Faculty of Mathematics and Information Science

WARSAW UNIVERSITY OF TECHNOLOGY

Automatyczne uczenie maszynowe

Budowanie modeli o największej zdolności predykcyjnej

Autorzy: Damian Lubaszka 313319, Mikołaj Nowak 313400

Kierunek: Informatyka i Systemy Informacyjne

Prowadząca laboratorium: dr mgr Katarzyna Woźnica

Warszawa, 15.01.2024 r.

Spis treści

1 Streszczenie

W niniejszym dokumencie zawarto analizę wyników uzyskanych podczas realizacji zadania. Ponadto w sposób skrupulatny został opisany proces przeprowadzonego doświadczenia oraz zostały opisane osiągnięte wnioski. Instrukcja uruchomienia kodu do wygenerowania rozwiązania oraz jego analizy wraz z wszelkimi uwagami dotyczącymi szczegółów implementacji została zawarta w pliku **README.md**. Ponadto w kodzie zawarto komentarze, w miejscach, gdzie uznano, że są niezbędne.

2 Przygotowanie danych

Zbiór danych treningowych oraz zbiór danych testowych został dostarczony autorom zadania przez prowadzących laboratorium. Zbiór danych treningowych zawierał dwa tysiące rekordów o pięciuset cechach wraz ze zmienną odpowiedzi, natomiast zbiór danych testowych zawierał sześćset rekordów o pięciuset cechach. Ze względu na ograniczoną liczbę rekordów zdecydowano się nie wydzielać ze zbioru treningowego zbioru walidacyjnego i polegać jedynie na wynikach uzyskanych z kross walidacji.

Dostarczone zbiory danych zostały sprawdzone pod kątem braków oraz istnienia duplikatów. Wykonano podstawową analizę eksploracyjną w celu redukcji liczby wymiarów danych. Na rysunku 1 widoczny jest rozkład wartości dla poszczególnych cech. Wykres pozwolił zaobserwować, że większość danych pochodziło z zakresu 370 - 670, co mogło wskazywać, że wiele cech jest ze sobą mocno skorelowane. Ponadto kilka cech posiadało znaczący inny rozkład niż pozostałe, co mogło wiązać się z ich kluczową rolą bądź szumem informacyjnym. Zbadano również skośność danych dla poszczególnych cech, aby wyeliminować niekorzystny wpływ w procesie uczenia modeli, jednakże dane okazały się być niemal symetryczne. Podczas analizy różnych algorytmów na redukcję wymiarowości zdecydowano się również zwrócić uwagę na wartości odstające. Zdecydowano się na eksperymentalne wyznaczenie najlepszej metody preprocessingu.

Jednym z użytych algorytmów do redukcji wymiarowości był algorytm **PCA** (Algorytm głównych składowych) [1]. Zdecydowano się na użycie algorytmu **PCA** dla różnej liczby komponentów (przyjęto wartości: 0.8, 0.6, 0.4, 0.3). Podstawą do przyjęcia takich wartości była analiza rysunku 2.

Kolejnym z użytych algorytmów do redukcji wymiarowości był algorytm napisany przez autorów rozwiązania polegający na wyrzucaniu cech na podstawie **macierzy korelacji** [2]. Zdecydowano odrzucać cechy dla różnych wartości zmiennej threshold (przyjęto wartości: 0.7, 0.075, 0.05).

Następnym z użytych algorytmów do redukcji liczby cech było pozyskanie najważniejszych cech z perspektywy algorytmu **LASSO** [3]. Została użyta regresja **LASSO**, ponieważ wprowadza regularyzację L1, co skutkowało wzięciem pod uwagę rzadkich wag poszczególnych cech.

Ostatnim z użytych algorytmów do redukcji liczby kolumn w dostarczonych zbiorach było pozyskanie najważniejszych cech z perspektywy algorytmu **Random Forest** [4]. Został użyty ze względu na jego zdolność do oceny ważności cech na podstawie kryterium Gini lub Entropii.

