Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Vito Abeln

ISKANJE SKRITIH INFORMACIJ V MERITVAH KVALITETE ZRAKA S POMOČJO PODATKOVNEGA RUDARJENJA

Diplomsko delo

Maribor, september 2023

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko

Vito Abeln

ISKANJE SKRITIH INFORMACIJ V MERITVAH KVALITETE ZRAKA S POMOČJO PODATKOVNEGA RUDARJENJA

Diplomsko delo

Maribor, september 2023

ISKANJE SKRITIH INFORMACIJ V MERITVAH KVALITETE ZRAKA S POMOČJO PODATKOVNEGA RUDARJENJA

Diplomsko delo

Študent(ka):	Vito Abeln
Študijski program:	visokošolski študijski program
	Informatika in tehnologije komuniciranja
Smer:	Razvoj informacijskih sistemov
Mentor(ica):	doc. dr. Iztok Fister ml.
Lektor(ica):	Patricija Gril

ZAHVALA

Zahvalil bi se rad mentorju doc. dr. Iztoku Fistru za vso podporo in pomoč. Rad bi se zahvalil tudi svojim domačim in prijateljem za podporo pri študiju.

Iskanje skritih informacij v meritvah kvalitete zraka s pomočjo podatkovnega rudarjenja

Ključne besede: rudarjenje asociativnih pravil, kvaliteta zraka, asociativna pravila

UDK: 004.62(043.2)

Povzetek

Diplomska naloga predstavlja postopek iskanja skritih informacij iz meritev kvalitete zraka s pomočjo rudarjenja asociativnih pravil. Rudarjenje asociativnih pravil je tehnika, preko katere lahko pridobimo zanimive povezave med podatki iz večjih podatkovnih množic. V zaključnem delu smo prikazali postopek pridobivanja podatkov, obdelavo podatkov, razlago uporabljenih algoritmov in njihovo implementacijo. Opisali smo algoritme Apriori, ECLAT in Fp-growth, ki se uporabljajo pri rudarjenju asociativnih pravil. Predstavili smo tudi numerično rudarjenje asociativnih pravil, pri katerem smo uporabili algoritem optimizacije roja delcev. Rezultati rudarjenja so razkrili različne povezave med vrednostmi meritev, ki smo jih razložili in vizualizirali s pomočjo raznih grafov.

Finding hidden information in air quality measurements with data mining

Keywords: association rule mining, air quality, association rules

UDC: 004.62(043.2)

Abstract

This thesis presents a procedure for finding hidden information from air quality measurements using associtation rule mining. Association rule mining is a technique through which interesting associations between data from large datasets can be extracted. In this thesis we show the data mining process, data processing, explanation of the algorithms used and their implementation. We describe the Apriori, ECLAT and Fp-growth algorithms used in associative rule mining. We also presented numerical associative rule mining which used the particle swarm optimisation algorithm. The mining results revealed different relationships between the measurement values, which we explained and visualised using different graphs.

IZJAVA O AVTORSTVU ZAKLJUČNEGA DELA

Univerza v Mariboru

Fakulteta za elektrotehniko,
računalništvo in informatiko
Koroška cesta 46
2000 Maribor, Slovenija

FERI

IZJAVA O AVTORSTVU ZAKLJUČNEGA DELA

Ime in priimek študent-a/-ke: Vito Abeln
Študijski program: Informatika in tehnologije komuniciranja
Naslov zaključnega dela: Iskanje skritih informacij v meritvah kvalitete zraka s pomočjo podatkovnega rudarjenja
Mentor: Doc. dr. Iztok Fister ml.

Podpisan-i/-a študent/-ka Vito Abeln

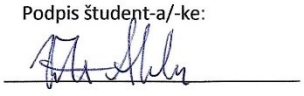
- izjavljam, da je zaključno delo rezultat mojega samostojnega dela, ki sem ga izdelal/-a ob pomoči mentor-ja/-ice oz. somentor-ja/-ice;
- izjavljam, da sem pridobil/-a vsa potrebna soglasja za uporabo podatkov in avtorskih del v zaključnem delu in jih v zaključnem delu jasno in ustrezno označil/-a;
- na Univerzo v Mariboru neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve avtorskega dela v elektronski obliki, pravico reproduciranja ter pravico ponuditi zaključno delo javnosti na svetovnem spletu preko DKUM; sem seznanjen/-a, da bodo dela deponirana/objavljena v DKUM dostopna široki javnosti pod pogoji licence Creative Commons BY-NC-ND, kar vključuje tudi avtomatizirano indeksiranje preko spleta in obdelavo besedil za potrebe tekstovnega in podatkovnega rudarjenja in ekstrakcije znanja iz vsebin; uporabnikom se dovoli reproduciranje brez predelave avtorskega dela, distribuiranje, dajanje v najem in priobčitev javnosti samega izvirnega avtorskega dela, in sicer pod pogojem, da navedejo avtorja in da ne gre za komercialno uporabo;
- dovoljujem objavo svojih osebnih podatkov, ki so navedeni v zaključnem delu in tej izjavi, skupaj z objavo zaključnega dela.

Uveljavljam permisivnejšo obliko licence Creative Commons: CC BY 4.0

Kraj in datum:

26. 3. 2023

Podpis študent-a/-ke:



KAZALO VSEBINE

KAZALO SLIK

KAZALO TABEL

KAZALO IZSEKOV KODE

KAZALO PSEVDOKODE

UPORABLJENI SIMBOLI IN KRATICE

ARM – rudarjenje asociativnih pravil (ang. *associations rule mining*)

ARSO – Agencija Republike Slovenije za okolje

GPIO – splošnonamenski vhod/izhod (ang. *general purpose input/output*)

HTTP – protokol za prenos hipertekstovnega besedila (ang. Hypertext Transfer Protocol)

JSON – objektni zapis JavaScript (ang. *JavaScript Object Notation*)

NARM – numerično rudarjenje asociativnih pravil (ang. *Numerical Association Rule Mining*)

nm – nanometer

PM – prašni delci (ang. *particulate matter*)

PSO – optimizacija roja delcev (ang. *particle swarm optimization*)

PyPI – indeks Python paketov (ang. *Python package index*)

REST – prenos predstavitvenega stanja (angl. *representational state transfer*)

TSMC – Tajvansko podjetje za proizvodnjo polprevodnikov (ang. Taiwan semiconductor manufacturing company)

µm – mikrometer

1 UVOD

1.1 Opredelitev problema

Zaradi vse večjega onesnaževanja je kvaliteta zraka postala ena izmed pomembnejših metrik, ki smo jo morali začeti spremljati. Prav pocenitev naprav za merjenje kvalitete zraka nam je omogočila lažji dostop in večjo količino izmerjenih podatkov. Ker so podatki postali dostopnejši in številčnejši, so posledično tudi primernejši za podatkovno rudarjenje [1]. Količina podatkov, ki jih dandanes zajemamo in obdelujemo, postaja vedno večja. Vse več podatkov pomeni, da je vse več tudi neraziskanih. S hitro rastjo procesorske moči in količine podatkov, ki so na voljo, se je razvila disciplina podatkovnega rudarjenja, ki je zelo multidisciplinarno področje. Zgled jemlje iz znanstvenih področij, kot so statistika, matematika, računalništvo, fizika in inženirstvo. Na področje podatkovnega rudarjenja so imele največji vpliv discipline, kot so »statistika z uporabo statističnih metod in vizualizacijo podatkov, umetna inteligenca z uporabo metod strojnega učenja, metode računske inteligence in sistemi podatkovnih baz« [2].

Preko podatkovnega rudarjenja lahko iz številnih obstoječih podatkov odkrivamo nove informacije ali napovedujemo določene dogodke oziroma trende. Pri podatkovnem rudarjenju obstajata dve paradigmi algoritmov, ki sta napovedovanje in odkrivanje informacij. Znotraj teh dveh paradigem so še podkategorije, kot so klasifikacija in regresija, gručenje, rudarjenje asociativnih pravil in iskanje osamelcev. Pri tej nalogi se bomo osredotočili predvsem na rudarjenje asociativnih pravil, ki opisuje odkrivanje relacij med atributi v transakcijski podatkovni bazi. Cilj je poiskati skrite informacije, ki se pojavljajo v obliki asociacij oziroma povezav med atributi izmerjenih podatkov kvalitete zraka [3].

V tem diplomskem delu bomo predstavili postopek rudarjenja asociativnih pravil na praktičnem primeru. Predstavili bomo proces od začetka, kjer se podatki pridobivajo, do konca, kjer preko uporabe podatkovnega rudarjenja pridemo do analize pridobljenih asociacij.

1.2 Cilji diplomskega dela

Namen diplomskega dela je prikazati proces rudarjenja asociacijskih pravil na praktičnem primeru s podatki kvalitete zraka.

Glavni cilji:

- Razložiti in prikazati celoten proces od pridobivanja podatkov, obdelovanja podatkov, uporabe algoritmov rudarjenja asociativnih pravil do prikaza ugotovitev.
- Opisati postopek izdelave in implementacije mikrokrmilnika s senzorjem za zajem podatkov kvalitete zraka.
- Opisati pridružen spletni vmesnik, ki deluje kot repozitorij za podatke.
- Razložiti pridobljene podatke, postopek obdelave in čiščenja podatkov.
- Predstaviti teorijo za delovanje posameznih algoritmov za rudarjenje.
- Prikazati uporabo različnih algoritmov za rudarjenje asociacijskih pravil na tej množici.
- Predstaviti rezultate algoritmov.

1.3 Predpostavke in omejitve diplomskega dela

Pri zbiranju podatkov meritev bomo uporabili senzor SEN55, ki je namenjen predvsem neprofesionalni rabi. Posledično bodo lahko pri meritvah vrednosti odstopale od realnih vrednosti, saj njegova natančnost ni primerljiva s profesionalnimi napravami. Merilna naprava bo postavljena na okenski polici in je lahko v določenem delu dneva neposredno izpostavljena sončni svetlobi. Zato bodo lahko naše izmerjene temperaturne vrednosti popačene in ne bodo 100 % reflektirale dejanskih vrednosti. Če bi imeli finančna sredstva za profesionalno opremo in možnost za boljšo postavitev merilne naprave, bi lahko naše meritve izboljšali. Predvidevamo, da bomo lahko s procesom čiščenja in obdelave podatkov določene napake in odstopanja odpravili.

Z meritvami naše merilne postaje bomo v štirih mesecih meritev zgradili dovolj veliko podatkovno množico, nad katero bomo lahko demonstrirali potek rudarjenja asociativnih pravil. Predvidevamo, da bo naša količina meritev obsegala približno 18.000

vrstic, kar za podatkovne množice ni ravno veliko, ampak za naš primer popolnoma dovolj. Meritve merilne postaje združene s podatki iz ARSO postaj nam bi morale zagotoviti dovolj obsežno zbirko podatkov z raznolikimi atributi, iz katere bomo lahko izluščili zanimive informacije.

S pomočjo knjižnic rudarjenja asociativnih pravil bomo implementirali zastavljene algoritme in izvedli podatkovno rudarjenje nad našo podatkovno množico. Predvidevamo, da za rudarjenje pravil ne bomo potrebovali veliko računske moči, saj je podatkovna množica precej majhna. Rezultate rudarjenja asociativnih pravil bomo predstavili na pregleden in razumljiv način.

1.4 Raziskovalna vprašanja

RV1 – Kako in na kakšen način pridobivamo podatke, primerne za podatkovno rudarjenje?

RV2 – Kako delujejo algoritmi za rudarjenje asociativnih pravil?

RV3 – Kateri algoritmi za rudarjenje asociativnih pravil so primernejši za našo podatkovno množico?

RV4 – Kako predstaviti in razložiti rezultate rudarjenja asociativnih pravil?

RV5 – Kakšne skrite informacije lahko odkrijemo iz podatkov lastnih meritev?

1.5 Struktura diplomskega dela

Diplomsko delo je sestavljeno iz sedmih poglavij. V začetnem poglavju bomo predstavili splošno tematiko podatkovnega rudarjenja ter cilje in metode raziskovanja. V drugem poglavju se bomo posvetili predvsem razlagi teorije za ARM kot tudi NARM. V tretjem poglavju bomo predstavili predlagane metode, kjer bomo opisali postopek pridobivanja podatkov, shranjevanja in uporabe podatkov v skripti ter predstavili postopek čiščenja in obdelave ter diskretizacijo podatkov. V četrtem poglavju bomo predstavili vse uporabljene algoritme in tudi njihovo implementacijo za naš primer. V petem poglavju

bomo predstavili primerjavo ARM algoritmov ter rezultate in ugotovitve za kategorične in numerične podatke. V šestem poglavju bomo odgovorili na prej zastavljena raziskovalna vprašanja. V zadnjem sedmem poglavju bomo povzeli celotno diplomsko delo.

2 PODATKOVNO RUDARJENJE

Podatkovno rudarjenje je proces odkrivanja podatkov, informacij in vzorcev v velikih podatkovnih zbirkah. Napredovanje v preteklih letih je omogočilo skokovit napredek na tem področju, da lahko pretvorimo surove podatke v uporabno znanje [2].

Podatkovno rudarjenje lahko razdelimo na dve veji. Če ga delimo po tem, za kaj se podatkovno rudarjenje uporablja, je prva veja opisovanje podatkovne množice, druga pa napovedovanje določenih izidov s pomočjo umetne inteligence. Ti pristopi omogočajo tako organizacijo in filtriranje podatkov kot tudi izpostavitev najzanimivejših informacij.

Proces poteka podatkovnega rudarjenja lahko opišemo s štirimi koraki:

1. postavitev ciljev,
2. priprava podatkov,
3. grajenje modela in rudarjenje vzorcev,
4. pregled rezultatov in implementacija znanja.

Takšen pristop bomo tudi v tej diplomski nalogi uporabili za odkrivanje skritih informacij [3].

