SZAKDOLGOZAT

Termékajánlás valós példán demonstrálva

Göntér Anna

Mérnök Informatikus BSc szak

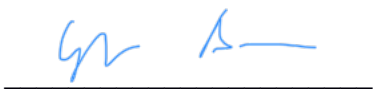
2020

Nyilatkozat

Alulírott, Göntér Anna (KHRV0P) Mérnök Informatikus, BSc szakos hallgató kijelentem, hogy a Termékajánlás valós példán demonstrálva című szakdolgozat feladat kidolgozása a saját munkám, abban csak a megjelölt forrásokat, és a megjelölt mértékben használtam fel, az idézés szabályainak megfelelően, a hivatkozások pontos megjelölésével.

Eredményeim saját munkán, számításokon, kutatáson, valós méréseken alapulnak, és a legjobb tudásom szerint hitelesek.

Győr, 2020.05.01.



hallgató

Kivonat

Termékajánlás valós példán demonstrálva

A termékajánló rendszerek egyre népszerűbbek lesznek a mai internetes világban. Amikor világhálón megtekintünk egy könyvet, filmet, zenét vagy akár szállást keresünk a termékajánló rendszerek folyamatosan dolgoznak a háttérben, hogy találjanak számunkra egy még jobb ajánlatot.

A termékajánló rendszerek hátterében komoly gépi tanulási algoritmusok állnak, amik nagyon sokrétűek lehetnek. Különböző algoritmusok, nagyon különböző eredményeket tudnak generálni. A hatékonyságuk függhet attól is, hogy milyen adatokat kap a rendszer a felhasználótól, vagy, hogy a felhasználó, mikor regisztrált, vannak-e előzmények. A dolgozatom első felében ezeket a lehetőségeket fogom körül járni és bemutatni.

A dolgozatom második felében pedig ki fogom próbálni ezeket az algoritmusokat egy adott adatkészleten, amit a Kaggle.com oldalon találtam. Az oldal prediktív és elemzési versenyek és kihívások számára lett kitalálva, ahova különböző kisebb-nagyobb vállalatok tölthetik fel az adatkészletüket és indíthatnak versenyeket.

Munkám célja egy a Kaggle.com oldalon talált adatkészleten kipróbálni a lehető legtöbb algoritmust és megtalálni azt, amelyik a legjobb ajánlást tudja készíteni az adott adatkészletre. A kiválasztott adatkészlet az Expedia szálláskereső oldal által felrakott verseny adatai. A verseny már lezárult, de az eredményeket továbbra is lehet látni, illetve az oldal továbbra is kiértékeli az általam feltöltött eredményeket.

Abstract

Recommendation Systems Demonstrating on Real Example

Nowadays the Recommender Systems are becoming increasingly popular, because of the world of internet, and social network. Whenever we are searching for books, movies or accommodations on the internet. The recommender systems are always working in the background to find a better item for us.

There are serious machine learning algorithms in the background of the recommendation systems. Different algorithms can produce very different outcoming results. The efficiency of the algorithms depends on several parameters, like the data from the users, the date of the registration or the history of the user.

In the first part of my thesis I am going to present these algorithms and the background of the recommender system.

In the second part of the thesis I am going to try these algorithms on a dataset what I found on the Kaggle.com website. On this website there are a lot of predictive and analytical competitions and challenges. Various smaller or larger companies can upload their datasets and launch competitions, which will help them improving their algorithms.

The aim of my work is to try out as many algorithms as possible on the dataset what I found on Kaggle.com and find the best recommendation algorithm for that dataset. The selected dataset is uploaded by Expedia accommodation search page. The competition is over, but the results are still visible, and the page can evaluate the results I upload.

Tartalomjegyzék

Bevezetés

Mindenki, aki vásárolt már online, találkozhatott az alábbi jelenséggel. Berakott egy terméket (például könyvet) a virtuális kosarába és ennek következményeképp az oldal azonnal ajánlásokat ad, hogy miket érdemes még megvennie mellé, vagyis feltűnik a jól ismert „Mit vettek még, akik ezt vették?” mondat. Az oldalnak egy algoritmus segít megtalálni a jó párosítást, az alapján, hogy a felhasználó eddig miket tekintett meg, miket vásárolt, miket vásároltak azok, akiknek hasonló érdeklődési körük van vagy ugyanazt a terméket vásárolták. Ma a legtöbb nagy cég használja a termékajánló rendszereknek az előnyeit és sok esetben jelentősen megnöveli a bevételeiket.

Az informatikusok először az 1990-es évek végén kezdtek el foglalkozni a gépi tanulással. [1] Majd 2002-ben megjelent a ***Torch: a modular machine learning software library*** című dokumentum, ami még jobb alapot adott neki. [2] De az ajánlórendszerek köztudatba robbanása 2006-ban történt, amikor a Netflix kihirdetett egy versenyt, aminek a főnyereménye 1.000.000$ volt, célja pedig, minél jobb becslést adni, hogy az adott illető mennyire értékelne egy adott filmet. Ezt gépi tanulással és az ajánlórendszer készítésével számolták ki. A számításhoz adott volt egy adatbázis, aminek az adatai alapján kellett a becsléseket készíteni. Több ezer csapat jelentkezett a világ minden tájáról, még Magyarországról is, akik 14. helyezést értek el. [3]

Termékajánlórendszerek

A termékajánló rendszer egy olyan rendszerre utal, ami a felhasználó számára megtalálja a személyre szabott ajánlatokat és csak azokat ajánlja. Azért van erre szükség, mert a digitális vásárlási folyamatnak köszönhetően túl sok választási lehetőségük van és ezeket szűkíteni kell. Régen az emberek nem online áruházban vásároltak, ahol a rendelkezésre álló termékek csak korlátozott számban voltak elérhetőek. Például egy könyvesboltban található könyvek száma csak annyi lehet, amennyi a boltban elfér, viszont az interneten nincsenek ilyen korlátok. Mivel nőtt a rendelkezésre álló termékek száma felmerült a probléma, hogy az emberek nagyon nehezen tudják kiválasztani, azokat a termékeket, amikre szükségük van, szeretnének nézni/olvasni. Ez a probléma szülte az ajánlórendszer létrejöttét. [4] Például termékajánlás az is, amikor bemegyünk egy könyvesboltba és vannak könyvek, amelyek a borítójukkal ’felénk néznek’, de vannak olyanok is, amelyeknek csak a gerincét látjuk. Így akarja a bolt első sorban azokat a könyveket ajánlani, amelyeknek a borítóját látjuk. Az ajánlórendszert többféleképpen hozhatjuk létre, például tartalomalapú, vagy együttműködésen alapuló szűréssel.

Az 1. képen egy karikatúra látható a termékajánló rendszerekről. [5]

Az 1. kép nagyon jól szemlélteti, hogy miről szólnak ezek a rendszerek. A termékajánlást minden boltban megtalálhatjuk, mivel nem véletlenül vannak a csokik a kassza mellett vagy a tejtermékek közel a kenyérfélékhez, mivel ezzel a bolt üzenni akar, hogy nekünk arra is szükségünk van és sugallja, ezt még megérdemled.

A termékajánló rendszerek nagyon hasznosak a vállalatok számára, mivel ezáltal jobban tudják reklámozni a termékeiket és több emberhez eljutnak. Ugyanakkor a felhasználó szemszögéből is hasznos lehet, mivel nem kell órákig keresnie azt, amit igazán szeretne. Ha szállást foglal a termékajánló rendszer azonnal kiadja neki azt amelyiket nagy

valószínűséggel jóra értékelne, és nem kap sok felesleges, irreleváns ajánlatot és információt.

Az ajánlórendszerek előnyei az eladó szemszögéből:

- Növeli az eladott termékek számát – Ez a fő oka a termékajánló rendszerek létrejöttének, hogy a szolgáltató minél több terméket el tudjon adni. Ami annak köszönhető, hogy a rendszer feldob más árukat is a vevőnek, hogy ez is érdekelheti és lehet, hogy a vevő magától nem keresett volna rá az ajánlott termékre, de mivel a rendszer megmutatta, rájön, hogy szüksége van rá. Például ilyen, amikor a boltokban direkt olyan sorrendben rakják a termékeket, hogy az ember minden soron végig menjen és több dolgot megkívánjon.
- Segít több különböző terméket eladni – A termékajánló rendszer képes olyan ajánlatot adni a felhasználónak, amit a rendszer nélkül nehezen talált volna meg. Például egy filmes ajánlórendszerben lehet, hogy a felhasználó előnyben részesíti az újabb filmeket és nem tud a régiekről, de a rendszer megtalálja neki a régiek között is, ami érdekelheti és a felhasználónak eszébe se jutott volna ott keresni.
- Növeli a felhasználó elégedettségét és hűségét a termék/oldal iránt – Ha jól működik az ajánlórendszer akkor a felhasználó már csak azért is meglátogatja az oldalt, mert kíváncsi, hogy éppen mit ad fel neki a rendszer, ami érdekelheti. Ezt egy személyes példával tudom alátámasztani, én az IMDB oldalát nagyon gyakran szoktam nézni, hogy ajánljon nekem hasonló filmeket, amiket korábban néztem, néha még akkor is ránézek, amikor éppen nem is akarok filmet nézni, csak azért, hogy találjak valami újdonságot, érdekeset.
- Segít megérteni, mit akar a felhasználó – Például, ha a keresési előzmények alapján működik az ajánlórendszer, akkor a rendszer jobban emlékszik az előzményekre, mint a felhasználó, így sokkal könnyebben, jobbat tud ajánlani, mintha a felhasználó kezdett volna el keresgélni millió film között.

Az ajánlórendszerek előnyei a felhasználó/vásárló szemszögéből:

- Megtalálja az összes releváns terméket a felhasználó számára – Az ajánlórendszer az összes lehetséges ajánlatot megtalálja a felhasználónak és generál egy várható értékelést, amit a felhasználó adna rá.
- Csomag ajánlása – Például sok nagyobb városban megtalálható város nevével ellátott kártya, amivel a felhasználóknak kedvezményeket biztosítanak különböző múzeumokba, éttermekbe és sokszor kínálnak ingyenes tömegközlekedést is. Ez

azért jó, mert lehet, hogy az ember nem akar elmenni egy múzeumba, de látja, hogy a kártyával megéri meglátogatni, és ezáltal kihagyhatatlan élményben lesz része.

- Böngészheti a neki ajánlott termékeket – Ebben az esetben a felhasználó több ajánlatot is kap, de nem direktben mindenféle felszólítással, hanem böngészés közben a számára legideálisabb termékeket mutatja az oldal először.
- Másokat segít, illetve befolyásol – Az értékeléseket olvasva a felhasználók könnyebben fel tudják ismerni, ha egy termék nagyon jó vagy nagyon rossz minőségű, esetleg tudnak találni összehasonlításokat is a vélemények között. A probléma ezzel, hogy a felhasználó az összes másik felhasználó által írt véleményt olvashatja, nem csak azokét a felhasználókét, akik hasonlóak hozzájuk. [6]

Az ajánlást több más fontos tényező is befolyásolhatja, amik által javulhat, de akár romolhat is az ajánlórendszer pontossága, ezek a következő tényezők lehetnek:

- Sokszínűség – A felhasználó elégedettebb, ha az ajánlat, amit kap sokoldalúbb és több szempontot figyelembe vevő. Például, ha tetszett neki egy film egy bizonyos rendezőtől, akkor a rendszer nem csak annak a rendezőnek a műveit ajánlja neki, hanem más rendezőktől, hasonló filmeket is mutat.
- Ajánlás megtartása – Előfordul, hogy hasznos az ajánlást újból felmutatni a felhasználónak, nem pedig újakat mutatni, mert lehet, hogy elkerülte a figyelmét vagy éppen nem olyan hangulatban volt, hogy azt a filmet nézze vagy zenét hallgassa viszont, ha újra ajánljuk, akkor előfordulhat, hogy pár nap múlva már ahhoz lesz kedve. [7]
- Bizalom – Az ajánlórendszer nem ér semmit, ha a felhasználó nem bízik benne. Ez nagyon logikus dolog, mivel a felhasználó nem tudja, hogy a program honnan tudja róla, hogy mi tetszik neki, ezt úgy lehet kiküszöbölni, hogy a rendszer kiírja azt is, hogy miért ajánlotta az adott könyvet. Például jó, ha mutatja a weboldal, hogy megtekintetted az x terméket ezért y is tetszeni fog. [8]
- Manipuláció – A termékajánló rendszerek kezdeti időszakában volt jellemző, hogy a felhasználók is részt vehettek az ajánlat elkészítésében, de hamar rájöttek, hogy így sok csalást végbe lehet vinni, ha valaki téves információkat ír a véleményében, és így akár hátrányba tudja szorítani a konkurenciáját. [9] Például bevett marketing fogás, hogy a nagy cégek felbérelnek akár több száz embert, hogy írjanak pozitív véleményt egy termékükről így több vásárlóhoz juthatnak, vagy ugyanez fordítva is igaz lehet, hogy negatív véleményt írnak a vetélytárs termékéről.

- Túl egyértelmű ajánlások – Nagyban ronthat a pontosságon, ha az ajánlás túl egyértelmű, mert akkor a felhasználó egyből maximum pontot fog neki adni és így hamisan emeli a pontosságot. Nagyon jó példa erre, hogy valaki bemegy az élelmiszerboltba és a rendszer ajánlja neki, hogy vegyen kenyeret, persze ez nagyon jó ajánlás, mert tényleg fontos, hogy legyen kenyér, viszont ugyanakkor nagyon ajánlás is, mivel a felhasználó pont azért megy be a boltba, hogy kenyeret vegyen. [10]

A segédmátrix

A termékajánló rendszerek létrehozásánál nagy segítségünkre vannak a segédmátrixok, amik tartalmazzák, hogy különböző felhasználók, hogyan értékelték a különböző termékeket, és ezt használja alapul sok algoritmus, megkeresik a hasonló felhasználókat és összehasonlítják őket.

Mind a tartalom alapú, mind az együttműködésen alapuló rendszer egy segédmátrixot használ, amiben minden érték egy termék előnyének a mértékét jelzi a felhasználó számára. Például a Netflix és sok más oldal esetében a felhasználók egy 1-5-ig tartó skálán tudják jelezni, hogy mennyire tetszett nekik az adott film. Szóval a mátrix értékei 1 és 5 között fognak mozogni. [11]

Termékajánló rendszerek a gyakorlatban

Szinte az összes B2C elven működő nagyvállalat rendelkezik termékajánló rendszerrel, mint a Netflix, Spotify, YouTube, Amazon, Apple Music, Google, Booking.com, Expedia.

