VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
INFORMACINIŲ SISTEMŲ INŽINERIJOS STUDIJŲ PROGRAMA

Duomenų tyryba. Klasterizavimo algoritmų analizė.

Savarankiško darbo ataskaita.

Atliko: Justinas Rimavičius, Edvardas Ražanskas

VU el. p.: edvardas.razanskas@mif.stud.vu.lt, justinas.rimavicius@mif.stud.vu.lt

Vertino: dr. Jolita Bernatavičienė

Vilnius
2024

TURINYS

1. ĮVADAS

1.1. Darbo tikslas ir uždaviniai

Šio darbo tikslas – SDSS (Sloan‘o skaitmeninio dangaus tyrimo DR17) duomenų imčiai su pilnu ir atrinktų požymių rinkiniu, bei tos paties duomenų imties sumažintos dimensijos duomenų rinkiniui pritaikyti „kMeans“ ir „DBSCAN“ klasterizavimo algoritmus, palyginti jų rezultatus, apibrėžti susidariusių klasterių specifiką.

Darbo uždaviniai:

1. Trumpai aprašyti tiriamą duomenų aibę, jos požymius, pagrindines savybes.
2. Pasirinkti ir pagrįsti pagal kokius požymius bus atliekamas klasterizavimas.
3. Naudojant „Elbow“ ir „Silhouette“ metodus įvertinti optimalų klasterių skaičių.
4. Suklasterizuoti duomenis naudojant „kMeans“ ir „DBSCAN“ klasterizavimo algoritmus.
5. Patikrinti, kokią įtaką daro klasterizavimui išskirtys bei duomenų dimensijos mažinimas. Kaip keičiasi tendencijos klasteriuose?
6. Apibendrinti rezultatus, pastebėtas tendencijas klasteriams, pateikti jų interpretaciją.

1.2. Darbo įrankiai

Duomenų apdorojimas, transformacija, analizė, dimensijų mažinimo metodai ir klasterizavimo metodai buvo pritaikyti naudojant „Python 3.12.0” programavimo kalbą ir jos bibliotekas (daugiau žiūrėti skyrių Kodas).

2. DUOMENŲ ANALIZĖ

2.1. Tiriamos duomenų aibės ir jos požymių aprašymas

Pateiktoje žvaigždžių klasifikacijos duomenų aibėje („Stellar Classification Dataset“) yra 100000 eilučių, 18 požymių stulpelių. Jutiklių matavimai yra „float“ tipo (t.y. priklauso realiųjų skaičių aibei) , „class“ požymis yra „object“ tipo (t.y. simboliai), likę požymiai yra „int“ tipo (t.y. priklauso sveikųjų skaičių aibei).

```
Data columns (total 18 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
 0   obj_ID       100000 non-null  float64
 1   alpha        100000 non-null  float64
 2   delta        100000 non-null  float64
 3   u            100000 non-null  float64
 4   g            100000 non-null  float64
 5   r            100000 non-null  float64
 6   i            100000 non-null  float64
 7   z            100000 non-null  float64
 8   run_ID       100000 non-null  int64
 9   rerun_ID     100000 non-null  int64
 10  cam_col      100000 non-null  int64
 11  field_ID     100000 non-null  int64
 12  spec_obj_ID  100000 non-null  float64
 13  class        100000 non-null  object
 14  redshift     100000 non-null  float64
 15  plate        100000 non-null  int64
 16  MJD          100000 non-null  int64
 17  fiber_ID     100000 non-null  int64
dtypes: float64(10), int64(7), object(1)
```

2.1 pav. pradinė duomenų aibė

Duomenų aibės požymių aprašymai:

- obj_ID = objekto identifikatorius, unikali dangaus kūno vertė, identifikuojanti objektą CAS naudojamame vaizdų kataloge.
- alpha = dešiniojo pakilimo kampas (pagal J2000 epochą)
- delta = deklinacijos kampas (pagal J2000 epochą)
- u = ultravioletinis astrofotometrinės sistemos filtras
- g = žaliasis astrofotometrinės sistemos filtras
- r = raudonasis astrofotometrinės sistemos filtras
- i = artimųjų infraraudonųjų spindulių filtras astrofotometrinė sistemoje

- z = infraraudonųjų spindulių filtras astrofotometrinė sistemoje
- run_ID = serijos numeris, naudojamas konkrečiam nuskaitymui identifikuoti
- rereun_ID = pakartotinio paleidimo numeris, nurodantis, kaip vaizdas buvo apdorotas
- cam_col = kameros stulpelis, skirtas skenavimo linijai nustatyti
- field_ID = lauko numeris kiekvienam laukui identifikuoti
- spec_obj_ID = unikalus optinių spektroskopinių objektų ID (tai reiškia, kad 2 skirtingi stebėjimai su tuo pačiu spec_obj_ID turi turėti bendrą išvesties klasę)
- class = objekto klasė (galaktika, žvaigždė arba kvazaras)
- redshift (raudonasis poslinkis) = raudonojo poslinkio vertė, pagrįsta bangos ilgio padidėjimu
- plate = plokštės ID, identifikuojantis kiekvieną SDSS plokštę
- MJD = modifikuota Julijaus data, naudojama nurodyti, kada buvo paimta tam tikra SDSS duomenų dalis
- fiber_ID = pluošto ID, identifikuojantis pluoštą, kuris nukreipė šviesą į židinio plokštumą kiekvieno stebėjimo metu

2.2. Požymių ir objektų apdorojimas

Pašalinti šie požymiai, nedarantys įtakos kosminio kūno klasifikavimui:

- „obj_ID“ požymis, nes tai identifikacinis numeris nedarantis įtakos duomenims;
- „alpha“ ir „delta“ požymiai nusako kosminio objekto poziciją, o jos nėra susijusios su skirtingų objektų (galaktikų, žvaigždžių, kvazarų) fizinėmis savybėmis;
- „spec_obj_ID“ požymis, nes 2 skirtingi stebėjimai su tuo pačiu spec_obj_ID turi turėti bendrą išvesties klasę, o visos šio požymio reikšmės yra skirtingos;
- „rerun_ID“ požymis, nes yra tik viena unikali reikšmė;
- „MJD“ požymis, nes ji simbolizuoja datą, kada užfiksuotas stebėjimas

Duomenų aibė neturėjo praleistų reikšmių. Tolimesniems uždaviniams pasirinkome „redshift“, „u“, „g“, „r“, „i“ ir „z“ požymius.

Duomenų aibė turėjo vieną eilutę, kurioje „u“, „g“ ir „z“ reikšmės buvo -9999, tad šią triukšmo eilutę panaikinome.

„Class“ požymis yra kategorinis požymis, kuris turi tris unikalias reikšmes duomenų aibėje: GALAXY – galaktika, QSO – kvazaras(ypač šviesus objektas galaktikos centre), STAR – žvaigždė. Kiekviena šių reikšmių buvo pakeista atitinkamai į skaičius 0, 1, 2.

2.3. Objektų atrinkimas

Tolimesniam duomenų analizavimui, apdorojimui ir vizualizavimui buvo atsitiktinai atrinkti po 1000 objektų iš kiekvienos klasės (2.2 pav.):

```
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   u          3000 non-null   float64
 1   g          3000 non-null   float64
 2   r          3000 non-null   float64
 3   i          3000 non-null   float64
 4   z          3000 non-null   float64
 5   run_ID     3000 non-null   int64
 6   cam_col    3000 non-null   int64
 7   field_ID   3000 non-null   int64
 8   class      3000 non-null   int64
 9   redshift   3000 non-null   float64
 10  plate      3000 non-null   int64
 11  fiber_ID   3000 non-null   int64
dtypes: float64(6), int64(6)
```

2.2 pav. duomenų aibė su pasirinktais objektais

2.4. Duomenų aibės normavimas

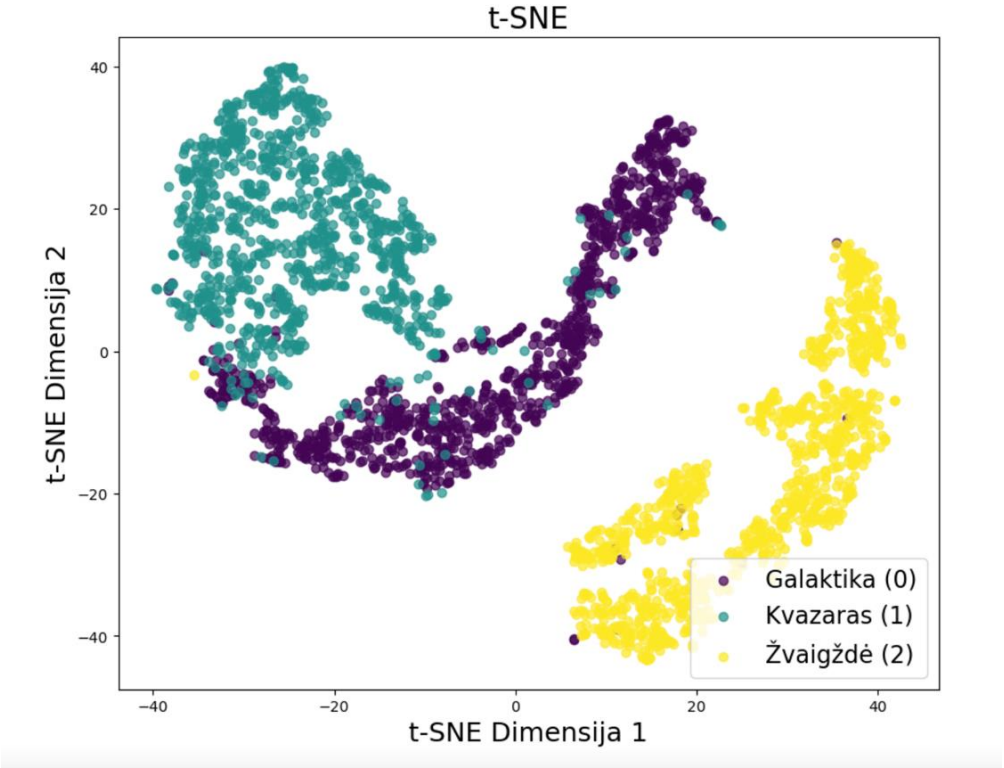
Duomenų normavimui buvo parinkti šie požymiai: „redshift“, „u“, „g“, „r“, „i“ ir „z“. Kiti požymiai buvo nenormuoti, nes jie yra arba identifikaciniai („run_ID“, „field_ID“, „cam_col“, „plate“, ir „fiber_ID“) arba kategoriniai („class“). Duomenys buvo normuoti naudojant „min-max“ metodą (2.3 pav.)

	u	g	r	i	z	run_ID	cam_col	field_ID	class	redshift	plate	fiber_ID
count	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000	3000.000000
mean	0.505650	0.534968	0.475907	0.557617	0.600594	4463.033667	3.531000	183.694667	1.000000	0.102498	5324.207333	451.430000
std	0.152596	0.139310	0.124104	0.149785	0.168291	1973.851258	1.592655	144.563996	0.816633	0.132210	3010.616165	274.568411
min	0.000000	0.000000	0.000000	0.000000	0.000000	109.000000	1.000000	12.000000	0.000000	0.000000	266.000000	1.000000
25%	0.395331	0.428814	0.383929	0.455900	0.485856	3051.250000	2.000000	82.000000	0.000000	0.000458	2682.750000	221.750000
50%	0.501572	0.564178	0.507172	0.584689	0.621816	4072.000000	4.000000	146.000000	1.000000	0.059794	5174.500000	438.000000
75%	0.607117	0.635859	0.569526	0.673526	0.726640	5415.000000	5.000000	239.250000	2.000000	0.162504	7696.250000	665.000000
max	1.000000	1.000000	1.000000	1.000000	1.000000	8162.000000	6.000000	980.000000	2.000000	1.000000	12547.000000	1000.000000

2.3 pav. min-max metodu normuotos duomenų aibės statistika

2.5. Sumažintos dimensijos duomenys

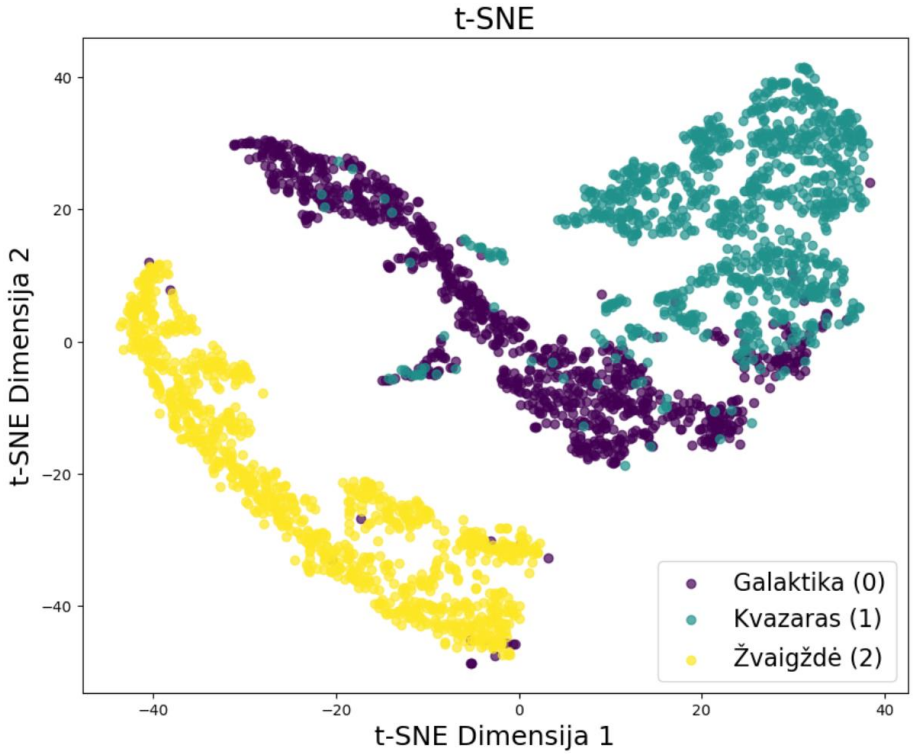
Žemiau esančiame grafike (2.4 pav.) matome „t-SNE“ grafiką, pritaikytą atrinktiems požymiams. „tSNE“ metodo parametrų reikšmės: "max_iter" reikšmė lygi 750, o "perplexity" lygus 50, "metric" lygi "canberra", „random_state“ reikšmė lygi 42. Metodas „t-SNE“ buvo taikytas atsitiktinai 3000 paimtiems objektų, kai parametras „random_state“ lygus 2. Šiuo metodu sumažinta iki 2 dimensijų duomenų aibė yra naudojama tolimesniuose žingsniuose – pagrinde „kMeans“ ir „DBSCAN“ klasterizavimo metodų vaizdavimui.



2.4 pav. "t-SNE" metodu sumažintos dimensijos duomenys, kai pradinis atsitiktinis duomenų rinkinys sudarytas naudojant „random_state“ lygu 2.

„DBSCAN“ klasterizavimui buvo naudojamas toks pat „t-SNE“ metodas, su tokiais pat parametrais, tačiau atsitiktiniai 3000 tiriamų objektų buvo paimti naudojant „random_state“ 42 (2.5 pav.) Matoma, kad objektai pasiskirstę panašiai, tačiau, kaip bus matoma vėliau (skyrius 0

Algoritmo jautrumas mažiems duomenų aibės pokyčiams), tai gali daryti didelę įtaką klasterizavimui.



2.5 pav. "t-SNE" metodu sumažintos dimensijos duomenys, kai pradinis atsitiktinis duomenų rinkinys sudarytas naudojant „random_state“ lygu 42.

3. KLASTERIZAVIMAS

3.1. „kMeans“ algoritmas/metodas

3.1.1. Aprašymas

kMeans (angl. „K-Means Clustering“) yra centroidų pagrindu veikiantis klasterizavimo algoritmas, kuris naudojamas grupėms (klasteriams) duomenyse identifikuoti pagal jų tarpusavio atstumus. Šis algoritmas priskiria taškus tam klasteriui, kurio centroidas yra arčiausiai, iteratyviai atnaujindamas centrų pozicijas, kol rezultatai stabilizuojasi. kMeans dažniausiai naudojamas tais atvejais, kai reikia aiškiai apibrėžtų klasterių, kurių skaičius (angl. „n_clusters“) nustatomas iš anksto. Dėl šios priežasties kMeans yra efektyvus, tačiau jo rezultatai labai priklauso nuo pradinio klasterių skaičiaus nustatymo.

kMeans algoritme svarbiausi parametrai yra:

- n_clusters: klasterių skaičius, kurį reikia iš anksto nustatyti pagal duomenų struktūrą arba optimizavimo metodus, tokius kaip Elbow ar Silhouette.
- init: pradinis centroidų nustatymo metodas, pvz., „k-means++“, kuris pagerina algoritmo efektyvumą, sumažindamas pradinių pozicijų jautrumą.
- max_iter: maksimalus iteracijų skaičius, per kurį centroidai optimizuojami.
- tol: tolerancijos riba, nustatanti, kiek centroidų pokytis tarp iteracijų gali būti ignoruojamas.

Šiuo atveju K-Means algoritmas buvo taikytas šioms normuotoms „min-max“ metodu duomenų aibėms:

- Pilnai duomenų aibei, ištrynus klasės požymį;
- Duomenų aibei su atrinktais požymiais („u“, „g“, „r“, „i“, „z“, „redshift“);
- Duomenų aibei su tais pačiais atrinktais požymiais, pritaikius „t-SNE“ metodą (dimensiškumo mažinimo metodą).

Taip pat buvo pastebėta, kad „n_clusters“ reikšmė turi būti nustatoma itin atidžiai, nes netinkamas klasterių skaičius gali lemti prastą grupių atskyrimą. Optimalus klasterių skaičius buvo identifikuotas naudojant Elbow ir Silhouette metodus.

