POLITECHNIKA KRAKOWSKA im. T. Kościuszki
Wydział Mechaniczny
Katedra Informatyki Stosowanej M-07

Kierunek studiów: Informatyka Stosowana
Specjalność: Brak

STUDIA STACJONARNE

PRACA ZALICZENIOWA

Jakub Niewelt, Arkadiusz Spadek, Wiktor Rudzki, Adrian Janowski

Narzędzie do wykrywania anomalii w odczytach czujników temperatur komputera

Kraków, rok akad. 2023/2024

SPIS TREŚCI

1. Cel i zakres pracy 4

2. Wstęp 4

2.1. Definicja uczenia maszynowego 4

2.2. Przykładowe zastosowania uczenia maszynowego 4

2.2.1. Medycyna 4

2.2.2. Finanse 5

2.2.3. Handel 5

2.3. Rodzaje uczenia maszynowego 5

2.3.1. Uczenie nadzorowane 5

2.3.2. Uczenie nienadzorowane 5

2.3.3. Uczenie półnadzorowane 6

2.3.4. Uczenie przez wzmacnianie 6

2.4. Szeregi czasowe 6

2.5. Definicja anomalii 7

2.6. Algorytmy wykrywania anomalii 7

2.6.1. K-średnich 7

2.6.2. Las izolacyjny 8

2.6.3. HBOS 8

2.6.4. Local Outlier Factor 9

2.7. Opis danych 9

3. Implementacja 10

3.1. Środowisko 10

3.1.1. Python 10

3.1.2. Anaconda 10

3.1.3. Jupyter Lab 10

3.2. Obróbka danych 11

3.3. Trenowanie modeli 12

3.3.1. Las izolacyjny 13

3.3.2. HBOS 14

3.3.3. K-Means 16

3.3.4. Local Outlier Factor 18

3.4. Porównanie wyników 20

4. Wnioski 20

5. Podsumowanie..... 21

Literatura 22

Załącznik 1..... 23

1. Cel i zakres pracy

Niniejsza praca zaliczeniowa została podzielona na dwie części. W pierwszej omówiono najważniejsze definicje oraz zastosowania uczenia maszynowego. Przedstawione zostały również najważniejsze modele służące do wykrywania nieprawidłowości w badanych danych.

W drugiej części pracy zostały poruszone kwestie związane z implementacją modeli uczenia maszynowego, w tym opis środowiska i narzędzi, obróbka danych oraz wytrenowanie modeli. Celem wytrenowania wszystkich modeli było porównanie oraz wybranie najlepszego modelu do identyfikacji anomalii w pomiarach temperatury procesora.

Dane, na których zostały przeprowadzone badania zostały wygenerowane przy użyciu oprogramowania MSI Afterburner [1]. Środowisko programistyczne, w którym odbywała się implementacja modeli, to Jupyter Lab [2].

2. Wstęp

2.1. Definicja uczenia maszynowego

Uczenie maszynowe (z ang. Machine learning) można zdefiniować jako programy, które działają coraz lepiej w miarę wykonywania pewnego zadania [2]. Jest to dziedzina nauki wchodząca w skład nauk zajmujących się sztuczną inteligencją. Jej celem jest wykonywanie akcji lub przewidywanie rezultatów na podstawie danych. Proces ten ma miejsce poprzez odkrywanie pewnych wzorców oraz zależności występujących pomiędzy wprowadzonymi danymi. Ogółem rzecz ujmując, jest to system informatyczny, który został stworzony z myślą o tym, aby był w on w stanie samemu podejmować jakieś konkretne decyzje na bazie wiedzy jaką zgromadził. W praktyce można ująć to tak, że zamiast programować system krok po kroku, jak ma rozwiązać konkretny problem, zostają mu przekazane odpowiednie dane, które on następnie przyswaja, tworzy wzorce oraz podejmuje samodzielnie decyzje na podstawie zaimplementowanych algorytmów.

2.2. Przykładowe zastosowania uczenia maszynowego

Uczenie maszynowe znajduje swoje zastosowania w wielu dziedzinach np.

2.2.1. Medycyna

Personalizacja opieki medycznej dla każdego pacjenta, zautomatyzowane operacje chirurgiczne wykonywane przez roboty, wykrywanie nowotworów oraz innych chorób na

zdjęciach z różnych maszyn oraz na podstawie różnych danych pozyskiwanych z krwi, moczu i innych badań [3].

2.2.2. Finanse

Prognozowanie cen akcji, nieruchomości, analiza ryzyka kredytowego, wykrycie podejrzanych transakcji.[4].

2.2.3. Handel

Prognoza popytu na dane artykuły, obsługa klientów (spersonalizowane reklamy, obsługa chatbotów oraz robotów znajdujących się w sklepie), wykrywanie oszustw, analiza ilości produktów [5].

2.3. Rodzaje uczenia maszynowego

Uczenie maszynowe można podzielić według metody trenowania modeli. Jest to najczęściej spotykany podział w literaturze naukowej i ów podział wskazuje 4 główne rodzaje algorytmów uczenia [2].

2.3.1. Uczenie nadzorowane

Uczenie nadzorowane polega na używaniu oznaczonych danych, czyli takich, które mają zarówno wartości jak i etykiety, do uczenia modelu. Uczony w ten sposób model jest w stanie przewidywać konkretne zjawiska tj. wartości liczbowe, np. wskaźnik przestępczości na podstawie innych czynników – takie zagadnienie nazywane jest regresją. Model wykorzystujący uczenie nadzorowane jest także w stanie przydzielać zbiory danych do konkretnych klas, np. uznawanie pacjentów za zdrowych lub chorych na podstawie wyników badań krwi – nazywane jest to powszechnie klasyfikacją [2].

2.3.2. Uczenie nienadzorowane

Uczenie nienadzorowane (z ang. Unsupervised learning) korzysta z danych, które nie są oznaczone. Tworzony model, otrzymuje informacje, które nie posiadają etykiet (odpowiedzi). Jego zadaniem jest tworzenie nowych, nieznanych na etapie tworzenia modelu wzorców. Algorytm nie posiada jasno określonego na wejściu celu, do którego dąży, a zatem rozwiązanie jakie nam zaprezentuje niekoniecznie musi być trafne. Pojęcie uczenia nienadzorowanego jest często stawiane w kontraście do uczenia nadzorowanego, gdzie model ma jasno zdefiniowany cel. Owe uczenie znajduje swoje zastosowanie w klasteryzacji – grupowanie podobnych danych należących do większego zbioru w osobne podzbiory, redukcji wymiarowości - zmniejszanie

liczby zmiennych danych w celu uproszczenia analizy czy też choćby w detekcji anomalii – identyfikacji nieprawidłowych, nienaturalnych wzorców.