Na rysunku 3 widać rozkład wartości cech po redukcji ilości kolumn używając konkretnego algorytmu. Podczas realizacji zadania okazało się, że najlepsze rezulaty otrzymujemy dzięki metodzie **PCA**. Stąd zdecydowano się przetestować więcej wartości liczb komponentów, co będzie poruszone w kolejnych rozdziałach.

3 Model przygotowany ręcznie

Przygotowanie modelu ręcznego zostało podzielone na etapy. Wyniki z poszczególnych etapów zapisane są w odpowiednich plikach csv. Za wartość kross walidacji przyjęto liczbę 7. Funkcją scoring zgodnie z treścią polecenia została miara **balanced_accuracy**.

W pierwszym etapie skupiono się na przetestowaniu wspomnianych w poprzednim rozdziale metod preprocessingu oraz znalezieniu optymalnego modelu. Zdecydowano się na budowę modelu będącego **ensebmlingiem**,

aby uzyskać lepszą wydajność niż pojedyncze modele. Zdecydowano się na **stacking**, czyli łączenie wyników kilku modeli poprzez dodanie meta modelu, uczącego się na podstawie wyników innych modeli. Zdecydowano się na budowę 2 poziomowego stacku, a meta modelem została regresja logistyczna.

Do budowy najwydajniejszego stacku zdecydowano się wybrać następujące algorytmy uczenia maszynowego:

- Random Forest Classifier [4],
- Gradient Boosting Classifier [5],
- SVM Classifier [6],
- K Neighbors Classifier [7],
- Decision Tree Classifier [8],
- MLP Classifier [9].

Użyto pakietu **GridSearchCV** z biblioteki **sklearn** aby znaleźć najlepszą kombinację 2 algorytmów przedstawionych powyżej w zależności od danej metody preprocessingu. Po wykonaniu tego etapu okazało się, że najlepsze rezultaty uzyskano dla metody **PCA**. Zaobserwowano zależność, że czym mniej cech zostawało w danych treningowych tym lepsze wyniki otrzymywano. Zdecydowano się więc na znalezienie minimum lokalnego dla metody **PCA** i jako liczbę komponentów przyjęto wartości 9, 8, 7, 6, 5, 4. Ponadto najlepsze uzyskane stacki zawierały zawsze SVM Classifier lub K Neighbors Classifier, więc ograniczono siatkę przeszukiwanych kombinacji modeli.

Drugi etap realizował to samo co pierwszy etap uwzględniając wiedzę zdobytą podczas wykonywania etapu pierwszego. Okazało się, że metoda preprocessingu **PCA** dla liczby komponentów 5 i 7 osiągała najlepsze rezultaty. Stąd tylko na nich skupiono się podczas realizowania dalszej części zadania. Ponadto wybrano poniższe kombinacje modeli do rozpatrzenia(w nawiasie zawarta jest liczba iteracji dla algorytmu RandomizedSearchCV użytego w kolejnym etapie):

- K NeighborsClassifier z SVC (1000)
- K NeighborsClassifier z RandomForestClassifier (1000)
- K NeighborsClassifier z GradientBoostingClassifier (1000)
- K NeighborsClassifier z DecisionTreeClassifier (10000)

Etap trzeci opierał się na tunowaniu hiperparametrów dla wyżej wymienionych stacków oraz metod preprocessingu. Użyto pakietu **RandomizedSearchCV** z biblioteki **sklearn**, ze względu na możliwość wykorzystania wszystkich wątków CPU do przeprowadzania obliczeń. W dodatku znajduję się tabela 5 zawierające użyte zakresy hiperparametrów. Najlepszy rezultat otrzymał stack składający się z K NeighborsClassifier oraz GradientBoostingClassifier.

Etap czwarty opierał się na ponownym tuningu najlepszego stacku używając jednakże pięcio krotnie więcej punktów w przestrzeni hiperparametrów niż w etapie trzecim.

Tabela 1 zawiera najlepsze uzyskane wyniki. Na rysunku 4 widoczny jest rozkład zmiennych wynikowych. Finalnym modelem został model mający największą zdolność predykcyjną niezależnie od wyników uzyskanych w udostępnionej sprawdzarce. Uzyskane wartości hiperparametrów dla najlepszego modelu zostały zarpezentowane w dodatku w tabeli 6. Więcej szczegółowych wyników można znaleźć w pliku **model-manual.ipynb**.

