EÖTVÖS LORÁND TUDOMÁNYEGYETEM

TERMÉSZETTUDOMÁNYI KAR

Bartalis Dávid

Szubmoduláris függvények alkalmazása redukciós feladatokra a gépi tanulásban

Témavezetők:

Bérczi-Kovács Erika

ELTE, Operációkutatási Tanszék

Béres Ferenc

ELKH SZTAKI, Informatikai Kutatólaboratórium

Budapest, 2022

Tartalomjegyzék

Köszönetnyilvánítás

Dolgozatom elején mindenképpen szeretnék köszönetet mondani témavezetőimnek: Bérczi-Kovács Erikának, aki segítségével, szakmai tudásával és lelkesítő mondataival nagy mértékben hozzájárult a szakdolgozatom elkészítéséhez; továbbá Béres Ferencnek a sok sok programozási segítségért, amit kaptam, ugyanis nélküle ez a dolgozat nem jött volna létre. Mindkét témavezetőmre igaz, hogy egész idő alatt támogattak, bármikor számíthattam rájuk, minden egyes kérdésemre készséggel és türelemmel válaszoltak. Köszönöm!

Továbbá szeretném hálámat kifejezni a családomnak és ezúton is köszönöm nekik a rengeteg támogatást. Természetesen szeretném megköszönni hallgatótársaim, barátaim segítségét, akik az alapképzés és / vagy a mesterképzés alatt mellettem álltak, támogattak.

Bevezetés

Manapság egyre gyakrabban felmerülő probléma a gépi tanulási folyamatok során, hogy rendkívül nagy mennyiségű adatot használunk különböző modellek tanítására, így ahhoz, hogy megfelelő eredményt kapjunk olykor nagy számítási teljesítmény mellett is hosszú a tanulási algoritmusok futásideje. A szubmoduláris optimalizálás egy lehetséges megoldás lehet arra, hogy a nagy adathalmazok kezelésével járó számítási terheket csökkentsük, ugyanis az adathalmazok méretének növelésével általában az ismétlődések száma is nő, viszont ezek a redundáns adatok nem mindig jelentenek hasznot. Tehát az adatokban rejlő információ szubmoduláris jellege miatt lehetnek hatékonyak a szubmoduláris maximalizálásra építő megközelítések.

A mesterképzés során témavezetőimmel az alapképzésen megkezdett munkát folytatva a fentiekben vázolt problémára a Washingtoni Egyetem kutatói által kifejlesztett Apricot [7] Python program csomag alkalmazási lehetőségeit tanulmányoztuk.

A mérések, tapasztalatok alapján azt láttuk, hogy a programot nemcsak a minták számának redukálására (sorredukció), hanem a dimenziócsökkentésre is lehet használni, így nem a sorok, hanem az oszlopok számának redukálásával csökkentenénk a tanítási adathalmaz méretét. Dolgozatomban bemutatom a szubmoduláris megközelítést a sorredukciós feladatra, majd összehasonlítom az Apricot által adott módszert a standard dimenziócsökkentési eljárásokkal. Megkísérlem kihangsúlyozni a módszer erősségeit és gyengeségeit is egyaránt.

Az első fejezetben a gépi tanulásról írunk általában, majd a második fejezetben a dimenziócsökkentési feladatot definiáljuk, a legelterjedtebb dimenziócsökkentési eljárásokat ismertetjük. Ezek után a szubmodularitással kapcsolatos alapvető fogalmakat mutatjuk be, majd a szubmoduláris dimenziócsökkentést fejtjük ki részletesebben. Az elméleti rész után rátérünk a használt adathalmazok, az elért eredmények, mérések ismertetésére. Végül egy összegzéssel zárjuk a dolgozatot.

1. Gépi tanulás

1.1. Alapfogalmak

A dolgozatban gépi tanulási (angolul Machine Learning) folyamatok felgyorsításával foglalkozunk, ezért szeretnénk néhány bekezdésben ismertetni a legfontosabb gépi tanulással kapcsolatos fogalmakat.
Attól függően, hogy hogyan tanítjuk a gépet, azaz a tanuló eljárás milyen formában szerez tudomást az adatpontokhoz tartozó célváltozó értékekről, három fő megközelítést szokás említeni [11]:

(1) Felügyelt tanulás (supervised learning): az adatpontokhoz tartozó célváltozóértékek ismertek és ezek segítségével tanítjuk a modellt. Az adatpontokból és a hozzájuk tartozó ismert célváltozóértékekből álló halmazt hívjuk tanítási adathalmaznak (training dataset). A cél, hogy ez alapján olyan modellt tanuljunk, ami korábban nem látott példákon (teszthalmazon, validációs halmazon) is helyesen működik.

A dolgozatban a következő formalizációt használjuk: $\mathcal{U} = \{(x^i, y^i)\}_{i=1\ldots m}$ legyen a teljes tanítási adathalmaz, ahol minden x^i egy d dimenziós vektor (rekord, minta), értékeit az X halmazból veszi fel és $y^i \in Y$ a hozzá tartozó címkeérték. Hasonlóan legyen V a validációs halmaz, a minták pedig $\{(x^i, y^i)\}_{i \in V}$.

(2) Felügyelet nélküli tanulás (unsupervised learning): A megfigyelésekhez nincs rendelkezésre álló célváltozóérték. Az adatok közt összefüggéseket, különböző mintázatokat keresünk. Célunk, hogy ezeket azonosítsuk. A tanítás itt a kérdéses mintázat definiálását jelenti.

(3) Megerősítéses tanulás (reinforcement learning): a gép folyamatosan akciókat hajt végre, melyekre néha kap megerősítést (mennyire jól csinálja a feladatot). Ezekből a visszajelzésekből tanul.

A mesterképzés alatt végzett munka során többnyire felügyelt gépi tanulással foglalkoztunk. Ezen belül két lényeges feladattípus különböztethető meg: osztályozási, azaz klasszifikációs és regressziós feladat. A klasszifikációs feladat esetén a

célunk, hogy az adatpontokat az előre adott osztályok közül az egyikbe besoroljuk, azaz megfelelő célváltozóértékkel, osztály címkével (class label) ellássuk. Ezzel szemben regresszió esetén a célváltozó nem diszkrét, hanem folytonos értékeket vesz fel, a feladatunk megjósolni, prediktálni, hogy a különböző adatpontokhoz milyen célváltozóérték tartozhat.
Ahhoz, hogy mérni tudjuk egy gépi tanuló eljárás jóságát szükségünk van egy kiértékelési metrikára, ami értékeli, számszerűsíti a modell teljesítményét.
Először négy olyan metrikát ismertetek, amivel az osztályozó, azaz klasszifikációs modellek teljesítményét jellemezhetjük. Ehhez tekintsük a következő tévesztési mátrixot (confusion matrix):

		Jósolt címke	
		1	0
Valódi címke	1	TP	FN
	0	FP	TN

A táblázatban a TP (true positive, magyarul valódi pozitív), a modell által helyesen prediktált 1-es címkével rendelkező minták számának felel meg, az FN (false negative, azaz hamis negatív), pedig azon 1-es célváltozóértékkel rendelkező minták számának felel meg, amelyeket a modell tévesen 0 címkéjűnek prediktál. Az FP (false positive, azaz hamis pozitív) azon 0 címkéjű minták számának felel meg, amelyeket a modell tévesen 1-esnek jósol, a TN (true negative, magyarul igaz negatív) a modell által helyesen prediktált 0 célváltozóértékkel rendelkező esetek számát jelöli.
Ezen jelölések segítségével a következőképpen definiálhatunk négy kiértékelési metrikát [11]:

(i) Accuracy (pontosság): a helyes jóslatok aránya, azaz

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN}$$

(ii) Precision (precizitás): azt fejezi ki, hogy amikor a modell 1-es osztálycímkét prediktált, akkor hány százalékban volt igaza.

$$\text{precision} = \frac{TP}{TP + FP}$$

(iii) Recall (szenzitivitás): azt mutatja meg, hogy az adathalmazban szereplő elvárt 1 címkéjű minták közül hányat talált meg/fedett le a modell.

$$\text{recall} = \frac{TP}{TP + FN}$$

(iv) Roc-Auc: A ROC görbe (Receiver Operating Characteristic curve) olyan teljesítményt jellemző grafikon, amely megadja a valódi pozitívak (azaz 1-esnek jósolt, valóban 1-es címkéjűek) arányát a hamis pozitív minták (tévesen 1-es címkéjűnek prediktáltak) arányának függvényében. Tehát a modell folytonos kimenetet eredményez, ami annál nagyobb, minél nagyobb valószínűséggel tartozik az adott minta a pozitív osztályba. Itt persze fontos, hogy milyen küszöbérték felett tekintjük a mintát pozitívnak. A valódi pozitívak arányát (TPR) definiáljuk úgy, mint a recall metrikát :

$$TPR = \frac{TP}{TP + FN},$$

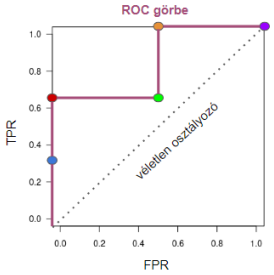
a hamis pozitívak arányát (FPR) pedig a következőképpen:

$$FPR = \frac{FP}{FP + TN}.$$

Ekkor a TPR-t az FPR függvényében ábrázolva kapott görbét nevezzük ROC görbének. A ROC-görbe alatti területet pedig AUC-vel (area under curve) jelöljük. Hogyha tökéletes osztályozó modellel van dolgunk és található olyan küszöbszám, amely mellett a modell kimenete megegyezik a tényleges osztályokkal, az AUC értéke 1. A véletlenszerű kimenetet adó modell esetében az AUC értéke 0,5.

p0	p1	y
0.950	0.050	1
0.952	0.048	1
0.97	0.03	0
0.99	0.01	1
0.99	0.01	0

FPR	TPR
0	1/3
0	2/3
1/2	2/3
1/2	1
1	1



1. ábra. ROC görbe. A bal oldali táblázatban az osztályokba tartozás valószínűségeit találjuk. A kérdés az, hogy milyen küszöb valószínűség mellett döntünk a két osztály között. A ROC görbe a hamis pozitívak és a valódi pozitvak arányát ábrázolja.

A regressziós feladatokban gyakran használt kiértékelő függvény az átlagos négyzetes eltérés (mean squared error - mse). Legyenek az y_i-k a valódi célváltozóértékek, $\bar{y_i}$-k pedig a jósolt értékek ($i = 1...n$). Ekkor

$$MSE = \frac{\sum_{i=1}^{n}(y_i - \bar{y_i})^2}{n}$$

A következőkben néhány gyakran használt klasszifikációs módszert említünk. Főleg azokra térünk ki, melyeket mi is használtunk a mérések során. Ezeket kis módosítással regressziós feladatok megoldására is lehet használni.

1.2. Osztályozó eljárások

1.2.1. Logisztikus regresszió

A logisztikus regresszió módszer [11] mögött a gondolat az, hogy az osztály előfordulásának valószínűségét jósoljuk meg adott bemenet esetén. Ehhez az úgynevezett logisztikus függvényt használjuk, ami a következő:

$$f(\bar{y}) = \frac{1}{1 + e^{-\bar{y}}}$$

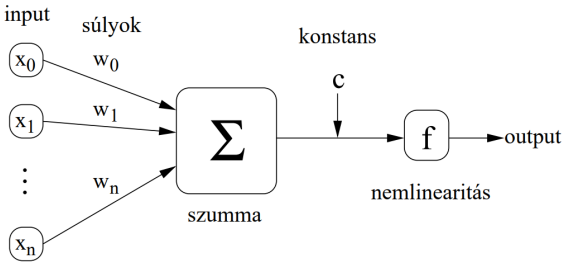
.

Szemben a lineáris regresszióval, nem az X és az Y között van lineáris kapcsolat, hanem $\ln\left(\frac{P(Y=1|X)}{1-P(Y=1|X)}\right)$ és $x^T w$ (az x pont jóslásához meghatározandó érték) között. Tehát

$$P(Y=1|X) = \frac{1}{1+e^{-X^T w}},$$

$$P(Y=0|X) = 1 - P(Y=1|X) = \frac{e^{-X^T w}}{1+e^{-X^T w}}.$$

Az 2. ábra a logisztikus regresszió módszer lényegét mutatja.



