VILNIAUS UNIVERSITETAS MATEMATIKOS IR INFORMATIKOS FAKULTETAS

DUOMENŲ MOKSLAS

TEMŲ MODELIAVIMAS

Projektinis darbas

Atliko: Simona Gelžinytė,
Ugnė Kniukštaitė,
Laineda Morkytė,
Austėja Valeikaitė
DM 4k. 2gr.

Vilnius

2023

TURINYS

1. ĮVADAS

Temų modeliavimas priskiriamas neprižiūrimam mašininiam mokymuisi, kurio metu dokumentai apdorojami siekiant nustatyti santykines temas. Tai labai svarbi tradicinio natūralaus apdorojimo metodo sąvoka, nes ji gali padėti gauti semantinį ryšį tarp žodžių, esančių dokumentų klasteriuose.

1.1 Tyrimo tikslas

Sukurti programą, kuri duomenų rinkinį suskirstytų į temas.

1.2 Tyrimo uždaviniai

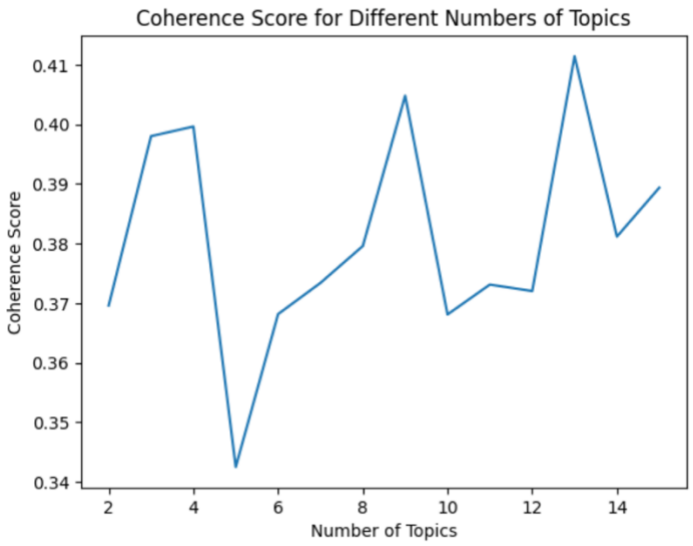
- Paruošti duomenis;
- Apdoroti tekstą: pašalinti skyrybos ženklus, konvertuoti didžiąsias raides į mažąsias, atlikti lemavimą – atstyti žodį į jo pagrindinę formą;
- Pritaikyti skirtingus temų modeliavimo algoritmus;
- Palyginti algoritmus;
- Pateikti rezultatus bei išvadas.

1.3 Duomenys ir programinė įranga

Pasirinktas duomenų rinkinys apie mokslinius straipsnius. Duomenų rinkinyje pateiktas straipsnio pavadinimas bei jo santrauka. Iš viso yra 6 mokslinių straipsnių temos: informatika, fizika, matematika, statistika, kiekybinė biologija ir kiekybiniai finansai. Duomenys buvo padalinti į mokymo ir testavimo aibes. Dirbsime tik su mokymo imtimi. Praleistų reikšmių nėra. Tyrimo metu naudota „Google Colab“ programinė įranga.

2. TEMŲ SKAIČIAUS PARINKIMAS

Norėdami rasti geriausią temų skaičių, suskaičiavome coherence balą. Gavome, jog aukščiausias balas bus gaunamas temų skaičių parinkus 13. Coherence balas, kai temų skaičius 13 yra ~0,41, kuo šis skaičius arčiau 1, tuo temos bus geriau modeliuojamos.



1 pav. coherence balas skirtingiems temų skaičiams

3. TEMŲ MODELIAVIMO TECHNIKOS

Prieš pritaikant kiekvieną algoritmą duomenys buvo tinkamai paruošti – pašalinti nereikšmingi žodžiai, tekstai atstatyti į pagrindines žodžių formas. Kiekvienam algoritmui buvo naudoti du vektorizavimo tipai: TF – IDF ir žodžių maišas.

Temų modeliavimo technikos gerumo įvertinimui buvo naudojamas coherence balas, kurio galimos reikšmės yra iš $[0, 1]$ intervalo, kuo reikšmė arčiau vieneto – tuo geresnis temų modeliavimas. Šis matas reiškia loginį temai priskirtų žodžių ryšį su tekstu.

3.1 LDA

3.1.1 LDA algoritmas

Latent Dirichlet Allocation - tikimybinis statistinis modelis, naudojamas dokumentų rinkinyje aptariamoms temoms nustatyti. Tai generatyvinis modelis, pagal kurį daroma prielaida, kad kiekvienas dokumentas yra nedidelio skaičiaus temų mišinys ir kad kiekvienas dokumente esantis žodis yra susijęs su viena iš temų. LDA yra trijų lygių hierarchinis Bajeso modelis, kurį sudaro "dokumento lygmuo, temos lygmuo ir žodžio lygmuo". Dokumento lygmeniu kiekvienas

dokumentas modeliuojamas kaip nedidelio skaičiaus temų mišinys. Temos lygmeniu kiekviena tema modeliuojama kaip žodžių pasiskirstymas, o žodžio lygmeniu kiekvienas žodis modeliuojamas kaip temų pasiskirstymas.

Algoritmo etapai:

1. Žodžiams atsitiktinai priskiriamos temos;
2. Kiekvieno žodžio tinkamumas temai yra vertinamas atsižvelgiant į kitų temų kontekstą ir žodžio dažnumą kitose temose;
3. Atsižvelgiant į visą duomenų rinkinį, siekiama nustatyti, kokie žodžiai dominuoja kiekvienoje temoje.
4. Koreguojama tol, kol visiems straipsniams sukuriamas nuoseklus temų rinkinys.

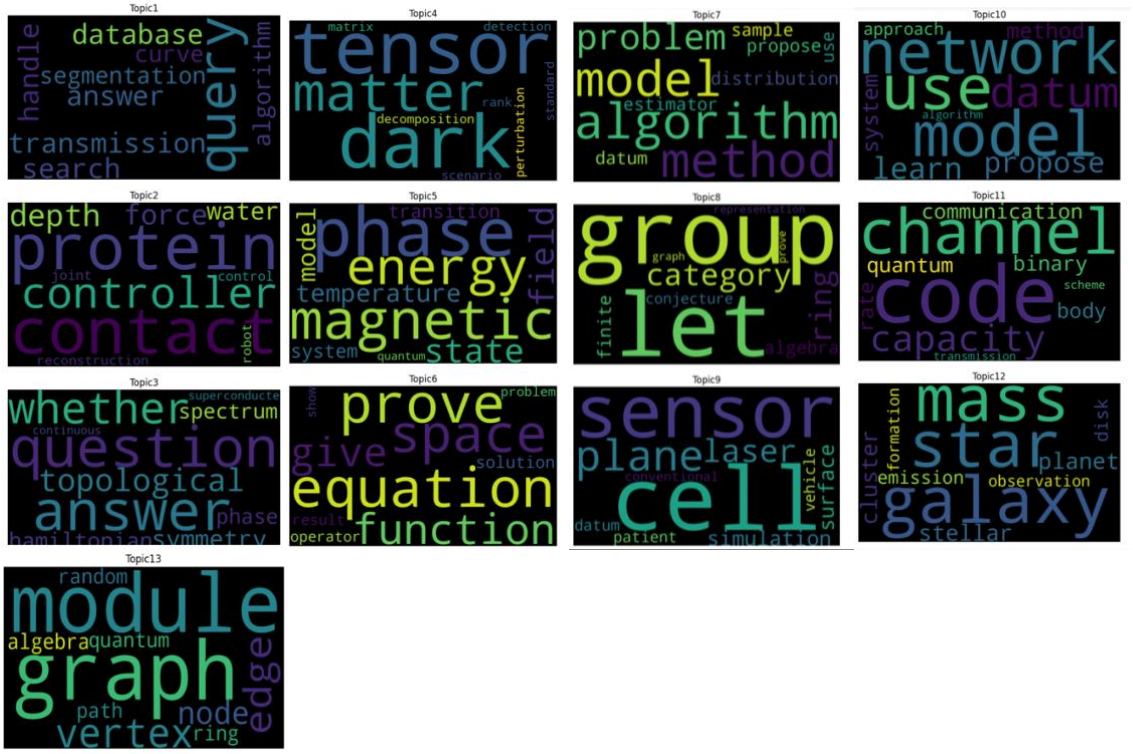
3.1.2 LDA algoritmo pritaikymas

Pirmiausia, pritaikėme LDA algoritmą tekstui, kuris buvo be nereikšmingų žodžių, su pagrindinėmis formomis ir vektorizuotas pagal TF - IDF. Galimų temų skaičius buvo pasirinktas 13, gavome kiekvienai temai priskirtą 10 žodžių, kurie labiausiai tinka tam tikrai temai (žodžiai išdėstyti svarbumo tvarka mažėjančiai):

```
Topic 0:
query transmission database answer search handle segmentation algorithm since curve
Topic 1:
protein contact controller depth force water joint reconstruction robot control
Topic 2:
question answer whether topological symmetry hamiltonian spectrum phase superconducte continuous
Topic 3:
tensor dark matter decomposition detection rank standard perturbation matrix scenario
Topic 4:
phase magnetic energy state field temperature model system transition quantum
Topic 5:
equation prove space function give solution operator result problem show
Topic 6:
algorithm model method problem distribution propose estimator datum use sample
Topic 7:
group let category ring algebra finite conjecture representation prove graph
Topic 8:
cell sensor plane laser simulation surface patient datum vehicle conventional
Topic 9:
network model use datum learn propose method system approach algorithm
Topic 10:
code channel capacity communication quantum body binary rate transmission scheme
Topic 11:
star galaxy mass planet stellar cluster emission disk observation formation
Topic 12:
graph module vertex edge node algebra quantum path random ring
```

2 pav. Naudojant LDA ir TF-IDF algoritmus gauti pagrindiniai temų žodžiai

Toliau, atvaizdavome, kurie žodžiai kiekvienoje temoje yra svarbiausi (kuo žodžio šriftas didesnis, tuo jis turi didesnį svorį būtent toje temoje)



3 pav. Temų žodžių debesys, naudojant LDA ir TF-IDF

Iš *2 pav.* galime aiškiai matyti, kurie temos žodžiai yra svarbiausi, pvz.: 5-oje temoje eina tokie žodžiai kaip: fazė, energija bei magnetinis, ši tema galėtų būti susijusi su fizika. 6-oje temoje svarbiausi žodžiai būtų: įrodymas, lygtis, funkcija, šie žodžiai primena tema, kuri būtų susijusi su matematika – lygčių sprendimu. 11-oje temoje akcentuojami žodžiai tokie kaip: kodas, talpa, kvantinis, tai galėtų būti tema apie fiziką bei programavimą. Vertinant temas pagal žodžių debesį, matyti, kad kai kurios temos turi panašių žodžių. Pavyzdžiui, žodžiai: algoritmas, kvantinis ir modelis, algebra pasikartoja keliuose temų debesiuose, tai rodo, jog ne visos temos gali būti gerai atskirtos.