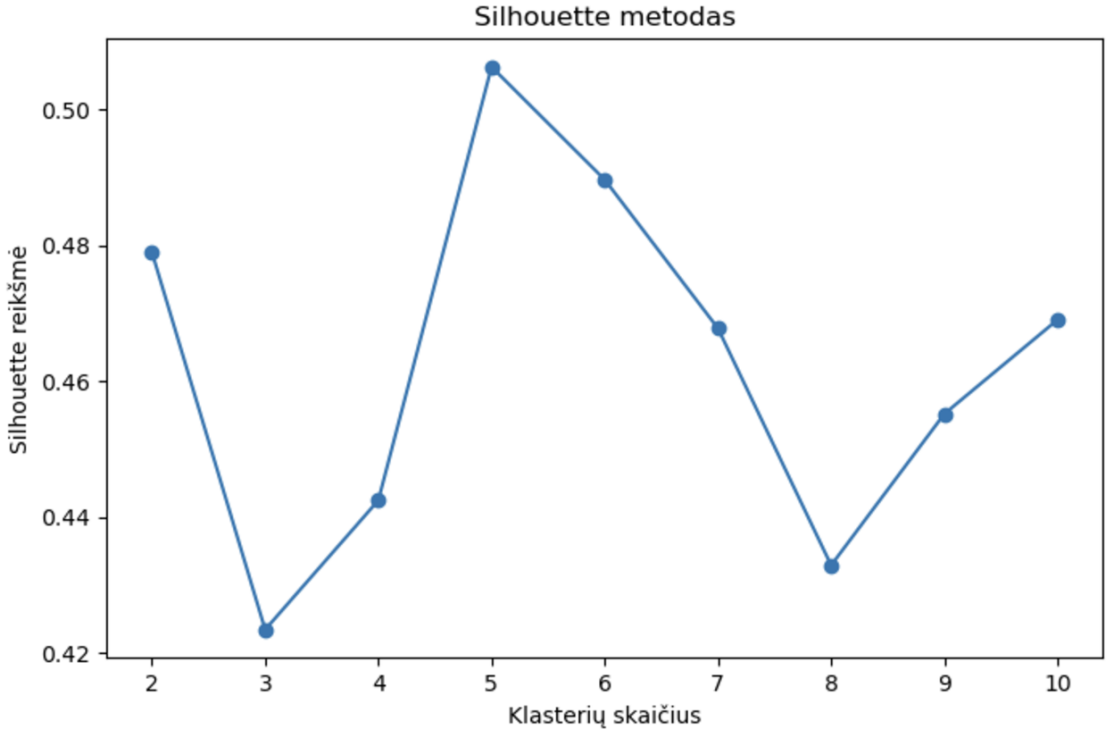
3.1.2. „n_clusters“ reikšmės nustatymas

Norint, kad „kMeans” metodas duotų reikšmingus rezultatus, svarbiausias parametras yra „n_clusters“. Šio parametro nustatymui naudojome 2 metodus – „Elbow“ ir „Silhouette“

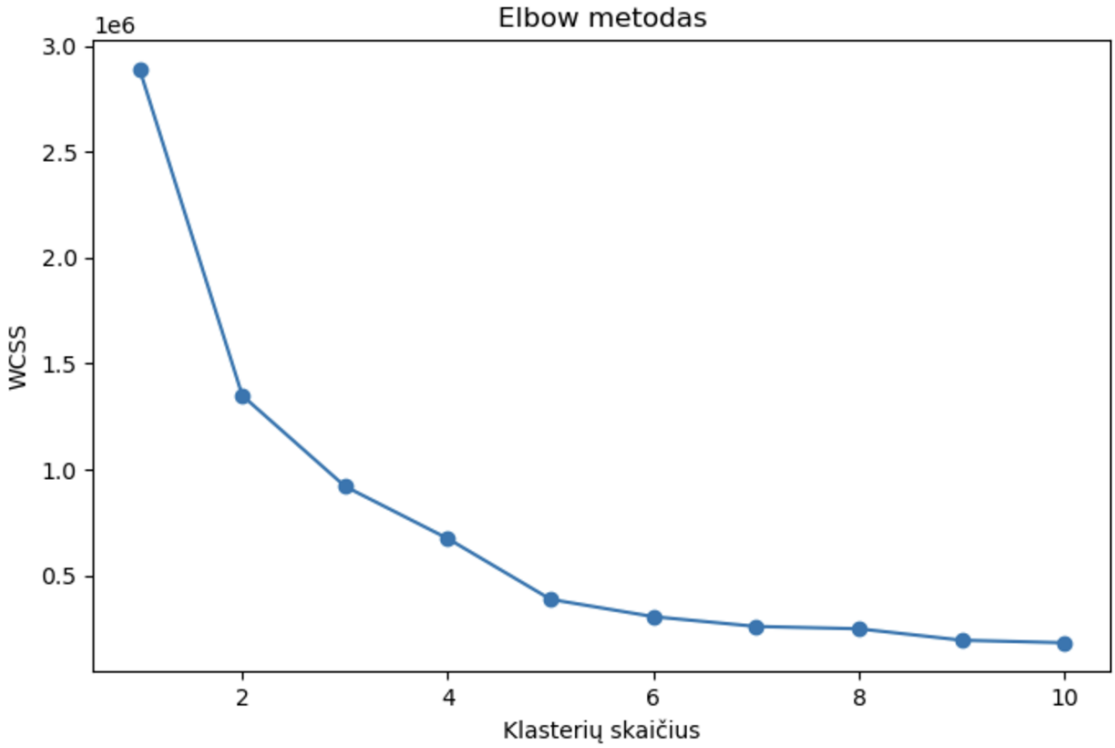
Elbow metodo esmė – apskaičiuoti klaidų sumą (SSE, angl. „Sum of Squared Errors”) kiekvienam klasterių skaičiui ir ją atvaizduoti grafike. SSE yra rodiklis, kuris parodo, kiek duomenų taškai nutolę nuo savo klasterio centro.

Kai klasterių skaičius didėja, SSE reikšmė mažėja, nes taškai yra arčiau savo klasterių centrų. „Elbow” metodo grafike dažnai pastebimas alkūnės taškas – vieta, kurioje SSE reikšmės mažėjimas sulėtėja. Šis taškas dažniausiai žymi optimalų klasterių skaičių, nes nuo šio momento didesnis klasterių skaičius neduoda reikšmingo pagerėjimo.

Silueto metodas yra naudojamas įvertinti klasterizavimo kokybę ir nustatyti optimalų klasterių skaičių. Šis metodas remiasi silueto koeficientu**,** kuris parodo, kaip gerai kiekvienas taškas priskirtas savo klasteriui, palyginti su kitais klasteriais.



3.1 pav. "Silhouette" reikšmių ir klasterių skaičiaus priklausomybės linijinė diagrama.



3.2 pav. "Elbow" reikšmių ir klasterių skaičiaus priklausomybės linijinė diagrama.

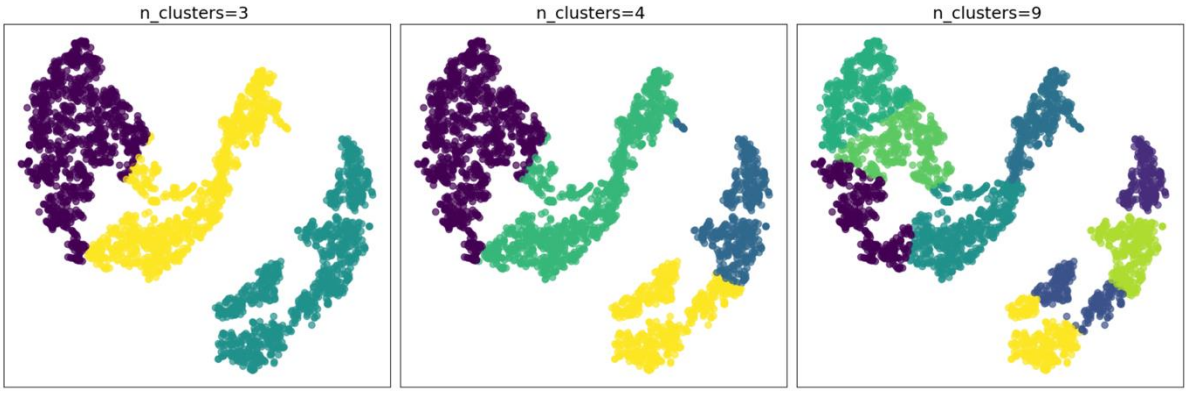
„Elbow“ metode (3.2 pav.) matoma alkūnė, kai klasterių skaičius yra 3, t.y. esant daugiau nei 3 klasteriams, klasterizavimas nesuteikia daug aiškesnio rezultato.

„Silhouette“ metode (3.1 pav.) matoma, jog esant 3 klasteriams, „Silhouette“ skaičius yra 0.42, o esant 5 jis yra 0.51, t.y. didžiausias. Tai nėra optimaliausio klasterių skaičiaus suradimas, nes šie metodai buvo pritaikyti t-SNE dimensijų mažintiems duomenims, o šis metodas neišlaiko globalių atstumų tarp objektų, kas yra svarbu šiam metodui. Sekančiame punkte bus aiškiau matoma, kaip „Silhouette“ metodo skaičius pasikeičia ir yra rodoma, jog optimalus klasterių skaičius yra 3, kai jo analizė yra atliekama nemažintiems dimensijų duomenims (3.8 pav. - 3.1.5 skiltis).

3.1.3. Algoritmo taikymas sumažintos dimensijos duomenims

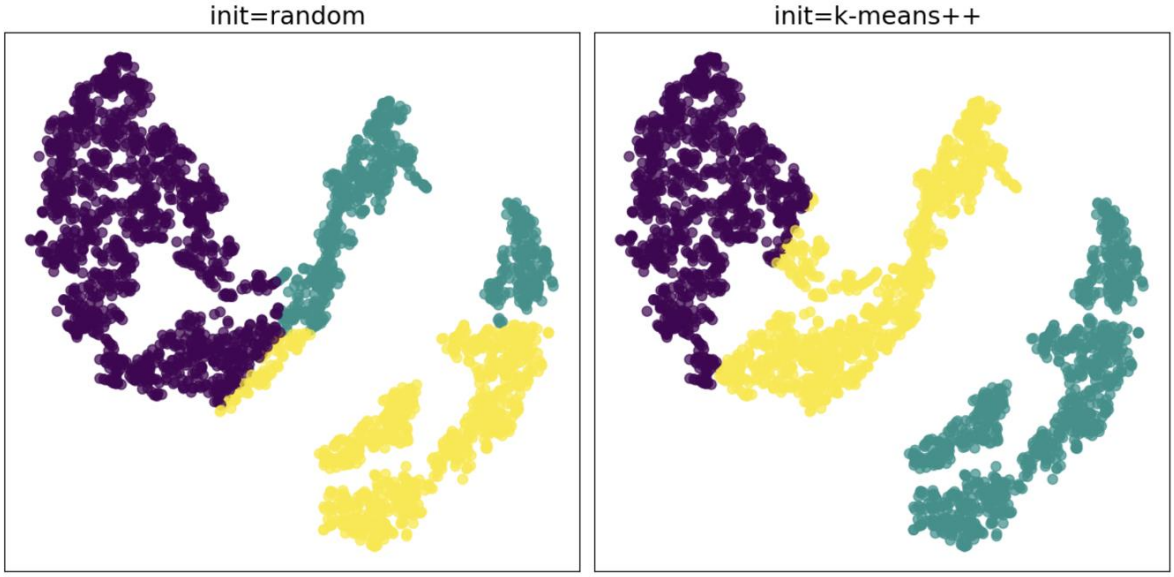
Sekančiuose grafikuose atvaizduojami „kMeans“ algoritmai t-SNE metodu sumažintos dimensijos duomenims.

Pirmasis grafikas(xx pav.) vaizduoja „n_clusters“ parametro pokytį. Matoma, jog efektyviausias rezultatas esant 3 klasteriams – sekančiuose grafikuose šis skaičius ir bus naudojamas.



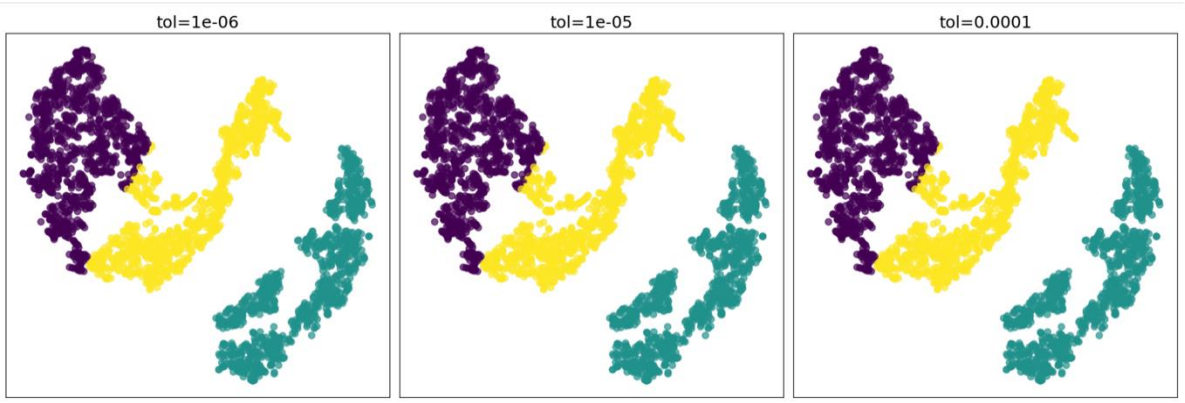
3.3 pav. "kMeans" klasterizavimas keičiantis "n_clusters" parametro reikšmėms.

Sekantis grafikas – „init“ parametro keitimas (3.4 pav.). Pasirinkus „k-means++“ reikšmę klasterių centrai yra optimalesnėse vietose, nei naudojant „random“.

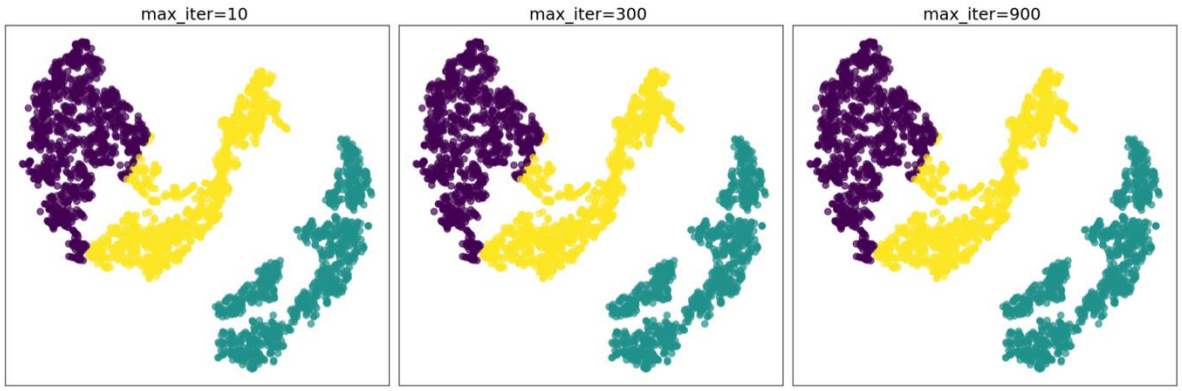


3.4 pav. "kMeans" klasterizavimas keičiant "init" parametro reikšmes.

Sekantys 2 grafikai(3.5 pav. ir 3.6 pav.) vaizduoja „max_iter“ ir „tol“ reikšmių pokyčius, tačiau šie parametrai „kMeans“ rezultato nepakeičia.



3.5 pav. "kMeans" klasterizavimas keičiant "tol" parametro reikšmes.



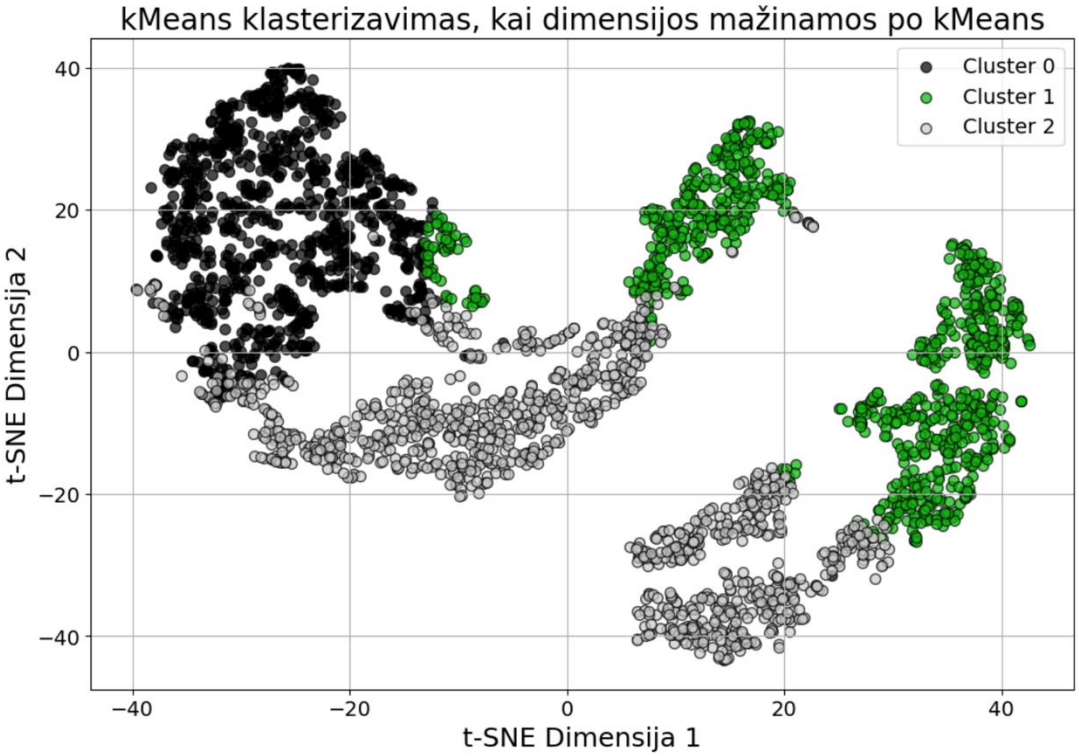
3.6 pav. "kMeans" klasterizavimas keičiant "max_iter" parametro reikšmes.

Tolimesnei analizei buvo pasirinkti šie parametrai: „n_clusters“ lygus 3, „init“ lygus „k-means++“, „tol“ lygus „0.0001“ ir „max_iter“ lygus 300.

3.1.4. Algoritmo taikymas prieš mažinant dimensijas.

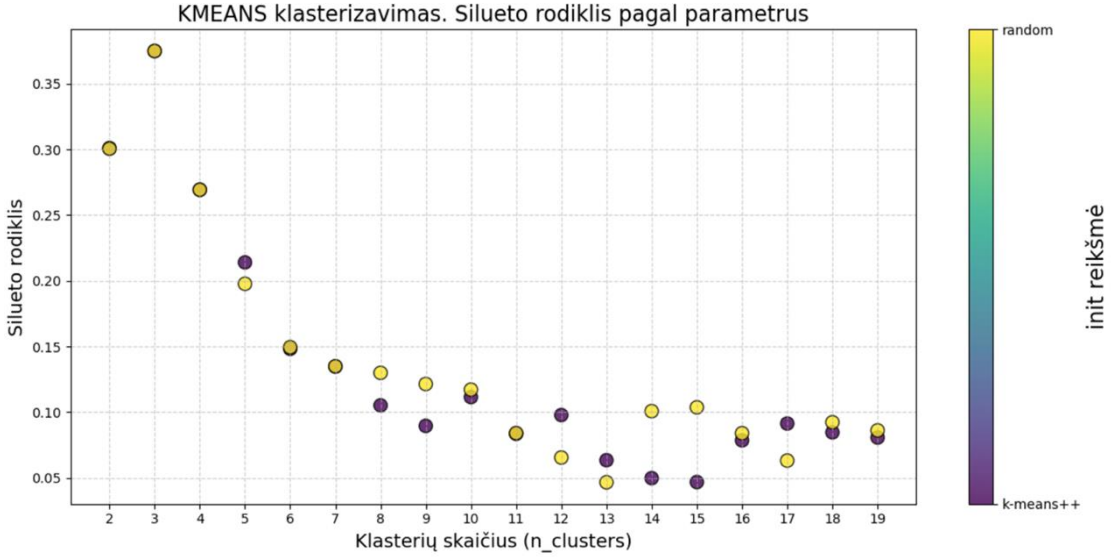
Naudojant tuos pačius optimalius parametrus(„n_clusters“ = 3; „init“ = „k-means++“) sekančiame grafike(3.7 pav.) yra matomas grafikas, kuris buvo gautas pirma atliekant „kMeans“ klasterizavimą, ir tik tada t-SNE dimensijos mažinimą rezultatų atvaizdavimui 2D ašyse. Matoma, kad klasterių pozicijos skiriasi nuo klasterizavimo, pirma darant t-SNE.