2.1 Rudarjenje asociativnih pravil

Rudarjenje asociativnih pravil (ang. associations rule mining, krajše ARM) je način odkrivanja asociacij oziroma povezav med elementi v transakcijski bazi. Ta koncept lahko predstavimo s pomočjo primera nakupovalne košarice, s čimer predstavimo, kateri izdelek se najpogosteje pojavlja z drugim. V našem primeru bi to pomenilo, katera vrednost parametra se pojavlja z drugo vrednostjo. To asociacijo lahko matematično definiramo kot implikacijo A ⇒ B. Kjer je spremenljivka A znana kot predpostavka (ang. *antecedent*) in spremenljivka B kot posledica (ang. *consequence*). Slednje lahko interpretiramo kot: če v naši transakcijski bazi obstaja element A, potem obstaja tudi element B. Pri rudarjenju asociativnih pravil število asociacij med podatki eksponentno narašča s številom elementov v podatkovni množici. Da omejimo najdene asociacije, si

pomagamo s pravili, ki pomagajo določiti uporabnost najdene asociacije. To so podpora (ang. *support*), zaupanje (ang. *confidence*) in dvig (ang. *lift*). Podpora asociacijskega pravila A ⇒ B je odstotek transakcij t znotraj vseh transakcij T, ki vsebujejo A ∪ B. Vrednost podpore nam pove, kako pogosto se asociacijsko pravilo pojavi v celotnem setu. Podporo lahko izračunamo s pomočjo enačbe (1). Zaupanje asociacijskega pravila A ⇒ B pa je razmerje med številom transakcij, ki vsebujejo A ∪ B, ter številom transakcij, ki vsebujejo samo A. Vrednost zaupanja nam pove moč oziroma zanesljivost pravila. Vrednost zaupanja lahko izračunamo s pomočjo enačbe (2). Dvig lahko interpretiramo kot odklon podpore od celotnega pravila glede na podporo A in B. Višje vrednosti dviga predstavljajo močnejše asociacije. Vrednost dviga lahko izračunamo s pomočjo enačbe (3) [4] [5].

$$sup(A \Rightarrow B) = \frac{|\{t \in T,\ \ A \cup t\}|}{|T|} \tag{1}$$

$$conf(A \Rightarrow B) = \frac{sup(A \cup B)}{sup(A)}\ [2] \tag{2}$$

$$lift(A \Rightarrow B) = \frac{sup(A \cup B)}{sup(A) * sup(B)} = \frac{conf(A \Rightarrow B)}{sup(B)}\ [5] \tag{3}$$

Za rudarjenje se uporabljajo razni algoritmi, kot so:

- Apriori,
- Fp-growth,
- ECLAT [6].

2.2 Numerično rudarjenje asociativnih pravil

Numerično rudarjenje asociativnih pravil (ang. numerical association rule mining, krajše NARM) je iz rudarjenja asociativnih pravil izpeljana metoda. Omogoča namreč hkratno delo z numeričnimi in kategoričnimi podatki. Deluje na isti osnovi kot ARM z merami podpore, zaupanja in dviga. Pravilo numeričnega asociacijskega pravila lahko predstavimo v obliki implikacije A ⇒ B, kjer sta predpostavka in posledica za množico

kategoričnih atributov predstavljeni v obliki množice A = {a_1, a_2, … a_n}, za numerične pa v obliki intervala A ∈ [a_1, a_2] [7]. NARM lahko modeliramo kot optimizacijski problem ter ga rešujemo s populacijskimi algoritmi po vzorih iz narave, pri čemer je vrednost ustreznosti (ang. *fitness*) seštevek uteži z vrednostmi podpore in zaupanja za določeno pravilo. Vrednost ustreznosti predstavlja, kako ustrezno je določeno pravilo glede na postavljene pogoje.

Večina NARM algoritmov temelji na stohastičnih populacijskih algoritmih po vzorih iz narave. Eden izmed načinov implementacije algoritmov po vzorih iz narave je uporabiti že obstoječe implementacije algoritmov. Za to obstaja v Pythonu knjižnica po imenu NiaPy, ki nam nudi skupek pripravljenih algoritmov po vzorih iz narave z enostavnim postopkom implementacije. Primeri podprtih algoritmov po vzorih iz narave v knjižnici NiaPy [8]:

- Differential Evolution,
- Bat Algorithm,
- Particle Swarm Optimization,
- … [9]

3 PREDLAGANA METODA

Predlagana metoda sestoji iz naslednjih korakov:

- pridobivanje podatkov z lastno merilno napravo,
- shranjevanje meritev in združevanje lastnih meritev z zunanjimi,
- uporaba in obdelava podatkov,
- podatkovno rudarjenje.

Vsi omenjeni koraki so predstavljeni v naslednjih podpoglavjih.

3.1 Pridobivanje podatkov

Kot metodo pridobivanja podatkov smo uporabili kombinacijo prosto dostopnih meritev iz profesionalnih merilnih postaj in lokalnih meritev, ki bi jih izvedli z lastno merilno napravo. S tem smo pridobili redundantnost določenih podatkov in širši nabor izmerjenih podatkov. Kot lokacijo za meritve smo si izbrali mesto Maribor, kjer smo imeli priložnost postaviti lastno merilno postajo. Poleg naše lastne merilne postaje se v Mariboru nahajata še dve merilni postaji, in sicer na Titovi in Vrbanski cesti. Podatki teh dveh merilnih postaj so prosto dostopni na spletni strani ARSO, hkrati pa lahko do njih dostopamo preko njihovih spletnih vmesnikov. Vsi podatki s spletne strani ARSO so zaščiteni pod licenco CC-BY-ND 4.0, kar nam omogoča prosto uporabo, dokler navedemo avtorja in podatkov ne predelamo [10]. Sami smo meritve opravljali na lokaciji Slomškov trg v središču Maribora. Večji del podatkov smo izmerili z lastno merilno postajo. Pridobljene podatke smo nato združili z meritvami oz. podatki iz prej omenjenih merilnih postaj ARSO. Merjeni podatki vsebujejo identifikacijsko številko meritve, vrednosti prašnih delcev velikosti 1, 2,5, 4, in 10 µm, relativno vlažnost, volatilne organske spojine, dušikove okside, čas meritve, žveplov dioksid, ozon, dušikov dioksid in benzen. Omenjene merjene vrednosti bomo podrobneje predstavili v naslednjem podpoglavju. Ker ocenjujemo, da pridobljeni podatki iz treh virov dobro pokrivajo celotno mesto Maribor, lahko meritve generaliziramo na celotno mesto, in ne veljajo samo za določeno lokacijo.

3.2 Razumevanje podatkov

Podatki, ki jih merimo z lastno merilno postajo, so:

- **PM1**: Prašni delci velikosti od 0,3 µm do 1,0 µm. Zaloga vrednosti je od 0 do 1000 µg/m^3. PM1 predstavljajo delci, kot so prah, prašni delci izpustov, bakterije in virusi.
- **PM2.5**: Prašni delci velikosti od 0,3 µm do 2,5 µm. Zaloga vrednosti je od 0 do 1000 µg/m^3. PM2,5 predstavljajo delci, kot so cvetni prah, spore in drugi organski delci.
- **PM4:** Prašni delci velikosti od 0,3 µm do 4 µm. Zaloga vrednosti je od 0 do 1000 µg/m^3.
- **PM10**: Prašni delci velikosti od 0,3 µm do 10 µm. Zaloga vrednosti je od 0 do 1000 µg/m^3. PM1 predstavljajo delci, kot so fin prah in organski delci [11], [12].
- **H**: Relativna vlažnost. Predstavlja razmerje med parnim tlakom zraka in njegovim nasičenim parnim tlakom. Zaloga vrednosti je od 0 do 100 % [13], [12].
- **T**: Temperatura. Zaloga vrednosti je od -10 do 50 °C.
- **VOC**: Volatilne organske spojine. Zaloga vrednosti je od 1 do 500 indeksnih točk. Volatilne organske spojine se sproščajo v zrak kot plini iz določenih trdnih ali tekočih snovi. Te snovi so v večini človeškega izvora, npr. goriva iz petroleja, hidravlične tekočine, razredčila za barvo itd. [14], [12].
- **NOx**: Dušikovi oksidi. Zaloga vrednosti je od 1 do 500 indeksnih točk. V to skupino spadajo plini, kot sta dušikov monoksid »NO« in dušikov dioksid »NO_2« [15], [12].

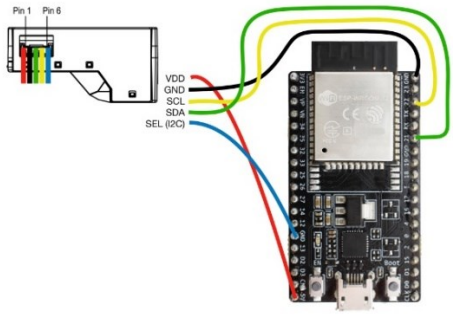
Meritve, ki jih pridobivamo iz zunanjih merilnih postaj ARSO:

- **O3**: Ozon. Zaloga vrednosti je od 0 do 10000 µg/m^3 [16].
- **NO2**: Dušikov dioksid. Zaloga vrednosti je od 0 do 10000 µg/m^3 [17].
- **PM10**: Prašni delci velikosti <10 µm. Zaloga vrednosti je od 0 do 10000 µg/m^3 [18].
- **Benzen**: Benzen C_6H_6. Zaloga vrednosti je od 0,1 do 380 µg/m^3 [19].

3.3 Merilna naprava

Jedro merilne naprave je dvojedrni mikrokrmilnik ESP32[1] proizvajalca Espressif Systems, ki omogoča WiFi in Bluetooth povezljivost. Prav tako vsebuje 34 GPIO, preko katerih lahko na krmilnik povežemo razne zunanje naprave. Mikrokrmilnik uporablja TSMC-jev čip ultra nizke porabe s 40 nm tehnologijo. Ta krmilnik smo izbrali zaradi nizke porabe energije, dobre povezljivosti in nizke cene [20].

Na mikrokrmilnik smo povezali senzor SEN55 proizvajalca Sensirion, predstavljen v spodnji sliki. Omogoča »all-in-one« rešitev za merjenje kvalitete zraka. Slednje pomeni, da je v en senzor združenih več funkcionalnosti za natančno merjenje raznih zunanjih parametrov. Senzor meri parametre zraka: PM1, PM2.5, PM4, PM10, temperaturo, relativno vlažnost, volatilne organske spojine in dušikove okside [21].



Slika 1 : Diagram povezave senzorja z mikrokrmilnikom [22]

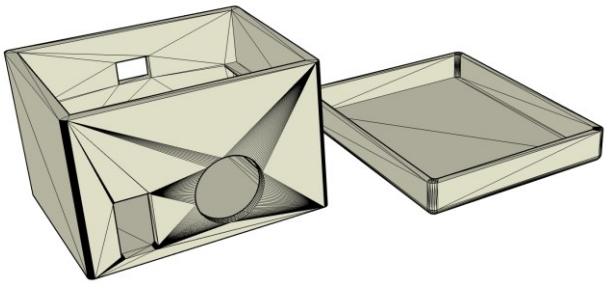
Kodo za mikrokrmilnik smo spisali v programskem jeziku C++ v razvojnem orodju Arduino IDE. Koda je sestavljena iz inicializacijskega bloka, kjer se vzpostavi povezava z WiFi omrežjem in senzorjem. Nato se požene ponavljajoči se del kode, ki na določen interval pošilja izmerjene podatke na spletni vmesnik. V našem primeru je bil interval nastavljen na 10 minut. Koda implementira tudi zaznavanje izpada povezave z WiFi omrežjem in ob izpadu samodejno poskuša vzpostaviti povezavo.

[1] ESP32 https://www.espressif.com/en/products/socs/esp32

```
// SETUP
void setup() {
  Serial.begin(115200);
  while (!Serial) {
    delay(100);
  }
  setUpWifi();
  setUpAirSensor();
}

// LOOP
void loop() {
  postAirQualityData();
  delay(repeatInterval);
}
```

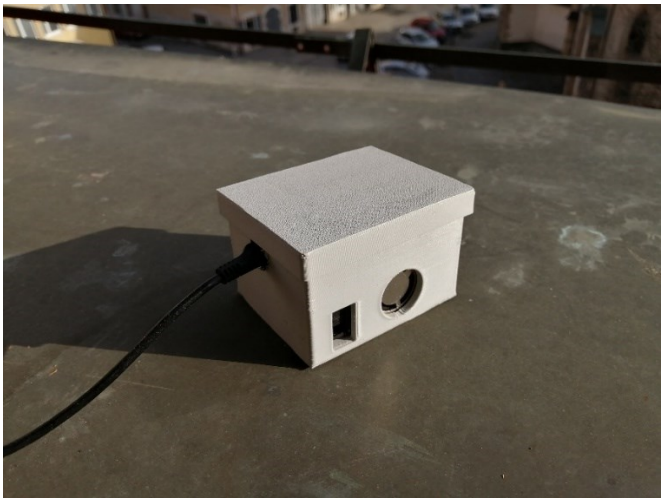
Slika 2 : Izsek glavnega dela kode

Ohišje za merilno postajo smo zmodelirali sami. Pri tem smo si pomagali s programom SketchUp, s katerim smo lahko 3D modelirali. Ohišje in pokrov sta bila narejena po merah senzorja, nad katerim sedi mikrokrmilnik z vhodom za napajanje. Pokrov in ohišje se tesno prilegata, slednje omogoča boljšo zaščito pred vodo. Ohišje in pokrov sta bila 3D tiskana z najlon filamentom, ki omogoča odlično trdnost in odpornost proti vročini [23].



Slika 3 : 3D model ohišja in pokrova

Pri modeliranju in izbiranju materiala smo bili še posebej pozorni na odpornost proti zunanjim vplivom. Ker je bila naprava dalj časa na prostem, je bilo to ključnega pomena tudi za dobro in kvalitetno zbiranje željenih podatkov.



Slika 4 : Sestavljena merilna postaja

3.4 Spletni vmesnik

Ker za ta projekt nismo potrebovali veliko funkcij in kompleksnosti, smo bili pri izbiri spletnega vmesnika pozorni na enostavno implementacijo. Tako smo se odločili za spletni vmesnik Node.js REST. Spletni vmesnik se povezuje z Sqlite podatkovno bazo, ki omogoča shranjevanje merjenih podatkov v trajnem stanju. Spletni vmesnik ima dve končni točki, in sicer:

- [GET] - /AirQuality,
- [POST] - /AirQuality.

Prva končna točka sprejema GET zahtevke in vrača celoten seznam meritev, razvrščenih od najnovejše do najstarejše.