2. ábra. Logisztikus regresszió [11]

Azt a $\bar{w}$ értéket kell megkeresni, amelyik a legkisebb hibát adja, viszont erre nincs szép zárt képlet, hanem gradiensképzésen alapuló iteratív, közelítő módszer használható.

1.2.2. Hibrid módszerek

Random Forest

A random forest [11] (vagy magyarul véletlen erdő) egy gépi tanulásban és adatbányászatban használt klasszifikációs és regressziós módszer, ami döntési fákat használ. A döntési fák alapötlete, hogy a bonyolult döntéseket egyszerű döntések sorozatára vezeti vissza. Egy ismeretlen minta osztályozásakor a gyökérből kiindulva a belső csúcsokban, csomópontokban feltett kérdésekre adott válasznak megfelelően lépkedünk lefelé a fában, mígnem eljutunk egy levélig. A döntést, azaz a klasszifikálást az adott levél címkéje segítségével hozzuk meg.

A döntési fát a tanítási halmaz segítségével rekurzív módon állítjuk elő: kiindulunk a tanítási adatbázisból és egy olyan kérést keresünk, aminek segítségével jól

szétvágható a tanítási adathalmaz. Egy szétvágásra akkor mondjuk, hogy jó, ha a célváltozóértékek eloszlása a keletkezett részekben kevésbé szórt. Az nem feltétlenül szükséges, hogy az egyes részfák mérete nagyjából megegyezzen, de néhány algoritmus erre is figyel. Ezt az eljárást rekurzívan alkalmazzuk.

A random forest módszer lényege, hogy több különböző döntési fát hoz létre, és ezek eredményeinek átlagolásával (vagy klasszifikáció esetén a legtöbb szavazatot kapott eredmény kiválasztásával, azaz többségi szavazással) adja meg a végeredményt. Ezzel a megközelítéssel kisebb eséllyel esünk a túltanulás hibájába.

Ahhoz, hogy a módszer jól működjön szükséges, hogy az egyes döntési fák a véletlennél jobban teljesítsenek, illetve hogy az előrejelzéseik (és így a hibáik) egymástól viszonylag függetlenek legyenek, azaz kicsi legyen köztük a korreláció. A több döntési fa együttes alkalmazása így ki tudja szűrni az egyes döntési fák által vétett hibákat.

Boosting

A boosting [11] eljárások során klasszifikáló modellek sorozatát hozzuk létre. Ezek az osztályozó modellek általában ugyanolyan típusúak (például mindegyik egy döntési fa), továbbá gyakran a modellt előállító algoritmus paramétereit sem változtatjuk. A modellek közti különbséget az adja, hogy a tanítási halmazban szereplő mintákat különböző súlyokkal vesszük figyelembe az egyes modellek létrehozása során: Az első modell létrehozásakor minden mintát azonos (például 1) súllyal veszünk figyelembe, majd ezt követően megnézzük, hogy a tanítási adatpontok közül kiket klasszifikál helyesen a modell és kiket nem. Azon adatpontok súlyát csökkentjük, akiknek eltaláltuk a célváltozóértékét, akikét nem, azoknak a súlyát növeljük. Ezeket az új súlyokat figyelembe véve egy újabb klasszifikációs modellt építünk. Ezt az eljárást folytatjuk egészen addig, amíg a kívánt számú osztályozó modellt létre nem hoztuk.

Egy ismeretlen osztályba tartozó, új minta osztályozása során a felépített klasszifikátorok súlyozott többségi szavazatát vesszük. (Regresszió esetén a modellek kimeneteinek súlyozott átlagát tekintjük.)

A boosting eljárásnak több változata létezik, melyek abban különböznek, hogy a modellek létrehozása során konkrétan hogyan változtatjuk az egyes mintákhoz

tartozó súlyértékeket, illetve abban, hogy az ismeretlen minta klasszifikálásakor milyen módon súlyozzuk az osztályozó modelleket. Néhány ismert változat: Adaboost, Gradient boosting, XGBoost.

1.2.3. Naiv Bayes

A Naiv Bayes [11] nevét onnan kapta, hogy feltételezi, hogy adott célváltozó mellett az attribútumok, minták függetlenek egymástól. A módszer során az X-et (a minták lehetséges értékeit) és az Y-t (lehetséges célváltozóértékeket) véletlen változókként kezelhetjük és kapcsolatukat $P(Y|X)$ segítségével írhatjuk le. Ezt a feltételes valószínűséget Y posteriori valószínűségének is nevezik, a $P(Y)$ értéket pedig priori valószínűségnek.
A tanítási folyamat során a $P(Y|X)$ a posteriori valószínűségeket meg kell tanulni az X és Y minden lehetséges kombinációjára. Ezek után egy X^* tesztrekord osztályzása úgy történik, hogy megkeressük azt az Y^* osztályt, ami a $P(Y^*|X^*)$ posteriori valószínűséget maximalizálja.
A feltételezett függetlenség miatt az adott osztályra vonatkozó feltételes valószínűség X minden kombinációjához történő kiszámítása helyett elég az egyes X_i -k feltételes valószínűségét megbecsülni adott Y mellett. Ez a megközelítés gyakorlati szempontból sokkal alkalmasabb, ugyanis ahhoz, hogy jó becslést kapjunk a valószínűségre, nincs szükség nagyon nagy méretű tanítási halmazra. Egy tesztrekord osztályozásához a naiv Bayes-klasszifikátor a következő valószínűségeket számolja és ezeknek keresi a maximumát:

$$P(Y|X) = P(Y)\prod_{i=1}^{d} P(X_i|Y).$$

2. Dimenziócsökkentés

Mint a bevezőben is említettük gyakran fordul elő, hogy az adatsorok kezelhetetlenné, használhatatlanná válnak, mivel túl sok dimenziósak, ahol a dimenzió alatt az egy mintához tartozó paraméter, feature számot értjük. Matematikai szempontból tekintve, a dimenziószám csökkentésének a célja az, hogy a vizsgált sokdimenziós adathalmaz (D dimenziós) egy alacsony dimenzionalitású (d-dimenziós) reprezentációját készísük el, amely leginkább megőrzi, tükrözi az adathalmazban rejlő fontos információkat, és magát a struktúrát is. A következőkben az adatokra számsorokként tekintünk, ezért például szövegek esetén különböző vektorbeágyazós módszerekkel át kell alakítani az adatot. A feladatot a következőképpen lehet formalizálni. Adott az M n sorból és D oszlopból álló mátrix, mely az adathalmazunkat tartalmazza. Tekintsük a mátrix sorait, azaz az adathalmaz sorait (rekordjait) tartalmazó halmazt. Legyen ez az $X = \{x_1, x_2, ..., x_n\}$ objektumhalmaz, ahol $x_i = [x_{i1}, x_{i2}, ..., x_{iD}]$ az i. objektum (rekord) melyet D tulajdonság jellemez. A dimenziócsökkentő eljárások a vizsgált X objektumhalmazt egy új, d ($d << D$) dimenziós Y objektumhalmazba képezik le, ahol $Y = y_1, y_2, ..., y_n$, $y_i = [y_i1, y_i2, ..., y_id]$. A dimenziószám csökkentéséhez használhatunk lineáris algebrai vagy statisztikai eszközöket is. A következőkben néhány standard megközelítést szeretnék ismertetni.

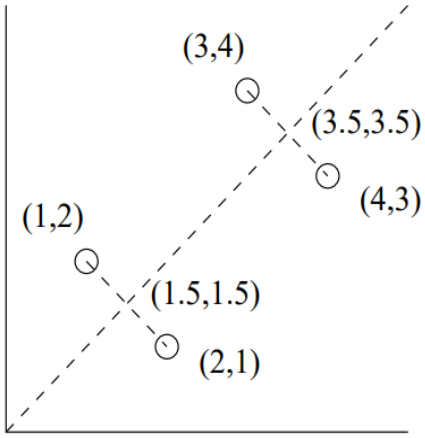
2.1. PCA - főkomponens analízis

A főkomponens-analízis [4] (angolul Principal Component Analysis, rövidítve PCA) egy többváltozós statisztikai, adatbányászati, elsősorban adatredukciós feladatokra használt eljárás. A módszert szokták Hotelling, vagy Karhunen-Loève transzformációnak is nevezni. Ez az eljárás a dimenziócsökkentést az objektumtér transzformálásával oldja meg, azaz a meglévő tulajdonságokból kiindulva új, de kevesebb számú tulajdonságot hoz létre. A cél lényegében az, hogy egy nagy dimenziós adatot levetítünk a legfontosabb tengelyeire, így egy jobban, egyszerűbben kezelhető, kisebb dimenziós adatot kapunk. A legfontosabb tengelyek a kovarianciamátrix nagy sajátértékeihez tartozó tengelyek lesznek, ugyanis ezen helyeken a legnagyobb

a szórásnégyzet. Ezeket hívjuk főkomponenseknek.
Másképpen fogalmazva az adatpontokat egy ortogonális lineáris transzformációval egy új koordinátarendszerbe transzformáljuk, ahol az első tengely a legnagyobb sajátértékhez tartozó, a második a második legnagyobb sajátértékhez tartozó és így tovább. Azaz elérjük azt, hogy az a komponens, amelynek legnagyobb a varianciája az első koordinátatengely mentén helyezkedjen el, a második legnagyobb varianciájú komponens a második koordináta mentén helyezkedjen el, stb. [**?**]
Az algoritmus először kiszámolja az $M^T M$ kovarianciamátrixot, majd ezen mátrix sajátvektorainak és sajátértékeinek számítása, a főkomponensek meghatározása történik. A kiszámolt sajátvektorokat a sajátértékek szerint csökkenő sorrendbe rendezzük, és ezekre, mint egy mátrixra tekintünk. Így a főkomponenseket a szignifikanciájuk, jelentőségük sorrendjében találjuk meg, a d legfontosabbat választjuk ki. Így kapunk egy feature vektort, vagyis egy ortogonális bázist. Végül az eredeti M adatpontokat tartalmazó halmaz transzponáltját beszorozzuk a feature vektor transzponáltjával és így megkapjuk a dimenziócsökkentett adathalmazt.
Könnyen látható, hogy az algoritmus legtöbb számítási kapacitást igénylő része a kovarianciamátrix kiszámítása és ennek a sajátértékfelbontása. A kovarianciamátrix $\mathcal{O}(nD\min(n,D))$ időben megkonstruálható (mátrixszorzás), a sajátértékfelbontás pedig legrosszabb esetben is $\mathcal{O}(D^3)$ időt vesz igénybe. [4] Mindezt összevetve az algoritmus futási ideje $\mathcal{O}(nD\min(n,D) + D^3)$.

2.1.1. Példa. *Egy egyszerű kétdimenziós példa a következő: adottak az (1,2); (2,1); (3,4); (4,3) pontok. Természetesen jelen példában az adatpontok száma és a dimenziószám sem indokolja a PCA használatát. Tekintsük az adatpontokat tartalmazó mátrixot:* $M = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 3 & 4 \\ 4 & 3 \end{bmatrix}$ *Ekkor az* $M^T M$ *mátrix a következő* $\begin{bmatrix} 30 & 28 \\ 28 & 30 \end{bmatrix}$. *A karakterisztikus egyenletet megoldva kapjuk a két sajátértéket:* $\lambda_1 = 58$, $\lambda_2 = 2$. *Ha kiszámoljuk a sajátértékekhez tartozó sajátvektorokat, majd ezekre, mint*

oszlopvektorokra egy mátrixként tekintünk kapjuk, hogy $E = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$. *Erre a mátrixra tekinthetünk úgy is, mint a 45 fokos forgatás mátrixa. Ha vesszük az* ME *szorzatmátrixot, látjuk, hogy például az (1,2) pontot a* $(\frac{3}{\sqrt{2}}, \frac{1}{\sqrt{2}})$ *pontba transzformáltuk.*



3. ábra. Adatpontok. A szaggatott vonal az új x-tengelyt jelöli. [4]

Az ábrán a szaggatott vonal jelöli az új x tengelyt. Látszódik, hogy az (1,2) pont új x tengelyre vett vetülete az origótól pont $\frac{3}{\sqrt{2}}$ *távolságra van. Az új y tengely természetesen merőleges a szaggatott vonalra. Vegyük észre, hogy az (1,2) pont és a szaggatott vonalra vett vetülete közti távolság pont* $\frac{1}{\sqrt{2}}$. *Valóban sikerült csökkenteni a dimenziót, sőt jelen esetben a pontok számát is.*

2.2. Szöveges adat dimenziócsökkentése

Szöveges adatok dimenziócsökkentésénél két fontos lépést kell elkülöníteni. Ahogy azt már fentebb is említettem, a szöveges adat esetén szükséges az adat számsorrá alakítása. Ehhez használhatjuk például a TF-IDF módszert. [5] A második lépés a dimenziócsökkentés, ami lényegében a gyakoriság alapján, a k leggyakrabban előforduló szó használatával történik.

