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

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

Uždaviniai:

Pasirinktų klasifikatorių teorinis tyrimas.

Optimalių parametrų, įskaitant geriausio požymių poaibio, parinkimas.

Klasifikavimo šablonų sudarymas.

Klasifikavimo kokybės įvertinimas ir modelių palyginimas.

Apibendrintų išvadų pateikimas.

2 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

3 Atliktos analizės aprašymas

Klasifikavimui pasirinktas dainų išleidimo dešimtmetis (požymis „Decade“) su dvejomis galimomis reikšmėmis (80-ieji arba 2010-ieji). Klasifikavimui atlikti pasirinkta naudoti visus 10 skaitinių požymių. Klasių pasiskirstymas subalansuotas – duomenų aibę sudaro 100 2010-ųjų dainų ir 96 80-ųjų dainos. Abiem 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.

3.1 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]:

$$P(y|x_1, \dots, x_n) = \frac{P(y)\prod_{i=1}^{n} P(x_i|y)}{P(x_1, \dots, x_n)}$$

Kur:

y – klasės kintamasis

x_i – požymių vektoriaus x i-toji reikšmė.

Šis modelis gaunamas iš Bajeso formulės:

$$P(y|x_1, \dots, x_n) = \frac{P(y)P(x_1, \dots, x_n|y)}{P(x_1, \dots, x_n)}$$

Laikant, kad požymiai yra tarpusavyje nepriklausomi:

$$P(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = P(x_i|y)$$

Kadangi kiekvienai mokymo aibei $P(x_1, \dots, x_n)$ yra konstanta, klasifikatorius iš tikimybinio modelio sudaromas pasirenkant klasę pagal:

$$\hat{y} = \underset{y}{argmax}\, P(y) \prod_{n=1}^{n} P(x_i|y)$$

Turint skaitinius požymius dažniausiai laikoma, kad $P(x_i|y)$ turi normalųjį skirstinį, kurio parametrai kiekvienai klasei įvertinami naudojant didžiausio tikėtinumo metodą. Tuo tarpu $P(y)$ 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 naivus 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.

Požymiai atrinkti godžiu algoritmu kiekviename žingsnyje šalinant tuo metu blogiausią požymį. Iš viso pašalintų požymių dalis *n_features_to_select* naudota kaip modelio parametras.

Optimalių parametrų ieškota naudojant parametrų tinklelį *n_features_to_selec*t={0.2,0.4,0.6,0.8,1.0}. Naudojant kryžminę validaciją geriausi rezultatai gauti su reikšme 0.6. Tiesa, vidutinio kryžminės validacijos tikslumo prasme rezultatai beveik identiški gaunamiems naudojant visus duomenų aibėje esančius požymius. Pilni optimalių parametrų rinkinio paieškos rezultatai pateikti žemiau (2 lentelė). Numatytosios modelio parametrų reikšmės ir su jomis gauti rezultatai šioje ir tolimesnėse lentelėse pateikiami pakreiptu šriftu.

3 lentelė Optimalių parametrų paieška naivaus Bajeso klasifikatoriui naudojant kryžminę validaciją

n_features_to_select	Vidutinis kryžminės validacijos tikslumas
1.0	*0.865*
0.2	0.814
0.4	0.852
0.6	0.866
0.8	0.859

Optimaliame klasifikatoriuje nenaudojami požymiai „Energy“, „Liveness“, „Valence“, „Tempo“. Toks gautas rezultatas yra natūralus, nes naivus Bajeso klasifikatorius priskiria vienodą svarbą visiems požymiams, todėl jeigu duomenų aibėje turima neinformatyvių požymių, tikėtina gauti prastesnius rezultatus negu jų modelyje nenaudojant.

3.2 Sprendimų medžio klasifikatorius

Sprendimų medžio (angl. decision tree) metodas sukuria klasifikavimo modelius medžio struktūros pavidalu. Duomenų rinkinys sprendimo mazguose (angl. decision node) yra suskaidomas į vis mažesnius poaibius, kartu palaipsniui kuriant susijusį sprendimų medį. Galutinis rezultatas yra medis su sprendimo mazgais ir lapų mazgais (angl. leaf node). Lapo mazgo, į kurį pateko klasifikuojamas objektas, reikšmė atitinką modelio priimtą sprendimą Sprendimo mazgai yra konstruojami pasirenkant tokį kintamąjį, pagal kurio reikšmes galima geriausiai padalinti duomenų rinkinį. Dažnai naudojamos metrikos geriausiam padalijimui rasti yra Gini priemaiša (Gini impurity), informacijos išlošis (information gain) [2].

Sprendimų medžiai lengvai suprantami ir interpretuojami. Kitas iš šio metodo 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 kategorinių kintamųjų, savaime atliekamas daugiau negu dviejų klasių klasifikavimas.

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 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.

Kadangi sprendimų medžiai požymį naudoja konstruoti sprendimų mazgui tik jeigu jis gerai atskiria klases (sprendimų medžiai atlieka savaiminį požymių atrinkimą), todėl nesitikima gauti rezultatų pagerėjimo atrenkant požymių poaibį. Optimalių parametrų ieškota naudojant parametrų tinklelį *max_depth*={4,5,6}, *min_samples_split*={2,5,10,15}, *n_features_to_select*={0.6,0.8,1.0}.

Kryžminės validacijos būdu geriausi rezultatai gauti su *max_depth*=5 ir *min_samples_split*=5, *n_features_to_select*=1.0 (4 lentelė). Pastebėta, kad šis parametrų parinkimas reikšmingai pagerino klasifikavimo rezultatus lyginant su pradiniais modelio parametrais. Fiksavus kitų parametrų reikšmes, bet naudojant mažesnes *n_features_to_select* reikšmes dažniausiai gauti prastesni rezultatai lyginant su didesne požymių aibe, tačiau šio parametro įtaka daug kartų mažesnė už likusiųjų. Taip yra todėl, nes mažiau informatyvūs požymiai retai naudojami konstruojant sprendimų mazgus. Šis rezultatas parodo, kad sprendimų medis savaime atlieka požymių atrinkimą. Apmokius modelį naudojant visą duomenų aibę ir optimalius parametrus, galutinis sudarytas sprendimų medis savo sprendimų mazguose nenaudojo požymių „Danceability“ ir „Liveness“, nors jie iš požymių aibės prieš tai nebuvo išmesti.