Druga končna točka sprejema POST zahtevke. Njen namen, je da lahko merilna postaja pošilja izmerjene podatke na to točko. V tej končni točki se ob prejemu zahtevka izmerjenih podatkov iz lokalne merilne postaje naredi še en zahtevek na strežnik ARSO, s katerega pridobimo podatke o kvaliteti zraka iz zunanjih postaj. Vsi podatki se nato združijo in shranijo kot en vnos v našo podatkovno bazo.

```
{
    "id": 3560,
    "pm1": 2,
    "pm25": 2,
    "pm4": 2,
    "pm10": 2,
    "h": 10.92,
    "t": 27.54,
    "voc": 129,
    "nox": 1,
    "dateTime": "2023-04-04 13:42:45",
    "arso_data": [
        {
            "id": 7073,
            "merilno_mesto": "MB Titova",
            "datum_od": "2023-04-04 14:00",
            "datum_do": "2023-04-04 15:00",
            "so2": null,
            "co": null,
            "o3": null,
            "no2": 22,
            "pm10": 11,
            "pm25": null,
            "benzen": 0,
            "measurements_id": 3560
        },
        {
            "id": 7074,
            "merilno_mesto": "MB Vrbanski",
            "datum_od": "2023-04-04 14:00",
            "datum_do": "2023-04-04 15:00",
            "so2": null,
            "co": null,
            "o3": 93,
            "no2": 1,
            "pm10": 8,
            "pm25": null,
            "benzen": null,
            "measurements_id": 3560
        }
    ]
},
```


Slika 5 : Struktura shranjenih podatkov

3.5 Skripta za pridobivanje podatkov

V okviru pridobivanja podatkov smo kreirali tudi Python skripto, ki nam omogoča enostavno pridobivanje izmerjenih podatkov. Skripto smo naredili s Python knjižnico, imenovano Requests, preko katere smo klicali HTTP zahtevek na naš spletni strežnik, ki nam kot odgovor vrne JSON objekt. Ta odgovor si nato lokalno shranimo, da ne kličemo vsakič spletnega strežnika, ko želimo pognati skripto. Ta skripta vključuje tudi funkcionalnosti čiščenja in obdelave prejetih podatkov ter implementacijo algoritmov podatkovnega rudarjenja, ki bodo uporabljali te podatke. Te funkcionalnosti in njene podrobnosti bomo razložili v naslednjih poglavjih.

3.6 Podatkovno rudarjenje

3.6.1 Čiščenje in obdelava

Po pridobitvi podatkov smo spremenili samo strukturo podatkov, tako da smo jih preuredili v objekt globine 1. Meritve iz zunanjih postaj ARSO smo združili na nivo lastnih meritev. S tem smo nivoju lastnih meritev dodali podatke o3, no2, benzen, pm10 in pm25. Za podatke o3, no2 in benzen smo izračunali povprečje med dvema zunanjima postajama in jih dodali na nivo lastne meritve. Pri parametrih pm10 in pm25 so že bili obstoječi podatki za lastne meritve in podatki iz zunanjih meritev, ki smo jih združili. Zato smo najprej izračunali povprečje za pm10 in pm25 med zunanjima postajama, nato pa smo še izračunali povprečje med povprečjem zunanjih postaj in obstoječih vrednosti za lastno meritev.

Slika 6 : Končna struktura podatkov

V nadaljevanju smo v skripti prilagodili vrednost izmerjene temperature za –3 °C. Za to smo se odločili, ker je v specifikaciji za senzor zraka navedeno, da ob uporabi ohišja prihaja do odstopanja temperature. Do tega pride zaradi gretja senzorja v ohišju, kar privede do oddajanja temperature in posledično napačne meritve. Za vrednost –3 °C smo se odločili na podlagi primerjanja trenutne temperature v Mariboru, ki je bila prikazana na spletni strani vremenske napovedi ARSO, in trenutne izmerjene temperature. Tako je povprečno odstopanje med izmerjenimi temperaturami znašalo –3 °C. Pri vrednostih O_3, NO_2 in benzena smo iz zunanjih postaj občasno dobili prazne vrednosti. Pri čiščenju podatkov smo se odločili za pristop zamenjave praznih vrednosti

z vrednostmi 0. Na koncu skripte smo očiščene podatke shranili v Excel datoteko, kjer smo imeli lažji pregled nad pridobljenimi podatki.

3.6.2 Diskretizacija podatkov

Za uporabo klasičnih algoritmov za rudarjenje asociativnih pravil smo morali podatke pretvoriti iz numeričnih vrednosti v diskretne podatke.

Za diskretizacijo vrednosti PM1, PM2.5, PM4 in PM10 smo prevzeli podatke s strani ARSO, in sicer iz tabele indeksa kakovosti zraka. V tabeli je bila navedena samo diskretizacija za PM2.5. Za vrednosti PM1 smo uporabili kar diskretizacijo PM2.5, saj za vrednosti PM1 nismo našli ustrezne diskretizacije. Tako lahko predpostavljamo, da se diskretizirajo enako, saj se PM1 delci le malo razlikujejo od delcev PM2.5. Maksimalne vrednosti za PM1 in PM2.5 smo določili glede na zalogo vrednosti senzorja [24].

Tabela 1 : Diskretizacija PM1 in PM2.5

	Zelo dobra	Dobra	Sprejemljiva	Slaba	Zelo slaba	Izredno slaba
PM1	0–10	10–20	20–25	25–50	50–75	75–1000
PM2.5	0–10	10–20	20–25	25–50	50–75	75–1000

Podobno smo storili s prevzemanjem diskretizacije delcev PM10. Isto diskretizacijo smo uporabili tudi za delce PM4, saj zanje ni bilo navedenih podatkov in so delci podobne velikosti. Maksimalne vrednosti za PM4 in PM10 smo določili glede na zalogo vrednosti senzorja.

Tabela 2 : Diskretizacija PM4 in PM10

	Zelo dobra	Dobra	Sprejemljiva	Slaba	Zelo slaba	Izredno slaba
PM4	0–20	20–40	40–50	50–100	100–150	150–1000
PM10	0–20	20–40	40–50	50–100	100–150	150–1000

Pri temperaturi smo diskretizacijo določili po enakomernih intervalih glede na zalogo vrednosti senzorja pri merjenju temperature.

Tabela 3 : Diskretizacija temperature

	Zelo hladno	Hladno	Srednje	Toplo	Vroče	Zelo vroče
Temperatura	–15–0	0–10	10–20	20–30	30–40	40–50

Pri vrednosti vlage, VOC in NOX smo diskretizacijo določili po enakomernih intervalih glede na zalogo vrednosti senzorja pri merjenju temperature.

Tabela 4 : Diskretizacija vlage, VOC in NOX

	Zelo nizka	Nizka	Srednja	Visoka	Zelo visoka
Vlaga	0–20	20–40	40–60	60–80	80–100
VOC	0–100	100–200	200–300	300–400	400–500
NOX	0–100	100–200	200–300	300–400	400–500

Pri vrednostih časa smo zanemarili datum in uporabili za intervale samo vrednosti ure, ki smo jih diskretizirali po enakomernih 6-urnih intervalih.

Tabela 5 : Diskretizacija časa

	Noč	Dopoldne	Popoldne	Zvečer
Čas	0–6	6–12	12–18	18–24

Za vrednosti O_3 in NO_2 smo diskretizacijo prevzeli s strani ARSO, iz tabele indeksa kakovosti zraka [24].

Tabela 6 : Diskretizacija O_3 in NO_2

	Zelo dobra	Dobra	Sprejemljiva	Slaba	Zelo slaba	Izredno slaba
O3	0–50	50–100	100–130	130–240	240–380	380–800
NO2	0–40	40–90	90–120	120–230	230–340	340–1000

Vrednost benzena smo diskretizirali samo na dva dela, in sicer v razponu zaloge vrednosti senzorja. Edini podatek za diskretizacijo benzena smo pridobili glede na zakonodajno letno mejno vrednost 5 $\mu g/m^3$. Posledično smo določili, da je vse pod to mejo dobro in vse nad omenjeno mejo slabo.

Tabela 7 : Diskretizacija benzena

	Dobro	Slabo
Benzen	0–5	5–380

Zavedamo se, da smo z diskretizacijo vseh podatkov izgubili določen del podatkov oziroma delež natančnosti. Kljub temu pa nam diskretizacija omogoča lažjo interpretacijo podatkov in možnost uporabe klasičnih algoritmov rudarjenja asociativnih pravil.

3.6.3 Rudarjenje asociativnih pravil

Ko imamo pripravljene podatke, lahko nad njimi izvedemo podatkovno rudarjenje. Izbrali smo algoritme Apriori, Eclat, Fp-growth za klasično rudarjenje asociativnih pravil in algoritem optimizacije roja delcev za numerično rudarjenje asociativnih pravil. Same implementacije algoritmov smo poiskali na repozitoriju Python knjižnic PyPI [25]. Z uporabo obstoječih knjižnic smo si olajšali implementacijo, saj nam ni bilo treba pisati lastnih implementacij algoritmov. Opis posameznih algoritmov in njihovo implementacijo bomo predstavili v naslednjem poglavju.

4 RUDARJENJE ASOCIACIJSKIH PRAVIL

4.1 Algoritmi

4.1.1 Apriori

Odkrila sta ga R. Agrawal in R. Srikant leta 1994 za namen iskanja pogostih množic. Apriori se imenuje ravno zato, ker uporablja predhodno znanje (ang. *prior knowledge*) pogostih množic [26]. Sodi med popularnejše algoritme rudarjenja asociativnih pravil. Razlog za popularnost je v zelo preprostem razumevanju. Je eden temeljnih algoritmov za analizo nakupovalne košarice (ang. *basket analysis*). Analiza nakupovalne košarice je študija strank, ki nam pove, katere izdelke stranke kupujejo v trgovinah. Iz analize izvemo, katere kombinacije produktov so kupljene skupaj. Takšne kombinacije produktov imenujemo pogoste kombinacije izdelkov (ang. *frequent itemsets*).

Apriori algoritem se ne uporablja samo za analizo nakupovalne košarice, ampak se lahko v teoriji uporabi za katerekoli pogoste kombinacije predmetov.

Vprašamo se lahko, zakaj ne samo iteriramo po naših podatkih in preštejemo najpogosteje pojavljajoče se predmete. Takšen iterativni pristop je precej neučinkovit in bi pri večji količini podatkov trajal zelo dolgo. Zato imamo na voljo algoritem Apriori [27].

Psevdokoda 1 : Apriori

```
Vhod: D = podatkovna zbirka
      minPodpora = minimalna podpora
Izhod: L = množica pogostih množic postavk

procedure Apriori(D, minPodpora)
begin
  L1 = iskanje_pogostih_1_množic(D, minSupport)
  L = L1
  k = 2

  while L is not empty do
    Ck = generiranje_kandidatov_pogostih_množic(L)
    Lk = iskanje_pogostih_k_množic(Ck, D, minSupport)
    if Lk is not empty then
      L = L ∪ Lk
      k = k + 1
    else
```

```
        break
      end if
    end while

    return L
end procedure
```

Koraki algoritma:

1. Iz baze transakcij D poiščemo vse pogoste množice postavk L z velikostjo 1, ki tudi zadostujejo minimalni podpori.
2. Prvi korak znotraj iteracije je, da za L generiramo nove kandidate pogostih množic, ki jih shranimo v Ck.
3. Nato iz množice novih kandidatov Ck in prvotne množice D pridobimo nove pogoste množice, ki zadostujejo minimalni podpori in jih shranimo v Lk.
4. Če Lk ni prazen, združimo obstoječo množico L in množico novih pogostih množic Lk.
5. Algoritem ponavljamo, dokler množica L ni prazna.
6. Na koncu vrnemo rezultat L, ki vsebuje množico vseh pogostih množic postavk.

4.1.2 Eclat

Eclat algoritem je okrajšava za »Equivalence Class Clustering and Bottom-Up Lattice Traversal«. Je algoritem za rudarjenje asociativnih pravil, ki vključuje odkrivanje pogostih sklopov predmetov [28]. Je eden izmed popularnejših algoritmov. Obenem je učinkovitejša in bolj skalabilna različica Apriori algoritma. Apriori algoritem deluje v horizontalnem smislu in imitira »Breadth-First« iskanje po grafu. Eclat algoritem pa deluje v vertikalne smislu in deluje tako kot »Depth-First« iskanje po grafu. Horizontalni pristop algoritmu Eclat napram Apriori algoritmu omogoča prednost v hitrosti. Eclat porabi tudi manj delovnega pomnilnika kot Apriori, ravno zaradi pristopa »Depth-First« iskanja. Hkrati porabi manj računske moči, saj ni treba ponovno preverjati individualnih podpornih vrednosti [29].

Večina algoritmov za rudarjenje asociativnih pravil ima več ključnih metrik za asociativna pravila. Algoritem Eclat jih nima toliko. Vsebuje samo vrednost podpore. A v zameno za to, da imamo manj metrik, pridobimo na hitrosti. V naši implementaciji smo sicer uspeli izluščiti tudi ostale metrike, kot sta zaupanje in dvig [28].

Psevdokoda 2 : Eclat

```
Vhod: D = podatkovna zbirka
      minPodpora = minimalna podpora
Izhod: L = množica pogostih množic postavk

procedure Eclat(D, minPodpora)
begin
  for each A in D1 do
    T = Ø
    for each B in D1 do
      if B > A then
        R = A U B
        tidset(R) = tidset(A) ∩ tidset(B)
        if tidset(R) > minPodpora then
          T = T ∪ {R}
          L = L ∪ {R}
        end if
    end for
    if T is not empty then
      L = L ∪ Eclat(T, minPodpora)
    end if
  end for
  return L
end procedure
```

Koraki psevdokode:

1. Gnezdeno iteriramo čez množico D in preverimo, ali je notranja iteracija večja od zunanje.
2. Nato v spremenljivko R shranimo rezultat unije A in B. Izračunamo tudi vrednost transakcijske identifikacijske množice, preko katere lahko nato preverjamo minimalno podporo.
3. Če vrednost minimalne podpore transakcijske identifikacijske množice zadostuje pogoju, shranimo v T rezultat unije množice T in prej izračunanega R. Prav tako shranimo v L rezultat unije množice L in prej izračunanega R.
4. Če spremenljivka T ni prazna, nato rekurzivno kličemo funkcijo Eclat, ki prejme argument novonastale množice transakcij in minimalno podporo. Rezultat rekurzivno klicane funkcije Eclat shranimo v množico pogostih postavk L. Rekurzivno izvajanje se konča, ko funkcija prejme prazno bazo transakcij.
5. Kot rezultat vrne množico vseh pogostih množic postavk.

4.1.3 Fp-growth

Algoritem Fp-growth ali »Frequent Pattern Growth« je učinkovita in skalabilna metoda za rudarjenje pogostih vzorcev s pomočjo vzorčne fragmentne rasti, ki uporablja razširjeno drevesno strukturo. Takšno drevo shranjuje stisnjene in ključne informacije o pogostih vzorcih in ga imenuje Fp-drevo (ang. *Fp-tree*).

Algoritem Fp-growth je alternativna možnost iskanja pogostih množic brez generiranja kandidatov, kar omogoči hitrejše izvajanje. Uporablja strategijo »deli in vladaj«. Jedro metode je uporaba posebne podatkovne strukture FP-drevo [30].