Tabela 1: Wyniki modeli zbudowanych ręcznie w zależności od zastosowanych metod preprocessingu.

Preprocessing	Model	Wynik
PCA 5	K NeighborsClassifier z SVC	0.88848
PCA 5	K NeighborsClassifier z RandomForestClassifier	0.88849
PCA 5	K NeighborsClassifier z GradientBoostingClassifier	0.89599
PCA 5	K NeighborsClassifier z DecisionTreeClassifier	0.88948

4 Model przygotowany z użyciem frameworków autoMLowych

W tej sekcji przeprowadzono budowanie modeli predykcyjnych za pomocą trzech różnych frameworków autoMLowych. Porównano modele z nich uzyskane i wybrano te o najlepszych własnościach predykcyjnych.

Wszystkie użyte frameworki dostarczają szeroki zestaw modeli, co pozwala na dostosowanie się do różnych problemów klasyfikacyjnych. Automatyzują proces budowy tych modeli eliminując potrzebę ręcznego wyszukiwania najlepszych narzędzi do posiadanych danych. Wykorzystują techniki ensemblingu pozwalające na łączenie wielu modeli w celu poprawy jakości wytrenowanego modelu. Wszystkie frameworki również wykrywają typ badanej klasyfikacji i w tym przypadku nie ma konieczności wskazywania, że rozważamy problem klasyfikacji binarnej.

Wykorzystano mechanizm pipeline z biblioteki *sklearn* w celu połączenia przygotowanego preprocessingu bezpośrednio przed wytrenowaniem modelu za pomocą użytych frameworków.

4.1 AutoGluon

Zbudowanie modelu za pomocą frameworku AutoGluon wymagało dostosowania do zastosowanych metod preprocessingu. W celu posłużenia się `TabularPredictor` w pipelinie konieczne było zaimplementowanie pomocniczej klasy implementującej metody `fit` oraz `transform`.

W metodzie `fit` następuje odpowiednie przygotowanie ramki danych, stworzenie `TabularPredictor` z metryką `balanced_accuracy` oraz wytrenowanie modelu. Zależało nam na jak najlepszej jakości modelu, więc zastosowano `best_quality` jako zestaw konfiguracyjny. Dzięki temu wyeliminowano potrzebę ręcznego dostosowania hiperparametrów. Według dokumentacji ręczny tuning hiperparametrów nie jest zalecany, ponieważ AutoGluon osiąga najlepsze wyniki przy wspomnianej konfiguracji `best_quality` [10].

Natomiast metoda `transform` została wykorzystana do wydobycia odpowiednich danych z prognozy prawdopodobieństw przynależności do klas oraz informacji o wytrenowanych modelach i ich jakości.

Każda przygotowana metoda preprocessingu została umieszczona w pipeline razem z klasyfikatorem AutoGluona. Wytrenowano modele i wydobyto z nich informacje o jakości i ich rankingu. W tabeli 2 przedstawiono najlepsze modele jakie zbudował AutoGluon w zależności od zastosowanej metody preprocessingu. Łącznie przeprowadzono uczenie dla 13 różnych metod preprocessingu, ale zaprezentowano 5 najlepszych z nich.

Tabela 2: Modele zbudowane przez AutoGluon w zależności od zastosowanych metod preprocessingu.

Preprocessing	Model	Wynik
PCA 5	WeightedEnsemble_L3	0.924
PCA 0.3	WeightedEnsemble_L3	0.913
PCA 7	WeightedEnsemble_L2	0.911
PCA 6	WeightedEnsemble_L2	0.910
Random Forest	WeightedEnsemble_L3	0.892

4.2 AutoSklearn 2.0

Klasyfikator frameworka autosklearn implementuje metody `fit` oraz `transform`, więc nie jest konieczna implementacja własnego klasyfikatora jak w przypadku AutoGluona.