Šio algoritmo coherence balas yra 0,35.

Toliau pateikėme LDA algoritmo pritaikymą, pašalinome nereikšmingus žodžius, atlikome lemavimą ir pritaikėme žodžių maišo vektorizavimo būdą.

```
Topic 0:
equation solution code nonlinear scheme boundary flow motion use method
Topic 1:
phase energy state quantum temperature transition magnetic system show material
Topic 2:
theory function operator space type field show property representation order
Topic 3:
problem algorithm method function optimization gradient propose stochastic optimal control
Topic 4:
algorithm problem matrix method propose game power time number strategy
Topic 5:
group result space give bound case also paper define show
Topic 6:
model distribution sample datum method estimate use parameter estimator estimation
Topic 7:
graph set number show give point prove result class proof
Topic 8:
network structure system two model measure social population cell different
Topic 9:
use network datum learn model propose method approach task neural
Topic 10:
use system dynamic field time robot present control process study
Topic 11:
cluster star galaxy find time use pattern region formation observation
Topic 12:
model mass agent rate line velocity use al value simulation
```

4 pav. Naudojant LDA ir žodžių maišo algoritmus gauti pagrindiniai temų žodžiai

Atvaizdavome, kurie žodžiai kiekvienoje temoje yra svarbiausi naudojant žodžių debesį.

Iš žodžių debesų, galime matyti, jog temos panašios, žodžiai vėl kartojasi debesyse (pvz.: metodas, modelis, tinklas). 1-oje ir 3-oje temoje vyraujantys žodžiai primena temas apie

skaičiavimus bei matematines formules. 12-oje temoje galėtų būti rašoma apie astronomiją, svarbiausi žodžiai: klasteris, žvaigždė, galaktika. Kaip ir taikant TF – IDF vektorizavimo būdą, taip ir šį, temos nėra lengvai atskiriamos bei interpretuojamos.

3.2 LSA

3.2.1 LSA algoritmas

Latentinė semantinė analizė (LSA) yra natūralios kalbos apdorojimo ir informacijos išgavimo algoritmas, kuris padeda atrasti paslėptus santykius tarp žodžių didelėje teksto kolekcijoje. Tai yra dimensijų sumažinimo technika, dažnai naudojama teksto analizei ir dokumentų panašumo nustatymui. Pagrindiniai žingsniai:

1. Konstruojama matrica, vadinamą žodžių-dokumentų matrica.
2. Atliekama singuliariųjų reikšmių dekompozicija.[1]
3. Sumažinama dimensija.
4. Naudojant sumažintas matricas, sukuriama nauja semantinė erdvė, kurioje žodžiai ir dokumentai yra atvaizduoti kaip vektoriai.
5. Gauti vektoriai naudojami, kad palyginti dokumentus ir nustatytų jų panašumą semantinėje erdvėje.

[1] Tai matematinė technika, skirta išskaidyti matricą į tris dalis: kairiuosius singuliarius vektorius, singuliariųjų reikšmių matricą ir dešiniuosius singuliarius vektorius. Tai leidžia efektyviai sumažinti dimensijas ir išgauti svarbias matricos savybes.

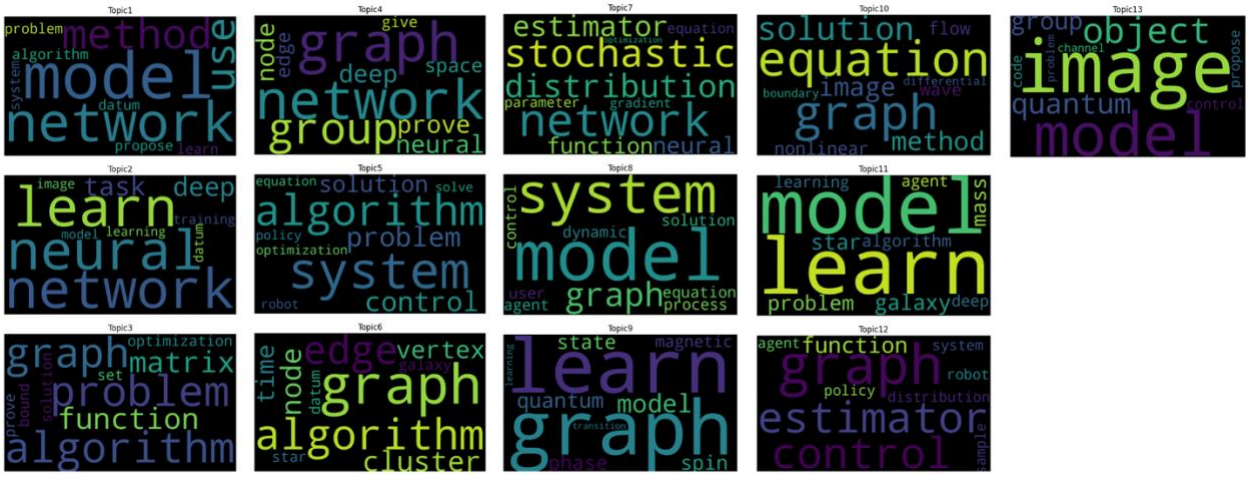
3.2.2 LSA algoritmo pritaikymas

Kaip ir ankstesniam algoritmui, LSA buvo pritaikytas nereikšmingų žodžių pašalinimas bei lemavimas be to papildomai buvo pritaikytas vienas iš vektorizavimo būdų - TF – IDF arba žodžių maišas. Pirmiausia pažiūrėkime rezultatus, kai pritaikytas TF – IDF vektorizavimo būdas.

```
Topic 0:
model network method use algorithm problem datum propose learn system
Topic 1:
network learn neural deep task training datum image learning model
Topic 2:
algorithm problem graph function matrix optimization solution prove bound set
Topic 3:
network graph group neural node deep prove space edge give
Topic 4:
system algorithm problem control solution optimization robot equation solve policy
Topic 5:
graph algorithm edge cluster node vertex time galaxy datum star
Topic 6:
network stochastic distribution estimator neural function parameter equation gradient optimization
Topic 7:
model system graph equation control dynamic solution user process agent
Topic 8:
graph learn model quantum phase state spin magnetic transition learning
Topic 9:
equation graph solution image method nonlinear wave flow differential boundary
Topic 10:
learn model galaxy star problem mass algorithm learning agent deep
Topic 11:
graph estimator control function distribution system policy robot agent sample
Topic 12:
image model object quantum group control propose code channel problem
```

5 pav. Naudojant LSA ir TF-IDF algoritmus gauti pagrindiniai temų žodžiai

pav. pateikti žodžiai, svarbumo tvarka (pirmas – svarbiausias, antras – mažiau svarbus ir t. t.), aprašantys temas. Taip pat galime pažiūrėti į žodžių debesis.



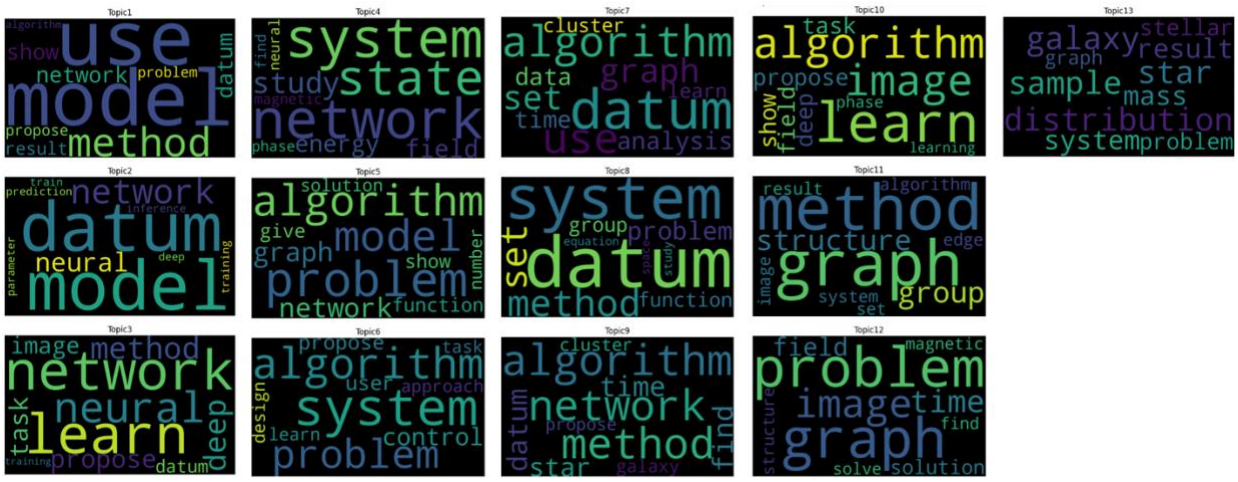
6 pav. Temų žodžių debesys, naudojant LSA ir TF-IDF

Iš pav. galime pamatyti, jog nemažai temų turi tuos pačius žodžius, tokius kaip model, network, graph ir pan., todėl kai kada tai gali sukelti problemų jas atskiriant. Pavyzdžiui Topic4, Topic6 ir Topic9 būtų galima priskirti vienodai temai – grafams, Topic3 ir Topic7 – optimizavimo temai.

Ne tokį gerą temų atskyrimą parodo ir ne toks geras coherance score, kuris siekia tik 0,32. Panašius rezultatus galime pamatyti ir pasirinkus žodžių maišo algoritmą.

```
Topic 0:
model use method network datum show result propose problem algorithm
Topic 1:
model datum network neural prediction deep inference training train parameter
Topic 2:
network learn neural method deep propose task image datum training
Topic 3:
network system state study field energy phase find neural magnetic
Topic 4:
problem algorithm model network graph function give show solution number
Topic 5:
system algorithm problem control propose user approach learn design task
Topic 6:
datum algorithm use graph set analysis time data learn cluster
Topic 7:
datum system method set problem function group equation space study
Topic 8:
algorithm method network time datum find star galaxy cluster propose
Topic 9:
learn algorithm image propose field task deep show phase learning
Topic 10:
graph method group structure algorithm system result edge set image
Topic 11:
problem graph image time field solution magnetic structure find solve
Topic 12:
distribution galaxy star sample system mass result problem stellar graph
```

7 pav. Naudojant LSA ir žodžių maišo algoritmus gauti pagrindiniai temų žodžiai



8 pav. Temų žodžių debesys, naudojant LSA ir žodžių maišą

pav. galime matyti, jog pritaikius žodžių maišą gauti šiek tiek geresni rezultatai. Nors ir matomi pasikartojantys žodžiai, tačiau šįkart matomas geresnis temų atsiskyrimas. Tai parodo ir šiek tiek geresnis coherenace score – 0,32. Galime matyti, jog Topic1 galima priskirti modeliavimo temai, Topic2, Topic3 ir Topic10 – neuroninių tinklų, Topic4 ir Topic6 – sistemų ir tyrimų, Topic5 – algoritmų grafiniuose tinkluose, Topic7 ir Topic8 – duomenų analizės, Topic9 – algoritmų taikymo erdvėje, Topic11 ir Topic12 – grafų, Topic13 – astronomijos temoms.