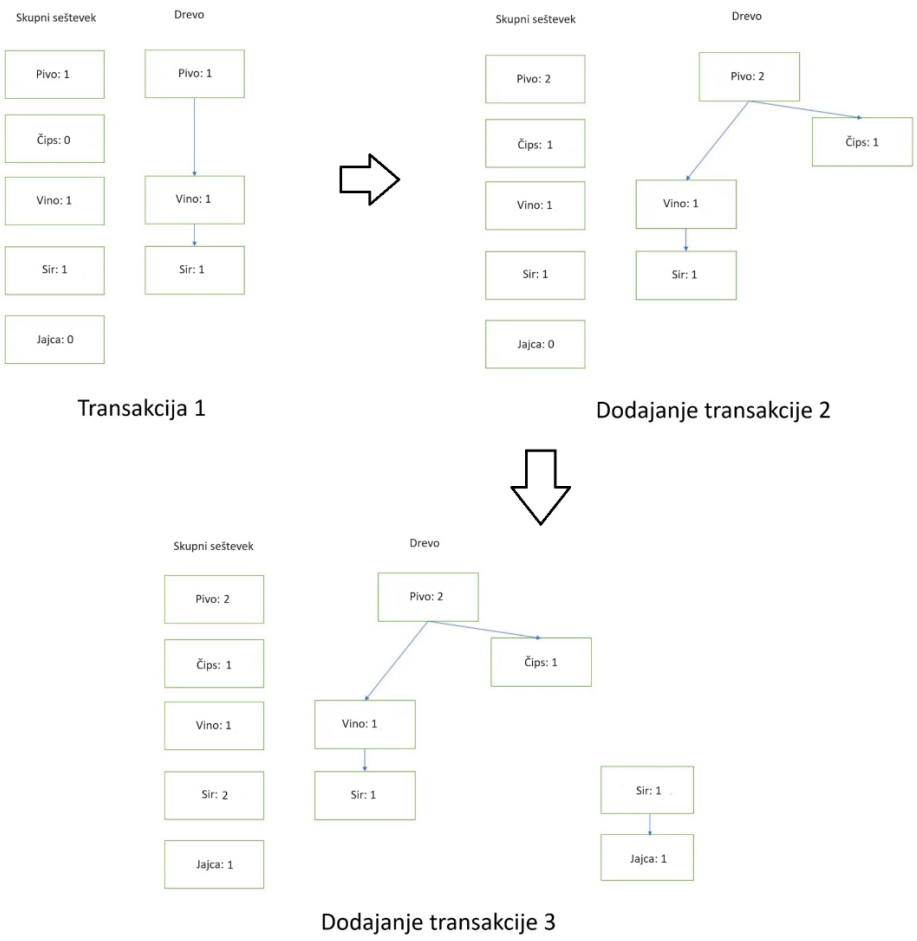
Koraki algoritma :

1. Štetje pojavitev predmetov. Prvi korak algoritma je prešteti pojavitev posameznih predmetov, katerih vrednosti bomo nato pretvorili v vrednost podpore.

2. Filtriranje redkih predmetov z minimalno podporo. Določiti moramo vrednost minimalne podpore, saj bo ta določila, kateri predmeti se dovolj pogosto pojavijo, da jih želimo obdržati.
3. Sortiranje množic glede na pojavitve. Za preostale predmete bomo oblikovali urejeno tabelo, ki bo urejena tako, da bodo množice z največ predmeti na vrhu in množice z najmanj predmeti na koncu.
4. Kreiranje drevesa in dodajanje posameznih transakcij. Za primer grajenja drevesa bomo uporabili primer nakupovalne košarice s primerom transakcij.

Tabela 8 : Primer transakcij

Transakcijska št.	Izdelki	Naročeni izdelki
1	pivo, vino, sir	pivo, vino, sir
2	pivo, čips	pivo, čips
3	jajca, moka, maslo, sir	sir, jajca



Slika 7 : Primer gradnje FP-drevesa

Začnemo s korenom, ki je privzeto določeno s prazno vrednostjo (*null*). Nato v drevo dodamo prvo transakcijo, ki je v tem primeru {pivo, vino, sir}, ki predstavlja vertikalno vejo. Ko dodamo drugo transakcijo {pivo, čips}, imata transakcija 1 in 2 skupni predmet čips. Zato lahko že obstoječ element {pivo} v drevesu uporabimo za drugo transakcijo, ki se nato razveja v element čips. Nato dodamo še tretjo transakcijo {sir, jajca}, ki nima nič skupnega z že obstoječim drevesom, zato vejo s transakcijo zapišemo posebej [31].

4.2 Numerično rudarjenje asociativnih pravil

4.2.1 Optimizacija roja delcev

Optimizacija roja delcev ali PSO (ang. *particle swarm optimization*) je algoritem, ki spada med algoritme inteligence rojev (ang. *swarm intelligence based algorithm*). Ta skupina algoritmov se zgleduje po obnašanju gruč insektov, čred živali, jat rib … Algoritmi inteligence rojev lahko spremenijo smer iskanja skozi sodelovanje in deljenje informacij med posamezniki. Imajo hitrejšo mero konvergence in lahko pridobivajo boljše rezultate kot evolucijski algoritmi, saj slabo delovanje posameznikov ne vpliva na rezultat celotnega roja.

V modelu algoritma optimizacije roja delcev si lahko predstavljamo posameznika kot delec in npr. čredo živali kot gručo delcev. Numerično rudarjenje asociativnih pravil poteka tako, da iz podatkovne množice najprej generiramo pravila, nad katerimi nato izvedemo algoritem, ki bo s pomočjo simulacije roja delcev poiskal najboljša pravila glede na želene metrike. Delci se prosto gibljejo po D-dimenzijskem prostoru in vsak delec ima svojo pozicijo, hitrost in vztrajnost. Glede na pozicijo v D-dimenzijskem prostoru v korelaciji z iskalnim prostorom lahko izračunamo vrednost primernosti, ki določa, kako primerno je določeno pravilo. Tako lahko gruča delcev iz nabora pravil izbere najboljše [32].

4.3 Implementacija

4.3.1 Obdelava podatkov

Ustvarili smo začetno Python skripto, v kateri so koraki za pridobivanje in obdelovanje podatkov. V tej skripti nadalje prožimo tudi posamezne algoritme. V nadaljevanju bomo razložili implementacijo iz Izseka kode 1.

Izsek kode 1 : Implementacija obdelave podatkov

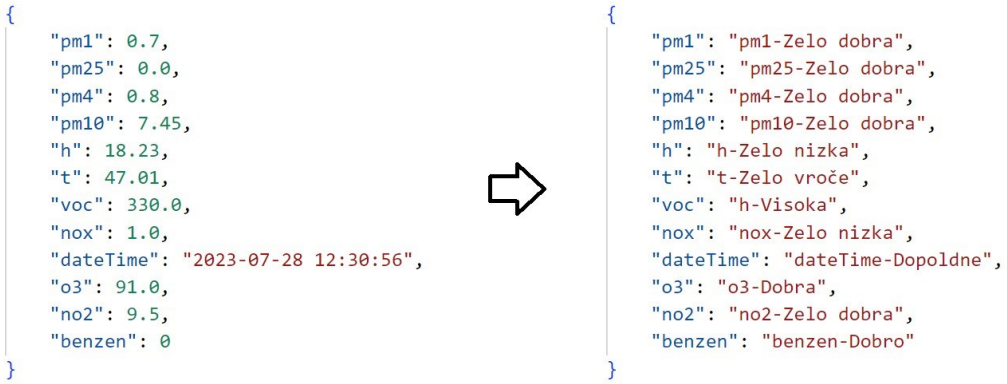
```
# Pridobivanje podatkov
ds = data_service(load_fresh_data)

# Strukturiranje podatkov v pandas dataframe
ds.structure_data()
# Popravek meritev temperature
ds.calibrate_temperature()

# Čiščenje podatkov
ds.clean_data_zero()

# Preverjanje ali obstajajo prazne vrednosti
ds.check_for_na_values()

# Odstranjevanje stolpcev, ki niso potrebni
ds.data = ds.data.drop('id', axis=1)
# Shranjevanje podatkov v CSV
ds.save_clean_data_to_CSV()

if(categorized_alg):
    # Diskretizacija podatkov
    ds.discretize_data()
```

1. Objekt **data_service** nam omogoča, da pridobimo podatkovno množico iz spletnega strežnika ali iz lokalno shranjene CSV datoteke. Preko tega objekta prožimo vse nadaljnje funkcije, ki bodo obdelale naloženo podatkovno množico.
2. Funkcija **structure_data** bo naložene podatke prestrukturirala in združila podatke ARSO ter naše lastne meritve v posamezne vrstice. Takšne podatke lahko nato pretvorimo v Pandas Dataframe.

Slika 8 : Pretvorba podatkovne strukture

3. Funkcija **calibrate_temperature** popravi vse temperaturne vrednosti za –3 stopinje, saj segrevanje senzorja v ohišju pripomore k višjim izmerjenim vrednostim.
4. Nato imamo funkcijo **clean_data_zero**, ki nadomesti manjkajoče vrednosti z ničlami.
5. Funkcija **check_for_na_values** še zadnjič preverja, ali imamo še kakšne izpuščene manjkajoče vrednosti.
6. Z **drop** funkcijo nad našo podatkovno množico izvedemo odstranitev vseh »id« vrednosti, ki niso potrebne in bi nam oteževale nadaljnje delo.
7. Funkcija **save_clean_data_to_CSV** nam obdelano podatkovno množico shrani v CSV format, ki nam omogoča pogled obdelane množice v Excelu.

8. Če želimo izvajati rudarjenje asociativnih pravil, kot so Apriori, Eclat ali Fp-growth, moramo podatkovno množico še pred izvedbo teh algoritmov pretvoriti s funkcijo **discretize_data** iz numeričnih podatkov v diskretizirane podatke.



Slika 9 : Pretvorba iz numeričnih podatkov v kategorične

4.3.2 One-hot encoding

One-hot encoding je tehnika, s katero lahko predstavimo kategorične podatke z numeričnimi vrednostmi. Omogoča izboljšanje modela, tako da pridobi več informacij za kategorično spremenljivko. Obenem lahko prepreči tudi težave določanja zaporedja pri kategoričnih podatkih, kjer ima spremenljivka naravni vrstni red (npr. majhno, srednje, veliko).

Deluje tako, da vse možne kategorične spremenljivke nastavimo kot ime stolpca. Vsaka vrstica pa predstavlja transakcijo z vrednostmi 0 in 1. Kot primer lahko vzamemo spodaj navedene štiri meritve, ki so zapisane vsaka kot svoja transakcija s številko »id« meritve.

Tabela 9 : Primer transakcij meritev

Id meritve	**Meritev**
1	Pm1-dobro
2	No2-zelo dobro
3	Benzen-slabo
4	dateTime-noč

Za pretvorbo v one-hot encoding vse možne vrednosti vstavimo v prvo vrstico kot imena stolpcev. Vsako nadaljnjo vrstico označimo z 1, če se je ta vrednost meritve pojavila v tej transakciji. Z 0 pa označimo podatek, če se vrednost v transakciji ni pojavila [33].

Tabela 10 : Primer one-hot encoding tabele

Id meritve	**Pm1-dobro**	**dateTime-noč**	**Benzen-slabo**	**No2-Zelo dobro**
1	1	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	1	0	0

4.3.3 Algoritem Apriori

Algoritem Apriori smo implementirali s pomočjo knjižnice Mlxtend. Knjižnica nam omogoča preprocesiranje podatkov, iskanje pogostih množic in rudarjenje asociativnih pravil. Parametri za našo funkcijo so **min_support** (najmanjša dovoljena podpora) in **min_lift** (najmanjši dovoljeni dvig). V nadaljevanju bomo razložili implementacijo iz Izseka kode 2.

Izsek kode 2 : Implementacija Apriori algoritma

```
def apriori(
    self,
    min_support:float,
    min_lift:float
):
    # Encode data
    encoded_data = self.get_onehot_encoded_data()
    # Run apriori
    frequent_itemset = apriori(encoded_data,
                               min_support = min_support,
                               use_colnames = True)
    frequent_itemset = self.sort_and_clean_frequent_itemset(frequent_itemset)
    # Get association rules
    arules = association_rules(frequent_itemset,
                               metric ="lift",
                               min_threshold = min_lift)
    arules = self.sort_and_clean_arules(arules)
```

1. Na začetku smo našo podatkovno množico pretvorili na način one-hot encoding s pomočjo funkcije **get_onehot_encoded_data**. Takšno kodiranje je potrebno za uporabo v funkciji Apriori.
2. Poženemo funkcijo **Apriori**, ki prejme one-hot kodirane podatke in najmanjšo dovoljeno vrednost podpore preko parametra **min_support**. Kot rezultat pridobimo pogoste množice podatkov, ki so sestavljene iz stolpca podpore in parov predmetov. Te pogoste množice nato sortiramo in očistimo, da jih lahko kasneje shranimo kot CSV-dokument.

Tabela 11 : Rezultati pogostih množic parov

Support	**Itemsets**
1	{'pm25-Zelo dobra'}
1	{'pm25-Zelo dobra','nox-Zelo nizka'}
1	{'nox-Zelo nizka'}
0,990999	{'no2-Zelo dobra', 'pm25-Zelo dobra', 'nox-Zelo nizka'}
0,990999	{'no2-Zelo dobra', 'pm25-Zelo dobra'}
0,990999	{'no2-Zelo dobra','nox-Zelo nizka'}

3. Nato lahko poženemo funkcijo **association_rules**, ki prejme objekt pogostih množic, nad katerim bo izvedel rudarjenje. Za metriko pri funkciji smo se odločili za dvig, katerega minimalno dovoljeno vrednost določimo s spremenljivko **min_lift**. Kot rezultat prejmemo Pandas Dataframe s podatki o asociacijskih pravilih, ki vsebujejo podatke, kot so predpostavka, posledica, podpora predpostavke, podpora posledice, podpora, zaupanje, dvig, vzvod, prepričanje in zhang metrika. Pridobljene podatke o asociacijskih pravilih nato sortiramo in očistimo, da jih lahko kasneje shranimo kot CSV.