Każda przygotowana metoda preprocessingu została umieszczona w pipeline razem z klasyfikatorem AutoSklearn 2.0. Wytrenowano modele i wydobyto z nich informacje o jakości i ich rankingu. W tabeli 3 przedstawiono najlepsze modele jakie zbudował AutoSklearn w zależności od zastosowanej metody preprocessingu. Łącznie przeprowadzono uczenie dla 8 różnych metod preprocessingu, ale zaprezentowano 5 najlepszych z nich.

Tabela 3: Modele zbudowane przez AutoSklearn 2.0 w zależności od zastosowanych metod preprocessingu.

Preprocessing	Model	Wynik
PCA 5	extra_trees	0.888
PCA 6	extra_trees	0.881
PCA 7	extra_trees	0.873
PCA 0.3	extra_trees	0.871
PCA 0.4	extra_trees	0.841

4.3 mljar

Klasyfikator frameworka autosklearn implementuje metody `fit` oraz `transform`, więc nie jest konieczna implementacja własnego klasyfikatora jak w przypadku AutoGluona.

Przygotowano odpowiednie parametry przyjmowane przez `AutoML` - metryka `accuracy`, maksymalny czas wykonania (1h - tyle samo ile w powyższych frameworkach) oraz liczba równolegle wykonywanych zadań w procesie optymalizacji hiperparametrów.

Każda przygotowana metoda preprocessingu została zawarta w pipeline razem z klasyfikatorem z mljar. Wytrenowano modele i wydobyto z nich informacje o jakości i ich rankingu. Na danych testowych obliczono prawdopodobieństwo przynależności do klasy 1. W tabeli 4 przedstawiono najlepsze modele jakie zbudował `AutoML` w zależności od zastosowanej metody preprocessingu. Łącznie przeprowadzono uczenie dla 6 różnych metod preprocessingu, ale zaprezentowano 5 najlepszych z nich.

Tabela 4: Modele zbudowane przez mljar w zależności od zastosowanych metod preprocessingu.

Preprocessing	Model	Wynik
PCA 6	Ensemble	0.887
PCA 5	Ensemble	0.885
PCA 0.3	Ensemble	0.879
PCA 7	Ensemble	0.878
Random Forest	Ensemble	0.875

4.4 Wnioski

Porównując modele uzyskane przez użyte frameworki, najlepszy rezultat uzyskał AutoGluon, który przy preprocessingu algorytmem PCA o 5 komponentach uzyskał wynik 0.924. Żaden inny framework nie przekroczył granicy 0.89, co AutoGluonowi udało się aż pięć razy. Zatem w przypadku badanego zbioru AutoGluon okazał się znacznie lepszym wyborem niż AutoSklearn 2.0 oraz mljar.