A "term frequency-inverse document frequency" (kifejezés gyakoriság – fordított dokumentum gyakoriság) statisztikai mérőszámra általában a TF-IDF rövidítéssel hivatkoznak a szakirodalomban. Főleg a szövegbányászati eljárások, azaz a mesterséges nyelvfeldolgozási feladatok, a tömörítési, lényegkiemelési, avagy dokumentumkeresési eljárások során használják. A módszerrel azt próbáljuk meghatározni, hogy egy adott szó mennyire fontos, miközben próbáljuk kiszűrni a lényegtelen, ámbár sokszor előforduló szavakat, mint például a névelőket, névmásokat stb...
Tegyük fel, hogy a "word" szó TF-IDF értékét szeretnénk kiszámolni a k-adik dokumentumban. Legyen az $I = \{1, \ldots, m\}$ a dokumentumokhoz tartozó indexhalmaz, D a dokumentumok halmaza és jelölje az i-edik dokumentumot D_i, ahol $i \in I$. A TF-IDF érték kiszámítása két metrika segítségével történik:
TF: Először meg kell határozni egy adott szó gyakoriságát az adott dokumentumban ("kifejezés gyakoriság"). Ennek kiszámolására több lehetséges eljárás is létezik. Egy egyszerű leszámlálással meg lehet határozni egy adott szó gyakoriságát, ezt akár arányosítani is lehet a szöveg hosszához, vagy a leggyakrabban előforduló szó gyakoriságához. A relatív gyakoriságot jelölje f. Ehhez 1-et hozzáadva és logaritmust véve kapjuk a TF értéket:

$$TF(word, D_k) = \log(1 + f(word, D_k))$$

IDF: Másodszor meg kell határozni azt, hogy egy adott szó milyen gyakori a dokumentum-halmazunkban ("fordított dokumentum gyakoriság"). Ha ez az érték közel van a 0-hoz, az azt jelenti, hogy az adott szó nagyon gyakori, ezért érdektelenné válik. Egyébként pedig közel lesz 1-hez. Kiszámolása a következőképpen történik:

$$IDF(word, D) = \log\Big(\frac{\{|D_i| : i \in I\}}{\{|D_i| : "word" \in D_i\}}\Big)$$

Tehát a kiszámításához vesszük az összes dokumentum számának és az adott szót tartalmazó dokumentumok számának a hányadosának a logaritmusát.
Ha ezt a két értéket összeszorozzuk, akkor megkapjuk a TF-IDF értéket:

$$TFIDF(word, D_k, D) = TF(word, D_k) \cdot IDF(word, D)$$

Minél nagyobb ez a szám, annál relevánsabb az adott szó a dokumentumban.
A TF-IDF értékek segítségével meg tudunk adni egy egyszerű dimenziócsökkentő algoritmust. Tegyük fel, hogy a szövegnek már létrehoztuk a one hot reprezentációját (one hot encoding: a "word" elemet az a vektor reprezentálja, aminek csak az az oszlopa 1, ami a "word" szónak felel meg). Vegyük az oszlopokhoz tartozó TF-IDF értékeket és tartsuk meg a legnagyobb TF-IDF értékkel rendelkezőket.
Az algoritmus futási ideje lineáris.

2.3. További statisztikai módszerek

A már említett módszerek mellett különböző statisztikai megközelítéseket is lehet alkalmazni a dimenzióredukciós, feature szelekciós feladatokhoz. Itt néhány olyan példát szeretnék megemlíteni, ami a scikit-learn (sklearn) Python ML (Machine Learning) csomagban implementálva van, könnyen elérhető.
Az egyik lehetséges megközelítés az, hogy töröljük azokat az oszlopokat, amiknek a variancia értéke egy bizonyos küszöbérték alatt van. Ezen módszer angol elnevezése "Variance Threshold" [14].
Egy másik lehetséges megoldás az egyváltozós (Univariate) feature selection [14], ami valamilyen egyváltozós statisztikai próbát használ (például χ^2 próba). A próba elvégzése után kiválaszthatjuk azt a k darabot, ami a legjobb eredményt adja, vagy hasonlóan az előző eljáráshoz kiválaszthatjuk azokat, amik egy adott korlát fölött vannak.
Gyakran használható megoldás, hogy egy modell tanítása során figyeljük, hogy melyik feature mennyire járult hozzá a folyamathoz, mennyire volt fontos. Ezek az úgynevezett "feature importance" értékek [14]. A használt modell lehet például a logisztikus regresszió (1.2.1) vagy akár valamilyen döntési fa alapú modell, mint például a random forest (véletlen erdő, 1.2.2).
A következő módszert, a szekvenciális feature kiválasztást [13] (Sequential Feature Selection - SFS) kétféleképpen is meg lehet valósítani. Az egyik ötlet, hogy kiindulunk az üres halmazból, majd egy mohó algoritmussal mindig hozzávesszük a már kiválasztott oszlopokhoz azt, ami jelenleg a legnagyobb haszonnal jár (ezt egy cross validációs módszerrel számoljuk ki). Ez hasonlít a legjobban a szub-

moduláris megközelítéshez. A másik ötlet lényegében ennek a megfordítása: az összes oszloppal kezdünk és mohó módon mindig elhagyjuk azt, ami a leglényegtelenebb. A két módszer nem feltétlenül hozza ugyanazt az eredményt. Sőt az is előfordulhat, hogy az egyik kifejezetten gyors, míg a másik nagyon lassan fut le.

3. Szubmodularitás

3.1. Fogalmak

3.1.1. Definíció (Szubmoduláris függvények [1])**.** *Az U alaphalmaz részhalmazain értelmezett $\mathcal{F} : 2^U \to \mathbb{R}$ halmazfüggvényt szubmodulárisnak, vagy teljesen szubmodulárisnak nevezzük, ha $\forall X, Y \subseteq U$ halmazpárra teljesül a következő egyenlőtlenség:*

$$\mathcal{F}(X) + \mathcal{F}(Y) \geq \mathcal{F}(X \cap Y) + \mathcal{F}(X \cup Y)$$

A szubmodularitás egy ekvivalens megfogalmazása található a következő állításban, ami szemlélteti, miért is szokás a szumodularitást a csökkenő hasznok elveként is emlegetni. [1]:

3.1.2. Állítás. *Az U alaphalmaz részhalmazain értelmezett $\mathcal{F} : 2^U \to \mathbb{R}$ halmazfüggvény pontosan akkor szubmoduláris, ha $\forall B \subseteq A \subseteq U$ halmazokra és $x \in \overline{\mathrm{A}}$ esetén igaz, hogy*

$$\mathcal{F}(A \cup x) - \mathcal{F}(A) \leq \mathcal{F}(B \cup x) - \mathcal{F}(B)$$

Bizonyítás. $\Rightarrow$
Legyen $B \subseteq A \subseteq U$ és $x \in \overline{\mathrm{A}}$. Legyen $X = B \cup x$ és $Y = A$. Ekkor az $X \cap Y = B$, az $X \cup Y = A \cup x$. Az X, Y halmazokra az 1.1.1. definíciót használva, majd behelyettesítve kapjuk az 1.1.2. állításban szereplő egyenlőtlenséget.
$\Leftarrow$
Legyen $X = B \cup x$ és $Y = A$. Ekkor az X és Y halmazok uniója az $A \cup x$, a metszete pedig B. Egyszerű átalakítással kapjuk:

$$\mathcal{F}(A \cup x) + \mathcal{F}(B) \leq \mathcal{F}(B \cup x) + \mathcal{F}(A) \quad \forall B \subseteq A \subseteq U \quad x \in \overline{\mathrm{A}}$$

□

A következő két példa jól tükrözi a szubmodularitás lényegét.

3.1.3. Példa. *Legyen S színes golyók egy halmaza, $\mathcal{F}(S)$ pedig jelölje azt, hogy az S halmazban hány különböző szín van. Könnyen látszik, hogy ha $S_1 \subseteq S_2$, akkor az*

$x \notin S_1$ esetén az x S_1-hez hozzáadott értéke 1, az S_2-höz hozzáadott értéke pedig legfeljebb 1. Vagyis az $\mathcal{F}$ halmazfüggvény szubmoduláris.

3.1.4. Példa. *A maximális lefedettségi probléma (max-k-cover) az alaphalmaz legfeljebb k darab halmazának kiválasztása úgy, hogy az uniójuk a lehető legnagyobb legyen. Ezt úgy is megfogalmazhatjuk, hogy egy hipergráf élei közül szeretnénk kiválasztani legfeljebb k darabot úgy, hogy minél több pontot lefedjek. Itt is teljesül a csökkenő hasznok elve ugyanis egy új él kiválasztása egy S_2 élhalmazhoz legfeljebb annyi még fedetlen pont lefedésével jár járhat, mint ha az $S_1 \subseteq S_2$ élhalmazhoz veszem hozzá.*

A következő fejezetben további szubmoduláris függvényeket ismertetünk.
A dolgozat további részében gyakran monoton növő szubmoduláris függvényekről lesz szó, ugyanis egyes maximalizáló algoritmusok használni fogják a monotonitást.
Megjegyzés: A véges-értékű, monoton növő, szubmoduláris függvényeket polimatroid függvényeknek is szokás nevezni.

3.1.5. Definíció (Szupermoduláris függvények [2])**.** *Egy $\mathcal{F} : 2^U \to \mathbb{R}$ U alaphalmaz részhalmazain értelmezett halmazfüggvényt szupermodulárisnak, vagy teljesen szupermodulárisnak nevezünk, ha $-\mathcal{F}$ szubmoduláris.*

3.1.6. Definíció (Moduláris függvények [2])**.** *Az olyan szubmoduláris $\mathcal{F}$ függvényeket, melyek az 1.1.1.-es definícióban szereplő egyenlőtlenséget egyenlőséggel teljesítik, moduláris függvényeknek nevezzük.*

Más szóval ha egy függvény szubmoduláris és szupermoduláris is, akkor moduláris. Az olyan moduláris függvények, melyekre igaz, hogy $\mathcal{F}(\emptyset) = 0$, felírhatóak olyan alakban, hogy

$$\mathcal{F}(X) = \sum_{x \in X} \mathcal{F}(x) \quad \forall X \subseteq U,$$

ami azt jelenti, hogy $\mathcal{F}$-et az egyelemű halmazokon vett értékei meghatározzák.

3.2. Tulajdonságok

Az előzőekben definiált függvények néhány fontos tulajdonságát szeretnénk megemlíteni. [2] A bizonyítások nagy részét korábbi szakdolgozatunkban [15] ismer-

tettük, ezért most bizonyítás nélkül mondjuk ki ezen állításokat.

3.2.1. Állítás. *Szubmoduláris függvények nemnegatív lineáris kombinációja is szubmoduláris, azaz ha $\mathcal{F}$, $\mathcal{G}$ szubmodulárisak és $\alpha, \beta \geq 0$, akkor a $\mathcal{H}(S) := \alpha\mathcal{F}(S) + \beta\mathcal{G}(S)$ függvény is szubmoduláris.*

3.2.2. Lemma. *Ha $\mathcal{F}$ monoton növő szubmoduláris függvény, akkor $\forall X, Y \subseteq U$ esetén $\mathcal{F}(X) \leq \mathcal{F}(Y) + \sum_{k \in X \setminus Y} \rho_k(Y)$, ahol $\rho_k(Y) = \mathcal{F}(Y \cup k) - \mathcal{F}(Y)$.*

A következő konstrukcióval egy tetszőleges függvényt monoton függvénnyé alakíthatunk.