4 lentelė Optimalių parametrų paieška sprendimų medžio klasifikatoriui naudojant kryžminę validaciją

max_depth	min_samples_split	n_features_to_select	Vidutinis kryžminės validacijos tikslumas
None	*2*	*1.0*	*0.820*
4	2	0.6	0.846
4	2	0.8	0.859
4	2	1.0	0.872
4	5	0.6	0.846
4	5	0.8	0.859
4	5	1.0	0.872
4	10	0.6	0.839
4	10	0.8	0.852
4	10	1.0	0.859
4	15	0.6	0.814
4	15	0.8	0.82
4	15	1.0	0.833
5	2	0.6	0.859
5	2	0.8	0.852
5	2	1.0	0.884
5	5	0.6	0.852
5	5	0.8	0.846
5	5	1.0	0.884
5	10	0.6	0.846
5	10	0.8	0.839
5	10	1.0	0.865
5	15	0.6	0.814
5	15	0.8	0.807
5	15	1.0	0.826
6	2	0.6	0.846
6	2	0.8	0.833
6	2	1.0	0.846
6	5	0.6	0.852
6	5	0.8	0.852
6	5	1.0	0.865
6	10	0.6	0.846
6	10	0.8	0.846
6	10	1.0	0.852
6	15	0.6	0.814
6	15	0.8	0.807
6	15	1.0	0.82

3.3 Atsitiktinio miško klasifikatorius

Metodai, sukonstruojantys daugiau negu vieną medį vadinami ansamblių (angl. ensemble) metodais. Bagging (Bootstrap aggregating) sprendimų medis sudaro kelis sprendimų medžius, kiekvienam medžiui konstruoti naudodamas atskirą duomenų aibę, gautą imant tokio pačio dydžio imtį su grąžinimu iš originalios duomenų aibės (saviranka, angl. bootstrap). Modelio sprendimas gaunamas sujungiant visų medžių individualius sprendimus į vieną galutinį [3].

Lyginant su bagging sprendimų medžiu, atsitiktinio miško (angl. random forest) metodas sudarant kiekvieną sprendimo mazgą geriausio galimo skaidymo (angl. split) ieškomo tik tarp atsitiktinai parinkto požymių poaibio [4]. Šio į modelį pridedamo atsitiktinumo esmė yra sumažinti modelio priklausomumą nuo to, kokie duomenys buvo naudojami apmokyti modelį (angl. variance), kas yra dažnai kylanti problema naudojant paprastus sprendimo medžius.

Sudarant atsitiktinį mišką išvengiama sistemingų klaidų net jeigu dalis medžių priima klaidingą sprendimą, tačiau atsitiktinių miškų sudarymas trunka daugiau laiko nei sprendimų medžių ar kitų paprastesnių klasifikatorių sudarymas.

Svarbiausi modelio parametrai yra atsitiktinai parenkamų požymių skaičius *max_features* ir konstruojamų medžių skaičius *n_estimators.* Galimi parinkti ir parametrai, naudojami paprastiems sprendimų medžiams, tačiau jie nebėra tokie svarbūs.

Kadangi atsitiktinis miškas paremtas sprendimų medžiais, kaip ir sprendimų medžio atveju nesitikima gauti rezultatų pagerėjimo atrenkant reikšmingus požymius. Požymių atrinkimas atsitiktiniams miškams naudingas tada, kai neinformatyvių požymių skaičius daug kartų didesnis už sprendimų mazgams konstruoti atsitiktinai pasirenkamų požymių skaičių ir optimalus duomenų aibės skaidymas gali būti pasirenkamas vien iš neinformatyvių požymių. Optimalus parametrų rinkinys ieškotas naudojant parametrų tinklelį n_estimators={25,50,100}, *max_features*={2,3,4}, *min_samples_split*={2,5,10}, *n_features_to_select*={0.8,0.9,1.0}.

Dėl atsitiktinumo atsitiktinio miško mokymo procese, kiekvieną kartą galima gauti kitą optimalių parametrų rinkinį, todėl prasmingą kalbėti tik apie geriausius parametrus fiksavus tam tikrą *random_state* reikšmę. Geriausi rezultatai naudojant kryžminę validaciją rezultatai gauti su parametrų reikšmėmis *n_estimators*=100, *max_features*=2, *min_samples_split*=2, *n_features_to_select*=1.0 (5 lentelė). *n_features_to_select* įtaka vėlgi nebuvo tokia didelė kaip kitų modelio parametrų. Pastebėta, kad tiek pats optimalus parametrų rinkinys, tiek su juo gautas vidutinis kryžminės validacijos tikslumas tik minimaliais skyrėsi nuo numatytųjų atsitiktinio miško parametrų.

