Kauno technologijų universitetas

Informatikos fakultetas

Intelektikos pagrindai (P176B101)

Inžinerinio darbo ataskaita

Nojus Stauskas IFF-1/1
Vytenis Kriščiūnas IFF-1/1
Studentai

Arnas Nakrošis
Audrius Nečiūnas
Dėstytojai

Kaunas, 2024

Turinys

1. Užduotis

Reikalavimai:

- Uždavinys: neprižiūrimojo mašininio mokymosi algoritmų realizacija pasirinktam duomenų rinkiniui
- Komandą sudaro 2 žmonės
- Darbo pateikimo deadline: gegužės 31 diena.
- Ataskaita + programinis kodas

Ką reikia atlikti:

Realizuoti 2 neprižiūrimo mašininio mokymosi algoritmus (tarkime *K*-vidurkių ir SOM) ir palyginti gautus rezultatus. Kadangi nėra kaip patikrinti modelio tikslumą (duomenų rinkinyje nėra išvesties) tai turi būti atliktas klasterių vertinimas (pvz., silueto koeficientas). Svarbu atlikti kuo įvairesnius eksperimentus ir pakomentuoti rezultatus

Ataskaitoje reikia pateikti:

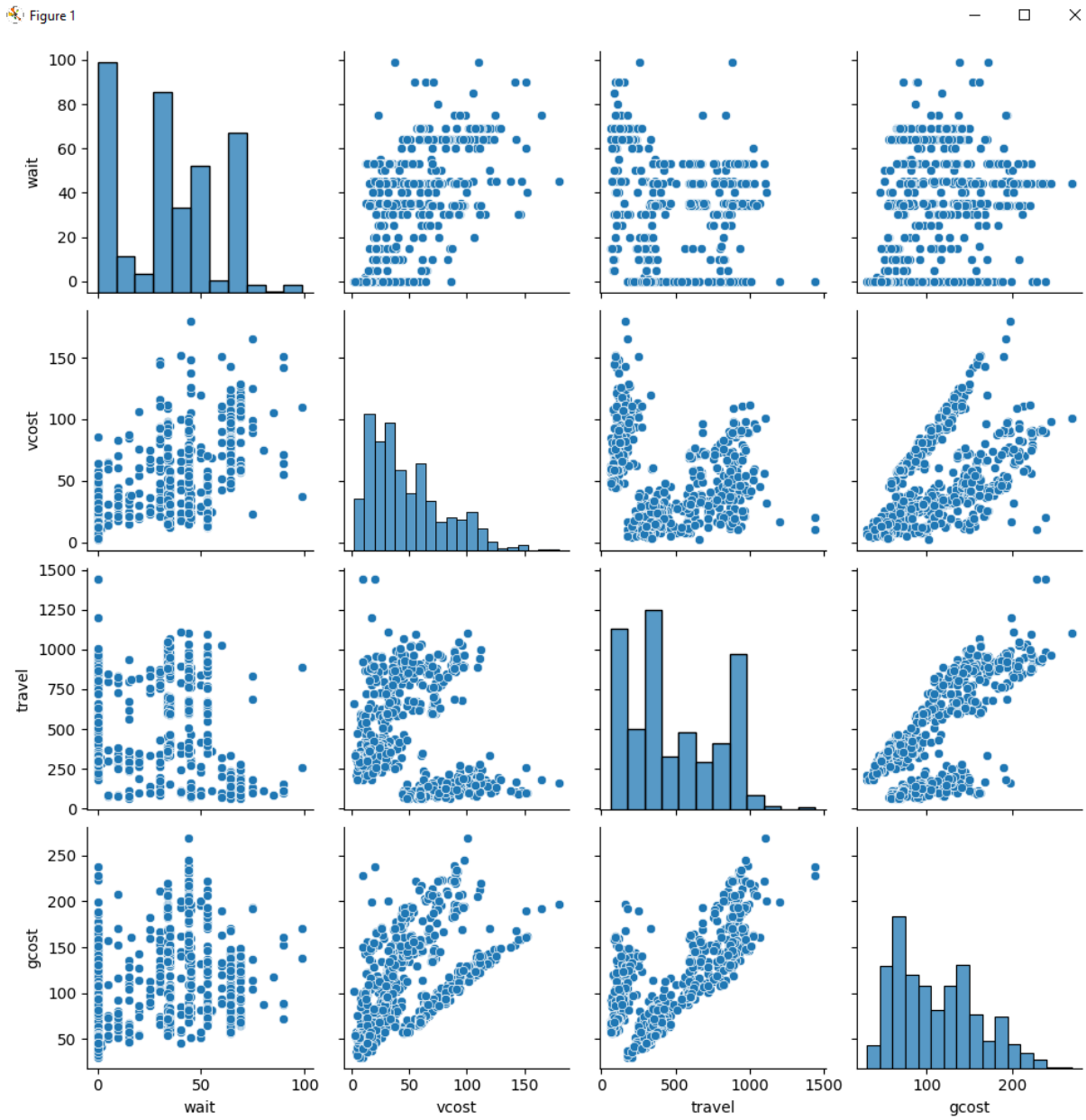
- Duomenų rinkinį ir jo aprašymą (gali būti jau naudotas rinkinys (be išvesties tik), arba naujas)
- Metodų, kuriuos planuojate naudoti trumpas aprašymas
- Atstumo metrikos (pvz., *Euklido* arba kita).
- Eksperimentus, kurie apima:
- 1 algoritmas tarkime *SOM*: keisti klasterių skaičių (pvz., k=3, 4, 5) ir pateikti atsakymus grafiškai. Rezultatus pakomentuoti.
- 2 algoritmas tarkime *K-vidurkių*: keisti klasterių skaičių (pvz., k=3, 4, 5) ir pateikti atsakymus grafiškai. Taip pat atlikti eksperimentus pagal skirtingus atributus. Pvz., turite brangakmenių duomenų rinkinį, tai X=aukštis, Y=plotis; X=skaidrumas, Y=svoris;... . Paskaičiuoti *inerciją* ir *Siluteto koeficiento*, kurių atsakymus pateikti ir grafiškai, ir skaitinėmis reikšmėmis.
- Palyginti gautus abiejų algoritmų rezultatus ir pateikti savo išvadas, ar duomenų rinkiniui tinkamas klasterizavimo metodas, koks klasterių skaičius geriausias, kokie atributai tinkamiausi ir pan.

2. Duomenų rinkinys

Šiam komandiniam darbui atlikti pasirinkome vieno iš mūsų jau naudotą duomenų rinkinį. Duomenų rinkinys sudarytas iš šešių tolydinio ir trijų kategorinių tipo duomenų. Trūkstamų reikšmių duomenyse nėra. Kategoriniai duomenys, nesusieti tolydiniai duomenys ir mažo kardinalumo tolydiniai duomenys nebus įtraukiami į tyrimą.
Mūsų duomenys apibūdina skirtingus transporto priemonių pasirinkimus. Galimos transporto priemonės: „car“, „bus“, „air“ ir „train“. Tokie stulpeliai, kaip „income“ ir „size“ neturi susietumo su kitais tolydiniais duomenimis ir yra vienodi kitų atributų atžvilgiu: „wait“, „vcost“, „travel“ ir „gcost“.
Duomenų rinkinys: https://vincentarelbundock.github.io/Rdatasets/doc/AER/TravelMode.html
Duomenų rinkinio atributai:

- „Individual“ – faktorius nurodantis individą nuo 1 iki 210 lygio;
- „Mode“ – faktorius indikuojantis kelionės rūšį: mašina, oru, traukiniu ar autobusu;

- „Choise“ – faktorius nurodantis pasirinkimą taip ar ne;
- „Wait“ – laukimo laikas terminale, 0 keliaujant mašina;
- „Vcost“ – transporto priemonės kaina;
- „Travel“ – kelionės trukmė transporto priemonėje;
- „Gcost“ – bendra kelionės kaina;
- „Income“ – uždarbis;
- „Size“ – žmonių kiekis.



1 pav. „Scater plot“ matrica

Galima pastebėti, kad „vcost“-„gcost“, „travel“-„gcost“ ir „travel“-„vcost“ duomenys lyg ir susigrupuoja į dvi atskiras grupes, šiuos duomenis ir naudosime klasterizavimui.

3. K-Vidurkių

3.1. K-Vidurkių algoritmo aprašymas

K-Vidurkių klasterizacijos metodas yra vienas iš populiariausių neprižiūrimos mokymosi algoritmų, naudojamų duomenų klasterizavimui. Šis metodas siekia padalinti duomenų rinkinį į *k* klasterių, kur kiekvienas duomuo priskiriamas artimiausiam centroidui (klasterio centrui).

K-Vidurkių algoritmas vykdomas šiais pagrindiniais žingsniais:

Centroidų Inicializavimas:
- Pasirenkamas *k* klasterių skaičius.
- Atsitiktinai parenkami *k* taškai iš duomenų rinkinio kaip pradiniai centroidai.

Duomenų Taškų Priskyrimas:
- Kiekvienas duomenų taškas priskiriamas artimiausiam centroidui. Artimumas matuojamas naudojant Euklido atstumą (tiesinis atstumas tarp taškų).

Centroidų Atnaujinimas:
- Kiekvienas centroidas atnaujinamas kaip visų jam priskirtų taškų vidurkis. Tai reiškia, kad centroidas perkeliamas į naują poziciją, kuri yra jo klasterio taškų vidutinė pozicija.

Kriterijų Tikrinimas:
- Žingsniai 2 ir 3 kartojami tol, kol centroidų padėtis nebesikeičia arba pasiekiamas maksimalus iteracijų skaičius. Šiuo momentu klasterizacija laikoma baigta.

Algoritmo Įvertinimas

K-Vidurkių algoritmo našumą galima įvertinti naudojant du pagrindinius rodiklius:

- **Inercija:**
  - Tai yra suma kvadratinių atstumų nuo kiekvieno taško iki jo centroido. Mažesnė inercija rodo geresnę klasterizaciją.
- **Silueto koeficientas:**
  - Šis koeficientas matuoja, kaip gerai taškai priskirti savo klasteriui, lyginant su kitais klasteriais. Silueto koeficientas svyruoja nuo -1 iki 1, kur didesnės vertės reiškia geresnę klasterizaciją.