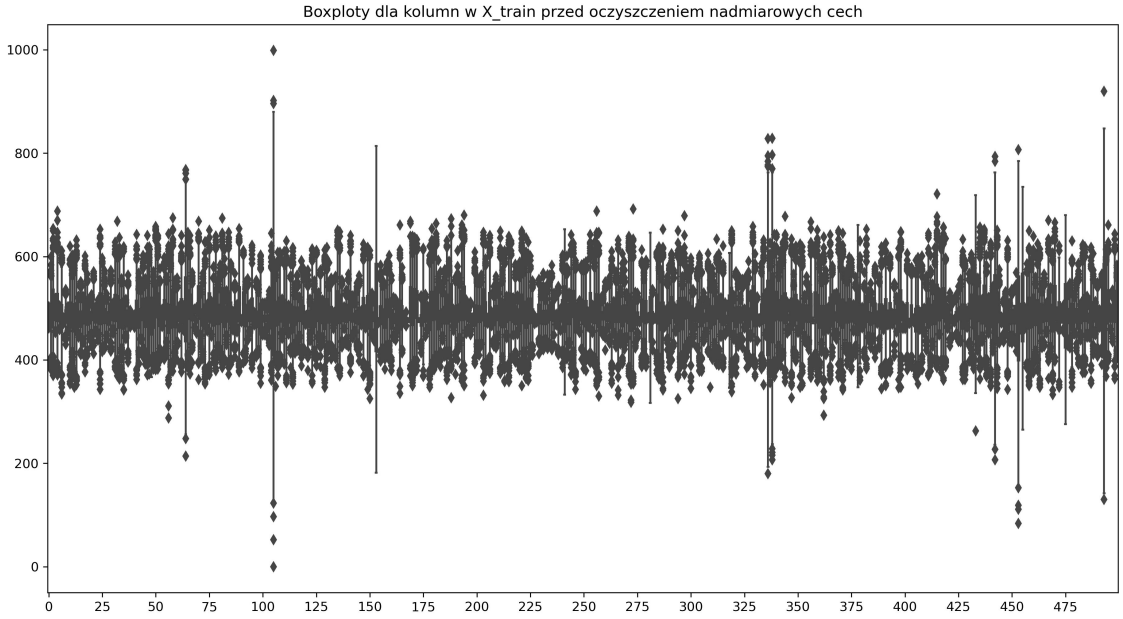
5 Dodatek

Tabela 5: Tabela zawierająca użyte zakresy hiperparametrów.

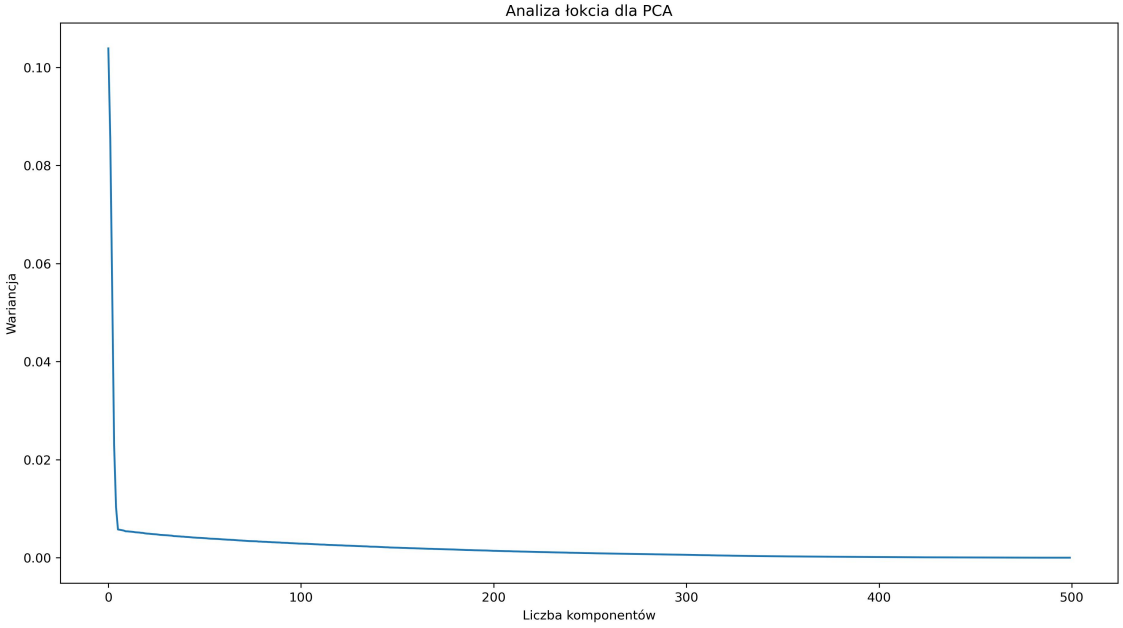
Algorytm uczenia maszynowego	**Hyperparametr**	**dolny zakres**	**górny zakres**
Random Forest Classifier	n_estimators	10	100
Random Forest Classifier	max_depth	1	10
Random Forest Classifier	min_samples_split	1	5
Random Forest Classifier	min_samples_leaf	1	10
Random Forest Classifier	max_features	['sqrt', 'log2']	
Gradient Boosting Classifier	n_estimators	5	100
Gradient Boosting Classifier	max_depth	1	8
Gradient Boosting Classifier	min_samples_split	1	10
Gradient Boosting Classifier	min_samples_leaf	1	5
Gradient Boosting Classifier	learning_rate	0.05	0.5
Gradient Boosting Classifier	subsample	0.75	1.0
SVM Classifier	C	0.01	10.0
SVM Classifier	kernel	['rbf', 'sigmoid']	
SVM Classifier	gamma	0.001	6.5
K Neighbour Classifier	n_neighbors	1	20
K Neighbour Classifier	weights	['uniform', 'distance']	
K Neighbour Classifier	p	[1, 2]	
Decision Tree Classifier	max_depth	1	10
Decision Tree Classifier	min_samples_split	1	5
Decision Tree Classifier	min_samples_leaf	1	10
Decision Tree Classifier	max_features	['sqrt', 'log2']	

