UNIVERSITATEA DIN
BUCUREȘTI
VIRTUTE ET SAPIENTIA

Facultatea de
Matematică şi Informatică
Universitatea din București

Detecția anomaliilor în serii de timp

Adela Petre-Șoldan, Maria Cătălina Nica, Alexandru Miclea

(adela.petre-soldan, maria-catalina.nica, alexandru.miclea)@s.unibuc.ro

30 ianuarie 2025

Cuprins

1 Introducere

1.1 Problemă

Detecția anomaliilor pe serii de timp reprezintă identificarea valorilor neobișnuite într-o secvență de date temporale. Aceste anomalii pot semnala evenimente critice, probleme tehnice sau schimbări neașteptate în comportamentul unui sistem.

În cadrul acestui proiect am folosit dataset-ul NVidia Stock Data Latest and Updated - @kalilurrahman, care monitorizează evoluția prețului la deschidere, închidere, valoare maximă, valoare minimă si volum de tranzacție. În mod exact, am comparat viteza de executare și punctele detectate drept anomalii în volumul de tranzacție al indicelui bursier pe anul 2024.

1.2 Justificare

Detectarea acestor anomalii poate fi critică în contextul prevenirii evenimentelor neplăcute (deteriorări ale sistemelor, atacuri cibernetice, etc). De asemenea în industrie, finanțe sau domeniul sănătății întârzierile în identificarea anomaliilor pot duce la pierderi financiare semnificative, avarii sau pot risca sănătatea oamenilor. Prin urmare, dezvoltarea unui sistem automatizat de detectare a erorilor care să poata face față volumului mare de date este indispensabil.

1.3 Biblioteci folosite

- **PyTorch & CUDA & Tensorflow** - Backend pentru modelele de AI
- **Keras** - API pentru dezvoltare
- **Matplotlib** - Prezentare de grafice cu observațiile făcute
- **NumPy** - Bibliotecă pentru prelucrat colecții de date matematic
- **Pandas** - Librărie de manipulat date în format CSV
- **Scikit-Learn** - Funcții de preprocesare a datelor (e.g. StandardScaler)

2 Abordare

2.1 Modele folosite

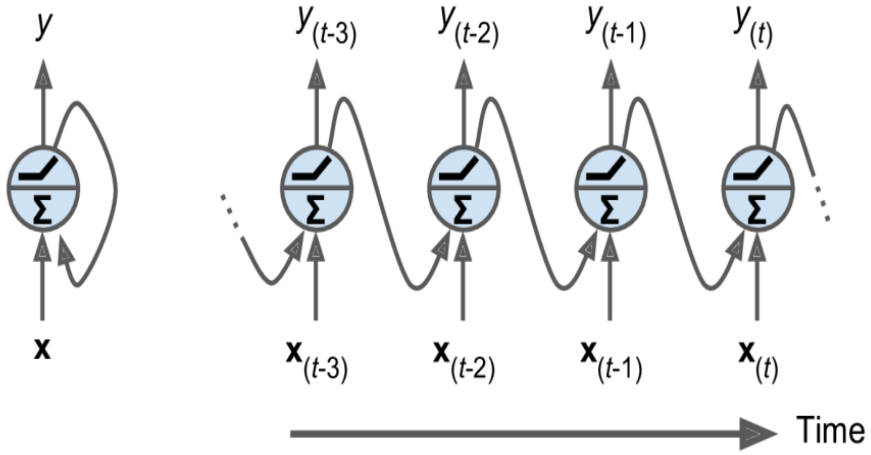
Abordarea noastră a constat în implementarea a cinci modele care să identifice comportament anormal:

- LSTM
- Autoencoders
- PCA
- Isolation Forests
- Transformers

2.2 LSTM

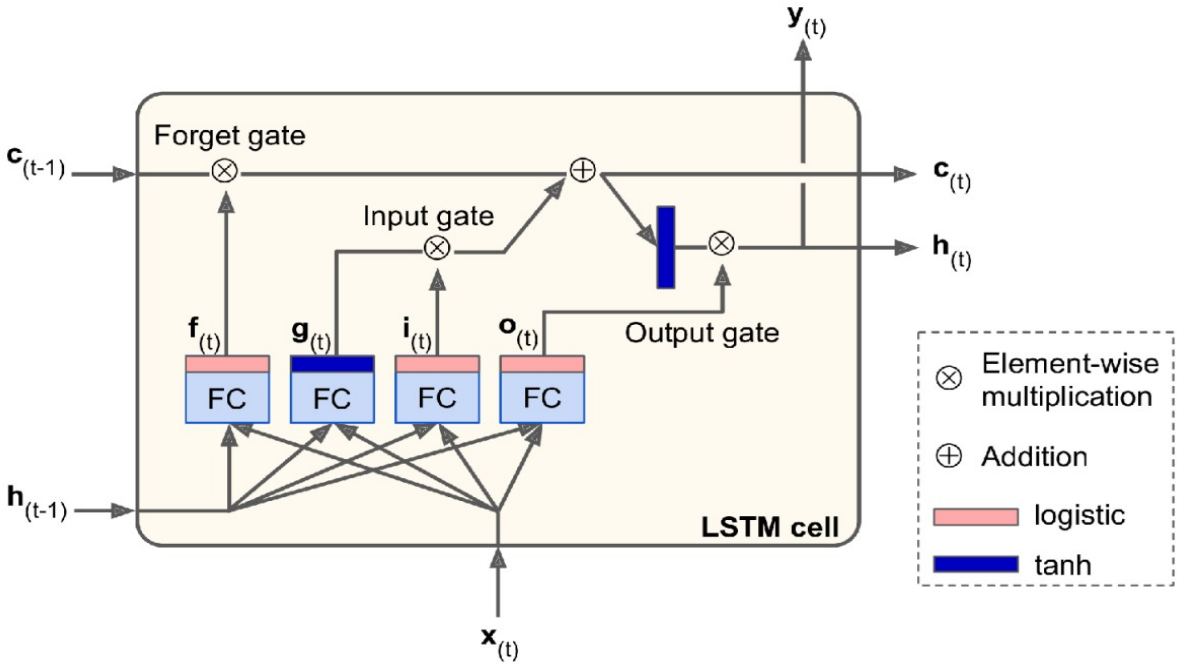
2.2.1 Concepte teoretice

RNN reprezintă o clasă de rețele neuronale folosite de obicei pentru procesarea secvențială a datelor. Structura fundamentală utilizată este unitatea recurentă (recurrent unit) care încorporează o formă de memorie ce este actualizată la fiecare pas în funcție de inputul curent și de starea ascunsă (hidden state) anterioară. Astfel rețeaua reușeste să învețe din inputurile anterioare.



Principalul dezavantaj al RNN-urilor este problema dispariției gradientului (vanishing gradient problem) care le împiedică să rețină dependințe pe termen lung. Acest lucru a fost rezolvat de apariția LSTM in 1997.

O celulă LSTM poate învăța să recunoască un input important (acesta este rolul porții de intrare/input gate), să îl păstreze în starea pe termen lung (long-term state), să îl păstreze atâta timp cât este necesar (acesta este rolul porții de uitare/forget gate), și să o extragă de câte ori e nevoie.



$C_{(t-1)}$ reprezintă starea pe termen lung (*long-term state*), $h_{(t-1)}$ starea pe termen scurt (*short-term state*), iar $x_{(t)}$ inputul curent.

Poarta de uitare ($f(t)$) decide ce informații din trecut sunt eliminate, contribuind la actualizarea memoriei. Funcția $g(t)$ analizează inputul curent împreună cu starea pe termen scurt anterioară, iar poarta de intrare controlează cât din acest input este adăugat la memoria pe termen lung. Poarta de ieșire ($o(t)$) determină ce parte din memoria pe termen lung, acum actualizată, contribuie atât la outputul curent, cât și la actualizarea stării pe termen scurt.

$$
\begin{aligned}
i_t &= \sigma\left(\mathbf{W}_{xi}^\top \mathbf{x}_t + \mathbf{W}_{hi}^\top \mathbf{h}_{t-1} + \mathbf{b}_i\right) \\
f_t &= \sigma\left(\mathbf{W}_{xf}^\top \mathbf{x}_t + \mathbf{W}_{hf}^\top \mathbf{h}_{t-1} + \mathbf{b}_f\right) \\
o_t &= \sigma\left(\mathbf{W}_{xo}^\top \mathbf{x}_t + \mathbf{W}_{ho}^\top \mathbf{h}_{t-1} + \mathbf{b}_o\right) \\
g_t &= \tanh\left(\mathbf{W}_{xg}^\top \mathbf{x}_t + \mathbf{W}_{hg}^\top \mathbf{h}_{t-1} + \mathbf{b}_g\right) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
y_t &= h_t = o_t \odot \tanh(c_t)
\end{aligned}
$$

In aceste ecuatii:

- $\mathbf{W}_{xi}, \mathbf{W}_{xf}, \mathbf{W}_{xo}, \mathbf{W}_{xg}$ sunt matricele de weight-uri ale fiecăruia dintre cele patru straturi pentru conectarea lor la vectorul de input $\mathbf{x}_t$.
- $\mathbf{W}_{hi}, \mathbf{W}_{hf}, \mathbf{W}_{ho}, \mathbf{W}_{hg}$ sunt matricele de weight-uri ale fiecăruia dintre cele patru straturi pentru conectarea lor cu starea short-term anterioara $\mathbf{h}_{t-1}$.
- $\mathbf{b}_i, \mathbf{b}_f, \mathbf{b}_o, \mathbf{b}_g$ sunt bias-urile pentru fiecare dintre cele patru straturi.

2.2.2 Implementare

Am utilizat o rețea LSTM, deoarece aceasta poate modela dependențele pe termen lung și tiparele complexe din seriile de timp. Modelul a fost antrenat pe date din trecut pentru a realiza predicții în viitor.

