Project-in-C

☑ Corectie, eliminarea excepțiilor si completarea datelor secvențiale

〰️În ziua de azi, majoritatea problemelor ajung să fie rezolvate cu tehnici de învățare automată. Pentru ca aceste tehnici să funcționeze, este nevoie de cantități semnificative de date. Aceste date provin de obicei dintr-o varietate de surse, iar majoritatea sunt nesigure.
〰️Deoarece în procesul de învățare calitatea datelor determină direct calitatea rezultatelor obținute, este foarte important ca datele să fie corecte și consistente. De aceea, o mare parte din timp, analiștii îl petrec încercând să corecteze și să completeze aceste date.

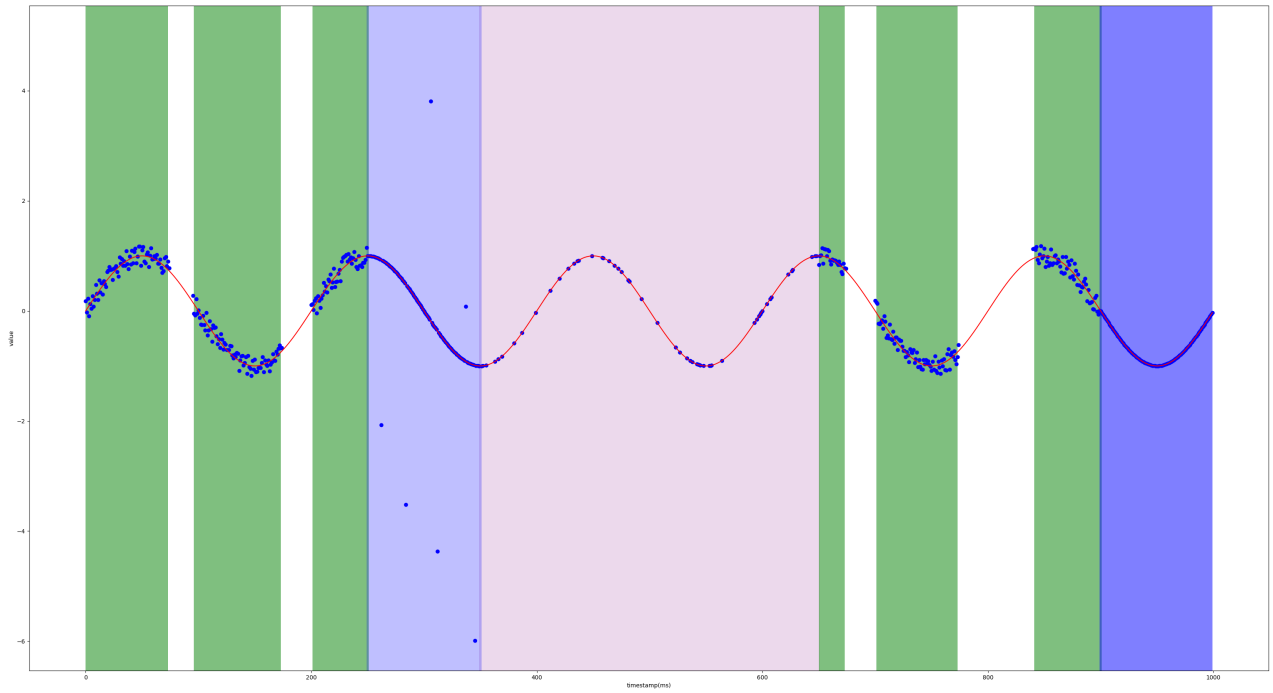
〰️Posibile probleme prezente într-un set de date:

- valori de date ce nu au sens și nu ar trebui să existe (outlier/valori excepționale);
- lipsa datelor în anumite intervale de timp, datorate unor defecțiuni temporare (gaps);
- oscilația prea mare a valorilor (zgomot/noise);
- datele sunt neuniform distribuite în timp (într-o secundă avem 5 înregistrări, în următoarea secundă 22 înregistrări).

- În figura următoare puteți vedea un exemplu de set de date generat de o piesă hardware cu defecte.

  -> fâșiile albe reprezintă intervale fără date;
  -> fâșiile verzi reprezintă intervale de date cu zgomot;
  -> fâșia albastră reprezintă date perfecte;
  -> fâșia cu albastru deschis reprezintă date cu valori de excepție;

-> zona colorată mov reprezintă un interval de date neuniform distribuit în timp. Cu roșu avem reprezentate datele corecte, dacă nu am fi avut nicio problema în procesul de colectare.



Rolul proiectului este de a implementa 4 metode de remediere a problemelor enunțate mai sus, folosind liste.