Tabela 6: Tabela zawierająca uzyskane hiperparametry dla najlepszego modelu znalezionego ręcznie.

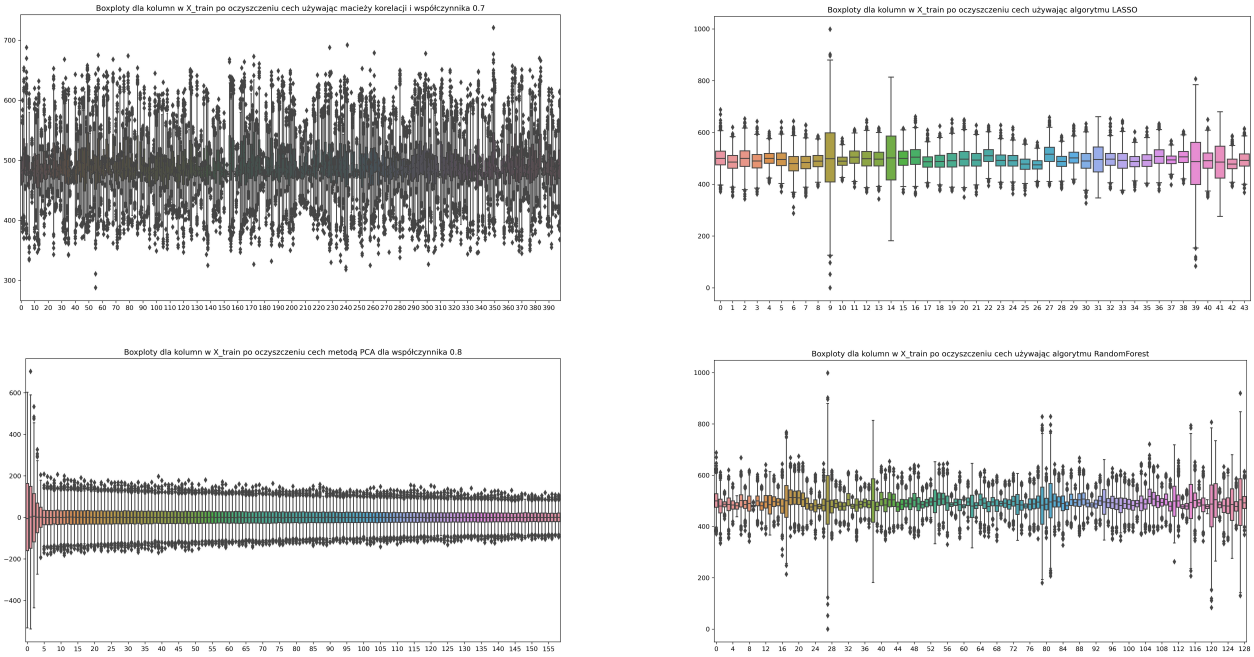
Algorytm uczenia maszynowego	**Hyperparametr**	**wartość**
Gradient Boosting Classifier	n_estimators	91
Gradient Boosting Classifier	max_depth	8
Gradient Boosting Classifier	min_samples_split	8
Gradient Boosting Classifier	min_samples_leaf	3
Gradient Boosting Classifier	learning_rate	0.21875
Gradient Boosting Classifier	subsample	0.83333
K Neighbour Classifier	n_neighbors	3
K Neighbour Classifier	weights	'distance'
K Neighbour Classifier	p	2