3.2. Programinis kodas

Skaičiavimų kodas:

```
def initialize_centroids(X, k):
    """Initialize centroids randomly from the dataset"""
    np.random.seed(0)
    indices = np.random.choice(X.shape[0], k, replace=False)
    return X[indices]

def assign_clusters(X, centroids):
    """Assign each point to the nearest centroid"""
    distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
    return np.argmin(distances, axis=1)

def update_centroids(X, labels, k):
    """Update centroids as the mean of all points assigned to each cluster"""
    centroids = np.zeros((k, X.shape[1]))
    for i in range(k):
        centroids[i, :] = X[labels == i].mean(axis=0)
    return centroids

def compute_inertia(X, labels, centroids):
    """Compute the inertia (sum of squared distances to the closest centroid)"""
    return np.sum((X - centroids[labels])**2)

def k_means_clustering(X, k, max_iters=300):
    centroids = initialize_centroids(X, k)
    for i in range(max_iters):
        labels = assign_clusters(X, centroids)
        new_centroids = update_centroids(X, labels, k)
        if np.all(centroids == new_centroids):
            break
        centroids = new_centroids
    inertia = compute_inertia(X, labels, centroids)
    silhouette_avg = silhouette_score(X, labels)
    return labels, centroids, inertia, silhouette_avg
```

Rezultatų parodymo grafiškai kodas:

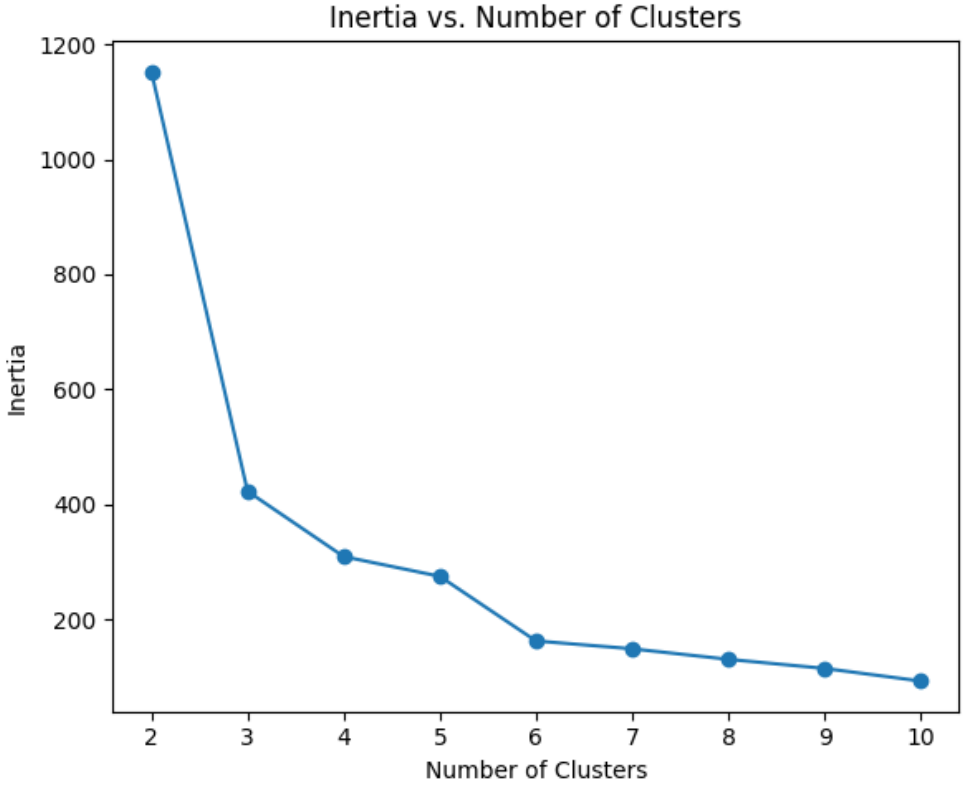
```
def plot_clusters(X, labels, centroids, k, feature1, feature2):
    plt.scatter(X[:, 0], X[:, 1], c=labels, s=50, cmap='viridis')
    plt.scatter(centroids[:, 0], centroids[:, 1], c='red', s=200, alpha=0.75,
marker='X')
    plt.title(f'K-Means Clustering with {k} Clusters')
    plt.xlabel(feature1)
```

```
    plt.ylabel(feature2)
    plt.show()

def plot_inertia_silhouette(X, max_k):
    inertia_values = []
    silhouette_values = []
    k_values = range(2, max_k + 1)

    for k in k_values:
        labels, centroids, inertia, silhouette_avg = k_means_clustering(X, k)
        inertia_values.append(inertia)
        silhouette_values.append(silhouette_avg)

    plt.figure(figsize=(12, 5))

    plt.subplot(1, 2, 1)
    plt.plot(k_values, inertia_values, marker='o')
    plt.title('Inertia vs. Number of Clusters')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Inertia')

    plt.subplot(1, 2, 2)
    plt.plot(k_values, silhouette_values, marker='o')
    plt.title('Silhouette Score vs. Number of Clusters')
    plt.xlabel('Number of Clusters')
    plt.ylabel('Silhouette Score')

    plt.tight_layout()
    plt.show()
```

3.3. Skaičiavimai

Pirmiausia skaičiavimams naudosime ,,vcost'' ir ,,travel'' stulpelius. Patikrinsime kokį klasterių skaičių būtų geriausia naudoti. Tai padarysime įvertinę silueto ir inercijos įverčius.

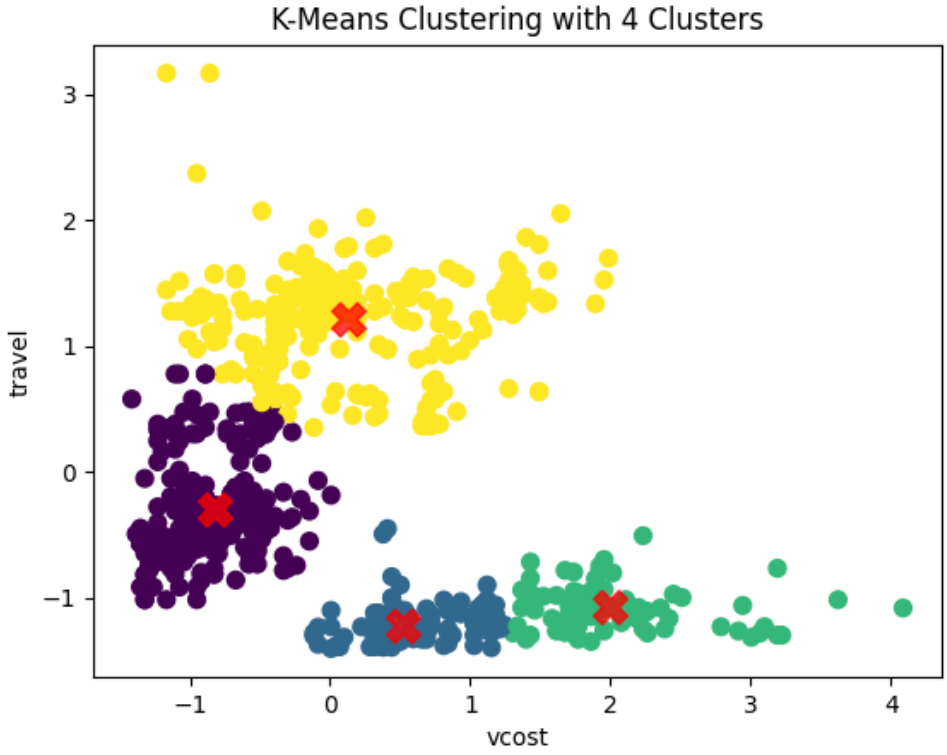
2 Clusters. Inertia: 1152.099272916562
2 Clusters. Silhouette Score: 0.38406049365121864
3 Clusters. Inertia: 423.20565857219844
3 Clusters. Silhouette Score: 0.5587815339739695
4 Clusters. Inertia: 309.66971298916656
4 Clusters. Silhouette Score: 0.5602756516360844
5 Clusters. Inertia: 275.3352818137067
5 Clusters. Silhouette Score: 0.4943218607464648
6 Clusters. Inertia: 163.06971239703347
6 Clusters. Silhouette Score: 0.5320098238969235
7 Clusters. Inertia: 149.43034756937735
7 Clusters. Silhouette Score: 0.5048018570469879
8 Clusters. Inertia: 131.13034943438103
8 Clusters. Silhouette Score: 0.45965486537392236
9 Clusters. Inertia: 115.37311602641967
9 Clusters. Silhouette Score: 0.4836124467865384
10 Clusters. Inertia: 93.46534521907526
10 Clusters. Silhouette Score: 0.4864230405930963



2 pav. „Vcost" ir „travel" inercijos diagrama pagal klasterių kiekį

3 pav. „Vcost“ ir „travel“ Silueto diagrama pagal klasterių kiekį

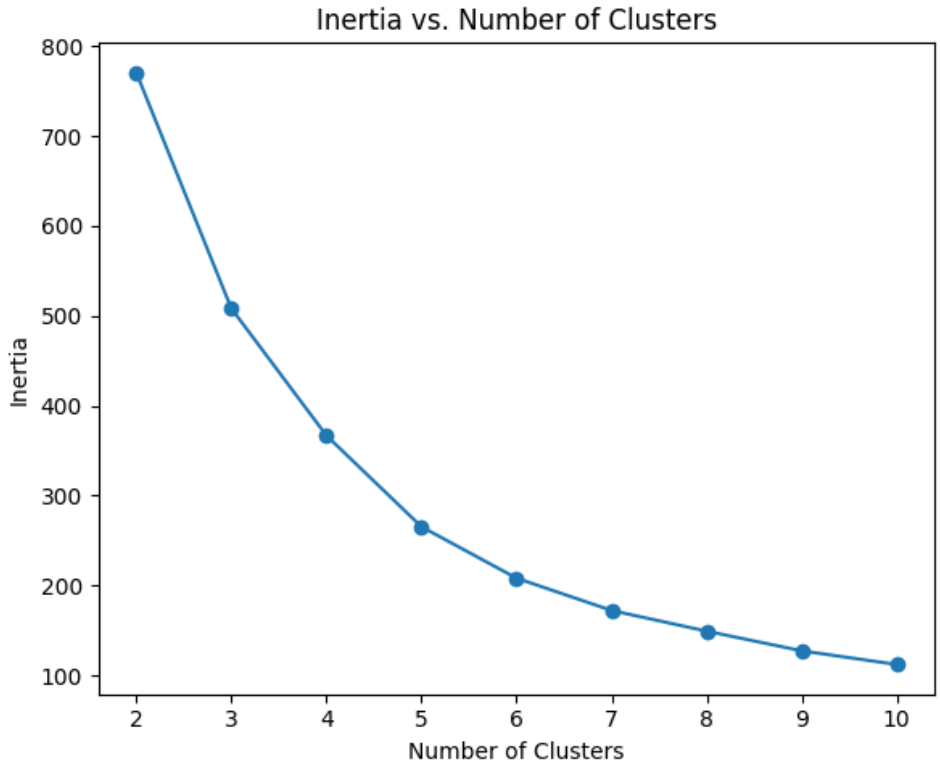
Galime matyti, kad kuo didesnis klasterių skaičius, tuos geresnis inercijos įvertinimas. Tačiau vertinant silueto įvertinimą, geriausias rezultatas pasiektas naudojant 4 klasterius.