2.3.3. Uczenie półnadzorowane

Uczenie półnadzorowane (ang. Semi-supervised learning) można zakwalifikować jako metodę pośrednią między uczeniem nadzorowanym, a nienadzorowanym. Algorytm otrzymuje etykiety oraz pewne informacje o podzbiorze danych, ale tylko dla części danych. Na podstawie tych informacji model ma za zadanie znaleźć zależności między danymi wejściowymi, wykorzystując zarówno dane etykietowane, jak i nieetykietowane, aby poprawić swoje przewidywania i skuteczność działania [6].

2.3.4. Uczenie przez wzmacnianie

Głównym zadaniem uczenia przez wzmacnianie (ang. Reinforcment-learning) jest interakcja ze środowiskiem na podstawie zebranych informacji. W przeciwieństwie do wymienionych wcześniej rodzajów, w uczeniu przez wzmacnianie nie przygotowuje się zestawu danych uczących, tylko środowisko, z którego model będzie zbierał dane automatycznie. Jego celem jest zmaksymalizowanie zwracanej przez środowisko nagrody, które może być zależne od celu nauki. W przypadku uczenia programu grającego w gry będzie to gra, wraz z jej zasadami lub prawdziwy świat, w przypadku programu uczącego się sterować łazikiem [7].

2.4. Szeregi czasowe

Szereg czasowy jest definiowany jako zbiór ilościowych obserwacji danego zjawiska, które są ułożone w postaci okresów w kolejności chronologicznej. Innymi słowami jest to zbiór punktów danych obserwowanych sekwencyjnie przez dany okres. W szeregach czasowych można wyróżnić pewne składowe, które są rezultatami wpływu różnych czynników na ów zjawisko [8].

Składowymi szeregów czasowych są:

1. Trendy, które mówią o długoterminowych wzrostach lub spadkach wartości.
2. Wahania sezonowe (sezonowości), które są wariacjami występującymi w konkretnych, regularnych interwałach nie większych od roku. Mogą być to różne okresy, tj. dzienne, tygodniowe, miesięczne lub roczne.
3. Wahania cykliczne, które są zmiennościami wartości na przestrzeni czasu, lecz nie występują one w ustalonych interwałach o konkretnych częstotliwościach a w pewnych cyklach, które są przeważnie powiązane z gospodarką danego obszaru (cykle koniunkturalne).

4. Stacjonarność, która cechuje takie szeregi czasowe, które nie wykazują żadnych trendów ani wahań sezonowych.

Proces identyfikacji, określania ilości oraz rozkładania na czynniki pierwsze wspomnianych składowych i nie tylko nazywany jest analizą szeregów czasowych [8].

2.5. Definicja anomalii

Termin anomalii jest używany w różnych dziedzinach nauki w celu opisania stanu lub zjawiska, który znacząco odbiega od normy bądź przewidywań. Anomalie bądź wartości odstające są często wynikami błędów pomiarowych lub szumów, ale co ważne ich obecność może mieć istotne znaczenie w zachowaniu obserwowalnego przez nas obiektu czy też zjawiska, co może doprowadzić do konieczności podjęcia przez nas różnych działań. W zależności od kontekstu, anomalia może objawiać się w różny sposób. W naszym przypadku, mamy do czynienia z anomaliami podczas odczytu temperatury z czujników w komputerze. Przykładem może tutaj być nagły, znaczny skok temperatury na procesorze, gdy jego temperatura nie wykazywała wcześniej takich tendencji, a następnie drastyczny spadek do pierwotnej temperatury. Wówczas taki odczyt wydaje się wyraźnie odbiegać od pozostałych, co wskazuje na to, iż jest on anomalią.

W statystyce oraz analizie danych częstym sposobem na stwierdzenie czy dany pomiar można określić mianem anomalii jest zastosowanie reguły 3 sigma. Reguła ta mówi, iż w rozkładzie normalnym wyniki, które są oddalone o więcej niż trzy odchylenia standardowe od średniej, uznaje się za anomalie. W idealnym rozkładzie normalnym, około 99,7% danych znajduje się w zakresie trzech odchyleń standardowych – 3 sigma, natomiast reszta, a zatem około 0,3% to anomalie [8].

2.6. Algorytmy wykrywania anomalii

2.6.1. K-średnich

Algorytm k-średnich (ang. k-means) to jeden z najstarszych i najczęściej używanych algorytmów analizy skupień. Algorytm ten ma na celu znalezienie *K* nienakładających się na siebie klastrów. Zbiory te są definiowane przez ich centroidy (geometryczne środki danych figur; w przypadku klastrów jest to średnia punktów w danym klastrze). Proces klasteryzacji według algorytmu centroidów przebiega następująco. Najpierw wybierane jest *K* początkowych centroidów, gdzie K jest dowolną liczbą wybraną przez użytkownika i wskazuje liczbę pożądanych klastrów. Następnie każdy punkt danych jest przypisywany do najbliższego centroida, a każdy zbiór punktów przypisanych do danego centroida tworzy klaster. Centroid

każdego klastra jest aktualizowany na podstawie przypisanych do niego punktów. Proces ten jest powtarzany do momentu, gdy żaden punkt nie zmienia klastra, do którego jest przypisany [9].

2.6.2. Las izolacyjny

Las izolacyjny (ang. Isolation forest) został wynaleziony przez Fei Tony Liu, Kai Ming Ting i Zhi-Hua Zhou w 2008 roku. Algorytm Isolation Forest opiera się na zestawie drzew decyzyjnych stosowanych na danym zbiorze danych. Główna zasada jego działania polega na „izolowaniu" anomalii poprzez generowanie drzew decyzyjnych przy użyciu losowych atrybutów. Proces losowego podziału prowadzi do tworzenia krótszych ścieżek dla anomalii, ponieważ anomalie jako mniej liczne, tworzą mniejsze partycje. Dzięki temu algorytm skutecznie odróżnia anomalie od danych normalnych, które mają bardziej typowe wartości. Las izolacyjny sprawdza się znakomicie w przypadku dużych i wielowymiarowych zbiorów danych, oferując efektywne i skuteczne rozwiązanie do wykrywania anomalii.

Kluczową obserwacją w algorytmie Isolation Forest jest to, że anomalie mają tendencję do znajdowania się bliżej korzeni drzew decyzyjnych. Ta właściwość stanowi podstawę do zbudowania funkcji punktacji, zwanej scoringiem anomalii. W uproszczeniu, funkcja ta oblicza średnią długość ścieżki od korzenia drzewa do danego punktu. Autorzy algorytmu sugerują, aby wartości tej funkcji mieściły się w przedziale od 0 do 1, jednak różne implementacje mogą stosować odmienne skale [10].