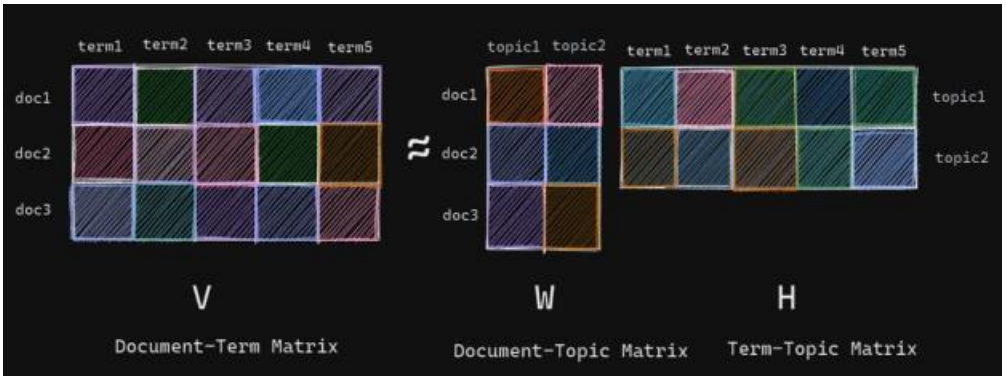
Apibendrindami galime teigti, jog abu vektorizavimo būdai neveikė labai gerai su LSA algoritmu – daugelis žodžių pasikartojo, temos buvo panašios, tačiau remiantis coherance score ir žodžių debesimis, šiek tiek geresnis rezultatas matomas naudojant žodžių maišo vektorizavimo būdą.

3.3 NMF

3.3.1 NMF algoritmas

Ne neigiamų matricų faktorizavimas (trumpinamas *NMF*) yra statistinis dekompozicinis metodas, padedantis sumažinti įvesties tekstų matmenis. Viduje jis naudoja faktorinės analizės metodą, kuris žodžiams suteiktų mažesnį svorį, kai jie turi mažesnį loginį ryšį su likusiais teksto žodžiais. Priklauso tiesinės algebros algoritmams, kurie yra skirti atpažinti latentinius kintamuosius ar paslėptas struktūras duomenyse. Šis metodas yra populiarus, nes pats sugeba išskirti faktorius [2].

Algoritmo dalys:



9 pav. NMF algoritmo kintamieji (šaltinis [1])

V yra mūsų įvesties matrica – eilutės atspindi dokumentus, o stulpeliai – žodžius. Susikertantys langeliai žymės, kiek žodžių kokiame dokumente turime. W yra dokumento – temos matrica, kuri parodo temų pasiskirstymą tarp visų teksto rinkinių, tai surastos temos tarp dokumentų. H yra terminų ir temų matrica, kuri parodo žodžių reikšmingumą visose temose. Būtina, kad visi W ir H įrašai būtų neneigiami [1][2].

Algoritmo veikimo principas:

Siekiame surasti tokius W ir H, kad skirtumas tarp V ir $W \cdot H$ būtų kuo mažesnis. Dažniausiai W ir H matricos yra inicializuojamos atsitiktiniu būdu. Jų reikšmės bus keičiamos tol, kol bus pasiektas maksimalus iteracijų skaičius arba matricų sandauga priartės prie V per minimalią paklaidą, arba aproksimacijos paklaida sukonverguos [1][2]. Sprendžiamas uždavinys yra:

$$min\|V - W \cdot F\|_F, \qquad kur\ \|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}\ yra\ Frobenijaus\ norma.$$

Algoritmo privalumai:

- Kadangi priklauso tiesinės algebros algoritmams, NMF gerai veikia su trumpesniu tekstu.
- Remiasi vektorizuotais duomenimis, o ne neapdorotais žodžių dažniais.
- Tinkamas algoritmas, kai duomenyse turime triukšmo [3].
- Lyginant su LDA yra lengviau manipuliuoti parametrais ir surasti optimaliausią parametrų rinkinį [4].
- NMF būdas yra lengvai interpretuojamas, nes visi matricų W ir H elementai yra neneigiami, o didesnis balas, priskirtas temos žodžiui, reiškia didesnį svarbumą [1].

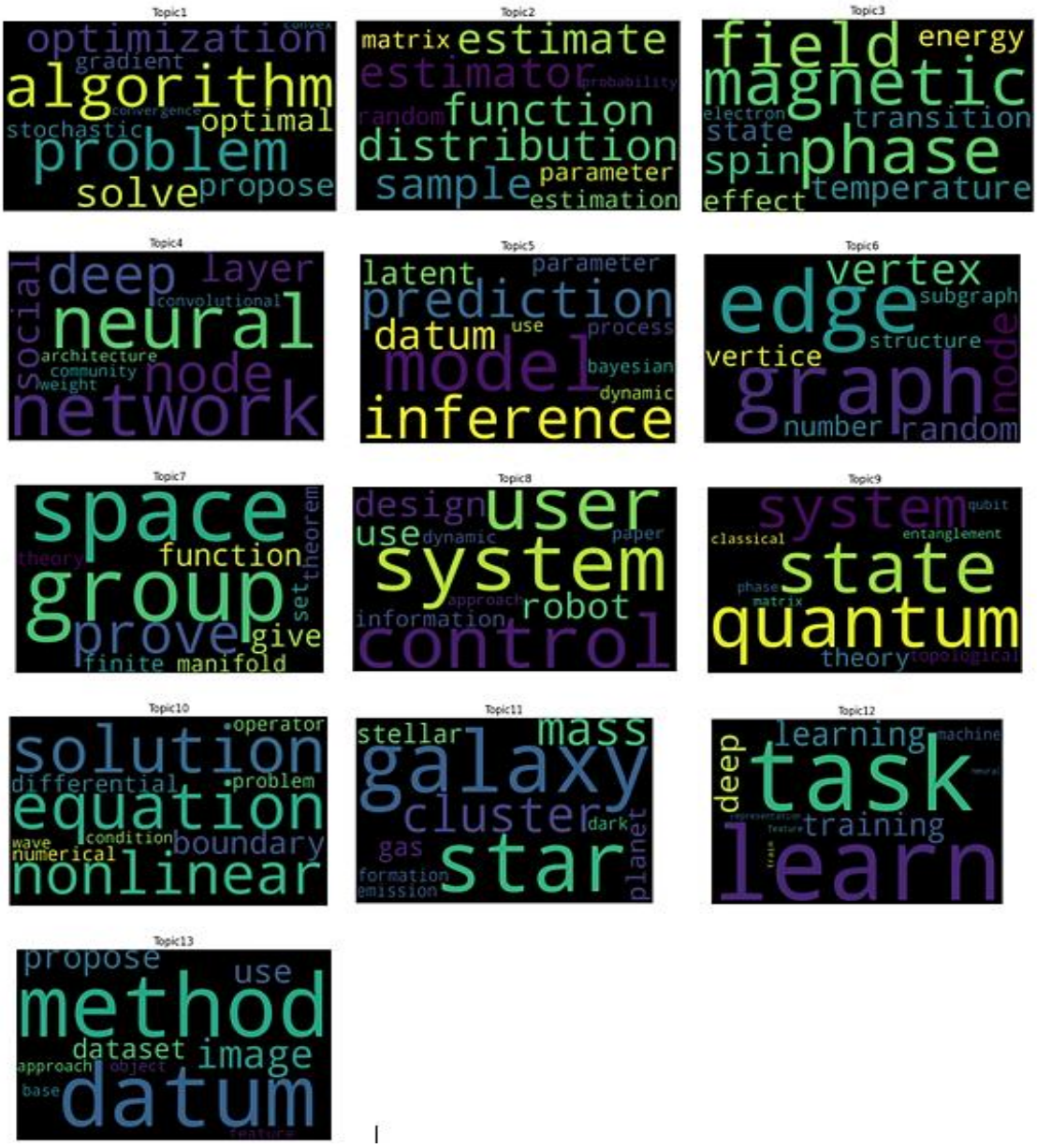
3.3.2 NMF algoritmo pritaikymas

Toliau bus pateiktas NMF algoritmo pritaikymas, kai temų skaičius yra 13, nereikšmingi žodžiai buvo pašalinti, atliktas lemavimas ir žodžiai vektorizuoti, taikant TF – IDF vektorizavimo metodą.

```
Topic 1: convex,convergence,stochastic,gradient,optimal,propose,solve,optimization,problem,algorithm
Topic 2: probability,matrix,random,parameter,estimation,estimate,function,sample,estimator,distribution
Topic 3: electron,effect,state,energy,transition,temperature,spin,field,phase,magnetic
Topic 4: weight,community,convolutional,architecture,social,layer,node,deep,neural,network
Topic 5: dynamic,use,bayesian,process,parameter,latent,datum,prediction,inference,model
Topic 6: structure,subgraph,number,graphs,vertice,random,node,vertex,edge,graph
Topic 7: theory,theorem,set,manifold,finite,function,give,prove,space,group
Topic 8: approach,paper,dynamic,information,use,robot,design,user,control,system
Topic 9: qubit,phase,classical,matrix,entanglement,topological,theory,system,state,quantum
Topic 10: wave,condition,numerical,operator,problem,differential,boundary,nonlinear,solution,equation
Topic 11: dark,emission,formation,gas,planet,stellar,mass,cluster,star,galaxy
Topic 12: feature,train,neural,representation,machine,training,deep,learning,task,learn
Topic 13: base,object,feature,approach,dataset,use,propose,image,datum,method
```

10 pav. Naudojant NMF ir TF-IDF algoritmus gauti pagrindiniai temų žodžiai

() pateikti žodžiai, kurie labiausiai aprašo temas didėjimo tvarka. Toliau pateiksime kiekvienos temos žodžių debesis, kuo didesnis žodis, tuo jis labiau nusako temą.



11 pav. Temų žodžių debesys, naudojant NMF ir TF-IDF

Iš () matome, jog 1 temoje gali būti nagrinėjamos optimizavimo problemos. 2 tema šiek mažiau aiški, tačiau joje gali būti parametrų vertinimo problemos nagrinėjamos. 3 tema susijusi su fizika ir astronomija - magnetinio lauko tyrimai. 4 temoje nagrinėjami neuroniniai tinklai, o 5 temoje nagrinėjami modeliai ir prognozavimas. 6 temoje yra nagrinėjami grafai. 7 tema šiek tiek neaiškesnė, nes joje atsiranda žodis erdvė, kuris daug nulemia temos priskyrimą, o likusieji žodžiai susiję su matematiniu įrodinėjimu. 8 temoje priklauso susijusios temos su informatika, informacinėmis technologijomis. 9 temos negalėtume aiškiai apibendrinti. 10 tema nagrinėja

matematines problemas, kurios yra susijusios su netiesinėmis lygtimis. 11 temoje taip pat yra astronomija – žvaigždynai. 12 tema kaip ir 4 temoje pagrindinės temos yra neuroniniai tinklai. 13 temoje labiausiai nagrinėjami ir tiriami duomenys. Apibendrintai galime teigti, jog didžioji dalis temų gerai atsiskiria, tačiau turime keletą pasikartojančių temų bei temų, kurių negalime sutrauktai apibendrinti. Galime įtarti, jog temų skaičius yra per didelis.