Akivaizdu, jog dimensijų mažinimas prieš klasterizavimą ar po jo turi didelę įtaką rezultatams – šiai duomenų aibei su specifiškais parametrais pirma atliekant dimensijų mažinimą, gaunamų klasterių centrų pozicijos yra tikslesnės.



3.7 pav. "kMeans" klasterizavimas prieš taikant "t-SNE" metodą.

3.1.5. Silueto rodiklis

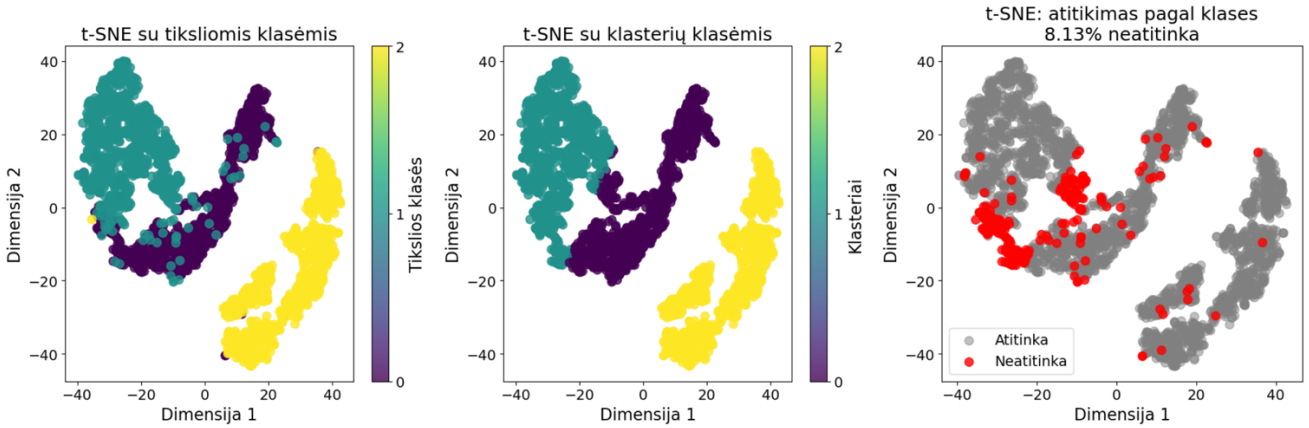


3.8 pav. Siluetų metodo rodiklis keičiantis klasterių skaičiui.

Aukščiau esančiame grafike(3.8 pav.) yra matomi „Silhouette“ algoritmo rezultatai, kai šiam algoritmui yra naudojami originalūs, nesumažintų dimensijų duomenys. Kai „n_clusters“ reikšmė lygi 3, Silueto rodiklio reikšmė yra didžiausia. Šis klasterių kiekis sutampa su „elbow“ metodu gautu klasterių kiekiu ir su vizualiai matomu klasterių kiekiu, tad galima daryti išvadą, jog šiai duomenų aibei optimalus klasterių kiekis yra 3.

Šie rezultatai, lyginant su pradžioje gautu “Silhouette” grafiku(pav. 3.1.2 skyriuj ten), yra tikslūs. Kai naudojamas silueto algoritmas nemažinant dimensijų(3.8 pav.), atstumai tarp objektų pozicijų išlieka originalūs, o tai yra svarbu šiam algoritmui.

3.1.6. Klasterizavimo tikslumas



3.9 pav. “t-SNE” dimensijų mažinimo ir “kMeans” klasterizavimo grafikai, jų persidengimo grafikas.

3.9 pav. pateikiami grafikai, kuriuose matoma „t-SNE“ dimensijų mažinimo, „kMeans“ klasterizavimo ir jų persidengimo grafikai:

Pirmas grafikas kairėje su klasėmis, gautomis iš „t-SNE“ dimensijų mažinimo algoritmo, antras grafikas viduryje vaizduoja „kMeans” klasterizavimo rezultatą ir trečias grafikas dešinėje – šių grafikų persidengimą(atitikimas – pilka spalva pažymėti objektai, neatitikimas – raudona). Matoma, jog 8.13% objektų, klasterizuotų su „kMeans“, neatitiko su klasėmis, lyginant su „t-SNE“ rezultatais.

Galima pastebėti, kurie objektai po „kMeans“ klasterizavimo neatitinka, lyginant su klasėmis iš „t-SNE“(3.16 pav.). 0 klasės objektų yra 146, 1 klasės – 97, ir 2 klasės tik 1. Tai reiškia, jog „kMeans“ klasterizavimas pasirinktais parametrais 2 klasę klasterizavo beveik tobulai, tačiau 0 ir 1 klasėse buvo neatitikimų.

```
Neatitinkančių objektų kiekis 0 klasei: 146
Neatitinkančių objektų kiekis 1 klasei: 97
Neatitinkančių objektų kiekis 2 klasei: 1
```

Pav. 3.16. Klasių pasiskirstymas klasterizavimo rezultatuose, kai klasterių objektai neatitinka klasių.

3.1.7. Išvados

Atlikus klasterizavimo analizę naudojant „kMeans“ algoritmą, nustatyta, kad optimalus klasterių skaičius mūsų pasirinktai duomenų aibei yra 3. Tokią pačią išvadą galima daryti ir panaudojus „Elbow“ metodą sumažintos dimensijos duomenų rinkiniui ir „Silhouette“ metodą nesumažintos dimensijos duomenų rinkiniui, abiejų rezultatai sutampa – optimalus klasterių skaičius yra 3. Geriausi rezultatai pasiekiami su parametrais, kai „n_clusters“ lygus 3 ir „init“ lygus „k-means++“. Likę parametrai didelės įtakos rezultatams nedarė.

Taip pat buvo pastebėta, jog naudojant „t-SNE“ algoritmu sumažintos dimensijos rezultatus klasteriai buvo aiškiai atskirti, tačiau klasterių pozicijos skiriasi priklausomai nuo to, ar „t-SNE“ buvo atliktas prieš klasterizavimą, ar po jo. Klasterizavimas parodė, jog „kMeans“ efektyviai identifikuoja pagrindines duomenų grupes, tačiau neatitikimai (apie 8.13%) su tikrosiomis klasėmis rodo, kad kai kurie taškai yra sunkiai klasifikuojami dėl jų panašumo į skirtingus klasterius(dėl klasių persidengimo). Optimalūs rezultatai buvo gauti pašalinus išskirtis ir tinkamai normalizavus duomenis, o tai sustiprino klasterių atsiskyrimą ir pagerino modelio tikslumą.

Taip pat buvo pastebėtas klasterių nestabilumas. Naudojant kitokią „seed“ reikšmę duomenų normavimo žingsnyje, atsitiktinių objektų pasirinkimo žingsnyje ar dimensijos mažinimo žingsnyje algoritmas gali klasterizuoti duomenis kitaip, tokiu atveju neatitikimo su „t-SNE“ klasėmis procentas būtų daug didesnis nei 8.13%.

3.2. „DBSCAN“ algoritmas/metodas

3.2.1. Aprašymas

DBSCAN (angl. „*Density-Based Spatial Clustering of Applications with Noise*“) yra tankiu pagrįstas klasterizavimo algoritmas, kuris naudojamas identifikuoti grupes (klasterius) duomenyse pagal jų tankį. Algoritmas yra efektyvus aptinkant įvairių formų ir dydžių klasterius, kuriuose gali neiškarto matytis atskiros taškų grupės. Taip pat, jis automatiškai nustato triukšmo taškus (angl. „outliers“). „DBSCAN“ neskiria fiksuoto skaičiaus klasterių, kaip tai daro kai kurie kiti algoritmai (pvz., K-means). Vietoj to, klasterių skaičius priklauso nuo duomenų struktūros ir jų tankumo.

„DBSCAN“ svarbiausi du parametrai yra:

- **eps (epsilon)**: atstumas, kuris nustato, kaip arti taškai turi būti vienas kito, kad jie būtų laikomi to paties klasterio dalimi.
- **min_samples**: minimalus taškų skaičius, reikalingas klasteriui suformuoti.

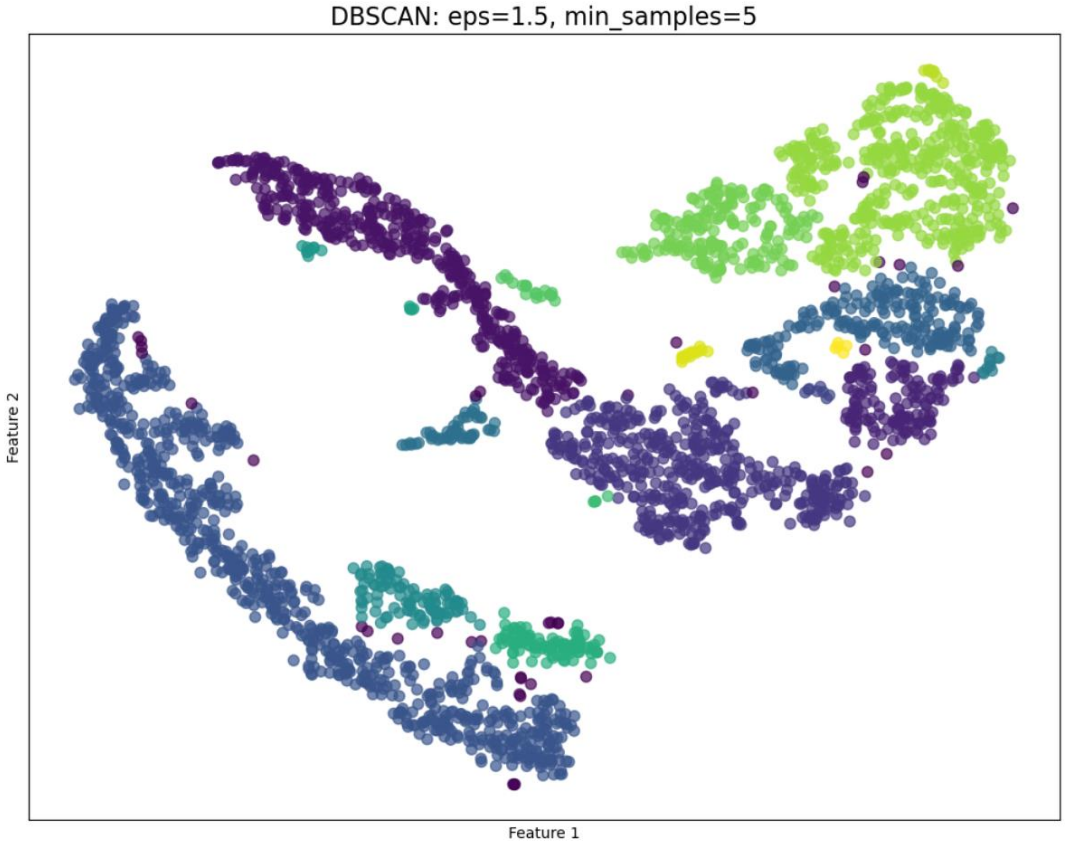
Šiuo atveju „DBSCAN“ algoritmas buvo taikytas šioms normuotoms „min-max“ metodu duomenų aibėm:

- pilnai duomenų aibei, ištrynus klasės požymį;
- duomenų aibei su atrinktais požymiais („u“, „g“, „r“, „i“, „z“, „redshift“);
- duomenų aibei su tai pačiais atrinktais požymiais, pritaikius „t-SNE“ metodą (dimensiškumo mažinimo metodą).

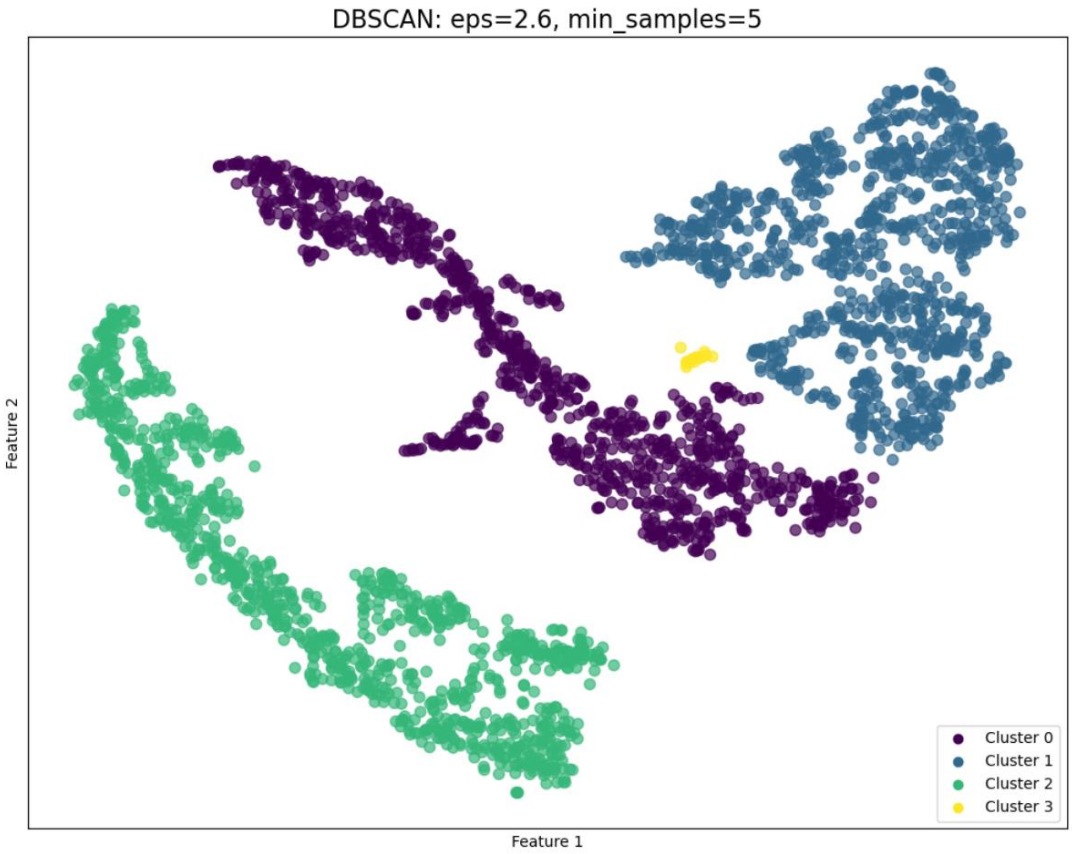
3.2.2. Algoritmo taikymas sumažintos dimensijos duomenims

„DBSCAN“ klasterizavimo algoritmas su skirtingais parametrais taikytas normuotai duomenų aibei su atrinktais požymiais („u“, „g“, „r“, „i“, „z“, „redshift“) ir „t-SNE“ metodo pagalba sumažintos dimensijos iki 2 duomenims suteikė prasmingą vizualizaciją.

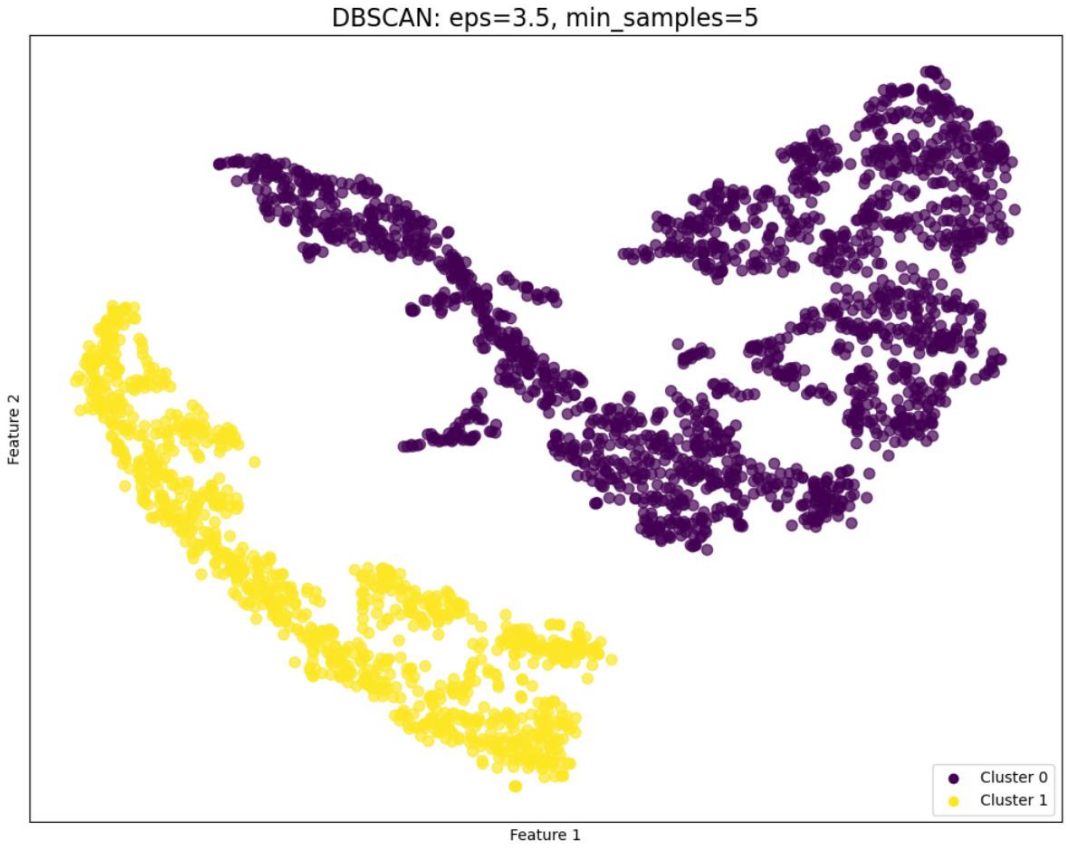
Galima pastebėti, kad ir nežymus „eps“ parametro reikšmės didėjimas ženkliai sumažina klasterių kiekį. Kai „min_samples“ parametro reikšmė išlieka ta pati, o „eps“ reikšmė lygi 1.5 (3.10 pav.) susidaro 20 klasterių (įskaitant triukšmo klasterį), padidėjus „eps“ reikšmei iki 2.6 (3.11 pav.) susidaro 4 klasteriai, o dar padidinus iki 3.5 (3.12 pav.) susidaro tik du klasteriai – kadangi „eps“ reikšmė didelė, taškai ir jų grupės turi tarpusavyje būti pakankamai toli, kad būtų jas galima atskirti į klasterius.