2.6.3. HBOS

Algorytm HBOS (z ang. Histogram-based Outlier Score) to algorytm używany do uczenia nienadzorowanego, który służy do wykrywania anomalii. W tym celu wykorzystuje on histogramy. Głównymi cechami tego algorytmu jest niska złożoność czasowa, gdyż jest ona liniowa oraz założenie, iż cechy są niezależne, co znacznie ułatwia proces obliczeniowy. Działa on w oparciu o wyżej wspomniane histogramy, które są tworzone dla każdej z cech występującej w zbiorze danych. Jeśli cecha zawiera dane kategorialne, wykonywane jest proste zliczanie wartości każdej kategorii i obliczana jest względna częstotliwość (wysokość histogramu).

W przypadku cech numerycznych można zastosować dwie różne metody: statyczne histogramy szerokości kubełków lub dynamiczne histogramy szerokości kubełków.

W kontekście algorytmu HBOS, „kubełki" (z ang. bins) to przedziały, na które dzielone są dane podczas tworzenia histogramu. Kubełki są wykorzystywane do grupowania danych w celu analizy rozkładu wartości. Owe histogramy są następnie normalizowane, tak aby maksymalna wysokość każdego z nich wynosiła 1. Zapewnia to równą wagę każdej cechy dla wyniku odstającego. Algorytm ten bardzo dobrze sprawdza się do pracy nad dużymi zbiorami danych ze

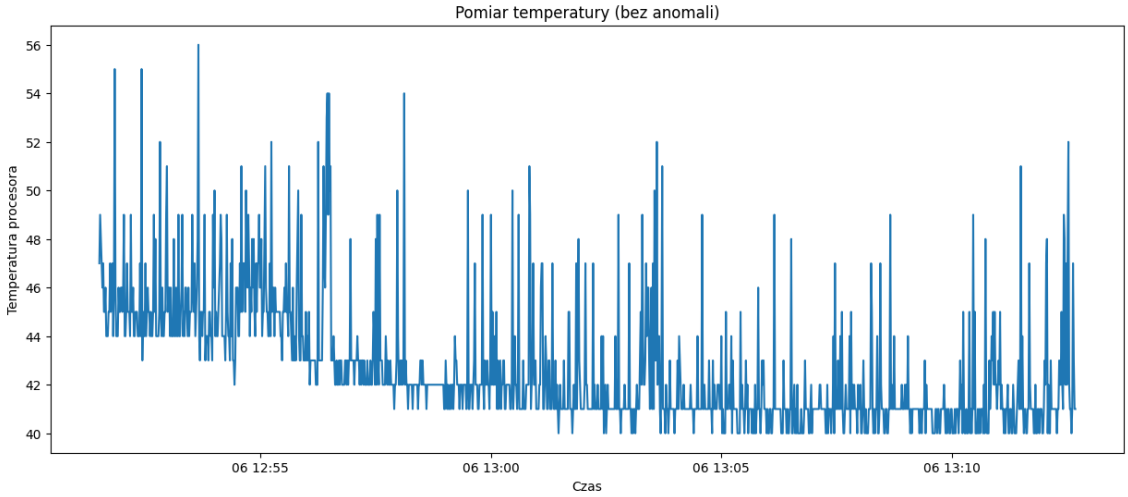
względu na niską złożoność czasową. Dobrze radzi sobie również z wykrywaniem globalnych anomalii, ale ma z kolei problemy z wykryciem tych lokalnych [11].

2.6.4. Local Outlier Factor

Algorytm Local Outlier Factor (LOF) to jeden z najbardziej znanych algorytmów do wykrywania lokalnych wartości odstających. Zlicza on całkowite odchylenie gęstości danej próbki z uwzględnieniem jej sąsiadów. Jest to metoda lokalna ze względu na to, że wynik anomalii jest zależny od tego jak bardzo dany obiekt jest wyizolowany od otaczających go sąsiadów. Innymi słowy lokalność jest podawana na podstawie k-najbliższych sąsiadów, których dystans jest wykorzystywany do oszacowania lokalnego zagęszczenia. Następnie poprzez porównanie lokalnego zagęszczenia próbki do lokalnych zagęszczeń sąsiednich próbek możliwym jest zidentyfikowanie próbek o znacznie mniejszym zagęszczeniu od ich sąsiadów. Takie próbki uważane są za outlier'y, czyli anomalie [12].

2.7. Opis danych

Wykorzystane dane zostały wygenerowane przy użyciu oprogramowania MSI Afterburner [1]. Dwa zbiory danych zostały dostarczone do aplikacji w postaci plików z rozszerzeniem .csv. Każdy ze zbiorów posiada dwie kolumny. Jedna z nich mówi o aktualnej temperaturze procesora, natomiast druga informuje o czasie wykonania pomiaru. Pomiary temperatury były wykonywane co sekundę przez okresu około 20 minut.



Rys. 2.7.1 Wykres przedstawiający temperaturę procesora stacji roboczej bez anomalii

3. Implementacja

3.1. Środowisko

3.1.1. Python

W ramach tworzenia modeli nauczenia maszynowego użyty został wysoko poziomowy język programowania Python wspierający programowanie obiektowe. Język ten jest przeważnie używany w sektorach data science, sztucznej inteligencji, machine learningu oraz badań naukowych. Jest on darmowy i wieloplatformowy, co pozwala na używanie go na dowolnym systemie operacyjnym [13].

Dodatkowo w ramach naszego projektu wykorzystane będą biblioteki, które wspomagają nauczanie maszynowe oraz ułatwiają pewne operacje i funkcje matematyczne. Tymi bibliotekami są:

- pandas – ułatwia ona pracę ze zbiorami danych, dzięki funkcjom przeznaczonym do analizy, oczyszczania i manipulowania danymi [14].
- NumPy – wspomaga pracę z tablicami komputerowymi, poprzez szeroką gamę funkcji z domeny algebry liniowej, transformacji Fouriera oraz macierzy [15].
- Matplotlib – usprawnia tworzenie statycznych, animowanych oraz interaktywnych wizualizacji – wykresów [16].
- scikit-learn – narzędzie do predykcyjnej analizy danych, jest to projekt typu open source oparty na NumPy, SciPy i matplotlib [17]

3.1.2. Anaconda

W trakcie projektu wykorzystywane będzie środowisko Anaconda – darmowe oprogramowanie, które dostarcza zestaw narzędzi dopasowanych do pracy w zakresie data science i machine learningu [13]. Główną zaletą Anacondy jest możliwość tworzenia i zarządzania środowiskami wirtualnymi. Środowiska te umożliwiają odseparowywanie wykorzystywanych przez użytkownika bibliotek, instalacji Pythona czy też zmiennych. Dzięki temu korzystając z jednego systemu możliwym jest pracowanie nad kilkoma projektami, które mają konkretne wymogi co do wersji i rodzajów bibliotek, bez ingerencji w nie w momencie przełączania się z jednego projektu na drugi [2].

3.1.3. Jupyter Lab

Cały kod będzie pisany w zintegrowanym środowisku programistycznym Jupyter Lab, który pozwala na szybkie tworzenie i zarządzanie projektami poprzez pracę w notatnikach. Środowisko

umożliwia uruchamianie specjalnych instancji interpretera Pythona (kernels), które wykonują kod zamieszczony w notatnikach, co pozwala na współpracę wielu użytkowników. Każdy notatnik ma możliwość dodawania 3 różnych typów komórek: kodu, Markdown i surowych danych [2].