Tabela 12 : Rezultat asociativnih pravil

Antecedents	Consequents	Antecedent support	Consequent support	Support	Confidence	Lift	Leverage	Conviction	Zhang metric
{'pm25-Zelo dobra'}	{'nox-Zelo nizka'}	1	1	1	1	1	0	Inf	0
{'nox-Zelo nizka'}	{'pm25-Zelo dobra'}	1	1	1	1	1	0	Inf	0

{'no2-Zelo dobra'}	{'nox-Zelo nizka'}	0.990998594	1	0.990999	1	1	0	Inf	0
{'no2-Zelo dobra'}	{'pm25-Zelo dobra'}	0.990998594	1	0.990999	1	1	0	inf	0

4.3.4 Algoritem Eclat

Algoritem Eclat smo implementirali s pomočjo knjižnice pyECLAT. Ta knjižnica nam omogoča iskanje pogostih množic, ki je pogoj za rudarjenje asociativnih pravil. Parametri za našo funkcijo so **min_support** (najmanjša dovoljena podpora), **min_combination** (najmanjše dovoljeno št. kombinacij), **max_combination** (največje dovoljeno št. kombinacij) in **min_lift** (najmanjši dovoljeni dvig). Za rudarjenje asociativnih pravil smo si pomagali s knjižnico Mlxtend, saj knjižnica PyECLAT tega ne omogoča. V nadaljevanju bomo razložili implementacijo iz Izseka kode 3.

Izsek kode 3 : Implementacija Eclat algoritma

```
   def Eclat(
    self,
    min_support: float = 0.08,
    min_combination: float = 1,
    max_combination: float = 3,
    min_lift:float = 1
):
    # Encode data
    encoded_data = self.Eclat_encode_data()
    # Run Eclat
    Eclat_instance = ECLAT(encoded_data, verbose=False)
    get_ECLAT_indexes, get_ECLAT_supports = Eclat_instance.fit(
                                                min_support=min_support,
                                                min_combnation=min_combination,
                                                max_combination=max_combination,
                                                separator=' & ',
                                                verbose=False)
    # Encode for use in association rules
    frequent_itemset = self.Eclat_supports_to_df(_ECLAT_supports)
    frequent_itemset = frequent_itemset.sort_values(by=['support','itemsets'],
                                            ascending=False)

    # Get association rules
    arules = association_rules(frequent_itemset,
                               metric = "lift",
                               min_threshold = min_lift)
```

```
arules = self.sort_and_clean_arules(arules)
```

1. Na začetku smo podatkovno množico pretvorili v format, ki ga potrebuje Eclat objekt. To smo naredili tako, da smo imena stolpcev zamenjali s številkami od 0 do 11.

	0	1	2	3	...	8	9	10	11
0	pm1-Zelo dobra	pm25-Zelo dobra	pm4-Zelo dobra	pm10-Zelo dobra	...	dateTime-Dopoldne	o3-Dobra	no2-Zelo dobra	benzen-Dobro
1	pm1-Zelo dobra	pm25-Zelo dobra	pm4-Zelo dobra	pm10-Zelo dobra	...	dateTime-Dopoldne	o3-Dobra	no2-Zelo dobra	benzen-Dobro
2	pm1-Zelo dobra	pm25-Zelo dobra	pm4-Zelo dobra	pm10-Zelo dobra	...	dateTime-Dopoldne	o3-Dobra	no2-Zelo dobra	benzen-Dobro
3	pm1-Zelo dobra	pm25-Zelo dobra	pm4-Zelo dobra	pm10-Zelo dobra	...	dateTime-Dopoldne	o3-Dobra	no2-Zelo dobra	benzen-Dobro
4	pm1-Zelo dobra	pm25-Zelo dobra	pm4-Zelo dobra	pm10-Zelo dobra	...	dateTime-Dopoldne	o3-Dobra	no2-Zelo dobra	benzen-Dobro

Slika 10 : Pretvorjena podatkovna množica za ECLAT

2. Naložili smo pretvorjeno podatkovno množico v **ECLAT** objekt, preko katerega smo lahko pognali metodo **fit**, ki izračuna pogoste množice in transakcijske id množice. Metoda **fit** kot parametre prejme najmanjšo dovoljeno podporo, najmanjše in največje dovoljeno št. kombinacij, separator in možnost izpisa poteka v konzolo.
3. Naslednji korak je sortiranje in pretvorba rezultatov **fit** metode v format, ki ga sprejme funkcija za rudarjenje asociativnih pravil.
4. Funkcija **association_rules** kot parameter prejme prej izračunane pogoste množice in najmanjšo dovoljeno vrednost za dvig. Prejmemo enak rezultat kot pri algoritmu Apriori v enakem formatu. Rezultat kasneje shranimo v CSV datoteko.

4.3.5 Algoritem Fp-growth

Algoritem Fp-growth smo implementirali s pomočjo knjižnice Mlxtend. Ta knjižnica nam omogoča preprocesiranje podatkov, iskanje pogostih množic in rudarjenje asociativnih pravil. Parametri za našo funkcijo so **min_support** (najmanjša dovoljena podpora) in **min_lift** (najmanjši dovoljeni dvig). V nadaljevanju bomo razložili implementacijo iz Izseka kode 4.

Izsek kode 4 : Implementacija Fp-growth algoritma

```
def fp_growth(
    self,
    min_support: float = 0.08,
    min_lift: float = 1
):
    # Encode data
    encoded_data = self.get_onehot_encoded_data()

    # Run Fp-growth
    frequent_itemset = fpgrowth(encoded_data,
                                min_support=min_support,
                                use_colnames = True)
    frequent_itemset = self.sort_and_clean_frequent_itemset(frequent_itemset)

    # Get association rules
    arules = association_rules(frequent_itemset,
                               metric="lift",
                               min_threshold=min_lift)
    arules = self.sort_and_clean_arules(arules)
```

1. Na začetku smo našo podatkovno množico pretvorili z one-hot encoding preko funkcije **get_onehot_encoded_data**.
2. Poženemo funkcijo **fpgrowth**, ki prejme one-hot kodirane podatke, vrednost minimalne podpore in najmanjšo dovoljeno vrednost podpore preko spremenljivke **min_support**. Kot rezultat pridobimo pogoste množice podatkov, ki so sestavljene iz stolpca podpore in parov predmetov. Te pogoste množice nato še sortiramo in očistimo, da jih lahko kasneje shranimo kot CSV-dokument.
3. Funkcija **association_rules** prejme kot parameter prej izračunane pogoste množice in najmanjšo dovoljeno vrednost za dvig. Prejmemo enak rezultat kot pri algoritmu Apriori v enakem formatu. Rezultat kasneje shranimo v CSV datoteko.

4.3.6 Algoritem optimizacije roja delcev

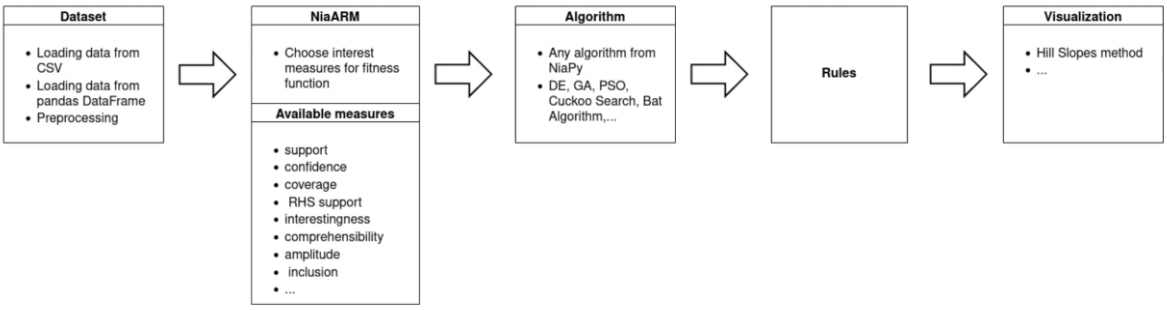
Algoritem optimizacije roja delcev smo implementirali s pomočjo knjižnice NiaARM. Sama knjižnica NiaARM vsebuje tudi knjižnico NiaPy, preko katere lahko implementiramo algoritme po vzorih iz narave [8]. Parametri za našo funkcijo so **min_support** (najmanjša dovoljena podpora) in **min_lift** (najmanjši dovoljeni dvig). V nadaljevanju bomo razložili implementacijo iz Izseka kode 5.

Izsek kode 5 : Implementacija algoritma optimizacije roja delcev

```
def particle_swarm_optimization(
    self,
    min_support: float = 0.5,
    min_lift: float = 1
):
    # Load data
    dataset = Dataset(self.data)

    # Set algorithm and metrics
    metrics = ('support', 'confidence')
    algorithm = ParticleSwarmAlgorithm(population_size=25,
                                       c1=2.0,
                                       c2=2.0,
                                        w=0.7,
                                        min_velocity=-1.5,
                                        max_velocity=1.5)
    # Get association rules
    rules, run_time = get_rules(dataset, algorithm, metrics,
                                max_iters=50, logging=True)

    # Filter rules with min_support and min_lift
    rules = self.filter_min_threshold(rules, min_support, min_lift)
```

1. Na začetku naše podatke iz formata Pandas DataFrame naložimo v objekt **Dataset**, ki obdela našo podatkovno množico za nadaljnjo uporabo.
2. V spremenljivko shranimo vrednosti metrik, ki nas zanimajo. V našem primeru sta to podpora in zaupanje (ang. *support*, *confidence*). V knjižnici jih je na voljo še veliko več.
3. Inicializiramo **ParticleSwarmAlgorithm** in mu določimo inicializacijske vrednosti.

4. Kličemo funkcijo **get_rules**, ki pridobi našo obdelano podatkovno množico, izbran algoritem in želeno število iteracij. Funkcija kot rezultat vrne objekt vseh izračunanih pravil.
5. Nato nad pravili izvedemo še funkcijo **filter_min_threshold**, ki iz dobljenih pravil izloči vsa, ki ne zadostujejo najmanjšim dovoljenim vrednostim za podporo in dvig, ki smo jih nastavili na začetku.



Slika 11 : Potek implementacije NiaARM [8]

5 EKSPERIMENTI IN REZULTATI

Namen eksperimenta primerjave algoritmov je prikazati njihovo učinkovitost ter kako se obnesejo nad našo podatkovno množico. Prav tako želimo preveriti vzporednico med teoretično platjo prednosti in slabosti algoritmov ter našimi lastnimi ugotovitvami. Za naš eksperiment smo uporabili strojno opremo s procesorjem Intel i5-12500H in 16 GB rama. Skripto smo pognali s Python različico 3.8 v urejevalniku Visual Studio Code. Vsakemu izvedenemu algoritmu smo nastavili najmanjšo dovoljeno podporo na 0,1 in najmanjši dovoljeni dvig na 0,1.

Po izvedenem eksperimentu smo tudi predstavili rezultate in ugotovitve za kategorične (ARM) kot tudi numerične (NARM) podatke. Podatke smo vizualizirali s pomočjo raznih grafov za lažje razumevanje, jih razložili ter predstavili novo ugotovljene povezave.

5.1 Primerjava algoritmov za kategorične podatke

Poleg skritih podatkov rudarjenja asociativnih pravil nas zanima tudi primerjava različnih algoritmov. Pri izvajanju algoritmov nad našo množico nas zanima, kako hitro delujejo in koliko pomnilnika porabijo. Najprej bomo predstavili teoretične prednosti in slabosti za vsak algoritem. Nato bomo predstavili še pogoje, pod katerimi smo izvedli primerjavo in izmerjene rezultate. Na koncu bomo predstavili še ugotovitve primerjave.

5.1.1 Apriori

Prednosti:

- enostaven za razumevanje;
- zmore obdelavo večjih podatkovnih množic, iz katerih lahko generiramo asociacijska pravila.

Slabosti:

- računsko zahteven za večje podatkovne množice zaradi generiranja kandidatov množic;

- generira veliko množic kandidatov, med katerimi ni nujno, da so ti pogosti, kar upočasnjuje delovanje;
- lahko je neučinkovit za podatkovne množice, kjer je vrednost najmanjše dovoljene podpore nizka.

5.1.2 ECLAT

Prednosti:

- učinkovitejši kot algoritem Apriori, saj uporablja vertikalni pristop;
- bolje se obnese za manjše podatkovne množice kot algoritem Apriori;
- generira manj množic kandidatov, kar pospeši delovanje.

Slabosti:

- njegovo delovanje pri množicah z nizko vrednostjo minimalne dovoljene podpore ni tako učinkovito;
- ne generira vseh pogostih množic, ampak samo tiste s specifičnim objektom.

5.1.3 Fp-growth

Prednosti:

- učinkovit in skalabilen, še posebej za večje podatkovne množice;
- generira kompaktno podatkovno strukturo, ki jo lahko uporabimo za učinkovito rudarjenje pogostih množic;
- dobro se obnese pri podatkovnih množicah z nizko vrednostjo najmanjše dovoljene podpore.

Slabosti:

- lahko porabi veliko pomnilnika za večje podatkovne množice;
- za generiranje drevesne strukture potrebuje več prehodov skozi celotno podatkovno množico, kar lahko porabi veliko časa;
- ni primerno za podatkovne množice z velikim številom unikatnih vrednosti [34].

5.1.4 Rezultati primerjave

Da smo lahko primerjali vse tri algoritme, smo za vse tri uporabili isto množico podatkov s 17.774 vrsticami in 11 stolpci, ki je velika 5,13 MB. Za vsak algoritem smo določili tudi najmanjšo dovoljeno podporo vrednosti 0,01 in najmanjši dovoljeni dvig vrednosti 0,01. Naša množica se lahko opredeli kot majhna, zato lahko predvidevamo, da bo za naš primer najbolj ustrezal algoritem Eclat.