5 lentelė Optimalių parametrų paieška atsitiktinio miško klasifikatoriui naudojant kryžminę validaciją

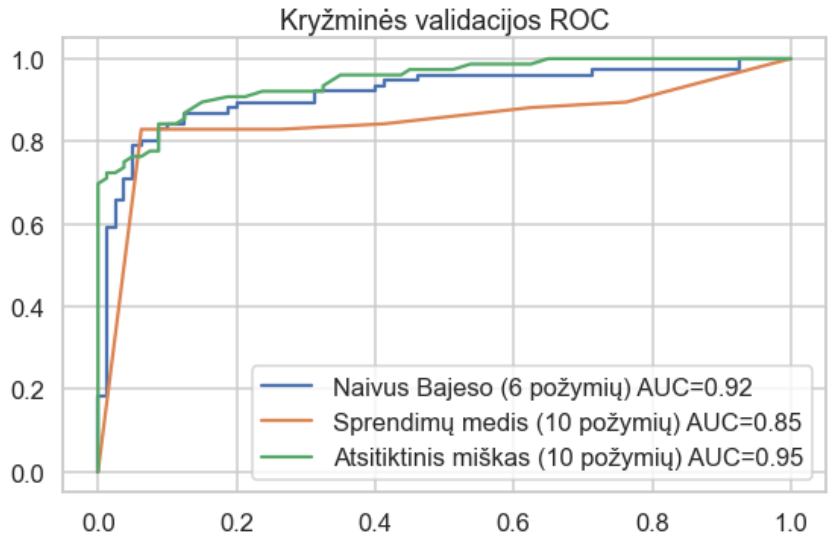
max_features	min_samples_split	n_estimators	n_features_to_select	Vidutinis kryžminės validacijos tikslumas
sqrt(10)=3	*2*	*100*	*1.0*	*0.871*
2	2	25	0.8	0.852
2	2	25	0.9	0.852
2	2	25	1.0	0.859
2	2	50	0.8	0.852
2	2	50	0.9	0.852
2	2	50	1.0	0.878
2	2	100	0.8	0.859
2	2	100	0.9	0.859
2	2	100	1.0	0.878
2	5	25	0.8	0.839
2	5	25	0.9	0.859
2	5	25	1.0	0.878
2	5	50	0.8	0.852
2	5	50	0.9	0.865
2	5	50	1.0	0.878
2	5	100	0.8	0.852
2	5	100	0.9	0.872
2	5	100	1.0	0.878
2	10	25	0.8	0.846
2	10	25	0.9	0.859
2	10	25	1.0	0.865
2	10	50	0.8	0.846
2	10	50	0.9	0.859
2	10	50	1.0	0.859
2	10	100	0.8	0.846
2	10	100	0.9	0.859
2	10	100	1.0	0.852
3	2	25	0.8	0.839
3	2	25	0.9	0.846
3	2	25	1.0	0.84
3	2	50	0.8	0.859
3	2	50	0.9	0.859
3	2	50	1.0	0.859
3	2	100	0.8	0.846
3	2	100	0.9	0.859
3	2	100	1.0	0.872
3	5	25	0.8	0.833
3	5	25	0.9	0.859
3	5	25	1.0	0.872
3	5	50	0.8	0.846
3	5	50	0.9	0.833
3	5	50	1.0	0.865
3	5	100	0.8	0.859

3	5	100	0.9	0.846
3	5	100	1.0	0.859
3	10	25	0.8	0.84
3	10	25	0.9	0.84
3	10	25	1.0	0.852
3	10	50	0.8	0.833
3	10	50	0.9	0.833
3	10	50	1.0	0.84
3	10	100	0.8	0.846
3	10	100	0.9	0.846
3	10	100	1.0	0.852
4	2	25	0.8	0.839
4	2	25	0.9	0.852
4	2	25	1.0	0.84
4	2	50	0.8	0.846
4	2	50	0.9	0.865
4	2	50	1.0	0.852
4	2	100	0.8	0.846
4	2	100	0.9	0.865
4	2	100	1.0	0.859
4	5	25	0.8	0.833
4	5	25	0.9	0.852
4	5	25	1.0	0.84
4	5	50	0.8	0.846
4	5	50	0.9	0.865
4	5	50	1.0	0.859
4	5	100	0.8	0.833
4	5	100	0.9	0.865
4	5	100	1.0	0.865
4	10	25	0.8	0.827
4	10	25	0.9	0.833
4	10	25	1.0	0.859
4	10	50	0.8	0.833
4	10	50	0.9	0.84
4	10	50	1.0	0.872
4	10	100	0.8	0.846
4	10	100	0.9	0.852
4	10	100	1.0	0.859

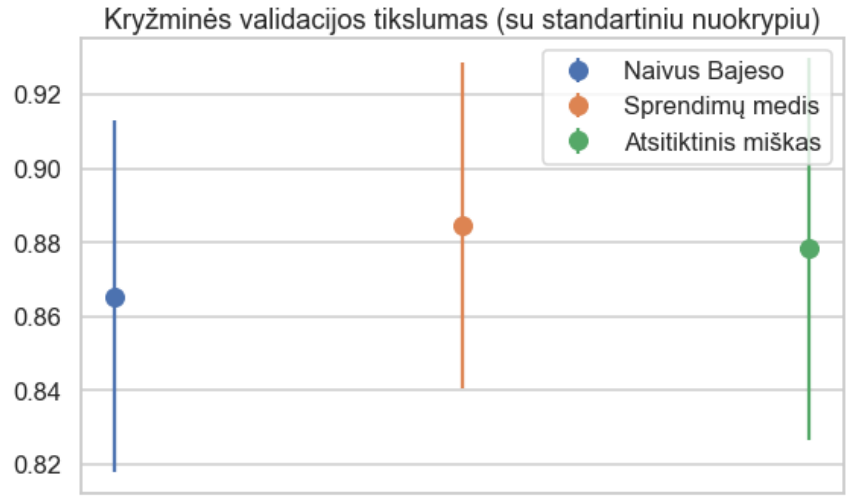
3.4 Klasifikavimo kokybės įvertinimas ir modelių palyginimas

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

Naudojant kryžminę validaciją matomas naivaus Bajeso ir atsitiktinio miško metodų pranašumas lyginant su sprendimų medžiu pagal ROC grafiką (1 pav.). Pagal tikslumo grafiką (2 pav.) ir pagal kitas modelio kokybės įvertinimo metrikas (6 lentelė) matomi tik minimalūs skirtumai tarp modelių. Iš tikslumo grafikų pastebimas labai mažas atsitiktinio miško standartinis nuokrypis. Šis rezultatas susijęs su šio metodo konstravimo principu, kuris į modelio konstravimą įdeda papildomo atsitiktinumo siekiant sumažinti rezultatų priklausomumą nuo tam tikros mokymo aibės. Tuo tarpo kitų metodų rezultatai stipriai kito kryžminės validacijos metu. Šios du modelius pritaikius anksčiau nematytiems duomenims tikėtina gauti prastesnius rezultatus negu tikimasi.



