Vilniaus Universitetas

Dimensijos mažinimas klasifikavime

Darbą atliko:

Vainius Gataveckas, Matas Gaulia, Dovydas Martinkus

Duomenų Mokslas

3 kursas 2 gr.

Vilnius, 2022

**Turinys**

[1 Tikslas ir uždaviniai 3](#_Toc103630304)

[2 Duomenų aibė 4](#_Toc103630305)

[3 Atliktos analizės aprašymas 5](#_Toc103630306)

[3.1 Pradinis apdorojimas 5](#_Toc103630307)

[3.2 Naivus Bajeso klasifikatorius 7](#_Toc103630308)

[3.3 Sprendimų medžio klasifikatorius 8](#_Toc103630309)

[3.4 Atsitiktinio miško klasifikatorius 9](#_Toc103630310)

[3.5 Klasifikavimo kokybės įvertinimas ir modelių palyginimas 10](#_Toc103630311)

[4 Išvados 14](#_Toc103630312)

[5 Šaltiniai 15](#_Toc103630313)

[Priedas 16](#_Toc103630314)

# Tikslas ir uždaviniai

Tikslas: Naudojant pasirinktą duomenų aibę ištirti pasirinktus klasifikatorius, įvertinti klasifikavimo kokybę ir tarpusavyje palyginti klasifikavimo metodus.

Uždaviniai:

Pasirinktų klasifikatorių teorinis tyrimas.

Optimalių parametrų parinkimas.

Klasifikavimo šablonų sudarymas.

Klasifikavimo kokybės įvertinimas ir modelių palyginimas.

Apibendrintų išvadų pateikimas.

# Duomenų aibė

Spotify Past Decades Songs duomenų aibė

Duomenų aibės šaltinis: Kaggle

Nuoroda per internetą: <https://www.kaggle.com/cnic92/spotify-past-decades-songs-50s10s?select=1990.csv>

Duomenų aibę sudaro tokie požymiai:

„Number“ – (kategorinis, nominalusis) dainą identifikuojantis kodas

„Title“ – (kategorinis, nominalusis) dainos pavadinimas

„Artist“ – (kategorinis, nominalusis) atlikėjas arba grupė

„Top Genre“ – (kategorinis, nominalusis) dainos žanras

„Year“ – (kiekybinis, diskretusis, intervalinė skalė) išleidimo metai

„Decade“ – (kiekybinis, diskretusis, intervalinė skalė) išleidimo dešimtmetis

„Tempo“ – (kiekybinis, tolydus, santykių skalė) dainos tempas

“Loudness (dB)“ - (kiekybinis, tolydus, intervalų skalė) dainos garsumas

„Duration“ – (kiekybinis, tolydus, santykių skalė) dainos trukmė

„Energy“ – (kiekybinis, tolydus, santykių skalė) dainos energija

„Danceability“ – (kiekybinis, tolydus, santykių skalė) lengvumas šokti pagal dainą

„Liveness“ – (kiekybinis, tolydus, santykių skalė) kaip tikėtina, kad daina yra gyvas įrašas

„Valence“ – (kiekybinis, tolydus, santykių skalė) dainos pozityvumas

„Acousticness“ – (kiekybinis, tolydus, santykių skalė) dainos akustiškumas

„Speechiness“ – (kiekybinis, tolydus, santykių skalė) kiek dainoje yra kalbama

„Popularity“ - (kiekybinis, tolydus, santykių skalė) dainos populiarumas pagal perklausų skaičių

# Atliktos analizės aprašymas

## Pradinis apdorojimas

Klasifikavimui pasirinktas dainų išleidimo dešimtmetis (požymis „Decade“) su dvejomis galimomis reikšmėmis (80-ieji arba 2010-ieji). Klasių pasiskirstymas subalansuotas – duomenų aibę sudaro 100 2010-ųjų dainų ir 96 80-ųjų dainos. Abiems dešimtmečiams apskaičiuotos aprašomosios statistikos (1 ir 2 lentelės).