Tabela 13 : Rezultati performans algoritmov

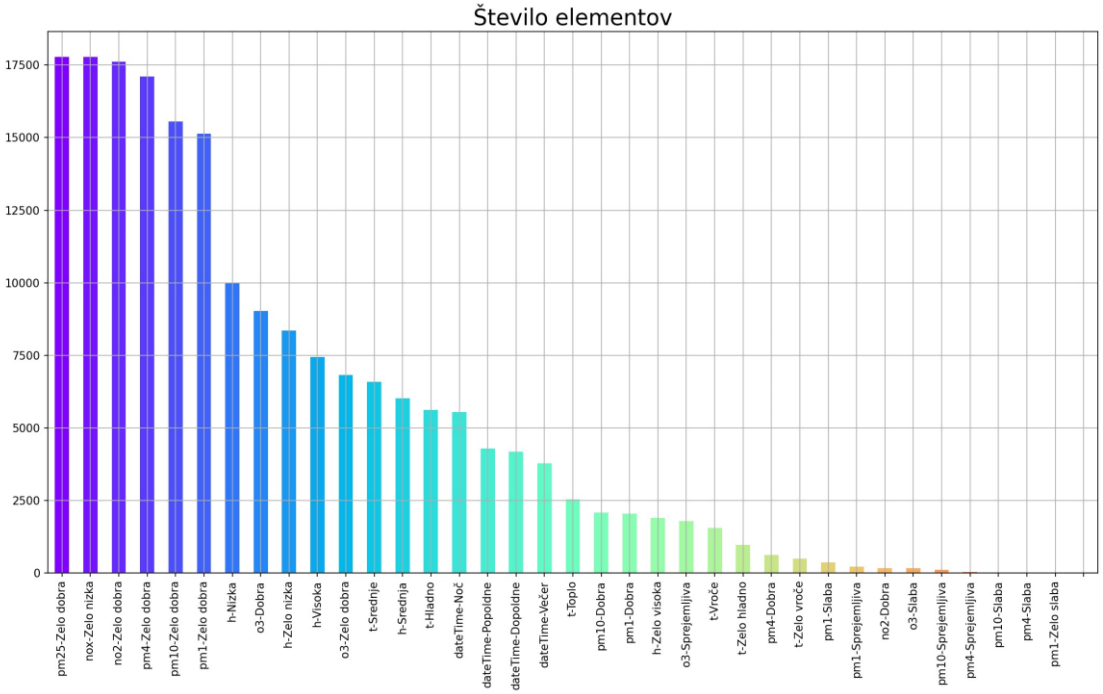
	Trajanje izvajanja (sekunde)	Porabljen pomnilnik (MB)
Apriori	140	6900
Eclat	55	410
Fp-growth	40	4300

5.1.5 Ugotovitve primerjave

Iz eksperimenta je razvidno, da je najpočasnejši izmed algoritmov Apriori, ki potrebuje največ časa za izvajanje in porabi tudi največ pomnilnika. Algoritma Eclat in Fp-growth sta občutno hitrejša – za skoraj trikrat. Drug najhitrejši algoritem je Eclat, ki v primerjavi z ostalima dvema porabi izredno malo pomnilnika. Najhitrejši algoritem pa je Fp-growth, ki porabi sicer občutno več pomnilnika kot Eclat, a je zato približno 30 % hitrejši. Da bi izbrali najboljši algoritem za našo množico, bi se morali odločati med algoritmoma Eclat in Fp-growth. Odvisno bi bilo od tega, kaj nam je pomembnejše, hitrost ali učinkovitost uporabe pomnilnika.

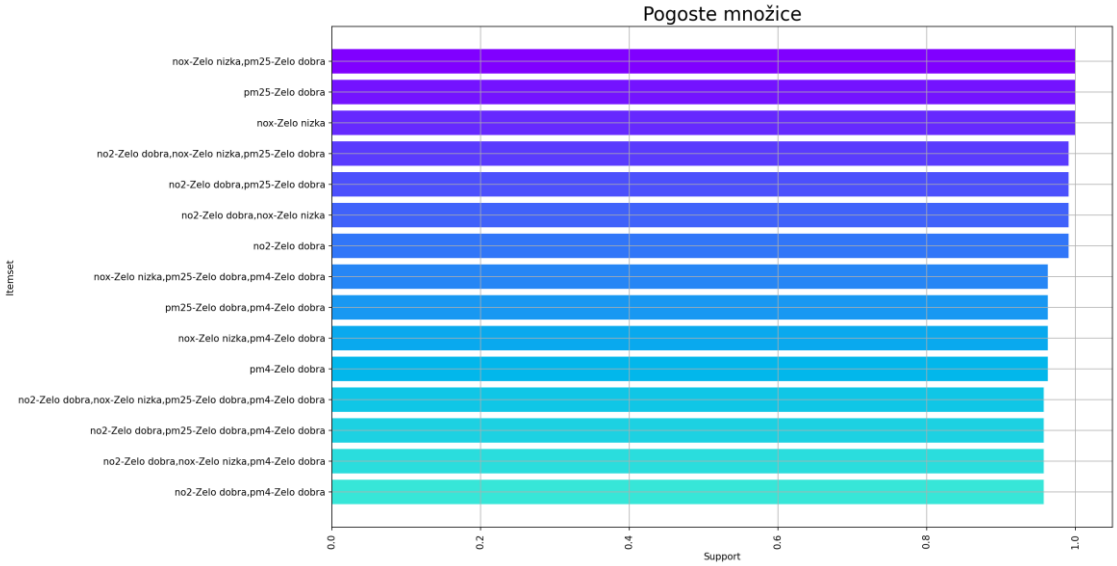
5.2 Rezultati za kategorične podatke

Rezultati rudarjenja asociativnih pravil iz kategoričnih podatkov so enaki pri vseh treh algoritmih. Edini razliki med njimi sta hitrost in količina pomnilnika, ki jo porabijo. Za boljšo predstavo podatkov smo oblikovali stolpčni graf posameznih pojavljajočih se vrednosti in število njihovih pojavitev. Iz grafa je razvidno, da se vrednosti, kot so »pm25-Zelo dobra«, »nox-Zelo nizka«, »no2-Zelo dobra« in »pm4-Zelo dobra«, zelo pogosto pojavljajo. Najmanjkrat pa se ponavljajo »pm1-Zelo slaba«, »pm4-Slaba«, »pm4-Sprejemljiva« in »pm10-Sprejemljiva«.



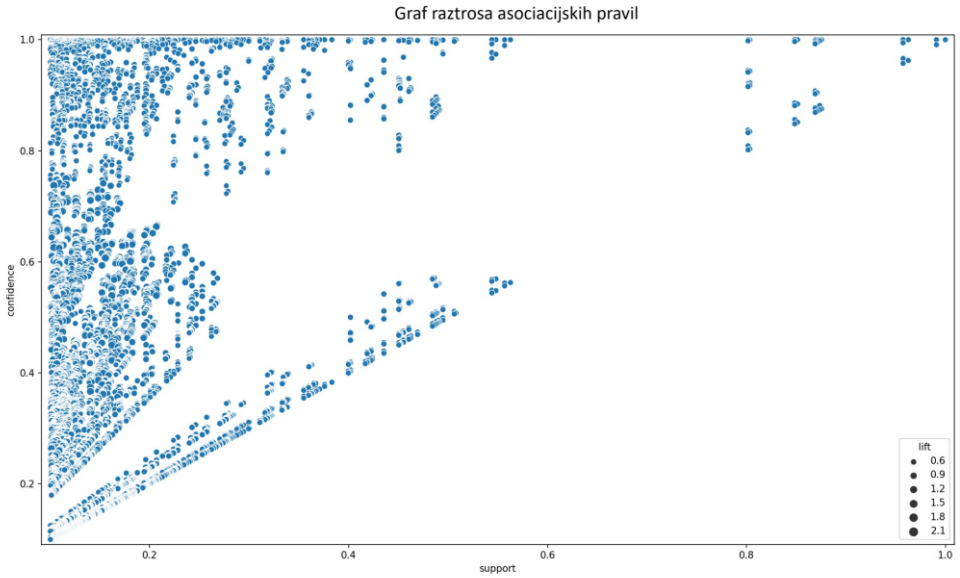
Slika 12 : Stolpični graf števila pojavitev elementov

Prvi korak pri rudarjenju je vedno izračun pogostih množic, ki predstavljajo osnovo za nadaljnje rudarjenje asociativnih pravil. Pri implementaciji in izračunu pogostih množic smo uporabili najmanjšo dovoljeno vrednost podpore 0,9 in najmanjšo dovoljeno vrednost dviga 1. S tem smo pridobili 15 posameznih pogostih množic z vrednostjo podpore, večjo kot 0,9.



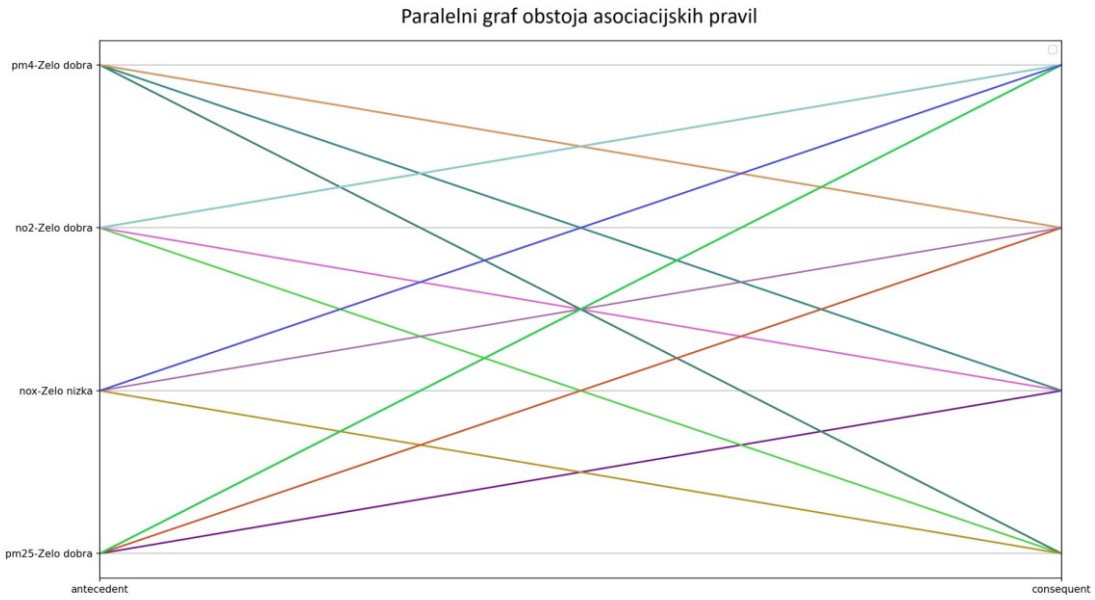
Slika 13 : Stolpični graf pogostih množic s podporo

Za globalno predstavitev vseh podatkov, ki so na voljo, smo s celotno množico podatkov izračunali asociacijska pravila, ki vključujejo podporo, zaupanje in dvig. Ugotovitve smo predstavili v grafu raztrosa, kjer položaj na osi y predstavlja zaupanje, položaj na osi x predstavlja podporo in velikost posameznega elementa predstavlja vrednost dviga. Iz grafa je razvidno, da imamo največ elementov na levi strani, kar pomeni, da ima večina asociacij nizko podporo. Manjši del v zgornjem desnem kotu pa ima visoko podporo in zaupanje. Ravno slednje nas tudi najbolj zanima.



Slika 14 : Graf raztrosa asociacijskih pravil

Iz izračunanih asociacijskih pravil, ki imajo podporo višjo od 0,9 in dvig višji od 1, smo lahko oblikovali paralelni graf prikaza, ali med elementi obstaja asociacijsko pravilo. V našem primeru lahko vidimo, da so vsi elementi povezani med seboj.



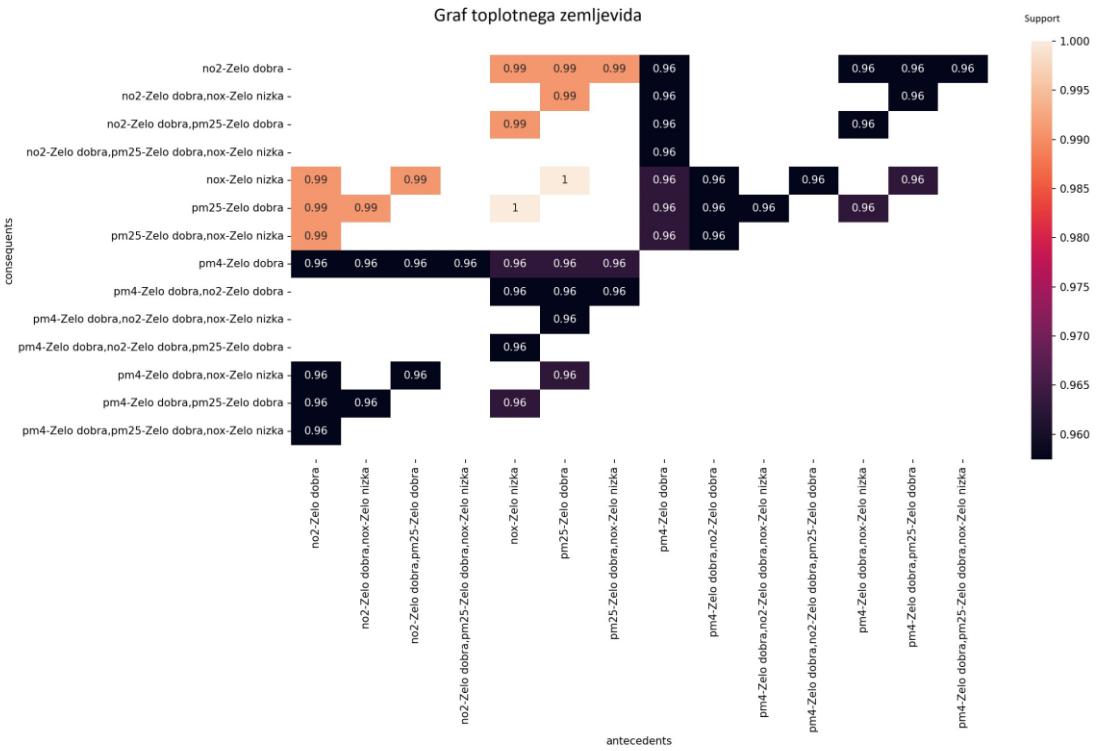
Slika 15 : Paralelni graf obstoja asociacijskih pravil

Eden izmed preglednejših pristopov prikaza asociacijskih pravil je graf toplotnega zemljevida. Toplotni zemljevid predstavlja, med katerimi pogostimi množicami obstaja asociativno pravilo in kakšna je vrednost podpore med njimi. Os x predstavlja predpostavke A, os y posledice B. Zemljevid torej prikazuje, v kolikšni meri se A in B pojavljata v isti transakciji. Iz toplotnega grafa je lepo razvidna najpogostejša asociacija, in sicer asociacija med elementom »nox-Zelo nizka« in »pm25-Zelo dobra«, ki se pojavlja v vsaki transakciji. Iz tega lahko razberemo, da je bila vrednost NOx in delcev PM2.5 v času merjenja vedno zelo nizka. Še nekaj asociacijskih pravil iz toplotnega grafa:

- (No2-Zelo dobra) ⇒ (pm25-Zelo dobra, nox-Zelo nizka),
- (No2-Zelo dobra) ⇒ (pm25-Zelo dobra),
- (No2-Zelo dobra) ⇒ (nox-Zelo nizka),
- (No2-Zelo dobra, nox-Zelo nizka) ⇒ (pm25-Zelo dobra),
- (No2-Zelo dobra, pm25-Zelo dobra) ⇒ (nox-Zelo nizka).