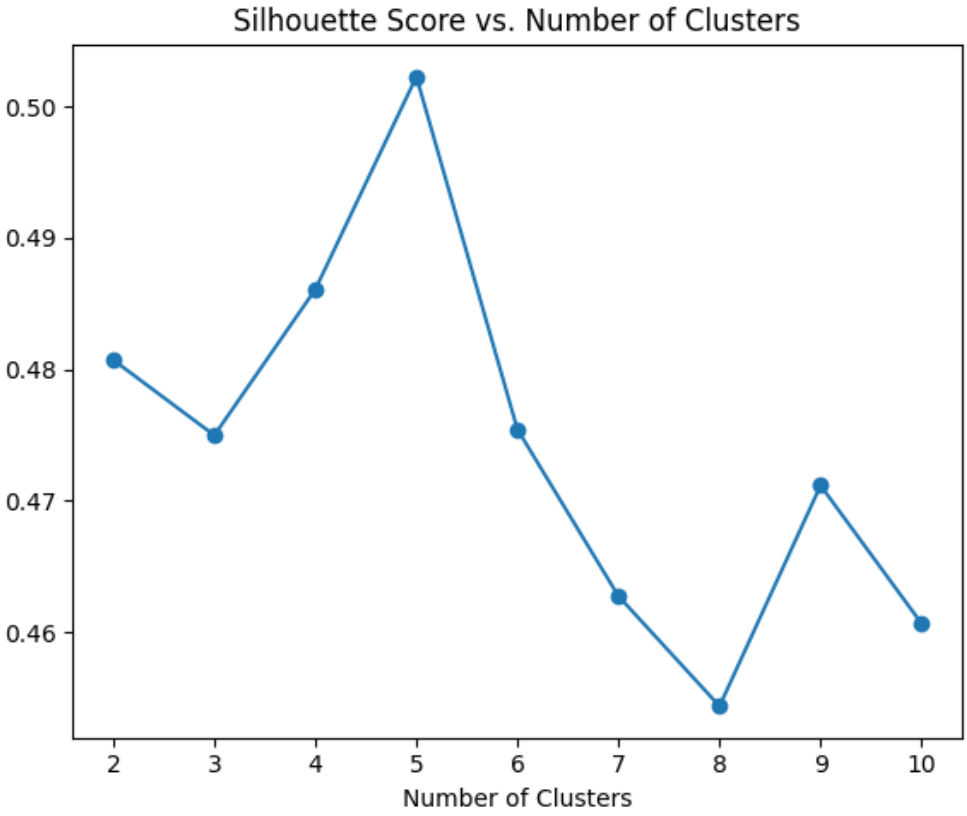
4 pav. „Vcost“ ir „travel“ klasterių diagrama

Toliau skaičiavimams naudosi ,,vcost'' ir ,,gcost'' stulpelius.

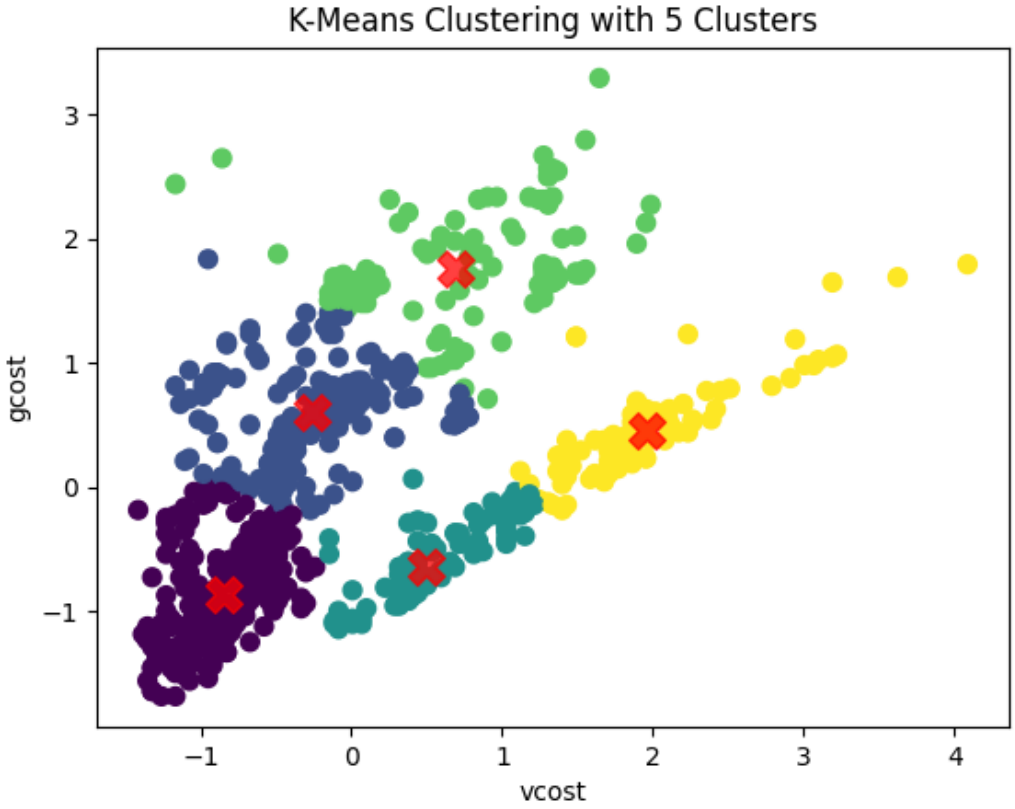
2 Clusters. Inertia: 770.3834177714408
2 Clusters. Silhouette Score: 0.48071045176553856
3 Clusters. Inertia: 508.1250939462858
3 Clusters. Silhouette Score: 0.474976384532771
4 Clusters. Inertia: 366.789794819393
4 Clusters. Silhouette Score: 0.486079905797722
5 Clusters. Inertia: 265.30392717675915
5 Clusters. Silhouette Score: 0.502264031161918
6 Clusters. Inertia: 208.16002757327163
6 Clusters. Silhouette Score: 0.4754271964736103
7 Clusters. Inertia: 172.212930595586
7 Clusters. Silhouette Score: 0.4627381814139474
8 Clusters. Inertia: 149.21861473831265
8 Clusters. Silhouette Score: 0.4543681601327612
9 Clusters. Inertia: 127.3459888234211
9 Clusters. Silhouette Score: 0.4711577194808689
10 Clusters. Inertia: 112.11564661104838
10 Clusters. Silhouette Score: 0.46065206419312316



5 pav. „Vcost“ ir „gcost“ inercijos diagrama pagal klasterių kiekį



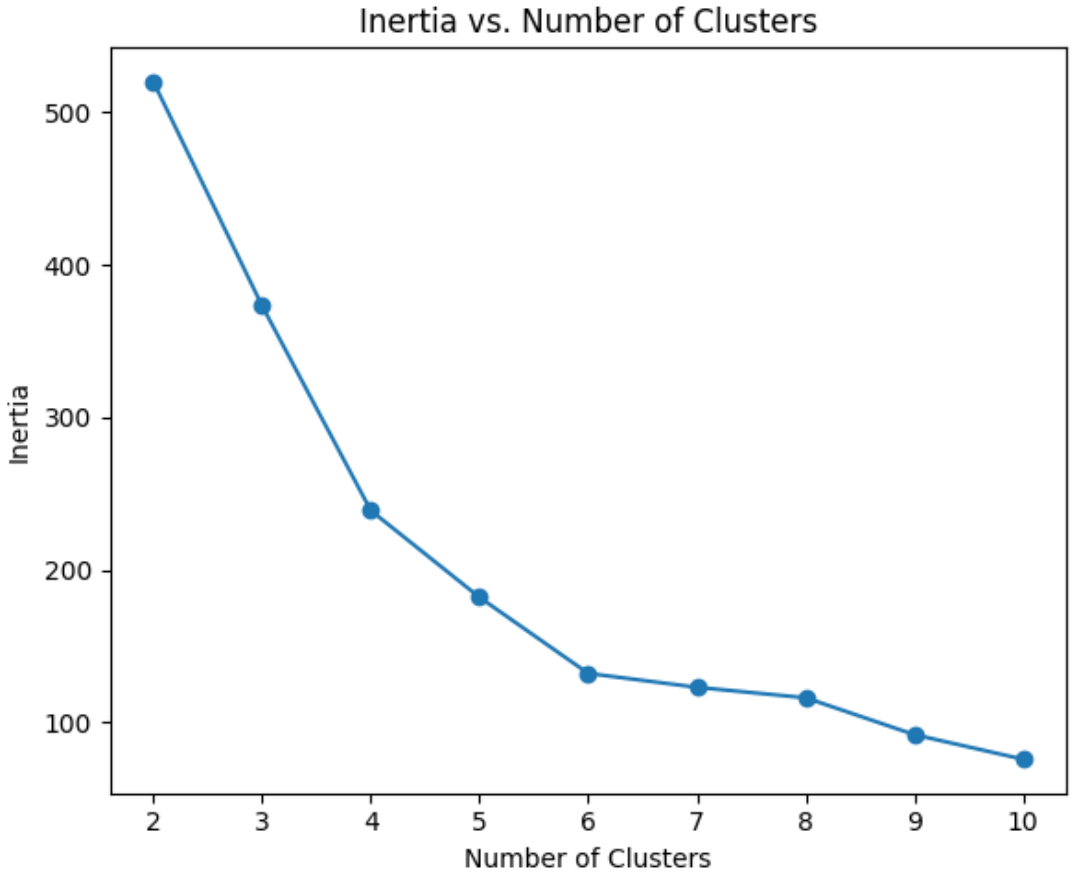
6 pav. „Vcost“ ir „gcost“ Silueto diagrama pagal klasterių kiekį



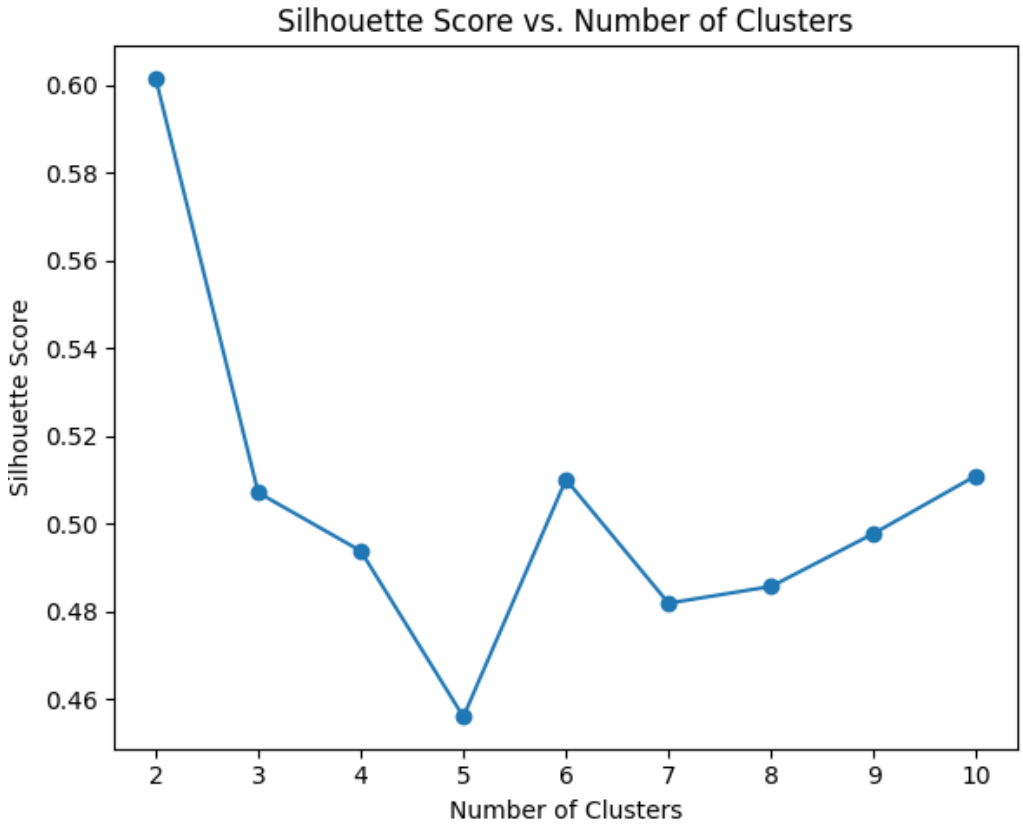
7 pav. „Vcost“ ir „gcost“ klasterių diagrama