-> Eliminare de excepții folosind metode statistice

Eliminarea excepțiilor se va realiza prin parcurgerea listei de valori, folosind o fereastră de dimensiune k = 5 pentru care se va calcula la fiecare pas media și deviația standard. Pentru simplitate, am considerat k impar. Fereastra va fi construită în jurul unui nod și nodul respectiv va fi eliminat dacă diferă prea mult de celelalte nodurile din fereastră. Primele int(k/2) și ultimele int(k/2) valori din lista vor fi ignorate în procesul de eliminare și vor rămâne în listă.

-> Eliminare de zgomot folosind filtre

Timestamp-ul datelor rămâne identic cu cel init,ial. Se modifică doar valoarea. Cu alte cuvinte, voi modifca valoarea datelor, nu și timestamp-ul. Pentru fiecare valoare calculată cu unul din filtrele de mai jos, timestamp-ul va fi cel din centrul sublistei folosite în calculul respectiv.

-> Filtrare mediană

Valoarea mediană am obținut-o sortând mulțimea și luând valoarea de poziția din mijloc. Filtrarea mediană funcționează considerând o listă de lungimea k = 5 în jurul fiecărui nod. Pentru fiecare fereastră am considerat valoarea mediană, și am creat o nouă listă din aceste valori.

-> Filtrare folosind media aritmetică

Am parcurs lista de valori considerând subliste de lungime k = 5. Pentru fiecare sublistă, am calculat media aritmetică și am adăugat rezultatul într-o altă listă.

Uniformizarea frecvenței în timp a datelor

Anumite intervale de timp pot să conțină mai multe date decât altele. Ideal, ar fi ca frecvența datelor să fie constantă, însă, datorită modului de colectare sau senzorilor, rareori se întâmplă acest lucru. Astfel am mutat datele din zonele cu frecvență mare către zonele cu frecvența mai mică. Dacă diferența temporală între oricare două date consecutive se află în intervalul $\in [0.1, 1]$ secunde, atunci am înlocuit valoarea nodului curent cu valoarea medie dintre nodul anterior și nodul curent.

Completarea datelor

Dacă intervalele lipsa sunt mai mari de un prag fixat, atunci procesul de completare va fi mai complex decât o simplă medie între margini. Pentru fiecare interval am considerat cei mai apropiați vecini la stânga și la dreapta. Pentru simplitate, am luat k = 3 vecini pentru fiecare margine a intervalului curent. Folosind aceste valori, am aproximat valorile care ar fi trebuit să apară în acel interval, folosind o medie ponderată a vecinilor. Pentru a putea parametriza această formulă, am introdus și un factor de scalare C, care mi-a permis să generez puncte oriunde în întervalul ∈ [right[k].timestamp, lef t[k].timestamp].

Deoarece am vrut ca influența punctelor să scadă pe măsură ce mă depărtez de intervalul curent, am introdus un coeficient care va scădea influența valorilor pe măsură m-am depărtat de interval. Parametrul i semnifică poziția curentă a punctului, iar k reprezintă numărul de puncte luate în considerare.

Am folosit să mă asigur că fiecare interval de 1 secundă din set are cel puțin 5 valori (5Hz).

Statistici

Se consideră intervalul R împărțit în intervale de lungime δ. Pentru fiecare interval am înnumărat de câte ori am un punct în mulțimea de date care se află în acel interval. La final, am afișat pe câte o linie intervalele sortate după marginea inferioară și numărul de elemente ce s-au regăsit în acel interval.

Mod de rulare

Citirea și scrierea se vor face la stdin respectiv stdout.

-> Argumentele pe care programul le poate primi sunt:

- e1 - eliminare de excepții folosind statistici;

- e2 - eliminare de zgomot prin filtru median;
- e3 - eliminare de zgomot folosind filtrul bazat pe media aritmetică;
- u - uniformizarea frecvenței;
- c - completarea datelor lipsă;
- st< delta > - Se vor calcula statisticile conform descrierii din secțiunea “Statistici”. Intervalul de timp δ a fost extras din argument.

-> Exemplu :

./exec −−st1000

-> Argumentele vor fi executate în ordinea în care apar. De exemplu:

./exec −−command1 −−command2 −−command3

Formatul datelor

/* Mai multe exemple de fisiere input se pot gasi in folderul tests/inputs , in functie de fiecare argument / set de argumente atribuindu-se un fisier. */

Modul de reprezentare a datelor:
D = {x|x = (timestamp, value), timestamp ∈ [0, 300000], value ∈ R}

// timestamp - valoarea este primită ca număr întreg, reprezentând timpul în milisecunde.

-> Formatul datelor de intrare/ieșire:

numarPerechi
timestamp valoareReala timestamp valoareReală

...
timestamp valoareReală