A Netflix egy olyan oldal, ahol körülbelül 17 ezer film és sorozat található. A felhasználónak nagyon nehéz dolga van, ha nincs egy konkrét film, amit meg szeretne nézni, hanem csak böngészne hátha talál valami olyat, ami tetszik neki. 17 ezer filmet végig nézni nem kis időbe telik a felhasználónak, így kitalálták, hogy létrehoznak egy termékajánló rendszert, ami különböző módokon ajánl filmeket ezzel megkönnyítve a felhasználók életét. Azért nagyon fontos a Netflixnek, hogy minél jobb ajánlásokat tudjon mutatni a felhasználóinak, mivel havonta kell érte fizetni és nincs hűségidő, bármikor le lehet mondani az előfizetést. Mivel már szinte bármilyen filmet le lehet tölteni az internetről, így a Netflixnek ez az egyetlen előnye a letöltéssel szemben, hogy ki tudja találni, mit szeretne a felhasználó és megmutatja neki. [12]

A Netflix weboldalán található egy leírás az ajánlórendszerükről. Azért, hogy a legjobb

ajánlást tudják adni a felhasználó számára a következő dolgokat használják fel: a felhasználó kapcsolatát a szolgáltatással, ajánlanak a megtekintési előzmények és a megtekintett értékelések alapján. Segítségükre van az az elv, hogy hasonló embereknek hasonló az ízlése, ezt hívják társadalom alapú szűrésnek. Továbbá felhasználják az adatokból, hogy mikor nézték a filmet, milyen eszközről, és mennyi ideig. Ezek az adatok, mind segítik az ajánlórendszer hatékonyságát. Viszont nem veszik figyelembe a demográfiai adatokat, mint például a kort és a nemet. Nem csak így segítik a megfelelő film megtalálását, hanem a keresővel is ajánlanak. Például, ha keresünk egy filmet, akkor nem csak az az egy fog megjelenni, hanem több ahhoz hasonló is. Amikor a felhasználó regisztrál, akkor opcionálisan kiválaszthat filmeket, amik tetszettek neki. Később az ajánlórendszer ezekhez a filmekhez hasonlókat fog ajánlani. Ha valaki nem választja ki ezeket a filmeket, akkor a rendszer a legnépszerűbb filmeket fogja neki feldobni az elején. Később pedig minden esetben tud alapozni a megtekintési előzményekre. A felhasználó ezeket az ajánlásokat megtekintheti a kezdőképernyőn, minden esetben, amikor belép az oldalra. A filmeket több sorban, balról jobbra rendezve találjuk, kezdve a legerősebb ajánlással a gyengébbekig. Ilyen sorok például a Videó folytatása, Népszerű filmek, Díjnyertes filmek, Hasonló filmek ahhoz, amit legutóbb néztél stb. [13]

2006-ban a Netflix egy nyílt versenyt hirdetett ki, azzal a céllal, hogy megtalálják a legjobb algoritmust a vélemények, értékelések alapján, úgy, hogy nincs információjuk a filmről vagy a felhasználóról. Az induló csapatok kaptak egy három összetevőből álló adathalmazt, ami tartalmazta a felhasználót, a filmet, és az értékelés dátumát. Viszont a zsűri tudott egy negyediket is, amit a csapatoknak kellett kitalálnia a lehető legnagyobb pontossággal, ez volt maga az értékelés, amit egy 1-5 -ig skálán tudtak megtenni. [14]

A lényeg az volt, hogy a lehető legkisebb eltérés legyen az eredményük és a validációs adatok között. A győztes csapat, BellKor’s Pragmatic Chaos névvel 0,8567-es pontosságot ért el, ezzel 10,06%-ot javított a Netflix ajánlórendszerén. Az eredménytáblán látszik, hogy az eredmények között csak ezred vagy tízezred különbségek vannak és azok is nagyon sokat számítanak. Érdekesség, hogy a 14. helyen lévő Gravity néven futó csapat magyar informatikusokból állt, akik 2007 januárjától 2007 májusáig az első helyen álltak. És ma már termékajánló rendszerekkel foglalkozó vállalkozást indítottak. [3]

Termékajánló módszerek

Kollaboratív ajánlás

A kollaboratív ajánlásokat az első termékajánló rendszerek között tartják számon, amit a GroupLens nevezetű csoport talált ki először. A GroupLens Minnesotai Egyetem egy csoportja, akik úttörőnek számítanak az ajánlórendszerek területén. [15] A kollaboratív ajánlási módszerek minden egyes felhasználónak egyedi ajánlást készít, a felhasználó érékelései, véleményei és korábbi vásárlásai alapján. Ehhez nem kell külső információ, mint például kor, lakhely sem a felhasználóról, sem a termékről. [6] A kollaboratív ajánlás alapja, hogy a hasonló felhasználóknak azonos az érdeklődési körük és hasonló termékeket vásárolnak. A kollaboratív ajánlási rendszerek úgy működnek, hogy felépítenek egy adatbázist, ami felhasználó-termék mátrixba rendezi a termékeket és felhasználókat. Ahogy a különböző algoritmusok végig járnak ezen a mátrixok összegyűjtik a hasonló profillal rendelkező felhasználókat és egy csoportba helyezi őket, amit szomszédoknak neveznek. Egy felhasználó arról a termékről fog ajánlást kapni, amelyet ő még nem értékelt vagy nem látott viszont a csoportjában már valaki értékelte és tetszett neki, így valószínűleg a felhasználónak is tetszeni fog. [16]

	Film 1	**Film 2**	**Film 3**
Felhasználó 1	5	3	1
Felhasználó 2	2	1	4
Felhasználó 3	?	3	2

A 2. képen a felhasználó-termék mátrix bemutatása.

A 2. képen látszik, hogy a *Felhasználó 1* és a *Felhasználó 3* a hasonló filmeket szereti, így őket egy kategóriába sorolja az algoritmus, azaz szomszédok lesznek. Így már könnyedén ki lehet találni, hogy a **Film 1**-et hogyan értékelné a *Felhasználó 3* és van-e értelme ajánlani neki azt a filmet.

Két véleményfajtát különböztetünk meg:

- Explicit értékelés: Azt mutatja meg közvetlenül, hogy a felhasználó hogyan értékelte az adott terméket (ez lehet film, könyv, kávézó stb.). Ez egy egyértelmű vélemény, biztosan tudjuk, hogy a felhasználó, mit gondol a termékről.

- Implicit értékelés: Nevezhetjük ezt ’sütinek’ is, ez szolgáltatja az információt arról, hogy mire kattintott rá, milyen oldalt/terméket nézett meg a felhasználó, mire keresett rá, mit

kedvelt a közösségi médiában, mennyi időt töltött egy oldalon. Ez csak helyettesként szolgál és heurisztikát nyújt arról, hogy a felhasználó mennyire kedvel egy terméket. Mondhatjuk, hogy ez egy közvetett vélemény. Például van egy zeneszám, amit a felhasználó csak egyszer hallgat meg és nem ad róla véleményt. Ilyenkor lép életbe az implicit véleményalkotás, mivel nem tudhatjuk, hogy azért hallgatta egyszer mert éppen nem volt több ideje vagy azért, mert nem tetszett neki. [6]

- Hibrid értékelés: Az explicit, és az implicit vélemények keveréke, ami tartalmazza mindkettő erősségeit, és csökkenti a gyengeségeiket, hogy egy még jobb rendszer jöhessen létre. Ezt úgy érjük el, hogy ha implicit adatokat használunk, akkor ellenőrizzük az explicit véleményeket is. [16]

Összevetve a kettőt az explicit vélemény sokkal egyszerűbb és egyértelműbb, mint az implicit, mivel nem kell kitalálnunk, hogy mit szeret a felhasználó, hanem ő maga megmondja, míg implicit vélemény esetében csak sejtésünk lehet.

Szomszéd alapú megközelítés

A legkorábban fejlesztett kollaboratív algoritmusok között találhatóak. Ezeknek az algoritmusoknak az az alapja, hogy a hasonló felhasználók, hasonlóan értékelnek és a hasonló termékek hasonló értékeléseket kapnak.

Felhasználó alapú modellek

Ez az úgynevezett „Hozzád hasonló felhasználók ezt a terméket is kedvelték.” Ebben az esetben az algoritmusok megkeresik a célfelhasználóhoz hasonló felhasználókat és megvizsgálja, hogy a hasonló felhasználók által jóra értékelt termékek közül, melyiket nem értékelte még a célfelhasználó és ezeket a termékeket fogja a rendszer ajánlani neki. [15]

Pearson – féle korreláció

A Pearson korrelációt Karl Pearson, brit matematikus alkotta. A Pearson – féle korrelációt, akkor használjuk, ha meg szeretnénk találni a lineáris kapcsolatot kettő elem között. Az értéke negatív 1 és pozitív 1 között mozoghat.

Ez az egyik mérték, amivel meg tudjuk határozni két elem közötti hasonlóságot. Először ki kell számolni az átlagát mindegyik felhasználónak, majd utána használhatjuk a korrelációt:

$$U_{ik} = \frac{\sum_j (v_{ij} - v_i)(v_{kj} - v_k)}{\sqrt{\sum_j (v_{ij} - v_i)^2 \sum_{kj} (v_{kj} - v_k)^2}}$$

Az egyenletben az u jelenti a felhasználókat, v az értékeléseket, az i és k különbözteti meg a két felhasználót, j jelöli az értékelt terméket. [17]

Termék alapú modellek

Ezt a módszer úgy lehet egyszerűen megfogalmazni, hogy „Felhasználók, akik ezt a terméket kedvelték, ezt a terméket is kedvelték.” Ez a fajta szűrő nem a felhasználók közötti hasonlóságot nézi, hanem a termékek közöttit, amiket a felhasználó értékelt. A hasonlóságot ebben az esetben is a fent említett két módon tudjuk kiszámítani. A fő különbség a felhasználó és a termék alapú szűrők között, hogy a termék-alapú szűrők függőlegesen töltik ki a sorokat a táblázatban, amíg a felhasználó alapúak vízszintesen. [6]

Cosinus hasonlóság

Az úgynevezett nyers értékeléseknél használják ezt a számolási módszert. Általában ezt a számolást nem a hasonlóan értékelt termékekre végzik, hanem az összes lehetséges elemet megvizsgálják vele. [15] Ebben az esetben i és j a két termék, amik között a hasonlóságot keressük, u a felhasználó és s az értékelés.

$$AdjustedCosine(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i} s_{uk}^2} \sqrt{\sum_{u \in U_j} s_{uj}^2}}$$

Tartalom alapú termékajánló rendszerek

„Ha van 4,5 millió vásárlónk, nem elég, ha egy boltunk van. Akkor 4,5 millió bolt kell.”

Mondta Jeff Bezos az Amazon alapítója a Washington Postnak. [18]

A tartalom alapú ajánló rendszereknek ez az alapja. Valóban igaz, hogy minden felhasználónak más az érdeklődése, például egy fiatal anyuka baba játékokat keres és nem veszi jó néven, ha csak a programozást ismertető könyveket találja, ezért nagy eséllyel legközelebb nem fogja meglátogatni az oldalt.

A tartalom alapú rendszer megtanulja, hogy olyan elemeket ajánljon, amelyek hasonlóak, mint amiket a felhasználó a múltban kedvelt. Az elemek hasonlóságának kiszámolása, úgy történik, hogy a rendszer összehasonlítja a két elem jellemzőit. Például, ha a felhasználó pozitívan értékel egy filmet, ami a komédia műfajba tartozik, akkor az ajánlórendszer meg tudja tanulni, hogy milyen más filmeket tud ajánlani ugyanebben a kategóriában.

Az ajánlási folyamat három lépésben történik, melyek mindegyikét külön komponens kezeli.

Tartalomelemző: Ha az információnak nincs megfelelő felépítése, szükséges egy előfeldolgozási lépés, hogy ki tudjuk szűrni a nekünk fontos információt. Ez a lépés fontos, hogy a következő lépésekben olyan információt tudjunk továbbadni, amivel könnyebben lehet dolgozni. Ez szolgáltatja a bemenetet a következő két lépéshez.

Profilelemző: Ez a modul összegyűjti a felhasználó által előnyben részesített adatokat, és általánosítja ezeket, a felhasználó profiljának felépítésének érdekében. Általában ennek a realizációnak az alapja a gépi tanulás, ami képes arra, hogy felállítsa a felhasználó érdeklődési körét arra támaszkodva, hogy ismeri a felhasználó múltjából kedvelt/nem kedvelt termékeket, és tárolja a felhasználó korábbi visszajelzéseit.
Szűrő komponens: Ez a modul hasznosítja a felhasználó profilját, ahhoz, hogy megtalálja a megfelelő terméket, azzal, hogy összehasonlítja a profilon található elemeket az ajánlandó elemmel. Ha az elem megfelelő, akkor hozzáadja egy lehetséges érdeklődési listához. [6]

Előnyök és hátrányok

Előnyei:
- Felhasználói függetlenség: Csak azokat az ajánlásokat használja, amiket a felhasználó profilja biztosít, mivel ezek a saját előzményei alapján történnek nem pedig a Kollaboratív ajánlás esetében megismert hasonló felhasználók alapján.
- Átlátszóság: Egyértelműen fel tudja sorolni a termékek jellemzőit vagy leírásukat az ajánlások listájáról. Ezek a jellemzők alapján tudjuk eldönteni, hogy megéri-e ajánlani a terméket. Amíg ez a Kollaboratív ajánlás esetében egy fekete doboz, mivel csak a hasonló felhasználókat ismerjük.
- Új termék: Képes olyan terméket is ajánlani, amit még előtte senki nem értékelt.
Hátrányai:
- Korlátozott a tartalom-elemzés: Ahhoz, hogy hatékonyan működjön a rendszer ismerni kell a termék bizonyos jellemzőit, például film esetében a rendezőt, színészeket. Fontos, hogy legyen elég információnk.
- Túlműködés: Ez azt jelenti, hogy a rendszer csak hasonló értékeléseket keres és emiatt nehéz az újakat megtalálni.
- Új felhasználó: Ahhoz, hogy el tudjuk készíteni az ajánlást először kellenek előzmények, viszont egy új felhasználó esetében nincsenek még értékelések, szóval a rendszer még nem képes megbízható javaslatot létrehozni. [6]

Tudás alapú termékajánló rendszerek

A tudás alapú ajánlórendszerekkel a nem mindennapi termékeket szokás ajánlani, hanem ha az ember valami értékesebbet szeretne vásárolni. Például autót, utazást, lakást vagy luxuscikkeket. Amiknél nagyon fontos, hogy az ajánlórendszer jól ismerje a vevőt, hogy

tudjon neki megfelelő termékeket ajánlani. A felhasználó, ha egy házat vesz sokkal jobban körül kell járnia a témát, mint egy film megnézésénél, aminek nem lehet semmi anyagi következménye, hogy megnézte-e vagy sem. Viszont, ha vesz egy kocsit, ami fél év múlva már nem indul el, akkor annak nagyon rossz következményei lehetnek. Általában, ha az ember ilyen terméket vesz, nagyobb eséllyel ír véleményt a termékről, vagy az ajánlásról, mint egy film esetében. Ezért, ha úgy nézzük nincs nehéz dolga az ajánlórendszernek, mivel explicit értékeléseket tudnak használni. Ha a felhasználó konkrétan leírja, hogy milyen terméket szeretne, ezért a tudás alapú ajánlórendszereknek az interaktivitás kulcsfontosságú eleme. Nehéz egyértelműnek venni az értékeléseket, mivel általában nagyon összetett termékekről van szó. Például egy autó esetében előfordulhat az, hogy egy a felhasználó számára minden paraméter megfelelő, kivéve a színe. Ilyen esetben nehéz eldönteni, hogy ajánlhatjuk-e ezt az autót vagy sem. Azaz kompromisszum képes-e a felhasználó. Még egy probléma a tudás alapú ajánlásokkal, hogy ezek a rendszerek idő-érzékenyek, azaz, ha van egy 10 éves laptop, amiről nagyon jó vélemények vannak, az nem jelenti azt, hogy ez a laptop még mai szemmel is ugyanolyan jónak számít, mint 10 évvel ezelőtt. [15]
A tudás alapú módszerek két fő csoportba sorolhatók aszerint, hogy milyen interakciót használnak, ezek a kényszer alapú és az eset alapú ajánlórendszerek.