Šio algoritmo *coherence* balas (loginio ryšio įvertinimas) yra 0,58.

Toliau bus pateiktas NMF algoritmo pritaikymas, kai temų skaičius yra 13, nereikšmingi žodžiai buvo pašalinti, atliktas lemavimas ir žodžiai vektorizuoti, taikant žodžių maišos vektorizavimo metodą.

```
Topic 1: information,application,sample,time,provide,approach,set,data,analysis,datum
Topic 2: variable,bayesian,dynamic,distribution,inference,prediction,propose,process,parameter,model
Topic 3: show,architecture,layer,information,structure,social,node,deep,neural,network
Topic 4: temperature,effect,transition,study,quantum,energy,magnetic,phase,state,field
Topic 5: efficient,paper,new,number,performance,show,base,time,propose,algorithm
Topic 6: base,time,propose,present,design,user,dynamic,paper,control,system
Topic 7: paper,matrix,demonstrate,sample,apply,estimate,new,base,propose,method
Topic 8: performance,test,base,design,information,present,technique,result,approach,use
Topic 9: case,study,also,set,group,space,give,show,result,function
Topic 10: training,dataset,approach,learning,feature,deep,propose,task,image,learn
Topic 11: vertice,give,random,set,node,vertex,structure,number,edge,graph
Topic 12: set,consider,control,approach,time,optimal,optimization,solve,solution,problem
Topic 13: formation,large,sample,distribution,stellar,find,cluster,mass,star,galaxy
```

12 pav. NMF ir žodžių maišos metodu gauti kiekvieną temą labiausiai nusakantys žodžiai

() matome kiekvienos temos pagrindinius žodžius, kurie yra išdėstyti didėjimo tvarka pagal svarbą temai. Toliau bus pateikti kiekvienos temos žodžių debesys, kuo didesnis žodis – tuo jis labiau nusako tą temą.

13 pav. Temų žodžių debesys, naudojant NMF ir žodžių maišą

Iš () matome, jog 1 temoje nagrinėjama duomenų analizė. 2 temoje modelių parametrų vertinimas, susiję procesai su modelių taikymu. 3 temą galėtumėme apibendrinti kaip neuroniniai tinklai. 4 tema priklauso astronomijai – magnetiniai laukai. 5 tema atrodo platesnė, ją galėtumėme trumpinti kaip algoritmų taikymas. 6 temoje nagrinėjamos sistemos, jų valdymas. 7 temoje nagrinėjami nauji modeliai. 8 tema susijusi su informacijos apdorojimu. 9 temoje analizuojamos

funkcijos. 10 temoje nagrinėjami vaizdų atpažinimo uždaviniai. 11 temoje sprendžiami uždaviniai susiję su grafais. 12 temoje bandoma rasti optimizavimo uždavinių sprendiniai ir 13 tema susijusi su astronomija – žvaigždėmis. Temos nėra lengvai atskiriamos ir interpretuojamos.

Šio algoritmo *coherence* balas (loginio ryšio įvertinimas) yra 0,60.

Apibendrintai galime teigti, jog naudojant žodžių maišą gaunamos mažiau atskiriamos temos nei lyginant su TF – IDF vektorizavimo algoritmu, nors ir *coherence* balo įvertis yra gaunamas blogesnis TF – IDF vektorizavimo būdu (0,58 ir 0,60).

3.4 BERTopic

3.4.1 BERTopic algoritmas

BERTopic yra temų modeliavimo metodas, kuris naudoja transformerius ir pasirinktą klase pagrįstą TF-IDF, kad sukurtų tankius klasterius, leidžiančius lengvai interpretuoti temas, išlaikant svarbius žodžius temų aprašymuose.

Transformeris yra gilaus mokymosi modelis, pagrįstas savęs įsidėmėjimo mechanizmu, kuris skirtingai įvertina kiekvienos įvesties duomenų dalies svarbą.

BERTopic algoritmas yra paremtas BERT (dvikrypčio kodavimo atvaizdai iš transformerių) transformerio architektūra. Šis algoritmas apima **tris** pagrindinius etapus:

- **Dokumento įterpimas**: pirmiausia reikia sukurti dokumento įterpimą. Numatytasis metodas - naudoti sakinių-transformerių modelius, kuriuos galima naudoti tiek angliškiems, tiek daugiakalbiams dokumentams.
- **Dokumentų klasterizavimas**: ką tik sukurti dokumentų įterpiniai yra didelės dimensijos, todėl prieš juos klasterizuojant reikia atlikti dimensijos mažinimą naudojant UMAP, kuris išsaugo tiek vietinę, tiek globalią įterpinių struktūrą, tada galima taikyti tankumu pagrįstą klasterizavimo metodą, pvz., HDBSCAN, kad sukurti teminius klasterius ir, jei įmanoma, nustatyti nukrypimus.
- **Temos atvaizdavimas**: norint atvaizduoti kiekvieną temą, galima pakeisti TF-IDF balą. Kai dokumentų rinkiniui taikomas TF-IDF, lyginama žodžių svarba tarp dokumentų. Jei visi klasterio dokumentai būtų laikomi vienu dokumentu ir būtų

skaičiuojamas TD-IDF, gautume klasteryje esančių žodžių svarbos balus. Jei išskirtume svarbiausius žodžius kiekviename klasteryje, būtų gauti temų aprašymai. Šis metodas vadinamas klase pagrįstu TF-IDF [5]

3.4.2 BERTopic algoritmo pritaikymas

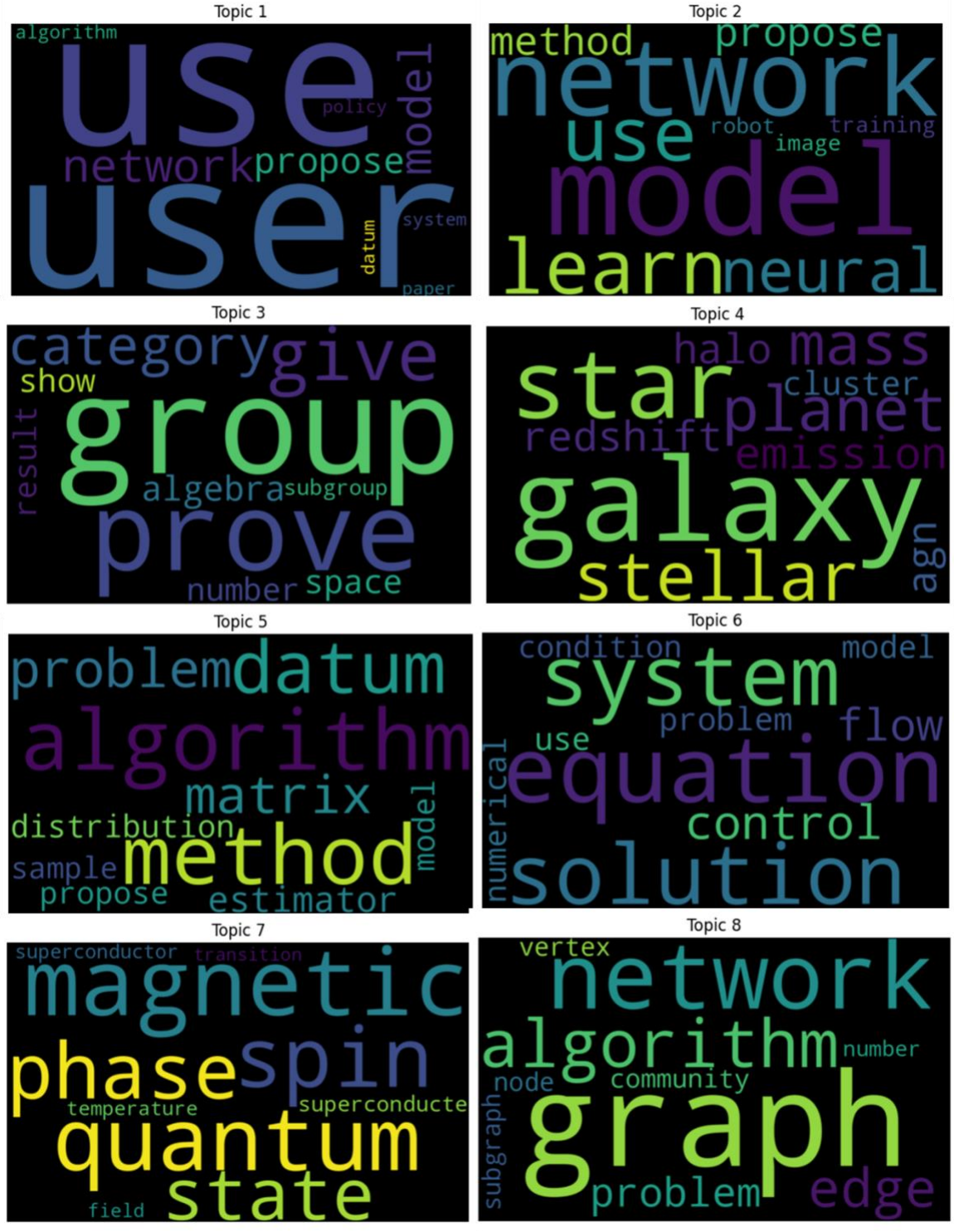
Pirmiausia, pritaikėme BERTopic algoritmą tekstui, kuris buvo be nereikšmingų žodžių, su pagrindinėmis formomis ir vektorizuotas pagal TFIDF. Iš pateiktos lentelės () matome, jog iš viso yra 14 temų, tačiau -1 tema nurodo išskirtis, todėl ją ignoruojame. Matome, jog dažniausia tema dokumente yra susijusi su šiais žodžiais: modelis, metodas, pasiūlymas, tinklas.