3.2.3. Állítás. *Legyen $\mathcal{F}$ az U alaphalmaz részhalmazain definiált tetszőleges halmazfüggvény. Legyen*

$$\mathcal{F}_{mon}(X) := \min_{Y \subseteq X} \mathcal{F}(Y)$$

Ekkor $\mathcal{F}_{mon}$ monoton csökkenő, továbbá ha $\mathcal{F}$ szubmoduláris, akkor $\mathcal{F}_{mon}$ is szubmoduláris.

Megjegyzés: Ha az $\mathcal{F}$ és $\mathcal{G}$ szubmoduláris függvények olyanok, hogy $\mathcal{F} - \mathcal{G}$ monoton, akkor $\min\{\mathcal{F}, \mathcal{G}\}$ is szubmoduláris.
A konkáv függvények és a szubmodularitás kapcsolatát mutatja az alábbi állítás. [3]

3.2.4. Tétel. *Legyen $\mathcal{F} : 2^U \to \mathbb{R}$ monoton növő szubmoduláris halmazfüggvény, $f : \mathbb{R} \to \mathbb{R}$ monoton növő konkáv függvény. Ekkor a két függvény kompozíciója, azaz $\mathcal{F}' = f \circ \mathcal{F} : 2^U \to \mathbb{R}$ monoton növő és szubmoduláris.*

4. Szubmoduláris maximalizálás

A sorredukciós és dimenziócsökkentési eljárásokat tekintve nem a szubmoduláris megközelítés a legelterjedtebb, viszont már született néhány cikk, amiben használják a szubmodularitást a sorok, illetve az oszlopok redukálására. Az általános módszer mohó megközelítést használ, a már említett szekvenciális feature kiválasztáshoz (2.3) hasonlít. A szubmoduláris függvények k korlátos (k az input része) maximalizálási feladata a következő:

$$\max\{\mathcal{F}(S) : S \subseteq U, |S| \leq k\},$$

ahol az U az alaphalmaz (jelen esetben az oszlopokat, vagy sorokat tartalmazza), az $\mathcal{F} : 2^U \to \mathbb{R}$ egy szubmoduláris halmazfüggvény és a $k \in \mathbb{R}$ a felhasználótól függő paraméter. A továbbiakban feltesszük, hogy adott egy orákulum, ami minden $S \subseteq U$ és bármely $\mathcal{F}$ szubmoduláris függvény esetén konstans időben megadja az $\mathcal{F}(S)$ értéket .
A következőkben ismertetett megközelítések közül néhány a sorok redukálására van kimondva, viszont ezt egy egyszerű trükkel, az adathalmaz transzponálásával egyből tudjuk dimenziócsökkentési feladat megoldására is használni.

4.0.1. Állítás. *A szubmoduláris függvények k-korlátos maximalizálása NP-nehéz feladat. [10]*

Bizonyítás. Mivel az imént definiált feladatra egyszerűen visszavezethető a 3.1.4-es példában ismertetett *max-k*-fedés probléma, ami pedig közismerten egy NP-nehéz feladat, következik, hogy a szubmoduláris függvények maximalizálása is NP-nehéz. □

Megjegyzés: Szubmoduláris függvények minimalizálása megoldható polinom időben. [6]
Monoton növő szubmoduláris függvény esetén továbbra is NP-nehéz a maximalizálási feladat, viszont könnyen adható egy $(1 - \frac{1}{e}) \sim 0,63$ approximációs faktorral rendelkező közelítő algoritmus. (lásd: 4.2 fejezet)

4.1. Függvények

Néhány olyan szubmoduláris függvényt fogunk ismertetni, ami hasznosnak bizonyult a gépi tanulási folyamatok, ezen belül a sorredukciós, dimenziócsökkentési feladatok során.

4.1.1. Entrópia

Legyen n darab diszkrét valószínűségi változónk, melyeket jelöljünk X_i-vel, továbbá legyen adott a $V = \{1, 2, \ldots, n\}$ indexhalmaz. Először definiáljuk az entrópiát egy valószínűségi változóra, majd 2, illetve n db diszkrét valószínűségi változó közös entrópiáját fogalmazzuk meg.

4.1.1. Definíció (Entrópia)**.** *Az X diszkrét valószínűségi változó entrópiája:*

$$H(X) = \sum_{x \in X} p(x) \log \frac{1}{p(x)}$$

4.1.2. Definíció (Közös entrópia 2 valószínűségi változóra)**.** *Az X_1 és X_2 diszkrét valószínűségi változók közös entrópiája:*

$$H(X_1, X_2) = \sum_{x_1 \in X_1, x_2 \in X_2} p(x_1, x_2) \log \frac{1}{p(x_1, x_2)}$$

4.1.3. Definíció (Közös entrópia n valószínűségi változóra)**.** *Legyen n darab diszkrét valószínűségi változónk: $X_1, X_2, ...X_n$. Ezek közös entrópiája:*

$$H(X_1, ..., X_n) = \sum_{x_1 \in X_1, ...x_n \in X_n} p(x_1, ..., x_n) \log \frac{1}{p(x_1, ..., x_n)}$$

Az entrópia és a következőkben definiált közös információ rendkívül szoros kapcsolatban áll egymással.

4.1.4. Definíció (Közös információ)**.** *Legyen az A, B két diszkrét valószínűségi változó. Az A és B közös információja legyen definíció szerint a következő:*

$$I(A, B) = \sum_{a \in A} \sum_{b \in B} p(a, b) \log \frac{p(a, b)}{p(a)p(b)}$$

ahol $p(a,b)$ az A és B közös eloszlásfüggvénye, $p(a)$, $p(b)$ pedig a megfelelő peremeloszlásfüggvény.

4.1.5. Definíció (Feltételes közös információ)**.** *Legyen az A, B és C három diszkrét valószínűségi változó úgy, hogy közülük bármelyik kettő diszjunkt. Az A és B halmaz C-re vett feltételes közös információja legyen definíció szerint a következő:*

$$I(A,B|C) = \sum_{a\in A, b\in B, c\in C} p(a,b,c)\log\frac{p(a,b,c)p(c)}{p(a,c)p(b,c)}$$

ahol $p(x,y)$ a megfelelő valószínűségi változók közös eloszlásfüggvénye, $p(x)$ pedig a megfelelő peremeloszlásfüggvény.

A feltételes közös információ egy fontos tulajdonsága az, hogy nemnegatív, azaz $I(A,B|C) \geq 0 \quad \forall A,B,C$ esetén.

4.1.6. Állítás. *Definiáljuk az entrópiát egy halmazfüggvényként, azaz legyen az $\mathcal{F}_{ent} : 2^U \to \mathbb{R}$ függvény $\forall S \subseteq U$ halmazra a következő:*

$$\mathcal{F}_{ent}(S) = H(\{X_s | s \in S\}) = -\sum p(X_s)\log p(X_s).$$

Ekkor az $\mathcal{F}_{ent}$ függvény szubmoduláris.

Bizonyítás. [12]
Használjuk a 4.1.4 definíciót. A feltételes közös információ az entrópia segítségével könnyen kifejezhető:

$$I(A,B|C) = H(A,C) + H(B,C) - H(A,B,C) - H(C).$$

Legyen $X = A \cup B$, $Y = B \cup C$. Használjuk a feltételes közös információ nemnegativitását. Behelyettesítés után kapjuk, hogy

$$\mathcal{F}(X) + \mathcal{F}(Y) + \mathcal{F}(X \cup Y) + \mathcal{F}(X \cap Y) \geq 0,$$

ami pont az $\mathcal{F}$ halmazfüggvény szubmodularitását jelenti.

□

4.1.2. Szolgáltató elhelyezési függvény

A klasszikus szolgáltató elhelyezési feladatban a célunk az, hogy a megnyitható üzleteknek egy olyan részhalmazát adjuk meg, hogy a potenciális fogyasztóktól a legközelebbi megnyitott üzlethez mért távolságok, illetve az üzletek megnyitási költségének az összege a lehető legkisebb legyen.
Ennek egy általánosítása a következő függvény:

4.1.7. Definíció (Általánosított szolgáltató elhelyezési függvény [7])**.** *Legyen a* $\Phi_d : 2^U \to \mathbb{R}_0^+$ *a két minta közötti hasonlóságot mérő függvény, ahol* U *a minták (például mondatok) halmaza. Ekkor (általánosított) szolgáltató elhelyezési függvénynek nevezzük a következő függvényt:*

$$\mathcal{F}_{szolg.}(A) = \sum_{u \in U} \max_{a \in A} \Phi(a, u), \quad A \subseteq U$$

Az általánosított szolgáltató elhelyezési függvények a nyelvfeldolgozási és a gépi tanulási feladatokban is gyakran megjelennek. A cél az, hogy a maximalizálási folyamat elvégzése után a legjobban reprezentáljuk az adatteret.

4.1.8. Állítás. *A szolgáltató elhelyezési függvények szubmodulárisak.*

Bizonyítás. Teljesül a csökkenő hasznok elve, mivel $A_2 \subseteq A_1 \subseteq V$ és $i \in V \setminus A_1$ esetén az i elem A_1-hez hozzáadott értéke nem nagyobb, mint a i-nek az A_2-höz adott értéke, mivel $\max_{a \in A_2} \Phi(a, i) \leq \max_{a \in A_1} \Phi(a, i)$, tehát az $A_1 \setminus A_2$-ben lehet olyan elem, ami a hasonlósági függvény szerint nagyon hasonló az i-hez. Ebben a speciális helyzetben ez azt jelenti, hogy lehetséges, hogy egy a v-hez hasonló mondat szerepel az $A_1 \setminus A_2$-ben. □

Megjegyzés: A leggyakrabban használt hasonlósági függvények közé tartozik a negatív euklideszi távolság, illetve a koszinusz hasonlóság. Ahhoz, hogy ezt szövegek esetén használni tudjuk szükség van olyan módszerre, ami a szöveges adatunkat számokká alakítja. Erre jó megoldás például a 2.2 alfejezetben bemutatott módszer.
A definícióból természetes, hogy a függvénynek szüksége van egy (nemnegatív) hasonlósági mátrixra. Ezt a később tárgyalt csomagban direkt úton is meg lehet

adni, illetve úgy is, hogy megadjuk az adathalmazt, majd egy függvényt, amivel kiszámoljuk a hasonlósági mátrixot.

4.1.3. Tulajdonság alapú függvény

Egyszerű meggondolni, hogy az $\mathcal{F}(A) = \Phi(|A|)$ függvény szubmoduláris minden $\Phi : \mathbb{R} \to \mathbb{R}$ monoton konkáv függvény esetén. [8]
Megjegyzés: [8] Ha a Φ függvény konvex, akkor az $\mathcal{F}$ függvény szupermoduláris. Az $\mathcal{F}(A) = \Phi(\sum_{a \in A} m(a))$ függvény szintén szubmoduláris, ha az $m(a)$ értékek nemnegatív valós számok, ugyanis egy moduláris és egy monoton növő konkáv függvény kompozíciója szubmoduláris. (lásd 3.2.4 tétel)
Ezt kiterjesztve kapjuk azon függvénycsaládot, melynek tagjait gyakran tulajdonság alapú függvényeknek (feature based functions) hívnak.