Kényszer alapú ajánlórendszer

A kényszer alapú ajánlórendszerek megengedik a felhasználónak, hogy specifikálja az igényeit, szóval egy ilyen ajánlás több szűrőt igényel. Nehézsége, hogy a felhasználó sokszor nem ugyanazokat a szűrőket emeli ki, mint amik a rendszerben megtalálhatóak, így az algoritmusoknak ez egy plusz lépés, hogy hasonló formára hozza a szűrőket. Egy ház ajánlásnál a következő szűrőket választhatja ki a felhasználó: családiállapot, család mérete, város vagy falu, minimum hálószoba, maximum érték, négyzetméter stb. Ha ezek közül párat megad a felhasználó, vagy implicit módon lehet következtetni, akkor abból a többit a rendszer ki tudja következtetni. Például, ha a felhasználó megadta, hogy egyedülálló, akkor az ajánlórendszer valószínűleg egy kisebb házat fog ajánlani, mint egy 3 gyerekes családnak. [15]

Eset alapú ajánlórendszer

Eset alapú ajánlórendszerek esetében keresnek hasonló eseteket és azokat veszik alapul az ajánlás elkészítéséhez. Hasonlóan a kényszer alapú ajánlásokhoz. Ennél a módszernél is

szűrőkre van szüksége az algoritmusnak, hogy ajánlást tudjon készíteni. Csak ennél a módszernél nincsenek annyira kötött értékek, mint például a szobák száma, hanem itt előre legyártott szűrők vannak. Ha az algoritmus nem talál egy olyan terméket, ami a felhasználó minden igényét kielégíti, akkor keres egy hasonlót és azt ajánlja. Itt nem probléma, ha a felhasználó és a rendszer nem pontosan ugyanazokat a szűrőket alkalmazzák. [15]

Az első tudás alapú ajánlórendszer az Entree Chicago, amit a chicagói egyetem informatikusai fejlesztettek 1996-ban és 1997-ben. Az oldalon étterem ajánlásokat lehetett kérni kettő féle módon. Az első verzió a kényszer alapú ajánlást használta, azaz több paraméterre rákérdezett, étel típusa, árak, stílus, légkör, események és ezek alapján keresett egy éttermet, ami a lehető legtöbb kérésnek eleget tudott tenni. A másik verzió pedig az eset alapú ajánlást használta, aminél csak egy éttermet kér és egy várost és ehhez az étteremhez hasonlót keresett az adott városban. Majd a találatokat szépen kilistázta és a felhasználó ki tudta választani a neki legjobban tetszőt. [19]

Társadalom alapú termékajánló rendszerek

Ez a fajtája a termékajánló rendszereknek a felhasználó barátaira építi az ajánlásait. Követi a jól ismert mondatot: „Mondd meg kik a barátaid, és én megmondom ki vagy.” Ez a rendszer felhasználja az embereknek azt a tulajdonságát, hogy jobban adnak a barátaik véleményére, mint idegen emberek által írt értékelésekre. Az ajánlórendszert nagyban segíti a közösségi oldalak virágzása, ahol az emberek akarva, akaratlanul is látják, ha egy ismerősük értékelt valamit. Mindenkivel így van, ha rákeres egy termékre valamelyik közösségi platformon és látja, hogy egy ismerőse jó véleményt írt róla, akkor szívesebben veszi meg, mintha idegenek véleményeit olvasgatná. [6]

Demográfiai termékajánló rendszerek

Ez a típusa az ajánló rendszereknek a felhasználó demográfiai alapjain nyugszik. Azt feltételezi, hogy különböző ajánlásokat kell gyártani a különböző demográfiai adatokkal rendelkező felhasználóknak. Például ilyen demográfiai adat az életkor, lakhely, nyelv. A marketingben ez a fajta megközelítés nagyon népszerű, viszont a termékajánló rendszerek esetében már nem örvend akkora népszerűségnek. [6]

A marketing esetében úgy figyelhető meg, hogy több üzletben megkérdezik a vásárlóktól az irányítószámukat. Azért, kérdezik mert így fogják tudni, hogy melyik irányítószámú faluba, városrészre milyen újságot érdemes küldeni. Például észrevették, hogy akik a belvárosban

laknak azok kisebb valószínűséggel fognak fűnyírót vásárolni, mint a falusiak, így nekik nincs is értelme bedobni a prospektust az új fűnyírókról. Amíg régen erre nem figyeltek, hanem mindenki kapott mindenféle szóróanyagot, aminek az emberek nem örültek, nagyon sok szemét keletkezett vele és még a boltoknak is plusz kiadást jelentett.

Gépi tanulás alapú módszerek

A gépi tanulást, mint fogalmat először Arthur Samuel írta le 1959-ben, amit a következőképpen definiált: *„A tudomány azon területe, ami lehetővé teszi, hogy a számítógépek programozás nélkül tanuljanak”.* [20]

A gépi tanulás a mesterséges intelligenciának egy olyan ága, ami az önállóan tanulni képes rendszereket foglalja magába, ez a tanulás egy olyan algoritmust követ, ami becsléseket készít már meglévő adathalmazból. Például gyakran alkalmazzák szakértői rendszerek létrehozásakor, illetve az adatbányászat területén is. [21]

Felügyelet nélküli tanulás

A felügyelet nélküli tanulást, akkor célszerű használni, amikor nagyon nagy az adatkészletünk, ami még jelöletlen adatokat tartalmaz. Ebben az esetben az adatok megértéséhez olyan algoritmusokat használhatunk, amelyek képesek megtalálni az adatok közötti mintákat és ellátják őket címkékkel. Lényegében lehetővé teszi, hogy később felügyelt tanulást lehessen használni. Miután megtörténik a címkézés a felügyelt tanuláshoz hasonlóan a felügyelet nélküli tanulás is mintákat keres.

A kettő metódus között csak annyi a különbség, hogy a felügyelet nélküli tanuló algoritmusok nem értik az adatokat. Sok esetben a felügyelet nélküli tanulás gyorsabban képes, pontosabb eredményt megadni, mint a felügyelt tanulás, mivel maga az algoritmus építi fel a címkéket és így könnyebben tud vele dolgozni, mintha meg kéne értenie a minták elhelyezkedését.

Például ezeket az algoritmusokat használják a spam felismerő rendszerek is, mivel amikor beérkezik egy e-mail nincs azonnal címkével ellátva, hanem kell egy algoritmus, ami fel tudja címkézni, hogy spam vagy nem spam az adott levél. [21]

Felügyelt tanulás

A felügyelt tanulás egy adatkészlettel dolgozik, amiről tudjuk, hogy milyen osztálycímkékkel vannak ellátva benne lévő adatok. Minden adathoz tartozik egy osztálycímke. A felügyelt tanulás lényege, hogy megtalálja, hogy a még nem ismert példákhoz milyen osztálycímke rendelhető a különböző minták alapján. Ezeket a mintákat jellemzőkkel látjuk el, amikből következtethetünk egy még ismeretlen minta osztályára. [22]

Például vegyünk egy tanuló adathalmazt, ami zenéket tartalmaz. Ezt felruházzuk jellemzőkkel, lehet ez műfaj, ritmus, nyelv, kiadás éve, jelölések. Ezekhez rendelünk egy-egy címkét, amivel megnézzük, hogy egy adott személynek tetszettek-e az adathalmazban található számok vagy sem. Ezek után vizsgáljuk a teszt adathalmazban található zenéket és összehasonlítjuk a jellemzőiket a tanuló adathalmazban található zenék jellemzőivel és ezalapján tudunk ezekhez is címkéket rendelni a tanuló adathalmazban lévő zenékhez, amik egy becslést adnak.

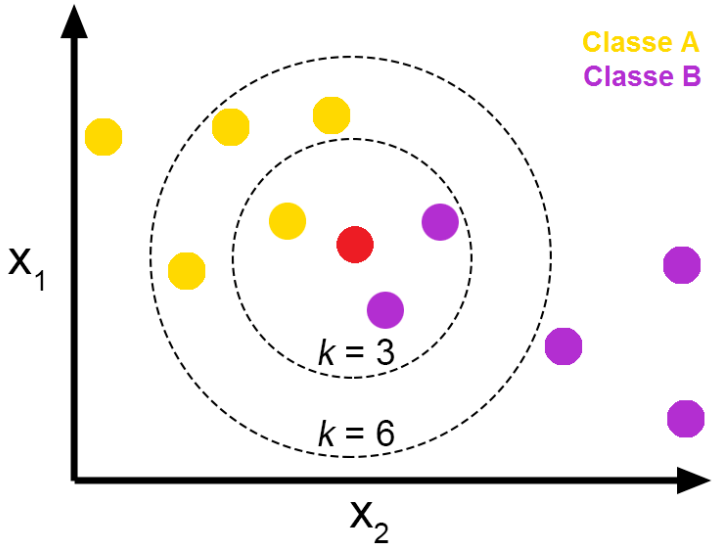
A felügyelt tanulásnak kettő verziója létezik. Amikor a címkék folyamatosak, akkor regressziónak hívjuk. Amikor az adatok egy véges készletnek az értékei, akkor pedig klasszifikációt használunk.

Számos probléma megoldására használják a felügyelt tanulási módszereket, mint például a beszéd felismerés, kockázat elemzés és amire én is használni fogom, ajánló rendszerek. [21]

Nearest Neighbors

A közeli szomszédok egy felügyelt tanuló algoritmus, így az elemei címkékkel vannak ellátva. Nagyon egyszerű logikára épül az algoritmus. Rendelkezésünkre áll egy nagy adatkészlet, aminek az elemei címkézve vannak, ez lesz a tanuló adatkészlet.

Az adatkészletet képzelhetjük sok kicsi pontnak, amik nagyjából címkék szerint vannak rendezve. Kapunk egy új adatot, aminek nincs még címkéje, de el kell róla döntenünk, hogy melyik csoportba tartozik. Az algoritmusnak van egy argumentuma, a k, azt határozza meg, hogy hány közeli szomszédot vizsgálunk. Az algoritmus úgy dönti el, hogy milyen címkével lássa el az új elemet, hogy megkeresi a k db legközelebbi elemet és összegzi, hogy melyik címkéből mennyi van, az új adatot pedig felruházza azzal a címkével, amelyik legtöbbször előfordul a k darab szomszéd között. [23]



A 3. kép a legközelebbi szomszéd algoritmust szemlélteti. [24]

A 3. képen látszik, hogy sokat tud változtatni az eredményen, ha kisebb vagy nagyobb k-t választunk. A képen, ha k egyenlő hárommal, akkor egyértelműen a lila csoportba fog tartozni a piros pötty, de ha k-t hatnak vesszük, akkor már torzulhat és a sárga címke lesz az erősebb.

Az eredményt befolyásolhatja, hogy milyen számítást alkalmazunk a szomszédok kereséséhez, azaz, milyen módszerrel számoljuk ki a távolságokat.

Decision Tree

Ez az algoritmus az egyik legrégebbi és legerősebb algoritmus. Különlegessége, hogy nemlineáris döntéseket tudunk vele hozni lineáris felületen. A döntési fák olyan nem paraméteres felügyelt tanulási metódusok, amelyeket klasszifikációra és regresszióra használhatunk. A cél egy olyan modell létrehozása, ami megjósolja egy célváltozó értékét azáltal, hogy megtanulja az egyszerű döntéseket az adatok jellemzőiből. A döntési fa algoritmust, úgy kell elképzelni, mint egy családfát, aminek van egy törzse és onnan ágaznak szét a további ágak, levelek. Azért is hívják döntési fának, mivel van egy alap állítás és az algoritmusnak el kell döntenie, hogy az állítás igaz-e vagy sem és ettől függően lép tovább a következő ágra, ahol újabb állítást talál, és ez így megy addig, amíg el nem éri az algoritmus az utolsó levelet.

Ajánlórendszereknél úgy lehet ezt elképzelni, hogy van egy felhasználó, akinek szeretnénk ajánlani egy olyan filmet, amit még nem látott, de valószínűleg tetszene neki. Az algoritmus kérdéseket tesz fel magának, mint látta már a felhasználó ezt a filmet, tetszenek neki az ehhez hasonló filmek stb. És ha mindegyik kérdésre igennel tud felelni az algoritmus akkor ajánlani fogja a felhasználónak a filmet. [25]

Előnyei: könnyű megérteni és interpretálni. A fákat könnyű vizualizálni. Képes több-kimenetes problémákat megoldására. Jobban kezeli a kiugró értékeket, mint az SVM.

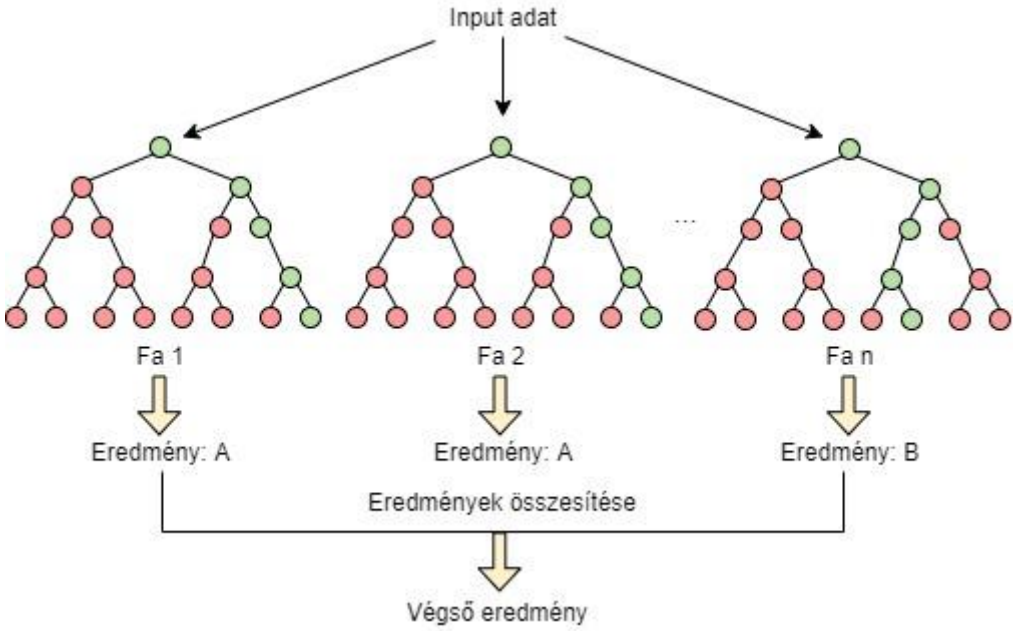
Hátrányai: létrejöhet egy olyan jelenség, aminek overfitting (túltanulás) a neve. Ez úgy jön létre, hogy az algoritmus túl mélyen beleássa magát a döntésekbe és felesleges lépéseket tesz, amik már nem segítik a becslést.