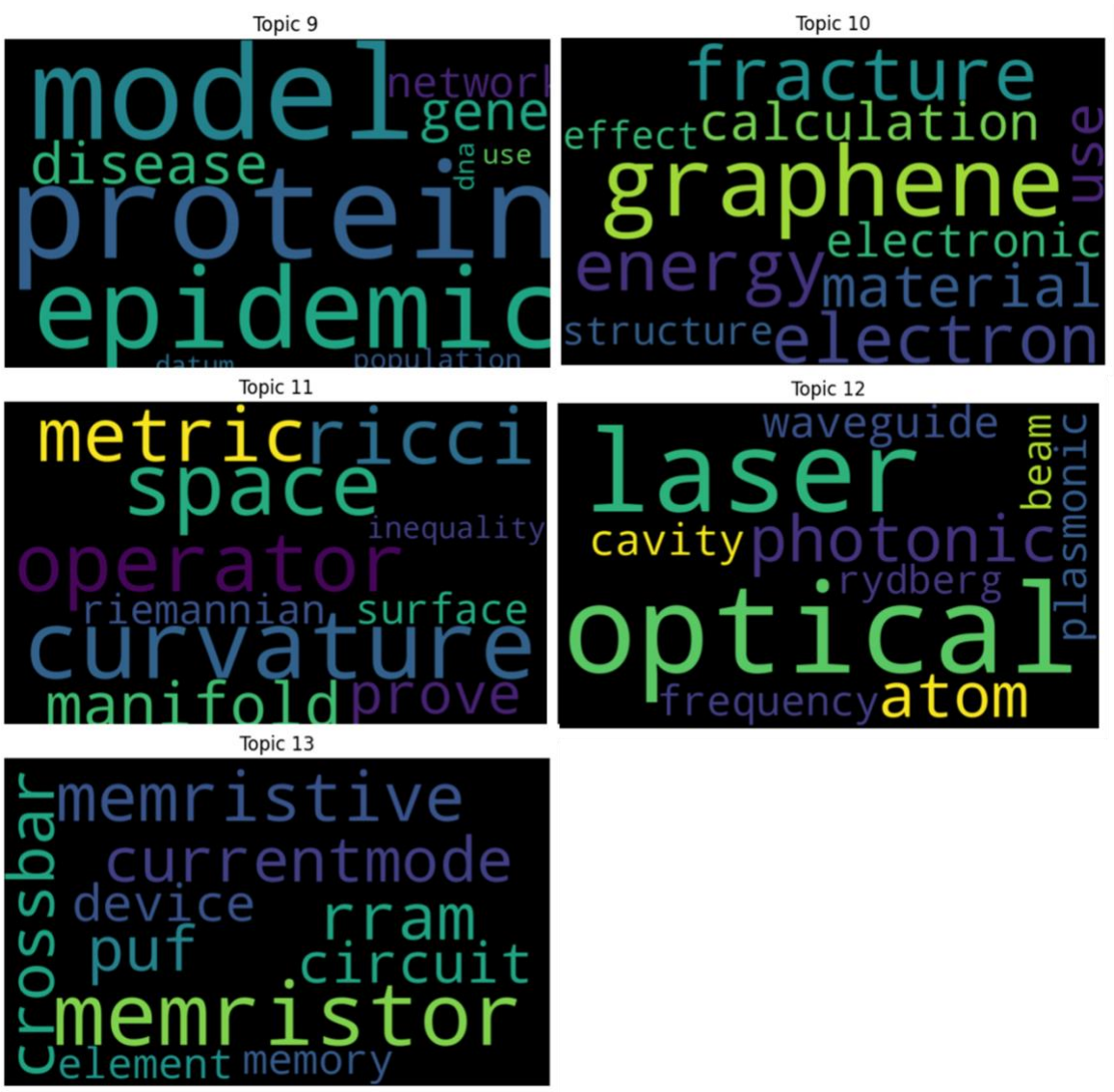
	Topic	Count	Name	Representation	Representative_Docs
0	-1	9138	-1_model_use_method_show	[model, use, method, show, problem, algorithm,...	[challenge take many variable account optimiza...
1	0	2699	0_user_use_network_model	[user, use, network, model, propose, algorithm...	[various economic environment people observe s...
2	1	2519	1_model_network_learn_use	[model, network, learn, use, neural, propose, ...	[deep learning become state art approach many ...
3	2	1076	2_group_prove_give_category	[group, prove, give, category, algebra, space,...	[article construct three explicit natural subg...
4	3	1069	3_galaxy_star_stellar_planet	[galaxy, star, stellar, planet, mass, emission...	[mass function galaxy cluster sensitive tracer...
5	4	1025	4_algorithm_method_datum_problem	[algorithm, method, datum, problem, matrix, es...	[paper consider general matrix factorization m...
6	5	936	5_equation_solution_system_control	[equation, solution, system, control, flow, pr...	[boundary value problem complete second order ...
7	6	759	6_magnetic_quantum_spin_phase	[magnetic, quantum, spin, phase, state, superc...	[report magnetic thermodynamic property mo mag...
8	7	475	7_graph_network_algorithm_edge	[graph, network, algorithm, edge, problem, com...	[study timevarye dynamic network graphs fundam...
9	8	401	8_protein_model_epidemic_disease	[protein, model, epidemic, disease, gene, netw...	[study challenge apply deep learn gene express...
10	9	329	9_graphene_fracture_energy_electron	[graphene, fracture, energy, electron, materia...	[direct growth graphene semiconducting insulat...
11	10	290	10_curvature_space_operator_ricci	[curvature, space, operator, ricci, metric, ma...	[show form connected sum homotopy sphere jconn...
12	11	244	11_optical_laser_photonic_atom	[optical, laser, photonic, atom, waveguide, ca...	[scalable quantum photonic system require effi...
13	12	12	12_memristor_memristive_rram_crossbar	[memristor, memristive, rram, crossbar, curren...	[interest memristor rise due possible applicat...

Temų santrauka

Toliau vizualizuojame temas braižant stulpelines diagramas iš TF-IDF balų bei naudojant žodžių debesį.

Žodžių svarba kiekvienoje temoje naudojant TF - IDF


Topic 1
use
user
algorithm
model
policy
network
propose
system
datum
paper
Topic 2
method
propose
network
use
robot
training
image
model
learn
neural
Topic 3
category
give
show
group
result
algebra
subgroup
prove
number
space
Topic 4
halo
mass
star
cluster
planet
redshift
emission
galaxy
stellar
agn
Topic 5
problem
datum
algorithm
matrix
distribution
method
model
sample
propose
estimator
Topic 6
condition
model
system
problem
flow
use
equation
numerical
control
solution
Topic 7
superconductor
transition
magnetic
phase
spin
temperature
superconducte
quantum
state
field
Topic 8
vertex
network
algorithm
number
node
community
graph
subgraph
problem
edge



Žodžių debesis naudojant TF - IDF

Iš diagramų matome kurie žodžiai yra svarbiausi kiekvienoje temoje. Pavyzdžiui 7, 9 ir 12 temose žodžiai grafas, grafinas ir memristorius turėjo didžiausią balą. Galime pastebėti, jog atsiskiria temos, susijusios su matematika, statistika, fizika, biologija. Vis dėlto iš žodžių debesio matome, jog kai kuriose temose dominuoja panašūs žodžiai, jų sinonimai (1 temoje žodžiai naudojimas ir naudotojas). Tai gali reikšti, jog šis algoritmas nėra labai tinkamas.

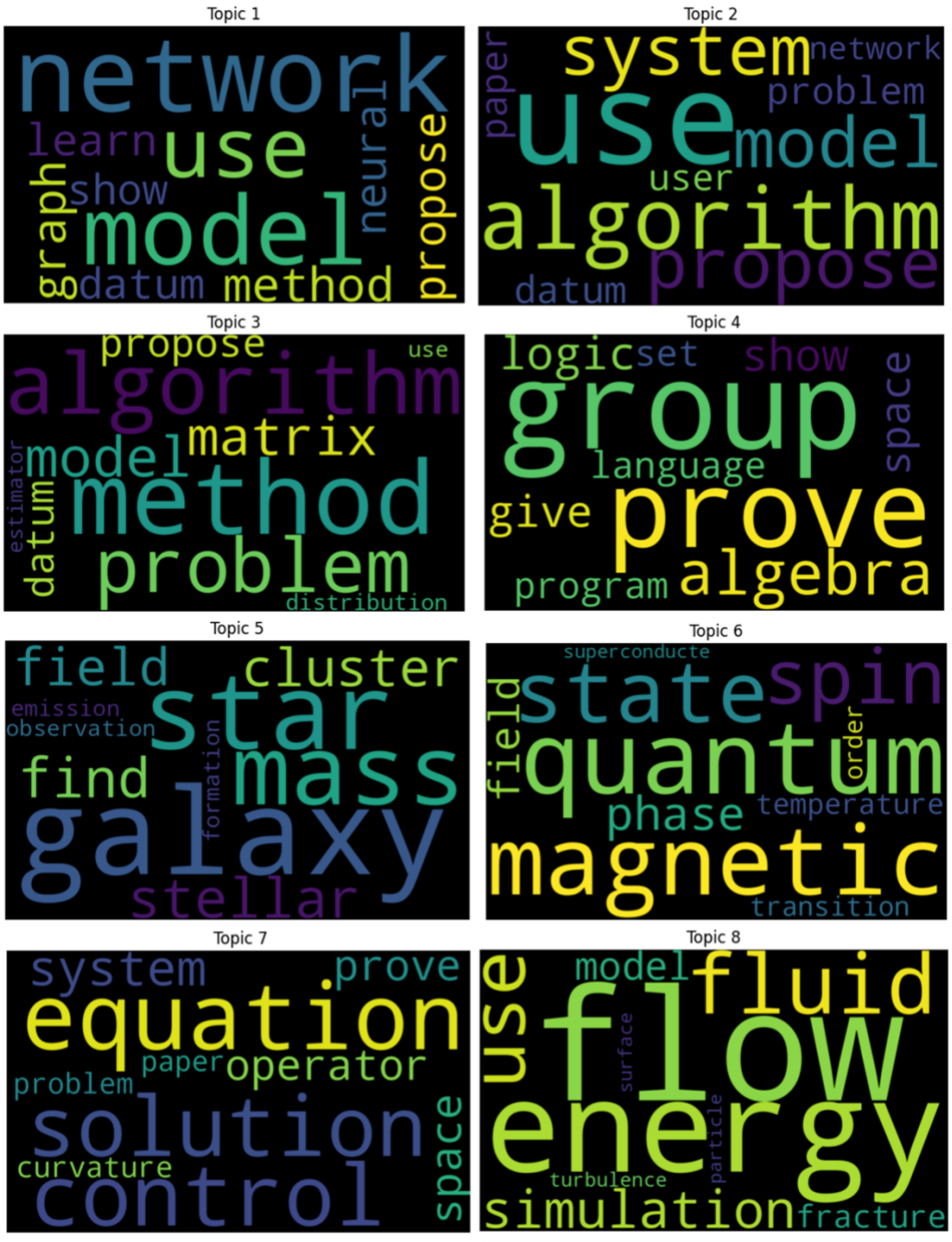
Galiausiai, suskaičiuojame *coherence* balą – 0,575. Taigi modelis veikia gana prastai.

