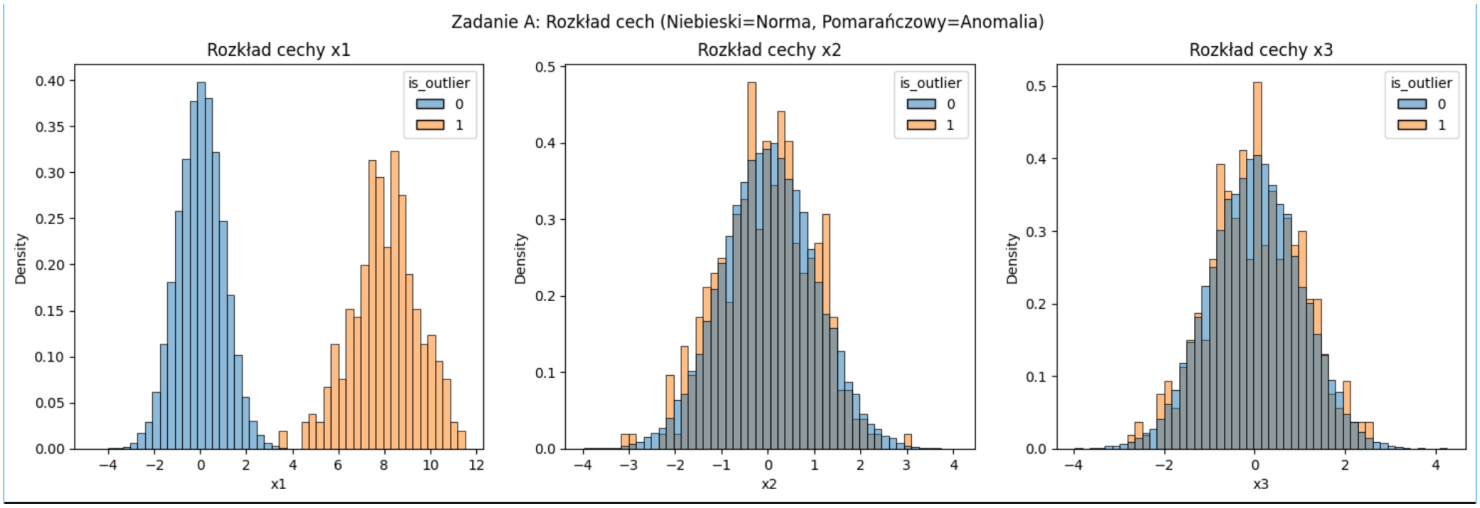
SGH, Aleksander Lasek

Detekcja anomalii w środowisku Big Data

Część A: Analiza danych i kontraktu cech

W celu utrudnienia detekcji zredukowano udział anomalii do 1% i ograniczono zaburzenie wyłącznie do cechy x1 (N(8,1.5)), pozostawiając cechy x2 i x3 zgodne z rozkładem normalnym (N(0,1)). Analiza: Histogramy potwierdzają, że anomalie są widoczne tylko w wymiarze x1, podczas gdy w x2 i x3 są całkowicie ukryte pod rozkładem normalnym.

Detekcja jest trudniejsza niż w wersji bazowej z uwagi na:
Mniejszą odległość euklidesową: Punkt anomalii jest bliżej centrum układu współrzędnych (zaburzenie tylko w 1 z 3 wymiarów).
Szum informacyjny: Autoenkoder łatwo rekonstruuje cechy x2 i x3 (bo są "normalne"), co obniża średni błąd rekonstrukcji (MSE) dla anomalii, zacierając granicę między klasami.



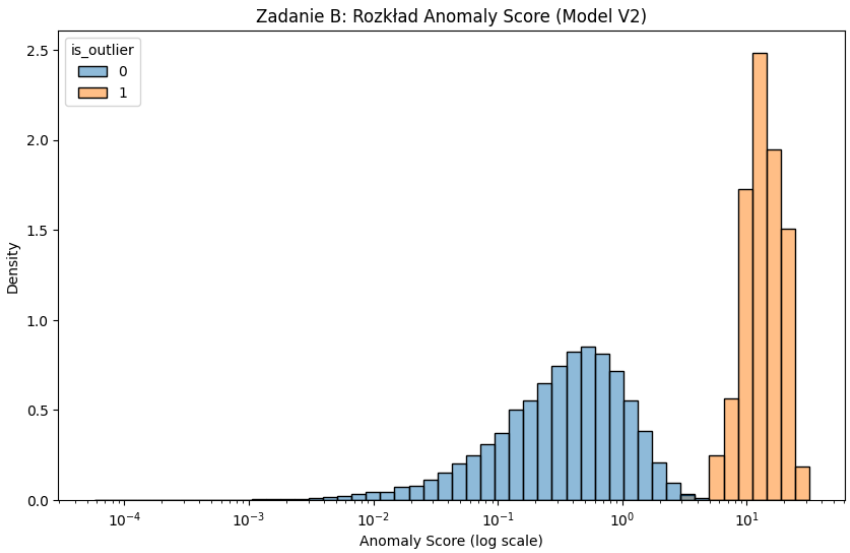
Część B: Architektura autoenkodera

Zmodyfikowano architekturę sieci: zredukowano wymiar przestrzeni latentnej (bottleneck) z 2 do 1 neuronu wymuszając silniejszą kompresję, zmieniono liczbę warstw enkodera poprzez dodanie warstwy Dropout (0.1) oraz rozszerzono warstwy gęste do 16 neuronów.