A működésének egy fa az alapja, ami döntések alapján választja szét az adatokat. Akkor célszerű ezt használni, amikor nem tudunk egy egyszerű vektort rakni az adatainkra, mivel nem helyezkednek el olyan szabályosan. Ilyenkor kérdéseket kell feltenni és a válaszok alapján felosztani az adatokat. Így több halmaz jön létre. Például adott egy olyan adatkészlet, aminek a jobb felső sarkában vannak egyfajta értékek a többi részében pedig egy másik fajta, ilyenkor célszerű először mondjuk az x-tengelyen elválasztani őket egy kérdéssel, hogy egy kiválasztott pont fölött vagy alatt vannak-e, majd ugyanezt megcsinálni az y-tengellyel is. [26]

Random Forest

A Random Forest Classifier kitalálása Leo Breiman, a Berkeley egyetem professzorához köthető, észrevételeit a Random Forests című publikációjában írta le 2001-ben. [27]

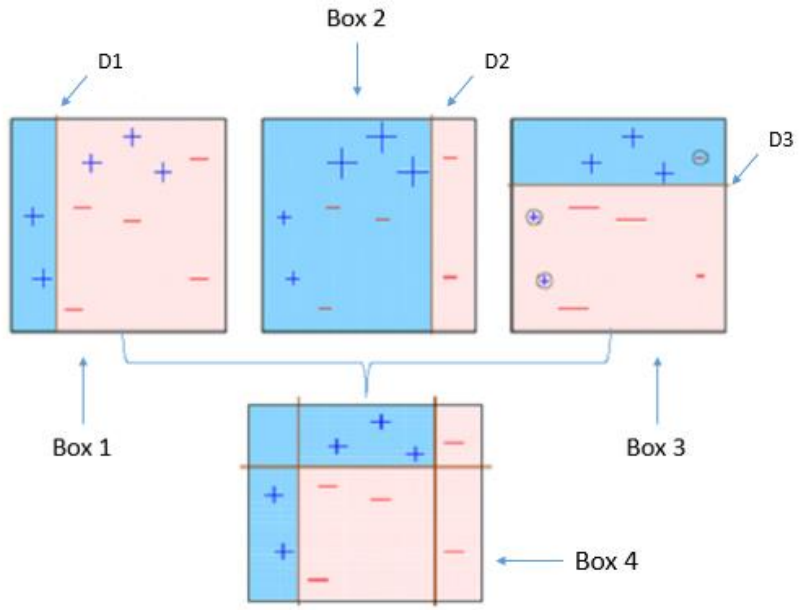
Ez egy olyan átlagolási algoritmus, amely véletlenszerű döntési fákon alapul. Ez azt jelenti, hogy sokféle osztályozót hoznak létre a véletlenszerűség bevezetésével az osztályozó konstrukcióban. Az együttes előrejelzése az egyes osztályozók átlagolt előrejelzése. Olyan adatbecslés, amely számos döntési fa osztályozóhoz illeszkedik az adatminták különböző részmintáihoz, és átlagolással használja a prediktív pontosságot és a túlszabályozás ellenőrzését. Az alminták mérete mindig megegyezik az eredeti beviteli minta méretével. Amikor hívásra kerül az algoritmus, bele kell írni, hogy hány darab fát szeretnénk benne használni, ezt jelenti a kódban látható ***n_estimators=100*** paraméter**.** A száz lett az új alapértelmezett értéke, viszont ezt lehet változtatni, amekkorára szeretnénk, hasonlóan, mint a döntési fáknál. [28]



A 4. képen a Random Forest algoritmus felépítése látható. [29] Alapján készült.

AdaBoost

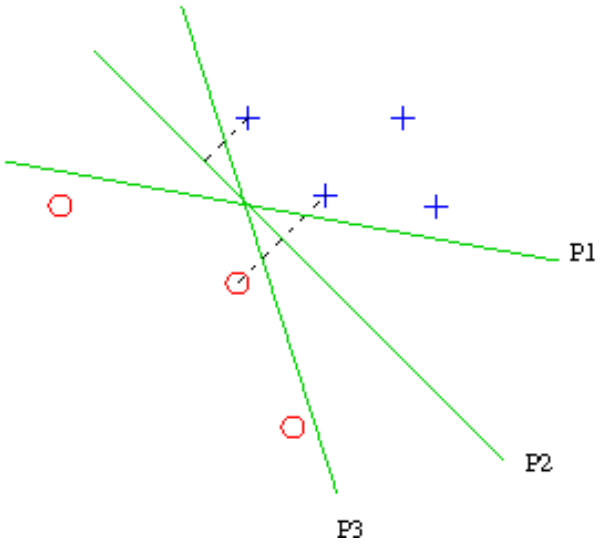
Az AdaBoost legfontosabb alapelve az, hogy gyenge tanulók sorozata, vagyis olyan modellek, amelyek csak kismértékben jobbak, mint a véletlenszerű találgatás. Mindegyikük előrejelzéseit súlyozott többségi szavazással kombinálják a végső előrejelzés elkészítéséhez. Minden egymást követő iterációnál a minta súlyait egyedileg módosítják, és a tanulási algoritmust újra alkalmazzák az újra számolt adatokra. Egy adott lépésnél azokat a képzési példákat, amelyeket az előző lépésben indukált modell által helytelenül előre jelöltek, súlyuk megnövekedett, míg a súlyok csökkentek a korábban előre jelzetteknél. Az iterációk folytatásaként a nehezen megjósolható példák egyre növekvő befolyással bírnak. Minden későbbi gyenge tanuló kénytelen koncentrálni a példákra, amelyeket a sorozatban az előzők nem fogadtak el. A tökéletes illeszkedés esetén a tanulási folyamat korai szakaszban leáll. [30]



Az 5. képen az AdaBoost algoritmus működési elve látható. [31]

Support Vector Machine

Rövidebb nevén SVM. A Support Vector Machine olyan felügyelt tanulási módszereket takar, amiket osztályozáshoz, regresszióhoz vagy kiugró érték felfedezéséhez használhatunk. Több dimenziós terekben is hatékonyan működik, illetve akkor is, ha több a dimenziók száma, mint a példák száma. A döntési funkciók egy részhalmazát vizsgálja, így nincs szüksége annyi memóriára. Több fajta klasszifikációja létezik. A működésére a nevéből is lehet következtetni, egy vektoron alapszik. Az algoritmus létrehoz egy vektort, amivel két részre bontja az adatokat, ezt nevezi hyperplane-nek (hipersík). Egy adatkészletben a hyperplane-t nagyon sokféleképpen lehet elhelyezni, ha csak azt vesszük figyelembe, hogy ketté bontsuk az adatkészletet. Ezért találták ki, hogy legyen egy margin (margó) nevezetű 'támasz'. Ennek az a lényege, hogy mindegyik oldalról a legközelebbi pontoknak a lehető legnagyobb távolságban kell lenniük. [32] Az ábrán láthatók a különböző lehetséges vektorok és a pontok, amik között a legjobb elválasztó vektort keressük.



A 6. ábra az SVM algoritmus működését szemlélteti. [33]

A 6. ábra nagyon jól mutat három lehetséges vektort is, de látszik, hogy a P1-es és a P3-as nagyon közel található az adatokhoz, ez zajokat okozhat a későbbiekben. Viszont a P2-es vektor az egy nagyon jó hipersík, mivel az található legtávolabb a két legközelebbi ponttól.

Artificial Neural Networks

A mesterséges neurális hálók az emberi agy működése alapján dolgozik, ahol szintén megfigyelhetőek a hálózatok, amiken keresztül az információ eljut egyik helyről a másikra. Minden neurális hálózat felépítési alapjai a neuronok, ezek általában egyszerű matematikai

függvények, amik szorzatokból és összegekből állnak. Ha kettő ilyen neuront összekötünk, akkor már van egy neurális hálózatunk. Ezeket a hálózatokat sokféleképpen fel lehet építeni, például lehet előrecsatolt, de léteznek olyan hálók is, amiknek vannak visszacsatolt elemei is. [34]

Az adatok

Az adatkészletet, amivel dolgoztam a Kaggle.com oldalon találtam, ahol prediktív és elemzési versenyek, kihívások találhatóak. A kiválasztott adatkészlet az Expedia szálláskereső oldal által feltett verseny adatai. A kihívásnak már vége van, viszont az eredmények továbbra is elérhetőek, és az oldal továbbra is kiértékeli az általam feltöltött eredményeket. A kihívás célja, az Expedia szálláskereső oldal segítése, hogy minél jobb, személyreszabott ajánlatokat tudjon adni a felhasználóinak. Az oldal a kihívás készítésekor keresési paramétereket használt a szállás ajánlásokhoz, de nincs elegendő ügyfél specifikus adatuk ahhoz, hogy minden felhasználó egyedi ajánlást kaphasson. Azért hozták létre a kihívást, hogy segítségül hívják a tapasztalt informatikusokat, jósolják meg, hogy a felhasználó a 100 különböző szállodai csoportból melyiket fogja választani. Az adatok az Expedia véletlenszerűen kiválasztott adatai és nem reprezentálja az általános összesített statisztikát. [35]

Az eredményeket a következő képlettel számolják, ahol |U| a felhasználó előfordulásainak száma, P(k) a pontosság a k-nál, n a becsült szállodai klaszterek száma, ami jelen esetben maximum 5 lehet. Ez a képlet nagyon fontos volt a becslésem során, mivel ez alapján döntöttem úgy, hogy milyen módszerrel végzem el a becsléseket.

$$MAP@5 = \frac{1}{|U|}\sum_{u=1}^{|U|} \cdot \sum_{k=1}^{min(5,n)} P(k)$$

A feltöltendő eredményfájlra komoly formai megkötéseket adtak meg. Minden felhasználónál maximum öt darab becslést lehet feltüntetni, szóközzel elválasztva, ezek alapján fog készülni a feljebb látható kiértékelés. A fájlnak 2528243 sort kell tartalmaznia, amiből az első a fejléc. [36] Amikor először töltöttem fel az eredményem erről még nem tudtam, így az első pár feltöltésem sikertelen lett, mivel nem volt pontos a sorok száma és nem tartalmazta a fejlécet.

id	*hotel_cluster*
0	91 19 32 63 27
1	16 7 29 4 15
2	50 18 41 27 47
3	21 40 43 88 65

A 7. kép tartalma az elvárt eredményfájl formája.

Az oldalon négy .csv kiterjesztésű, vesszővel tagolt fájlt lehet letölteni, a tanuló adatokat, a teszt adatokat, egy mintát arra, hogyan kell az eredményfájlnak kinéznie, és egyet, ami a szállodakeresés rejtett attribútumait tartalmazza, amiket az értékelési szövegekből szűrtek ki, de én az utóbbi kettőt nem használtam.

A tanuló és a teszt adatok dátum alapján vannak szétválasztva, a tanuló adatok tartalmazzák a 2013 és 2014-es adatokat, míg a teszt adatokban 2015-ös adatokat találhatunk. A kiértékelés is két adathalmazon történik, ami szintén évek szerint van elválasztva.

A tanuló adatkészlet 24 oszlopot és 37670293 sort tartalmaz, amíg a teszt adatkészlet 22 oszlopot és 2528243 sort.

date_time	időbélyeget tartalmazó oszlop
site_name	a keresett weboldal azonosítóját tartalmazó oszlop (pl. Expedia.com)
posa_continent	a *site_name* kontinensét tartalmazó azonosító oszlop
user_location_country	a felhasználó ország azonosítóját tartalmazó oszlop
user_location_region	a felhasználó régió azonosítóját tartalmazó oszlop
user_location_city	a felhasználó város azonosítóját tartalmazó oszlop
orig_destination_distance	fizikai távolság a keresett hotel és a felhasználó között, az keresés idejében, ha nulla értéket vesz fel, az azt jelenti, hogy a távolság nem kalkulálható
user_id	a felhasználó azonosítóját tartalmazó oszlop
is_mobile	1 értéket vesz fel, ha a felhasználó mobil eszközről keres, 0 az egyéb esetekben
is_package	1 értéket vesz fel, ha a felhasználó egy csomagban keres (pl. repjeggyel, autófoglalással együtt), 0 különben
channel	a marketing csatorna azonosítóját tartalmazó oszlop
srch_ci	a keresett bejelentkezési időpont
srch_co	a keresett kijelentkezési időpont
srch_adults_cnt	a keresett felnőttek száma szobánként
srch_children_cnt	a keresett gyerekek száma szobánként
srch_rm_cnt	keresett szoba mennyiségek száma
srch_destination_id	a helyszín azonosítója, ahonnan a hotelt keresték
srch_destination_type_id	a helyszín típusának az azonosítóját tartalmazó oszlop

hotel_continent	a hotel kontinens azonosítóját tartalmazó oszlop
hotel_country	a hotel ország azonosítóját tartalmazó oszlop
hotel_market	a hotel piac azonosítóját tartalmazó oszlop
is_booking	1 értéket vesz fel, ha történt foglalás, különben 0
cnt	hasonló keresések száma
hotel_cluster	a hotel csoport azonosítóját tartalmazó oszlop

Az adatkészletek oszlopait tartalmazó 1. táblázat.