3.10 pav. "DBSCAN" klasterizavimas. "eps" vertė 1.5, "min_samples" vertė 5.

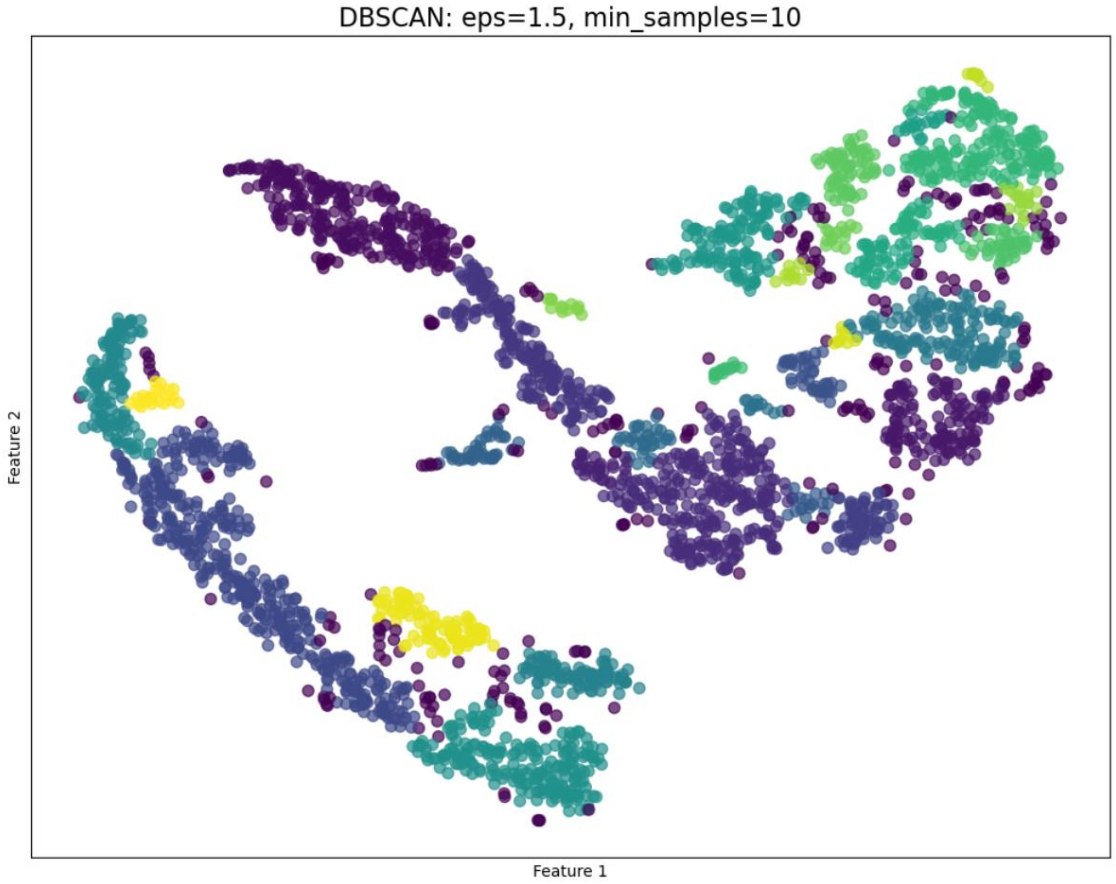


3.11 pav. "DBSCAN" klasterizavimas. "eps" vertė 2.6, "min_samples" vertė 5

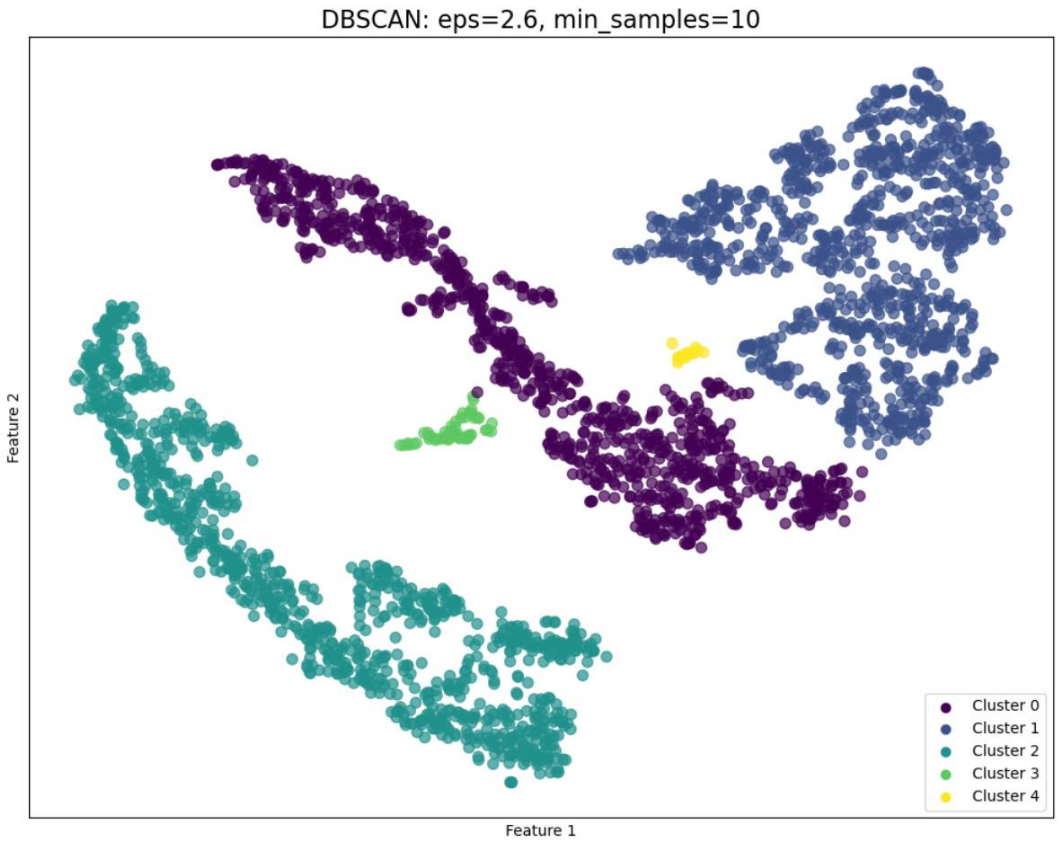


3.12 pav. "DBSCAN" klasterizavimas. "eps" vertė 3.5, "min_samples" vertė 5

Akivaizdi „min_samples“ parametro įtaka duomenims: kai „eps“ parametro reikšmė tokia pati, tačiau „min_samples“ kiekis padidėja iš 5 (3.13 pav.) į 10 (3.14 pav.), klasterių kiekis padidėja nuo 20 iki 32 klasterių (įskaitant triukšmo klasterį). Tą patį galima pastebėti ir kai „eps“ reikšmė lygi 2.6, o „min_samples“ kiekis padidėja dvigubai (3.11 pav. ir 3.14 pav.) – klasterių kiekis irgi padidėja, bet tik iš keturių į penkis klasterius.



3.13 pav. "DBSCAN" klasterizavimas. "eps" vertė 1.5, "min_samples" vertė 10



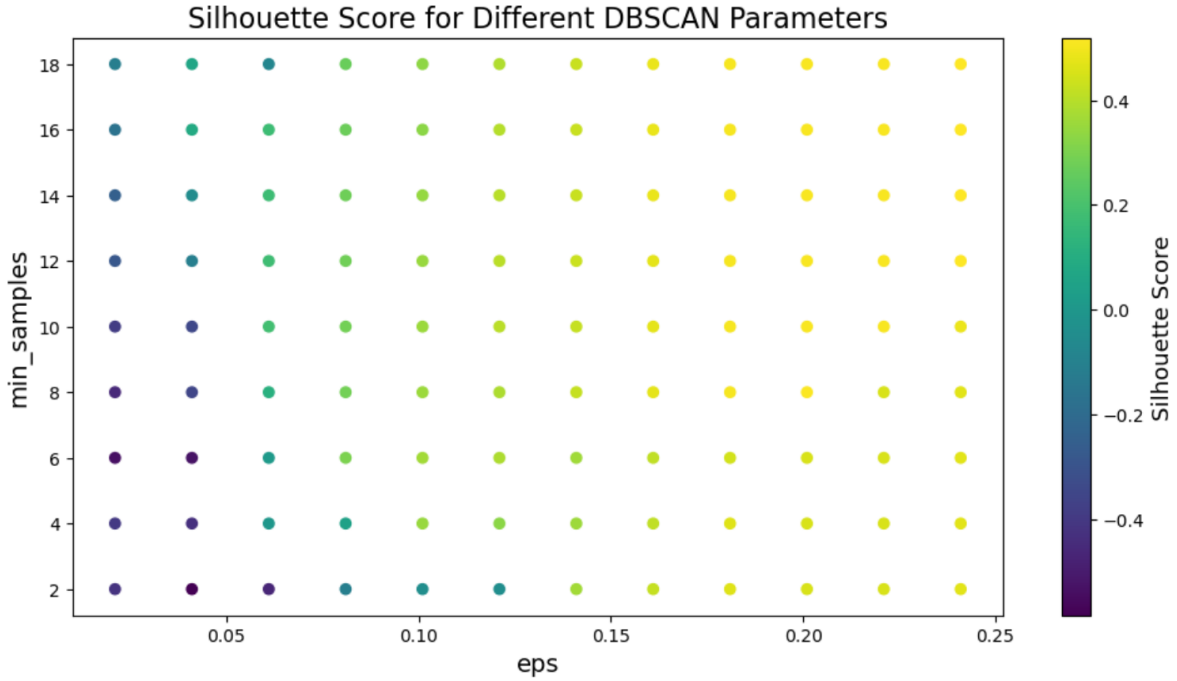
3.14 pav. "DBSCAN" klasterizavimas. "eps" vertė 2.6, "min_samples" vertė 10

Taigi, „DBSCAN“ algoritmo parametrų: didesnės „eps“ reikšmės lems mažiau klasterių, o didesnės „min_sample“ reikšmės lems daugiau klasterių, reikia atrasti tinkamą balansą tarp šių reikšmių, norint, kad algoritmas veiktų teisingai.

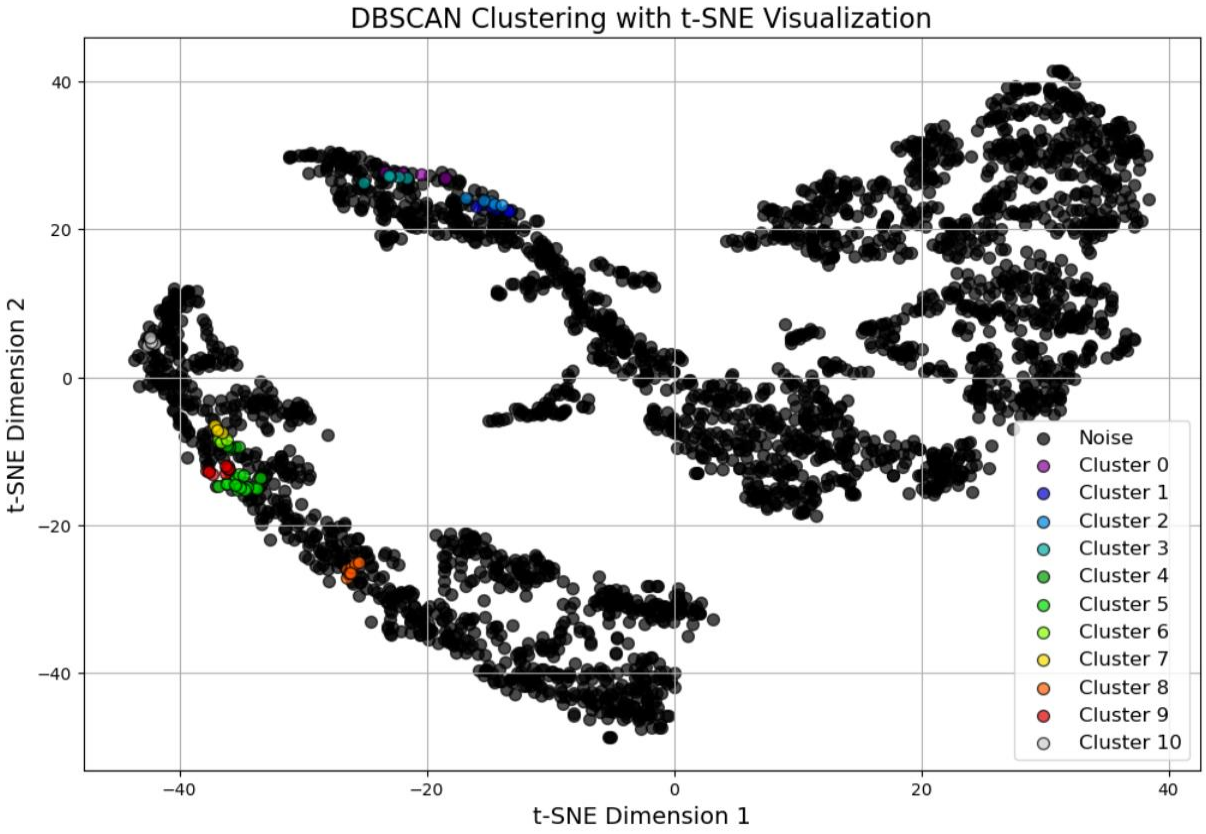
3.2.3. Algoritmo taikymas normuotoms duomenų aibėms

„DBSCAN“ algoritmo rezultatas normuotai duomenų aibei, neatrinkus pagrindinių požymių, bet išmetus visiškai nereikšmingus požymius ir klasės požymį, neduoda jokios informacijos. Šis algoritmas pilnai duomenų aibei buvo taikytas su įvairiausiais parametrais [..., ...]. Visada gaunamas tas pats rezultatas – vienas didelis klasteris.

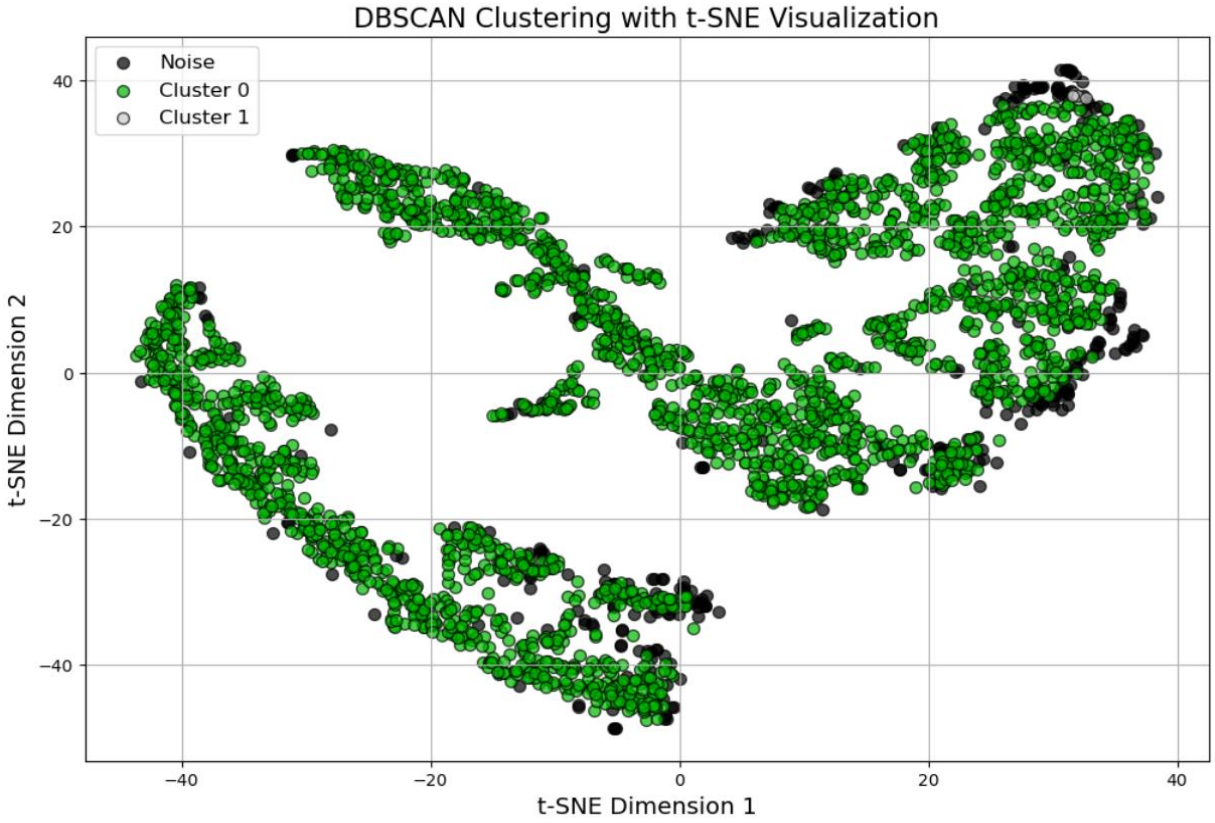
„DBSCAN“ algoritmas taip pat buvo bandytas taikyti normuotai duomenų aibei atrinktiems požymiams („z“, „i“, „r“). Šį kartą algoritmas suklasterizavo duomenų aibę, tačiau rezultatai neduoda jokios naudingos informacijos. Tai galime spręsti tiek iš žemiau esančių grafikų, tiek iš Siluetų metodo rodiklių (pav. XX). Nors aukštesni rodikliai, turėtų rodyti geresnius grafikus, tačiau pritaikius „t-SNE“ metodą, lengvesniam klasterizuotų duomenų atvaizdavimui, „DBSCAN“ algoritmo parametrai su aukštesnėmis Silueto rodiklio vertėmis rodo prastesnius grafikus. Taip pat matome, kad aukščiausios Silueto rodiklio vertės siekia vos daugiau negu 0.1, o tai yra labai nedaug. Šiuo atveju, geriausiai grafikai atrodo esant mažiausioms „Silhouette“ rodiklio vertėms, t.y. kai jos yra apie -0.6.