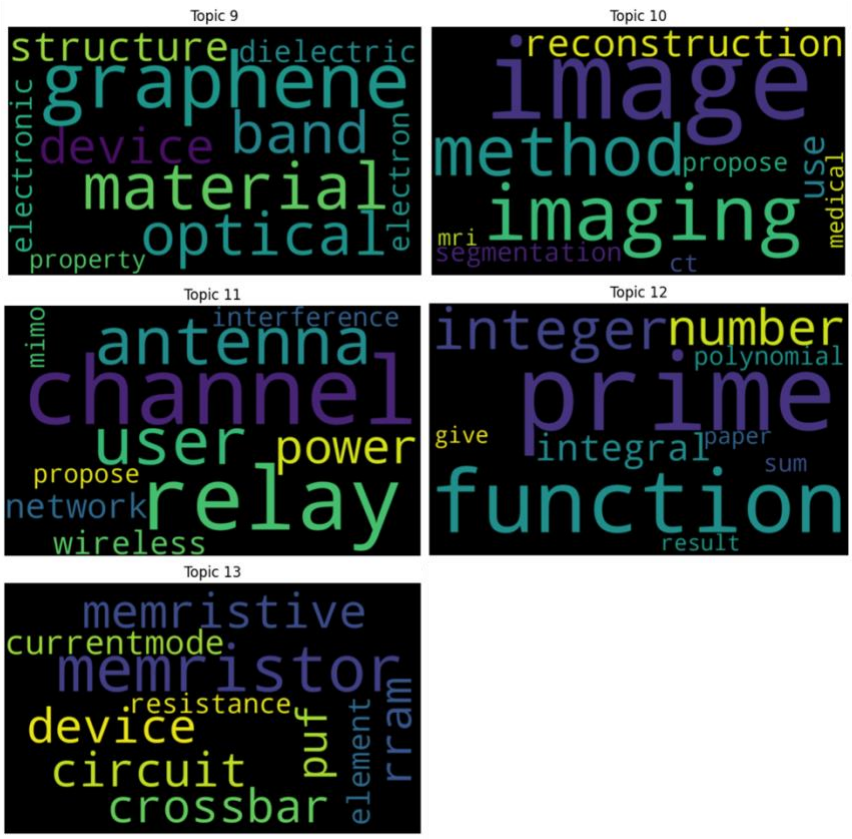
Taip pat BERTopic algortimas buvo pritaikytas tekstui, kuris buvo be nereikšmingų žodžių, su pagrindinėmis formomis ir vektorizuotas pagal *CountVectorizer*. Iš lentelės () matome, jog dažniausia tema yra susijusi su šiais žodžiais: tinklas, modelis naudojimas, metodas, grafas.

	Topic	Count	Name	Representation	Representative_Docs
0	-1	8916	-1_model_use_result_show	[model, use, result, show, method, problem, da...	[compute integral function expectation random ...
1	0	3373	0_network_model_use_method	[network, model, use, method, graph, propose, ...	[interpretability deep neural network recently...
2	1	2342	1_use_algorithm_system_propose	[use, algorithm, system, propose, model, probl...	[paper propose novel rank framework collaborat...
3	2	1394	2_method_algorithm_problem_model	[method, algorithm, problem, model, matrix, pr...	[article investigate large sample property mod...
4	3	1273	3_group_prove_algebra_logic	[group, prove, algebra, logic, show, give, spa...	[introduce notion depth finite group g define ...
5	4	955	4_galaxy_star_mass_stellar	[galaxy, star, mass, stellar, find, cluster, f...	[present millimetre dust emission measurement ...
6	5	841	5_quantum_magnetic_state_spin	[quantum, magnetic, state, spin, phase, field,...	[antiferromagnetic ising chain transverse long...
7	6	817	6_equation_control_solution_system	[equation, control, solution, system, space, o...	[study optimal boundary control problem twodim...
8	7	374	7_flow_energy_fluid_use	[flow, energy, fluid, use, simulation, model, ...	[numerical experimental turbulence simulation ...
9	8	260	8_graphene_material_optical_band	[graphene, material, optical, band, device, st...	[tune band gap twodimensional material great i...
10	9	182	9_image_imaging_method_reconstruction	[image, imaging, method, reconstruction, use, ...	[inherent noise observe eg scan binary documen...
11	10	121	10_channel_relay_antenna_user	[channel, relay, antenna, user, power, network...	[paper concerned channel estimation problem mu...
12	11	112	11_prime_function_integer_number	[prime, function, integer, number, integral, p...	[paper use refined approximation chebyshevs va...
13	12	12	12_memristor_device_memristive_circuit	[memristor, device, memristive, circuit, cross...	[interest memristor rise due possible applicat...

Temų santrauka

Žodžių svarba kiekvienoje temoje naudojant žodžių maišą


Topic 1
network
use
model
learn
graph
show
datum
method
neural
propose
Topic 2
system
network
use
paper
problem
model
user
algorithm
datum
propose
Topic 3
propose
use
algorithm
matrix
model
method
estimator
datum
problem
distribution
Topic 4
logic
set
show
group
space
language
give
prove
program
algebra
Topic 5
field
cluster
star
emission
observation
mass
find
formation
galaxy
stellar
Topic 6
superconducte
state
spin
field
quantum
order
phase
temperature
magnetic
transition
Topic 7
system
prove
equation
paper
operator
problem
solution
curvature
space
control
Topic 8
model
fluid
use
flow
surface
energy
particle
turbulence
simulation
fracture



Žodžių debesis naudojant žodžių maišą

Iš stačiakampių diagramų ir žodžių debesio matome, jog temų žodžiai šiek tiek skiriasi. Tačiau temos vis tiek išlieka tos pačios kaip ir prieš tai. Taip pat kaip ir naudojant TF-IDF vektorizavimą temose galima rasti žodžių sinonimų, todėl galima daryti išvadą, jog ir šis būdas nėra tinkamas.

Gautas *coherence* balas - 0,577, kuris yra šiek tiek aukštesnis nei naudojant TF-IDF metodą, tačiau modelis vis tiek nėra tinkamas.

4. ALGORITMŲ PALYGINIMAS

Visus taikytus algoritmus lyginsime pagal *coherence* balą. Gauti rezultatai taikant skirtingus vektorizavimo algoritmus pateikti ().

	TF-IDF	Žodžių maišas
LDA	0,35	0,32
LSA	0,32	0,33

NMF	0,58	0,60
BERTopic	0,58	0,58

Iš () matome, jog LDA ir LSA algoritmai pasirodė panašiai, jų *coherence* balai svyruoja nuo 0,32 iki 0,35 – tai nėra geri rezultatai. Taip pat galime matyti jog NMF ir BERTopic algoritmai irgi pasirodė panašiai, jų *coherence* balai svyruoja nuo 0,58 iki 0,60. Geriausiai pagal šį balą pasirodė ne neigiamų matricų faktorizavimo algoritmas su žodžių maišos vektorizavimo būdu (0,60).

5. IŠVADOS IR REKOMENDACIJOS

Naudojant NMF temų modeliavimo algoritmą ir naudojant žodžių maišą gaunamos mažiau atskiriamos temos nei lyginant su TF – IDF vektorizavimo algoritmu, nors ir *coherence* balo įvertis yra gaunamas blogesnis su TF – IDF vektorizavimo būdu (0,58 ir 0,60).

Naudojant LDA temų modeliavimo algoritmą ir TF – IDF, gaunamas geresnis *coherence* balo įvertis (0.35) nei naudojant algoritmą su žodžių maišu (0.32).

Naudojant LSA temų modeliavimo algoritmą bei vieną iš vektorizavimo būdų, rezultatas nebuvo labai geras – daugelis žodžių pasikartojo, temos buvo panašios, tačiau remiantis *coherence* balu ir žodžių debesimis, šiek tiek geresnis rezultatas matomas naudojant žodžių maišo vektorizavimo būdą (0,32), nei TF – IDF (0,31).

Naudojant BERTopic algoritmą su skirtingais vektorizavimo tipais *coherence* balas buvo gautas labai panašus (0,575, 0,577). Iš žodžių debesių buvo pastebėta, jog dominuoja temos susijusios su informacinėmis technologijomis, matematika, statistika, fizika ir biologija. Vis dėlto, šis algoritmas pasirodė nelabai tinkamas, nes žodžių debesyse dažnai buvo pavaizduoti žodžių sinonimai.

Atliekant tyrimą, geriausiai pasirodė pagal *coherence* balą NMF algoritmas, naudojant žodžių maišos vektorizavimo būdą.

Norint pagerinti gautus rezultatus, reikėtų ieškoti optimalių kiekvieno algoritmo parametrų reikšmių – taip būtų gaunamos labiau tarpusavyje atskiriamos temos. Taip pat dėl optimalių

parametrų suradimo tarp galimų temų atsirastų ir biologijos temą, kuri daugelyje algoritmų nebuvo surasta.

6. LITERATŪRA IR ŠALTINIAI

[1] B. Priya. *Topic Modeling Tutorial – How to Use SVD and NMF in Python,* 2023. https://www.freecodecamp.org/news/advanced-topic-modeling-how-to-use-svd-nmf-in-python/#topic-modeling-using-non-negative-matrix-factorization-nmf-

[2] C. Goyal. *Part15: Step by Step Guide to Master NLP – Topic Modelling using NMF*, 2021. https://www.analyticsvidhya.com/blog/2021/06/part-15-step-by-step-guide-to-master-nlp-topic-modelling-using-nmf/

[3] R. Egger, J. Yu. *A topic Modeling Comparison Between LDA, NMF, Top2Vec, and BERTopic to Demystify Twitter Posts*, 2022. https://www.frontiersin.org/articles/10.3389/fsoc.2022.886498/full

[4] A. Purpura. *Non – negative Matric Pactorization for Topic Modeling*, 2018. https://ceur-ws.org/Vol-2167/short5.pdf

[5] M. Nath. *Topic modeling algorithms*, 2023. https://medium.com/@m.nath/topic-modeling-algorithms-b7f97cec6005

[6] Loana. Latent Semantic Analysis: instuition, math, implemantation, 2020. https://towardsdatascience.com/latent-semantic-analysis-intuition-math-implementation-a194aff870f8#0c85

[7] A. Navlani. Latent Semantic Indexing using Sckit – Learn, 2021. https://machinelearninggeek.com/latent-semantic-indexing-using-scikit-learn/

[8] A. Navlani. Latent Semantic Analysis using Python, 2018. https://www.datacamp.com/tutorial/discovering-hidden-topics-python?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&utm_adgroupid=157156376311&utm_device=c&utm_keyword=&utm_matchtype=&utm_network=g&utm_adpostion=&utm_creative=6761360734

[9] Neha Seth. Part 3: Topic Modeling and Latent Dirichlet Allocation (LDA) using Gensim and Sklearn, 2021. https://www.analyticsvidhya.com/blog/2021/06/part-3-topic-modeling-and-latent-dirichlet-allocation-lda-using-gensim-and-sklearn/

[10] Mimi Dutta. Topic Modelling With LDA -A Hands-on Introduction, 2022. https://www.analyticsvidhya.com/blog/2021/07/topic-modelling-with-lda-a-hands-on-introduction/

[11] Selva Prabhakaran. Topic Modeling with Gensim (Python), 2018. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/#14computemodelperplexityandcoherencescore