1 pav. ROC kreivė naudojant kryžminę validaciją

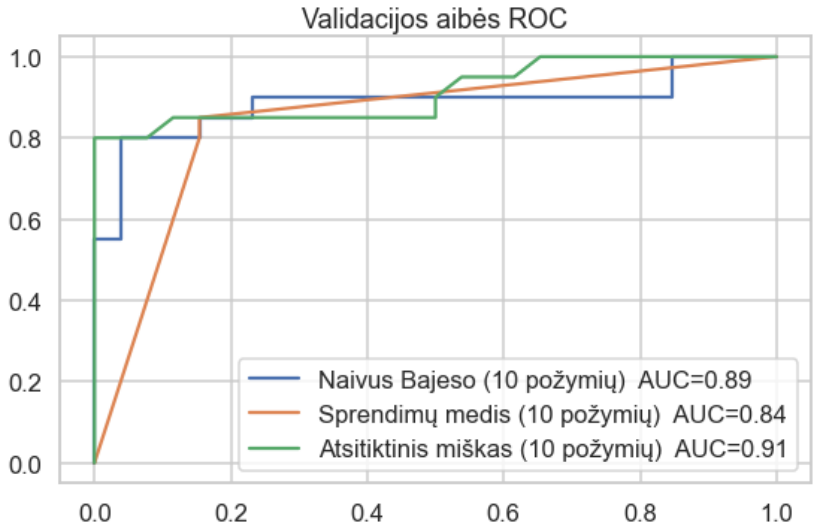


2 pav. Vidutinės tikslumo reikšmės su standartiniu nuokrypiu naudojant kryžminę validaciją

6 lentelė Klasifikavimo kokybės matai naudojant kryžminę validaciją

Modelis	Klasė	Precision	Recall	F1-Score	Accuracy
Naivus Bajeso	10-ieji	0.85	0.90	0.87	0.87
Naivus Bajeso	80-ieji	0.89	0.83	0.86	0.87
Sprendimų medis	10-ieji	0.85	0.94	0.89	0.88
Sprendimų medis	80-ieji	0.93	0.83	0.88	0.88
Atsitiktinis miškas	10-ieji	0.86	0.91	0.89	0.88
Atsitiktinis miškas	80-ieji	0.90	0.84	0.87	0.88

Modelius lyginant pagal ROC grafikus naudojant validacijos aibę vietoje kryžminės validacijos gauti panašūs rezultatai kaip ir prieš tai (3 pav.). Pagal beveik visas modelio kokybės skaitines metrikas matomas atsitiktinio miško pranašumas lyginant su naivaus Bajeso klasifikatoriumi. Prasčiausi rezultatai gauti spendimų medžio klasifikatoriui (7 lentelė).



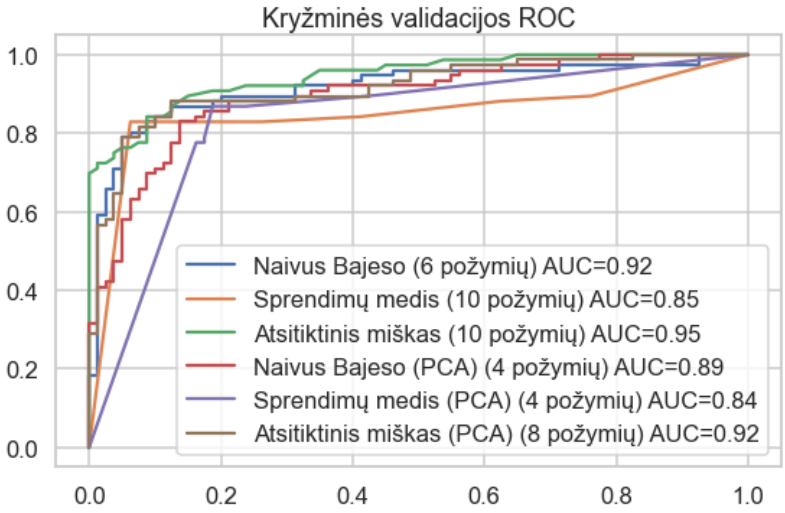
3 pav. ROC kreivė naudojant validacijos aibę

7 lentelė Klasifikavimo kokybės matai naudojant validacijos aibę

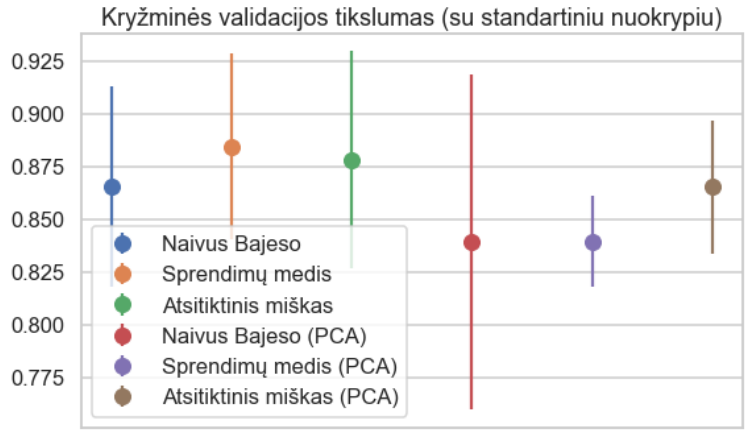
Modelis	Klasė	Precision	Recall	F1-Score	Accuracy
Naivus Bajeso	10-ieji	0.86	0.96	0.90	0.89
Naivus Bajeso	80-ieji	0.94	0.80	0.86	0.89
Sprendimų medis	10-ieji	0.88	0.85	0.83	0.85
Sprendimų medis	80-ieji	0.81	0.85	0.83	0.85
Atsitiktinis miškas	10-ieji	0.87	1.00	0.93	0.91
Atsitiktinis miškas	80-ieji	1.00	0.80	0.89	0.91

Naudojant ankščiau aprašytą optimalių parametrų radimo procedūrą, papildomai sudaryti modeliai, dimensijos mažinimui naudojantys PCA algoritmą vietoje rekursyvaus prasčiausių požymių eliminavimo.