A tanuló adatkészlet három plusz oszlopa az *is_booking*, ami a teszt adatoknál azért nincs, mert ott az összes adat foglalási adat, nem tartalmaz kattintásokat. A második plusz oszlop a *cnt.* A harmadik pedig a *hotel_cluster*, amit azért nem tartalmaz a teszt adatkészlet, mivel arra az oszlopra kell a becslést adni. A *hotel_cluster* oszlop értékeit egy belsős Expedia algoritmussal számolták ki és alkották meg a csoportokat, ami figyelembe vette az árakat, a felhasználó értékeléseket, csillagokat, a városközponthoz viszonyított helyzetet stb.
A teszt adatoknál található plusz oszlop az *id* oszlop, ami szintén a becsléshez lesz szükséges. [37]

A programozás menete

Első lépésként letöltöttem az Anaconda nevű programot. Az Anaconda egy ingyenes, nyíltforráskódú program, ami tartalmazza a Python és az R programozási nyelveket és a hozzájuk tartozó csomagokat. Nagy segítséget nyújt az adattudományokhoz és a gépi tanulás alkalmazásához. Tartalmazza az általam is használt scikit-learn, NumPy, pandas, matplotlib és Seaborn könyvtárakat és különböző környezeteket, mint például a Spyder és a Jupyter Notebook. [38] A Jupyter környezetet használtam a programozáshoz. Ez a környezet egy nyíltforráskódú böngészőben futtatott alkalmazás. Lehetőséget nyújt arra, hogy futtatás után az eredményt azonnal látni lehessen, ami nagy segítség volt ebben a feladatban, illetve nem kell mindig az egész kódot egyben lefuttatni, hanem lehet részletenként is, ami nagyban gyorsítja a munkát, mindez annak köszönhető, hogy a kód cellákra van osztva. [39]

A következő könyvtárakat használtam a dolgozatom során. Pandas, amivel különböző manipulációkat lehet az adatokon végezni, írni és olvasni az adatokat, továbbá segít az adatok tisztításában is. [40] A NumPy csomag alkalmazása a tudományos számításokat teszi lehetővé és segíti. [41] A matplitlib egy olyan könyvtár, amivel rengeteg grafikont, diagramot lehet készíteni az adatok szemléltetéséhez. [42] A Seaborn könyvtár hasonlít a matplotlibhez, mivel ez is a vizualizációkat segíti, illetve a matplotlib könyvtárra épül. Nagyon hasznos statisztikai grafikonok, heatmapek kirajzolásához. [43] A scikit-learn könyvtár a gépi tanulásért felelős. Könnyen és hatékonyan készíthetünk vele prediktív adat elemzéseket.

Mivel nagyon nagy adathalmazzal kellett dolgoznom, így szükségem volt még egy virtuális számítógépre, ami nagyobb erőforrással rendelkezett, mint az én laptopom. Ehhez a Microsoft Azure virtuális gépét használtam, amit diákként az egyetemi e-mail címemmel ingyenesen használhattam. [44] Ennek segítségével elértem egy 56 GiB memóriás, 6 virtuális magos Windows 10 Pro számítógépet egy hónapig, és ezen tudtam futtatni a programokat, amihez RDP-n keresztül tudtam csatlakozni. Ez nagyon meggyorsította az előrehaladásomat.

Adatok átalakítása

Mielőtt a becslést és a komolyabb számításokat elkezdtem, át kellett alakítani az adatokat, olyan formára, amivel az algoritmusok könnyebben tudnak dolgozni, illetve létrehoztam új oszlopokat, amik segítettek és kitöröltem olyanokat, amik hátráltatták a számításokat. Ezeket a változtatásokat egy Jupyter Notebookban végeztem el.

Ezután létrehoztam több függvényt, amiket majd a későbbiekben használni fogok. Először készítettem egy *get_season* nevű függvényt, amivel a hónapokból ki tudom majd számolni az évszakot. Ezután *calc_duration* nevű függvényt készítettem, amivel ki fogom számolni, hogy mennyi időre szeretne a felhasználó szállást foglalni, illetve, hogy mennyi idő telt el a keresés és a keresett bejelentkezési dátum között. Ezeket az új oszlopokat csak az adatok feldolgozásánál fogom használni. Ezek után kinyertem a bejelentkezés évét, hónapját, napját, a bejelentkezési dátumból. Hónapot rendeltem minden sorhoz a *get_season* függvényemmel. Összeadtam a keresett felnőttek és gyerekek számát, így könnyítve az algoritmusokon. Illetve kiszámoltam a fent már említett időtartamot tartalmazó oszlopokat. Kitöröltem a dátum formátumú oszlopokat, mivel ezek mindig bezavartak a számításokba. Legvégül pedig az összes adatot átkonvertáltam egész (int) típusúra, mivel azt vettem észre, hogy ezzel sokkal gyorsabban lefutnak a programok, mint a lebegőpontos (float) típussal.

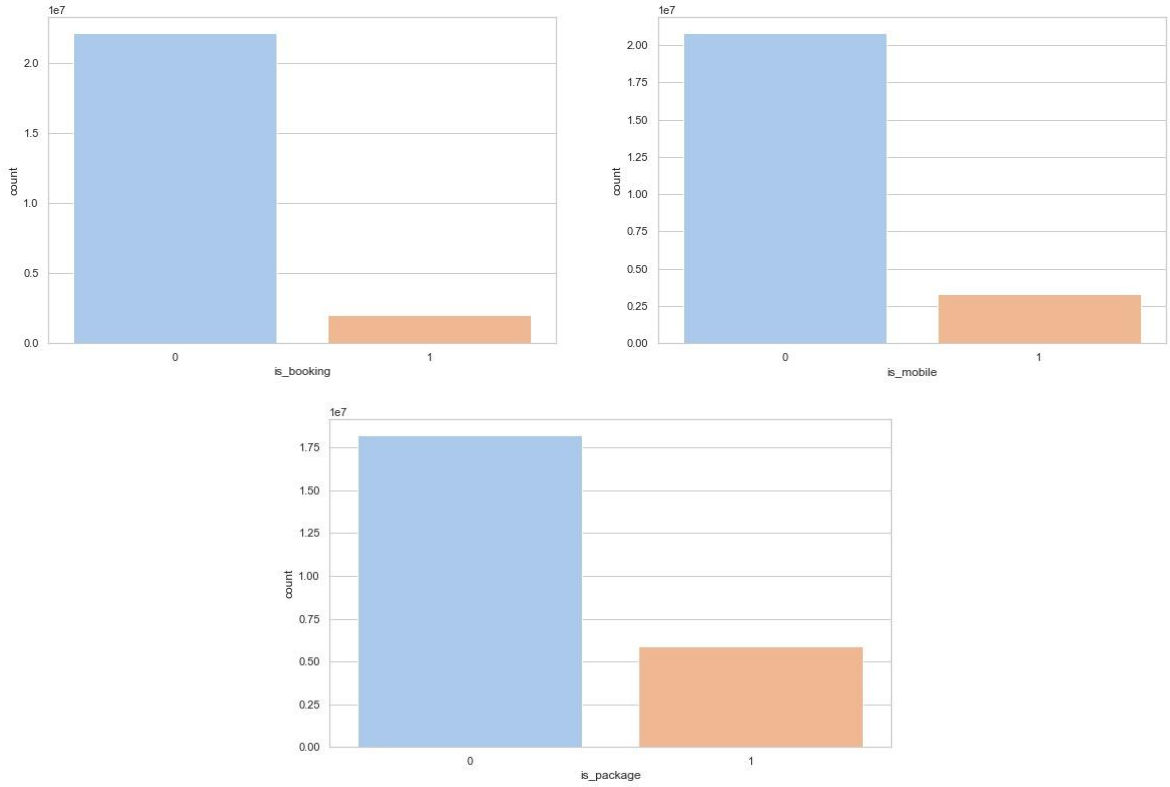
Mivel sok sor tartalmazott null adatokat, így létrehoztam egy tisztított adatkészletet a tanuló adatokból, ahonnan kitöröltem minden null adatot tartalmazó sort. Ezáltal a tanuló adatok 37670293 sora lecsökkent 24117894 sorra. Ezután még tovább csökkentettem a tanuló adatok számát 1985514 sorra, azzal, hogy kitöröltem belőle azokat az sorokat, amiknél az *is_booking* oszlopban nulla volt, azaz nem történt foglalás csak keresés volt. Erre azért volt szükség, mert a teszt adatkészlet is csak olyan sorokat tartalmaz, ahol történt foglalás, ezért a keresések ebben a feladatban nem relevánsak. Először számoltam úgy is, hogy nem voltak kitörölve ezek a sorok, de sokkal rosszabbak lettek az eredmények.

Ezek után létrehoztam egy fájlt, amibe a tisztított adatok, minden 23. sorát tettem, hogy összesen 1000000 sort tartalmazzon, majd ezen végeztem el a fent már említett módosításokat. Erre az adatkészletre az adatok elemzéséhez volt szükségem.

Majd létrehoztam egy újabb 1000000 soros adatkészletet a tanuló adatokból, amit validációs célra használtam. Végül a teszt adatokon végeztem el a módosításokat. Itt nem volt szükség sorok törlésére, csak az új oszlopokat határoztam meg.

Adatok elemzése

Ebben a részben még részletesebben ismertetem az adatokat, diagramok és ábrák segítségével. Mivel a tanuló adatokból fognak építkezni az algoritmusok, így alaposabban körbejártam és megkerestem az összefüggéseket a különböző oszlopok között.



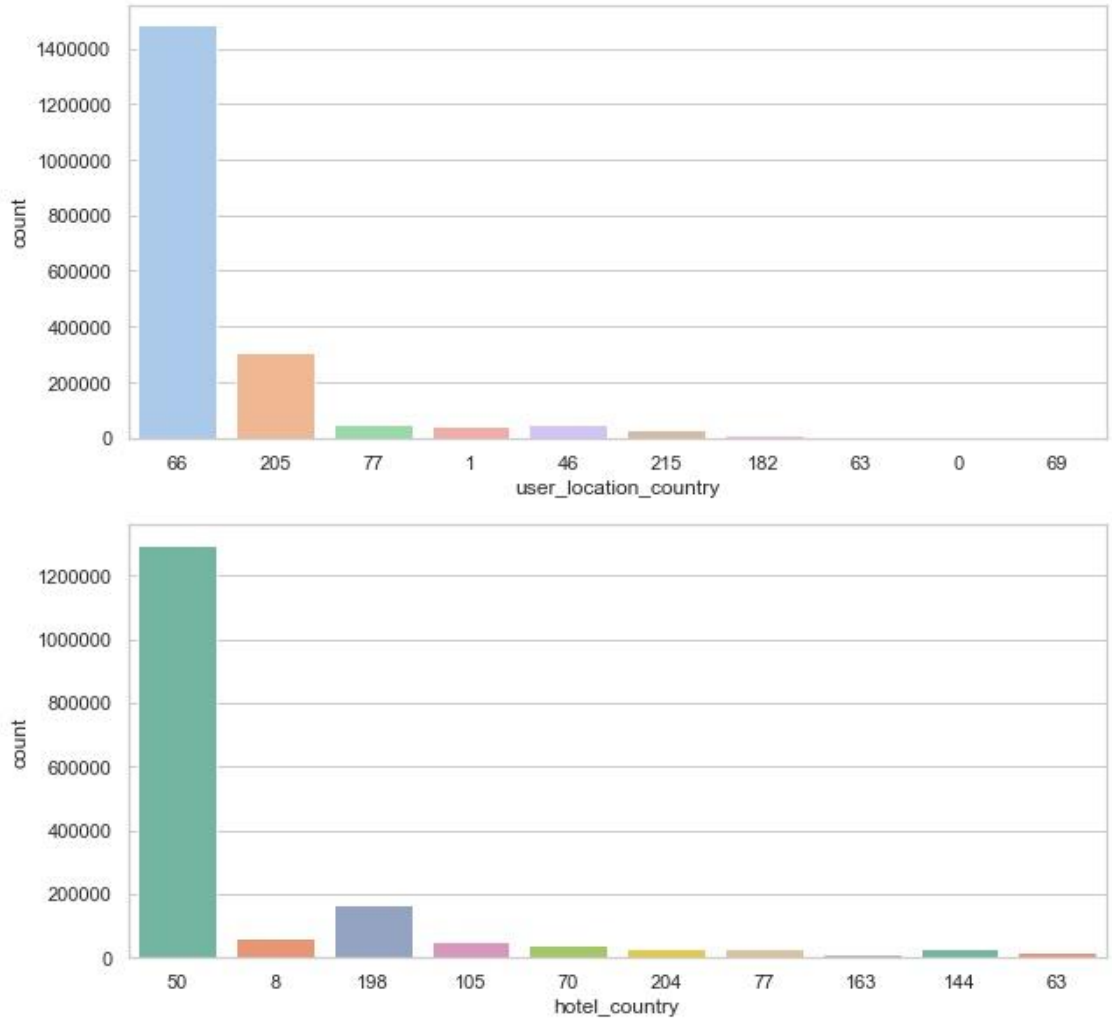
A 8. képen az is_booking, is_mobile és is_package értékeinek az összesített száma látható.

Ebből kiderül, hogy a felhasználók 92%-a nem foglalta le a keresett szállást, csak megnézte. A keresések mindössze 14%-a történt telefonról és 86%-a egyéb eszközről.

A felhasználók 24%-a keresett szállást egy csomag tartalmaként és 76%-a kereste csak a szállást.