Toliau skaičiavimams naudosime ,,gcost'' ir ,,travel'' stulpelius

2 Clusters. Inertia: 520.2219857799203
2 Clusters. Silhouette Score: 0.6015912697800699
3 Clusters. Inertia: 373.34147501897144
3 Clusters. Silhouette Score: 0.5070898397695399
4 Clusters. Inertia: 239.1090024225856
4 Clusters. Silhouette Score: 0.4937811057991685
5 Clusters. Inertia: 182.17441358579168
5 Clusters. Silhouette Score: 0.4560077289727085
6 Clusters. Inertia: 132.29600863520812
6 Clusters. Silhouette Score: 0.5100792186743618
7 Clusters. Inertia: 123.17864419477641
7 Clusters. Silhouette Score: 0.4818829265553181
8 Clusters. Inertia: 116.32235712442375
8 Clusters. Silhouette Score: 0.4857029336543623
9 Clusters. Inertia: 92.13391166331219
9 Clusters. Silhouette Score: 0.49777272818166757
10 Clusters. Inertia: 76.00933437914908
10 Clusters. Silhouette Score: 0.5109895718536148



8 pav. „Gcost“ ir „travel“ inercijos diagrama pagal klasterių kiekį



9 pav. „Gcost“ ir „travel“ Silueto diagrama pagal klasterių kiekį

10 pav. „Gcost“ ir „travel“ klasterių diagrama

4. SOM

4.1. SOM (Self-Organizing-Map) algoritmo aprašymas

Saviorganizuojantis tinklas (SOM) yra neprižiūrimo mokymosi DNT – naudoja duomenis ir sprendžia uždavinius, kai nėra žinoma išvestis. Yra siekiama surasti parametrus, kuriais remiantis būtų galima klasterizuoti duomenis pagal pasirinktą klasterių skaičių. Panašūs taškai yra atvaizduojami šalia vienas kito SOM tinkle. Šis algoritmas leidžia lengviau suprasti ir vizualizuoti duomenis.

Veikimo etapai:

- Sugeneruojami pradiniai svorių vektoriai, priklausomai nuo klasterių skaičiaus;
- Parenkamas pavyzdinis vektorius iš įvesties duomenų;
- Kiekviena žemėlapio viršūnė yra tikrinama, kurios iš jų svoriai yra panašiausi į pavyzdinio vektoriaus svorius. Naudojamas Euklido atstumo kvadratas tam nustatyti;
- Išrenkamas laimėtojas BMU (best matching unit);
- Apskaičiuojamas BMU kaimynystės spindulys;
- Visi neuronai pažymėti kaip kaimynai yra „apdovanojami “ - kiekvienos kaimyninės viršūnės svoriai yra modifikuojami, siekiant padidinti panašumą į pavyzdinį vektorių;
- Visi žingsniai yra kartojami nuo antrojo punkto, keičiant pavyzdinį vektorių kitu. Išbandžius visus įvesties vektorius, visi etapai yra kartojami n iteracijų skaičių (pvz: 1000).

SOM algoritmo kokybės nustatymui bus naudojami du kriterijai: inercijos ir Silueto koeficientai. Inercijos koeficientas yra rodiklis, kaip toli nutolę vienas nuo kito yra klasterio taškai. Ji skaičiuoja atstumų sumą nuo kiekvieno taško klasteryje iki centroido. Jei inercijos pokytis pasidaro labai mažas galima nutraukti skaičiavimus, nes klasterių skaičius nebedaro reikšmingos įtakos algoritmo veikimo teisingumui. Silueto koeficientas nurodo kaip taškai viename klasteryje yra nutolę nuo taškų kitame klasteryje. Kuo šis koeficientas yra didesnis, tuo duomenys yra geriau klasterizuoti.

4.2. SOM algoritmo skaičiavimai

Šiam algoritmui įgyvendintas buvo naudota „Python“ programavimo kalba. Siekta suklasterizuoti ir vizualizuoti duomenis. Kiekvienas taškas dvimačiame žemėlapyje turi savo svorius. Viso proceso metu kiekvienam neuronui yra randamas geriausias BMU. Jis ir jo kaimynų svoriai yra koreguojami, kad taptų artimesni įvesties vektoriui.

Iteracijų skaičius: 1000.

Programinis kodas:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import silhouette_score

# Self-Organizing Map (SOM) class
class SOM:
```

```
    def __init__(self, m, n, dim, num_iterations=100, alpha=0.5):
        self.m = m  # Grid height
        self.n = n  # Grid width
        self.dim = dim  # Dimensionality of input data
        self.num_iterations = num_iterations
        self.alpha = alpha  # Initial learning rate
        self.weights = np.random.rand(m, n, dim)  # Initialize weights

    def _neighborhood_function(self, distance, radius):
        return np.exp(-distance**2 / (2 * radius**2))

    def _update_weights(self, x, bmu_idx, iteration): # Klasteriu svoriai
optimizuojami
        learning_rate = self.alpha * (1 - iteration / self.num_iterations) # a

        radius = np.exp(-iteration / self.num_iterations) # sigma
        for i in range(self.m):
            for j in range(self.n):
                neuron = np.array([i, j])
                bmu = np.array(bmu_idx)
                dist = np.linalg.norm(neuron - bmu)
                if dist <= radius:
                    influence = self._neighborhood_function(dist, radius) # h
                    self.weights[i, j, :] += influence * learning_rate * (x -
self.weights[i, j, :]) #Pakeiciamas esamas ir gretimi svoriai

    def train(self, data):
        for iteration in range(self.num_iterations):
            for x in data:
                distances = np.linalg.norm(self.weights - x, axis=-1)
                bmu_idx = np.unravel_index(np.argmin(distances), (self.m,
self.n)) # Geriausias esamas klasteriu svoris
                self._update_weights(x, bmu_idx, iteration)

    def map_data(self, data):
        mapped_data = []
        for x in data:
            distances = np.linalg.norm(self.weights - x, axis=-1)
            bmu_idx = np.unravel_index(np.argmin(distances), (self.m, self.n))
            mapped_data.append(bmu_idx)
        return mapped_data

    def get_centroids(self):
        centroids = self.weights.reshape(-1, self.weights.shape[-1])
        return centroids

# Load the dataset
data = pd.read_csv("Normalizuoti_duomenys.csv")
```

```
data = data.head(1000)

# Select the attributes for clustering
attributes = ['wait', 'vcost', 'travel', 'gcost']



# Loop through each pair of attributes
for i in range(len(attributes)):
    for j in range(i + 1, len(attributes)):
        X = data[[attributes[i], attributes[j]]].values

        # Store inertia and silhouette scores for different grid sizes
        inertias = []
        silhouette_scores = []
        inertia_changes = []

        for grid_size in range(1, 11):  # Klasteriu skaicius
            som = SOM(m=1, n=grid_size, dim=2, num_iterations=100, alpha=0.5)
            som.train(X)
            mapped_data = som.map_data(X)

            # Calculate inertia
            inertia = sum(np.min(np.linalg.norm(som.weights - x, axis=-1))**2
for x in X) #Sum of best mathcing units
            inertias.append(inertia)

            # Calculate silhouette score if there are at least 2 clusters
            if grid_size > 1:
                labels = [x[0] * grid_size + x[1] for x in mapped_data]
                silhouette = silhouette_score(X, labels) # how similar an object
is to its own cluster compared to other clusters.
                silhouette_scores.append(silhouette)
                print(f"Inertia for {grid_size} clusters: {inertia}")
                print(f"Silhouette Score for {grid_size} clusters:
{silhouette}")
            else:
                silhouette_scores.append(-1)  # Placeholder for 1 cluster case

            # Calculate inertia change
            if grid_size > 1:
                inertia_change = (inertias[-2] - inertias[-1]) / inertias[-2]
                inertia_changes.append(inertia_change)
                if inertia_change < 0.02:  # Stopping criterion
                    print(f"Stopping at grid size {grid_size} due to small
inertia change: {inertia_change:.4f}")
                    break
            else:
```

```
                inertia_changes.append(None)

        # Plot inertia vs number of clusters
        plt.figure(figsize=(10, 6))
        plt.plot(range(1, len(inertias) + 1), inertias, marker='o', linestyle='-
-')
        plt.xlabel('Number of clusters (Grid size)')
        plt.ylabel('Inertia')
        plt.title('Inertia and Number of Clusters')
        plt.grid(True)
        plt.show()

        # Plot silhouette score vs number of clusters, skipping 1 cluster case
        plt.figure(figsize=(10, 6))
        plt.plot(range(2, len(silhouette_scores) + 1), silhouette_scores[1:],
marker='o', linestyle='--')
        plt.xlabel('Number of clusters')
        plt.ylabel('Silhouette Score')
        plt.title('Silhouette Score and Number of Clusters')
        plt.grid(True)
        plt.show()

        # Plot inertia change vs number of clusters
        plt.figure(figsize=(10, 6))
        plt.plot(range(2, len(inertia_changes) + 1), inertia_changes[1:],
marker='o', linestyle='--')
        plt.xlabel('Number of clusters')
        plt.ylabel('Inertia Change')
        plt.title('Inertia Change and Number of Clusters')
        plt.grid(True)
        plt.show()

        # Plot the SOM clusters for the best grid size (highest silhouette
score)
        best_grid_size = np.argmax(silhouette_scores[1:]) + 2
        som = SOM(m=1, n=best_grid_size, dim=2, num_iterations=100, alpha=0.5)
        som.train(X)
        mapped_data = som.map_data(X)
        centroids = som.get_centroids()
        labels = [x[0] * best_grid_size + x[1] for x in mapped_data]
        best_silhouette = silhouette_score(X, labels)