4.1.9. Definíció (Tulajdonság alapú függvény [8])**.** *Legyen a $\Phi : \mathbb{R} \to \mathbb{R}$ konkáv függvény, továbbá legyenek az $m_d(A)$ számok $\forall d \in D$ esetén nemnegatív valósak. Ekkor tulajdonság alapú függvénynek nevezzük a következő halmazfüggvényt:*

$$\mathcal{F}_{tul.}(A) = \sum_{d \in D} \Phi\big(\sum_{a \in A} m_d(a)\big)$$

4.1.10. Állítás. *A tulajdonság alapú függvények szubmodulárisak.*

Bizonyítás. Az aktuális alfejezet elején részletezett tulajdonságok alapján már csak azt kell használni, hogy a szubmodularitás zárt a nemnegatív lineáris kombinációképzésre. (3.2.1 Állítás) □

Megjegyzés: A Φ konkáv függvényre gyakran használt példa a négyzetgyök, illetve a logaritmus függvény.
A tulajdonság alapú függvények a sorredukciós feladatokban azért lehetnek hasznosak, mert a D halmazt el tudjuk úgy képzelni, mint az oszlopokat tartalmazó halmazt. Minden oszlophoz, feature-höz tartozik egy monoton konkáv függvény, amelyben el lehet kódolni, hogy az az adott oszlop mennyire fontos, továbbá az $m_d(a)$ érték megmondja, hogy az a mintában mennyire lelhető fel az adott d tulajdonság. A függvény szubmodularitása miatt azt jutalmazza, ha minél különbözőek

a minták. Mint arra a fejezet elején utaltam, az adathalmaz transzponálásval ezt a módszert lehet használni a dimenziócsökkentési eljárásokban is.
A tulajdonság alapú függvényeknek egy speciális esete a fedési függvény (max-coverage), ami a 3.1.4-es példában említett maximális lefedettségi problémáról (max-k-cover) kapta a nevét. Tegyük fel, hogy a vizsgált adathalmaz bináris értékeket tartalmaz, azaz az i-edik sor j-edik eleme pontosan akkor 1, ha az i-edik mintában fellelhető a j-edik tulajdonság. A fedési függvénnyel a célunk azon oszlopok számának maximalizálásása, amik legalább egy kiválasztott mintában 1-es értéket vesznek fel.

4.1.11. Definíció (Fedési függvény [8])**.** *Jelölje a vizsgált $|D|$ dimenziós, azaz $|D|$ oszlopból álló adathalmazt, bináris értékeket tartalmazó mátrixot M. Legyen A a soroknak egy részhalmaza. Ekkor a fedési függvény a következő halmazfüggvény:*

$$\mathcal{F}_{fed}(A) = \sum_{d \in D} \min\left(1, \sum_{a \in A} M_{a,d}\right)$$

4.1.12. Állítás. *A fedési függvény szubmoduláris.*

Megjegyzés: Mint már említettük, az imént definiált fedési függvény a 3.1.4 példában definiált fedési függvény sepciális esete, tehát szubmoduláris. Másik megközelítés, hogy a fedési függvényre úgy is lehet tekinteni, mint egy speciális tulajdonság alapú függvény, ahol a Φ függvény a $min(a, 1)$ konkáv függvény. Mivel a tulajdonság alapú függvényről beláttuk, hogy szubmoduláris, ezért a fedési függvény is az.

4.2. Szubmoduláris függvények maximalizálása

Néhány fontos szubmoduláris függvény ismertetése után most a 4. fejezet elején definiált szubmoduláris függvények k korlátos maximalizálási feladatának általános megoldását mutatjuk be. Arról már a szó esett, hogy a szubmoduláris függvények maximalizálása egy NP-nehéz feladat (4.0.1), viszont bizonyos esetekben mégis jól kezelhető a probléma:
Ha az $\mathcal{F}$ függvény monoton növő szubmoduláris, akkor van egy $(1 - \frac{1}{e}) \sim 0,63$-

approximáló mohó algoritmus (Fischer, Nemhauser, Wolsey 1978). Ezen algoritmus [10] lépései:

Legyen $S = \emptyset$
while $|S| \leq K$ **do**
Adjuk hozzá S-hez azt az i elemet, ami maximalizálja $\mathcal{F}(S \cup i)$-t
end while

Ez az egyszerű mohó algoritmus az alapja a szubmoduláris optimalizálásnak.

4.2.1. Tétel ([10]). *Ha $\mathcal{F}$ monoton növő, szubmoduláris és $\mathcal{F}(\emptyset) = 0$, akkor a fenti mohó algoritmus olyan S megoldást ad, amire*

$$\mathcal{F}(S) \geq \left(1 - \frac{1}{e}\right) \cdot OPT$$

ahol $OPT = \max_{S:|S|\leq K} \mathcal{F}(S)$.

A tétel bizonyítása teljes indukcióval történik. A bizonyítás részletei az alapképzés során írt szakdolgozatomban megtalálhatók. [15]

4.2.2. Tétel (Feige, 1998 [10]). *A fönti feladatra polinom idejű, $(1 - \frac{1}{e}) + \varepsilon$ ($\varepsilon > 0$ tetszőleges) közelítő algoritmus nem létezik, ha $P \neq NP$.*

4.3. Példa szubmoduláris dimenziócsökkentésre

Szeretnénk egy gyakorlati, adathalmaz redukciós példát mutatni arra, ahol a feladat optimalizálása ekvivalens egy speciális tulajdonság alapú függvény maximalizálásával. [9]
Használjuk az 1.1 fejezetben definiált fogalmakat, jelöléseket, illetve tegyük fel, hogy a tanítási modellünk a Naive-Bayes modell. Definiáljuk az L veszteségfüggvényt az $S \subseteq U$ részhalmazon a következőképpen, ahol a Θ paramétert jelöl: $L(\Theta, S) = \sum_{i \in S} L(\Theta, x^i, y^i)$. Ezek alapján legyen $L_T(\Theta, U)$ a tanítási halmazon, $L_V(\Theta, V)$ a validációs halmazon mért teljes veszteség. A feladat az, hogy a tanítási halmaz egy olyan S részhalmazát válasszuk ki, hogy a validációs halmazon

mért veszteség minimális legyen. Formalizálva:

$$\arg\min_{S\subseteq U,|S|\leq k} L_V(\arg\min_{\Theta} L_T(\Theta,S),V)$$

Ha az L_T, L_V veszteségfüggvényeket a log-likelihood függvényekkel (LL_T, LL_V) helyettesítjük, akkor egy maximalizálási feladatot kapunk, melyet nevezzünk **Max-1** problémának:

$$\arg\max_{S\subseteq U,|S|\leq k} LL_V(\arg\max_{\Theta} LL_T(\Theta,S),V)$$

Néhány további jelölésre még szükség van. Először is partícionáljuk a tanítási halmazunkat a célváltozóértékek szerint: $U = \cup_{y\in Y} U^y$, ahol U^y azon adatpontok, melyekhez tartozó címke y. A tanítási halmaz minden részhalmazához kiszámolt maximum-likelihood becslést (MLE) jelölje $\Theta(S)$.
Definiáljuk a tanítási halmaz részhalmazain értelmezett $\mathcal{F}^v_{NB}$ halmazfüggvényt:

$$\mathcal{F}^v_{NB}(S) = \sum_{j=1}^{d}\sum_{x_j\in X}\sum_{y\in Y} m_{x_j,y}(V)\log m_{x_j,y}(S),$$

ahol $m_{x_j,y}(S) = \sum_{i\in S}\chi(x^i_j = x_j \wedge y^i = y)$, azaz megadja, hogy hány adatpontra teljesül, hogy a j-edik tulajdonsága x_j és a hozzá tartozó célváltozó érték y. A továbbiakban a következő függvényt is használni fogjuk, ezért szeretném itt megemlíteni: $m_y(S) = \sum_{i\in S}\chi(y^i = y)$. Ezen jelöléseket használva ki tudjuk fejezni a $\Theta_{x_j|y}(S)$ feltételes valószínűségeket: $\Theta_{x_j|y}(S) = \frac{m_{x_j,y(S)}}{m_y(S)}$, továbbá $\Theta_y(S) = \frac{m_y(S)}{|S|}$.

4.3.1. Állítás. *Az $\mathcal{F}^v_{NB}$ függvény szubmoduláris.*

Bizonyítás. Használjuk, hogy az $\mathcal{F}^v_{NB}$ függvény egy speciális tulajdonság alapú függvény (4.1.9). □

Definiáljuk a **Max-2** problémát:

$$\arg\max_{S\subseteq U,|S\cap V^y|=k\frac{|V^y|}{|V|},|S|=k} \mathcal{F}^v_{NB}(S)$$

4.3.2. Állítás. *A Max-1 probléma a naiv Bayes modell alkalmazása esetén ekvi-*

valens a Max-2 szubmoduláris maximalizálási feladattal. [9]

Bizonyítás. [9]
Írjuk fel a log-likelihood függvényt a validációs halmazon és használjuk a bevezetett jelöléseket.

$$l_V^{NB}(S) = \sum_{i \in V} \log p(x^i | y^i; \Theta_S) + \log p(y^i; \Theta_S) =$$

$$= \sum_{i \in V} \sum_{j=1}^{d} \log \Theta_{x_j|y}(S) + \sum_{i \in V} \log \Theta_y(S) = \sum_{i \in V} \sum_{j=1}^{d} \log \frac{m_{x_j,y}(S)}{m_y(S)} + \sum_{i \in V} \log \frac{m_y(S)}{|S|} =$$

$$= \sum_{j=1}^{d} \sum_{x_j \in X} \sum_{y \in Y} m_{x_j,y}(V) \log m_{x_j,y}(S) - (d-1) \sum_{y \in Y} m_y(V) \log m_y(S) - |V| \log |S|$$

Tehát a log-likelihood függvényt a következő tagokra tudjuk bontani:

(i) $\sum_{j=1}^{d} \sum_{x_j \in X} \sum_{y \in Y} m_{x_j,y}(V) \log m_{x_j,y}(S)$

(ii) $(d-1) \sum_{y \in Y} m_y(V) \log m_y(S)$

(iii) $|V| \log |S|$

Az (i) definíció szerint az $\mathcal{F}_{NB}^{v}$ szubmoduláris függvény S részhalmazon felvett értéke. Hogyha megköveteljük, hogy $|S| = k$, akkor a (iii) tag konstans. Ha pedig azzal a további feltétellel élünk, hogy S kiegyensúlyozott, azaz $|S \cap V^y| = k\frac{|V^y|}{|V|}$, akkor a (ii) tag is konstans lesz. Ezzel bebizonyítottuk, hogy a Max-1 és a Max-2 optimalizálási feladatok ekvivalensek. □

Hasonlóan egy szubmoduláris maximalizálási feladatot kapunk, ha a legközelebbi szomszédok (nearest-neighbor) modellt használjuk. [9]
A cikk írói [9] egy GLISTER-ONLINE (GeneraLIzation based data Subset selecTion for Efficient and Robust learning) nevű algoritmust adtak a problémára, ami egyszerre valósítja meg a részhalmaz kiválasztási feladatot és a paraméter tanulást.

4.4. További mohó algoritmusok

Az Apricot csomagot használva nem csak az alkalmazni kívánt függvényt, hanem magát az algoritmust is megválaszthatjuk. Ezek az algoritmusok lényegében a

már bemutatott mohó algoritmusra (4.2) épülnek. Hat algoritmust mutatunk be részletesebben.

4.4.1. Naiv mohó algoritmus

A naiv mohó (naive greedy [1]) algoritmus a legegyszerűbb megközelítés a szubmoduláris függvények optimalizálására. Minden iterációban sorra veszi a még nem kiválasztott összes elemet és kiszámolja, hogy melyiknek mennyi a hozzáadott értéke. A legnagyobb hozzáadott értékű elemet fogja kiválasztani. Ez ugyanaz, mint a 4.2 fejezetben ismertetett algoritmus.

4.4.2. Lusta mohó algoritmus

Mivel a lusta mohó (lazy greedy [2]) algoritmus és az naiv mohó algoritmus csak az implementációjukban különböznek, ezért igaz a következő egyszerű állítás.