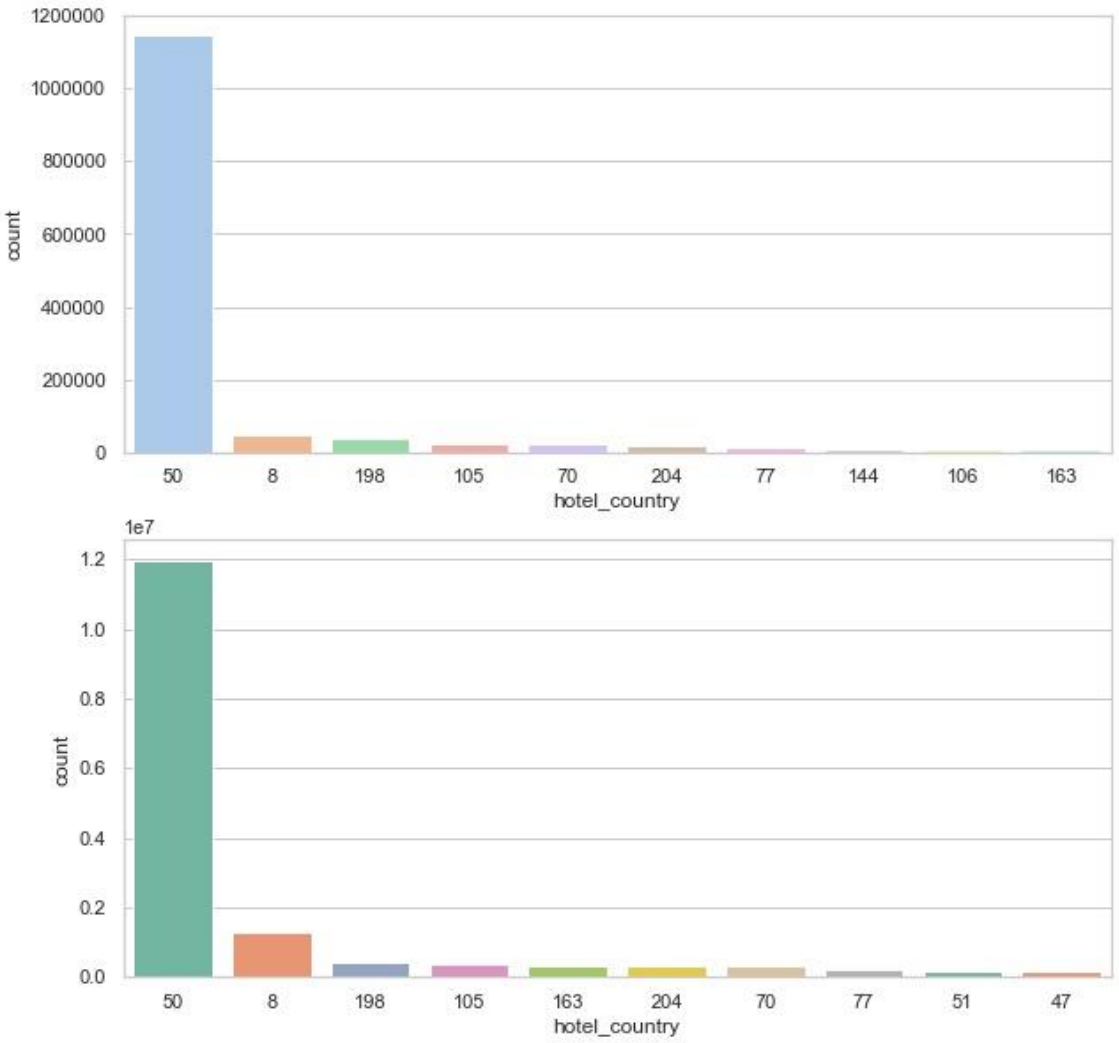
Ezekből az adatokból sokat megtudhatunk a felhasználók viselkedéséről, nem csak a feladatomhoz hasznos, hanem a cég szempontjából kiderül, hogy jobban megéri a böngészőben lévő keresőt fejleszteni, mint egy mobil alkalmazást, mivel a felhasználók kis százaléka fogja azt használni. Marketing szempontjából is hasznos információ, hogy milyen felületen jelentessenek meg több reklámot. Illetve kiderül az is, hogy az emberek szeretnek sok szállást megnézni foglalás előtt, az adatok alapján 10-ből egy foglalás történik, azaz 9 szállást néz meg mielőtt foglalna.

A 9. képen látható diagram azt mutatja, hogy melyik országból hány felhasználó foglalt szállást. A felhasználók 75%-a, 1482408 felhasználó foglalt szállást a 66-os országból, mivel ez a szám ilyen magas a többihez képes és 51 régió tartozik hozzá, így arra lehet következtetni, hogy ez az Amerikai Egyesült Államok. Az ezt követő leggyakoribb ország a 205-ös, amiből 306674 ember foglalta le a pihenését.
A 10. ábrán az ország, ahol hotel található előfordulási számai láthatók, ahova a felhasználó foglalta a szállást. Ebből kiderül, hogy a felhasználók 65%-a foglalt szállást az 50-es számú országba.



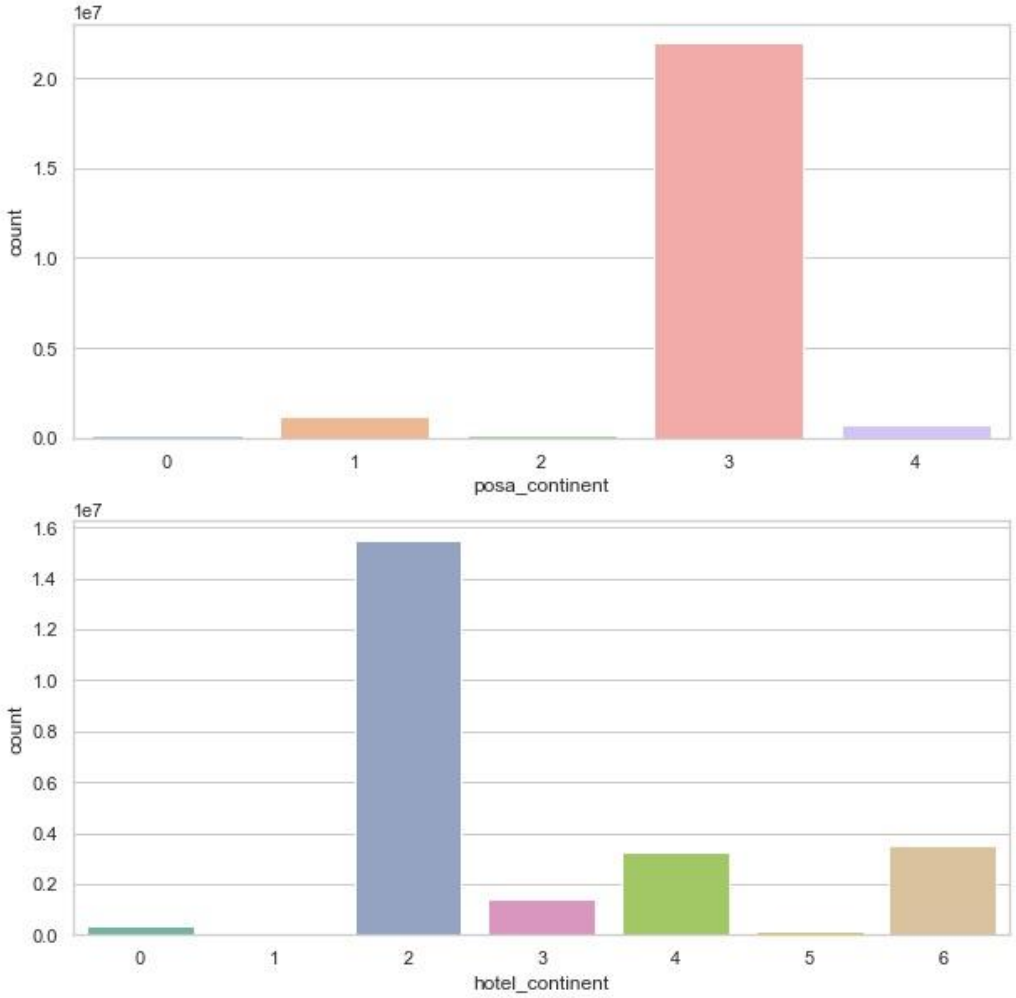
A 9. képen a felhasználó, a 10. képen a hotel kontinenseinek előfordulási összesítése.

A 11. és 12. ábra azt mutatja, hogy a 66-os számú országból melyik számú országba foglaltak és kerestek a legtöbben. A 11. ábra mutatja azokat az adatokat, amikor történt foglalás is. Az 12. ábrán pedig láthatóak a keresések is. Érdekesnek találtam, hogy arányaiban mennyivel többen kerestek egy-egy országot, mint ahányan oda foglaltak. Látszik az is, hogy a sorrend nem egyezik a két diagramnál. Az első négy hely megegyezik, de utána vannak különbségek a keresések és foglalások száma között. Erre az lehet a magyarázat, hogy sokan szeretnének elmenni valahova, de végül egy egészen más ország lesz a célpont.



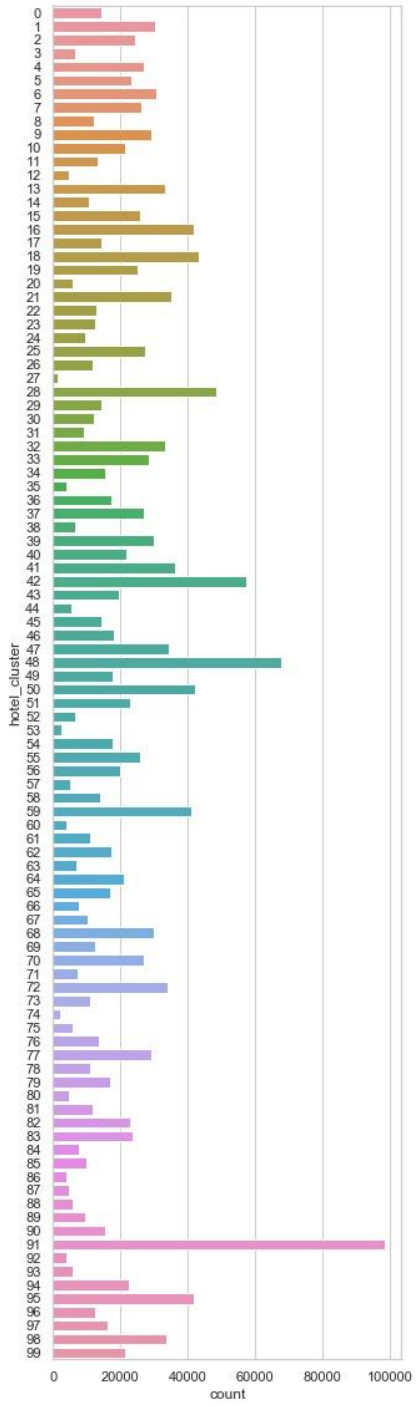
A 11. és 12. képen 66-os országból utazó felhasználók első 10 célországának az összesítése.

A 13. és 14. ábrán a felhasználó és a hotel kontinensének előfordulásait láthatjuk. A legtöbb felhasználó a 3-as kontinensről keresett. A 3-as kontinensről 21919801felhasználó keresett és 1823964 foglalt szállást. Ez azt jelenti, hogy a 66-os ország a 3-as kontinensen található. A hotel kontinensét tekintve pedig a 2-es volt a legnépszerűbb, ahova 15487235 felhasználó keresett és 1459937 foglalta le pihenését.



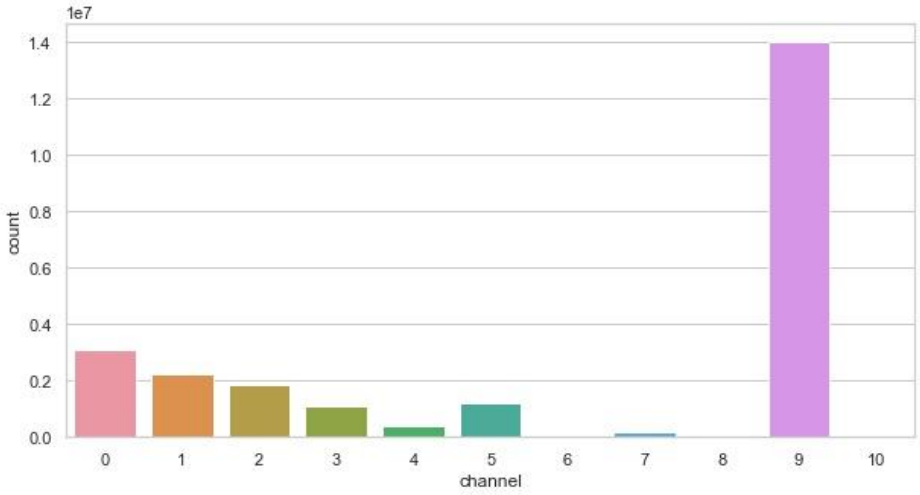
A 13. a keresés, a 14. ábrán a hotel kontinenseinek összehasonlítása.

A 15. képen a hotel csoportjainak az összesítése látható. A legtöbbet foglalt és keresett hotel típus 91-es. Összesen 829368 felhasználó keresett olyan hotelre, amit az Expedia algoritmusa a 91-es csoportba sorolt, ebből 98550 foglalt ebbe a csoportba tartozó szállást. Az ábra megmutatja, hogy nincs olyan szálláscsoport, amit a felhasználók nem kerestek és nem foglaltak.



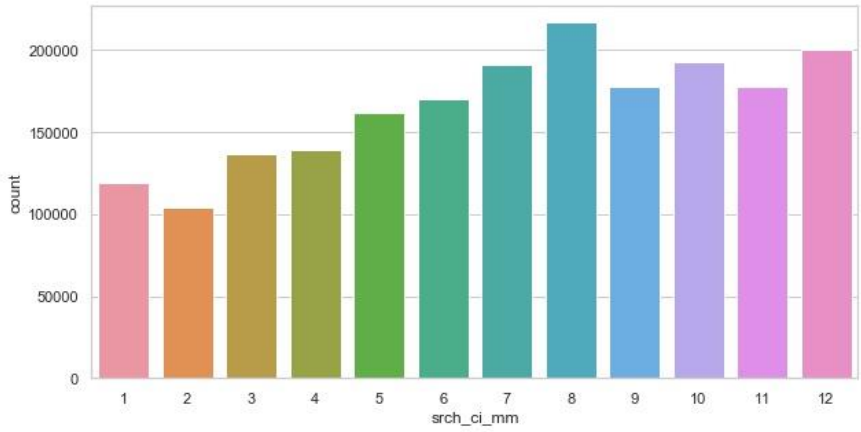
A 15. ábra hotel klasztereinek előfordulási összesítése.

A 16. ábrán a marketing csatornák előfordulásai láthatók, azaz melyik marketing csatornáról kerestek és foglaltak legtöbben. A 9-es marketing csatorna volt a legnépszerűbb. Mivel korábban már látszódott, hogy a felhasználók kevésbé részesítik előnyben a telefonos keresést/foglalást, így nagy valószínűséggel a 9-es csatorna egy böngészőben előforduló reklám, amire 14000141 felhasználó kattintott és ebből 8%-a foglalta le az utazást.



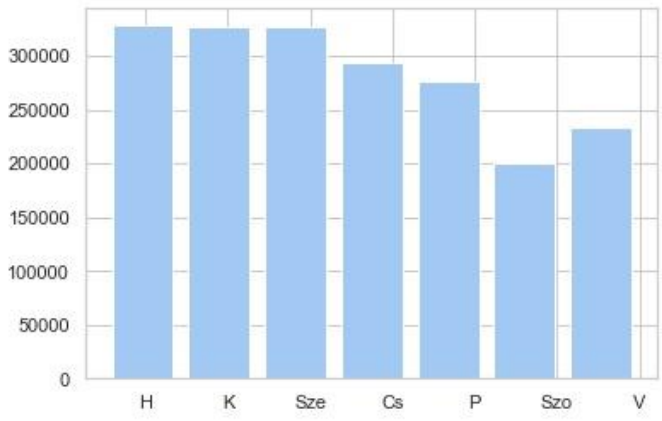
A 16. kép a marketing csatornák előfordulásainak összesítése.

A 17. képen az utazás időpontjával kapcsolatos érdekességek láthatók. Az ábra a foglalások számának eloszlását mutatja a hónapokra lebontva. Kiderül, hogy leggyakrabban augusztusra foglaltak szállást és februárban volt a legkevesebb utazás. Meglepőnek gondolom, hogy a téli hónapok ennyire elmaradtak a tavasztól és ősztől, mivel úgy gondoltam, hogy a téli sportok miatt gyakrabban foglalnak szállást a téli hónapokra, mint a változékony tavaszra. Illetve az is látszik, hogy a második legnépszerűbb hónap a december, ez azért lehet, mert nagyon sok felhasználó töltötte a szilvesztert szállodában, várva az új évet.