3.15 pav. "Silhouette" metodo reikšmių priklausomybės nuo "min_samples" ir "eps" parametrų reikšmių taškinė diagrama



3.16 pav. "DBSCAN" klasterizavimas. "eps" vertė 0.01, "min_samples" vertė 5



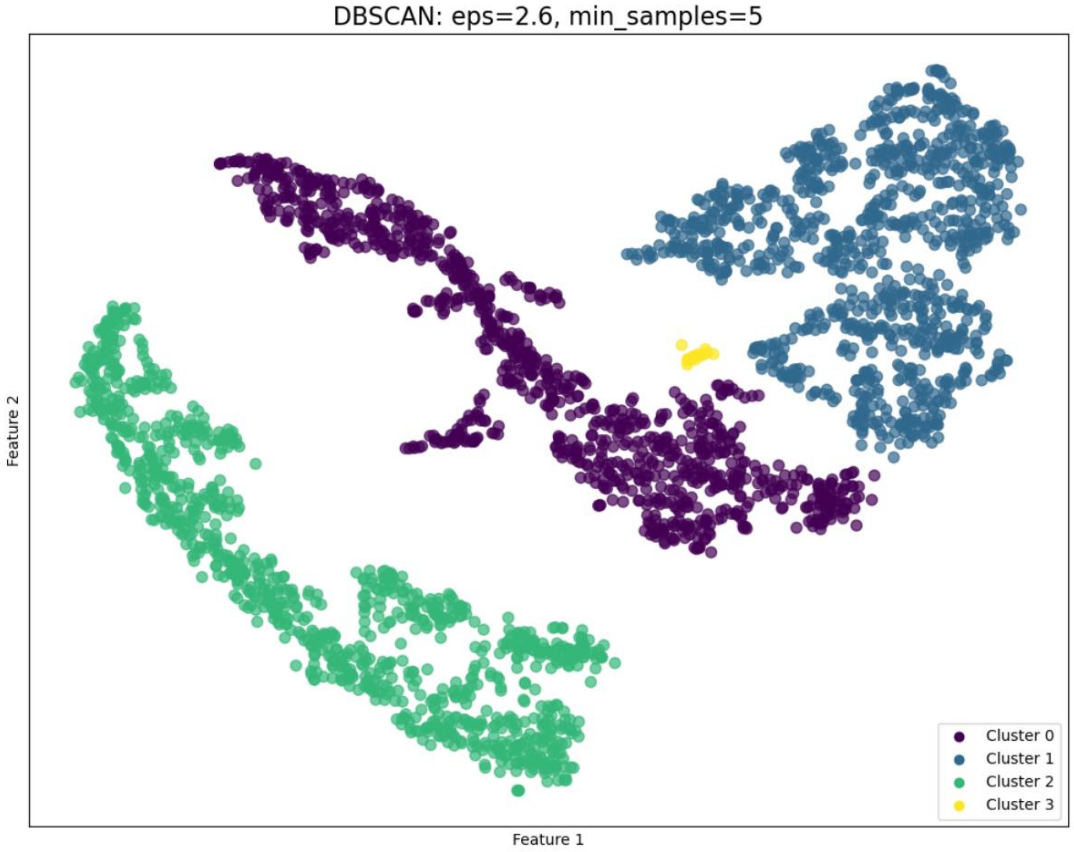
3.17 pav. "DBSCAN" klasterizavimas. "eps" vertė 0.05, "min_samples" vertė 5

Akivaizdu, kad pirmu atveju gaunamas rezultatas (3.16 pav.), kai „eps“ vertė lyg 0.01, o „min_samples“ vertė lygi 5, turi ženkliai per daug klasterių, esančių sąlyginai mažame plote, ir didžioji dalis objektų (net 2914 objektų) buvo priskirti triukšmui. Antroje diagramoje (3.17 pav.), kai „eps“ vertė lyg 0.05, o „min_samples“ vertė lygi 5, matome nemažą kiekį objektų priskirtų triukšmui (317), tačiau beveik visi objektai (2670) yra nuspalvinti žaliai, t.y. priskirti klasteriui „Cluster 0“, ir beveik nėra objektų priskirtų „Cluster 1“ (tik 8 objektai).

Taigi, galime daryti išvadas, kad “DBSCAN” klasterizavimo algoritmo taikymas šiai duomenų aibei yra netikslingas. Negauname jokių rezultatų su visus požymius turinčia duomenų aibe (visi objektai priklauso vienai aibei), ir gauname jokios naudingos informacijos neduodančius rezultatus su duomenų aibe, kurioje yra atrinkti požymiai - daugiau nei 85% objektų yra priskirti triukšmui arba vienai duomenų aibei.

3.2.4. Klasterizavimo tikslumas

Palyginus pradinės duomenų aibės klases ir „DBSCAN“ priskirtas etiketes („labels“) galime palyginti kokia dalis taškų buvo nustatyta teisingai. Rezultatų palyginimui buvo naudotas optimaliai suklasterizuoti duomenys, kai „DBSCAN“ parametrai „eps“ yra lygus 2.6, o „min_samples“ lygus 5.



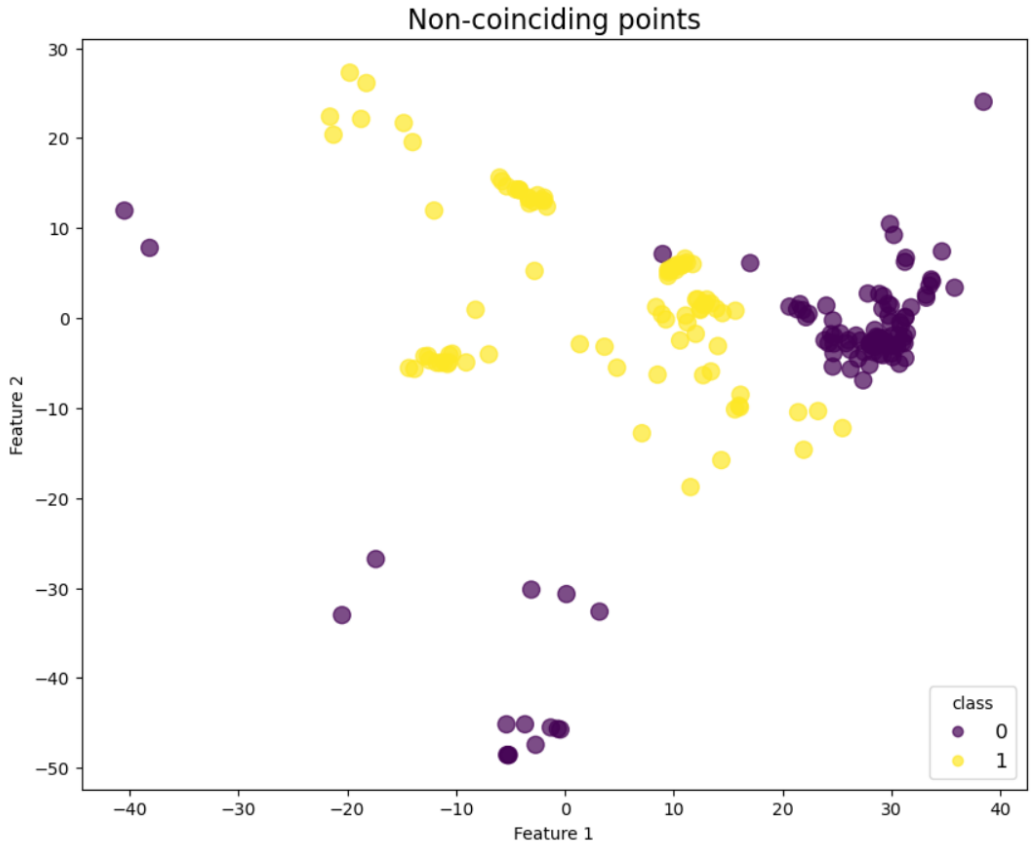
3.18 pav. optimaliai suklasterizuoti naudojantis „DBSCAN“

„Cluster 0“ atitinka klasę 0 (žvaigždė), „Cluster 1” atitinka klasę 0 (galaktika) ir „Cluster 2” atitinka klasę 2 (kvazaras), „Cluster 3“ neatitinka jokios klasės (3.18 pav.).

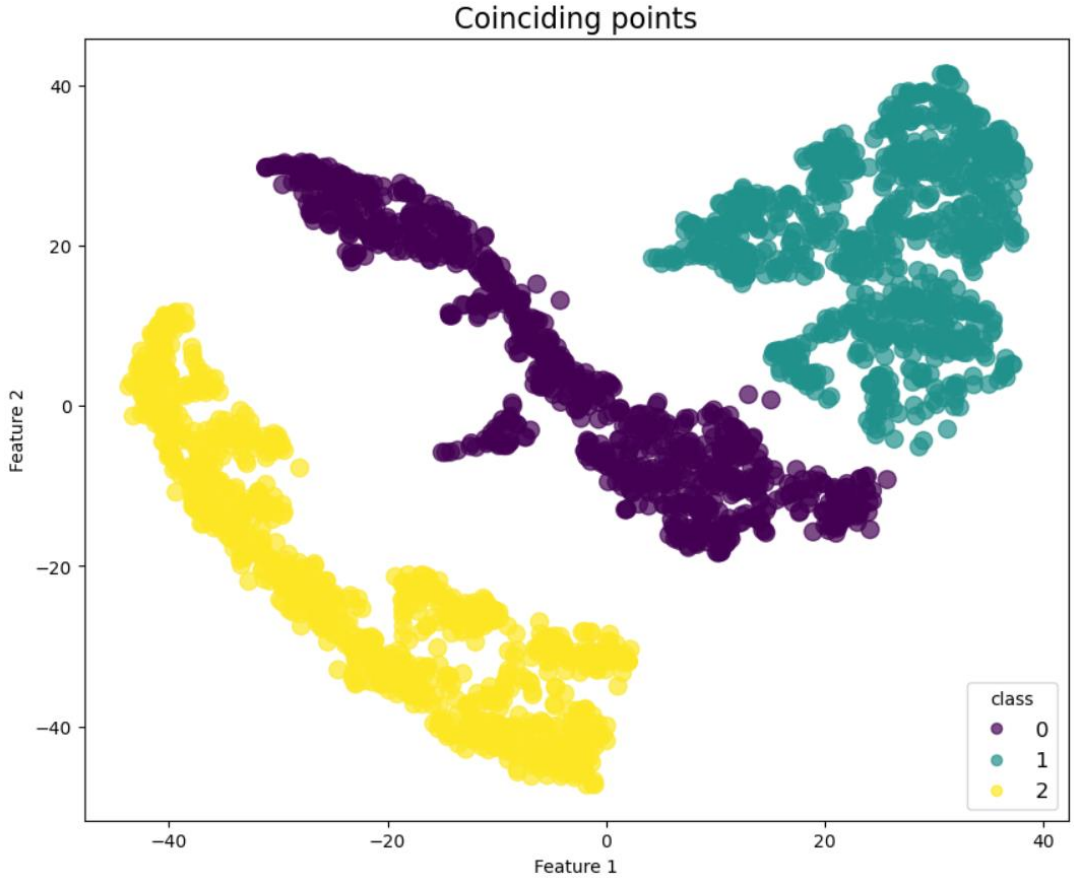
Atlikus skaičiavimus buvo rasta, kad „DBSCAN“ neteisingai suklasterizavo 188 objektus, t.y. ~6.26% visų atrinktų objektų buvo priskirta klaidinga klasė. Galima teigti, kad šio algoritmo, kai jo parametrai „eps“ yra lygus 2.6, o „min_samples“ kiekis yra lygus 5, tikslumas yra 93.74%.

Šiame grafike, matome taškus, kurie buvo priskirti ne tai klasei (grafike taškai nuspalvinti ta spalva, kuriai klasei jie turėtų priklausyti pagal originalią duomenų aibę, o „DBSCAN“ algoritmo buvo priskirti ne tai klasei).

Galima pastebėti, kad „DBSCAN“ algoritmas daliai objektų priskyrė kitokias etiketes, negu yra to objekto realios klasės. Viena iš to priežasčių yra ir ne 100% tikslus „tSNE“ algoritmo – apatiniai violetiniai taškai (3.19 pav.), turėtų priklausyti klasei 0, tačiau ir „tSNE“ vizualizacijoje, yra matyti, kad šie taškai priskirti pirmos klasės grupei. Todėl pačio „DBSCAN“ algoritmo tikslumas galimai būtų didesnis, jei visi taškai būtų teisingiau sugrupuoti „tSNE“ algoritmo.



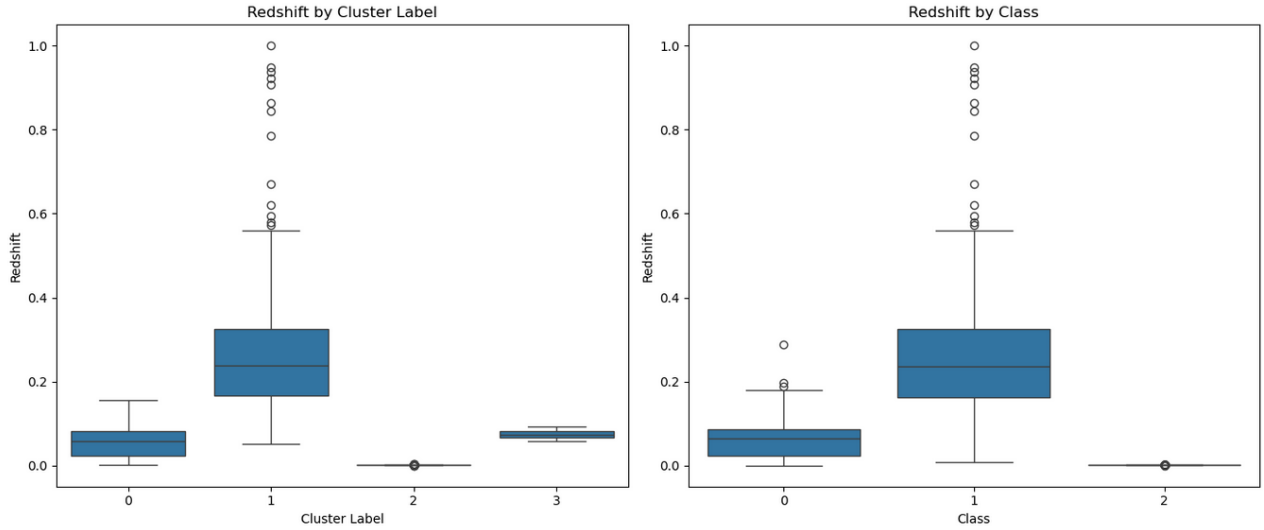
3.19 pav. "DBSCAN" suklasterizuoti objektai, kurių etiketės nesutampa su objektų klasėmis. Objektai nuspalvinti pagal jų klases.



3.20 pav. "DBSCAN" suklasterizuoti objektai, kurių etiketės sutampa su objektų klasėmis. Objektai nuspalvinti pagal jų klases.

Tačiau yra matomas didelis ir aiškus objektų grupių atsiskyrimas žiūrint tik į teisingai atskirtus taškus žemiau (3.20 pav.)

Klasterizuotų ir neklasterizuotų duomenų aibės, jų aprašomosios statistikas pagal klases ir jas atitinkančias etiketes (angl. „label“) yra labai panašios, nes neatitinkančių taškų kiekis yra mažas ir pakankamai pasiskirstęs tarp klasių. Kaip pavyzdį, galime palyginti „redshift“ požymį stačiakampėmis diagramomis (3.21 pav.). Akivaizdu, kad „redshift“ reikšmės ir jų pasiskirstymas tarp skirtingų klasių yra beveik toks pat kaip ir tarp skirtingų klasterių etikečių. Vieninteliai du aiškiai matomi skirtumai, yra ketvirtos etiketės atsiradimas, tačiau jai priklauso tik 15 taškų ir jos reikšmių vidurkis yra arti visų taškų reikšmių vidurkio. Taip pat klasė 0, po klasterizavimo nebeteko išskirčių (angl. „outliers“). Labai panašią informaciją gausime ir lyginant kitus aktualius požymius („z“, „i“, „r“, „g“, „u“).

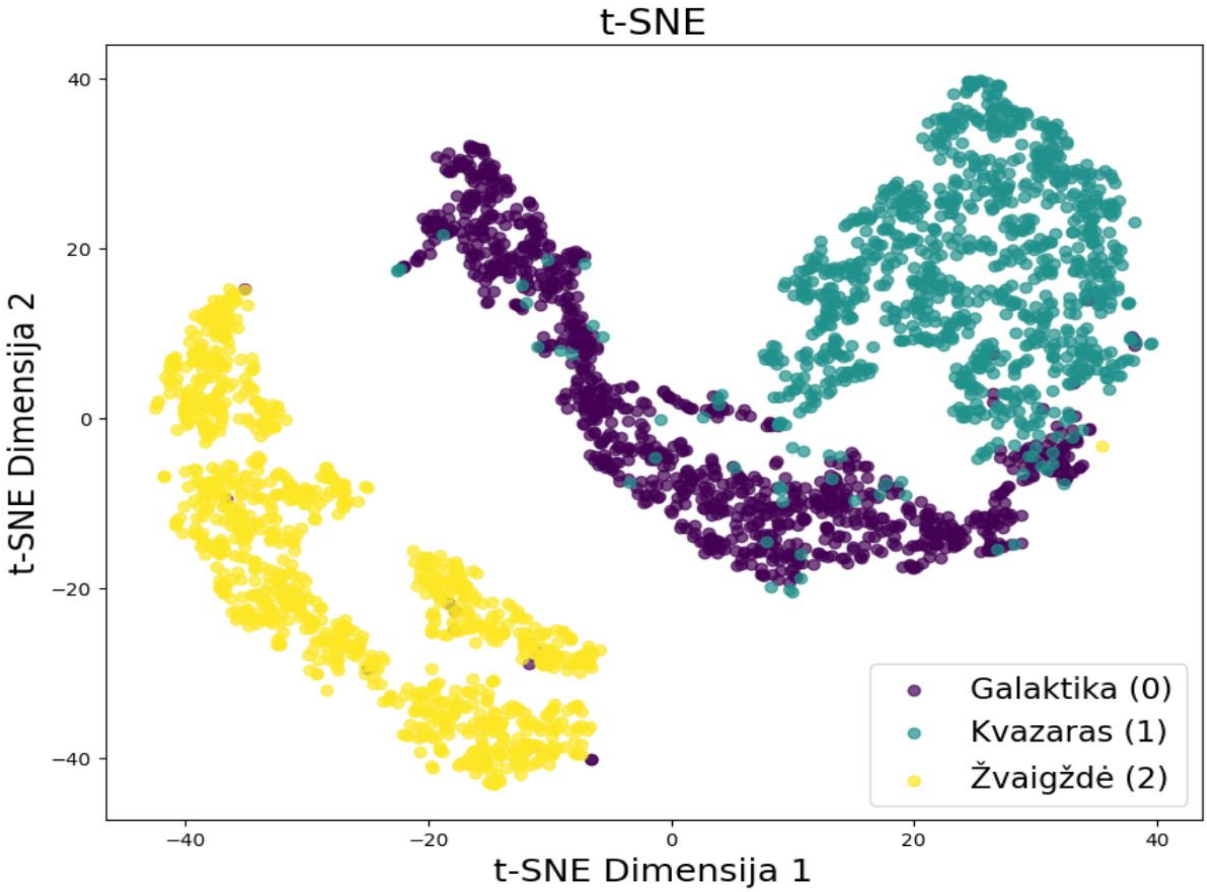


3.21 pav. stačiakampės diagramos lyginančios „redshift“ reikšmių pasiskirstymą tarp priskirtų etikečių ir originalių klasių.