        plt.figure(figsize=(10, 6))
        plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis', s=50, alpha=0.5)
        plt.scatter(centroids[:, 0], centroids[:, 1], marker='x', c='red',
s=200, label='Centroids')
        plt.title(f'SOM Clustering: {attributes[i]} vs {attributes[j]} (Best
claster number:  {best_grid_size})')
```

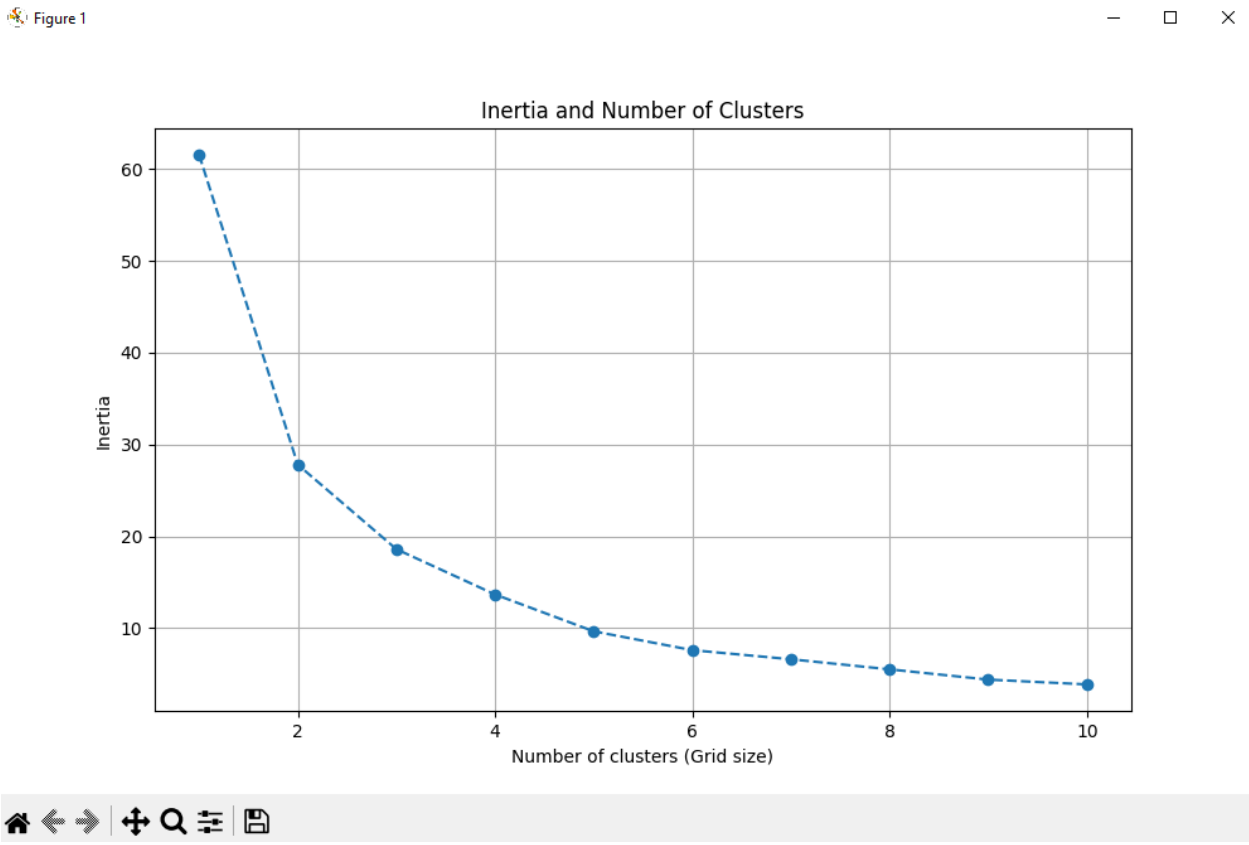
```
        plt.xlabel(attributes[i])
        plt.ylabel(attributes[j])
        plt.legend()
        plt.grid(True)
        plt.text(0.05, 0.95, f'Silhouette Score: {best_silhouette:.2f}',
transform=plt.gca().transAxes, fontsize=12, verticalalignment='top')
        plt.show()
```

„Vcost“-„gcost“ įvesties duomenys

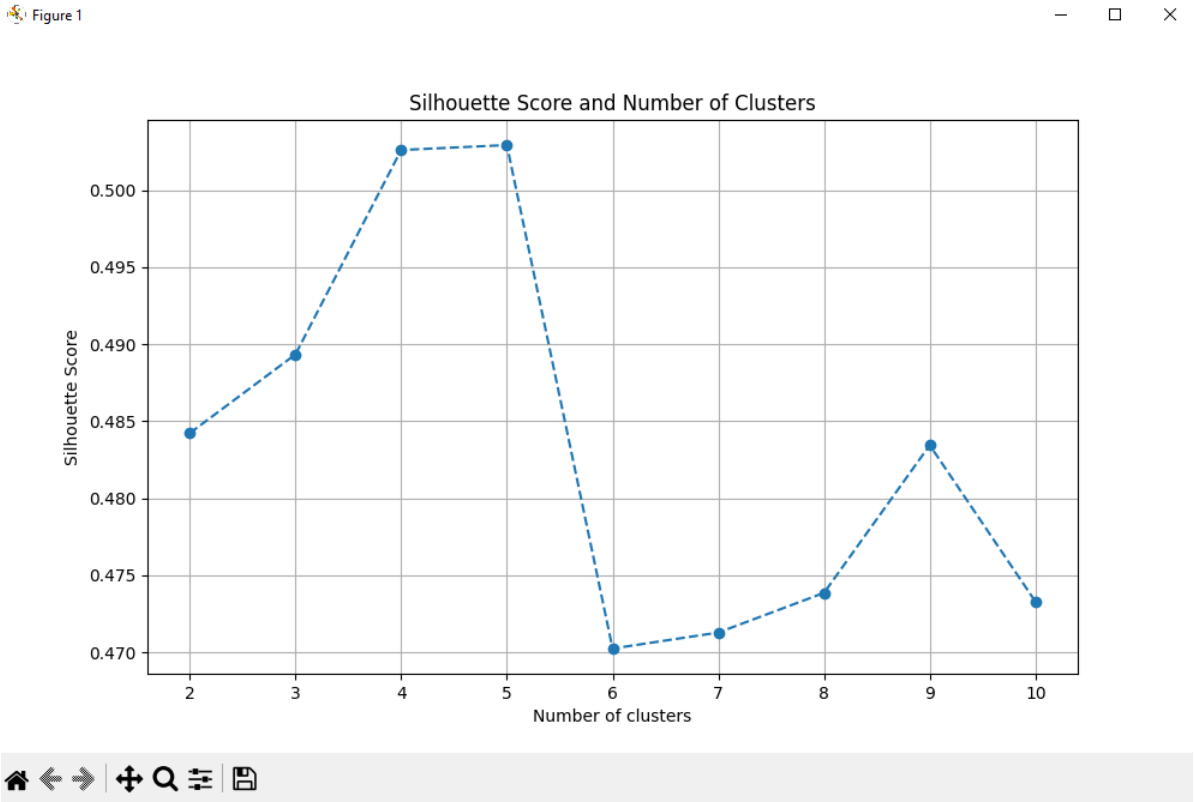
Galima pastebėti, kad geriausias klasterių skaičius bus 5, nes Silueto įvertis tokiu atveju yra didžiausias.

Skaitinės inercijos ir Silueto koeficientų reikšmės:

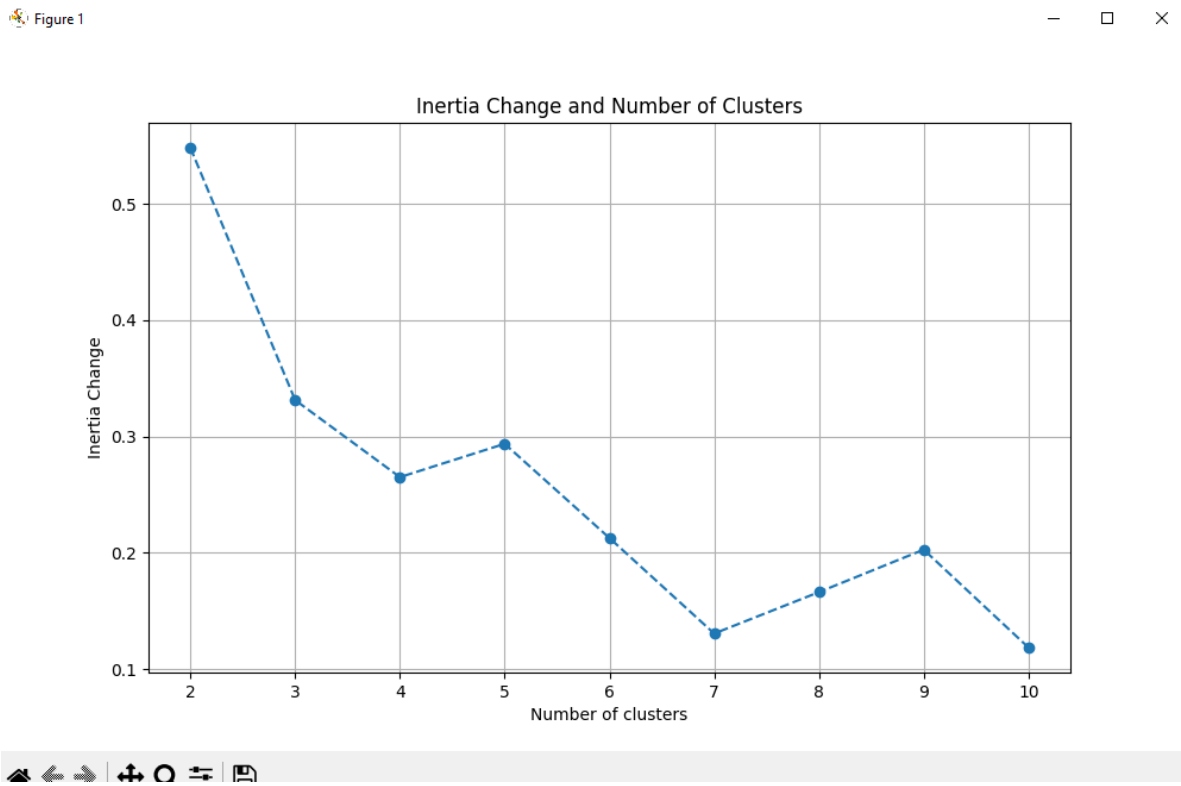
Inertia for 2 clusters: 27.814648924353637
Silhouette Score for 2 clusters: 0.4842251437140644
Inertia for 3 clusters: 18.601770410653785
Silhouette Score for 3 clusters: 0.48933616667864416
Inertia for 4 clusters: 13.672651197623534
Silhouette Score for 4 clusters: 0.5026041585744334
Inertia for 5 clusters: 9.657711055367379
Silhouette Score for 5 clusters: 0.5029254870926345
Inertia for 6 clusters: 7.603882154268014
Silhouette Score for 6 clusters: 0.47024385451466016
Inertia for 7 clusters: 6.611555821424457