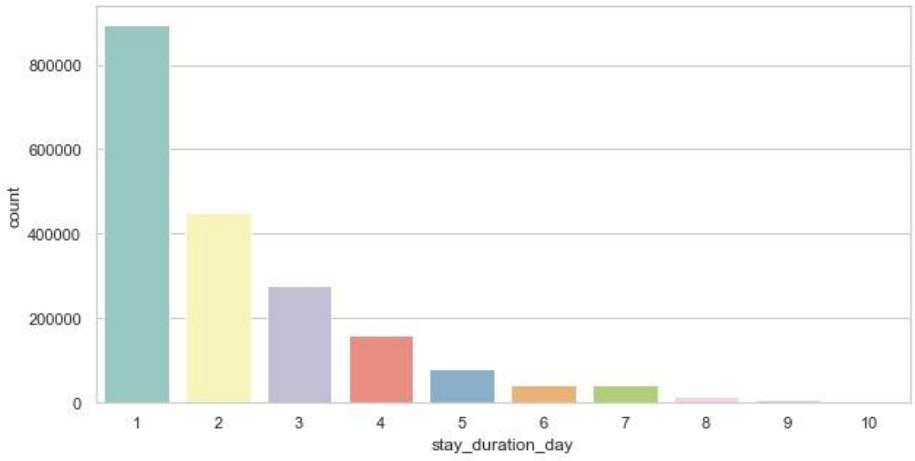
A 17. kép foglalt hónapok előfordulásainak összesítése.

A 18. látható ábra mutatja, hogy melyik napra hány felhasználó foglalt szállást. A napokat úgy számítottam ki a dátumokból, még a változtatásokkal együtt és hoztam létre egy új oszlopot a hét napjaival. Számomra nagy meglepetés volt, hogy hétvégére volt a legkevesebb foglalás, amíg a hét első felében nagyjából mindegyik napra ugyanannyi. Ez az információ minden utazással foglalkozó szolgáltatónak, mint szállás, repjegy, utazási irodák nagyon hasznos információ. Így ennek függvényében időzíthetik az akciókat. Az igaz, hogy akiknek munka miatt kell sokat utazniuk, azoknak hétköznap van szükségük szállásra. Viszont én eddig úgy gondoltam, hogy inkább a hétvégék népszerűek, mert akkor nem kell szabadságot kivenni.



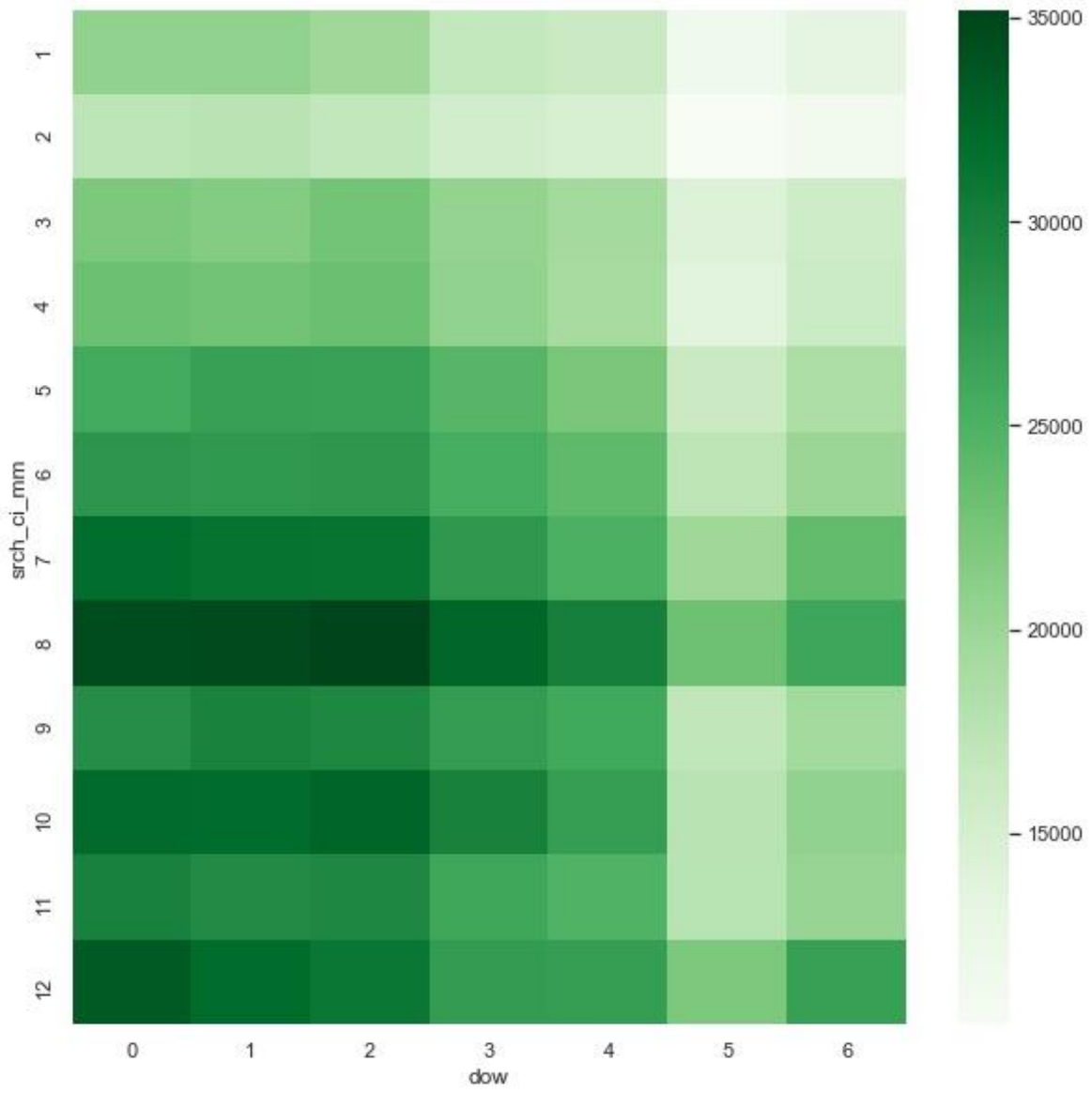
A 18. diagram a hét napjainak előfordulásai.

Számomra a 19. diagram adatai is különösek voltak. Meglepett, hogy az egy éjszakára foglalók száma ennyire kiugrik a többihez képest. Erre szintén magyarázat lehet a munkaügyben gyakran utazó emberek, de az is, hogy a felhasználók évente gyakrabban utaznak el egy-két napra, és inkább egy vagy maximum kétszer hosszabb időre.



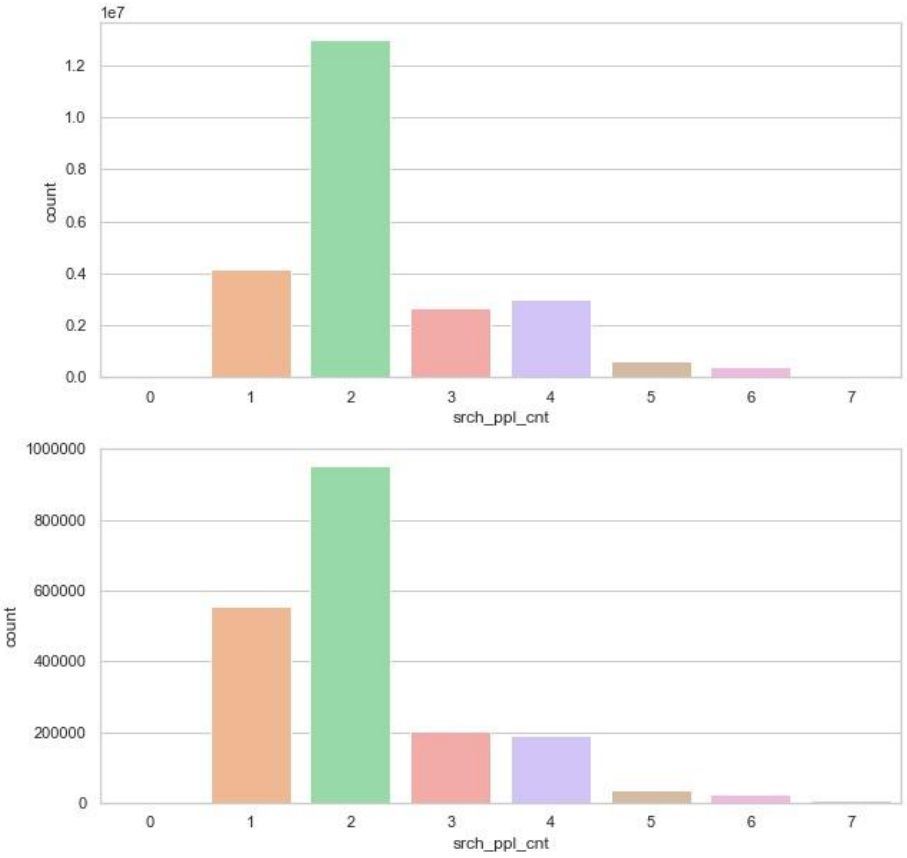
A 19. ábrán a lefoglalt utazások hosszának előfordulásai.

A következő érdekes heatmap összeveti a foglalás hónapját a hét napjaival. Mint korábban már kiderült a leggyakoribb hónap az augusztus a hetek napját tekintve pedig a hétköznapok. Ez a diagram megmutatja ezeknek az eloszlását egymáshoz képest. Amiből láthatjuk, hogy decemberben többen utaznak vasárnap, mint augusztusban, ami érthető, mivel sok ember megy el másik városba adventi vásárba és karácsonyra bevásárolni.



A 20. kép a foglalások hónapjainak és napjainak összevetése.

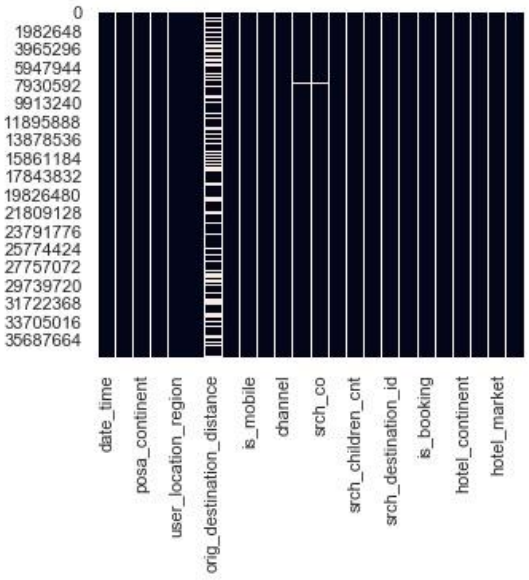
A 21 diagramon látható, hogy hány felhasználó hány ember számára keresett szállást.
A 22 diagram azt ábrázolja, hogy hány embernek foglaltak szállást a felhasználók. Ebben az esetben érdekes, hogy a kattintások és a foglalások mértéke nem arányos. Arányait tekintve többen foglaltak egy főre, mint ahányan keresték a szállást. Erre az lehet a magyarázat, hogy ha valaki egyedül utazik, az nem néz meg annyi szállást, mert nem számít neki, ha nem teljesen az elvárásának megfelelő. Viszont két embernél már jobban meg kell válogatni, hogy mindegyikőjük igényeit kielégítse, és ehhez többet kell keresgélni. Illetve, akik munkaügyben utaznak azok is csak egy személyre foglalnak. Az ábrákon látszik, hogy legtöbben két főre kerestek és foglaltak szállást. 953440 felhasználó foglalt két személyre, 554823 darab egy személyes és 191596 darab négy személyes foglalás történt.



A 21. és 22. képen az emberek számának összesítése.

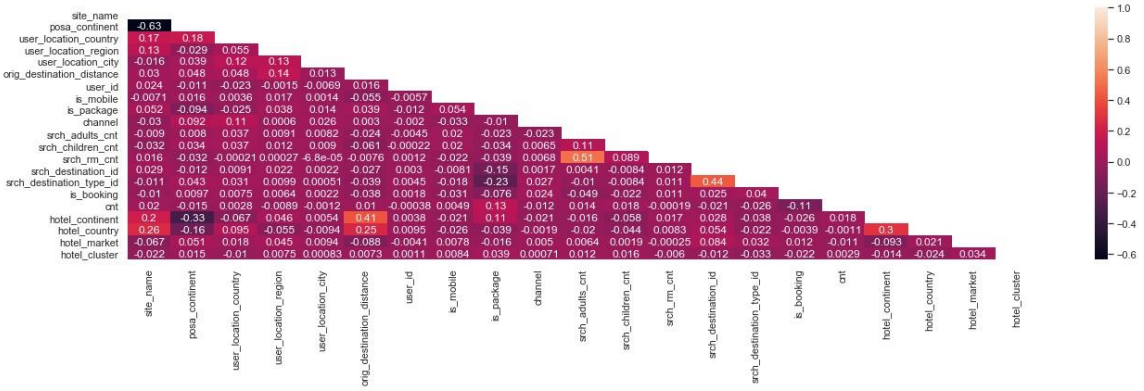
Az utolsó két diagramhoz az összes adatra szükségem volt, változtatások nélkül, mivel ezekből derültek ki, hogy melyik oszlopok mennyit érnek, mennyire hasznosak.

Először a null adatokat kerestem meg az oszlopokban egy heatmap segítségével. A heatmap megmutatta, hogy három oszlopban találhatóak null adatok, az *orig_destination_distance* oszlopban, ami a keresés és a szállás helye közötti távolságot tartalmazza 13525001 darab null található. Míg az *srch_ci* és *srch_co* oszlopokban 47083 és 47084 darab hiányos sor van. Ez azt jelenti, hogy ezek az oszlopok nem hasznosak, mivel a sok null adat félrevezeti a tanuló algoritmusokat. Ezért töröltem ki a null adatokat tartalmazó oszlopokat még a program legelején.



A 23. ábrán a null értékek száma.

A 24. ábra az oszlopok közötti korrelációt mutatja. Ami megmutatja minden két oszlop egymáshoz való viszonyát. Azaz mennyire függ egymástól a két oszlop értéke. Minél nagyobb szám látható a két oszlopnál, annál nagyobb mértékben befolyásolják egymást.



24. ábra az oszlopok közötti korreláció.

Becslés elvégzése

Ahogy a dolgozatom elején írtam a termékajánlórendszereket sok módon lehet fejleszteni. A dolgozatomban a gépi tanulási módszereket jártam alaposan körbe és próbáltam ki. A versenynek lettek jobb eredményei is, de a célomat elértem azzal, hogy sokat tanultam a gépi tanulásról, algoritmusokról. Azokat az algoritmusokat próbáltam ki, amelyekről a dolgozatom első felében írtam, mint a nearest neighbors, decision tree, random forest, stb.