Modeliai tarpusavyje palyginti naudojant kryžminę validaciją (4 ir 5 pav.). Visų trijų modelių atvejais šiuo dimensijos mažinimo metodu gauti prastesni rezultatai.

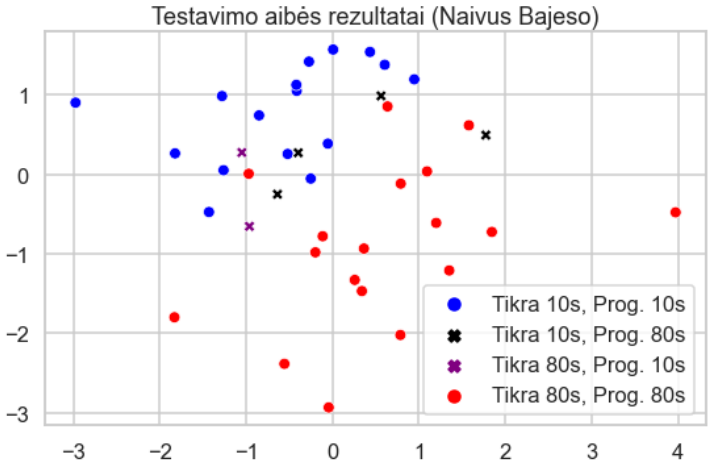


4 pav. ROC kreivė naudojant kryžminę validaciją modeliams su PCA metodu sumažinta dimensija



5 pav. Tikslumo grafikas naudojant kryžminę validaciją modeliams su PCA metodu sumažinta dimensija

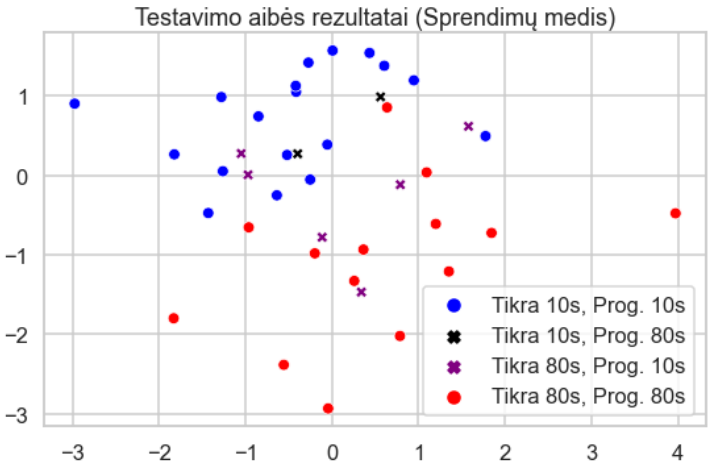
Sudaryti modeliai buvo panaudoti klasifikuoti testavimo aibėje esančius stebėjimus. Kiekvieno klasifikatoriaus rezultatai pateikti maišos matricomis (8,9 ir 10 lentelės) ir vizualiai (6, 7 ir 8 pav.). Matoma, kad blogai klasifikuoti stebėjimai PCA metodu sumažintoje erdvėje riboje yra tarp dviejų klasių klasterių. Taip pat pastebėta, kad visi sudaryti modeliai blogai klasifikavo tas pačias dainas: „Sun Is Shining“, „Summer Of '69“ ir „I Dont Wanna Live Forever“.



6 pav. Testavimo aibės rezultatai naivaus Bajeso klasifikatoriui

8 lentelė Maišos matrica naivaus Bajeso klasifikatoriui

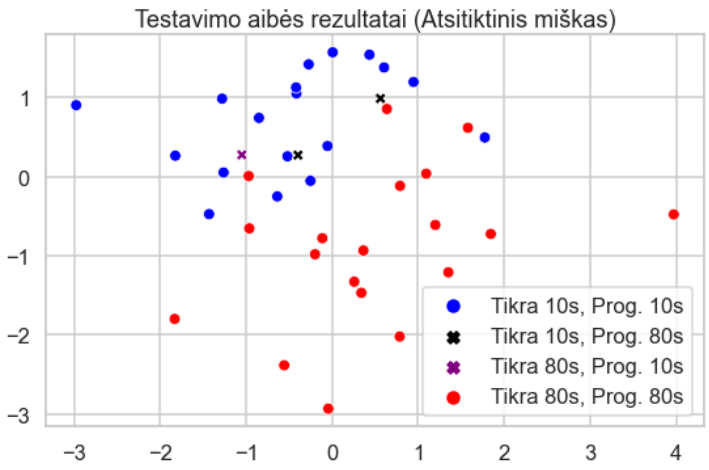
	Prognozuotos	
Tikros	16	4
	2	18



7 pav. Testavimo aibės rezultatai sprendimų medžio klasifikatoriui

9 lentelė Maišos matrica sprendimų medžio klasifikatoriui

	Prognozuotos	
Tikros	18	2
	6	14



8 pav. Testavimo aibės rezultatai atsitiktinio miško klasifikatorui

10 lentelė Maišos matrica atsitiktinio miško klasifikatoriui

	Prognozuotos	
Tikros	18	2
	1	19

4 Išvados

Duomenų aibę sudaro dviejų skirtingų dešimtmečių (80-ųjų ir 2010-ųjų) dainos su skaitiniais požymiais apie šias dainas. Klasifikavimui pasirinkta naudoti visus skaitinius požymius (n=10). Duomenys padalinti į mokymo ir testavimo aibes naudojant santykį 80-20, požymių matavimo skalės suvienodintos standartizuojant pagal vidurkį ir dispersiją. Klasifikavimui naudoti trys metodai: naivus Bajeso, sprendimų medis ir atsitiktinis miškas.