3.2. Obróbka danych

W obu zbiorach danych wymaganym była zamiana kolumn typu timestamp na datetime, gdyż większość transformacji i funkcji nie akceptuje typu timestamp. Do pierwszego pomiaru dodaliśmy ręcznie 4 sztuczne anomalie w celu wytrenowania naszego modelu na zbiorze, który z pewnością zawiera nieprawidłowości. Natomiast do drugiego pomiaru nie zostały dodane żadne dodatkowe punkty, aby przetestować model na prawdziwych danych. Wczytywanie danych odbywa się z pliku .csv za pomocą biblioteki pandas.

```
1  data = pd.read_csv('./data/CPUTemperatureLog_with_anomaly.csv')
2  data['Czas'] = pd.to_datetime(data['Czas'])
3
4  data_without_anomalies = pd.read_csv('./data/CPUTemperatureLog.csv')
5  data_without_anomalies['Czas'] = pd.to_datetime(data['Czas'])
```

Rys. 3.2.1. Kod wczytujący dane i wykonujący konwersję typu timestamp na datetime

Po wykonej transformacji sprawdzona została zawartość naszego zbioru.

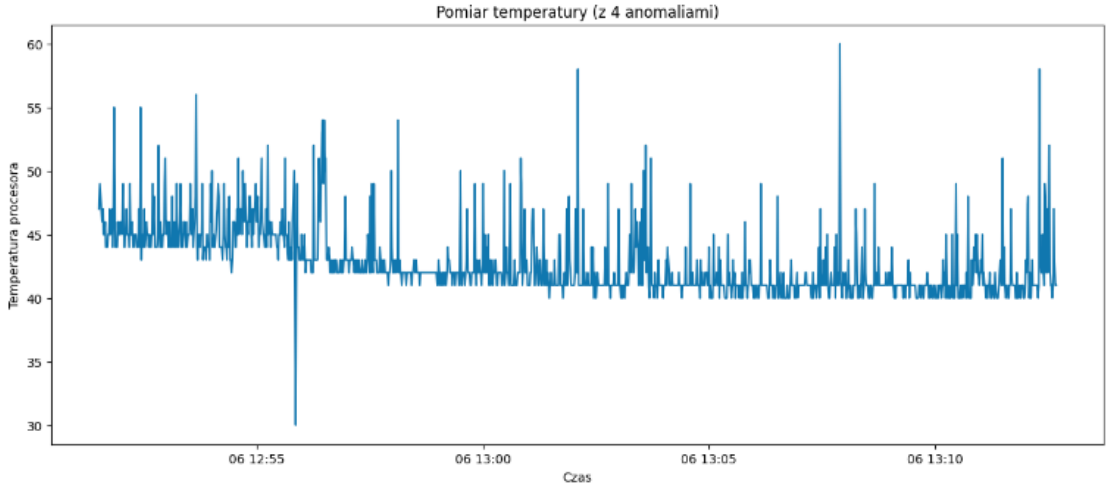
	Czas	Temperatura procesora
0	2024-08-06 12:51:30	47
1	2024-08-06 12:51:31	49
2	2024-08-06 12:51:32	48
3	2024-08-06 12:51:33	47
4	2024-08-06 12:51:34	46

Rys. 3.2.2. Nagłówek zbioru danych z anomaliami z pierwszymi wierszami

	Czas	Temperatura procesora
count	1272	1272.000000
mean	2024-08-06 13:02:05.500000	42.753145
min	2024-08-06 12:51:30	30.000000
25%	2024-08-06 12:56:47.750000128	41.000000
50%	2024-08-06 13:02:05.500000	42.000000
75%	2024-08-06 13:07:23.249999872	44.000000
max	2024-08-06 13:12:41	60.000000
std	NaN	2.713791

Rys. 3.2.2. Cechy charakterystyczne zbioru danych z anomaliami

Każda kolumna posiada taką samą ilość danych co oznacza, że nie występują żadne wartości brakujące, więc nie są wymagane dalsze transformacje.



Rys. 3.2.3. Wykres zbioru danych z 4 ręcznie dodanymi anomaliami

3.3. Trenowanie modeli

Bazując na modelach opisanych w podpunkcie *2.6* przeprowadzone zostało uczenie maszynowe na danych ze sztucznie wygenerowanymi anomaliami. Pozwoliło to na porównanie skuteczności modeli oraz weryfikację czy algorytmy poprawnie wykrywają wartości odstające.

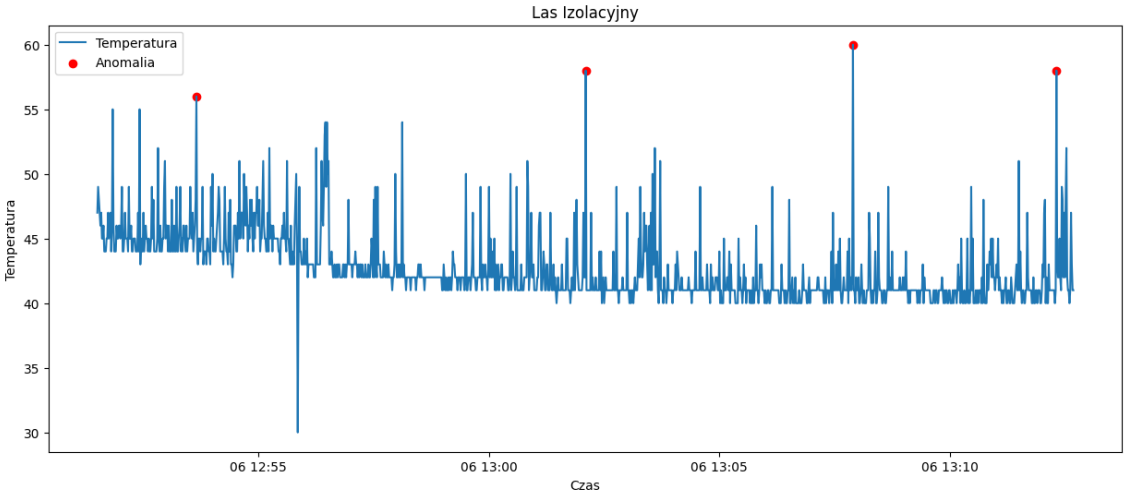
Następnie modele zostały poddane próbie wykrycia anomalii w zbiorach danych, w których nie występowały sztucznie umieszczone wartości sugerujące na wystąpienie niepoprawnego odczytu z czujnika temperatury procesora.