Wyniki:
Próg (Threshold) wzrósł z ~37 do 40.48, co świadczy o większym błędzie rekonstrukcji (skutek wąskiego gardła).
Mimo zmian, uzyskano TP=1.

Wnioski: Analiza wizualna rozkładu błędu wykazuje idealną separację anomalii (błędy >8) od normy (błędy <1). Niski wynik TP=1 nie wynika ze słabości modelu, lecz z błędu aproksymacji (1%) funkcji approxQuantile przy wyznaczaniu progu.

Pytanie kontrolne: Zbyt duża pojemność sieci w detekcji anomalii grozi przeuczeniem – model mógłby "nauczyć się na pamięć" rzadkich anomalii i rekonstruować je z małym błędem, czyniąc je niewykrywalnymi.



Część C: Definicja progu i interpretacja wyników

W celu weryfikacji hipotezy o błędnym progowaniu i realizacji zadania C1, zastąpiono domyślny próg percentylowy progiem wyznaczonym metodą analizy rozkładu (walidacyjną). Wyniki zestawiono w tabeli poniżej.

Metoda	**Próg**	**TP (na 3000)**	**Recall**	**Wnioski**
1. Oryginalna (approx 1%)	~39.58	1	0.0003	Błąd aproksymacji spowodował ustawienie progu na maksimum zbioru.
2. Precyzyjna (kwantyl 99%)	~5.13	2943	0.9810	Zmniejszenie parametru błędu epsilon pozwoliło poprawnie odciąć 1% danych.
3. Walidacyjna (Wizualna, stała=5.0)	**5.00**	**2951**	**0.9837**	**Próg dobrany na podstawie analizy wykresu B1 ("dziura" między rozkładami) dał najlepszy wynik (realizacja opcji 1 z polecenia).**

Eksperyment potwierdza, że model autoenkodera był poprawny już w poprzednich krokach. Zastosowanie precyzyjnego progu (Metoda 2 lub 3) pozwoliło osiągnąć Recall > 98% przy zachowaniu wysokiej Precyzji (~98%). Oznacza to, że system skutecznie wykrywa subtelne anomalie na cesze x1, ignorując szum na cechach x2 i x3. Kompromis (trade-off) jest widoczny w Metodzie 3: obniżenie progu z 5.13 do 5.00 pozwoliło wykryć 8 anomalii więcej, kosztem 11 fałszywych alarmów (FP).

Część D: Granice systemowe i skalowalność

D1: Wyniki testu wydajności:
Przeprowadzono pomiary czasu przetwarzania dla zmiennego wolumenu danych oraz test wpływu fragmentacji zadań (symulacja wielkości batcha).

Test wolumenu (300k vs 600k):

Czas ETL: Zmiana z 0.60s na 0.58s. Spadek potwierdza, że przy tej skali czas dominowany jest przez narzut uruchomienia zadania, a nie obliczenia.

Czas skoringu: Wzrost z 6.93s na 12.41s (mnożnik 1.79x). Skalowanie jest bliskie liniowemu (oczekiwane 2.0x), co oznacza wysoką efektywność mechanizmu mapInPandas przy odpowiedniej konfiguracji.

Test wielkości partii (Overhead):

Ze względu na blokadę konfiguracji maxRecordsPerBatch w środowisku Serverless (błąd CONFIG_NOT_AVAILABLE), zbadano wpływ wielkości batcha poprzez sterowanie liczbą partycji.

Wyniki: Przy silnej fragmentacji (200 partycji = małe batche) czas wyniósł 25.44s, natomiast przy małej fragmentacji (2 partycje = duże batche) czas spadł do 9.98s.

Wniosek: Przetwarzanie w większych blokach dało 2.55-krotne przyspieszenie. Potwierdza to eksperymentalnie, że koszt serializacji danych między Sparkiem (JVM) a Pythonem jest znaczący i dominuje czas obliczeń przy zbyt małych porcjach danych.

D2: Elementy nieskalujące się liniowo

Zidentyfikowano elementy pipeline'u, które przy dużej skali (TB) nie będą skalować się liniowo:

Koszt serializacji (Overhead): Jak wykazał test D1, narzut komunikacyjny jest stały dla każdej wywołanej funkcji. Przy miliardach małych zadań (mały batch), czas rośnie wykładniczo względem użytecznych obliczeń.

Shuffle (repartition): Operacja przetasowania danych wymaga przesłania ich przez sieć między węzłami. Koszt tej operacji rośnie nieliniowo (O(NlogN)) i jest zależny od przepustowości sieci klastra (Network I/O).

Agregacja globalna (approxQuantile): Wyznaczenie progu w Zadaniu C wymaga zebrania statystyk (częściowych histogramów) ze wszystkich partycji do jednego węzła (Drivera). Driver staje się wąskim gardłem, które nie przyspiesza wraz z dodawaniem nowych Workerów.