Kryžmines validacijos metodu ieškant optimalių parametrų naiviam Bajeso klasifikatoriui rasta, kad požymių dimensijos sumažinimas pašalinant neinformatyvius požymius daro teigiamą įtaką šiam klasifikatoriui. Geriausi rezultatai gauti pašalinus 4 mažiau informatyvius požymius „Energy“, „Liveness“, „Valence“, ir „Tempo“ (*n_features_to_selec*t=0.6). Tiesa, gauti rezultatai tik minimaliai geresni negu naudojant visus duomenų aibėje esančius požymius.

Reikšmingų požymių atrinkimas neturėjo teigiamos įtakos sprendimų medžio klasifikavimo rezultatams. Taip yra todėl, nes sprendimų medis savaime atlieka požymių atrinkimą konstruodamas sprendimų mazgus. Geriausi rezultatai (vidutinis kryžminės validacijos tikslumas - 0.872) gauti naudojant medžio gylį lygų 5 ir reikalaujant bent 5 stebėjimų norint toliau skaidyti duomenų aibę (*max_depth*=5, *min_samples_split*=5, *n_features_to_select*=1.0). Šie rezultatai ryškiai skyrėsi nuo rezultatų gautų naudojant numatytuosius modelio parametrus (vidutinis kryžminės validacijos tikslumas - 0.820).

Kadangi atsitiktinio miško metodas pagrįstas sprendimų medžiais, šiam metodui gauta panaši požymių atrinkimo įtaka, geriausi rezultatai taip pat gauti naudojant visą požymių aibę. Optimalus parametrų rinkinys (vidutinis kryžminės validacijos tikslumas - 0.872) gautas sudarant 100 medžių, kuriuose konstruoti sprendimų mazgams naudojami 2 atsitiktiniai požymiai su sąlyga, kad mazge yra bent 2 stebėjimai (*n_estimators*=100, *max_features*=2, *min_samples_split*=2, *n_features_to_select*=1.0). Tiesa, šie rezultatai tik minimaliai skyrėsi nuo numatytųjų modelio parametrų rinkinio (naudojančio *max_features*=3).

Geriausi klasifikavimo rezultatai, vertinimui naudojant kryžminės validacijos ir validacijos aibės metodus, gauti naudojant atsitiktinio miško klasifikatorių. Metodas pasižymi ilgai trunkančia apmokymo trukme, tačiau šiuo atveju dėl duomenų aibės mažo dydžio modelio apmokymas nepasižymėjo ilga trukme.

Beveik tokie patys geri rezultatai pagal modelio kokybės metrikas gauti naudojant naivų Bajeso klasifikatorių. Šio klasifikatoriaus požymių nepriklausomumo prielaida yra visai natūralios turimoje duomenų aibėje. Tiesa, dėl mažos turimos imties pilnai neišnaudojamas aukštas modelio mokymosi ir prognozavimo greitis.

Prasčiausi rezultatai iš tirtų modelių pagal beveik visas metrikas gauti naudojant sprendimų medį.

Palyginus dimensijos mažinimą naudojant rekursyvų prasčiausių požymių šalinimą ir PCA algoritmą, antruoju metodu visų trijų tirtų modelių atveju gauti prastesni rezultatai.

Apibendrinus abejais modelių tarpusavio palyginimo metodais gautus rezultatus, prasminga rinktis atsitiktinio miško (geresni rezultatai) arba naivų Bajeso (greitesnis veikimas su beveik tokia pačia klasifikavimo kokybe) klasifikatorių.

Modelius pritaikius testavimo aibei pasitvirtino prieš tai atlikto modelio tarpusavio palyginimo rezultatai: geriausi rezultatai gauti su atsitiktinio miško klasifikatoriumi, prasčiausi – naudojant sprendimų medį. Panaudojus PCA metodą sumažinti požymių dimensiją iki 2 rasta, kad visiems klasifikatoriams dauguma neteisingai klasifikuojamų dainų vaizduojamos riboje tarp dešimtmečių klasterių. Atliekant blogai klasifikuotų dainų analizę pastebėta, kad tam tikros dainos buvo klaidingai klasifikuojamos visų sudarytų klasifikatorių.

5 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] R. Z. M. Schonlau, „The random forest algorithm for statistical learning.,“ *Stata Journal,* pp. 3-29, 2020.

[4] L. Breiman, „Random Forests,“ *Machine Learning,* t. 45, pp. 5-32, 2001.

[5] S. D. P. Cunningham, „k-Nearest neighbour classifiers,“ *Mult Classif Syst,* t. 54, 2007.