Silhouette Score for 7 clusters: 0.4712836226309943
Inertia for 8 clusters: 5.513185989242345
Silhouette Score for 8 clusters: 0.4738595752213782
Inertia for 9 clusters: 4.395950289190671
Silhouette Score for 9 clusters: 0.48344499176749106
Inertia for 10 clusters: 3.875156037620725
Silhouette Score for 10 clusters: 0.4732886397455622



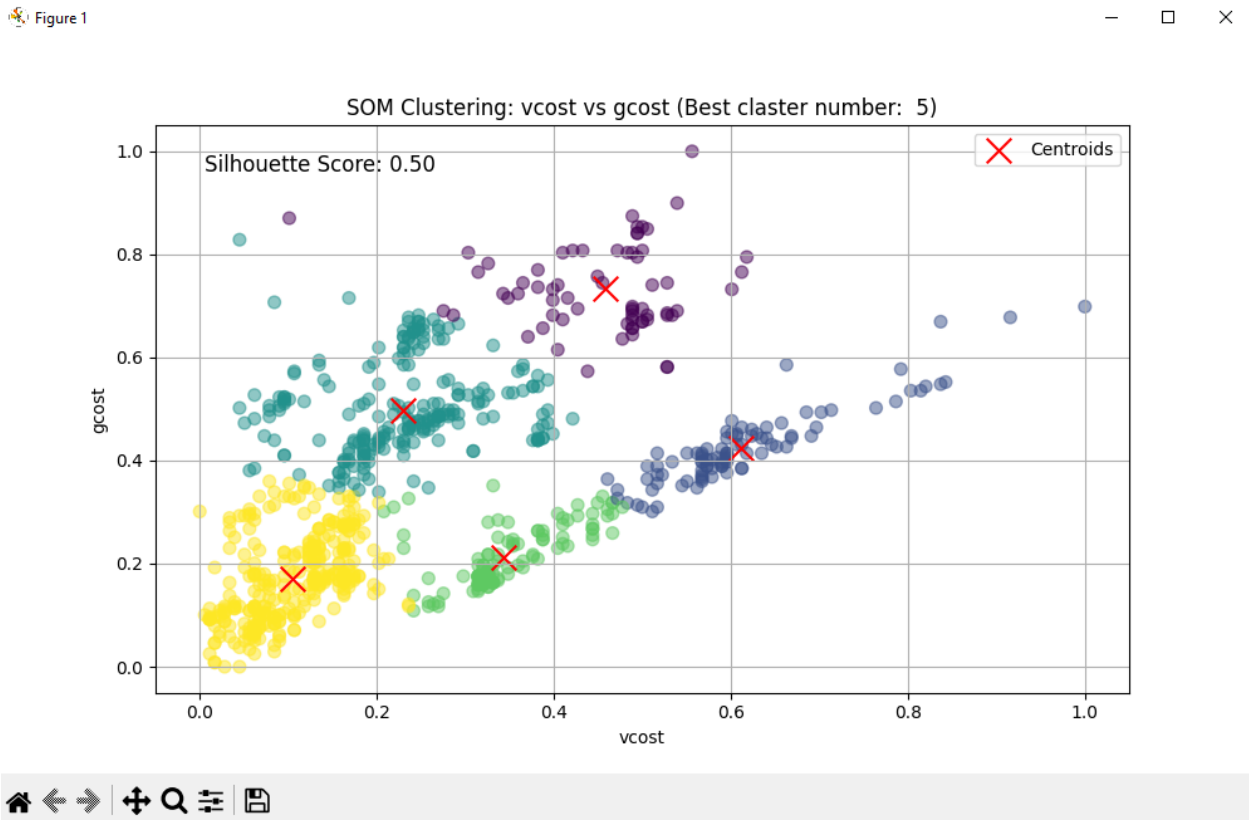
11 pav. „Gcost“ ir „Vcost“ inercijos diagrama pagal klasterių kiekį



12 pav. „Gcost“ ir „Vcost“ Silueto diagrama pagal klasterių kiekį



13 pav. „Vcost“ ir „gcost“ inercijos pokyčio diagrama pagal klasterių kiekį



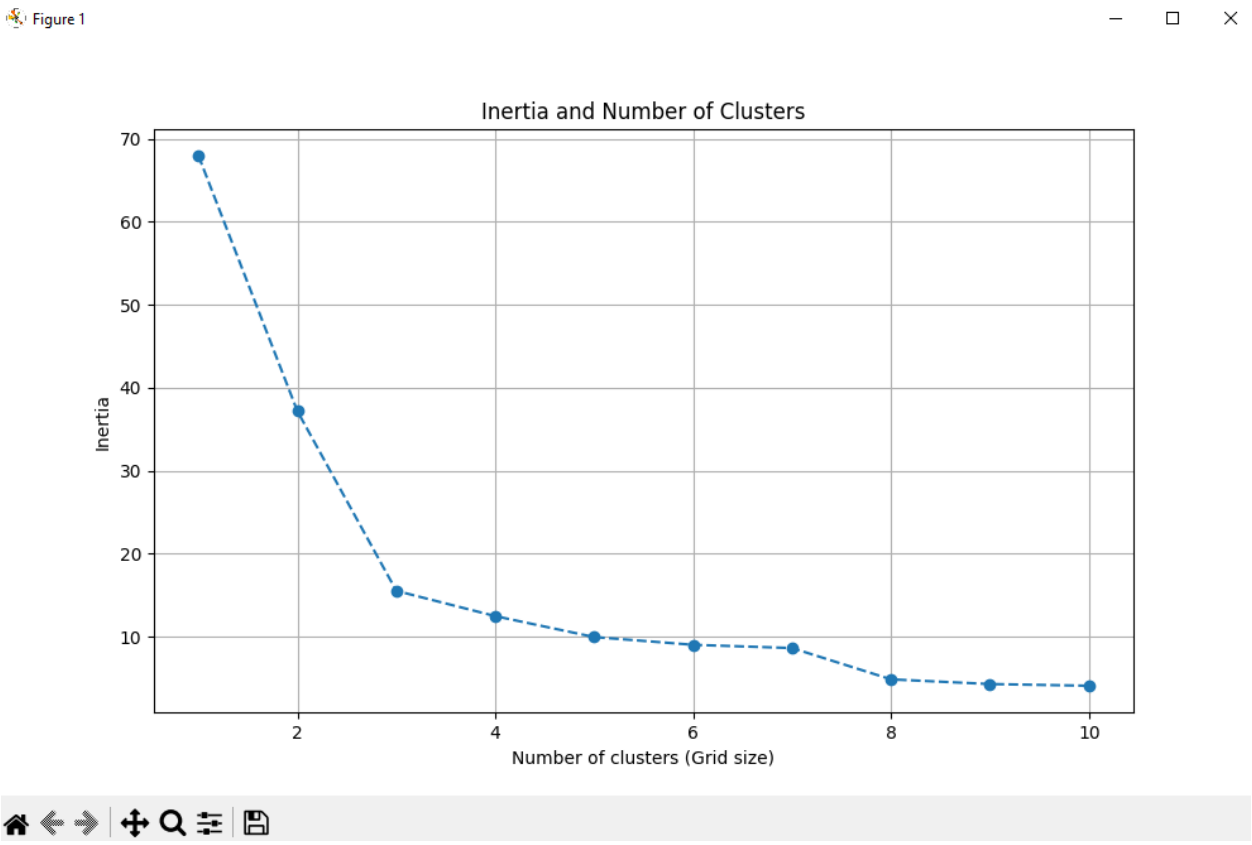
14 pav. „Gcost“ ir „Vcost“ Klasterių diagrama

„Travel“-„vcost“ įvesties duomenys

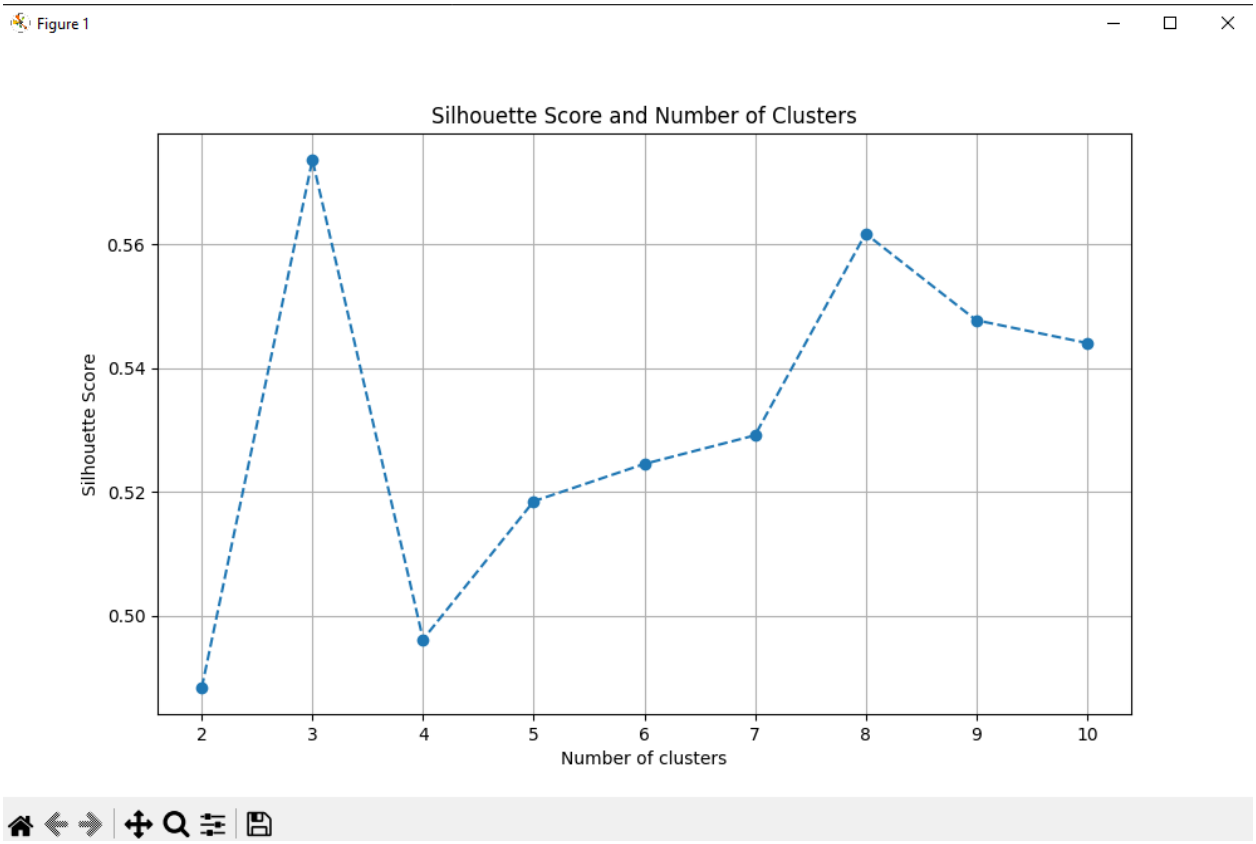
Iš gautų rezultatų matosi, kad geriausias klasterių skaičius yra 3.

Skaitinės inercijos ir Siluteto koeficientų reikšmės:

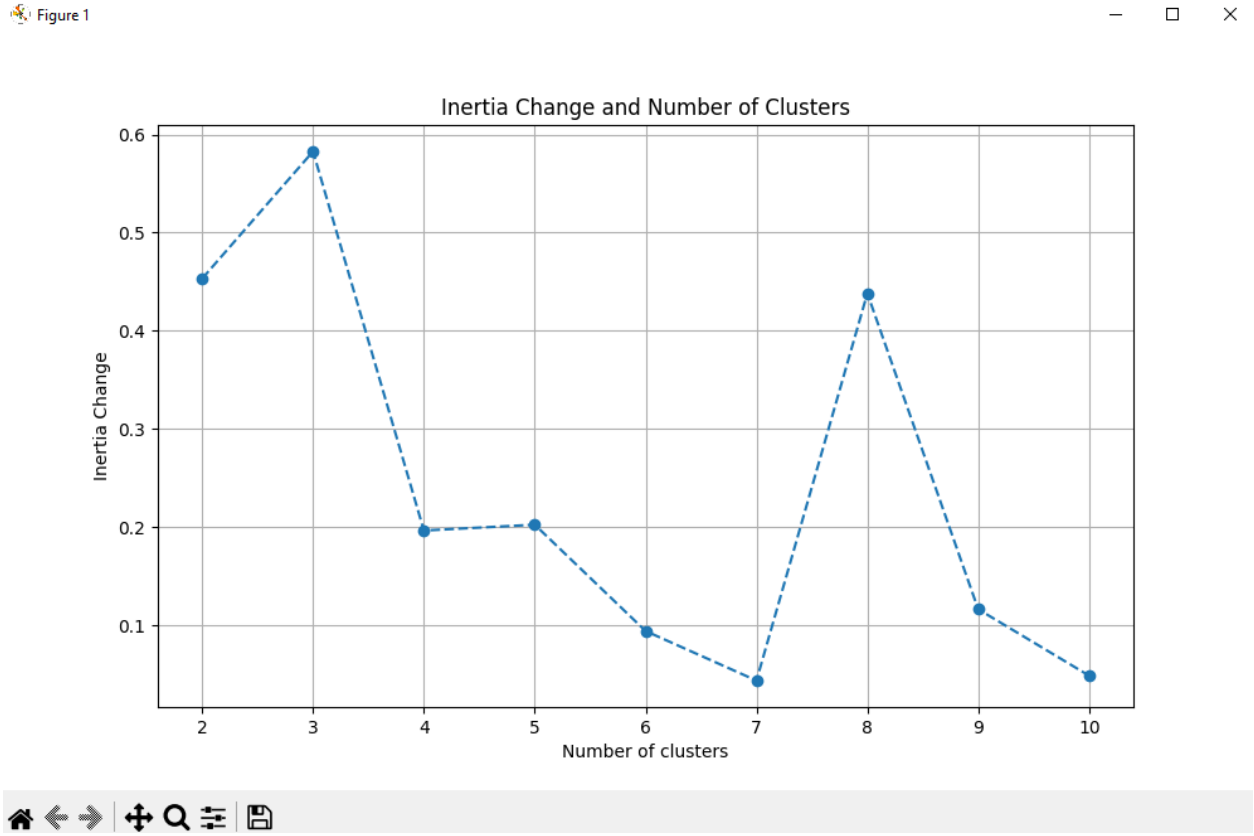
Inertia for 2 clusters: 37.1614745154546
Silhouette Score for 2 clusters: 0.4883504930608866
Inertia for 3 clusters: 15.508217725780264
Silhouette Score for 3 clusters: 0.5736134423114557
Inertia for 4 clusters: 12.461498623032957
Silhouette Score for 4 clusters: 0.4961496155516904
Inertia for 5 clusters: 9.939136747976901
Silhouette Score for 5 clusters: 0.5184589105994792
Inertia for 6 clusters: 9.004196403267866
Silhouette Score for 6 clusters: 0.5244886519023818
Inertia for 7 clusters: 8.6114577684303
Silhouette Score for 7 clusters: 0.5291065358953788