Am calculat eroarea (loss-ul) dintre predicții și valorile reale, iar dacă aceasta depășea un prag predefinit, punctul respectiv era considerat anomalie.

Modelul creat are stratul de input alcatuit din 64 celule LSTM, iar stratul de output este fully connected de dimensiune 1. Modelul contine un strat LSTM ascuns (hidden layer) de dimensiune 32. Pentru functia de pierdere am folosit MSE, iar pentru funcțiile de activare tanh:

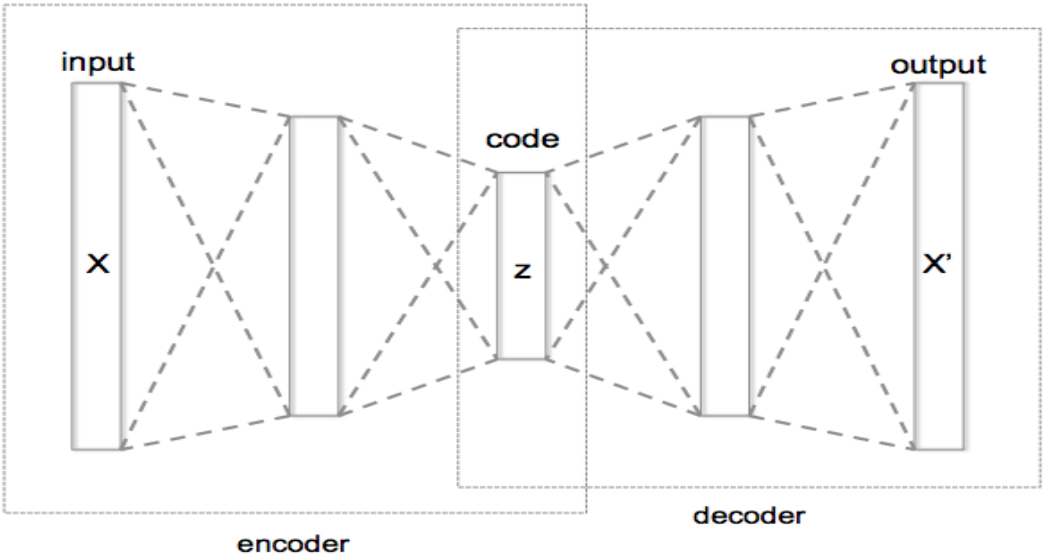
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{y}_i\right)^2 \qquad Tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

2.3 Autoencoders

2.3.1 Concepte teoretice

Autoencoderul reprezintă un tip de rețea neuronală folosit în învățarea nesupervizată. Rolul acestuia este de a deduce o reprezentare mai compactă sau mai dispersată a datelor de intrare (encoding), pe care ulterior să le folosească în a reproduce inputul inițial (decoding). Poate fi considerat un mod de reducere a dimensionalității (fie liniară fie non-liniară), alături de PCA. Modul prin care autoencoderele învață cele două funcții este prin a alege aceeași mulțime de antrenare și validare, minimizând eroarea între datele de intrare și datele rezultate (reconstruction loss).
Un autoencoder este echivalent cu PCA dacă componenta de encoding are un singur strat ascuns iar funcția de activare este funcția identitate.
Definim x ca fiind datele de intrare, $\hat{x}$ datele rezultate, funcția de encoding $f(x)$, funcția de decoding $g(x)$, z datele encodate. Fie următoarele relații între cele anterior definite: $f(x) = z, g(z) = \hat{x}$

Vrem să învățăm funcțiile f și g astfel încât $h(x) = g(f(x)) = \hat{x}$, unde $h(x)$ este o aproximare a funcției identitate.



2.3.2 Implementare

Un mic rundown al implementării din proiect:

- Normalizăm și spargem în ferestre de 12 săptămâni consecutive de trading (de ex. pe test din (166, 1) ma duc in (27, 60))
  - pentru a păstra dimensiunea mulțimii consistentă, eliminăm săptămânile parțiale de trading (săptămâni cu sărbători legale)
- Antrenăm Autoencoderul (funcția de reconstrucție este MSE: $L(x, \hat{x}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$)

- Facem diferența în modul între x și $\hat{x}$ (pe datele normalizate)
- Clasificăm punctele ca fiind anomalii dacă diferența în modul se află la o abatere standard de media diferenței