Taigi, tiksliausiai veikiančio „DBSCAN“ algoritmo parametrai „eps“ ir „min_samples“ yra lygūs atitinkamai 2.6 ir 5. Jis teisingai suklasterizuoja 93.74% visų šios duomenų aibės objektų. Dalis šios aibės objektų (15 objektų) buvo priskirti neegzistuojančiai naujai klasei, likę objektai buvo priskirti klaidingai klasei (173 objektai).

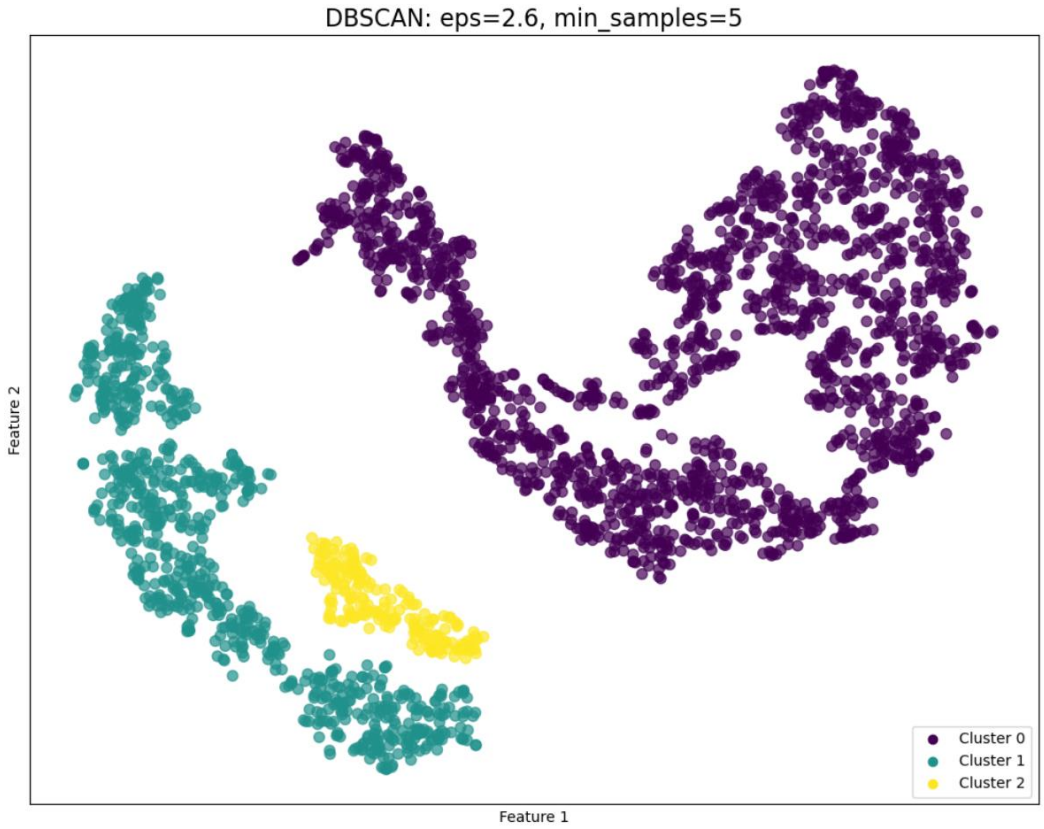
3.2.5. Algoritmo jautrumas mažiems duomenų aibės pokyčiams

Štai čia matome „t-SNE“ algoritmo pritaikymą ir vizualizaciją šiek tiek kitokiems duomenims: buvo paimti po 1000 atsitiktinių objektų iš kiekvienos klasės, tačiau naujo parametro „random_seed“ reikšmė lygi 2 , todėl buvo paimti šiek tiek kitokie objektai ir naujai duomenų aibei pritaikius „t-SNE“ metodą su tokiais pat parametrais („perplexity“ lygi 50, o „max_iter“ lygus 750) vizualizacija nežymiai skiriasi (3.22 pav.).

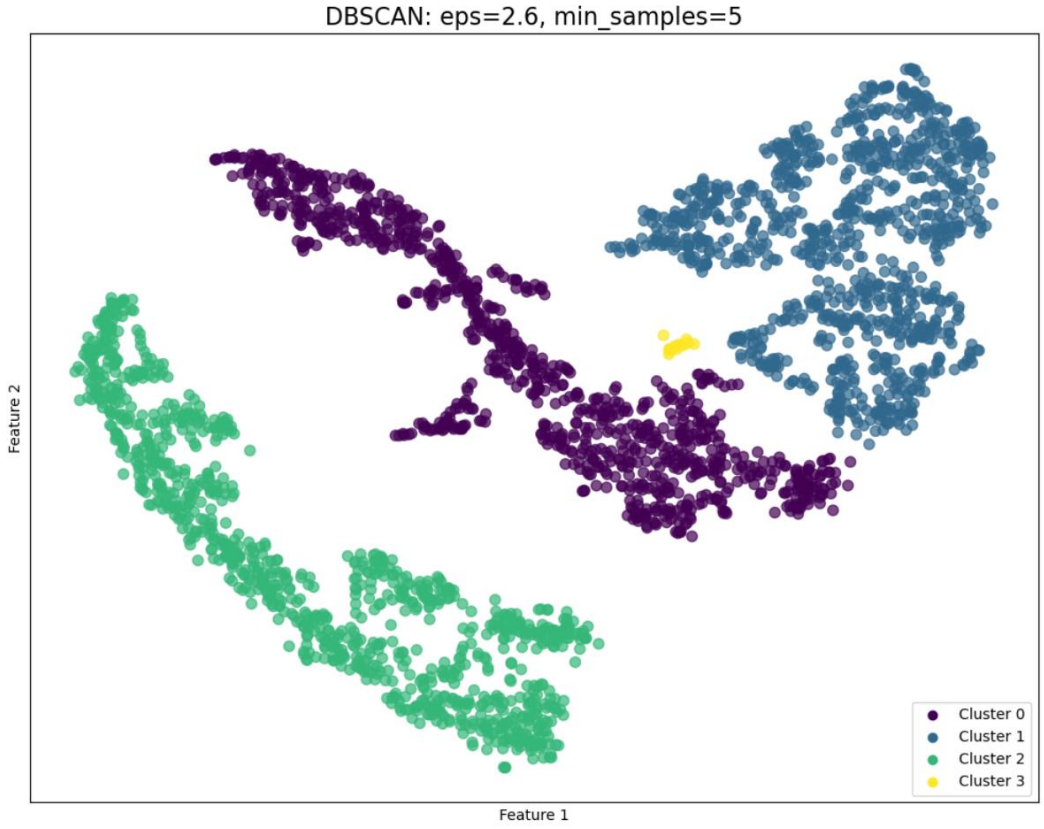


3.22 pav. "t-SNE" metodo taikymas atsitiktinai atrinktiems duomenims, kai "random_seed" reikšmė yra lygi 2.

Palyginus naujos ir buvusios duomenų aibių klasterizavimą „DBSCAN“ algoritmu su sąlyginai geriausiais parametrais (išrinkus iš poskyryje „3.1.2” vaizduotų grafikų optimalaus grafiko parametrus), prieš tai pritaikius „t-SNE“ dimensijų mažinimo metodą, matome didelius skirtumus grafikuose (3.23 pav. ir 3.24 pav.)



3.23 pav. "DBSCAN" klasterizavimas atsitiktinai atrinktiems duomenims, kai "random_seed" reikšmė lygi 2. "eps" vertė 2.6, "min_samples" vertė 5.



3.24 pav. "DBSCAN" klasterizavimas atsitiktinai atrinktiems duomenims, kai "random_seed" reikšmė lygi 42. "eps" vertė 2.6, "min_samples" vertė 5.

Naudojant tuos pačius parametrus panašiai duomenų aibei, galime gauti labai skirtingus rezultatus. Šiuo atveju, visus taškus, priklausančius 0 ir 1 klasėms, atitinkančius klasterius sujungė į vieną klasterį, o dalį 2 klasę atitinkančio klasterio taškų priskyrė naujam klasteriui. Viso klaidingai priskirta 1194 objektai. Šiame pavyzdyje labai aiškiai galime pamatyti, kad „DBSCAN“ algoritmas objektus jungia į klasterius pagal jų tarpusavio tankį ir nežymus pokytis tarp objektų atstumų, šiuo atveju sumažino tikslumą nuo 94.3% iki vos 60.2%.

Taigi, galima teigti, kad kiekvienai duomenų aibei gali tekti ieškoti ir taikyti naujus „DBSCAN“ parametrus, net jei iš pirmo žvilgsnio duomenų aibių aprašomoji statistika yra panaši, abi aibės turi po tiek pat objektų ir jų vizualizacijos atrodo panašiai.

3.2.6. Išvados

Nustatant šiuos parametrus buvo pastebėta, kad didėjančios „eps“ reikšmės mažina klasterių kiekį, nes vis daugiau taškų priskiria tam pačiam klasteriui, o didėjančios „min_samples“ reikšmės didina klasterių kiekį.

Taip pat buvo pastebėta, kad nežymus pokytis objektų aibėje lemia truputi kitokį taškų pasiskirstymą panaudojus dimensijos mažinimo metodą, ir dėl didesnio mažo kiekio taškų tankio „DBSCAN“ algoritmas su tais pačiais parametrais veikia stebėtinai prasčiau - visiškai nebeatskiria taškų priklausančių dviem klasėm.

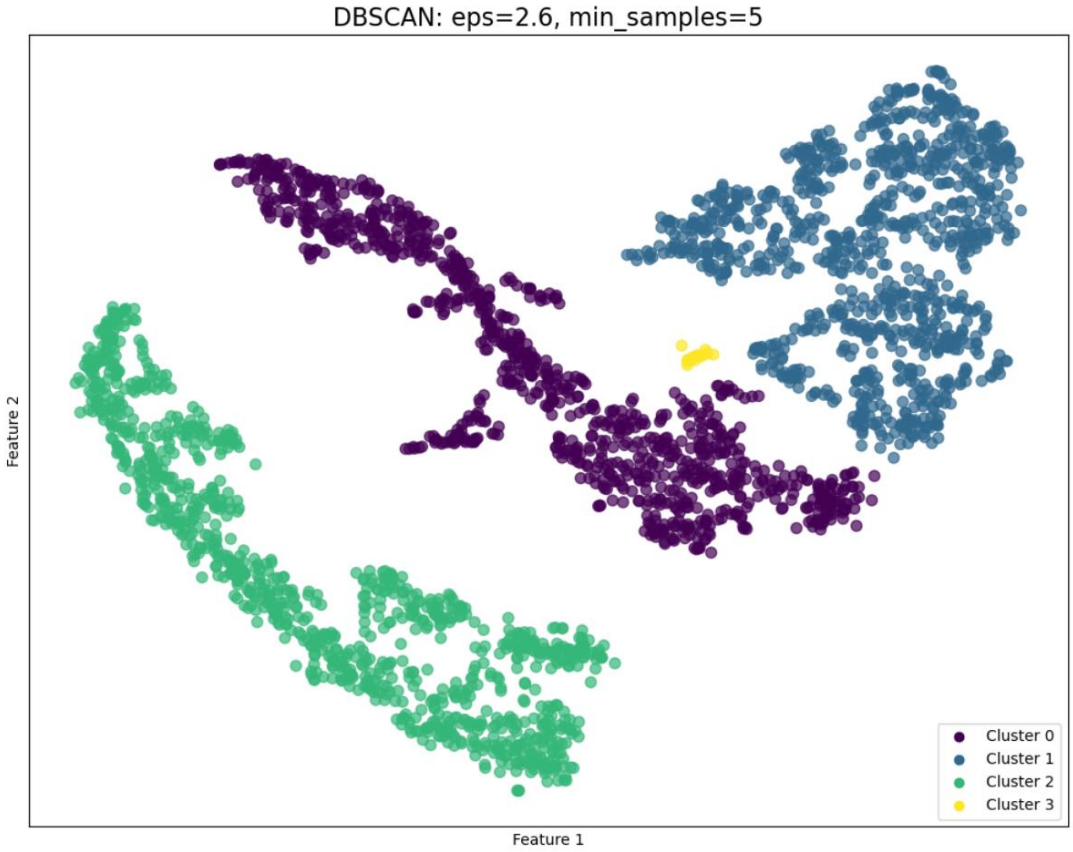
Geriausias „DBSCAN“ grafikas, kuris optimaliai priskiria objektus klasteriam yra pritaikytas sumažintos dimensijos „t-SNE“ metodo pagalba duomenim. Šiuo atveju parametrai „eps“ yra lygus 2.6, „min_samples“ yra lygus 5 ir algoritmas teisingai suklasterizavo 93.4% objektų.

4. IŠVADOS

Atlikus klasterizavimo analizę naudojant „kMeans“ ir „DBSCAN“ metodus, įvertinta, kad kiekvienas algoritmas turi savo privalumų ir trūkumų, priklausomai nuo duomenų struktūros ir pasirinktų parametrų. Naudojant „kMeans“, optimalus klasterių skaičius buvo nustatytas kaip 3, o geriausi rezultatai pasiekti su parametrais „n_clusters“ = 3 ir „init“ = „k-means++“. Duomenų normalizavimas ir išskirčių pašalinimas ženkliai pagerino klasterių atskyrimą ir buvo pasiektas 91.87% tikslumas.

Naudojant „DBSCAN“ algoritmą, buvo nustatyta, kad parametrų „eps“ ir „min_samples“ reikšmės stipriai veikia klasterių susidarymą: didesnės „eps“ reikšmės sumažina klasterių skaičių, o didesnės „min_samples“ reikšmės jį padidina. Išanalizuota, kad algoritmas veikė geriausiai, kai buvo pritaikytas „t-SNE“ metodu sumažintiems duomenims, ir su optimaliais parametrais „eps“ = 2.6, „min_samples“ = 5 buvo pasiektas 93.4% tikslumas. Tačiau pastebėta, kad mažas duomenų pokytis ar nedideli taškų tankio skirtumai gali reikšmingai paveikti klasterizacijos rezultatus, tai rodo klasterių nestabilumą. Tačiau visais atvejais „DBSCAN“ algoritmas sumažintos dimensijos duomenyse atskyrė objektus priklausančius „Cluster 2“ (4.1 pav.) objektų grupei nuo likusių objektų.

Apibendrinant, eksperimentų rezultatai rodo, kad atitaikius parametrus abu algoritmai pakankamai tiksliai suklastetrizuoja objektus į grupes atitinkančias objektų klases, tačiau „DBSCAN“ algoritmas šiai duomenų aibei (4.1 pav.) buvo nežymiai tikslesnis – 93.4%. Taip pat, nustatytas svyruojantis klasterių stabilumas – „Cluster 2“ objektų grupė visada atskiriama taikant „DBSCAN“ metodą, tačiau ne visada naudojant „kMeans“ metodą.



4.1 pav. Sumažintos dimensijos, optimaliai suklasterizuoti duomenys naudojantis „DBSCAN“

ŠALTINIAI

- https://scikit-learn.org/stable/modules/clustering.html
- https://machinelearningmastery.com/clustering-algorithms-with-python/
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html
- https://scikit-learn.org/1.5/modules/generated/sklearn.cluster.KMeans.html
- https://pandas.pydata.org/docs/
- https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss1

KODAS

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

# importing libraries

# libraries for file manipulation
import pandas as pd
import numpy as np

# libraries for easier visualisation
import matplotlib.pyplot as plt
import matplotlib.colors as mcolors
import seaborn as sns

# libraries for dimensionality reduction
import scipy.stats as stats
from sklearn import manifold
from sklearn.decomposition import TruncatedSVD
from sklearn.cluster import KMeans, DBSCAN, HDBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

#from sklearn import metrics
from matplotlib import cm
# setting options
pd.set_option('display.max_columns', None)
pd.set_option('float_format', '{:f}'.format)

# In[2]:

# importing dataset
df_orig = pd.read_csv("star_classification.csv", delimiter=',')

# ## Preparing data

# In[3]:

df_orig.drop(columns=['obj_ID', 'alpha', 'delta', 'spec_obj_ID', 'rerun_ID', 'MJD'],
inplace=True)

# In[4]:

# panaikinti ekstremalių atsiskyrėlių vieną eilutę, kurioje reikšmės yra -9999
df_orig = df_orig[(df_orig[['u', 'g', 'r', 'i', 'z']] != -9999).all(axis=1)]

# encoding labels
df_orig.replace(['GALAXY', 'QSO', 'STAR'], [0, 1, 2], inplace=True)

# Set a random seed for reproducibility
# random_seed = 42 for DBSCAN reproduction, 2 for kMeans reproduction
# random_seed = 42
random_seed = 2
# Sample 1,000 instances per class with a fixed seed
#df = df_orig.groupby('class', group_keys=False).apply(lambda x:
x.sample(n=1000)).reset_index(drop=True)
df = df_orig.groupby('class', group_keys=False).apply(lambda x: x.sample(n=1000,
random_state=random_seed)).reset_index(drop=True)

# kiti masyvai nebuvo normalizuot, nes jie buvo pavadinimai, kampo laipsniai,
kategorijos ar ID
normalization_cols = ['redshift', 'u', 'g', 'r', 'i', 'z']
```

```
# normavimas
dfminmax = df.copy()
for col in normalization_cols:

    #min-max normalization
    dfminmax[col] = (dfminmax[col] - dfminmax[col].min()) / (dfminmax[col].max() -
dfminmax[col].min())

# In[5]:

dfminmax.columns

# In[6]:

feature_cols = ['redshift', 'u', 'g', 'r', 'i', 'z']
no_class_col = ['u', 'g', 'r', 'i', 'z', 'run_ID', 'cam_col', 'field_ID', 'redshift',
'plate', 'fiber_ID']
data = dfminmax[feature_cols].values
data_full = dfminmax[no_class_col].values

# ## Dimensionallity reduction tSNE

# In[7]:

tsne = manifold.TSNE(n_components=2,
    perplexity=50,
    n_iter=750,
    metric='canberra',
    random_state=42)
data_tsne = tsne.fit_transform(dfminmax[feature_cols].values)

plt.figure(figsize=(10, 8))  # Set the figure size

class_labels = ["Galaktika (0)", "Kvazaras (1)", "Žvaigždė (2)"]
class_values = [0, 1, 2]
colors = cm.viridis(np.linspace(0, 1, len(class_values)))

for val, label, color in zip(class_values, class_labels, colors):
    class_mask = dfminmax['class'] == val
    plt.scatter(data_tsne[class_mask, 0], data_tsne[class_mask, 1], color=color,
label=label, alpha=0.7)

plt.title("t-SNE", fontsize=20)
plt.xlabel('t-SNE Dimensija 1', fontsize=18)
plt.ylabel('t-SNE Dimensija 2', fontsize=18)
plt.legend(loc="lower right", fontsize=16)