1 lentelė Aprašomosios statistikos charakteristikos 80-ųjų dainoms

|  |  |  |  |
| --- | --- | --- | --- |
| Požymis | Vidurkis | Standartinis nuokrypis | Mediana |
| tempo | 122.6 | 25.1 | 122.0 |
| energy | 64.9 | 20.2 | 68.0 |
| danceability | 62.3 | 13.4 | 63.0 |
| loudness | -9.1 | 3.7 | -9.0 |
| liveness | 16.4 | 13.9 | 11.0 |
| valence | 63.1 | 25.9 | 70.0 |
| duration | 258.9 | 51.2 | 251.0 |
| acousticness | 24.6 | 21.7 | 19.0 |
| speechiness | 4.4 | 2.8 | 4.0 |
| popularity | 67.7 | 7.3 | 68.0 |

2 lentelė Aprašomosios statistikos charakteristikos 2010-ųjų dainoms

|  |  |  |  |
| --- | --- | --- | --- |
| Požymis | Vidurkis | Standartinis nuokrypis | Mediana |
| tempo | 118.7 | 22.4 | 120.0 |
| energy | 68.0 | 16.3 | 68.0 |
| danceability | 65.4 | 11.9 | 67.0 |
| loudness | -5.5 | 2.0 | -5.0 |
| liveness | 17.9 | 13.8 | 13.0 |
| valence | 46.4 | 20.9 | 47.0 |
| duration | 209.2 | 24.1 | 209.0 |
| acousticness | 14.7 | 17.9 | 9.0 |
| speechiness | 8.4 | 7.8 | 6.0 |
| popularity | 76.0 | 9.2 | 78.0 |

Duomenys padalinti į mokymo ir testavimo aibes naudojant santykį 80-20.

Visų skaitinių požymių matavimo skalės suvienodintos standartizuojant juos pagal vidurkį ir dispersiją. Standartizavimui naudotos vidurkio ir dispersijos reikšmės gautos naudojant mokymo aibę.

## Naivus Bajeso klasifikatorius

Naivaus Bajeso (angl. naive Bayes) klasifikavimo metodas pagrįstas sąlyginės tikimybės modeliu gautu naudojantis Bajeso teorema. Metodas vadinamas „naivuoju“, nes taikoma požymių tarpusavio nepriklausomumo prielaida. Gautą tikimybinį modelį galima užrašyti [1]:

Šis modelis gaunamas iš Bajeso formulės:

Laikant, kad požymiai yra tarpusavyje nepriklausomi:

Kadangi kiekvienai mokymo aibeiyra konstanta, klasifikatorius iš tikimybinio modelio sudaromas pasirenkant klasę pagal:

Turint skaitinius požymius dažniausiai laikoma, kad turi normalujį skirstinį, kurio parametrai kiekvienai klasei įvertinami naudojant didžiausio tikėtinumo metodą. Tuo tarpu gaunamas pagal klasių pasiskirstymą mokymo aibėje.

Modelyje naudojama visų požymių tarpusavio nepriklausomumo sąlyga tikrose duomenų aibėse galioja retai. Daug dažnai pasitaikančių požymių negalima laikyti tarpusavyje nepriklausomais (pvz. asmens ūgis ir svoris). Kai kuriais atvejais tam tikros požymių kombinacijos iš vis nėra įmanomos (vadinasi, tikrai nėra nepriklausomos). Iš tikimybinio modelio taip pat matoma, kad naivaus Bajeso modelis reikalauja sąlygos, kad kiekvienas požymis po lygiai stipriai prisideda prie galutinio modelio sprendimo.

Metodas gerai veikia su mažu duomenų kiekiu, pasižymi greitu veikimu, savaime gali klasifikuoti k>2 klases, nėra stipriai veikiamas išskirčių. Nepaisant modeliui reikalingų prielaidų, tam tikrose taikymo srityse (pvz. dokumentų klasifikavime) šiuo modeliu praktikoje gaunami geri rezultatai. Modeliui nereikia parinkti jokių esminių parametrų.