Inertia for 8 clusters: 4.838815294069977
Silhouette Score for 8 clusters: 0.5616972177810303
Inertia for 9 clusters: 4.276202734241152
Silhouette Score for 9 clusters: 0.5477031703795685
Inertia for 10 clusters: 4.068325627865212
Silhouette Score for 10 clusters: 0.5440303270764552



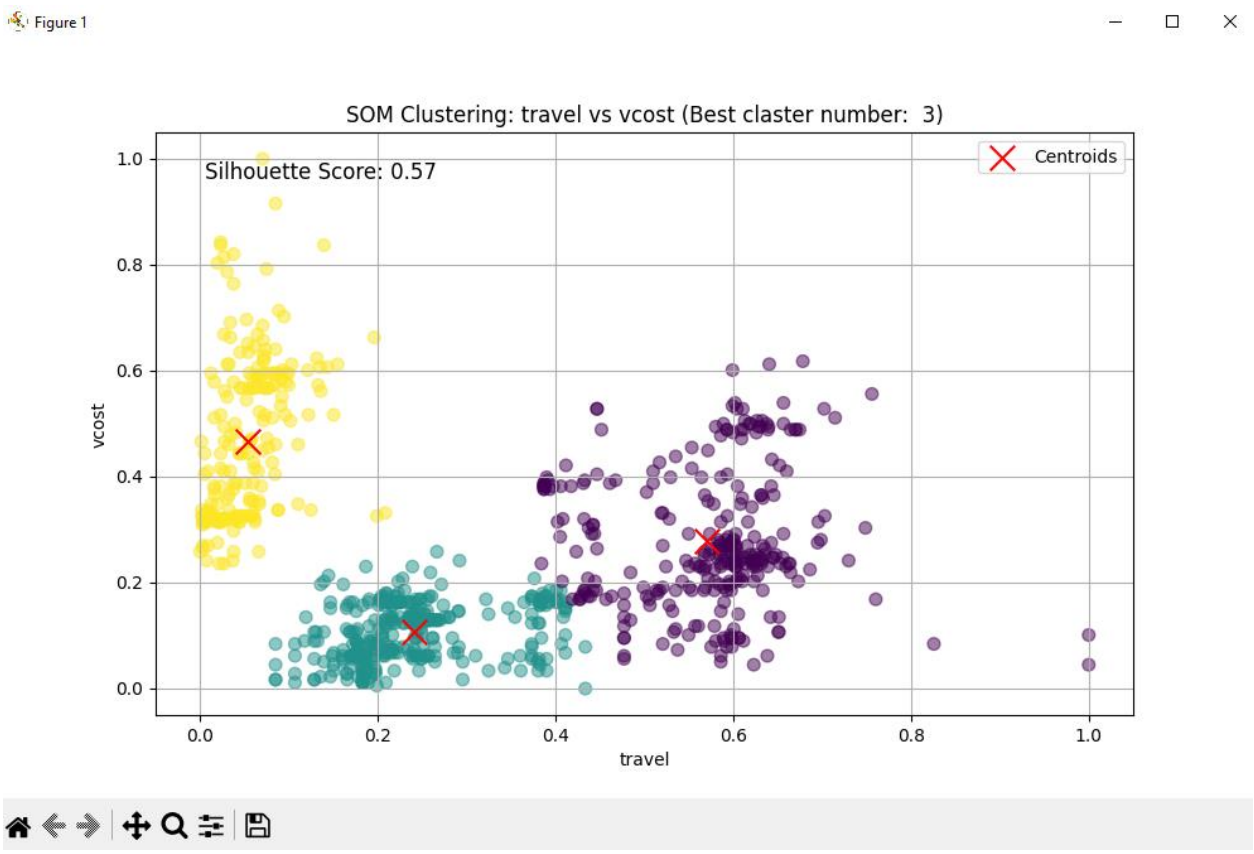
pav 15 „Travel“ ir „Vcost“ Inercijos diagrama pagal klasterių kiekį



pav 16 „Travel“ ir „Vcost“ Silueto diagrama pagal klasterių kiekį



17 pav. „Travel“ ir „Vcost“ Inercijos pokyčio diagrama pagal klasterių kiekį



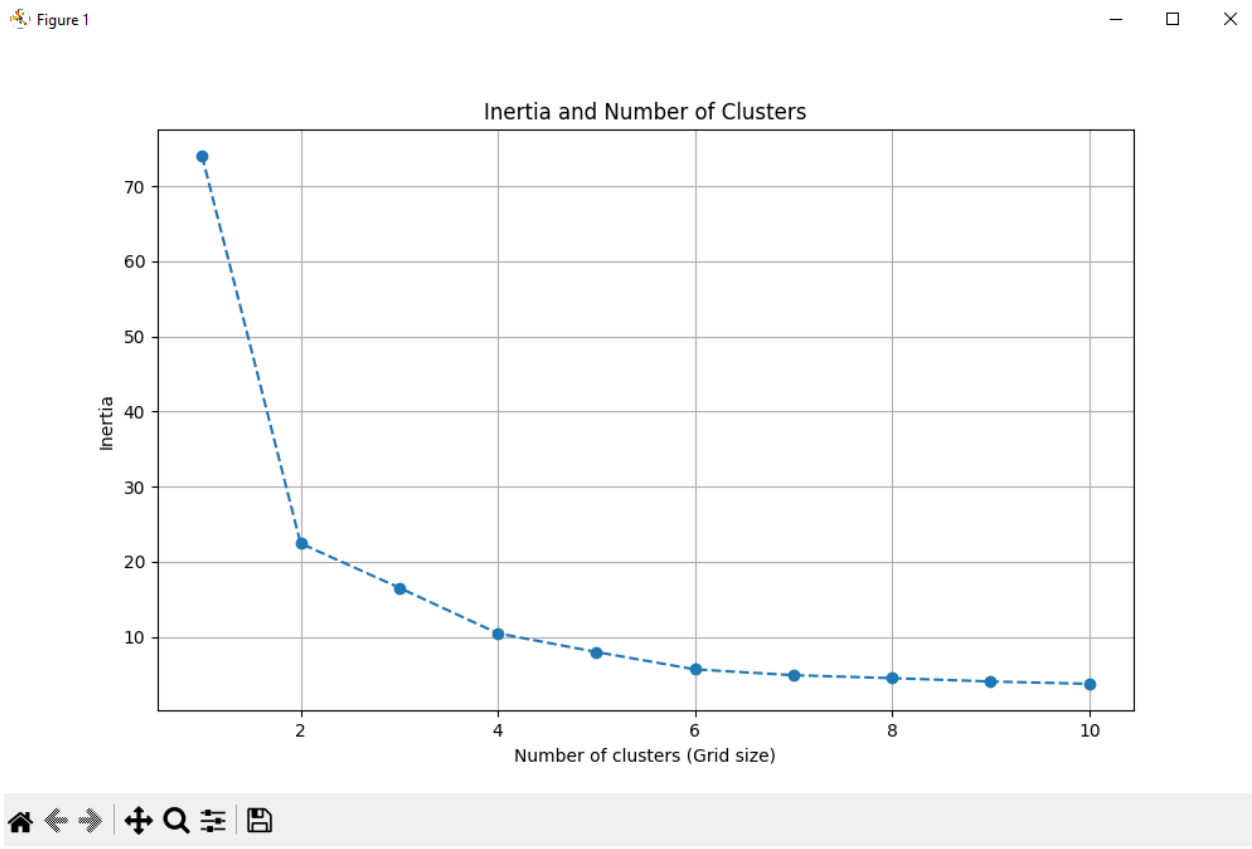
18 pav. „Travel“ ir „Vcost“ Klasterių diagrama

„Travel“-„gcost“ įvesties duomenys

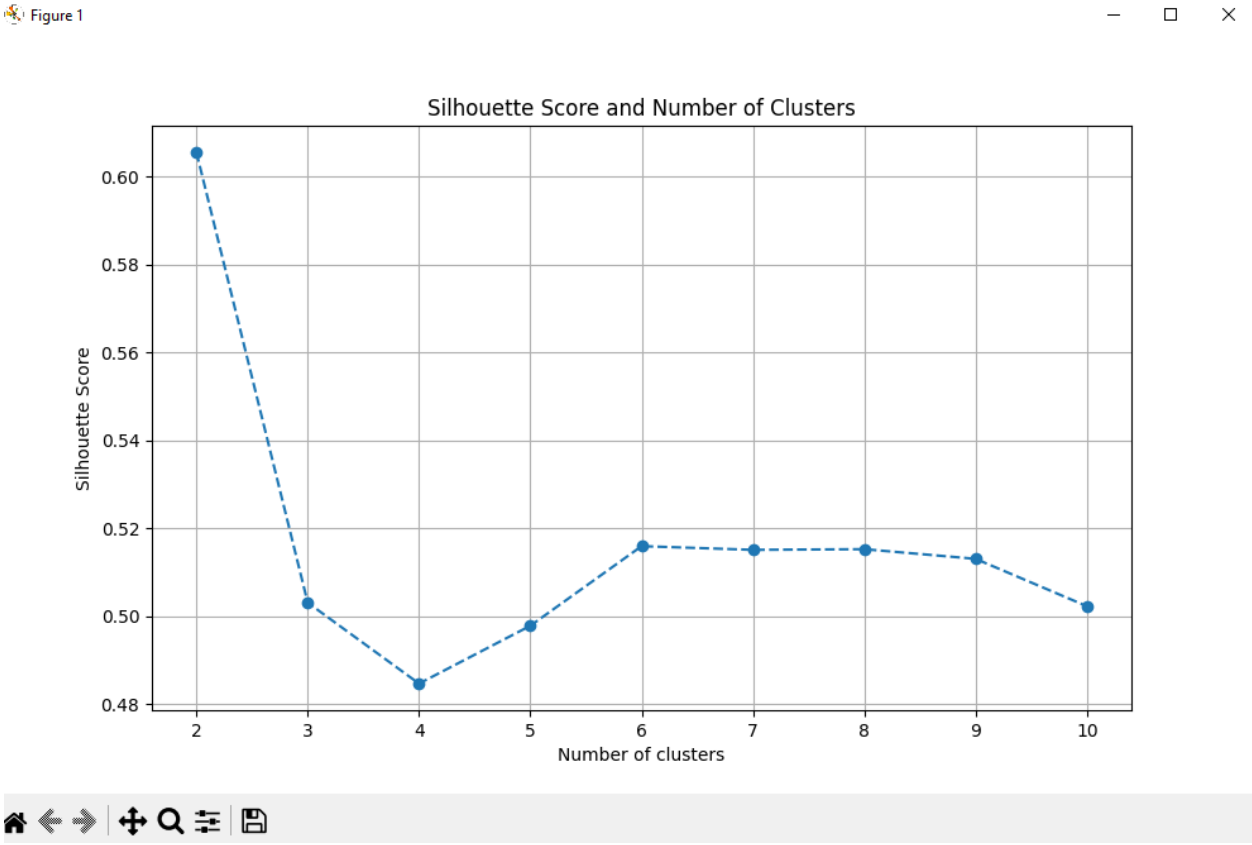
Iš gautų rezultatų matyti, kad geriausias klasterių skaičius yra 2.

Skaitinės inercijos ir Silueto koeficientų reikšmės:

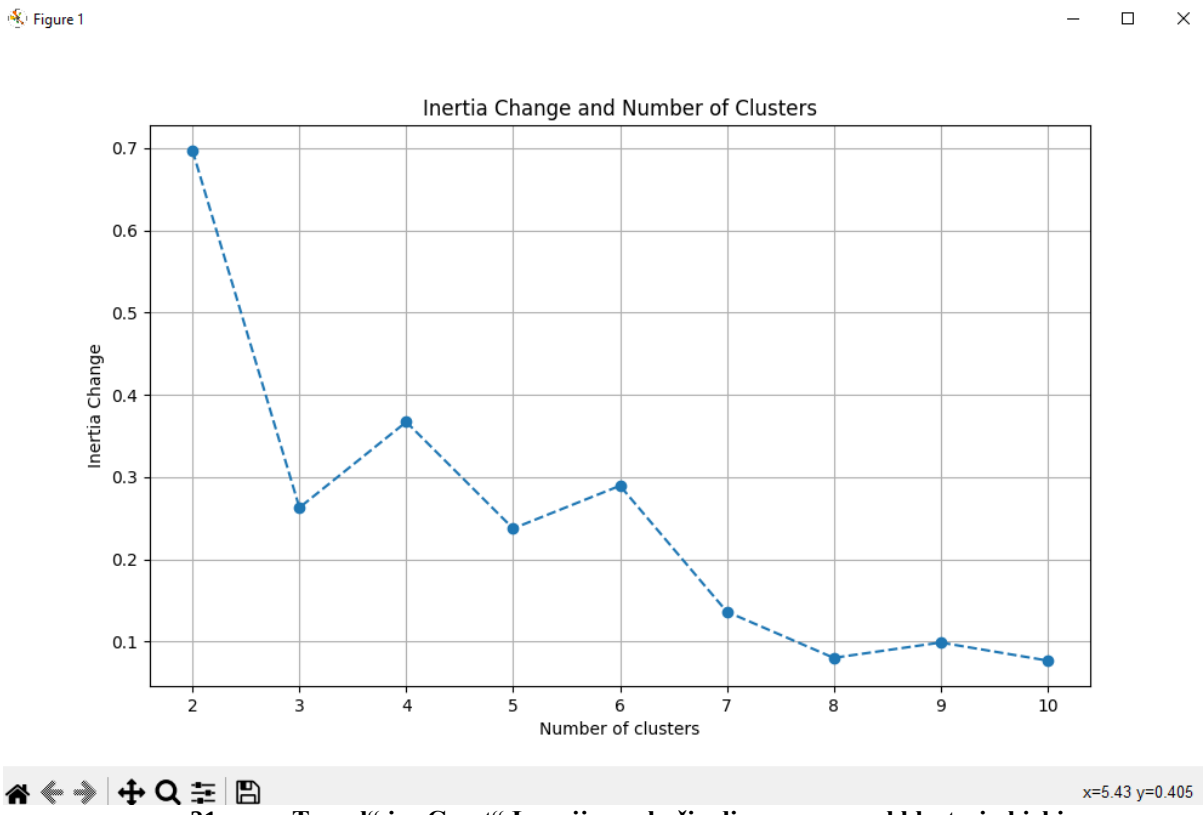
Inertia for 2 clusters: 22.445035989686083
Silhouette Score for 2 clusters: 0.605605624425065
Inertia for 3 clusters: 16.546232255184268
Silhouette Score for 3 clusters: 0.5030747611675253
Inertia for 4 clusters: 10.473880326097412
Silhouette Score for 4 clusters: 0.4846368959816691
Inertia for 5 clusters: 7.986991085863723
Silhouette Score for 5 clusters: 0.4978038092907644
Inertia for 6 clusters: 5.675235570689421
Silhouette Score for 6 clusters: 0.5159328100510354
Inertia for 7 clusters: 4.901818218131294
Silhouette Score for 7 clusters: 0.5150880452245894