3.3.1. Las izolacyjny

Pierwszym wykorzystanym algorytmem jest Las izolacyjny.

```
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import IsolationForest

data_isolation_forest = data.copy()

outliers_fraction = float(.003)

scaler = StandardScaler()
np_scaled = scaler.fit_transform(data_isolation_forest[["Temperatura procesora"]])
data_temperature_std = pd.DataFrame(np_scaled, columns=data_isolation_forest[["Temperatura procesora"]].columns)

model = IsolationForest(contamination=outliers_fraction)
model.fit(data_temperature_std)

data_isolation_forest['Anomalia'] = model.predict(data_temperature_std)
data_isolation_forest['Anomalia'] = data_isolation_forest['Anomalia'].apply(lambda x: 1 if x == -1 else 0)

show_anomaly_graph(data_isolation_forest, title='Las Izolacyjny')
```

Rys. 3.3.1.1 Kod implementujący algorytm Lasu Izolacyjnego

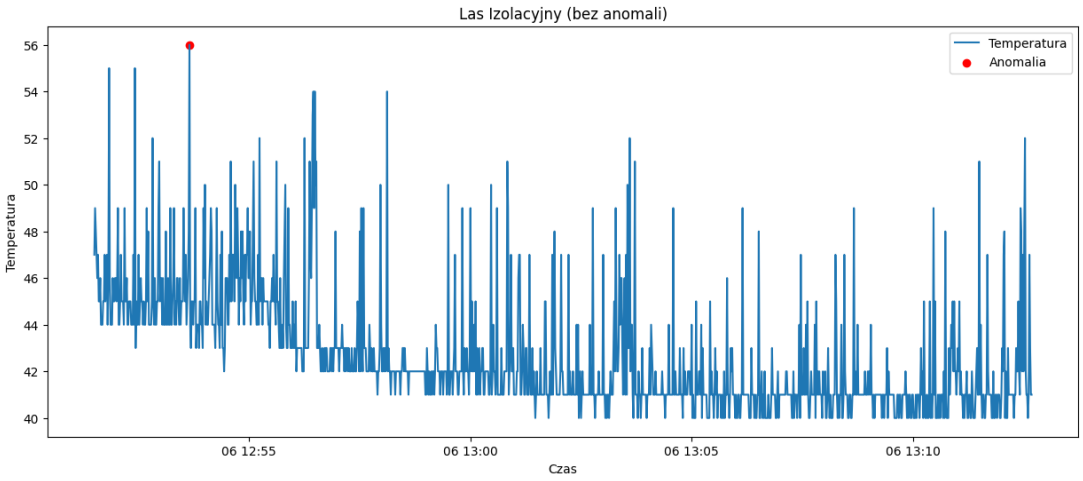


Rys. 3.3.1.2 Wykres wynikowy wytrenowanego modelu

Model wytrenowany na danych ze sztucznie umieszczonymi anomaliami poprawnie zidentyfikował 3 z 4 odstających wartości. Algorytm, niestety nie radzi sobie dobrze w przypadku punktów znajdujących się poniżej normalnych wartości pomiarów. Powoduje to nieprawidłową kwalifikację jednej z anomalii.

```
data_isolation_forest_without_anomalies = data_without_anomalies.copy()

np_scaled = scaler.transform(data_isolation_forest_without_anomalies[["Temperatura procesora"]])
data_temperature_std = pd.DataFrame(np_scaled, columns=data_isolation_forest_without_anomalies[["Temperatura procesora"]].columns)

data_isolation_forest_without_anomalies['Anomalia'] = model.predict(data_temperature_std)
data_isolation_forest_without_anomalies['Anomalia'] = data_isolation_forest_without_anomalies['Anomalia'].apply(lambda x: 1 if x == -1 else 0)

show_anomaly_graph(data_isolation_forest_without_anomalies, title='Las Izolacyjny (bez anomali)')
```

Rys. 3.3.1.3 Kod tworzący wykres dla danych bez anomalii



Rys. 3.3.1.4 Wykres wynikowy wytrenowanego modelu dla danych bez anomalii

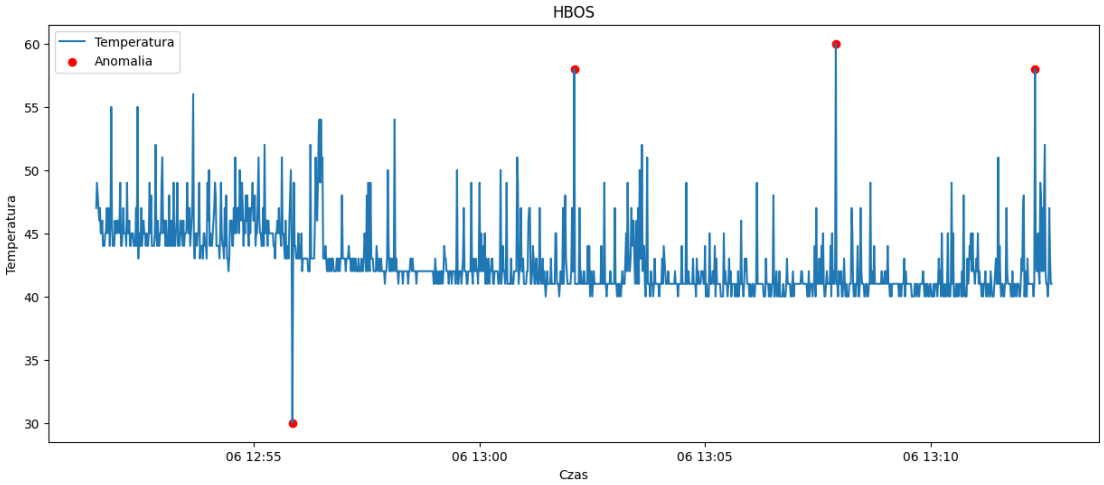
W przypadku predykcji anomalii na zbiorze nie zawierającym żadnej z sztucznie umieszczonych wartości, model w dalszym oznacza jedną z wcześniej niepoprawnie oznaczonych punktów jako anomalię.