Geriausi rezultatai ieškoti kryžminės validacijos metodu su *n\_features\_to\_selec*t={0.2,0.4,0.6,0.8,1.0}. Naudojant kryžminę validaciją geriausi rezultatai gauti su reikšme 0.8

## Sprendimų medžio klasifikatorius

Sprendimų medis (angl. decision tree) sukuria klasifikavimo modelius medžio struktūros pavidalu. Duomenų rinkinys suskaidomas į vis mažesnius poaibius, o kartu palaipsniui kuriamas susijęs sprendimų medis. Galutinis rezultatas yra medis su sprendimo mazgais (angl. decision node) ir lapų mazgais (angl. leaf node). Sprendimų medžių konstravimo algoritmai paprastai veikia iš viršaus į apačią kiekviename sprendimo mazge pasirenkant tokį kintamąjį, pagal kurio reikšmes galima geriausiai padalinti duomenų rinkinį pagal tam tikrą naudojamą metriką. Dažnai naudojamos metrikos yra Gini priemaiša (Gini impurity), informacijos išlošis (information gain). Medžio lapo, į kurį pateko klasifikuojamas objektas, reikšmė atitinką modelio priimtą sprendimą [2].

Sprendimų medžiai sugeba lengvai prisitaikyti prie struktūrų, esančių mokymo duomenyse, tačiau rezultatai itin stipriai priklauso nuo to, kokie duomenys turėti mokymo aibėje. Dėl šios priežasties sprendimu medžiu tikėtina gauti stipriai prastesnius rezultatus klasifikuojant prieš tai nematytus stebėjimus (tai vadinamasis low-bias high-variance tipo modelis). Su šia problema susijęs modelio parametrų parinkimas, pavyzdžiui: *max\_depth* kontroliuoja maksimalų medžio gylį *, min\_samples\_split* parametru parenkamas minimalus stebėjimų kiekis, reikalingas norint dar kartą skaidyti duomenų aibę*, min\_samples\_leaf –* minimalus reikiamas stebėjimų skaičius medžio lapuose. Šiuos parametrus siekiama parinkti taip, kad būtų gaunama gera klasifikavimo kokybė, tačiau išvengiama persimokymo (angl. overfitting).

Vienas iš sprendimų medžių privalumų yra beveik nereikalingas pradinis duomenų apdorojimas: Duomenys pateikiami neturi būti vienodoje skalėje, priklausomai nuo metodo implementacijos gali būti pateikiami objektai su praleistomis požymių reikšmėmis, nebūtina perkoduoti kategorinius kintamuosius. Sprendimo medžiai savaime gali atlikti daugiau negu dviejų klasių klasifikavimą. Sprendimų medžiai lengvai suprantami ir interpretuojami, taip pat metode turimas savaiminis požymių parinkimo mechanizmas: šakoms sudaryti požymis naudojamas tik tada, kai juo gautas padalijimas yra geriausias galimas.

Geriausių parametrų rinkinys ieškotas naudojant parametrų tinklelį *max\_depth*={2,3,4}, *min\_samples\_split*={15,20,30}. *n\_features\_to\_select*={0.6,0.8,1.0}

Geriausi rezultatai originalios dimensijos duomenims gauti su *max\_depth*=4 ir *min\_samples\_split*=15, *n\_features\_to\_select*=0.8.

## Atsitiktinio miško klasifikatorius

Metodai, sukonstruojantys daugiau negu vieną medį vadinami ansamblių (angl. ensemble) metodais. Bagging (Bootstrap aggregating) sprendimų medis sudaro kelis sprendimų medžius kiekvieną medį konstruodamas naudodamasis duomenų aibe, gauta imant tokio pačio dydžio imtį su grąžinimu iš originalios duomenų aibės (saviranka, angl. bootstrap). Galutinis sprendimas gaunamas sujungiant visų medžių sprendimus į galutinį sprendimą.

Atsitiktinio miško (angl. random forest) metode sudarant kiekvieną sprendimo mazgą, geriausio galimo skaidymo (angl. split) ieškoma tik tarp atsitiktinai parinkto požymių poaibio [3]. Šio pridedamo atsitiktinumo esmė yra sumažinti modelio priklausomumą nuo to, kokie duomenys buvo naudojami apmokyti modelį (angl. variance). Naudojant atsitiktinį mišką gali sumažėti modelio gebėjimas prisitaikyti prie sudėtingų struktūrų duomenyse (angl. bias) lyginant su sudėtingu sprendimų medžiu, tačiau praktiškai šis sumažėjimas būna daug mažesnis už klaidos dėl apmokymui naudotų duomenų sumažėjimą.