4.4.1. Állítás. *A naiv és a lusta mohó algoritmust ugyanazon adathalmazon futtatva ugyanazt az eredményt kapjuk.*

Itt egy maximum prioritású sorban tároljuk a még nem választott elemeket, ahol a legutoljára kiszámolt hozzáadott értékek lesznek a prioritások. A függvényünk szubmoduláris tulajdonságát, azaz azt, hogy a hozzáadott érték nem nőhet próbáljuk kihasználni ahhoz, hogy csökkentsük a futási időt. Ha a maximum prioritású sorban lévő első elem értékét újraszámoljuk és az még mindig nagyobb, mint a második elemé, akkor őt választjuk be ebben az iterációban, ellenkező esetben ezt folytatjuk tovább. Látszik, hogy lehet, hogy nem kell annyiszor újraszámolni az egy elem beválasztásával szerzett nyereséget, ezzel sokat nyerhetünk. Viszont a másik oldalról muszáj megemlíteni, hogy a maximum prioritású sor karbantartása olykor sok számítási kapacitást igényelhet.

[1] https://apricot-select.readthedocs.io/en/latest/optimizers/naive.html
[2] https://apricot-select.readthedocs.io/en/latest/optimizers/lazy.html

4.4.3. Kétfázisú mohó algoritmus

Ezen két algoritmus vegyítése a kétfázisú mohó (two-stage greedy [3]) algoritmus, ahol az első k (felhasználótól függő paraméter) lépésben a naiv módszert használjuk, majd átváltunk a lusta algoritmusra. Emögött az az intuíció rejlik, hogy a lusta mohó által használt adatstruktúra (a maximum prioritású sor) karbantartása az algoritmus elején a legnehezebb, ezért ekkor célszerű a naiv módszert használni, viszont jó ötlet lehet idővel váltani a két módszer között.

4.4.4. Nem determinisztikus mohó algoritmusok

A másik három módszer az eddigiekkel ellentétben már nem determinisztikus. A sztochasztikus mohó (stochastic greedy [4]) algoritmus minden iterációjában véletlenül választ egy részhalmazt, amiből a következő elemet választja; a mintavételező mohó (sample greedy [5]) algoritmus során pedig már az algoritmus legelején történik egy véletlen mintavételezés. A lusta közelítő mohó (approximate lazy [6]) algoritmus a lusta mohó algoritmus egy nagyon egyszerű kiterjesztése: nem feltétlenül azt az elemet vesszük hozzá a már kiválasztott részhalmazhoz, ami a legnagyobb haszonnal jár, hanem ami "közel legjobb haszonnal jár". Ezt százalékban lehet kifejezni és természetesen függ a felhasználótól. Az utóbbi 3 algoritmus a futásidő felgyorsítására, a számítási kapacitás csökkentésére törekszik, viszont már nem feltétlenül hoz olyan eredményt, mint az első három determinisztikus algoritmus.

[3] https://apricot-select.readthedocs.io/en/latest/optimizers/two-stage.html
[4] https://apricot-select.readthedocs.io/en/latest/optimizers/stochastic.html
[5] https://apricot-select.readthedocs.io/en/latest/optimizers/sample.html
[6] https://apricot-select.readthedocs.io/en/latest/optimizers/approx-lazy.html

5. Vizsgált adathalmazok

A vizsgált adatok, melyeken a már említett Apricot módszert kipróbáltuk a kaggle[7] oldalon megtalálhatók.

5.1. Titanic: Machine Learning from Disaster

Az első adathalmaz címe, amin kipróbáltuk a módszert: Titanic - Machine Learning from Disaster[8]. Ez az adathalmaz 881 sorból és 67 oszlopból áll. Minden sorban egy, a Titanicon utazó személy különböző adatai szerepelnek, illetve az, hogy túlélte-e a katasztrófát, vagy sem. A tanulási folyamat során egy klasszifikációs (1.1) feladatot szeretnénk megoldani, meg szeretnénk jósolni a különböző adatokból a túlélők halmazát.

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

4. ábra. A Titanic: Machine Learning from Disaster című adathalmaz első öt sora.

5.2. Disaster Tweets (NLP feladat)

A Disaster Tweets[9] című adathalmazzal egy természetes nyelvfeldolgozási (Natural Language Processing, NLP) feladatot vizsgáltunk. Az adat különböző tweetekből áll, melyeket klasszifikálni (1.1) szeretnénk aszerint, hogy valójában egy igazi katasztrófáról szólnak-e, vagy nem. Az adathalmaz beágyazása, tisztítása után random train-test vágást használva a training adathalmaz 5264 sorból és 10000 oszlopból állt.

[7] https://www.kaggle.com
[8] https://www.kaggle.com/competitions/titanic/data
[9] https://www.kaggle.com/competitions/nlp-getting-started/data

	id	text	target
3806	5408	Former Township fire truck being used in Phili...	0
3444	4922	The Dress Memes Have Officially Exploded On Th...	0
3443	4920	Well as I was chaning an iPad screen it fuckin...	0
6219	8875	So does Austin smoke too since he agreed to th...	0
3440	4917	Im Dead!!! My two Loves in 1 photo! My Heart e...	0
...	...	...	...
3663	5213	@Truly_Stings Yo Dm me	1
3660	5210	Driver fatalities down on Irish roads but pede...	1
3659	5209	Message boards will display updated traffic fa...	1
3673	5228	Kosciusko police investigating pedestrian fata...	1
7612	10873	The Latest: More Homes Razed by Northern Calif...	1

5. ábra. A Disaster Tweets című adathalmaz néhány sora.

5.3. Movie-box income

A harmadik vizsgált adathalmaz (Movie-box income[10]) 7000 rekordja mind egy-egy filmről szóló információkat tartalmaz. A feladat jelen esetben regressziós (1.1), egy adott filmhez a bevétel megjósolása a cél.

[10]https://www.kaggle.com/datasets/rounakbanik/the-movies-dataset

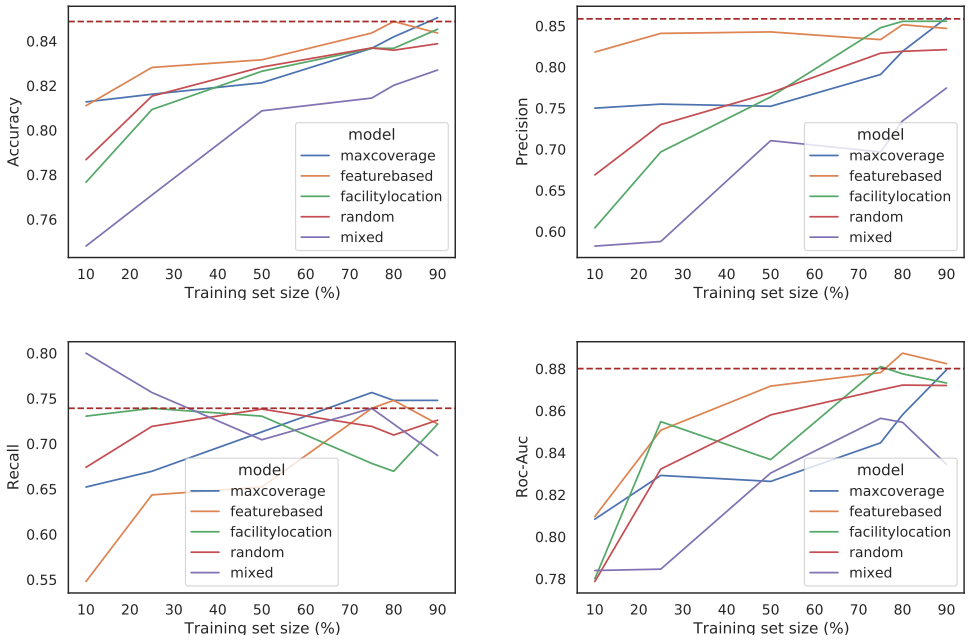
6. Eredmények

Az Apricot, Python csomag a -ben bemutatott redukciós feladatra ad egy megoldást, azaz egy lehetőséget biztosít hatalmas adathalmazok redukálására úgy, hogy az így kapott kisebb méretű adathalmaz reprezentatív legyen. Ezt a gépi tanulási folyamatok felgyorsítására lehet felhasználni oly módon, hogy a tanítási adathalmazt csökkentjük az Apricot programban implementált függvények segítségével, majd ezen a már jóval kevesebb sorból álló adathalmazon tanítjuk a modellünket. Ha a használt függvényt jól választjuk meg, akkor az így tanított modell eredményessége nem romlik számottevően, sőt a túltanulás jelensége miatt akár még jobb performanciabeli növekedés is lehet [8]. A csomag a feladatot szubmoduláris függvények (tulajdonság alapú függvény / featurebased (4.1.9), szolgáltató elhelyezési függvény / facilitylocation (4.1.7), fedési függvény / maxcoverage (4.1.11)) maximalizálásával oldja meg.

6.1. Sorredukció szubmoduláris módszerrel

Az Apricot program implementálói által publikált cikkekben [7] a hangsúlyt arra helyezték, hogy lehet a módszer segítségével a tanítási halmaz sorait redukálni. Témavezetőimmel először mi is ezzel a feladattal foglalkoztunk.
A Titanic: Machine Learning from Disaster című adathalmazon definiált klasszifikációs feladat (5.1) megoldásához a Gradient Boosting klasszifikációs modell (1.2.2) bizonyult a leghatékonyabbnak. Egy training-test vágás után a tanítási adathalmaz sorainak száma 590 volt (az eredeti adathalmaz $\frac{2}{3}$-a), ezt redukáltuk az Apricot-val (10%-ra, 25%-ra, 50%-ra, 75%-ra, 80%-ra, 90%-ra), és ezen a kisebb adaton tanítottuk a modellt, majd megvizsgáltuk, hogy az eredmény hogy változott a kiértékelésre félretett teszt adathalmazon. A már említett Apricot függvényeket használtuk, illetve ezek egy vegyítését (mixed), azaz olyat, amikor mindhárom függvénnyel kiválasztottunk ugyanannyi sort és ezeket egyesítettük, majd ezen tanítottuk a modellt. Emellett a random kiválasztást is kipróbáltuk és azért, hogy ebben az esetben is valós képet kapjunk, többszöri futtatás eredményeit átlagoltuk ki.

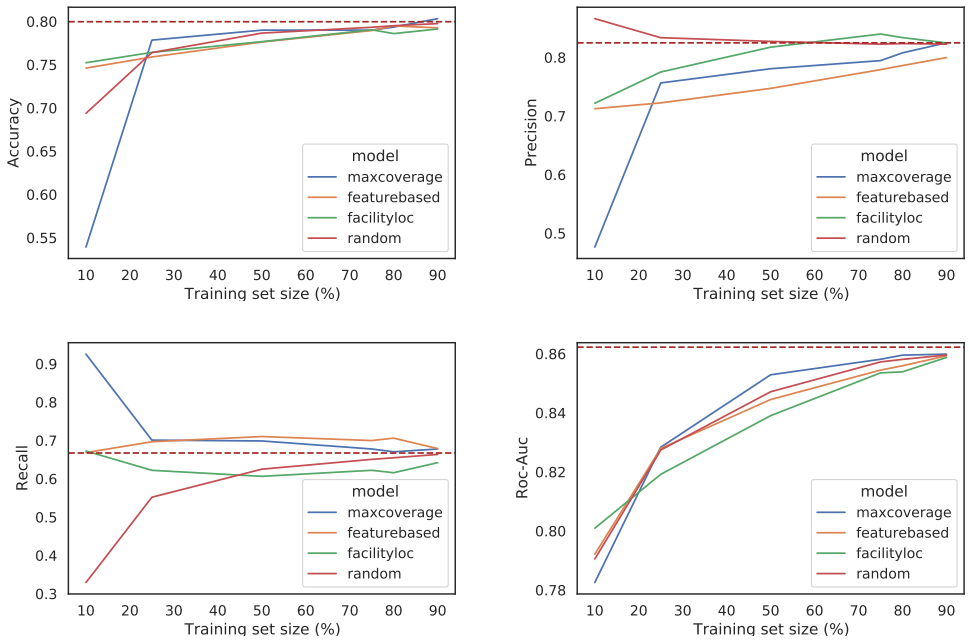
Az egyes modellek teljesítményét az 1. fejezetben bemutatott négy különböző metrikával (1.1) mértük: accuracy (a helyes jóslatok aránya), precision (azok közül, akiket a modell túlélőnek tart, valóban hányan élték túl), recall (azt mutatja, hogy a ténylegesen túlélők közül hányat talált meg a modell), roc auc (annak a valószínűsége, hogy a modell egy random túlélőként címkézett mintát előrébb helyez a rangsorban, mint egy random nem túlélőt).