2.4 PCA

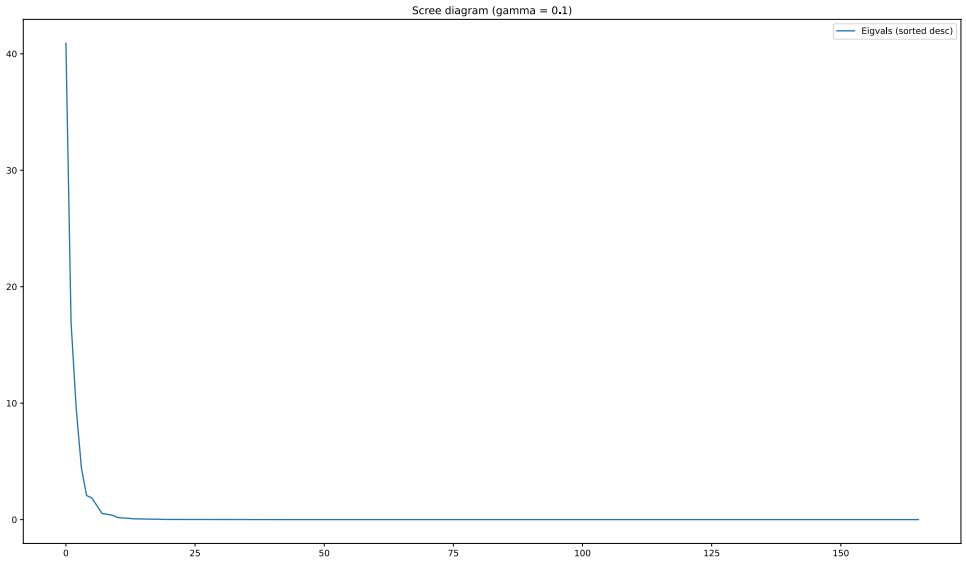
2.4.1 Concepte Teoretice - PCA

PCA (Principal Component Analysis) reprezintă o tehnică liniară de reducere a dimensionalității, obținută prin proiectarea spațiului trăsăturilor originale într-un spațiu de dimensiune mai mică. Componentele principale sunt date de n vectori ai unui spațiu care păstrează direcția în care se regăsește cea mai multă variație în datele originale.

În cazul problemei de detecție de anomalii, ne putem folosi de PCA în următorul fel:

- Strângem datele noastre într-o mulțime de forma (n, d), unde d este numărul de trăsături al unui punct (preț la deschidere, volum)
- Normalizăm datele (funcția de normalizare este $X_{i_{norm}} = \frac{X_i - \mu}{\sigma}$, μ = media, σ = abaterea standard), $i = \overline{1, d}$
- Calculăm matricea de covarianță peste datele normalizate: $\Sigma = \frac{1}{n} X X^t$
- Facem descompunerea în vectori și valori proprii pe această matrice

Pentru a ne asigura că totul a mers bine și pentru a vedea câte componente principale sunt de ales, ne folosim de o diagramă Scree:



O altă condiție de urmărit este ca matricea pe care o descompunem să fie simetrică și pozitiv definită. Definim o matrice ca fiind simetrică dacă $A = A^t$. Fie o matrice simetrică B. Dacă valorile proprii ale matricei B sunt pozitive, atunci matricea B este pozitiv definită.

2.4.2 Concepte Teoretice - Kernel PCA

Un impediment furnizat de un model PCA clasic este faptul că acesta nu poate interpreta relații non-liniare în datele pe care acesta le primește (în cazul dataset-ului pe volum avem a face cu puncte de volatilitate crescută / scăzută care nu pot fi deduse din datele trecute). Pentru a putea deduce rezultate folosind PCA, se poate folosi o funcție kernel în loc de matricea de covarianță.

O funcție nucleu reprezintă o transformare dintr-un spațiu R^d în alt spațiu R^D, $D >> d$. Modul în care putem aplica PCA folosind funcții kernel este următorul:

- Alegem o funcție kernel pe care să o folosim (e.g. RBF Kernel: $K(x, x^{'}) = exp(-\gamma||x - x^{'}||^2))$
- Calculăm matricea kernel (M) pentru elementele din setul nostru de date
- Centrăm matricea folosind următoarea formulă: $M_{centrat} = M - 1_n M - M 1_n + 1_n M 1_n$, unde: $1_n = \frac{1}{n} J_n = \frac{1}{n} \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$
- Facem descompunerea în vectori și valori proprii pe această matrice în locul matricei de covarianță

2.4.3 Implementare

În proiect există implementare atât pentru PCA cât și pentru Kernel PCA. Ce este comun la ambele implementări:

- Vom nota seria de timp cu S, matrice de forma (n, t), unde n reprezintă numărul de puncte eșantionate în seria de timp, iar t numărul de trăsături.
- După descompunerea în valori și vectori proprii sortez lista de valori proprii și matricea de vectori proprii după coloană astfel coloanei E_i să îi corespundă valoarea proprie i
- Aleg un număr p de componente principale (în cazul Kernel PCA, o valoare γ mai mare îmi va "disipa" seria de timp într-un număr mai mare de componente principale)
- Componentele principale vor fi puse într-o matrice de forma (n, p). Vom nota mai departe această matrice cu E
- Proiectez seria de timp originală în spațiul mai mic: $S^{'} = S^t \times E$
- Revin cu seria de timp în spațiul original: $\hat{S} = E \times S^{'t}$ (în cod este scrisă sub forma: $\hat{S} = (S^{'} \times E^t)^t$)
- Facem diferența în modul între S_i și $\hat{S}_i$, i fiind indicele coloanei pe care vrem să o analizăm (volumul în cazul nostru)
- Clasificăm punctele ca fiind anomalii dacă diferența în modul se află la o abatere standard de media diferenței