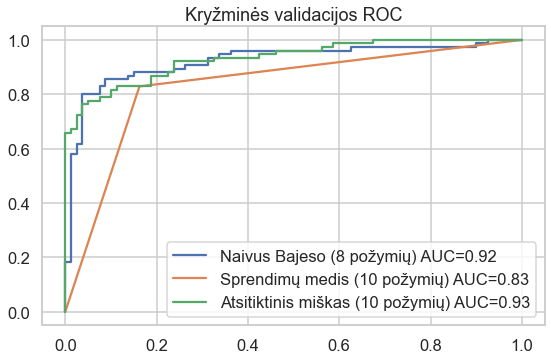
Svarbiausias modelio parametras yra atsitiktinai parenkamų požymių skaičius *max\_features*. Medžių skaičius *n\_estimators* kai kuriais atvejais gali būti parenkamas iš ankšto pagal turimos duomenų aibės ar klasifikavimo uždavinio specifiką*.* Galimi parinkti ir parametrai, naudojami paprastiems sprendimų medžiams, tačiau jie nebėra tokie svarbūs.

Sudarant atsitiktinį mišką siekiama, kad kiekvieno medžio prognozių koreliacijos turi būtų mažos. Taip išvengiama sistemingų klaidų net jeigu dalis medžių priima klaidingą sprendimą. Atsitiktinių miškų sudarymas yra daug sudėtingesnis ir užima daugiau laiko nei sprendimų medžių ar kitų paprastesnių klasifikatorių sudarymas.

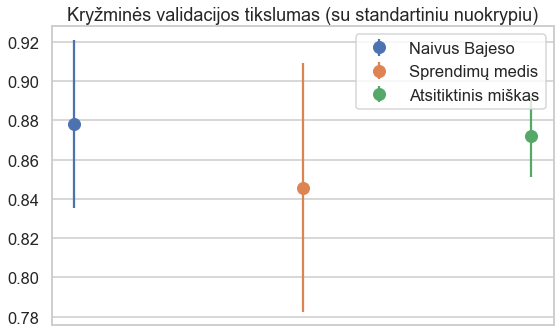
Optimalus parametrų rinkinys ieškotas naudojant parametrų tinklelį n\_estimators={10,25,50,75}, *max\_features*={3,4,5,6}, *min\_samples\_split*={10,20,30}, *n\_features\_to\_select*={0.8,0.9,1.0}. Geriausi kryžminės validacijos rezultatai gauti naudojant reikšmes *n\_estimators*=50, *max\_features*=5, *min\_samples\_split*=20, *n\_features\_to\_select*=1.0

## Klasifikavimo kokybės įvertinimas ir modelių palyginimas

Kadangi modelių palyginimui negali būti naudojama testavimo aibė, modeliai gali būti palyginti kitas strategijas: kryžminės validacijos (angl. k-fold cross validation) ir išlaikymo (validacijos) aibės (angl. validation / hold-out).



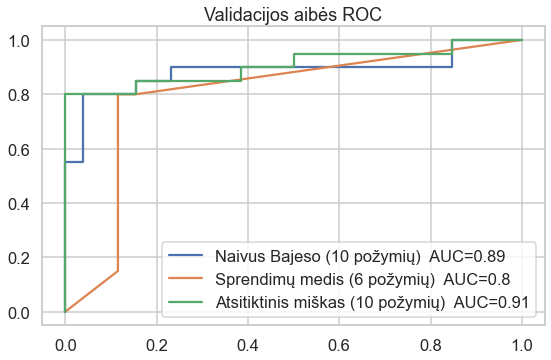
1 pav. ROC kreivė naudojant kryžminę validaciją



2 pav. Vidutinės tikslumo reikšmės su standartiniu nuokrypiu naudojant kryžminę validaciją