3.3.2. HBOS

Kolejnym algorytmem wykorzystanym do uczenia jest Histogram-based Outlier Score (HBOS).

```
from pyod.models.hbos import HBOS

data_hbos = data.copy()

model = HBOS(contamination=0.003)

model.fit(data_hbos[["Temperatura procesora"]])
data_hbos["Anomalia"] = model.predict(data_hbos[["Temperatura procesora"]])

show_anomaly_graph(data_hbos, title='HBOS')
```

Rys. 3.3.2.1 Kod implementujący algorytm HBOS

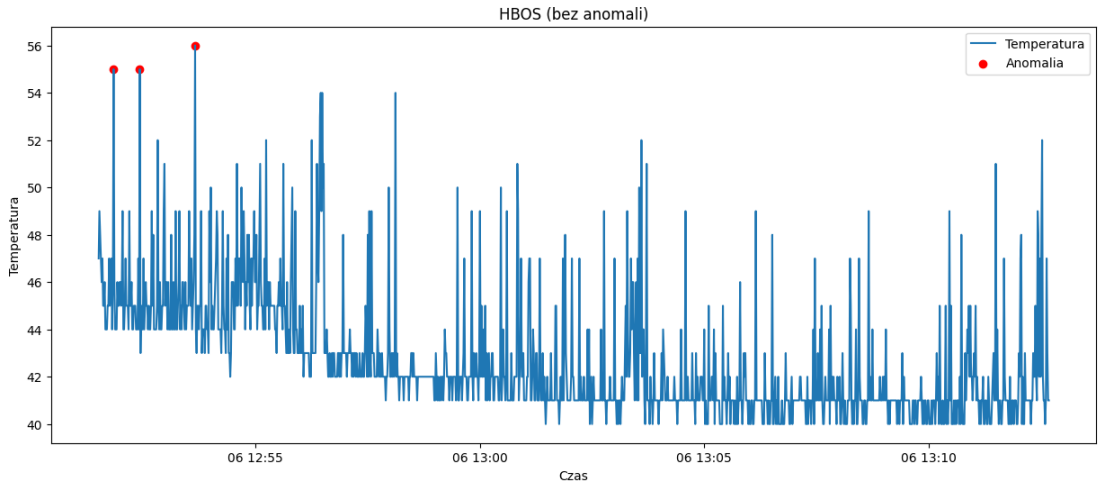


Rys. 3.3.2.2 Wykres wynikowy wytrenowanego modelu

Testowanie modelu na danych bez sztucznie wstawionych anomalii. Algorytm wykrywa poprawnie wszystkie ręcznie dodane anomalie.

```
1  data_hbos_wihout_anomalies = data_without_anomalies.copy()
2
3  model.fit(data_hbos_wihout_anomalies[["Temperatura procesora"]])
4  data_hbos_wihout_anomalies["Anomalia"] = model.predict(data_hbos_wihout_anomalies[["Temperatura procesora"]])
5
6  show_anomaly_graph(data_hbos_wihout_anomalies, title='HBOS (bez anomali)')
```

Rys. 3.3.2.3 Kod przepuszczający dane bez sztucznych anomalii przez model z HBOS



Rys. 3.3.2.4 Wykres wynikowy dla danych bez anomalii

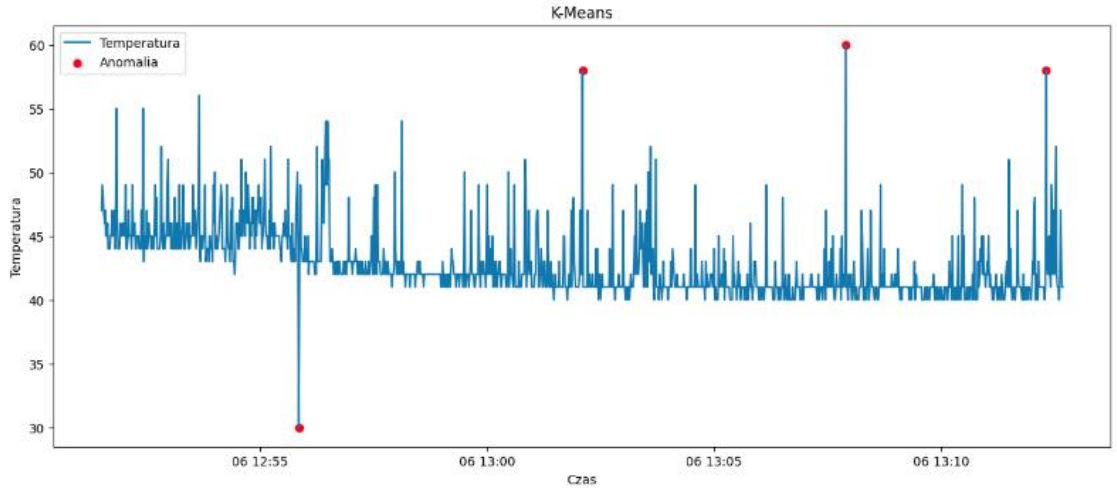
Model błędnie wskazuje anomalie pomimo ich braku w danych.

3.3.3. K-Means

Jako kolejny algorytm zastosowany został algorytm K-Means z biblioteki scikit-learn.

```
from sklearn.cluster import KMeans

data_k_means = data.copy()

kmeans = KMeans(n_clusters=3)
data_k_means['Klaster'] = kmeans.fit_predict(data_k_means[['Temperatura procesora']])

data_k_means['Odlegosc'] = np.linalg.norm(data_k_means[['Temperatura procesora']] - kmeans.cluster_centers_[data_k_means['Klaster']], axis=1)

threshold = np.percentile(data_k_means['Odlegosc'], 99.7)
data_k_means['Anomalia'] = (data_k_means['Odlegosc'] > threshold).astype(int)

show_anomaly_graph(data_k_means, title="K-Means")
```

Rys. 3.3.3.1 Kod implementujący algorytm K-Means

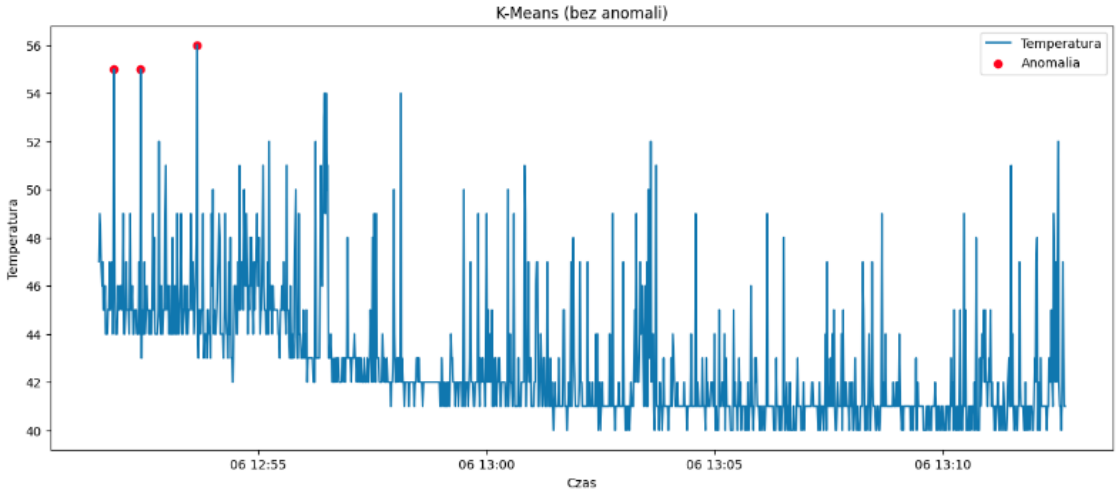


Rys. 3.3.3.2 Wykres wynikowy wytrenowanego modelu

Model na zbiorze treningowym wskazał wszystkie sztucznie dodane anomalie, więc na tej podstawie można go uznać za dobry.

```python
data_k_means_without_anomalies = data_without_anomalies.copy()

data_k_means_without_anomalies['Klaster'] = kmeans.predict(
    data_k_means_without_anomalies[['Temperatura procesora']]
)

data_k_means_without_anomalies['Odlegosc'] = np.linalg.norm(
    data_k_means_without_anomalies[['Temperatura procesora']] -
    kmeans.cluster_centers_[data_k_means_without_anomalies['Klaster']],
    axis=1
)

threshold = np.percentile(data_k_means_without_anomalies['Odlegosc'], 99.7)
data_k_means_without_anomalies['Anomalia'] = (
    data_k_means_without_anomalies['Odlegosc'] > threshold
).astype(int)

show_anomaly_graph(data_k_means_without_anomalies, title="K-Means (bez anomalii)")
```