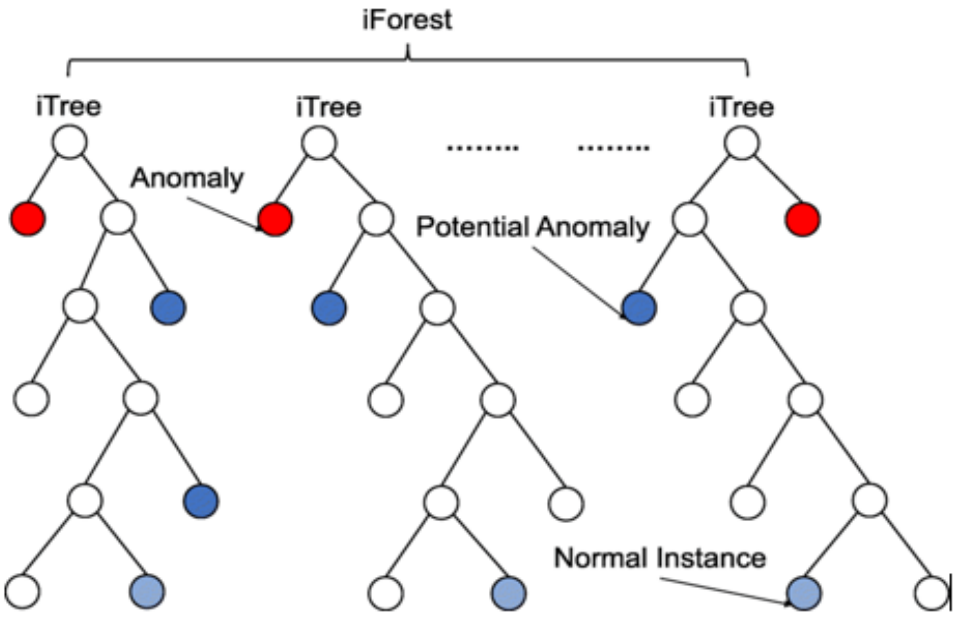
2.5 Isolation Forests

Isolation Forest este un algoritm de învățare nesupervizată, având ca scop principal detecția anomaliilor. Se bazează pe principiul izolării, punctele care sunt anomalii fiind în general izolate mai ușor, prin mai puține operații. Punctele normale fac parte din aglomerări dense, au valori care nu ies din tipare. Foloseste partiționarea binară și recursivă pentru a crea o pădure de arbori.

Primul pas este de a genera aleator n subsecvențe de o dimensiune stabilită (în general 128, 256). O valoare prea mare pentru dimensiunea subsecvențelor nu e eficientă, scade diversitatea pădurii și adâncimea anomaliilor crește, făcând mai dificilă detectarea lor. Aceste subsecvențe vor genera arborii din pădure. Mai mulți arbori permit comparația între mai multe eșantioane diferite. Între 50 si 100 de arbori sunt suficienți pentru majoritatea aplicațiilor. Secvențele pot fi alese și din eșantioane consecutive, astfel că eșantioanele vor fi comparate doar cu cele din vecinătate, detectând mai eficient extremele locale.

Crearea arborilor se face recursiv. La fiecare pas alegem aleator o caracteristică a seriei de timp și o valoare tot aleatoare a acesteia. Eșantioanele vor fi împărțite în 2 subsecvențe. Cele care au valoarea caracteristicii mai mică sau egală decât valoarea aleasă vor deveni fiul stâng, iar celelalte fiul drept. Un nod din arbori reține caracteristica aleasă, valoarea sa și cei 2 subarbori.

Există o valoare maximă pentru adâncimea unui arbore, pentru a nu izola variațiile minore și pentru a preveni supraînvățarea detaliilor nesemnificative, cum ar fi zgomotul. Limitarea adâncimii permite modelului să se concentreze pe tipare generale și nu pe detalii specifice. Pentru o subsecvență de dimensiune n, adâncimea optimă este aproximativ $log_2(n)$. Adancimea se leaga de faptul ca un arbore binar complet cu n frunze are adancimea $log_2(n)$. Asta ar insemna ca toate punctele au fost izolate complet si la adancimi egale.



Pentru etapa de testare, avem nevoie să calculăm coeficientul de anomalie, care este legat de cât de ușor a fost de izolat eșantionul în toți arborii din care face parte: $s(x,n) = 2^{-\frac{E(h(x))}{c(n)}}$

- $E(h(x))$ = media adâncimii la care se află în fiecare arbore
- $c(n)$ = constanta de normalizare

Constanta de normalizare are mai multe formule aproximative, toate fiind legate de lungimea posibila a drumului pentru un punct izolat într-un arbore binar cu n puncte. Scopul constantei este de a normaliza $E(h(x))$.