3 lentelė Klasifikavimo kokybės matai naudojant kryžminę validaciją

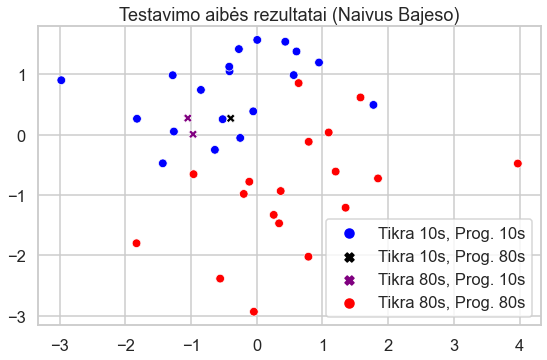
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modelis | Klasė | Precision | Recall | F1-Score | Accuracy |
| Naivus Bajeso | 10-ieji | 0.86 | 0.92 | 0.89 | 0.87 |
| Naivus Bajeso | 80-ieji | 0.89 | 0.8 | 0.84 | 0.87 |
| Sprendimų medis | 10-ieji | 0.84 | 0.62 | 0.71 | 0.72 |
| Sprendimų medis | 80-ieji | 0.63 | 0.85 | 0.72 | 0.72 |
| Atsitiktinis miškas | 10-ieji | 0.86 | 0.92 | 0.89 | 0.87 |
| Atsitiktinis miškas | 80-ieji | 0.89 | 0.8 | 0.84 | 0.87 |



3 pav. ROC kreivė naudojant validacijos aibę

4 lentelė Klasifikavimo kokybės matai naudojant validacijos aibę

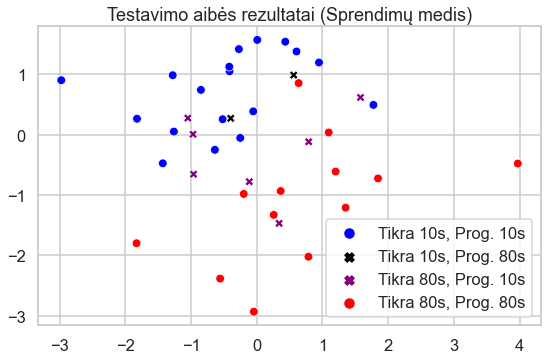
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Modelis | Klasė | Precision | Recall | F1-Score | Accuracy |
| Naivus Bajeso | 10-ieji | 0.86 | 0.96 | 0.91 | 0.89 |
| Naivus Bajeso | 80-ieji | 0.94 | 0.8 | 0.86 | 0.89 |
| Sprendimų medis | 10-ieji | 0.8 | 0.92 | 0.86 | 0.83 |
| Sprendimų medis | 80-ieji | 0.88 | 0.7 | 0.78 | 0.83 |
| Atsitiktinis miškas | 10-ieji | 0.85 | 0.88 | 0.87 | 0.85 |
| Atsitiktinis miškas | 80-ieji | 0.84 | 0.8 | 0.82 | 0.85 |



4 pav. Testavimo aibės rezultatai naivaus Bajeso klasifikatoriui

5 lentelė Maišos matrica naivaus Bajeso klasifikatoriui

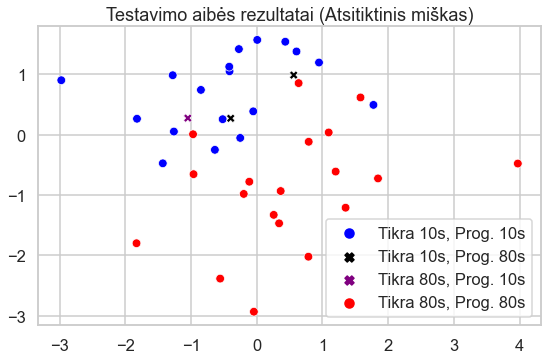
|  |  |  |
| --- | --- | --- |
|  | Prognozuotos | |
| Tikros | 19 | 1 |
| 2 | 18 |



5 pav. Testavimo aibės rezultatai sprendimų medžio klasifikatoriui

6 lentelė Maišos matrica sprendimų medžio klasifikatoriui

|  |  |  |
| --- | --- | --- |
|  | Prognozuotos | |
| Tikros | 18 | 2 |
| 7 | 13 |