Ta pravila nam povedo sledeče: ko je vrednost NO_2 zelo dobra, je tudi vrednost PM2.5 dobra in prav tako vrednost NOx. Izpisana pravila veljajo tudi v nasprotno smer, torej ne samo A⇒B, ampak tudi B⇒A. Pojavnost teh pravil je v vseh izmerjenih meritvah 99%.

Predstavili smo samo nekaj ugotovljenih pravil, saj lahko iz grafa razberemo še številna druga z nižjo vrednostjo podpore.



Slika 16 : Graf toplotnega zemljevida

Do sedaj smo samo predstavljali vrednost podpore med asociacijskimi pravili. Za prej navedena pravila želimo izvedeti tudi vrednost zaupanja, torej kako močno je to pravilo med A ⇒B. Zato si bomo pomagali s paralelnim kategoričnim grafom, ki na levi strani predstavlja vse predpostavke in na levi vse posledice, ter povezavo z vrednostjo zaupanja med množicama.

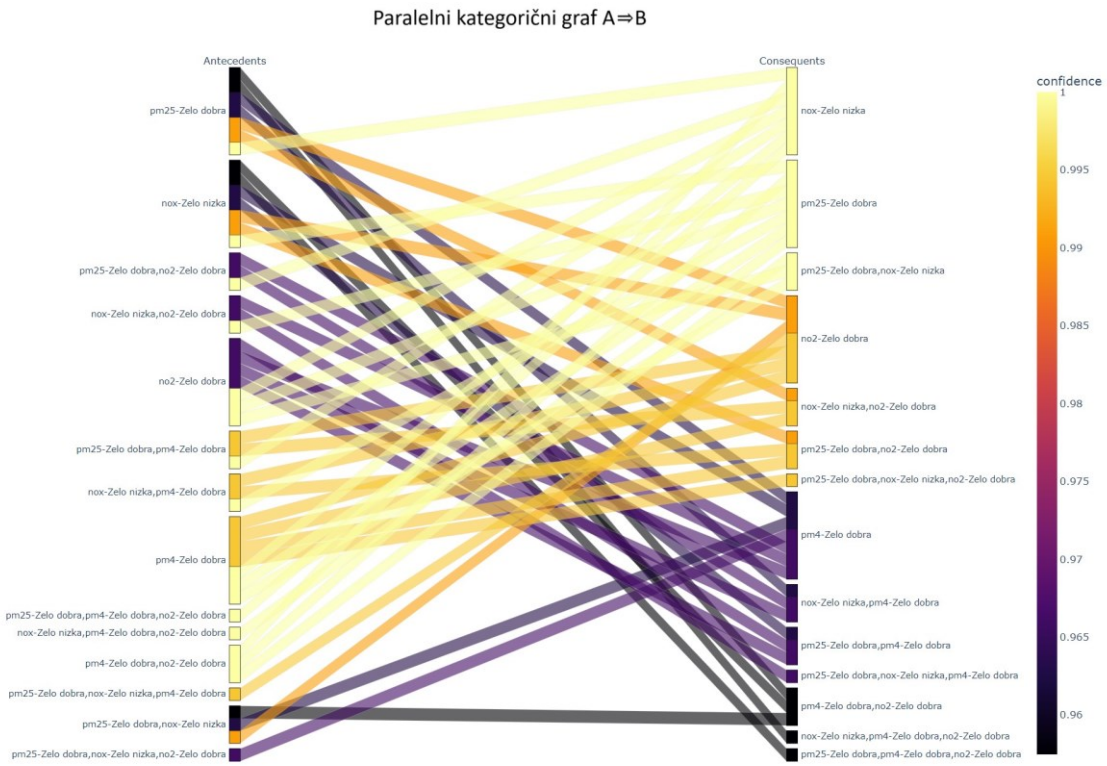
Če se obrnemo na prej odkrito asociativno pravilo med »nox-Zelo nizka« in »pm25-Zelo dobra« s podporo asociacije vrednosti 1, lahko iz grafa razberemo, da je vrednost zaupanja asociacije med elementoma tudi 1. Torej lahko sklepamo, da je to pravilo zelo močno.

V tem grafu lahko najdemo ista asociacijska pravila, kot smo jih že našteli pri toplotnem grafu:

- (No2-Zelo dobra) ⇒ (pm25-Zelo dobra, nox-Zelo nizka),

- (No2-Zelo dobra) ⇒ (pm25-Zelo dobra),
- (No2-Zelo dobra) ⇒ (nox-Zelo nizka),
- (No2-Zelo dobra, nox-Zelo nizka) ⇒ (pm25-Zelo dobra).

Za našteta asociacijska pravila lahko iz grafa razberemo, da ima vsako izmed teh pravil prav tako vrednost podpore 1, kar pomeni, da so pravila zelo močna.

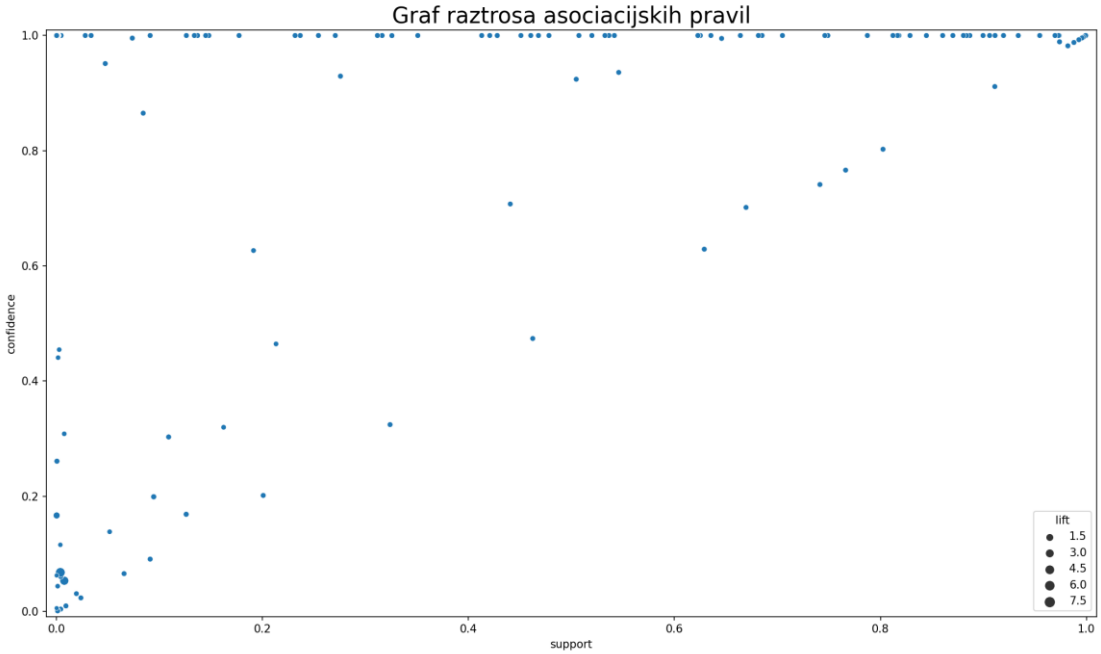


Slika 17 : Paralelni kategorični graf

5.3 Rezultati za numerične podatke

Za predstavitev vseh pravil, ki smo jih izračunali in nismo uporabili filtra, smo uporabili graf raztrosa, ki vključuje vrednosti podpore, zaupanja in dviga. Na grafu položaj na osi y predstavlja zaupanje, položaj na osi x pa podporo in velikost posameznega elementa vrednost dviga. Iz grafa je razvidno, da imamo velik delež asociacijskih pravil, ki imajo

zelo visoko vrednost zaupanja. Majhna gruča pravil se je ustvarila tudi v spodnjem levem kotu, kjer sta podpora in zaupanje zelo nizka.



Slika 18 : Graf raztrosa asociacijskih pravil za NARM

Za vsa asociacijska pravila, ki smo jih filtrirali z vrednostjo najmanjše dovoljene podpore vrednosti 0,9 in najmanjšega dovoljenega dviga vrednosti 1, smo izrisali graf toplotnega zemljevida. Os x predstavlja predpostavko A, os y pa posledico B. Vrednosti v grafu predstavljajo podporo med A in B. Pravila z najvišjimi vrednostmi podpore z vrednostjo 1 ali 0,99 so med atributom ozona (o3) in temperaturo (t) :

- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 49.99]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 49.67]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 46.44]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 43.65]).

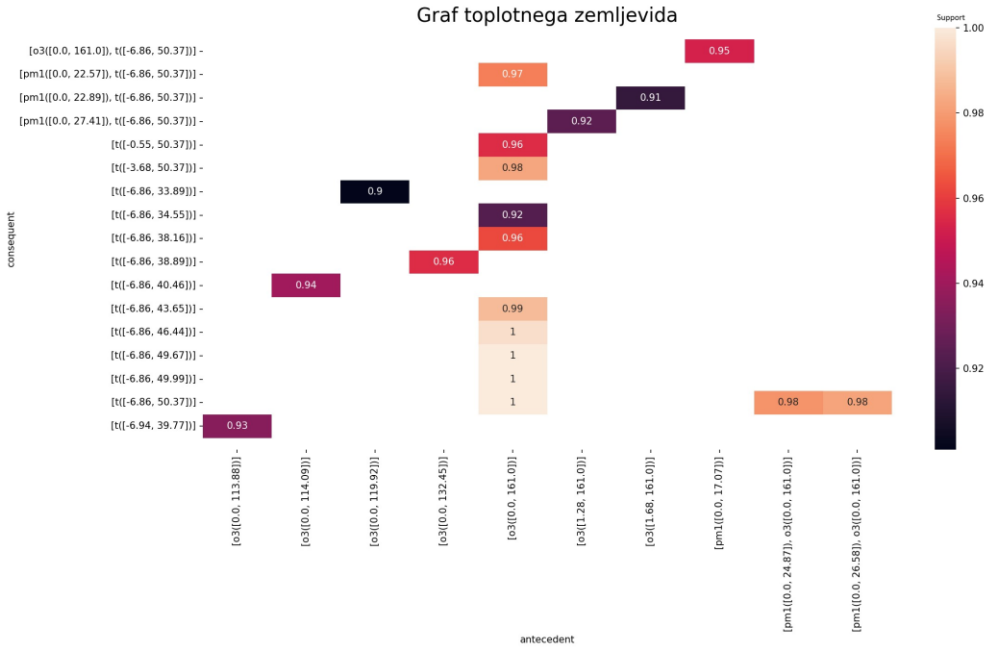
To pomeni, da se v primeru, ko je vrednost ozona med 0 in 161 µg/m^3, pojavi tudi vrednost temperature med –6,86 do 43–50 °C.

Prav tako lahko iz pravil z vrednostjo podpore 0,98 razberemo, da obstaja povezava med atributi PM1 (pm1), ozonom (o3) in temperaturo (t):

- (pm1 ∈ [0, 24.87], o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]),
- (pm1 ∈ [0, 26.58], o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]).

Iz teh povezav lahko sklepamo, da ko se pojavi vrednost PM1 med 0 in približno 25 µg/m^3 ter vrednost ozona med 0 do 161 µg/m^3, se bo pojavila temperatura med vrednostjo –6,86 in 50,37 °C.

Predstavili smo samo nekaj ugotovljenih pravil, saj lahko iz grafa razberemo še številna druga z nižjo vrednostjo podpore.



Slika 19 : Graf toplotnega zemljevida NARM

Za predstavitev zaupanja med izračunanimi pravili smo se ponovno odločili za paralelni kategorični graf. Na levi strani grafa imamo vse predpostvake A in na desni strani vse posledice B. Toplota barve povezav med A in B predstavlja vrednost zaupanja med množicama. V našem primeru imamo pravila z najvišjimi vrednostmi zaupanja 1 med:

- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 49.99]),
- (o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 49.67]).

Pravila, predstavljena v prejšnjem grafu, so imela podporo 1. To pomeni, da se ta pravila pojavljajo v vsaki meritvi in imajo tudi zelo močno zaupanje v pravilo.

Naslednja pravila imajo vrednost zaupanja prav tako 1:

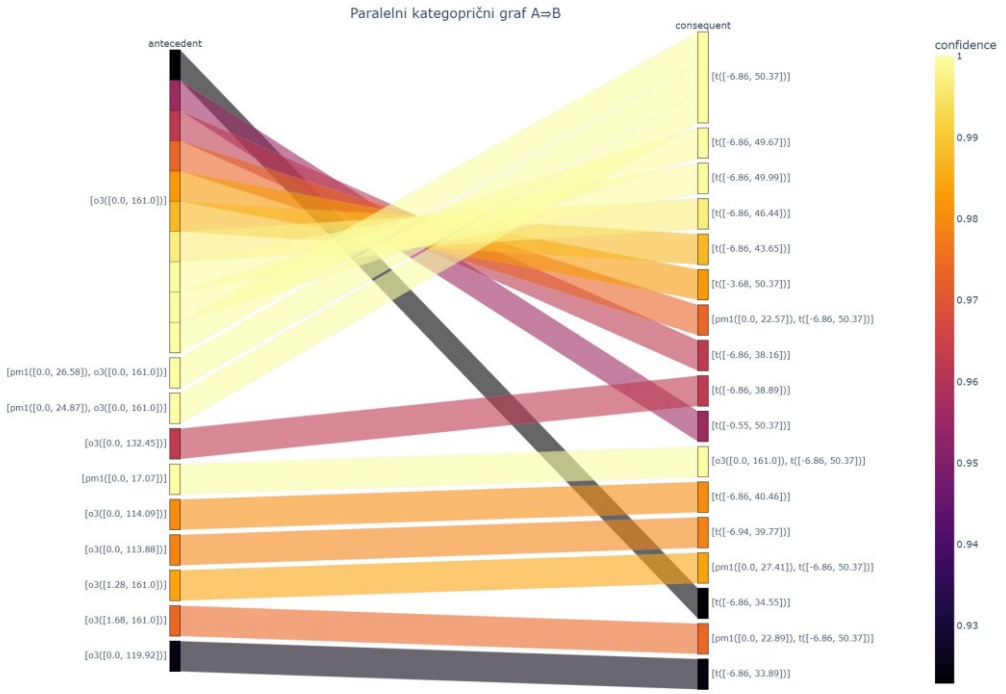
- (pm1 ∈ [0, 24.87], o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]),
- (pm1 ∈ [0, 26.58], o3 ∈ [0, 161]) ⇒ (t ∈ [-6.86, 50.37]).

Zgornji pravili smo predstavili že v prejšnjem grafu in imata vrednost podpore 0,99, kar pomeni, da se pojavita v 99 % vseh meritev in imata močno zaupanje v pravilo.