6. ábra. A Titanic - Machine Learning from Disaster című adaton definiált klasszifikációs feladaton elért performanciabeli eredmények az egyes kiértékelési metrikák (accuracy, precision, recall, roc-auc) szerint, a tanítási halmaz méretének csökkentése mellett. A szaggatott vonal minden ábrán azt jelzi, hogy milyen eredményt értünk el amikor a teljes tanítási halmazon tanítottuk a modellt.

Az eredményeket grafikonokon (6) szemléltettük, melyekről könnyen leolvasható, hogy a legjobb értékeket a feature-based modellel értük el, viszont meglepő eredmény, hogy a tanulási adathalmaz véletlen redukcióját nem sokkal múlja felül az Apricot módszer. Ennek oka lehet például, hogy jelen esetben egy viszonylag kis

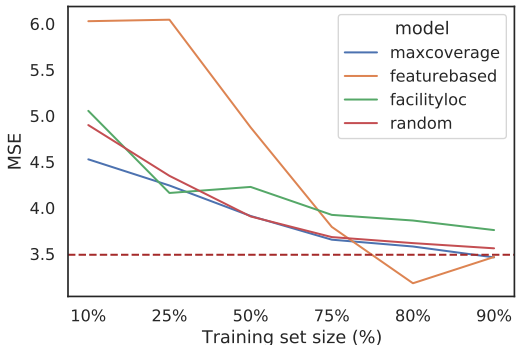
adathalmazzal dolgoztunk, illetve az is egy ok lehet, hogy nem volt eléggé redundáns az adat. Azt még fontos kiemelni, hogy a szubmoduláris kiválasztás ereje a legjobban az adathalmaz $10 - 25\%$-ára való csökkentésekor látszik. Itt a random módszer rendre gyengébb eredményt mutat, mint például a feature-based.
Az első adathalmaz kis mérete, illetve a redundancia hiánya miatt nem sok információt árult el a feltérképezendő program működéséről, hatékonyságáról, ezért másik adaton, a Disaster Tweets nevű adathalmazon (5.2) folytattuk a munkát.



7. ábra. Az NLP - Disaster Tweets című adaton definiált klasszifikációs feladaton elért performanciabeli eredmények az egyes kiértékelési metrikák (accuracy, precision, recall, roc-auc) szerint, a tanítási halmaz méretének csökkentése mellett. A használt modell a logisztikus regresszió. A szaggatott vonal minden ábrán azt jelzi, hogy milyen eredményt értünk el amikor a teljes tanítási halmazon tanítottuk a modellt.

Először a sorok számát próbáltuk csökkenteni az Apricot programot használva $10, 25, 50, 75, 80, 90$ százalékra, majd ezen tanítottuk a modellt, mely először a

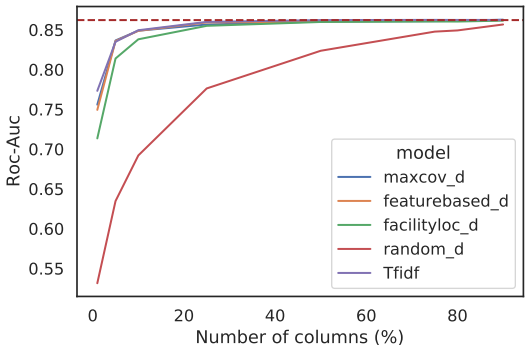
logisztikus regresszió (1.2.1) volt. A redukcióhoz ismét az Apricot csomagban implementált függvényeket, illetve random kiválasztást használtunk (ez utóbbit többször, majd kiátlagoltuk az eredményeket). A mért eredményeket az 7 ábrák szemléltetik. Ezekről leolvasható, hogy ha az adathalmazunk sorait 10%-ra csökkentjük, akkor jelentősen romlik az elért eredmény, viszont a sorok számát felére csökkentve már nem olyan lényeges a különbség, az így betanított, illetve az egész adaton tanított modell között. Ami szintén szembetűnő, hogy sajnos az Apricot nem sokkal múlja felül a random módszert, olykor egyáltalán nem.
Ezután megpróbáltuk más modellek tanítására használni a módszert, mint például a Gradient Boosting (1.2.2) vagy a Random Forest (1.2.2) klasszifikációs modell. A Gradient Boosting Classifier esetében először találkoztunk azzal a problémával, hogy nem volt eléggé jól paraméterezve a modell, ezért nagymértékben túltanult. Ennek következtében, amikor az Apricot módszerrel csökkentettük a sorok számát, javulást tapasztaltunk a modell eredményességét tekintve, viszont a modell további hiperparaméterezése után ez a jelenség már nem volt jelen. Összességében az mondható el, hogy a sorok redukciójával nem értük el a várt eredményt a vizsgált adathalmazon.
Végül a harmadik vizsgált adaton (Movie-box income (5.3)) definiált regressziós feladathoz is teszteltük a módszert. A legjobbnak bizonyult modell a lightGBM boosting (1.2.2), a használt kiértékelési metrika pedig az átlagos négyzetes eltérés (mse - mean squared error (1.1)) volt. A 8. ábráról leolvasható, hogy az adathalmaz sorainak random módszerrel vett redukálását kezdetben a maxcoverage, később pedig a feature-based függvénnyel való redukálás múlja felül. Viszont azt is láthatjuk, hogy performanciát tekintve nincs nagy nyereség.



8. ábra. A MOV movie-box income című adaton definiált regressziós feladaton elért performanciabeli eredmények az átlagos négyzetes eltérés (mse) kiértékelési metrika szerint, a tanítási halmaz sorai számának csökkentése mellett. A használt modell a lightGBM boosting. A szaggatott vonal azt jelzi, hogy milyen eredményt értünk el amikor a teljes tanítási halmazon tanítottuk a modellt.

6.2. Dimenziócsökkentés szubmoduláris módszerrel

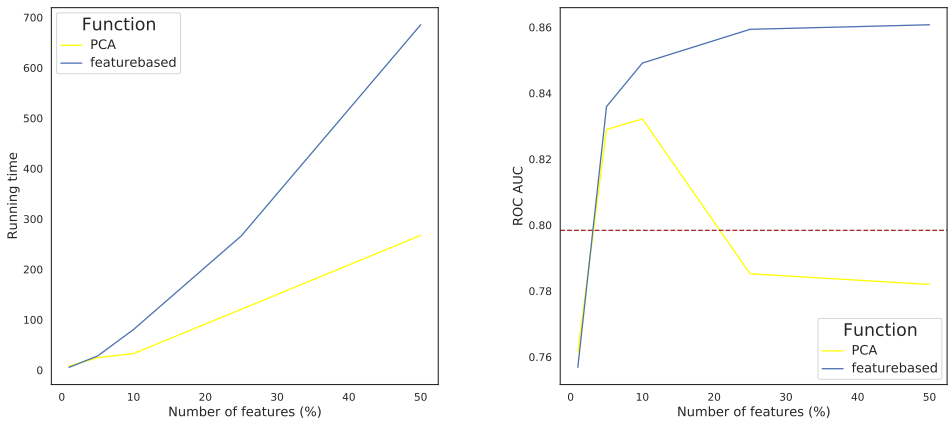
Ebben az alfejezetben olyan méréseket mutatunk be, amelyek a sorok helyett az oszlopok hatékony redukciójára keresnek megoldást. Mint már említettük az Apricot csomagot bemutató cikkekben a sorok redukálására használták a módszert. Ehhez a Distaster Tweets (5.2) című adatot használtuk. Az adathalmazt tartalmazó mátrixot transzponáltuk, majd az így kapott mátrix sorait az Apricot módszerrel csökkentve kapott mátrixot visszatranszponáltuk. Végül ezen tanítottuk a modellt. A logisztikus regresszió modellt (1.2.1) és a ROC értéket (1.1) tekintve a következő eredményt értük el:



9. ábra. Az NLP - Disaster Tweets című adaton definiált klasszifikációs feladaton elért performanciabeli eredmények a roc-auc kiértékelési metrika szerint, a tanítási halmaz oszlopai számának csökkentése mellett. A használt modell a logisztikus regresszió. A szaggatott vonal azt jelzi, hogy milyen eredményt értünk el amikor a teljes tanítási halmazon tanítottuk a modellt.

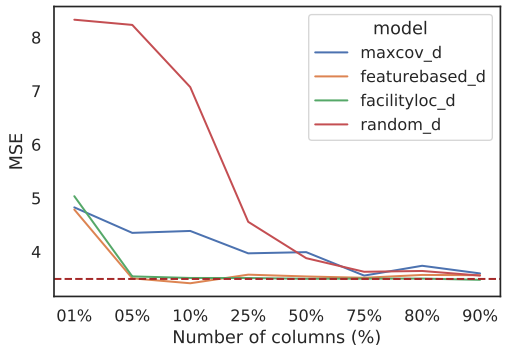
A 9. ábráról leolvasható, hogy a 10000 oszlopot már 10%-ra csökkentve is alig romlik a modell. Azt is fontos hangsúlyozni, hogy ebben az esetben a szubmoduláris kiválasztás (bármely függvényt használva) sokkal jobb eredményt produkál, mint a random szelekció. A szubmoduláris megközelítés mellett a Tfidf (term frequency–inverse document frequency) értékeket használó szöveges adatoknál jól működő dimenziócsökkentő módszert (2.2) is megvizsgáltuk az adathalmazon. Ezen módszerrel az Apricot-hoz hasonló eredmény született. (Az ábrán ezt a sárga görbe jelzi.) Úgy gondoltuk, hogy hasznos lehet ebben az irányban folytatni a kutatást és esetleg olyan adathalmazokon kipróbálni az oszlopok redukálását, ahol nem tudjuk a Tfidf módszert használni.
Az Apricot módszert a már bemutatott PCA dimenziócsökkentő eljárással (2.1) is összehasonlítottuk. Azért éreztük ennek fontosságát, mert az előző bekezdésben említett módszer egy kifejezetten NLP feladatokra működő algoritmus. Szerettük volna egy sokkal általánosabb eljárással, más típusú adatokon is működő dimenziócsökkentő módszerrel is összehasonlítani a szubmoduláris megközelítést. Az összehasonlítást a feature-based függvénnyel, ennek is a legjobb paraméterezésével (6.4) végeztük. A két módszert teljesítményt és futási időt tekintve is megmértük.

A PCA sok számítási kapacitást igénylő módszer, ennek ellenére jobb futási időt adott, mint az Apricot (10. ábra). Performanciát tekintve (10. ábra) viszont a PCA éri el a gyengébb eredményt. A 10. ábráról leolvasható, hogy az AUC értékeket tekintve a feature-based függvénnyel nagyságrendekkel jobb eredményt kapunk, mint a PCA-val.



10. ábra. Az Apricot (kék) és a PCA (sárga) módszer összehasonlítása futásidőt (bal), illetve teljesítményt (jobb) tekintve. A teljesítmény méréséhez kiértékelési metrikaként az auc értékeket használtuk. A jobb oldali ábrán a szaggatott vonal a dimenziócsökkentés nélkül tanított logisztikus regresszióval elért eredményt szemlélteti.

Nemcsak ezen az adathalmazon, hanem a Movie-box income című kaggle adaton (5.3) is kipróbáltuk a szubmoduláris dimenziócsökkentést. A tanításhoz ismét a lightGBM boosting (1.2.2) módszert használtuk, ugyanis ez bizonyult a legeredményesebbnek. Az elért eredmények a 11. ábrán láthatók. A 8. ábrával összevetve látszódik, hogy a dimenziócsökkentési feladat esetén sokkal jobb eredményt lehet elérni az Apricot módszerrel, mint a sorredukálás esetén.