6 pav. Testavimo aibės rezultatai atsitiktinio miško klasifikatorui

7 lentelė Maišos matrica atsitiktinio miško klasifikatoriui

|  |  |  |
| --- | --- | --- |
|  | Prognozuotos | |
| Tikros | 18 | 2 |
| 1 | 19 |

# Išvados

Duomenų aibę sudaro skirtingų dešimtmečių dainos su skaitiniais požymiais apie šias dainas. Duomenys

# Šaltiniai

|  |  |
| --- | --- |
| [1] | I. Rish, „An Empirical Study of the Naïve Bayes Classifier,“ *Empirical Methods for Artificial Intelligence,* 2001. |
| [2] | J. F. R. O. C. S. L. Breiman, Classification and Regression Trees, Belmont, CA: Wadworth, 1984. |
| [3] | L. Breiman, „Random Forests,“ *Machine Learning,* t. 45, pp. 5-32, 2001. |
| [4] | S. D. P. Cunningham, „k-Nearest neighbour classifiers,“ *Mult Classif Syst,* t. 54, 2007. |

# Priedas

Žemiau pateiktas naudotas programinis kodas:

# #### Read-in the data

import matplotlib.pyplot as plt

import pandas as pd

import numpy as np

import seaborn as sns

def read\_clean\_data(filename):

df = pd.read\_csv(filename)[['title','artist','year','bpm', 'nrgy', 'dnce', 'dB','live', 'val', 'dur','acous', 'spch','pop']]

df = df.rename({'bpm':'tempo','nrgy':'energy','dnce':'danceability','dB':'loudness','live':'liveness',

'val':'valence','dur':'duration','acous':'acousticness','spch':'speechiness','pop':'popularity'},

axis = 1)

df['decade'] = filename[2:4] + 's'

return df

from sklearn.preprocessing import StandardScaler

filenames = ['1950.csv','1980.csv','2010.csv']

df = pd.concat([read\_clean\_data(i) for i in filenames]).reset\_index()

df\_id = df[["title","artist","decade"]]

df = df.iloc[:,4:len(df.columns)-1]

# pozymiai, pagal kuriuos klasteriuojama

df.describe().T.drop("count",axis=1)

x = StandardScaler().fit\_transform(df)

from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN

from sklearn.manifold import MDS

from sklearn.decomposition import PCA

def do\_mds(x,n\_components=2,\*\*kwargs):

mds = MDS(n\_components,\*\*kwargs)

x\_trans = mds.fit\_transform(x)

return x\_trans

def do\_pca(x,n\_components=2,\*\*kwargs):

pca = PCA(n\_components,\*\*kwargs)

x\_trans = pca.fit\_transform(x)

return x\_trans

# klasterizavimas originaliems duomenims ir sumazinus dimensija

x\_small = do\_pca(x,n\_components=2)

df\_plot = pd.DataFrame(do\_pca(x))

df\_plot.columns = ["x","y"]

fig, ax = plt.subplots(1,1,figsize=(8, 6))

plot = sns.scatterplot(x="x",y="y",hue=df\_id["decade"],data=df\_plot,palette="Dark2",ax=ax)

plot.set\_title("K-means clustering (reduced dimensionality data)")

# #### KMeans

def do\_kmeans(x, standartize = True,\*\*kwargs):

model = KMeans(\*\*kwargs)

pred = model.fit\_predict(x)

return pred, model

from yellowbrick.cluster import SilhouetteVisualizer, KElbowVisualizer

visualizer = KElbowVisualizer(KMeans(), k=(2,10))

visualizer.fit(x)

visualizer.show()

visualizer = KElbowVisualizer(KMeans(), k=(2,10))

visualizer.fit(x\_small)

visualizer.show()

silhouette\_scores = []

for i in list(range(2,11)):

model = KMeans(i)

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')

visualizer.fit(x)

silhouette\_scores.append(visualizer.silhouette\_score\_)

plt.clf()

plt.plot(range(2,11),silhouette\_scores)

plt.title("Silhoutte method optimal number of clusters (original dimensionality data)")

plt.xlabel("Number of clusters")

plt.ylabel("Average silhouette width")

silhouette\_scores = []

for i in list(range(2,11)):

model = KMeans(i)