Inertia for 8 clusters: 4.508711022003197
Silhouette Score for 8 clusters: 0.5152405793071706
Inertia for 9 clusters: 4.06311056131773
Silhouette Score for 9 clusters: 0.513042956838019
Inertia for 10 clusters: 3.7505521183194643
Silhouette Score for 10 clusters: 0.5022612510610255



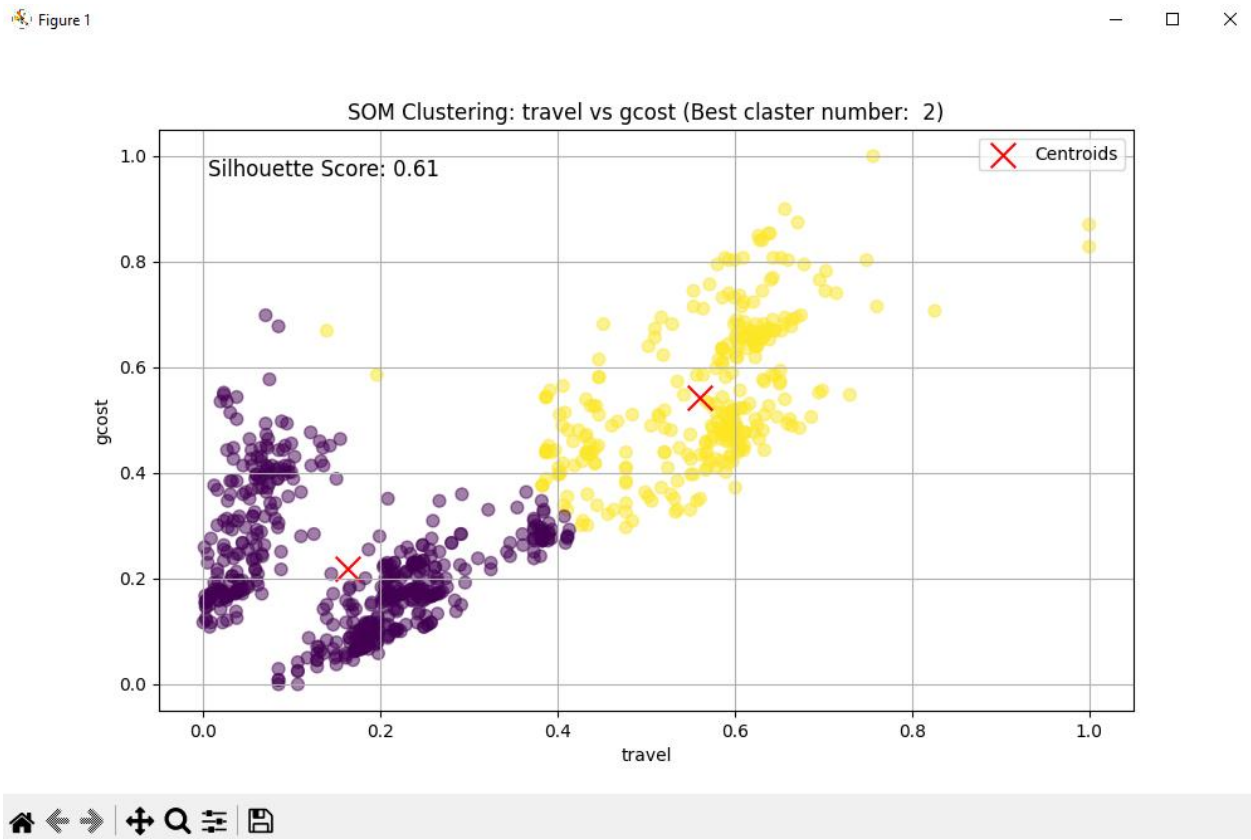
19 pav. „Travel“ ir „Gcost“ Inercijos diagrama pagal klasterių kiekį



20 pav. „Travel“ ir „Gcost“ Silueto diagrama pagal klasterių kiekį



21 pav. „Travel“ ir „Gcost“ Inercijos pokyčio diagrama pagal klasterių kiekį



22 pav. „Travel“ ir „Gcost“ Klasterių diagrama

5. Palyginimas

„Vcost“-„gcost“ įvesties duomenys

	K-vidurkių	SOM
Geriausias Silueto koeficientas	0.502	0.502
Klasterių skaičius	5	5
Darbo laikas (sec.)	0.774	41.119

„Travel“-„vcost“ įvesties duomenys		
	K-vidurkių	**SOM**
Geriausias Silueto koeficientas	0.560	0.573
Klasterių skaičius	4	3
Darbo laikas (sec.)	0.860	31.262

„Travel“-„gcost“ įvesties duomenys		
	K-vidurkių	**SOM**
Geriausias Silueto koeficientas	0.601	0.605
Klasterių skaičius	2	2
Darbo laikas (sec.)	0.825	25.852

6. Išvados

Apibendrinant galima teigti, kad duomenų rinkinys yra tinkamas klasterizavimui atlikti. Pagal gautą Silueto koeficientą geriausi pradinių duomenų atributai yra „travel“ ir „gcost“, kai klasterių skaičius lygus 2. K-vidurkių ir SOM algoritmų gauti rezultatai yra labai panašūs, tačiau vykdymo laikai skiriasi labai stipriai, todėl būtų logiška pasirinkti K-vidurkių algoritmą klasterizavimui.