Rys. 3.3.3.3 Kod przepuszczający dane bez sztucznych anomalii przez model z K-Means



Rys. 3.3.3.4 Wykres wynikowy dla danych bez sztucznych anomalii

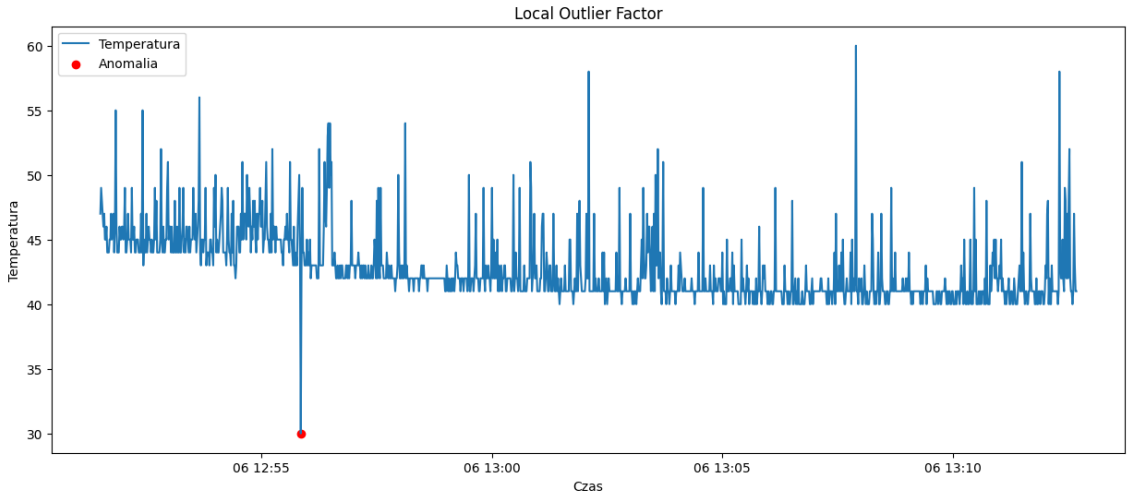
Mimo, że w powyższym zbiorze danych teoretycznie nie powinno być żadnych anomalii, to model wykrył 3 anomalie na samym początku szeregu.

3.3.4. Local Outlier Factor

Ostatnim z algorytmów wykorzystanym w naszej pracy, jest algorytm Local Outlier Factor, którego implementację również można znaleźć w bibliotece scikit-learn.

```
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import LocalOutlierFactor

data_local_outlier_factor = data.copy()

scaler = StandardScaler()
temperatures_std = scaler.fit_transform(data_local_outlier_factor[["Temperatura procesora"]])

lof = LocalOutlierFactor(n_neighbors=5, contamination=0.003, novelty=True)

lof.fit(temperatures_std)

data_local_outlier_factor['Wynik anomali'] = lof.predict(temperatures_std)

data_local_outlier_factor['Anomalia'] = (data_local_outlier_factor['Wynik anomali'] == -1).astype(int)

show_anomaly_graph(data_local_outlier_factor, title="Local Outlier Factor")
```

Rys. 3.3.4.1 Kod implementucjąy algorytm Local Outlier Factor

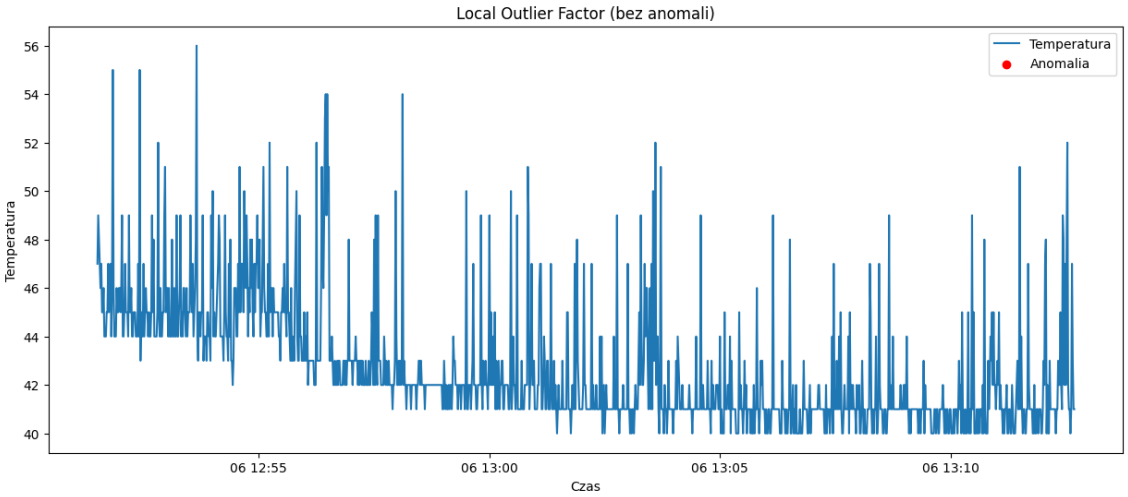


Rys. 3.3.4.2 Wykres wynikowy modelu Local Outlier Factor

Model utworzony przy pomocy algorytmu Local Outlier Factor, wykrył jedną z czterech anomalii utworzonych w serii danych. Jest to wartość odstająca, wskazująca dużo niższą temperaturę od swoich sąsiadów, co sugeruje nam, że model w całkiem dobry sposób jest w stanie nakreślić dolną granicę.

```
data_local_outlier_factor_without_anomalies = data_without_anomalies.copy()

temperatures_std = scaler.transform(data_local_outlier_factor_without_anomalies[["Temperatura procesora"]])

data_local_outlier_factor_without_anomalies['Wynik anomali'] = lof.predict(temperatures_std)
data_local_outlier_factor_without_anomalies['Anomalia'] = (data_local_outlier_factor_without_anomalies['Wynik anomali'] == -1).astype(int)

show_anomaly_graph(data_local_outlier_factor_without_anomalies, title="Local Outlier Factor (bez anomali)")
```

Rys. 3.3.4.3 Kod tworzący wykres dla danych bez anomalii



Rys. 3.3.4.4 Wykres wynikowy modelu Local Outlier Factor dla danych bez anomalii

Wytrenowany model został sprawdzony również na innych danych, które nie zawierały żadnej anomalii. Wówczas nie została wykryta żadna z nich, co jest dobrym znakiem, jako że model nie dopisuje sobie “na siłę” nieistniejących anomalii.