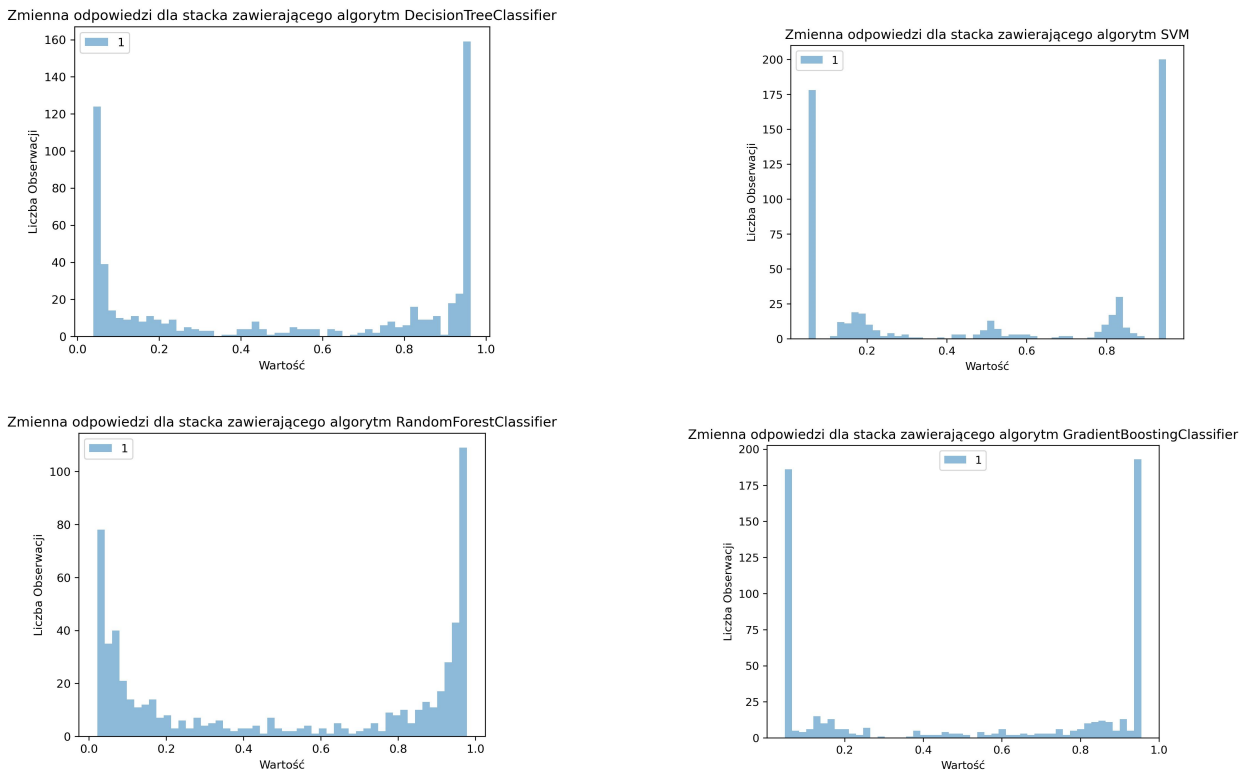
Rysunek 1: Wykres rozkładu wartości dla danych cech dla oryginalnych danych trenigowych.



Rysunek 2: Wykres przedstawiający zmienność wariancji w zależności od liczby cech.



Rysunek 3: Wykresy rozkładu wartości dla danej cechy po użyciu konkretnej metody preprocessingu.



Rysunek 4: Rozkład zmiennej odpowiedzi w zależności od algorytmu uczenia maszynowego użytego w stacku obok KNeighborsClassifier oraz LogisticRegression.

Literatura

[1] Dokumentacja PCA

[2] Macierz korelacji

[3] Regresor LASSO

[4] Dokumentacja sklearn Random Forest Classifier

[5] Dokumentacja sklearn Gradient Boosting Classifier

[6] Dokumentacja sklearn SVM Classifier

[7] Dokumentacja sklearn K Neighbors Classifier

[8] Dokumentacja sklearn Decision Tree Classifier

[9] Dokumentacja sklearn MLP Classifier

[10] Dokumentacja AutoGluon

[11] Dokumentacja AutoSklearn

[12] Dokumentacja MLJar