Priedas

Žemiau pateiktas naudotas programinis kodas:

```
# ### Reading-in the data

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

sns.set(rc={'figure.figsize':(9,5.5)})
sns.set_style("whitegrid")
sns.set_context("talk")


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'
    df = df[~df["year"].between(1991,1999)]
    return df


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

df = pd.concat([read_clean_data(i) for i in filenames]).reset_index()
X = df.iloc[:,4:len(df.columns)-1]
y = df["decade"].values



# Descriptive statistics
print(df["decade"].value_counts()) # balanced classes

df[df["decade"]=="80s"].describe().T.drop("count",axis=1)

df[df["decade"]=="10s"].describe().T.drop("count",axis=1)



from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

X_train,        X_test,        y_train,        y_test,        indices_train,        indices_test        =
train_test_split(X,y,np.arange(len(y)),test_size = 0.20, random_state = 0)

scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV, KFold, PredefinedSplit
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score

cv = KFold(5)

indices = np.zeros_like(y_train)
indices[int(len(y_train)*0.3):] = -1
validate = PredefinedSplit(indices)



def fit_model(X_train, y_train, model, cv, param_grid = {}, output=True):
    grid_search = GridSearchCV(model,param_grid=param_grid,cv=cv)
    grid_search.fit(X_train,y_train)
    # results for different parameter values
    if output:
        results_df = pd.DataFrame(grid_search.cv_results_)
        results_df       =       results_df.iloc[:,(results_df.columns.str.contains("param_"))       |
(results_df.columns == "mean_test_score")]
        print(results_df)
    results_df.round(3).to_csv("cv_results.csv",index=False)
    return grid_search


# ### Naive Bayes


from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SequentialFeatureSelector,RFE
nb = GaussianNB()

nb_pipe = Pipeline([
   ('select', SequentialFeatureSelector(nb,direction="backward")),
   ('model', nb)])

param_grid_nb = {
    'select__n_features_to_select': [0.2,0.4,0.6,0.8,1.0]}

nb_best = fit_model(X_train,y_train,nb_pipe,cv,param_grid_nb).best_estimator_

nb_best.get_params()


# dropped columns
used_columns = nb_best["select"].get_support(indices=True)
X.drop(columns=X.columns[used_columns])


nb_pca_pipe = Pipeline([
   ('select', PCA()),
   ('model', nb)])

param_grid_pca_nb = {
    'select__n_components': [2,4,6,8,10]}

nb_pca_best = fit_model(X_train,y_train,nb_pca_pipe,cv,param_grid_pca_nb).best_estimator_
```

```
nb_pca_best.get_params()


# ### Decision Tree

dt = DecisionTreeClassifier(random_state=100)

dt_pipe = Pipeline([
   ('select', RFE(dt)),
   ('model', dt)])


param_grid_dt = {
    'select__n_features_to_select': [0.6,0.8,1.0],
    "model__max_depth":[4,5,6],"model__min_samples_split":[2,5,10,15]}

dt_best = fit_model(X_train,y_train,dt_pipe,cv,param_grid_dt).best_estimator_

dt_best.get_params()


# dropped columns
used_columns = dt_best["select"].get_support(indices=True)
X.drop(columns=X.columns[used_columns])


dt_pca_pipe = Pipeline([
   ('select', PCA()),
   ('model', dt)])


param_grid_pca_dt = {
    'select__n_components': [4,6,8,10],
    "model__max_depth":[4,5,6],"model__min_samples_split":[2,5,10,15]}

dt_pca_best = fit_model(X_train,y_train,dt_pca_pipe,cv,param_grid_pca_dt).best_estimator_

dt_pca_best.get_params()


# ### Random forest


pd.set_option('display.max_rows', 1000)

rf = RandomForestClassifier(random_state=100)

rf_pipe = Pipeline([
   ('select', RFE(rf)),
   ('model', rf)])

param_grid_rf= {
              'select__n_features_to_select': [0.8,0.9,1.0],
              "model__max_features":[2,3,4],
              "model__min_samples_split":[2,5,10],
              "model__n_estimators":[25,50,100]}


rf_best = fit_model(X_train,y_train,rf_pipe,cv,param_grid_rf).best_estimator_
```

```
rf_best.get_params()


# dropped columns
used_columns = rf_best["select"].get_support(indices=True)
X.drop(columns=X.columns[used_columns])


rf_pca_pipe = Pipeline([
   ('select', PCA()),
   ('model', rf)])

param_grid_pca_rf= {
              'select__n_components': [4,6,8,10],
              "model__max_features":[2,3,4],
              "model__min_samples_split":[2,5,10],
              "model__n_estimators":[25,50,100]}


rf_pca_best = fit_model(X_train,y_train,rf_pca_pipe,cv,param_grid_pca_rf).best_estimator_

rf_pca_best.get_params()


# ### Model comparison



from sklearn.model_selection import cross_val_predict, cross_validate
from sklearn.metrics import accuracy_score



def cross_validate_ROC_AUC(X_train,y_train,models,cv):

    fig, ax = plt.subplots()
    fig2, ax2 = plt.subplots()
    x = 0

    for i,j in zip(models,names):
        print(j)

        y_proba = cross_val_predict(i,X_train,y_train,method="predict_proba",cv=cv)
        fpr, tpr, _ = roc_curve(y_train, y_proba[:,1],pos_label="80s")
        auc = round(roc_auc_score(y_train, y_proba[:,1]),2)
        ax.plot(fpr,  tpr,  label=j  +  "  ("  +  str(i["model"].n_features_in_)  +  "  požymių)  "  +
"AUC="+str(auc))

        y_pred = cross_val_predict(i,X_train,y_train,cv=cv)
        cr = classification_report(y_train,y_pred,output_dict=True)
        cr = pd.DataFrame(cr).transpose()
        print(cr)


        scores = cross_validate(i,X_train,y_train,cv=cv)["test_score"]
        mean = np.mean(scores)
        sd = np.sqrt(np.var(scores))
        ax2.errorbar(x, mean, yerr=sd,fmt="o",label=j,markersize=12)
        x = x + 1

        # this is to convert to a 'writable' format
```

```
        cr = cr.reset_index()