# Show the plot
plt.show()

# ## Clustering

# ### DBSCAN

# #### Defining functions

# In[127]:

def dbscan_plot_one(X, params):
    # Initialize and fit DBSCAN with the provided parameters
    dbscan = DBSCAN(**params)
    labels = dbscan.fit_predict(X)

    # Plot the clusters
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7)
```

```
    # Set title
    plt.title(f"DBSCAN: eps={params['eps']}, min_samples={params['min_samples']}",
fontsize=16)

    # Add legend if there are fewer than 10 unique labels
    unique_labels = np.unique(labels)
    if len(unique_labels) < 10:
        labels_list = []
        for label in unique_labels:
            if label == -1:
                # Outliers in grey
                labels_list.append(plt.scatter([], [], color='grey', label='Outlier'))
            else:
                # Use the colormap for cluster labels
                color = scatter.cmap(label / (max(unique_labels) if max(unique_labels)
> 0 else 1))
                labels_list.append(plt.scatter([], [], color=color, label=f'Cluster
{label}'))

        # Add legend with labels
        plt.legend(handles=labels_list, loc='lower right')

    # Set axis labels and remove ticks
    plt.xlabel('Feature 1')
    plt.ylabel('Feature 2')
    plt.xticks([])
    plt.yticks([])

    plt.tight_layout()
    plt.show()

    return labels

def dbscan_plot_2d(X, parameters):
    # Create a figure with six subplots arranged vertically
    fig, axes = plt.subplots(2, 3, figsize=(20, 12))
    axes = axes.flatten()

    # Loop through each set of parameters and corresponding axis
    for ax, params in zip(axes, parameters):
        # Initialize and fit DBSCAN with the current parameters
        dbscan = DBSCAN(**params)
        labels = dbscan.fit_predict(X)

        # Plot the clusters
        scatter = ax.scatter(
            X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7
        )

        # Set title
        ax.set_title(
            f"DBSCAN: eps={params['eps']}, min_samples={params['min_samples']}",
            fontsize=12
        )

        # Add legends to each subplot if there are fewer than 10 unique labels
        unique_labels = np.unique(labels)
        if len(unique_labels) < 10:
            # I cant create labels for legend globally somewhy, so I need to add to
the list of labels each time seperately
            labels_list = []
            for label in unique_labels:
                if label == -1:
                    # Outliers in grey
```

```
                    labels_list.append(ax.scatter([], [], color='grey',
label='Outlier'))
                else:
                    # Use the colormap for cluster labels
                    color = scatter.cmap(label / (max(unique_labels) if
max(unique_labels) > 0 else 1))
                    labels_list.append(ax.scatter([], [], color=color, label=f'Cluster
{label}'))

            # Add legend with labels
            ax.legend(handles=labels_list, loc='lower right')

        # Set axis labels and remove ticks
        ax.set_xlabel('Feature 1')
        ax.set_ylabel('Feature 2')
        ax.set_xticks([])
        ax.set_yticks([])

    plt.tight_layout()
    plt.show()

def dbscan_tsne_plot(X, dbscan_params=None):

    # run DBSCAN
    dbscan = DBSCAN(**dbscan_params)
    labels = dbscan.fit_predict(X)

    # reduce data to 2D using tSNE
    tsne = manifold.TSNE(n_components=2,
                         perplexity=50,
                         n_iter=750,
                         metric='canberra',
                         random_state=42)
    data_tsne = tsne.fit_transform(X)

    # plot the data
    plt.figure(figsize=(12, 8))

    # Number of clusters in labels, ignoring noise if present (-1 label)
    unique_labels = set(labels)
    n_clusters = len(unique_labels) - (1 if -1 in labels else 0)

    # Generate colors for the clusters
    colors = cm.nipy_spectral(np.linspace(0, 1, n_clusters))

    # Plot each cluster
    for k, col in zip(sorted(unique_labels), colors):
        if k == -1:
            # Black color for noise
            col = [0, 0, 0, 1]
            label_name = 'Noise'
        else:
            label_name = f'Cluster {k}'

        class_member_mask = (labels == k)
        xy = data_tsne[class_member_mask]

        plt.scatter(
            xy[:, 0],
            xy[:, 1],
            c=[col],
            label=label_name,
            edgecolors='k',
            alpha=0.7,
            s=50
        )
```

```
    plt.title('DBSCAN Clustering with t-SNE Visualization', fontsize=16)
    plt.xlabel('t-SNE Dimension 1', fontsize=14)
    plt.ylabel('t-SNE Dimension 2', fontsize=14)
    plt.legend(loc='best', fontsize=12)
    plt.grid(True)
    plt.show()

    return labels

# #### DBSCAN for reduced data

# In[128]:

# params for tSNE data
dbscan_params = [
    {'eps': 1.5, 'min_samples': 5},
    {'eps': 2.6, 'min_samples': 5},
    {'eps': 3.5, 'min_samples': 5}, ## geras
    {'eps': 1.5, 'min_samples': 10},
    {'eps': 2.6, 'min_samples': 10},
    {'eps': 3.5, 'min_samples': 10}
]

dbscan_plot_2d(data_tsne, dbscan_params)

# In[129]:

label_list = []
for param in dbscan_params:
    labels = dbscan_plot_one(data_tsne, param)
    label_list.append(labels)

# In[130]:

for label in label_list:
    print(np.unique(label))

# #### DBSCAN for not reduced data (cia oof grafikai xd)

# In[131]:

labels = dbscan_plot_one(data_tsne, {'eps': 2.1, 'min_samples': 5, 'metric':
'cosine'})

# In[132]:

dbscan_params = {
    'eps': 0.01,
    'min_samples': 5
}

labels = dbscan_tsne_plot(data, dbscan_params)

# In[133]:

dbscan_params = {
    'eps': 0.25,
    'min_samples': 18
}

labels2 = dbscan_tsne_plot(data, dbscan_params)

# In[134]:

np.unique(labels)
```

```
# In[135]:

pd.DataFrame(labels2).value_counts()

# In[ ]:



# In[136]:

dbscan_params = {
    'eps': 2,
    'min_samples': 15
}

dbscan_tsne_plot(data_full, dbscan_params)

# neveikia, reikes parodyt, kad ant data_full sitas algo tsg neveikia:D

# #### Evaluating clustering results

# In[137]:

########## duomenu aibe su atrinktais duomenim ##########
# Finding optimal eps

scores_list = []

# Finding optimal eps and min_samples
for _eps in np.arange(0.001, 0.25, 0.02):
    for _min_sample in np.arange(2, 20, 2):
        dbscan = DBSCAN(eps=_eps, min_samples=_min_sample)
        labels = dbscan.fit_predict(data)

        # Make sure there is more than one cluster
        if len(set(labels)) > 1:
            sil_score = silhouette_score(data, labels)

            # Save the parameters and scores
            scores_list.append({
                'eps': _eps,
                'min_samples': _min_sample,
                'silhouette_score': sil_score,
            })
    print('round done')

scores_df = pd.DataFrame(scores_list)

# In[138]:

#### PLOTTING SCORES ####
# Plot Silhouette Score
plt.figure(figsize=(12, 6))
scatter = plt.scatter(scores_df['eps'], scores_df['min_samples'],
c=scores_df['silhouette_score'], cmap='viridis')
colorbar = plt.colorbar(scatter)
colorbar.set_label('Silhouette Score', fontsize=13)
plt.xlabel('eps', fontsize = 14)
plt.ylabel('min_samples',  fontsize = 14)
plt.title('Silhouette Score for Different DBSCAN Parameters', fontsize = 16)
plt.show()

# In[139]:
```

```
########## tSNE duomenu aibe ##########
# Finding optimal eps

scores_list = []

# Finding optimal eps and min_samples
for _eps in np.arange(1.5, 3.25, 0.25):
    for _min_sample in np.arange(2, 20, 2):
        dbscan = DBSCAN(eps=_eps, min_samples=_min_sample)
        labels = dbscan.fit_predict(data_tsne)

        # Make sure there is more than one cluster
        if len(set(labels)) > 1:
            sil_score = silhouette_score(data_tsne, labels)

            # Save the parameters and scores
            scores_list.append({
                'eps': _eps,
                'min_samples': _min_sample,
                'silhouette_score': sil_score,
            })
    print('round done')

scores_df = pd.DataFrame(scores_list)

# In[140]:

scores_df

# In[141]:

#### PLOTTING SCORES ####
# Plot Silhouette Score
plt.figure(figsize=(12, 6))
scatter = plt.scatter(scores_df['eps'], scores_df['min_samples'],
c=scores_df['silhouette_score'], cmap='viridis')
colorbar = plt.colorbar(scatter)
colorbar.set_label('Silhouette Score', fontsize=13)
plt.xlabel('eps', fontsize = 14)
plt.ylabel('min_samples',  fontsize = 14)
plt.title('Silhouette Score for Different DBSCAN Parameters', fontsize = 16)
plt.show()

# In[142]:

########## NORMAL DATA ##########
# Finding optimal eps
for _eps in np.arange(0.05, 1.0, 0.05):
    _eps = round(_eps, 2)

    dbscan = DBSCAN(eps=_eps, min_samples=5)
    labels = dbscan.fit_predict(data)

    # Calculate the Silhouette Score (ignoring noise points labeled as -1)
    if len(set(labels)) > 1:  # Make sure there is more than one cluster
        score = silhouette_score(data, labels)
        print(f"DBSCAN with eps={_eps}, min_samples=5 - Silhouette Score:
{score:.3f}")
    else:
        print(f"DBSCAN with eps={_eps}, min_samples=5 - 1 cluster detected.")

# Finding optimal min_sample
for _min_sample in np.arange(2, 50, 10):
    dbscan = DBSCAN(eps=0.05, min_samples=_min_sample)
    labels = dbscan.fit_predict(data)
```

```
    # Calculate the Silhouette Score (ignoring noise points labeled as -1)
    if len(set(labels)) > 1:  # Make sure there is more than one cluster
        score = silhouette_score(data, labels)
        print(f"eps=0.6, min_samples={_min_sample} - Silhouette Score: {score:.3f}")
    else:
        print(f"eps=0.05, min_samples={_min_sample} - 1 cluster detected.")

# In[143]:

########## FULL DATA ##########
for _eps in np.arange(0.1, 10.0, 0.5):
    #_eps = round(_eps, 2)

    dbscan = DBSCAN(eps=_eps, min_samples=100)
    labels = dbscan.fit_predict(data_full)

    # Calculate the Silhouette Score (ignoring noise points labeled as -1)
    if len(set(labels)) > 1:  # Make sure there is more than one cluster
        score = silhouette_score(data_full, labels)
        print(f"DBSCAN with eps={_eps}, min_samples=5 - Silhouette Score:
{score:.3f}")
    else:
        print(f"DBSCAN with eps={_eps}, min_samples=5 - Only one cluster detected,
score not applicable.")

### Tried calculating with different min_samples (2, 4, 10, 20, 100) and different eps
(from 0.01 to 10.0) and clustering always fails

# In[ ]:




# #### Analysing best plots

# In[144]:

dbscan_labels = dbscan_plot_one(data_tsne, {'eps': 2.6, 'min_samples': 5})

# In[145]:

df_tsne = pd.DataFrame(data_tsne, columns=['feature1', 'feature2'])
df_tsne['class'] = dfminmax['class']
df_tsne['cluster_label'] = dbscan_labels
df_tsne.value_counts('cluster_label')

# Jeigu reiktu
df_tsne.loc[df_tsne['cluster_label'] == 1, 'cluster_label'] = 100
df_tsne.loc[df_tsne['cluster_label'] == 2, 'cluster_label'] = 1
df_tsne.loc[df_tsne['cluster_label'] == 100, 'cluster_label'] = 2
# In[146]:

df_tsne_different = df_tsne[df_tsne['class'] != df_tsne['cluster_label']]
df_tsne_different.value_counts('class')

# In[147]:

df_tsne_different.value_counts('cluster_label')

# In[148]:

df_tsne_different.value_counts('cluster_label').sum()

# In[149]:

dfminmax['cluster_label'] = df_tsne['cluster_label']
```

```
dfminmax[dfminmax['cluster_label'] == 2].describe()

# In[150]:

dfminmax[dfminmax['class'] == 0].describe()

# In[ ]:



# In[151]:

df_tsne.describe()

# #### Plot (in)correctly assigned points

# In[152]:

df_tsne_different = df_tsne[df_tsne['class'] != df_tsne['cluster_label']]
plt.figure(figsize=(10, 8))

plt.title(f"Non-coinciding points", fontsize=16)
scatter = plt.scatter(df_tsne_different['feature1'], df_tsne_different['feature2'],
c=df_tsne_different['cluster_label'], cmap='viridis', s=100, alpha=0.7)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
handles, labels = scatter.legend_elements()
plt.legend(handles, labels, loc='lower right', title="Cluster Label")

# In[153]:

df_orig.head()

# In[154]:

df_tsne_different = df_tsne[df_tsne['class'] != df_tsne['cluster_label']]
plt.figure(figsize=(10, 8))

plt.title(f"Non-coinciding points", fontsize=16)
scatter = plt.scatter(df_tsne_different['feature1'], df_tsne_different['feature2'],
c=df_tsne_different['class'], cmap='viridis', s=100, alpha=0.7)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
handles, labels = scatter.legend_elements()
plt.legend(handles, labels, loc='lower right', title="class", fontsize=12)

# In[155]:

df_tsne_same = df_tsne[df_tsne['class'] == df_tsne['cluster_label']]
plt.figure(figsize=(10, 8))

plt.title(f"Coinciding points", fontsize=16)
scatter = plt.scatter(df_tsne_same['feature1'], df_tsne_same['feature2'],
c=df_tsne_same['class'], cmap='viridis', s=100, alpha=0.7)
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
handles, labels = scatter.legend_elements()
plt.legend(handles, labels, loc='lower right', title="class", fontsize=12)

# In[156]:

dfminmax['cluster_label'] = df_tsne['cluster_label']

# In[157]:
```

```
dfminmax.head()

# In[158]:

dfminmax['incorrect_labels'] = dfminmax[dfminmax['class'] !=
dfminmax['cluster_label']].value_counts('cluster_label')

# In[159]:

plt.figure(figsize=(8, 10))
# Create boxplots for 'redshift' grouped by 'cluster_label' and 'class'
plt.figure(figsize=(14, 6))

# Boxplot for 'redshift' by 'cluster_label'
plt.subplot(1, 2, 1)
sns.boxplot(x=dfminmax['cluster_label'], y=dfminmax['redshift'])
plt.title("Redshift by Cluster Label")
plt.xlabel("Cluster Label")
plt.ylabel("Redshift")

# Boxplot for 'redshift' by 'class'
plt.subplot(1, 2, 2)
sns.boxplot(x=dfminmax['class'], y=dfminmax['redshift'])
plt.title("Redshift by Class")
plt.xlabel("Class")
plt.ylabel("Redshift")

plt.tight_layout()
plt.show()


# In[160]:

df_outliers = dfminmax[dfminmax['class'] != dfminmax['cluster_label']]

# In[161]:

df_outliers.loc[df_outliers['class'] == 2, 'redshift'].mean(),
df_outliers.loc[df_outliers['cluster_label'] == 2, 'redshift'].mean()

# #### Jei butu daugiau reiksmiu butu belekoks palyginimas cia gautas

# In[162]:

# ok cia pas mane nelabai kas yra daryti ig, bet tipo galima pastebeti, kad redshift =
0 arba labai labai maza reiksme
sns.boxplot(x=dfminmax['class'], y=dfminmax['redshift'])

# In[163]:

sns.boxplot(x=df_outliers['class'], y=df_outliers['redshift'])

# In[164]:

sns.boxplot(x=df_outliers['cluster_label'], y=df_outliers['redshift'])

# In[165]:

sns.boxplot(x=dfminmax['class'], y=dfminmax['i'])

# In[166]:

sns.boxplot(x=df_outliers['class'], y=df_outliers['i'])

# In[ ]:
```

```
# In[167]:

#### normal data analysing

# In[168]:

dbscan_params = {
    'eps': 0.01,
    'min_samples': 5
}

labels = dbscan_tsne_plot(data, dbscan_params)

# In[169]:

dbscan_params = {
    'eps': 0.05,
    'min_samples': 5,
    'leaf_size': 25
}

labels2 = dbscan_tsne_plot(data, dbscan_params)

# In[170]:

dbscan_params = {
    'eps': 0.05,
    'min_samples': 5,
    'leaf_size': 35
}

labels2 = dbscan_tsne_plot(data, dbscan_params)

# In[171]:

df_tsne = pd.DataFrame(data_tsne, columns=['feature1', 'feature2'])
df_tsne['label'] = dfminmax['class']
df_tsne['cluster_label'] = labels
df_tsne.value_counts('cluster_label')

# In[172]:

dfminmax['cluster_label'] = df_tsne['cluster_label']

# In[173]:

plt.figure(figsize=(8, 10))
# Create boxplots for 'redshift' grouped by 'cluster_label' and 'class'
plt.figure(figsize=(14, 6))

# Boxplot for 'redshift' by 'cluster_label'
plt.subplot(1, 2, 1)
sns.boxplot(x=dfminmax['cluster_label'], y=dfminmax['redshift'])
plt.title("Redshift by Cluster Label")
plt.xlabel("Cluster Label")
plt.ylabel("Redshift")

# Boxplot for 'redshift' by 'class'
plt.subplot(1, 2, 2)
sns.boxplot(x=dfminmax['class'], y=dfminmax['redshift'])
plt.title("Redshift by Class")
plt.xlabel("Class")
```

```
plt.ylabel("Redshift")

plt.tight_layout()
plt.show()


# In[ ]:




# In[ ]:




# ### KMeans - getting the amounts of clusters with 'elbow', 'silhouette'. Using it on
t-SNE data(WHICH IS BAD).

# In[8]:

wcss = []  # List to store WCSS values

# Try different numbers of clusters
for n_clusters in range(1, 11):
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(data_tsne)
    wcss.append(kmeans.inertia_)

# Plot the WCSS values
plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), wcss, marker='o')
plt.xlabel('Klasterių skaičius')
plt.ylabel('WCSS')
plt.title('Elbow metodas')
plt.show()

# In[9]:

silhouette_scores = []

# Try different numbers of clusters
for n_clusters in range(2, 11):  # Silhouette score is not defined for 1 cluster
    kmeans = KMeans(n_clusters=n_clusters, init='k-means++', max_iter=300, tol=0.001,
random_state=42)
    labels = kmeans.fit_predict(data_tsne)
    silhouette_scores.append(silhouette_score(data_tsne, labels))