Pravilo, ki ima prav tako vrednost zaupanja 1, je :

- (pm1 ∈ [0, 17.07]) ⇒ (o3 ∈ [0, 161], t ∈ [-6.86, 50.37]).

To pravilo predstavlja pogoj: kjer se pojavi atribut PM1 z vrednostjo med 0 in 17,07 µg/m^3, se nato pojavita tudi ozon z vrednostjo med 0 in 161 µg/m^3 in temperatura med –6,86 in 50,37 °C. Iz toplotnega zemljevida je razvidno, da se to pravilo pojavlja v 95 % vseh meritev in ima prav tako zelo močno zaupanje.



Slika 20 : Paralelni kategorični graf NARM

6 DISKUSIJA

V uvodnem delu smo postavili nekaj raziskovalnih vprašanj, na katera smo posredno in tudi neposredno odgovorili skozi diplomsko delo. Tukaj bomo na kratko predstavili odgovore na postavljena vprašanja.

RV1: Kako in na kakšen način pridobivamo podatke, primerne za podatkovno rudarjenje?

V diplomskem delu smo na to vprašanje odgovorili v 3. poglavju. Na praktičnem primeru smo predstavili, kako z mikrokrmilnikom in senzorjem na preprost način pridobivamo podatke o kvaliteti zraka in jih shranjujemo na spletnem strežniku. Predstavili smo tudi postopek, kako te pridobljene podatke združimo z zunanjimi viri in jih nato sistematično očistimo ter pripravimo za uporabo v algoritmih rudarjenja asociativnih pravil.

RV2: Kako delujejo algoritmi za rudarjenje asociativnih pravil?

Delovanje algoritmov je v diplomskem delu natančno opisano v 4. poglavju. V tem poglavju smo opisali algoritme Apriori, Eclat, Fp-growth in algoritem optimizacije roja delcev. Teoretično smo predstavili njihovo delovanje in tudi praktičen način implementacije. Na kratko, algoritmi za rudarjenje asociativnih pravil delujejo tako, da iz neke podatkovne množice zberejo elemente, ki se pogosto pojavljajo skupaj. Preko teh pogostih množic lahko pridobimo podatke o določenih povezavah med elementi oziroma asociacijah med njimi. Različni algoritmi uporabljajo različne pristope, kako pridejo do teh asociacijskih pravil. Osnovni cilj za rudarjenje asociativnih pravil ostaja enak pri vseh algoritmih, samo postopek, kako posamezni algoritmi pridejo do rezultatov, so različni.

RV3: Kateri algoritmi za rudarjenje asociativnih pravil so primernejši za našo podatkovno množico?

Na to vprašanje smo natančneje odgovorili v poglavju 5.1, kjer smo primerjali delovanje algoritmov Apriori, Eclat in Fp-growth nad našo podatkovno množico. Izmed izmerjenih algoritmov je bil najhitrejši Fp-growth. Bil je trikrat hitrejši od najpočasnejšega Apriori in 30 % hitrejši od drugega najhitrejšega algoritma Eclat.

RV4: Kako predstaviti in razložiti rezultate rudarjenja asociativnih pravil?

Na to vprašanje smo neposredno odgovorili v poglavjih 5.2 in 5.3, kjer smo predstavili rezultate za kategorične in numerične podatke. Rezultate smo predstavili predvsem s pomočjo grafov, saj smo tako lahko zajeli večjo sliko, kot če bi vse rezultate ubesedili. Pri vsakem grafu smo tudi opisali, katere parametre smo uporabili, ter opisali, kaj prikazuje graf. Uporabili smo stolpčni graf za predstavitev pogostih množic s podporo, graf raztrosa za predstavitev razmerja med podporo, zaupanjem in dvigom, paralelni graf za prikaz obstoja asociacijskih pravil, graf toplotnega zemljevida za prikaz podpore med predpostavkami in posledicami ter paralelni kategorični graf za prikaz zaupanja med predpostavkami in posledicami. Nekaj asociacijskih pravil, ki so imele najvišjo podporo, smo razložili in jih predstavili v obliki implikacije (A) ⇒ (B).

RV5: Kakšne skrite informacije lahko odkrijemo iz podatkov lastnih meritev?

Kot skrite informacije smo v našem primeru odkrivali asociacijska pravila med podatki. Odkrita asociacijska pravila in njihove ugotovitve smo predstavili v poglavjih 5.2 in 5.3. Ker so si bile naše meritve precej podobne, je veliko podatkov imelo zelo visoko podporo in smo prejeli veliko število asociacijskih pravil. V diplomskem delu smo predstavili samo tistih nekaj z najvišjimi vrednostmi podpore. Pri kategoričnih podatkih smo odkrili, da se zelo pogosto skupaj pojavljajo no2, pm25 in nox. Pri numeričnih podatkih pa smo odkrili, da se zelo pogosto hkrati pojavljajo o3, t in pm1. Iz različnih grafov smo lahko tudi razbrali, kje se nahaja večina pogostih množic, kot na primer pri grafu raztrosa. V glavnem smo ugotovili predvsem, kateri podatki se pojavljajo skupaj z drugimi in tudi to, v kolikšni meri.

7 SKLEP

Na začetku diplomskega dela smo v teoretičnem delu predstavili rudarjenje asociativnih pravil. Opisali smo tako klasično (ARM) kot tudi numerično (NARM) rudarjenje. Nato smo predstavili proces pridobivanja podatkov iz merilne postaje, tj. kako smo podatke shranjevali na spletnem strežniku in jih preko preproste skripte pridobivali za nadaljnjo obdelavo. Izmerjene podatke smo predstavili, jih očistili ter obdelali. Za klasično rudarjenje asociativnih pravil smo podatke tudi diskretizirali.

S predhodno pridobljenim znanjem teorije rudarjenja asociativnih pravil smo lahko na preprost način predstavili, kako delujejo algoritmi Apriori, ECLAT in Fp-growth. Za numerično rudarjenje asociativnih pravil smo predstavili algoritem optimizacije roja delcev, enega izmed številnih algoritmov, delujočih po vzorih iz narave. Za vse prej opisane algoritme smo predstavili načine implementacije in s praktičnim primerom prikazali implementacijo ter delovanje.

Predstavili smo tudi prednosti in slabosti ARM algoritmov Apriori, ECLAT in Fp-growth. Na primeru naše podatkovne množice smo primerjali njihove performanse in hitrost. Za konec smo vizualno prikazali vse rezultate rudarjenja asociativnih pravil in obrazložili, kaj posamezni najdeni rezultati pravil pomenijo.

Podatkovna množica, ki smo jo uporabili, je bila s samo 17.000 vrsticami precej majhna, kar je omejilo naše ugotovitve. Namesto da bi pridobili veliko podatkovno množico s spleta, smo se lotili lastnega zbiranja podatkov. Menimo, da je bil postopek prikaza pridobivanja podatkov dober pokazatelj, koliko dela gre v zbiranje kvalitetnih in relevantnih podatkov. Diplomsko delo ni bilo mišljeno kot odkrivanje nečesa povsem nepoznanega. Namen je bil predstaviti pristop k rudarjenju asociativnih pravil od zbiranja podatkov, obdelave, razumevanja podatkov, razumevanja algoritmov do implementacije in interpretacije samih rezultatov.

VIRI IN LITERATURA

[1] N. S. Represa, A. Fernández-Sarría, A. Porta, in J. Palomar-Vázquez, „Data Mining Paradigm in the Study of Air Quality“, *Environ. Process.*, let. 7, št. 1, str. 1–21, mar. 2020, doi: 10.1007/s40710-019-00407-5.

[2] I. Fister jr in I. Fister, „Algoritem BatMiner za rudarjenje asociativnih pravil“, *Presek*, št. letnik 44, številka 5, str. 26-29., jan. 2017.

[3] „What is Data Mining? | IBM“. https://www.ibm.com/topics/data-mining (pridobljeno 7. april 2023).

[4] D. H. Goh in R. P. Ang, „An introduction to association rule mining: An application in counseling and help-seeking behavior of adolescents“, *Behavior Research Methods*, let. 39, št. 2, str. 259–266, maj 2007, doi: 10.3758/BF03193156.

[5] M. Hahsler, B. Grün, in K. Hornik, „**arules** - A Computational Environment for Mining Association Rules and Frequent Item Sets“, *J. Stat. Soft.*, let. 14, št. 15, 2005, doi: 10.18637/jss.v014.i15.

[6] J. Heaton, „Comparing Dataset Characteristics that Favor the Apriori, Eclat or FP-Growth Frequent Itemset Mining Algorithms“. arXiv, 30. januar 2017. doi: 10.48550/arXiv.1701.09042.

[7] M. Kaushik, R. Sharma, I. Fister Jr., in D. Draheim, „Numerical Association Rule Mining: A Systematic Literature Review“. arXiv, 2. julij 2023. Pridobljeno: 28. avgust 2023. [Na spletu]. Dostopno na: http://arxiv.org/abs/2307.00662

[8] Ž. Stupan in I. F. Jr., „NiaARM: A minimalistic framework for NumericalAssociation Rule Mining“, *JOSS*, let. 7, št. 77, str. 4448, sep. 2022, doi: 10.21105/joss.04448.

[9] G. Vrbančič, L. Brezočnik, U. Mlakar, D. Fister, in I. Fister, „NiaPy: Python microframework for building nature-inspired algorithms“, *Journal of Open Source Software*, let. 3, št. 23, str. 613, mar. 2018, doi: 10.21105/joss.00613.

[10] „Zadnje urne in dnevne ravni onesnaževal“. https://www.arso.gov.si/zrak/kakovost%20zraka/podatki/dnevne_koncentracije.html (pridobljeno 7. april 2023).

[11] „PM1 - The new focus to protect human health“, *Camfil*. https://www.camfil.com/en/insights/standard-and-regulations/pm1-is-most-harmful (pridobljeno 7. april 2023).

[12] „Sensirion_Datasheet_Environmental_Node_SEN5x.pdf“. Pridobljeno: 7. april 2023. [Na spletu]. Dostopno na: https://sensirion.com/media/documents/6791EFA0/62A1F68F/Sensirion_Datasheet_Environmental_Node_SEN5x.pdf

[13] „Relative Humidity - an overview | ScienceDirect Topics“. https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/relative-humidity (pridobljeno 7. april 2023).

[14] O. US EPA, „What are volatile organic compounds (VOCs)?“, 19. februar 2019. https://www.epa.gov/indoor-air-quality-iaq/what-are-volatile-organic-compounds-vocs (pridobljeno 7. april 2023).

[15] „Nitrogen oxides, NOx — European Environment Agency“. https://www.eea.europa.eu/help/glossary/eper-chemicals-glossary/nitrogen-oxides-nox (pridobljeno 7. april 2023).

[16] „APOA-370“. https://www.horiba.com/int/process-and-environmental/products/detail/action/show/Product/apoa-370-450/ (pridobljeno 13. april 2023).
[17] „APNA-370“. https://www.horiba.com/int/process-and-environmental/products/detail/action/show/Product/apna-370-451/ (pridobljeno 13. april 2023).
[18] „APDA-372“. https://www.horiba.com/usa/process-and-environmental/products/detail/action/show/Product/apda-372-194/ (pridobljeno 13. april 2023).
[19] H. Tx, „To contact us: sales@chromatotec.com“.
[20] R. Teja, „Getting Started with ESP32 | Introduction to ESP32“, *Electronics Hub*, 17. februar 2021. https://www.electronicshub.org/getting-started-with-esp32/ (pridobljeno 7. april 2023).
[21] „Smart sensor solutions“. https://sensirion.com/products/catalog/SEN55/ (pridobljeno 7. april 2023).
[22] „arduino-ble-gadget/documents/SEN55_BLE_Gadget_Tutorial.md at master · Sensirion/arduino-ble-gadget“, *GitHub*. https://github.com/Sensirion/arduino-ble-gadget/blob/master/documents/SEN55_BLE_Gadget_Tutorial.md (pridobljeno 31. avgust 2023).
[23] L. Sánchez, „Everything You Need to Know About Nylon Filament for 3D Printing“, *BCN3D Technologies*, 5. februar 2020. https://www.bcn3d.com/everything-you-need-to-know-about-nylon-filament-for-3d-printing/ (pridobljeno 7. april 2023).
[24] „Razlaga pojmov in priporočila“. https://www.arso.gov.si/zrak/kakovost%20zraka/podatki/amp/razlaga_pojmov.html (pridobljeno 20. avgust 2023).
[25] „PyPI · The Python Package Index“, *PyPI*. https://pypi.org/ (pridobljeno 11. september 2023).
[26] „Apriori Algorithm“, *GeeksforGeeks*, 4. september 2018. https://www.geeksforgeeks.org/apriori-algorithm/ (pridobljeno 11. september 2023).
[27] J. Korstanje, „The Apriori algorithm“, *Medium*, 24. september 2021. https://towardsdatascience.com/the-apriori-algorithm-5da3db9aea95 (pridobljeno 20. avgust 2023).
[28] J. Korstanje, „The Eclat algorithm“, *Medium*, 29. september 2021. https://towardsdatascience.com/the-eclat-algorithm-8ae3276d2d17 (pridobljeno 21. avgust 2023).
[29] „ML | ECLAT Algorithm“, *GeeksforGeeks*, 11. junij 2019. https://www.geeksforgeeks.org/ml-eclat-algorithm/ (pridobljeno 21. avgust 2023).
[30] „FP Growth Algorithm in Data Mining - Javatpoint“, *www.javatpoint.com*. https://www.javatpoint.com/fp-growth-algorithm-in-data-mining (pridobljeno 22. avgust 2023).
[31] J. Korstanje, „The FP Growth algorithm“, *Medium*, 28. september 2021. https://towardsdatascience.com/the-fp-growth-algorithm-1ffa20e839b8 (pridobljeno 22. avgust 2023).

[32] G. Li, T. Wang, Q. Chen, P. Shao, N. Xiong, in A. Vasilakos, „A Survey on Particle Swarm Optimization for Association Rule Mining“, *Electronics*, let. 11, št. 19, Art. št. 19, jan. 2022, doi: 10.3390/electronics11193044.

[33] „One Hot Encoding in Machine Learning“, *GeeksforGeeks*, 12. junij 2019. https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/ (pridobljeno 25. avgust 2023).

[34] Tarek, „Mastering Association Rule Learning: Pros and Cons of Apriori, Eclat, and FP-growth“, *Medium*, 5. maj 2023. https://medium.com/@tarek.tm/mastering-association-rule-learning-pros-and-cons-of-apriori-eclat-and-fp-growth-530ff46ed1d9 (pridobljeno 26. avgust 2023).