3.4. Porównanie wyników

Przeprowadzone badania wskazują na to, że aż 3 z 4 utworzonych modeli posiadają tendencje do wskazywania anomalii pomimo ich rzeczywistego braku. Jedynym modelem, który nie wskazał anomalii w sytuacji, w której nie powinien jej wskazać jest model oparty na algorytmie Local Outlier Factor. Ten model z kolei miał problemy ze znalezieniem sztucznie utworzonych anomalii, które wskazują na temperaturę wyższą niż przewidywana, a znalazł ten, który jest zaniżony. Z tym problemem dość dobrze poradził sobie Las izolacyjny, który znalazł wszystkie anomalie, które były w górnej części wartości temperatur. Nie znalazł on jednak dolnej anomalii.

Modelami, które spełniły oba z tych warunków są pozostałe utworzone modele oparte na algorytmach: K-Means oraz HBOS, które wykryły wszystkie anomalie jakie zostały załączone do pliku. Problem u nich natomiast następował przy danych, w którym owych anomalii nie było, gdyż zarówno jeden jak i drugi model znajdował je potencjalnie “na siłę”.

4. Wnioski

Celem naszej pracy było znalezienie takiego algorytmu, który jest w stanie wykrywać anomalie w czujnikach temperatur znajdujących się w komputerze. Zgodnie z działaniem komputera, normalnym jest skok temperatury do wyższych wartości podczas większej wydajności np. procesora, a następnie spadek do wartości początkowych. Niestety algorytmy, które zastosowaliśmy miały problemy ze zdiagnozowaniem sytuacji, gdy temperatura się radykalnie zmieniała. Większość z nich w niezły sposób wykrywała anomalie w temperaturze procesora, który był w stanie spoczynku definiując najpewniej granice: górną i dolną, powyżej oraz poniżej której wartości były uznawane za odstające, a co za tym idzie – za anomalie. W związku z tym, nie udało nam się w 100% zrealizować założonego celu. Najprawdopodobniej algorytmy, jakich użyliśmy są niewystarczające do tak skomplikowanego wykresu temperatury, jakim jest temperatura procesora na przestrzeni czasu. W naszym przypadku każdy z modeli wydaje się definiować anomalie globalnie, pomimo że anomalie w temperaturze wyżej wspomnianego procesora powinny być wykrywane lokalnie na podstawie otaczających dany punkt sąsiadów.

Algorytm Local Outlier Factor wydawał się być najbliższy temu zadaniu, ale on również nie podołał wyzwaniu w 100% prawidłowo, gdyż nie wykrywał on wszystkich anomalii, jakie były umieszczone w szeregu danych. Istnieje szansa, iż w przypadku problemu zdefiniowanego jako temat pracy, dobrze sprawdziłyby się autoencodery oraz szeroko pojęte sieci neuronowe, których zastosowanie może dawać bardziej satysfakcjonujące wyniki.

5. Podsumowanie

Rezultatem naszej pracy są utworzone cztery modele uczenia maszynowego nienadzorowanego. Każdy posiada ten sam cel, ale każdy z nich zwraca nieco inne rozwiązanie. Cały proces tworzenia pracy wraz z dobieraniem danych, trenowaniem oraz porównywaniem wyników okazał się być czasochłonnym procesem, który wymagał wielu korekt, a w trakcie realizacji projektu, często stosowana metoda “prób i błędów” pozwalała zrozumieć pewne aspekty działania poszczególnych algorytmów, co następnie ułatwiało dalszą pracę oraz kreację modeli. Na przestrzeni całej pracy jednym z najważniejszym kroków był odpowiedni dobór parametrów oraz danych do modeli, który był wielokrotnie zmieniany w trakcie procesu tworzenia. W zależności od tego jakie parametry oraz dane zostały wykorzystane do trenowania, wyniki różniły się od siebie, co po przebadaniu wielu możliwości pozwoliło dobrać relatywnie optymalny rezultat.

Literatura

[1] Oprogramowanie MSI Afterburner w wersji 4.6.5, https://www.msi.com/Landing/afterburner/graphics-cards (Dostęp 07.06.2024)

[2] Lewiński M.: *Algorytmy i metody uczenia maszynowego*,

[3] G. S. Handelman, H. K. Kok, R. V. Chandra, A. H. Razavi, M. J. Lee, H. Asadi: *eDoctor: machine learning and the future of medicine*

[4] D.Hoang, K.Wiegratz: *Machine learning methods in finance: Recent applications and prospects*

[5] M.Hutsch, T.Wulfert: *A Structured Literature Review on the Application of Machine Learning in Retail*

[6] Olivier C., Bernhard S., Alexander Z.: *Semi-Supervised Learning*

[7] Miśtak S. *Podział modeli uczenia maszynowego wraz z przykładami zastosowania* https://www.gov.pl/web/popcwsparcie/podzial-modeli-uczenia-maszynowego-wraz-z-przykladami-zastosowania (Dostęp 5.06.2024)

[8] Auffarth B.: *Machine Learning for Time-Series with Python: Forecast, predict, and detect anomalies with state-of-the-art machine learning methods*, Packt Publishing, 2021

[9] Wu J.: *Advances in K-means Clustering: A Data Mining Thinking*, Springer, 2012

[10] Mamczur M..; *Las izolacji (Isolation forest) – jak to działa?*; https://miroslawmamczur.pl/las-izolacji-isolation-forest-jak-to-dziala/ (Dostęp 06.06.2024)

[11] Goldstein M. and Dengel A.: *Histogram-based Outlier Score (HBOS): A fast Unsupervised Anomaly Detection Algorithm*, German Research Center for Artifcial Intelligence (DFKI)

[12] Breunig, M. M., Kriegel, H. P., Ng, R. T., & Sander, J.: *LOF: identifying density-based local outliers. In ACM sigmod record*, (2000, May)

[13] Rolon-Mérette, Damien, Ross, Matt, Rolon-Mérette, Thaddé, Church, Kinsey: *Introduction to Anaconda and Python: Installation and setup*, The Quantitative Methods for Psychology, Vol. 16 no 5, 2020

[14] pandas, https://pandas.pydata.org/, 08.06.2024

[15] NumPy, https://numpy.org/, 08.06.2024

[16] matplotlib, https://matplotlib.org/, 08.06.2024

[17] scikit-learn, https://scikit-learn.org/, 08.06.2024

Załącznik 1

Kod źródłowy pracy zaliczeniowej znajduje się w serwisie Github:

https://github.com/wiktorrudzki/MiW