7. PRIEDAS

```
# Commented out IPython magic to ensure Python compatibility.
import nltk
import pandas as pd
import re
from nltk.corpus import stopwords
nltk.download('stopwords')
nltk.download('punkt')
import string
import spacy
import numpy as np
from pprint import pprint
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
import matplotlib.pyplot as plt
# %matplotlib inline
from wordcloud import WordCloud
from sklearn.decomposition import LatentDirichletAllocation
import gensim
from gensim import corpora
from gensim.models import LdaMulticore, CoherenceModel
import matplotlib.pyplot as plt
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer

train = pd.read_csv("/content/train.csv")

"""TEKSTO APDOROJIMAS"""

#ar yra tusciu reiksmiu
train.isnull().sum()

"""1. Nereikšmingų žodžių, skyrybos ženklų šalinimas ir didžiųjų raidžių
pavertimas į mažąsias"""

#mokymo aibei
zdz = set(stopwords.words('english'))
```

```
train['clean_abstract_processed'] = train['ABSTRACT'].apply(lambda x: '
'.join([word for word in nltk.word_tokenize(re.sub(r'[^a-zA-Z\s]', '',
str(x)))
                                                                    if
word.lower() not in zdz]))
# salinam skyrybos zenklus
train['clean_abstract'] = \
train['clean_abstract_processed'].map(lambda x: re.sub('[,\.!?]', '', x))
# Konvertuojam didziasias raides i mazasias
train['clean_abstract'] = \
train['clean_abstract'].map(lambda x: x.lower())


train.head()

"""2. Pagrindinės formos"""

nlp = spacy.load('en_core_web_sm')

#mokymo aibei
lemmatized = [' '.join([token.lemma_ for token in nlp(text)]) for text in
train['clean_abstract']]

train['clean_abstract_lemmatized'] = lemmatized

train.head()

# Vaizdavimas, kurie zodziai dazniausiai naudojami tekstuose
long_string = ','.join(list(train['clean_abstract'].values))
wordcloud = WordCloud(background_color="white", max_words=5000,
contour_width=4, contour_color='steelblue')
wordcloud.generate(long_string)
wordcloud.to_image()

"""# Latent Dirichlet Allocation (LDA)

Nuorodos:
* https://www.analyticsvidhya.com/blog/2021/07/topic-modelling-with-lda-a-
hands-on-introduction/
* https://www.analyticsvidhya.com/blog/2021/06/part-3-topic-modeling-and-
latent-dirichlet-allocation-lda-using-gensim-and-sklearn/
* https://www.machinelearningplus.com/nlp/topic-modeling-gensim-
python/#14computemodelperplexityandcoherencescore

## be stopwords + lemmatized + tfidf
"""

vect = TfidfVectorizer(max_features=1000)
vect_text=vect.fit_transform(train['clean_abstract_lemmatized'])

lda_tfidf=LatentDirichletAllocation(n_components=13,learning_method='online',
random_state=42,max_iter=1)
lda_top=lda_tfidf.fit_transform(vect_text)

vocab = vect.get_feature_names_out()
for i, comp in enumerate(lda_tfidf.components_):
     vocab_comp = zip(vocab, comp)
```

```
    sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
    print("Topic "+str(i)+": ")
    for t in sorted_words:
           print(t[0],end=" ")
    print(" ")

H = lda_tfidf.components_
words = np.array(vect.get_feature_names_out())

for i, topic in enumerate(H):
  sarasas=",".join([str(x) for x in words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

from gensim.models import CoherenceModel
import gensim.corpora as corpora

def get_Cv(model, df_columnm):
  topics = model.components_

  n_top_words = 10
  texts = [[word for word in doc.split()] for doc in df_columnm]

  # create the dictionary
  dictionary = corpora.Dictionary(texts)
  # Create a gensim dictionary from the word count matrix

  # Create a gensim corpus from the word count matrix
  corpus = [dictionary.doc2bow(text) for text in texts]

  feature_names = [dictionary[i] for i in range(len(dictionary))]

  # Get the top words for each topic from the components_ attribute
  top_words = []
  for topic in topics:
      top_words.append([feature_names[i] for i in topic.argsort()[:-
n_top_words - 1:-1]])

  coherence_model = CoherenceModel(topics=top_words, texts=texts,
dictionary=dictionary, coherence='c_v')
  coherence = coherence_model.get_coherence()
  return coherence

get_Cv(lda_tfidf, train['clean_abstract_lemmatized'])

"""## be stopwords + lemmatized + CountVectorizer"""

count_vec = CountVectorizer(max_features = 1000)
count_vec_text = count_vec.fit_transform(train['clean_abstract_lemmatized'])

lda_count
=LatentDirichletAllocation(n_components=13,learning_method='online',random_st
```

```
ate=42,max_iter=1)
lda_top=lda_count.fit_transform(count_vec_text)

vocab = count_vec.get_feature_names_out()
for i, comp in enumerate(lda_count.components_):
     vocab_comp = zip(vocab, comp)
     sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
     print("Topic "+str(i)+": ")
     for t in sorted_words:
            print(t[0],end=" ")
     print("")

H = lda_count.components_
words = np.array(count_vec.get_feature_names_out())

for i, topic in enumerate(H):
  sarasas=",".join([str(x) for x in words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

get_Cv(lda_count, train['clean_abstract_lemmatized'])

"""# Coherence score for different numbers of topics"""

# Skaidom teksta i tokenus

def sent_to_words(sentences):
    for sentence in sentences:
        yield(gensim.utils.simple_preprocess(str(sentence), deacc=True))

data = train.clean_abstract_lemmatized.values.tolist()
data_words = list(sent_to_words(data))

# Create Dictionary
id2word = corpora.Dictionary(data_words)
# Create Corpus
texts = data_words
# Skaiciuojam zodziu dazni
corpus = [id2word.doc2bow(text) for text in texts]

# Calculate coherence scores for different numbers of topics
def calculate_coherence_score(corpus, id2word, texts, max_topics):
    coherence_scores = []
    for num_topics in range(2, max_topics + 1):
        model = LdaMulticore(corpus=corpus, id2word=id2word,
num_topics=num_topics)
        coherence_model = CoherenceModel(model=model, texts=texts,
dictionary=id2word, coherence='c_v')
        coherence_score = coherence_model.get_coherence()
        coherence_scores.append(coherence_score)
    return coherence_scores
```

```
# Set the maximum number of topics to explore
max_topics_to_explore = 15

# Calculate coherence scores
coherence_scores = calculate_coherence_score(corpus, id2word, texts,
max_topics_to_explore)

# Plot the coherence scores
plt.plot(range(2, max_topics_to_explore + 1), coherence_scores)
plt.xlabel("Number of Topics")
plt.ylabel("Coherence Score")
plt.title("Coherence Score for Different Numbers of Topics")
plt.show()

"""# Latent Semantic Analysis (LSA)

Nuorodos:
* https://www.datacamp.com/tutorial/discovering-hidden-topics-
python?utm_source=google&utm_medium=paid_search&utm_campaignid=19589720824&ut
m_adgroupid=157156376311&utm_device=c&utm_keyword=&utm_matchtype=&utm_network
=g&utm_adpostion=&utm_creative=676136073491&utm_targetid=dsa-
2218886984100&utm_loc_interest_ms=&utm_loc_physical_ms=9062284&utm_content=&u
tm_campaign=230119_1-sea~dsa~tofu_2-b2c_3-row-p2_4-prc_5-na_6-na_7-le_8-pdsh-
go_9-na_10-na_11-na&gad_source=1&gclid=Cj0KCQiAr8eqBhD3ARIsAIe-
buOgoxaaymygUiZrQ5x6RjCCiuVxyQoKex5-Z6kd4kM9tJd7rIXvVfEaAibZEALw_wcB
* https://machinelearninggeek.com/latent-semantic-indexing-using-scikit-
learn/

Be nereikšmingų žodžių ir su pagrindinėmis formomis + TfidfVectorizer()
"""

from sklearn.decomposition import TruncatedSVD
import matplotlib.pyplot as plt
import numpy as np

#mokymo aibei
#ivertiname zodziu svarbuma
tfidf = TfidfVectorizer()
tfidf.fit(train['clean_abstract_lemmatized'])

X = tfidf.transform(train['clean_abstract_lemmatized'])

num_topics = 13
lsa = TruncatedSVD(n_components=num_topics, random_state=42)
lsa_result = lsa.fit_transform(X)

topics = pd.DataFrame(lsa_result, columns=[f'Topic {i+1}' for i in
range(num_topics)])

vocab = tfidf.get_feature_names_out()
for i, comp in enumerate(lsa.components_):
     vocab_comp = zip(vocab, comp)
     sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
     print("Topic "+str(i)+": ")
     for t in sorted_words:
            print(t[0],end=" ")
     print(" ")
```

```
def plot_top_words_per_topic(lsa_model, feature_names, n_words=10):
    topics = []
    for topic_idx, topic in enumerate(lsa_model.components_):
        top_words_idx = topic.argsort()[:-n_words - 1:-1]
        top_words = [feature_names[i] for i in top_words_idx]
        topics.append(top_words)
        print(f"Topic #{topic_idx + 1}: {', '.join(top_words)}")

        plt.figure(figsize=(8, 6))
        plt.barh(range(len(top_words)), topic[top_words_idx], align='center',
alpha=0.7)
        plt.yticks(range(len(top_words)), top_words)
        plt.xlabel('Word Weight')
        plt.title(f'Topic #{topic_idx + 1}')
        plt.show()

    return topics

feature_names = np.array(tfidf.get_feature_names_out())

topics = plot_top_words_per_topic(lsa, feature_names)

H = lsa.components_
words = np.array(tfidf.get_feature_names_out())

for i, topic in enumerate(H):
  sarasas=",".join([str(x) for x in words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

from gensim.models import CoherenceModel
import gensim.corpora as corpora

def get_Cv(model, df_columnm):
  topics = model.components_

  n_top_words = 10
  texts = [[word for word in doc.split()] for doc in df_columnm]

  # create the dictionary
  dictionary = corpora.Dictionary(texts)
  # Create a gensim dictionary from the word count matrix

  # Create a gensim corpus from the word count matrix
  corpus = [dictionary.doc2bow(text) for text in texts]

  feature_names = [dictionary[i] for i in range(len(dictionary))]

  # Get the top words for each topic from the components_ attribute
  top_words = []
  for topic in topics:
```

```
      top_words.append([feature_names[i] for i in topic.argsort()[:-
n_top_words - 1:-1]])

  coherence_model = CoherenceModel(topics=top_words, texts=texts,
dictionary=dictionary, coherence='c_v')
  coherence = coherence_model.get_coherence()
  return coherence

get_Cv(lsa, train["clean_abstract_lemmatized"])

"""Be nereikšmingų žodžių ir su pagrindinėmis formomis + CountVectorizer()"""

vectorizer = CountVectorizer()
vectorizer.fit(train['clean_abstract_lemmatized'])

X = vectorizer.transform(train['clean_abstract_lemmatized'])

num_topics = 13
lsa = TruncatedSVD(n_components=num_topics, random_state=42)
lsa_result = lsa.fit_transform(X)

topics = pd.DataFrame(lsa_result, columns=[f'Topic {i+1}' for i in
range(num_topics)])

def plot_top_words_per_topic(lsa_model, feature_names, n_words=10):
    topics = []