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')

visualizer.fit(x\_small)

silhouette\_scores.append(visualizer.silhouette\_score\_)

plt.clf()

plt.plot(range(2,11),silhouette\_scores)

plt.title("Silhoutte method optimal number of clusters (reduced dimensionality data)")

plt.xlabel("Number of clusters")

plt.ylabel("Average silhouette width")

for i in [2,5]:

model = KMeans(i,random\_state=123)

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')

visualizer.fit(x)

visualizer.show()

for i in [2,5]:

model = KMeans(i,random\_state=123)

visualizer = SilhouetteVisualizer(model, colors='yellowbrick')

visualizer.fit(x\_small)

visualizer.show()

# elbow metodas

fig, ax = plt.subplots(1,2,figsize=(17, 6))

ax = ax.flatten()

df\_plot["cluster"], \_ = do\_kmeans(x,n\_clusters=5,random\_state=123)

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[0])

plot.set\_title("K-means clustering (original dimensionality data)")

df\_plot["cluster"], \_ = do\_kmeans(x\_small,n\_clusters=5,random\_state=123)

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[1])

plot.set\_title("K-means clustering (reduced dimensionality data)")

# kuo skiriasi klasteriai pagal desimtmecius

sns.set\_context("talk")

df\_plot["decade"] = df\_id["decade"]

sns.relplot(x="x",y="y",kind="scatter",data=df\_plot,col="cluster",hue="decade",palette="Set2",col\_wrap=3)

# silhouette metodas

sns.set\_context("notebook")

fig, ax = plt.subplots(1,2,figsize=(17, 6))

ax = ax.flatten()

df\_plot["cluster"], \_ = do\_kmeans(x,n\_clusters=2,random\_state=123)

df["cluster\_original"] = df\_plot["cluster"]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[0])

plot.set\_title("K-means clustering (original dimensionality data)")

df\_plot["cluster"], \_ = do\_kmeans(x\_small,n\_clusters=2,random\_state=123)

df["cluster\_reduced"] = df\_plot["cluster"]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[1])

plot.set\_title("K-means clustering (reduced dimensionality data)")

def statistics(df):

statistics\_1 = df.groupby("cluster\_original").describe().T

statistics\_1 = statistics\_1.reset\_index()[statistics\_1.reset\_index()["level\_1"].isin(["50%"])]

columns = list(statistics\_1.columns)

columns[0] = "variable"

columns[1] = "statistic"

for i in range(2,len(columns)):

columns[i] = str(columns[i]) + "\_original"

statistics\_1.columns = columns

statistics\_2 = df.groupby("cluster\_reduced").describe().T

statistics\_2 = statistics\_2.reset\_index()[statistics\_2.reset\_index()["level\_1"].isin(["50%"])]

columns = list(statistics\_2.columns)

columns[0] = "variable"

columns[1] = "statistic"

for i in range(2,len(columns)):

columns[i] = str(columns[i]) + "\_reduced"

statistics\_2.columns = columns

statistics = statistics\_1.merge(statistics\_2,how="inner")

statistics = statistics.reindex(sorted(statistics.columns), axis=1)

return statistic

# klasteriu palyginimas

print(df["cluster\_original"].value\_counts())

print(df["cluster\_reduced"].value\_counts())

statistics(df)

# kuo skiriasi klasteriai pagal desimtmecius

df\_plot["decade"] = df\_id["decade"]

sns.relplot(x="x",y="y",kind="scatter",data=df\_plot,col="cluster",hue="decade",palette="Set2")

df\_plot["energy"] = df["energy"]

sns.relplot(x="x",y="y",kind="scatter",hue="energy",size="energy",data=df\_plot,col="cluster",palette="Reds")

df\_plot["acousticness"] = df["acousticness"]

sns.relplot(x="x",y="y",kind="scatter",hue="acousticness",size="acousticness",data=df\_plot,col="cluster",palette="Greens",hue\_norm = (-10,100))

df\_plot["popularity"] = df["popularity"]

sns.relplot(x="x",y="y",kind="scatter",hue="popularity",size="popularity",data=df\_plot,col="cluster",palette="PuBuGn",

hue\_norm = (-10,100))