- Formula clasică: $c(n) = 2H(n-1) - n^2(n-1)$
- $H(n–1)$ = suma armonică $\approx ln(n–1) + \gamma$
- $\gamma = 0.57725$ = constanta lui Euler-Mascheroni
- H(n) =$\sum_{k=1}^{n} \approx ln(n) + \gamma$
- Constanta y apare pentru a compensa diferenta dintre H(n) si ln(n). Pentru c(n) folosim ln(n) + y in loc de H(n) pentru ca este o metoda precisa si mai rapida decat calculul direct al lui H(n), reducand astfel complexitatea.

Pe baza coeficienților obținuți vom stabili anomaliile. Alegem un prag care dacă este depășit indică o anomalie. Pragul poate avea o valoare de peste 0.5, sau putem considera anomalii primele 5 procente din valorile coeficienților. Dacă avem cunoștinte în domeniul datelor cu care lucrăm putem intui un prag potrivit.

2.6 Transformers

2.6.1 Concepte teoretice

Un transformer este o rețea neuronală cu următoarele componente de bază:
Self Attention: permite eșantioanelor să interacționeze unul cu celălalt și să obțină informații de la fiecare dacă are nevoie. Fiecare eșantion are următoarele informații:
Q = ce informație caută un eșantion de la altele
K = ce informație deține fiecare
V = informație deținută în detaliu

- Înmulțind Q cu transpusa lui K obținem scorurile de atenție, adică pentru fiecare eșantion reținem cât de importante sunt celelalte eșantioane pentru el.
- Rezultatului aplicăm funcția softmax, care normalizează scorurile și le transformă în probabilități, suma lor devenind 1. Fără această normalizare, scorurile pot fi oricât de mici sau mari, ceea ce poate duce la situaţii în care unele elemente sunt complet ignorate iar altele suprasolicitate. Formula pentru softmax:$P(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$
- Apoi înmulțim rezultatul cu V, adică fiecare element știe de cât de multă informație are nevoie de la celelalte și o obține. Formula self attention:

$$\text{Attention}(Q,K,V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) * V$$

Positional Encoding: operațiile unui Transformer sunt realizate în mod paralel, nu secvențial, deci e nevoie de o dimensiune nouă pentru a reține pozițiile fiecărui element. Folosim un embedding de o anumită dimensiune(ex: 16, 32), care e un mod de a reprezenta datele într-un format compact. Transformers folosesc des embedding-uri pentru a reprezenta datele în spații vectoriale pentru a permite modelului să învețe eficient. $PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$, $PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$, d=dimensiunea embedding-ului

Eșantioanele au inițial pozițiile (pos) 0, 1, 2, 3,... iar la final devin arrays de dimensiune dim embedding. Fiecare element din acest embedding = poziția inițială a esantionului (0, 1, 2, 3, etc) * factor de scalare. Apoi aplicăm sin pentru embedding par si cos pentru impar. Factorii de scalare sunt numitorii de la atributul sin/cos.

Paralelismul modelului apare în self attention prin înmulțirile de matrici, astfel că punctele interacționează simultan cu alte puncte din secvență.

Feed Forward: Este o rețea neuronală cu un singur sens folosit după fiecare apel Self Attention, cu rol de a aprofunda și mai mult informația. Acesta mărește dimensiunea embedding-ului pentru a permite calcule mai complexe, aplică o funcţie de activare(relu), iar apoi revine la dimensiunea inițială pentru a se potrivi cu următoarele straturi.

Encoder: Conține layer de Self Attention, urmat de layer normalizare, layer de Feed Forward și din nou layer de normalizare.

2.6.2 Implementare

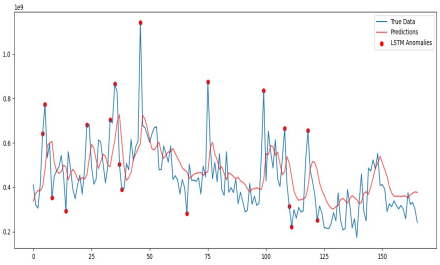
Vom nota batch size = B (numărul de subsecvențe procesate simultan), dimensiune subsecventă = S, dimensiune vector embedding = E. Inputul modelului are dimensiunea (B, S, E). Deci pentru datele de train impărțite pe subsecvențe am adăugat dimensiunea pentru embedding. Se aplică PE și apoi mai multe straturi de encodere. Pentru fiecare encoder, Q, K si V (din Self Attention) sunt layere fully connected la stratul anterior. Își pastreaza dimensiunea (B, S, E), dar aprofundează informația existentă. După ultimul encoder dimensiunea layerului a rămas tot (B, S, E). Aceasta este o structură specifică tensor-ilor din Transformer.

Pentru că în cazul curent prezicem o singură valoare deodată, trebuie să ajungem la un singur număr. Mai întâi aplicăm Global Average Pooling (o operație folosită în rețele neuronale pentru reducerea dimensiunii spațiale a datelor, păstrând informația). Pentru fiecare subsecvență rezumă conținutul său.