    for topic_idx, topic in enumerate(lsa_model.components_):
        top_words_idx = topic.argsort()[:-n_words - 1:-1]
        top_words = [feature_names[i] for i in top_words_idx]
        topics.append(top_words)
        print(f"Topic #{topic_idx + 1}: {', '.join(top_words)}")

        plt.figure(figsize=(8, 6))
        plt.barh(range(len(top_words)), topic[top_words_idx], align='center',
alpha=0.7)
        plt.yticks(range(len(top_words)), top_words)
        plt.xlabel('Word Weight')
        plt.title(f'Topic #{topic_idx + 1}')
        plt.show()

    return topics

feature_names = np.array(tfidf.get_feature_names_out())

topics = plot_top_words_per_topic(lsa, feature_names)

vocab = vectorizer.get_feature_names_out()
for i, comp in enumerate(lsa.components_):
     vocab_comp = zip(vocab, comp)
     sorted_words = sorted(vocab_comp, key= lambda x:x[1], reverse=True)[:10]
     print("Topic "+str(i)+": ")
     for t in sorted_words:
            print(t[0],end=" ")
     print(" ")

H = lsa.components_
words = np.array(vectorizer.get_feature_names_out())
```

```
for i, topic in enumerate(H):
  sarasas=",".join([str(x) for x in words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

get_Cv(lsa,  train["clean_abstract_lemmatized"])

"""# Non-negative Matrix factorization (NMF) - pasikeiciau tema

Naudojamos nuorodos:
https://medium.com/voice-tech-podcast/topic-modelling-using-nmf-2f510d962b6e

https://www.freecodecamp.org/news/advanced-topic-modeling-how-to-use-svd-nmf-
in-
python/#:~:text=Topic%20Modeling%20Using%20Non%2DNegative%20Matrix%20Factoriz
ation%20(NMF),-
Non%2Dnegative%20Matrix&text=Non%2Dnegative%20Matrix%20Factorization%20acts,t
he%20documents%20in%20the%20corpus.

https://predictivehacks.com/topic-modelling-with-nmf-in-python/
"""

import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import NMF

"""Atliksime tyrima:


*   Be stopwords, lematizuota ir vektorizuota Tfidf
*   Be stopwords, lematizuota ir vektorizuota count vectorizer

## be stopwords, lematized + tfidf
"""

vectorizer_nmf_tfidf = TfidfVectorizer()
X_nmf_tfidf =
vectorizer_nmf_tfidf.fit_transform(train['clean_abstract_lemmatized'])

words = np.array(vectorizer_nmf_tfidf.get_feature_names_out())

nmf_tfidf = NMF(n_components=13)
W_tfidf = nmf_tfidf.fit_transform(X_nmf_tfidf)
H_tfidf = nmf_tfidf.components_

for i, topic in enumerate(H_tfidf):
     print("Topic {}: {}".format(i + 1, ",".join([str(x) for x in
words[topic.argsort()[-10:]]])))

import matplotlib.pyplot as plt
```

```
for i, topic in enumerate(H_tfidf):
  sarasas=",".join([str(x) for x in  words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

components_df = pd.DataFrame(nmf_tfidf.components_,
columns=vectorizer_nmf_tfidf.get_feature_names_out())

for topic in range(components_df.shape[0]):
    tmp = components_df.iloc[topic]
    print(f'For topic {topic+1} the words with the highest value are:')
    print(tmp.nlargest(10))
    print('\n')

"""## be stopwords + lemmatized + countvectorizer algoritmas"""

from sklearn.feature_extraction.text import CountVectorizer

vectorizer_nmf_cv = CountVectorizer()
X_nmf_cv =
vectorizer_nmf_cv.fit_transform(train['clean_abstract_lemmatized'])

words = np.array(vectorizer_nmf_cv.get_feature_names_out())

nmf_cv = NMF(n_components=13)
W_cv = nmf_cv.fit_transform(X_nmf_cv)
H_cv = nmf_cv.components_

for i, topic in enumerate(H_cv):
     print("Topic {}: {}".format(i + 1, ",".join([str(x) for x in
words[topic.argsort()[-10:]]])))

for i, topic in enumerate(H_cv):
  sarasas=",".join([str(x) for x in  words[topic.argsort()[::-1][:10]]])
  wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color='black',
relative_scaling=0).generate(sarasas)
  plt.imshow(wc)
  plt.title(f"Topic{i+1}")
  plt.axis("off")
  plt.show()

components_df_cv = pd.DataFrame(nmf_cv.components_,
columns=vectorizer_nmf_cv.get_feature_names_out())

for topic in range(components_df_cv.shape[0]):
    tmp = components_df_cv.iloc[topic]
    print(f'For topic {topic+1} the words with the highest value are:')
    print(tmp.nlargest(10))
    print('\n')

"""Ivertinimas
```

```
https://radimrehurek.com/gensim/models/coherencemodel.html

Topic coherence is a way to judge the quality of topics via a single
quantitative, scalar value. There are many ways to compute the coherence
score. For the u_mass and c_v options, a higher is always better. Note that
u_mass is between -14 and 14 and c_v is between 0 and 1.
https://datascience.oneoffcoder.com/topic-modeling-gensim.html

https://stackoverflow.com/questions/66877729/calculate-coherence-for-non-
gensim-topic-model
"""

from gensim.models import CoherenceModel
import gensim.corpora as corpora

def get_Cv(model, df_columnm):
  topics = model.components_

  n_top_words = 10
  texts = [[word for word in doc.split()] for doc in df_columnm]

  # create the dictionary
  dictionary = corpora.Dictionary(texts)
  # Create a gensim dictionary from the word count matrix

  # Create a gensim corpus from the word count matrix
  corpus = [dictionary.doc2bow(text) for text in texts]

  feature_names = [dictionary[i] for i in range(len(dictionary))]

  # Get the top words for each topic from the components_ attribute
  top_words = []
  for topic in topics:
      top_words.append([feature_names[i] for i in topic.argsort()[:-
n_top_words - 1:-1]])

  coherence_model = CoherenceModel(topics=top_words, texts=texts,
dictionary=dictionary, coherence='c_v')
  coherence = coherence_model.get_coherence()
  return coherence

get_Cv(nmf_cv, train["clean_abstract_lemmatized"])

get_Cv(nmf_tfidf,  train["clean_abstract_lemmatized"])

"""# BERTopic

Be nereikšmingų žodžių ir su pagrindinėmis formomis + TfidfVectorizer()
"""

vectorizer_model_1 = TfidfVectorizer()

!pip install bertopic

from bertopic import BERTopic
model = BERTopic(verbose=True,
                 language="english",
```

```
                nr_topics = 14,
                vectorizer_model = vectorizer_model_1)
headline_topics, _ = model.fit_transform(train['clean_abstract_lemmatized'])

freq = model.get_topic_info()
print("Number of topics: {}".format( len(freq)))
freq

model.get_topic(0)  # Select the most frequent topic

model.visualize_barchart(top_n_topics=13)

def create_wordcloud(topic_model, topic):
    text = {word: value for word, value in topic_model.get_topic(topic)}
    wc = WordCloud(width=1000, height=600, margin=3,
prefer_horizontal=0.7,scale=1,background_color="black", max_words=1000)
    wc.generate_from_frequencies(text)
    plt.imshow(wc, interpolation="bilinear")
    plt.axis("off")
    plt.title(f"Topic {topic + 1}")
    plt.show()

#create_wordcloud(model, topic=1)
for topic in range(13):
    create_wordcloud(model, topic)

vectorizer = model.vectorizer_model
analyzer = vectorizer.build_analyzer()

# Extract features for Topic Coherence evaluation
words = vectorizer.get_feature_names_out()
tokens = [analyzer(doc) for doc in train['clean_abstract_lemmatized']]
dictionary = corpora.Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]
topic_words = [[words for words, _ in model.get_topic(topic)]
               for topic in range(len(set(headline_topics))-1)]

# Evaluate
coherence_model = CoherenceModel(topics=topic_words,
                                 texts=tokens,
                                 corpus=corpus,
                                 dictionary=dictionary,
                                 coherence='c_v')
coherence = coherence_model.get_coherence()
print(coherence)

vectorizer = model.vectorizer_model
analyzer = vectorizer.build_analyzer()

# Extract features for Topic Coherence evaluation
words = vectorizer.get_feature_names_out()
tokens = [analyzer(doc) for doc in train['clean_abstract_lemmatized']]
dictionary = corpora.Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]
topic_words = [[words for words, _ in model.get_topic(topic)]
               for topic in range(len(set(headline_topics))-1)]
```

```
# Evaluate
coherence_model = CoherenceModel(topics=topic_words,
                                 texts=tokens,
                                 corpus=corpus,
                                 dictionary=dictionary,
                                 coherence='u_mass')
coherence = coherence_model.get_coherence()
print(coherence)

"""Be nereikšmingų žodžių ir su pagrindinėmis formomis + CountVectorizer()"""

vectorizer_model_2 = CountVectorizer()

model_2 = BERTopic(verbose=True,
                   language="english",
                   nr_topics = 14,
                   vectorizer_model = vectorizer_model_2)
headline_topics, _ =
model_2.fit_transform(train['clean_abstract_lemmatized'])

freq = model_2.get_topic_info()
print("Number of topics: {}".format( len(freq)))
freq

model_2.get_topic(0)  # Select the most frequent topic

model_2.visualize_barchart(top_n_topics=13)

for topic in range(13):
    create_wordcloud(model_2, topic)

vectorizer = model_2.vectorizer_model
analyzer = vectorizer.build_analyzer()

# Extract features for Topic Coherence evaluation
words = vectorizer.get_feature_names_out()
tokens = [analyzer(doc) for doc in train['clean_abstract_lemmatized']]
dictionary = corpora.Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]
topic_words = [[words for words, _ in model_2.get_topic(topic)]
               for topic in range(len(set(headline_topics))-1)]

# Evaluate
coherence_model = CoherenceModel(topics=topic_words,
                                 texts=tokens,
                                 corpus=corpus,
                                 dictionary=dictionary,
                                 coherence='c_v')
coherence = coherence_model.get_coherence()
print(coherence)

vectorizer = model_2.vectorizer_model
analyzer = vectorizer.build_analyzer()

# Extract features for Topic Coherence evaluation
words = vectorizer.get_feature_names_out()
tokens = [analyzer(doc) for doc in train['clean_abstract_lemmatized']]
```

```
dictionary = corpora.Dictionary(tokens)
corpus = [dictionary.doc2bow(token) for token in tokens]
topic_words = [[words for words, _ in model_2.get_topic(topic)]
               for topic in range(len(set(headline_topics))-1)]

# Evaluate
coherence_model = CoherenceModel(topics=topic_words,
                                 texts=tokens,
                                 corpus=corpus,
                                 dictionary=dictionary,
                                 coherence='u_mass')
coherence = coherence_model.get_coherence()
print(coherence)
```