# Plot the Silhouette Scores
plt.figure(figsize=(8, 5))
plt.plot(range(2, 11), silhouette_scores, marker='o')
# best params - 'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 0.001

plt.xlabel('Klasterių skaičius')
plt.ylabel('Silhouette reikšmė')
plt.title('Silhouette metodas')
plt.show()

# ### kMeans

# #### Defining functions

# In[10]:

def kmeans_plot_one(X, params):
    # Initialize and fit KMeans with the provided parameters
```

```
    kmeans = KMeans(**params, random_state=42)
    labels = kmeans.fit_predict(X)

    # Plot the clusters
    plt.figure(figsize=(10, 8))
    scatter = plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7)

    # Set title
    plt.title(f"KMeans: n_clusters={params['n_clusters']}, init={params.get('init',
'k-means++')}", fontsize=18)

    # Add legend for cluster labels
    unique_labels = np.unique(labels)
    if len(unique_labels) < 10:  # Add legend only if there are fewer than 10 clusters
        labels_list = []
        for label in unique_labels:
            # Use the colormap for cluster labels
            color = scatter.cmap(label / (max(unique_labels) if max(unique_labels) > 0
else 1))
            labels_list.append(plt.scatter([], [], color=color, label=f'Cluster
{label}'))

        # Add legend with labels
        plt.legend(handles=labels_list, loc='lower right', fontsize=14)

    # Set axis labels and remove ticks
    plt.xlabel('Feature 1', fontsize=18)
    plt.ylabel('Feature 2', fontsize=18)
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)

    plt.tight_layout()
    plt.show()

    return labels

def kmeans_visual_comparison(X, param_grid):
    # Create subplots for all parameter combinations
    n_params = len(param_grid)
    n_rows = (n_params + 2) // 3  # Rows for the grid layout
    fig, axes = plt.subplots(n_rows, 3, figsize=(18, 6 * n_rows))
    axes = axes.flatten()

    for i, (ax, params) in enumerate(zip(axes, param_grid)):
        # Run KMeans with the current parameters
        kmeans = KMeans(**params, random_state=42)
        labels = kmeans.fit_predict(X)

        # Plot the clusters
        scatter = ax.scatter(
            X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.7
        )

        # Add title with parameters
        title = (
            # f"n_clusters={params.get('n_clusters', 3)} "
            # f"init={params.get('init', 'k-means++')} "
            # f"n_iter={params.get('n_iter', 300)} "
            f"tol={params.get('tol', 1e-4)}"
        )
        ax.set_title(title, fontsize=18)
        ax.set_xticks([])
        ax.set_yticks([])

    # Hide unused subplots
    for ax in axes[len(param_grid):]:
```

```
        ax.axis('off')

    plt.tight_layout()
    plt.show()

def kmeans_tsne_plot(X, kmeans_params=None):
    # Run KMeans
    kmeans = KMeans(**kmeans_params, random_state=42)
    labels = kmeans.fit_predict(X)

    # Reduce data to 2D using t-SNE
    tsne = manifold.TSNE(n_components=2,
                perplexity=50,
                n_iter=750,
                metric='canberra',
                random_state=42)
    data_tsne = tsne.fit_transform(X)

    # Plot the data
    plt.figure(figsize=(12, 8))

    # Number of clusters in labels
    n_clusters = kmeans_params['n_clusters']
    colors = cm.nipy_spectral(np.linspace(0, 1, n_clusters))

    # Plot each cluster
    for k, col in zip(range(n_clusters), colors):
        label_name = f'Cluster {k}'

        class_member_mask = (labels == k)
        xy = data_tsne[class_member_mask]

        plt.scatter(
            xy[:, 0],
            xy[:, 1],
            c=[col],
            label=label_name,
            edgecolors='k',
            alpha=0.7,
            s=50
        )

    plt.title('kMeans klasterizavimas, kai dimensijos mažinamos po kMeans',
fontsize=20)
    plt.xlabel('t-SNE Dimensija 1', fontsize=18)
    plt.ylabel('t-SNE Dimensija 2', fontsize=18)
    plt.legend(loc='best', fontsize=14)
    plt.grid(True)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.show()

    return labels

# #### KMEANS for reduced data

# In[11]:

# Define parameter combinations for testing
param_grid = [
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    # {'n_clusters': 4, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    # {'n_clusters': 9, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    #  n_clusters - 3 best.

    # {'n_clusters': 3, 'init': 'random', 'max_iter': 300, 'tol': 1e-4},
```

```
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    # init method - k-means++ better.

    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 10, 'tol': 1e-4},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 900, 'tol': 1e-4},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 500, 'tol': 1e-4},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 800, 'tol': 1e-4},
    # max_iter - no change really.

    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-6},
    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-5},
    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-4},
    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 0.001},
    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-2},
    {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 1e-1},

    # tol - 0.001 best. Can't seem to get it to be perfect though.
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 10, 'tol': 0.0001},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 0.0001},
    # {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 0.1},



]

# best params - 'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol': 0.001

# Run the comparison
kmeans_visual_comparison(data_tsne, param_grid)

# ### KMEANS for NOT reduced data, then after the data gets reduced

# In[12]:

kmeans_tsne_plot(data, {'n_clusters': 3, 'init': 'k-means++', 'max_iter': 300, 'tol':
0.001})

# #### Evaluating clustering results. results are used visually in next steps,
visualising with 'silhouette'.

# In[13]:

####### tSNE DATA ##########
# Parameter search for KMeans
scores_list = []

# Finding optimal n_clusters and initialization method
for n_clusters in range(2, 20):  # Test different numbers of clusters
    for init_method in ['k-means++', 'random']:  # Test different initialization
methods
        kmeans = KMeans(n_clusters=n_clusters, init=init_method, random_state=42)
        labels = kmeans.fit_predict(data)  # Run KMeans on the raw data

        # Ensure there is more than one cluster
        if len(set(labels)) > 1:
            sil_score = silhouette_score(data_tsne, labels)  # Compute silhouette
score on t-SNE data

            # Save the parameters and scores
            scores_list.append({
                'n_clusters': n_clusters,
                'init_method': init_method,
                'silhouette_score': sil_score,
            })
    print(f'n_clusters={n_clusters} round done')
```

```
# Convert results into a DataFrame
scores_df = pd.DataFrame(scores_list)

# Display the top results
print(scores_df.sort_values(by='silhouette_score', ascending=False).head())

# In[14]:

#### PLOTTING SCORES ####
import matplotlib.pyplot as plt
import seaborn as sns

# Set up the figure
plt.figure(figsize=(12, 6))

# Create a scatter plot of silhouette scores
scatter = plt.scatter(
    scores_df['n_clusters'],
    scores_df['silhouette_score'],
    c=scores_df['init_method'].apply(lambda x: 0 if x == 'k-means++' else 1),
    cmap='viridis',
    s=100,
    alpha=0.8,
    edgecolor='k'
)

# Add a colorbar for the initialization method
cbar = plt.colorbar(scatter, ticks=[0, 1])
cbar.ax.set_yticklabels(['k-means++', 'random'])
cbar.set_label('init reikšmė', fontsize=16)

# Customize the plot
plt.xlabel('Klasterių skaičius (n_clusters)', fontsize=14)
plt.ylabel('Silueto rodiklis', fontsize=14)
plt.title('KMEANS klasterizavimas. Silueto rodiklis pagal parametrus', fontsize=16)
plt.grid(True, linestyle='--', alpha=0.6)

# Adjust x-axis ticks to include all tested n_clusters
plt.xticks(range(scores_df['n_clusters'].min(), scores_df['n_clusters'].max() + 1))

# Show the plot
plt.tight_layout()
plt.show()

# #### 'Elbow' testing on original data(THE GOOD WAY).

# In[15]:

wcss_list = []

# Range of cluster numbers to test
n_clusters_range = range(1, 20)  # Adjust as needed

# Initialization methods to test
init_methods = ['k-means++', 'random']

# Perform KMeans clustering and compute WCSS for each combination
for n_clusters in n_clusters_range:
    for init_method in init_methods:
        kmeans = KMeans(n_clusters=n_clusters, init=init_method, random_state=42)
        kmeans.fit(data)  # Use original data for clustering

        # Retrieve WCSS (inertia_)
        wcss = kmeans.inertia_
```

```
        # Save the parameters and WCSS
        wcss_list.append({
            'n_clusters': n_clusters,
            'init_method': init_method,
            'wcss': wcss,
        })
    print(f'n_clusters={n_clusters} round done')

# Convert results into a DataFrame
wcss_df = pd.DataFrame(wcss_list)

# Display the results
print(wcss_df.head())

# Optional: Plot WCSS values to visualize the elbow
plt.figure(figsize=(10, 6))

for init_method in init_methods:
    subset = wcss_df[wcss_df['init_method'] == init_method]
    plt.plot(subset['n_clusters'], subset['wcss'], marker='o', label=f"Init:
{init_method}")

plt.xlabel('Number of Clusters (n_clusters)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.figure(figsize=(10, 6))

for init_method in init_methods:
    subset = wcss_df[wcss_df['init_method'] == init_method]
    plt.plot(subset['n_clusters'], subset['wcss'], marker='o', label=f"Init:
{init_method}")

plt.xlabel('Number of Clusters (n_clusters)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal Number of Clusters')
plt.legend()
plt.grid(True, linestyle='--', alpha=0.6)

# Add this line to set x-axis ticks to integer cluster numbers
plt.xticks(n_clusters_range)

plt.tight_layout()
plt.show()

# #### Analysing best plots

# In[16]:

dbscan_labels = kmeans_plot_one(data_tsne, {'n_clusters': 3, 'init': 'k-means++',
'max_iter': 300, 'tol': 0.001})

# In[18]:

df_tsne = pd.DataFrame(data_tsne, columns=['feature1', 'feature2'])
df_tsne['label'] = dfminmax['class']
df_tsne['cluster_label'] = dbscan_labels

# In[19]:

df_tsne.describe()

# In[20]:
```

```
df_tsne_different = df_tsne[df_tsne['label'] != df_tsne['cluster_label']]
df_tsne_different.value_counts('label')

# In[21]:

# Apkeiciam cluster_label vietomis, kad atitiktu klases
df_tsne.loc[df_tsne['cluster_label'] == 1, 'cluster_label'] = 100
df_tsne.loc[df_tsne['cluster_label'] == 2, 'cluster_label'] = 1
df_tsne.loc[df_tsne['cluster_label'] == 100, 'cluster_label'] = 2

# In[22]:

df_tsne_different = df_tsne[df_tsne['label'] != df_tsne['cluster_label']]
df_tsne_different.value_counts('label')

# In[23]:

df_tsne_different.value_counts('cluster_label')

# In[22]:

# Find the best mapping from cluster_label to label
from scipy.optimize import linear_sum_assignment
from sklearn.metrics import confusion_matrix

# Create a confusion matrix between true labels and cluster labels
conf_matrix = confusion_matrix(df_tsne['label'], df_tsne['cluster_label'])

# Use Hungarian algorithm to find the optimal label-to-cluster mapping
row_ind, col_ind = linear_sum_assignment(-conf_matrix)

# Create a mapping dictionary
mapping = {cluster: label for cluster, label in zip(col_ind, row_ind)}

# Map cluster labels to match the true labels
df_tsne['cluster_label'] = df_tsne['cluster_label'].map(mapping)

# Recalculate mismatched rows after remapping
df_tsne_different = df_tsne[df_tsne['label'] != df_tsne['cluster_label']]

# Display updated mismatch counts
print("Neatitinakčių klasių objektų skaičius:")
print(df_tsne_different.value_counts('label'))

# #### Plot incorrectly assigned points

# In[23]:

import matplotlib.pyplot as plt
import numpy as np

def visualize_tsne_results(df_tsne):
    mismatches = df_tsne['label'] != df_tsne['cluster_label']
    num_mismatches = mismatches.sum()
    total_samples = len(df_tsne)
    percentage_mismatched = (num_mismatches / total_samples) * 100

    # Scatter plot for true labels
    plt.figure(figsize=(18, 6))

    plt.subplot(1, 3, 1)
    scatter = plt.scatter(df_tsne['feature1'], df_tsne['feature2'],
c=df_tsne['label'], cmap='viridis', s=70, alpha=0.8)
    cbar = plt.colorbar(scatter, ticks=np.unique(df_tsne['label']))  # Set colorbar
ticks to unique label values
    cbar.set_label('Tikslios klasės', fontsize=16)
```

```
    cbar.ax.tick_params(labelsize=14)
    plt.title('t-SNE klasteriai', fontsize=18)
    plt.xlabel('Dimensija 1', fontsize=16)
    plt.ylabel('Dimensija 2', fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)

    # Scatter plot for cluster labels
    plt.subplot(1, 3, 2)
    scatter = plt.scatter(df_tsne['feature1'], df_tsne['feature2'],
c=df_tsne['cluster_label'], cmap='viridis', s=70, alpha=0.8)
    cbar = plt.colorbar(scatter, ticks=np.unique(df_tsne['cluster_label']))  # Set
colorbar ticks to unique cluster labels
    cbar.set_label('Klasteriai', fontsize=16)
    cbar.ax.tick_params(labelsize=14)
    plt.title('kMeans klasės', fontsize=18)
    plt.xlabel('Dimensija 1', fontsize=16)
    plt.ylabel('Dimensija 2', fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)

    # Scatter plot for mismatches
    plt.subplot(1, 3, 3)
    plt.scatter(
        df_tsne.loc[~mismatches, 'feature1'],
        df_tsne.loc[~mismatches, 'feature2'],
        c='gray',
        s=70,
        alpha=0.5,
        label='Atitinka'
    )
    plt.scatter(
        df_tsne.loc[mismatches, 'feature1'],
        df_tsne.loc[mismatches, 'feature2'],
        c='red',
        s=70,
        alpha=0.8,
        label='Neatitinka'
    )
    plt.legend(fontsize=14)
    plt.title(f't-SNE atitikimas pagal klases\n{percentage_mismatched:.2f}%
neatitinka', fontsize=18)
    plt.xlabel('Dimensija 1', fontsize=16)
    plt.ylabel('Dimensija 2', fontsize=16)
    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)

    plt.tight_layout()
    plt.show()

    # Print the percentage of mismatches
    print(f"Percentage of mismatched objects: {percentage_mismatched:.2f}%")

# In[24]:

visualize_tsne_results(df_tsne)

# In[37]:

df_tsne.describe()

# In[25]:

dfminmax['cluster_label'] = df_tsne['cluster_label']

# In[26]:
```

```
dfminmax['incorrect_labels'] = dfminmax[dfminmax['class'] !=
dfminmax['cluster_label']].value_counts('cluster_label')

# In[27]:

plt.figure(figsize=(8, 10))
# Create boxplots for 'redshift' grouped by 'cluster_label' and 'class'
plt.figure(figsize=(14, 6))

# Boxplot for 'redshift' by 'cluster_label'
plt.subplot(1, 2, 1)
sns.boxplot(x=dfminmax['cluster_label'], y=dfminmax['redshift'])
plt.title("Redshift by Cluster Label")
plt.xlabel("Cluster Label")
plt.ylabel("Redshift")

# Boxplot for 'redshift' by 'class'
plt.subplot(1, 2, 2)
sns.boxplot(x=dfminmax['class'], y=dfminmax['redshift'])
plt.title("Redshift by Class")
plt.xlabel("Class")
plt.ylabel("Redshift")

plt.tight_layout()
plt.show()


# In[28]:

df_outliers = dfminmax[dfminmax['class'] != dfminmax['cluster_label']]
# df_outliers - neatitinakcios klasterizavimo rezultato klases
df_outliers.describe()

# In[29]:

num_class_0_outliers = df_outliers[df_outliers['class'] == 0].shape[0]
print(f"Neatitinkančių objektų kiekis 0 klasei: {num_class_0_outliers}")
num_class_1_outliers = df_outliers[df_outliers['class'] == 1].shape[0]
print(f"Neatitinkančių objektų kiekis 1 klasei: {num_class_1_outliers}")
num_class_2_outliers = df_outliers[df_outliers['class'] == 2].shape[0]
print(f"Neatitinkančių objektų kiekis 2 klasei: {num_class_2_outliers}")

# In[30]:

df_outliers.loc[df_outliers['class'] == 2, 'redshift'].mean(),
df_outliers.loc[df_outliers['cluster_label'] == 2, 'redshift'].mean()

# nu ir ok, cia tavo padaryta, nu ir gerai, man sito nereikia, nes tik 1 objektas 2
kategorijos blogai kateogiruoztas, nu tai lol?

# In[31]:

for col in normalization_cols:
    plt.figure(figsize=(14, 6))

    # First subplot: Boxplot of the variable by 'class' in dfminmax
    plt.subplot(1, 2, 1)
    sns.boxplot(x='class', y=col, data=dfminmax)
    plt.title(f'{col.capitalize()} reikšmė, t-SNE algoritmas.', fontsize=16)
    plt.xlabel('Klasė', fontsize=14)
    plt.ylabel(col.capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # Second subplot: Boxplot of the variable by 'class' in df_outliers
```

```
    plt.subplot(1, 2, 2)
    sns.boxplot(x='class', y=col, data=df_outliers)
    plt.title(f'{col.capitalize()} reikšmė, kMeans persidengiančių objektų
neatitikimas.', fontsize=16)
    plt.xlabel('klasė', fontsize=14)
    plt.ylabel(col.capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # Adjust layout and display the plots
    plt.tight_layout()
    plt.show()

# #### Comparison of df_outliers(mismatching KMeans labels and classes) and dfminmax

# In[32]:

for col in normalization_cols:
    plt.figure(figsize=(14, 6))

    # First subplot: Boxplot of the variable by 'class' in dfminmax
    plt.subplot(1, 2, 1)
    sns.boxplot(x='class', y=col, data=dfminmax)
    plt.title(f'{col.capitalize()} reikšmė, t-SNE algoritmas.', fontsize=16)
    plt.xlabel('Klasė', fontsize=14)
    plt.ylabel(col.capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # Second subplot: Boxplot of the variable by 'class' in df_outliers
    plt.subplot(1, 2, 2)
    sns.boxplot(x='class', y=col, data=df_outliers)
    plt.title(f'{col.capitalize()} reikšmė, kMeans persidengiančių objektų
neatitikimas.', fontsize=16)
    plt.xlabel('klasė', fontsize=14)
    plt.ylabel(col.capitalize(), fontsize=14)
    plt.xticks(fontsize=12)
    plt.yticks(fontsize=12)

    # Adjust layout and display the plots
    plt.tight_layout()
    plt.show()

# In[33]:

df_tsne = pd.DataFrame(data_tsne, columns=['feature1', 'feature2'])
df_tsne['label'] = dfminmax['class']
df_tsne['cluster_label'] = labels
df_tsne.value_counts('cluster_label')

# In[34]:

dfminmax['cluster_label'] = df_tsne['cluster_label']
```