        cr = cr.iloc[0:2,:4]
        cr["accuracy"] = accuracy_score(y_train,y_pred)
        cr.columns.values[0] = "klasė"
        cr.insert(0,"modelis",j)
        cr.columns = pd.Series(cr.columns).str.title()
        cr.round(2).to_csv("cross_validate_"+ j + ".csv",index=False)

        print("\n")

    ax.set_title("Kryžminės validacijos ROC")
    ax.legend()
    ax2.set_title("Kryžminės validacijos tikslumas (su standartiniu nuokrypiu)")
    ax2.axes.get_xaxis().set_ticks([])
    ax2.legend()



def validate_ROC_AUC(X_train,y_train,models,cv):

    fig, ax = plt.subplots()

    for i,j in zip(models,names):
        print(j)

        split = list(cv.split())
        train_indices, validate_indices = split[0]
        X_validate = X_train[validate_indices]
        y_validate = y_train[validate_indices]
        X_train_small = X_train[train_indices]
        y_train_small = y_train[train_indices]

        i.fit(X_train_small,y_train_small)

        y_proba = i.predict_proba(X_validate)
        fpr, tpr, _ = roc_curve(y_validate, y_proba[:,1],pos_label="80s")
        auc = round(roc_auc_score(y_validate, y_proba[:,1]),2)
        ax.plot(fpr,  tpr,  label=j+  "  ("  +  str(i["model"].n_features_in_)  +  "  požymių)  "  +  "
AUC="+str(auc))

        y_pred = i.predict(X_validate)
        cr = classification_report(y_validate,y_pred,output_dict=True)
        cr = pd.DataFrame(cr).transpose()
        print(cr)


        # this is to convert to a 'writable' format
        cr = cr.reset_index()
        cr = cr.iloc[0:2,:4]
        cr["accuracy"] = accuracy_score(y_validate,y_pred)
        cr.columns.values[0] = "klasė"
        cr.insert(0,"modelis",j)
        cr.columns = pd.Series(cr.columns).str.title()
        cr.round(2).to_csv("validate_"+ j + ".csv",index=False)


        print("\n")
    ax.set_title("Validacijos aibės ROC")
    ax.legend()
```

```
models = [nb_best,dt_best,rf_best]
names = ["Naivus Bajeso","Sprendimų medis","Atsitiktinis miškas"]


cross_validate_ROC_AUC(X_train,y_train,models,cv)


models = [nb_best,dt_best,rf_best,nb_pca_best,dt_pca_best,rf_pca_best]
names = ["Naivus Bajeso","Sprendimų medis",
         "Atsitiktinis miškas","Naivus Bajeso (PCA)",
         "Sprendimų medis (PCA)","Atsitiktinis miškas (PCA)"]


cross_validate_ROC_AUC(X_train,y_train,models,cv)


nb_best_2 = fit_model(X_train,y_train,nb_pipe,validate,param_grid_nb).best_estimator_

dt_best_2 = fit_model(X_train,y_train,dt_pipe,validate,param_grid_dt).best_estimator_

rf_best_2 = fit_model(X_train,y_train,rf_pipe,validate,param_grid_rf).best_estimator_

models_2 = [nb_best_2,dt_best_2,rf_best_2]


validate_ROC_AUC(X_train,y_train,models_2,validate)


nb_pca_best_2 = fit_model(X_train,y_train,nb_pca_pipe,validate,param_grid_pca_nb).best_estimator_

dt_pca_best_2 = fit_model(X_train,y_train,dt_pca_pipe,validate,param_grid_pca_dt).best_estimator_

rf_pca_best_2 = fit_model(X_train,y_train,rf_pca_pipe,validate,param_grid_pca_rf).best_estimator_

models_2 = [nb_best_2,dt_best_2,rf_best_2,nb_pca_best_2,dt_pca_best_2,rf_pca_best_2]


validate_ROC_AUC(X_train,y_train,models_2,validate)



# this is a helper function to paste into a word table
filenames   =   ['validate_Naivus   Bajeso.csv','validate_Sprendimų   medis.csv','validate_Atsitiktinis
miškas.csv']
filenames2 = ['cross_validate_Naivus Bajeso.csv',
              'cross_validate_Sprendimų medis.csv','cross_validate_Atsitiktinis miškas.csv']

data = pd.concat([pd.read_csv(i) for i in filenames])
data.to_csv("word_table_validate.csv",index=False)


data = pd.concat([pd.read_csv(i) for i in filenames2])
data.to_csv("word_table_cross_validate.csv",index=False)


# ### Model evaluation

from sklearn.decomposition import PCA

pca = PCA(2)
pca.fit(X_train)
```

```
X_small = pca.transform(X_test)

def metric(model, name, X_test, y_test, X_small, df=df):
    fig, ax = plt.subplots()

    y_pred = model.predict(X_test)
    cr = classification_report(y_test,y_pred,output_dict=True)
    cr = pd.DataFrame(cr).transpose()
    print(name)
    print(cr)
    print("\n")
    print("Confusion matrix")
    print(confusion_matrix(y_test,y_pred))
    print("\n")

         # this is to convert to a 'writable' format
    cr = cr.reset_index()
    cr = cr.iloc[0:2,:4]
    cr["accuracy"] = accuracy_score(y_test,y_pred)
    cr.columns.values[0] = "klasė"
    cr.insert(0,"modelis",name)
    cr.columns = pd.Series(cr.columns).str.title()
    cr.round(2).to_csv("test_"+ name + ".csv",index=False)

    classes = []
    for i,j in zip(y_test,y_pred):
        if i == "80s" and j == "10s":
            classes.append("Tikra 80s, Prog. 10s")
        if i == "80s" and j == "80s":
            classes.append("Tikra 80s, Prog. 80s")
        if i == "10s" and j == "80s":
            classes.append("Tikra 10s, Prog. 80s")
        if i == "10s" and j == "10s":
            classes.append("Tikra 10s, Prog. 10s")
    classes = pd.Categorical(classes)
    ax = sns.scatterplot(X_small[:,0],X_small[:,1],hue=classes,
                    style=classes,s=75,markers=["o","X","X","o"],
                    palette=["blue","black","purple","red"],ax=ax)
    ax.set_title("Testavimo aibės rezultatai " +  "(" + name + ")")
    print("Blogai klasifikuotos dainos:")
    print(df.iloc[indices_test].iloc[(y_pred != y_test),1:4])
    print("\n\n")


for i,j in zip(models,names):
    metric(i,j,X_test,y_test,X_small)


filenames3 = ['test_Naivus Bajeso.csv',
              'test_Sprendimų medis.csv','test_Atsitiktinis miškas.csv']

data = pd.concat([pd.read_csv(i) for i in filenames3])
data.to_csv("word_table_test.csv",index=False)
```