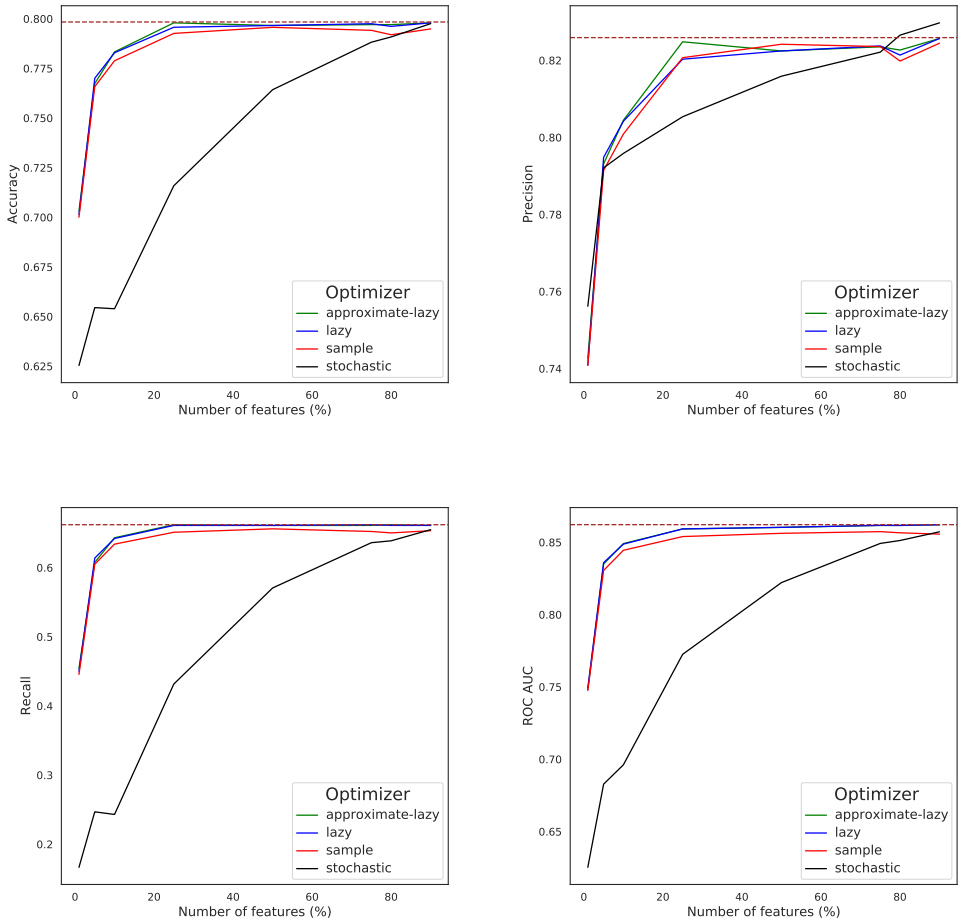
11. ábra. A MOV movie-box income című adaton definiált regressziós feladaton elért performanciabeli eredmények az átlagos négyzetes eltérés (mse) kiértékelési metrika szerint, a tanítási halmaz oszlopai számának csökkentése mellett. A használt modell a lightGBM boosting. A szaggatott vonal azt jelzi, hogy milyen eredményt értünk el amikor a teljes tanítási halmazon tanítottuk a modellt.

Megjegyzés: Úgynevezett dupla redukcióval is próbálkoztunk, azaz először csökkentettük az oszlopok számát, majd a sorok számát is, természetesen az Apricot csomagban implementált függvényekkel, viszont így nagy mértékben romlott az elért erdmény.

6.3. Algoritmusok összehasonlítása

Egy másik fontos feladat, mellyel a mesterképzés során témavezetőimmel foglalkoztunk az volt, hogy a csomagban implementált mohó algoritmusokat (4.4) teszteljük, ezért ezeket performanciájukat és futásidejüket tekintve is megpróbáltuk összehasonlítani. Ezt az összehasonlítást a Disaster Tweets (5.2) című adathalmazon végeztük. A dolgozat 4.4. fejezetében írtak után látszik, hogy a naiv (naive (4.4.1)), lusta (lazy (4.4.2)) és a kétfázisú (two-staged (4.4.3)) algoritmusoknak elég csupán a futásidejüket vizsgálni, mivel ugyanazt az eredményt adják, csak az implementáció más. Mivel a kétfázisú algoritmus azt a célt szolgálná, hogy a naiv (és esetleg a lusta) algoritmus futásidejét gyorsítsa, ezért azt vártuk, hogy gyorsabb lesz, mint a többi módszer. Ennek ellenére az 1. táblázatból látszik,

hogy a vizsgált adaton még a naiv módszertől is lassabb volt, a lusta módszer pedig nagyságrendekkel jobb nála.



12. ábra. A vizsgált mohó algoritmusok összehasonlítása teljesítményt tekintve a Disaster Tweets című adaton. A szaggatott vonal a dimenziócsökkentés nélkül tanított logisztikus regresszióval elért eredményt mutatja.

A lusta algoritmus még a nemdeterminisztikus eljárásoktól (4.4.4) is gyorsabban futott le. Ezt úgy vizsgáltuk, hogy a lusta közelítő (approximate greedy), a mintavételező (sample greedy) és a sztochasztikus mohó (stohastic greedy) algorit-

Optimizer \ Size	1 %	5%	10 %	25%	50%
approximate-lazy	9.87	38.29	85.97	318.35	838.84
lazy	**5.85**	**8.65**	**24.38**	**166.21**	**670.32**
naive	8.79	21.41	49.76	216.48	783.08
sample	12.15	47.94	96.67	257.53	870.33
stochastic	8.07	18.83	43.26	217.42	785.12
two-stage	6.71	24.62	50.72	238.43	827.12

1. táblázat. A vizsgált mohó algoritmusok összehasonlítása futásidőt tekintve a Disaster Tweets című adathalmazon. A tanítási modell a logisztikus regresszió volt, a módszerrel a tanítási halmaz oszlopainak számát redukáltuk.

musokat többször futtattuk, a futásidőket kiátlagoltuk és így hasonlítottuk össze a lusta algoritmus futásidejével. A kapott eredmény azért is meglepő, mert ezen algoritmusoknak a célja az lenne, hogy bizonyos mértékű performancia romlással számítási kapacitás, valamint futásidő csökkentést érjünk el. Ez a mérés alapján a vizsgált adaton nem működött. (A futásidőbeli különbségeket az 1. táblázat, a teljesítménybeli különbségeket a 12. ábra mutatja.)

6.4. Legjobb paraméterezés megkeresése

A feature-based szubmoduláris függvény legjobb paraméterezésének vizsgálatával (az AUC értéket tekintve) is sok időt töltöttünk. A kapott eredményeket a 2. táblázat mutatja.

Size	Best parameters
1 %	{"function": "log", "optimizer": "lazy"}
5 %	{"function": "log", "optimizer": "approximate-lazy"}
10 %	{"function": "sqrt", "optimizer": "approximate-lazy"}
25 %	{"function": "log", "optimizer": "approximate-lazy"}
50 %	{"function": "log", "optimizer": "lazy"}

2. táblázat. A feature-based függvény legjobb paraméterezése az auc értékek szerint, a különböző méretű redukciók során. A mérésekhez a Disaster Tweets című adathalmazt használtuk.

Meglepő, hogy a Disaster Tweets nevű adaton (5.2) futtatott mérések alapján az approximate lazy, azaz a lusta közelítő algoritmus (4.4.4) is szerepel a táblázatban, viszont konstatálható, hogy jó döntés, ha a konkáv függvényünket a $\log(X + 1)$-nek, a maximalizálásokat végző mohó algoritmust a lazy-nek, azaz a lusta mohónak (4.4.2) választjuk.

7. Összegzés

A dolgozatban a gépi tanulási fogalmak, osztályozó és regressziós modellek, elterjedt dimenziócsökkentési eljárások, továbbá az alapfogalmak ismertetése után megmutattuk, hogy a gépi tanulásban gyakorta előforduló redukciós feladatokat visszavezethetjük szubmoduláris függvények maximalizálására. Ez a redukciós feladat lehet sorredukció, vagyis az adathalmaz sorainak csökkentése, illetve oszlopredukció, más néven dimenziócsökkentés is.
Különböző szubmoduláris függvényeket és több mohó algoritmust is ismertettünk, melyeket jól lehet alkalmazni a gépi tanulásban. Az elméleti összefoglaló után bemutattunk egy, a Washingtoni Egyetem kutatói által fejlesztett Python csomagot [7], mellyel a fenti redukciós feladat megvalósítható. A méréseket, eredményeket tekintve összességében elmondható, hogy van létjogosultsága a szubmoduláris függvényekkel való adathalmaz redukciónak. Főleg a dimenziócsökkentés területén bizonyult hasznosnak a módszer. A random kiválasztástól jobb performanciabeli eredményeket kapunk a dimenziócsökkentés területén, valamint azt is érdemes hangsúlyozni, hogy a módszert a determinisztikus jellegének köszönhetően elég egyszer futtatni. A már széleskörben használt dimenziócsökkentő módszereket (pca, szöveges adat dimenziócsökkentése) a szubmoduláris módszerrel összehasonlítva nem tapasztaltunk lényegesen nagy teljesítménybeli különbséget.

Irodalomjegyzék

[1] Frank András, Összefüggések a kombinatorikus optimalizálásban II. - Szubmoduláris optimalizálás és poliéderes kombinatorika, Operációkutatási Tanszék és MTA-ELTE Egerváry Kutatócsoport, Eötvös Loránd Tudományegyetem, Budapest (2009)

[2] L. Lovász, Mathematical Programming The State of the Art, Submodular functions and convexity, *Springer*, (1983), 235-257

[3] Hui Lin, Jeff Bilmes, A Class of Submodular Functions for Document Summarization, *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, (2011), 510–520

[4] Jure Leskovec, Anand Rajaraman, Jeff Ullman, Mining of Massive Datasets, Chapter 11: Dimensionality Reduction, Palo Alto, California, (2014), 405-437

[5] Sammut, C., Webb, G.I. (eds) Encyclopedia of Machine Learning. *Springer*, Boston, MA., (2011), 986–987

[6] Alexander Schrijver, A combinatorial algorithm minimizing submodular functions in strongly polynomial time, *Journal of Combinatorial Theory, Series B, Volume 80, Issue 2*, (2000), 346-355,

[7] Jacob Schreiber, Jeffrey Bilmes, William Stafford Noble, Apricot: Submodular selection for data summarization in Python, (2019)

[8] Jeffrey A. Bilmes, Submodularity In Machine Learning and Artificial Intelligence, *Arxiv, abs/2202.00132*, University of Washington, Seattle, Jan (2022)

[9] Krishnateja Killamsetty, Durga Sivasubramanian, Ganesh Ramakrishnan, Rishabh Iyer, GLISTER: Generalization based Data Subset Selection for Efficient and Robust Learning *Proceedings of the AAAI Conference on Artificial Intelligence 35. 9* (2021), 8110-8118

[10] Feige, Uriel, A threshold of ln n for approximating set cover. *Journal of the ACM, 45(4)*,(1998), 634 – 652

[11] Dr. Bodon Ferenc, Adatbányászati algoritmusok Elektronikus jegyzet (2002-2010) http://www.cs.bme.hu/ bodon/magyar/index.html

[12] Krause, Andreas, Singh, A., Guestrin, C. Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *The Journal of Machine Learning Research, 9* (2008), 235-284.

[13] F. J. Ferri, P. Pudil, M. Hatef, J. Kittler, Comparative Study of Techniques for Large-Scale Feature Selection (1994)

[14] https://scikit-learn.org/stable/modules/feature_selection.html

[15] Bartalis Dávid, Szubmoduláris függvények alkalmazása a mesterséges nyelvfeldolgozásban *Eötvös Loránd Tudományegyetem Természettudományi Kar*, Budapest (2020)

NYILATKOZAT

Név: Bartalis Dávid

ELTE Természettudományi Kar, szak: Alkalmazott matematikus MSc

NEPTUN azonosító: ZIR9ET

Diplomamunka címe:
Szubmoduláris függvények alkalmazása redukciós feladatokra a gépi tanulásban

A **diplomamunka** szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom önálló szellemi alkotásom, abban a hivatkozások és idézések standard szabályait következetesen alkalmaztam, mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2022. december 31.

a hallgató aláírása