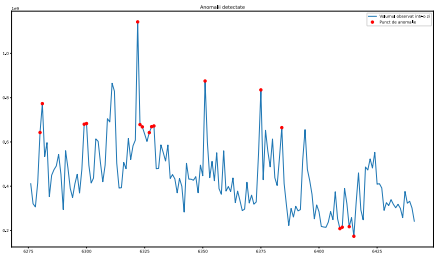
Deci acum layer-ul va avea dimensiunea (B, E). Se observă folosirea embedding-ului pentru reținerea de informații.

Aplicăm un layer de tip Dense (fully connected) cu un singur nod pentru a obţine o singură valoare, rezultatul predicției. Layer-ele Dense sunt utile pentru că toate unitățile lor sunt conectate la toate unitățile stratului anterior, deci reușește să învețe relațiile și să obțină o singură valoare (B, 1).

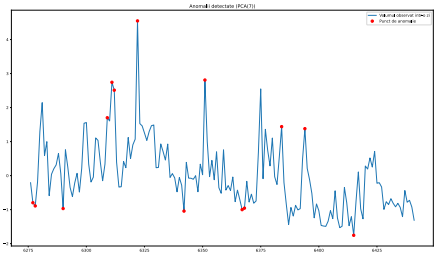
3 Rezultate



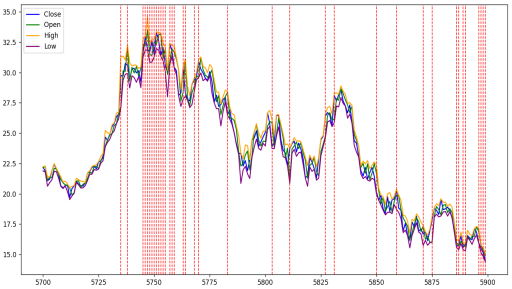
(a) LSTM



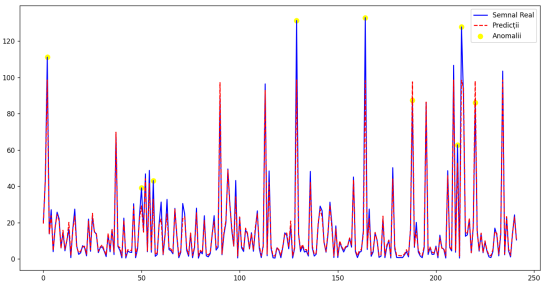
(b) Autoencoder



(c) PCA

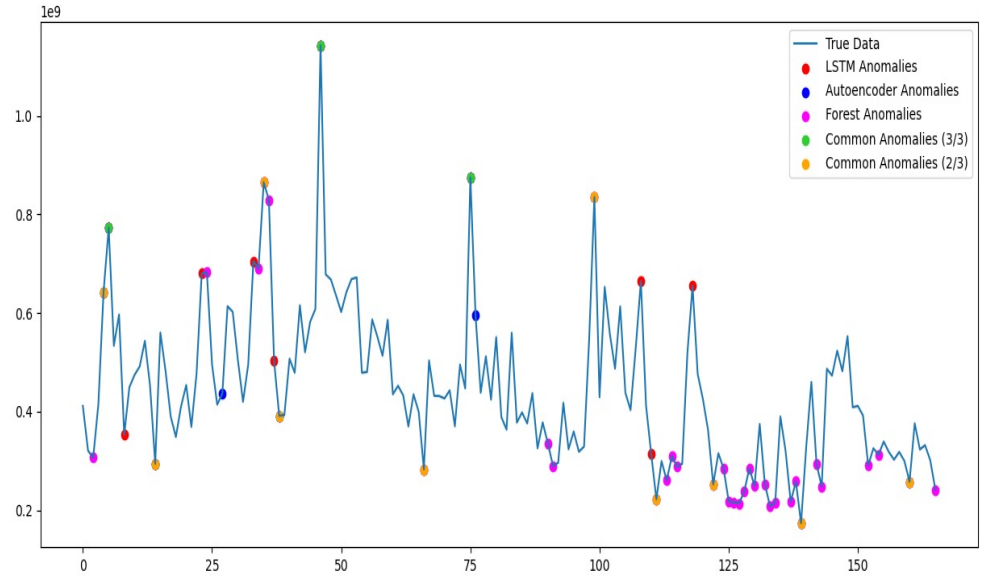


(d) Isolation Forest

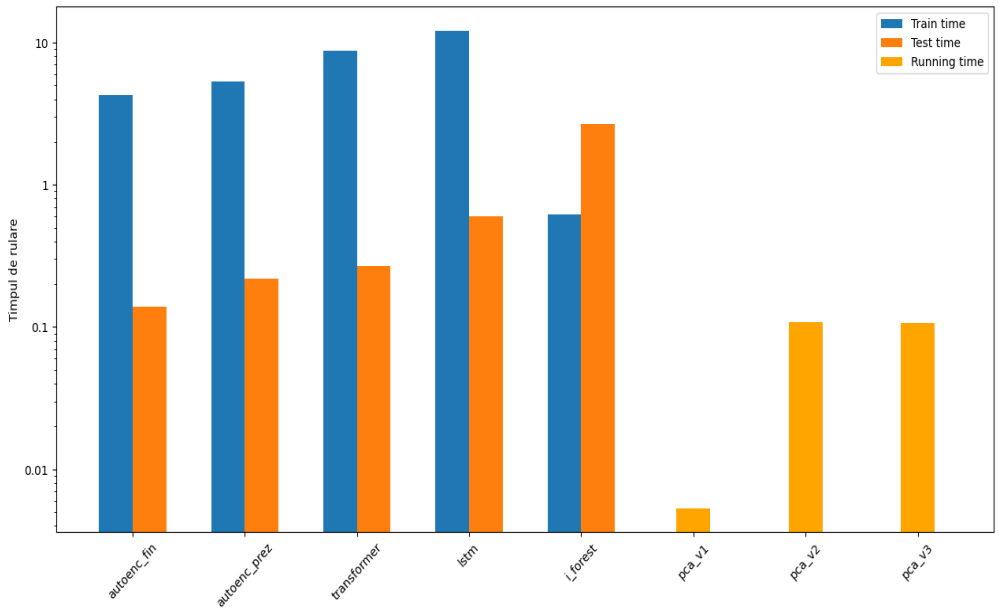


(e) Transformers

- Comparație predicții



- Comparație timpi de rulare



- La autoencoder calculele pot fi făcute simultan, fără dependințe între elementele secvenței. Spre deosebire de LSTM, nu trebuie să proceseze elementele secvențial. De asemenea, nu are nevoie să calculeze atenția între elementele secvenței cum face Transformer.
- LSTM e încetinit de faptul că hidden state trebuie actualizată la fiecare pas.
- Transformer e rapid datorită paralelismului, dar mecanismul de self attention are complexitate mare ($O(n^2 \cdot e)$, e = dimensiune embedding.

4 Concluzii

- Transformer
  - se bazează pe Self Attention pentru a detecta relațiile dintre eșantioane pe perioade foarte mari. Poate înțelege dependințe complexe. De aceea este bun pentru limbaj natural, pentru că percepe întregul context, nu doar cuvintele din vecinătate.
  - nu funcționează bine rulat pe seturi de date imprevizbile și nici dacă e antrenat pe secvențe scurte de date
  - este mult mai potrivit decât Isolation Forest în multe cazuri, pentru că Isolation Forest nu poate captura relaţii complexe între puncte, ci doar detectează puncte izolate statistic. Isolation Forest este util când anomaliile sunt puncte evidente.
- Isolation Forest
  - detectează anomalii atât în serii unidimensionale cât și serii cu mai multe dimensiuni, fără a fi nevoie de ajustări majore.