# #### Hierarchical

def do\_hierarchical(df, standartize = True,\*\*kwargs):

model = AgglomerativeClustering(\*\*kwargs)

pred = model.fit\_predict(x)

return pred, model

from scipy.cluster.hierarchy import dendrogram

def plot\_dendrogram(model, \*\*kwargs):

counts = np.zeros(model.children\_.shape[0])

n\_samples = len(model.labels\_)

for i, merge in enumerate(model.children\_):

current\_count = 0

for child\_idx in merge:

if child\_idx < n\_samples:

current\_count += 1 # leaf node

else:

current\_count += counts[child\_idx - n\_samples]

counts[i] = current\_count

linkage\_matrix = np.column\_stack(

[model.children\_, model.distances\_, counts]

).astype(float)

fig, ax = plt.subplots(1,1,figsize=(17, 6))

dendrogram(linkage\_matrix, \*\*kwargs, ax = ax)

\_, model = do\_hierarchical(x,distance\_threshold=0, n\_clusters=None,linkage="ward")

plot\_dendrogram(model, color\_threshold=0)

plt.title("Hierarchical Clustering Dendrogram Ward linkage")

\_, model = do\_hierarchical(x\_small,distance\_threshold=0, n\_clusters=None,linkage="ward")

plot\_dendrogram(model, color\_threshold=0)

plt.title("Hierarchical Clustering Dendrogram Ward linkage")

fig, ax = plt.subplots(1,2,figsize=(17, 6))

ax = ax.flatten()

df\_plot["cluster"], \_ = do\_hierarchical(x,n\_clusters=2,linkage="ward")

df["cluster\_original"] = df\_plot["cluster"]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[0])

plot.set\_title("Hierarchical clustering (original dimensionality data)")

df\_plot["cluster"], \_ = do\_hierarchical(x\_small,n\_clusters=2,linkage="ward")

df["cluster\_reduced"] = df\_plot["cluster"]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot,palette="Dark2",ax=ax[1])

plot.set\_title("Hierarchical clustering (reduced dimensionality data)")

# #### DBSCAN

from sklearn.neighbors import NearestNeighbors

def do\_dbscan(df, \*\*kwargs):

model = DBSCAN(\*\*kwargs)

pred = model.fit\_predict(x)

return pred, model

x.shape, x\_small.shape

nn=NearestNeighbors(n\_neighbors=20).fit(x)

distances, indices = nn.kneighbors(x)

farthest = distances[:,-1]

fig, ax = plt.subplots(1,1,figsize=(5, 5))

plt.plot(np.sort(farthest))

plt.xlabel("index")

plt.ylabel("distance")

plt.show()

# ##### Originaliam duomenų rinkiniui: minSamples = 2n = 20 , eps = 3.5

nn=NearestNeighbors(n\_neighbors=min\_samples\_small).fit(x\_small)

distances, indices = nn.kneighbors(x\_small)

farthest = distances[:,-1]

fig, ax = plt.subplots(1,1,figsize=(5, 5))

plt.plot(np.sort(farthest))

plt.xlabel("index")

plt.ylabel("distance")

plt.show()

# ##### Sumažintos dimensijos duomenų rinkiniui: minSamples = 4, eps = 0.7

fig, ax = plt.subplots(1,2,figsize=(17, 6))

ax = ax.flatten()

df\_plot["cluster"], \_ = do\_dbscan(x,min\_samples=20,eps=3.5)

df\_noise = df\_plot[df\_plot["cluster"]==-1]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot[df\_plot["cluster"]!=-1],palette="Dark2",ax=ax[0])

plot.scatter(df\_noise["x"],df\_noise["y"],c="grey",alpha=0.5)

plot.set\_title("DBSCAN clustering (original dimensionality data)")

df\_plot["cluster"], \_ = do\_dbscan(x\_small,min\_samples=4,eps=0.7)

df\_noise = df\_plot[df\_plot["cluster"]==-1]

plot = sns.scatterplot(x="x",y="y",hue="cluster",data=df\_plot[df\_plot["cluster"]!=-1],palette="Dark2",ax=ax[1])

plot.scatter(df\_noise["x"],df\_noise["y"],c="grey",alpha=0.5)

plot.set\_title("DBSCAN clustering (reduced dimensionality data)")