pozytywnych predykcji okazało się właściwe, drugi zaś, jak wiele pozytywnych wartości zostało przewidziane. Im wartość f1 jest wyższa, tym lepiej świadczy to o modelu. Tak uzyskane wartości wypisano na wykresach dla każdego algorytmu i wariantu zadania (Rysunek 8).



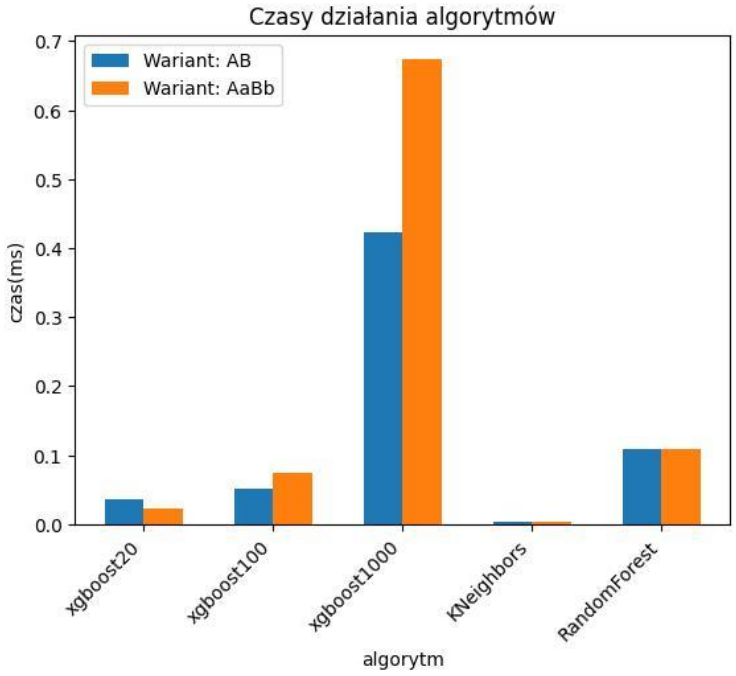
Rysunek 8- Współczynniki f1

Jako ostatnią miarę jakości wyliczono średni błąd bezwzględny (*mean absolute error*), czyli średnia odległość wyznaczonych punktów od wartości rzeczywistych. Jest to suma wartości bezwzględnych różnic wartości każdego z punktów wyznaczonych przez model i wartości rzeczywistych, podzielona przez liczbę wszystkich predykcji. Wartości te powinny być możliwie najmniejsze, by model można było uznać za dobry. Wartości tej miary również umieszczono na wykresie ().



Rysunek 9- współczynniki MAN

Zmierzono również czas trwania każdego z algorytmów, jednakże zastosowanie wysokiej jakości bibliotek oraz stosunkowo niewielkiego zbioru danych skutkowało bardzo krótkim czasem działania każdego z nich. Niemniej, dostrzec można spore różnice, które zaprezentowano na wykresie (Rysunek 10).



Rysunek 10- czasy działania algorytmów

Wyniki działania każdego z algorytmów przechowywano w zmiennych typu *dict*, osobnych dla każdej z badanej wartości: jakości algorytmu, predykcji oraz czasu trwania. Na zakończenie działania programu słowniki te printowane były w czytelny sposób przy użyciu biblioteki pprint (Wynik 1), a następnie przekazywane do funkcji generującej wykresy użyte w niniejszym sprawozdaniu.

```
...
=====WARIANT: AaBb=====
Rozmiar danych uczonych: 105
Rozmiar danych testowych: 27
=====XGBOOST=====
...
[0, 2, 2, 1, 3, 1, 3, 2, 0, 2, 3, 1, 1, 1, 1, 0, 0, 1, 3, 3, 2, 2, 2, 1, 3, 1, 0]
dobrość (średnia): 0.6666666666666666
=====KNeighbors=====
[2, 1, 2, 1, 1, 0, 3, 3, 0, 2, 3, 1, 1, 1, 1, 0, 0, 0, 3, 3, 2, 1, 2, 0, 1, 1, 0]
[3, 1, 0, 1, 3, 3, 2, 2, 0, 2, 3, 1, 1, 1, 1, 0, 3, 1, 3, 3, 3, 2, 2, 2, 3, 1, 2]
dobrość: 0.5185185185185185
=====RANDOM FOREST=====
[0, 3, 2, 1, 3, 1, 3, 1, 0, 2, 3, 1, 1, 1, 1, 0, 0, 1, 3, 3, 2, 2, 2, 0, 3, 1, 0]
[3, 1, 0, 1, 3, 3, 2, 2, 0, 2, 3, 1, 1, 1, 1, 0, 3, 1, 3, 3, 3, 2, 2, 2, 3, 1, 2]
dobrość: 0.6296296296296297
dobrość (średnia ogólna): 0.5925925925925926
=====WYNIKI=====
{'AB': {'KNeighbors': 0.7407407407407407,
        'RandomForest': 0.9259259259259259,
        'xgboost100': 0.9259259259259259,
        'xgboost1000': 0.9259259259259259,
        'xgboost20': 0.8888888888888888,
        'xgboostAverage': 0.9259259259259259,
        'Średnia': 0.9259259259259259},
 'AaBb': {'KNeighbors': 0.5185185185185185,
          'RandomForest': 0.6296296296296297,
          'xgboost100': 0.6666666666666666,
          'xgboost1000': 0.6666666666666666,
          'xgboost20': 0.6666666666666666,
          'xgboostAverage': 0.6666666666666666,
          'Średnia': 0.5925925925925926}}
...
```

Wynik 1- Fragment przykładowego wyjścia programu