  - nu se bazează pe distribuția normală a datelor. Metode precum z-score folosesc presupunerea că majoritatea valorilor se află aproape de medie, considerând că valorile care se află cu mult în afara abaterii standard sunt anomalii. Dacă distribuția datelor nu este normală, media și abaterea standard nu mai sunt la fel de utile. Isolation Forest în schimb izolează punctele care sunt diferite de restul, indiferent de modul în care sunt distribuite datele.
- Autoencoders
  - pot fi folosite pentru a detecta anomalii atât pe serii de timp liniare sau neliniare, cât periodice (e.g. ECG) sau non-periodice (e.g. volumul unei acțiuni).
  - prezintă flexibilitate mare, existând multe configurații de rețele, pentru scopuri diferite (variational autoencoders, convolutional autoencoders)
- PCA
  - este o metodă liniară de reducere a dimensionalității, bazat pe descompunerea în vectori și valori proprii a unei matrice
  - merge foarte rapid, dat fiind faptul că nu trebuie antrenat înainte (asta îl poate face folositor pe serii de timp live, unde detecția în timp scurt este critică)
  - nu funcționează bine dacă datele nu sunt liniare, în astfel de cazuri Kernel PCA merge mai bine
- Kernel PCA
  - extinde PCA folosind funcții kernel pentru a putea deduce neliniarități in date
- LSTM - prin arhitectura lor, reușesc să identifice anomaliile prin modelarea comportamentului normal al datelor și detectarea deviațiilor semnificative față de aceste tipare

5 Bibliografie

- Keras - Timeseries Anomaly Detection using Autoencoders
- YouTube (Markus Thill) - Temporal Convolutional Networks
- Principal Component Analysis for Time Series (Medium - @heyamit10)
- Unleashing the Power of Kernel PCA (Medium - @khwabkalra1)
- Centering matrix (Wikipedia)
- What exactly is the procedure to compute principal components in kernel PCA? (Stack Overflow)
- GeeksForGeeks - How Isolation Forests work
- (Link Amazon) Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2nd Edition, Aurélien Géron
- (Medium) Transformer — A detailed explanation from perspectives of tensor shapes and PyTorch implementation.