Mint az adatok elemzésénél és módosításánál, először itt is a használt könyvtárakat kellett importálni.

A tanuló adatkészlet, amit a becsléshez használtam, már nem tartalmazott null adatú sorokat és klikkeléseket. De az újonnan létrehozott oszlopokat már igen.

A teszt adatkészlet is tartalmazza az új oszlopokat.

A korábban bemutatott elemzés alapján a következő oszlopokat hagytam meg a tanuláshoz: *user_location_city, is_mobile, is_package, srch_destination_id, hotel_country, hotel_market, hotel_cluster, srch_ci_season, srch_ppl_cnt, stay_duration_day, search_duration_day*

Miután beolvastam az adatokat és kitöröltem a feleslegesnek vélt oszlopokat, felosztottam a tanuló és a teszt adatkészletet jellemzőkre (*features*) és címkékre (*labels*). A jellemzők tartalmazzák azokat az adatokat, amelyekből az algoritmus tanulni fog. Jelen esetben ezek a beolvasott oszlopok. A tanuló és a tesztelő jellemzőinek ugyanolyanoknak kell lenniük, ugyanazokat az oszlopokat és adattípusokat kell tartalmazniuk. A címkék a becslés eredményei, jelen esetben ez lesz a hotel klasztere. A tanuló adatok esetében ez a címke a *hotel_cluster* oszlop. A teszt adatoknál pedig ez lesz a kimenete a becslésnek.

Ezután írtam egy függvényt, ami egy listából az öt legjobbat beleteszi egy karaktertömbbe. Ez kerül majd bele az eredményfájlba.

Miután felosztottam az adatokat jöhetett az algoritmus meghatározása. Ezekről a következő részben fogok részletesebben írni.

Ezek után jön az algoritmus tanítása és a teszt címkék létrehozása. A címkék létrehozása több függvénnyel is történhet. Először a *.predict* -et használtam, ami elvégzi a becslést, hogy megkapjuk a megadott adatok címkéit. Ebben az esetben az volt a probléma az esetemben, hogy egy eredményt adott a címkéhez. A verseny lehetővé tette, hogy maximum ötöt írjak és ez alapján jobb pontosságot tud kiszámítani a feljebb már említett MAP@5 képlettel. Ehhez a *predict_proba*-t kellett meghívnom. Ez a teszt adatokhoz tartozó becslés valószínűségeket adja vissza. Ebből kerestem ki a fent említett algoritmussal az öt legjobbat.

Miután megtörtént a teszt címkéinek számítása már csak az eredmények fájlba írása maradt a megadott követelmények alapján. Végül ezt az eredmény fájlt töltöttem fel a Kaggle oldalra kiértékelésre.

Amikor az első számolásokat végeztem nagy hibát követtem el azzal, hogy nem figyeltem több dologra, amik, mint kiderült döntők voltak a pontosságban. Az első ilyen hibám volt, hogy az összes oszlopot meghagytam. Ez nem csak a tanulási folyamatot lassította, de a becslés pontosságán is nagyon sokat rontott. A sok adat azért is lassította nagyon a számításokat, mert voltak benne olyan sorszámot tartalmazó oszlopok, amikkel nem tudott mit kezdeni. Nem csak azért, mert túl sok adat volt, hanem azért is, mert nem vettem figyelembe, hogy a tanuló és a teszt adatkészlet más dimenziójú és nem ugyanazokat az oszlopokat tartalmazzák, ez az egész számítást lerontotta. További hibám volt, hogy a tanuló adatokból rossz oszlopot adtam meg a tanuló címkéknek, a hotel klasztere helyett véletlenül a sorszámot tartalmazó oszlopot használtam. Sokat javított a pontosságon az, amikor utánanéztem a kiértékelésnek és láttam, hogy egy felhasználóhoz több klasztert is be lehet írni, és ezt a feljebb említett képlettel számolja ki. Ezután tértem át a sima becsléses függvényről a valószínűséget becslő függvényre és készítettem el ehhez az öt legjobbat megkereső algoritmust.

GridSearchCV()

A GridSearch-et hiperparaméterek optimalizálására használják Az algoritmusoknak nagyon sok paramétere van. A célom az volt, hogy megtaláljam az adataimhoz passzoló legjobb paraméter párosításokat és értékeiket. Ehhez a GridSearch algoritmust hívtam segítségül, ami megtalálja az adott paraméter értékekből az optimálisat. [45] Ennek a meghatározásához ugyanazokat az adatokat használtam, mint a becsléshez. Az a különbség, hogy még a dolgozatom elején készítettem validációs adatokat a tanuló adatokból, itt azokat fogom használni a teszt adatok helyett, mivel ez tartalmazza már a megoldást, így könnyen le lehet ellenőrizni. Miután meghatároztam a tanuló és a validációs adatok jellemzőit és címkéit meg kell adnom, hogy melyik algoritmusnak szeretném megtalálni a legjobb paraméter értékeit. Az elküldött példában ez a közeli szomszédok klasszifikáció (KNeighborsClassifier). Először paraméterek nélkül tanítottam az algoritmust, majd meghatároztam a becslés eredménye alapján a teszt címkéket. Ezután megadtam, hogy milyen paramétereket, milyen értékekkel szeretnék vizsgálni. Jelen esetben a *leaf_size, n_neihbors* és a *weights* paramétereknek kerestem a legjobb értékeket. A GridSearch algoritmusnak van egy olyan paramétere, ami alapján felosztja az adatokat a keresztvalidációhoz, a neve *cv*. Ezt a paramétert állítva különböző pontosságot és más legjobb paraméter beállításokat kaptam, így ezt változtattam, amikor a legjobbat kerestem a gépi tanulási algoritmusoknak. Miután megtörtént a keresztvalidáció megnéztem a pontosságot és a legjobb paraméter értékeket és ezután tértem rá az éles becslésekre.

Az algoritmusok

KNeighborsClassifier()

A közeli szomszédok esetében a GridSearch-vel négy paramétert szabtam személyre. Az *algorithm, leaf_size, n_neighbors* és a *weights* paramétereket.

Az első paramétere az *n_neighbors* alapértelmezett beállítása 5. Ez a paraméter adja meg, hogy hány szomszédot vizsgáljon az algoritmus.

A *weigths* paraméter két értéket vehet fel. Az egyik az alapértelmezett *'uniform'*, a másik a *'distance'*. Ezzel határozhatjuk meg, hogy az algoritmus, hogyan súlyozza a közeli szomszédokat. A *'uniform'* értékkel az összes közeli szomszéd ugyanúgy van súlyozva, míg a *'distance'* esetében a közeli szomszédok jobban befolyásolják a becslés eredményét, mint a távolabbiak.

Az *algorithm* paraméterrel állíthatjuk be, hogy milyen algoritmust használjon a közeli szomszédok számításához. Ez négy féle lehet, *'brute', 'kd_tree', 'ball_tree'* és *'auto'*. A *'brute'* algoritmus nagyon hatékonyan tud működni kicsi adatmintákra, az algoritmus hatékonysága $O[DN^2]$, ahol D a dimenziót, azaz az adatok jellemzőinek a számát, N a sorok darabszámát jelöli. A *'kd_tree'* nagyobb adatkészletek esetében is hatékonyabban tud dolgozni, mint a *'brute'*, mivel az alap ötlet mögötte az, hogy ha A pont nagyon távol van B ponttól, és B pont nagyon közel van a C ponthoz, akkor tudhatjuk, hogy A pont is nagyon távol van a C ponttól, és ahhoz, hogy ez kiderüljön számunkra nem kell kifejezetten kiszámolni a pontos távolságokat. Ennek az algoritmusnak a lépésszáma $O[DN\log(N)]$ vagy kevesebb, ami jelentős javulást eredményez a *'brute'*-hoz képest nagy N esetén, viszont több dimenziónál sokat veszít a hatékonyságából. Az én esetemben a *'ball_tree'* bizonyult a legjobbnak, ami több dimenziós adatok esetében hatékonyabban működik, mint a társai. A *'ball_tree'* rekurzívan osztja szét az adatokat csomópontokra egy középpont és egy sugár között. A lehetséges szomszédok keresését egy háromszög egyenlőtlenséggel határozza meg: $|x + y| \leq |x| + |y|$. Így ennek az algoritmusnak $O[D\log(N)]$ lesz a hatékonysága. Az utolsó lehetőségünk az *'auto'* algoritmus, ami az adatok fényében automatikusan állítja be a legjobbnak vélt algoritmust.

A *leaf_size* alapértelmezett beállítása 30. Ez a paraméter adja meg a levelek számát *'ball_tree'* vagy *'kd_tree'* algoritmusok esetében. Ennek az értéke befolyásolja a tanulás és a becslés sebességét és memória igényét. Minél nagyobb az értéke, annál gyorsabban tudja a fát felépíteni, mivel így kevesebb csomópontot kell létrehoznia. A *leaf_size* értékének növelésével a fa tárolásához szükséges memória igény csökken.

[46] [47]

A táblázat tartalmazza a vizsgált paraméterek legjobbnak ítélt értékeit, az eredményt, amit a kiértékelés után kaptam, a GridSearch-nél használt cv paramétert és a GridSearch általi pontosságot. Ezzel az algoritmussal a legjobb eredményem 0.10454 lett, ami 10%-os egyezést jelent. Úgy gondolom elsőre ez egy nagyon jó eredmény.

algorithm	***leaf_size***	***n_neighbors***	***weights***	**Kaggle eredmény**	**GridSearch CV**	**GridSearch pontosság**
'ball_tree'	200	30	*'distance'*	0.10205	3	0.059
'ball_tree'	200	40	*'distance'*	0.10269	5	0.078
'ball_tree'	200	200	*'distance'*	**0.10454**	8	0.099
'ball_tree'	20	100	*'distance'*	0.10390	9	0.106
'ball_tree'	100	5	*'distance'*	0.09402	10	0.095
'ball_tree'	200	50	*'distance'*	0.10299	11	0.103
'ball_tree'	200	10	*'distance'*	0.09903	12	0.104
'ball_tree'	200	150	*'distance'*	0.10426	22	0.13

DecisionTreeClassifier()

A döntési fa esetében a GridSearch-vel három paramétert szabtam személyre. Az *max_depth, min_samples_leaf,* és a *min_samples_split* paramétereket. Továbbá megismerkedtem még a *splitter* és a *max_features* paraméterekkel is, de azokat az alapértelmezett beállításokkal használtam.

Az első vizsgált paraméter a *splitter* volt. Ennek kettő értéke lehet, *'best'* és *'random'*. Ez a paraméter a stratégiát takarja, ami alapján az algoritmus kiválasztja a csomópontok felosztását. Az alapértelmezett beállítás a *'best'*, ami a legjobb felosztást választja. Illetve választhatjuk még a *'random'* értéket is, ami véletlenszerűen rendezi a felosztást. Én mindig a *'best'* paramétert használtam, mivel a GridSearch minden esetben azt adta meg jobbnak.

A *max_depth* paraméter adja meg a maximum mélységét a fának. Bármilyen int értéket adhatunk neki vagy *None* értéket. Ha *None*, akkor a csomópontokat addig bővítjük, amíg az összes levél hibátlan nem lesz, vagy amíg nem kevesebb, mint a *min_samples_split*. Minél mélyebb a fa, annál több felosztást fog tartalmazni és annál több információt fog tudni az adatról. Könnyen abba a hibába lehet esni túl a nagy mélység beállításánál, hogy felül tanulás jön létre.

A *min_samples_split* adja meg a belső csomópontok felosztásához szükséges minták minimális számát. Minél nagyobb ez az érték, a fa annál korlátozottabb lesz, mivel minden egyes csomópontnál több mintát kell figyelembe vennie és nem tud meg eleget az adatokról. Ebből kifolyólag alultanulás jöhet létre.

A *min_samples_leaf* határozza meg a csomópontokban lévő minták minimális számát, amik majd a fa alapját fogják adni.

Max_features mutatja meg, hogy a legjobb felosztás keresésekor mennyi jellemzőt vegyen figyelembe. Alapértelmezett beállítás szerint ez a szám egyenlő a jellemzők számával, ami a tanulásnál létrejött, ezt nevezzük *n_features*-nek. Lehet még *'auto'*, ami egyenlő az *'sqrt'* értékével, ekkor a *max_features* = sqrt(*n_features*). 'log2' az utolsó lehetséges értéke, aminél a következőképp alakul a képlet: *max_features* = log2(*n_features*). Én minden számításomnál az alapértelmezett beállítást használtam. [48] [49]

A táblázat tartalmazza a *max_depth, min_samples_leaf,* és a *min_samples_split* paraméterek legjobbnak ítélt értékeit, az eredményt, amit a kiértékelés után kaptam, a GridSearch-nél használt cv paramétert és a GridSearch általi pontosságot. Ebben az esetben összességében jobb erdményeket kaptam, mint az előző algoritmusnál. A legjobb eredmény 0.19603 lett a lent látható paraméterekkel.

max_depth	*min_samples_leaf*	*min_samples_split*	**Kaggle eredmény**	**GridSearch cv**	**GridSearch pontosság**
9	0.001	20	0.15539	2	0.074
9	5	5	0.15957	3	0.084
9	0.001	10	0.15539	5	0.076
10	5	75	0.16870	8	0.085
9	5	50	0.15959	9	0.084
9	5	10	0.15957	11	0.09
24	1	25	0.18172	10	0.096
32	1	25	0.17285	12	0.093
10	0.01	20	0.13102	15	0.089
10	5	5	0.16860	18	0.097
16	1	25	**0.19603**	20	0.096
10	5	10	0.16860	22	0.095

RandomForestClassifier()

A véletlen erdő esetében a GridSearch-vel négy paramétert vizsgáltam. A *max_depth, min_samples_leaf, min_samples_split* és a *n_estimators* paramétereket. Illetve ennél az algoritmusnál is megtalálható a *max_features* paraméter, de mivel a véletlen erdők a döntési fán alapulnak, így ebben az esetben ugyanazokat az értékeket veheti fel és én ebben az esetben mindenhol az alapértelmezett *'auto'* érték szerint végeztem a becslést.

Az első három vizsgált paraméterem a *max_depth, min_samples_leaf, min_samples_split* teljesen ugyanúgy működik és viselkedik, mint a döntési fák esetében, mivel az erdőben fák találhatóak.

Az *n_estimators* változás az előbbiekhez képest, mivel ez mondja meg, hogy hány fa legyen az erdőben. Általában, ha több fát tartalmaz az erdő, az algoritmus többet tud tanulni az adatokról. De, mint mindenhol, itt is előfordul túltanulás, ha túl nagyra vesszük ennek az értékét. [50]

A véletlen erdőkkel kaptam a legjobb eredményeket. Ezzel sikerült 20%-os eredményt elérni a versenyben.

max_ depth	***min_samples_leaf***	***min_samples_split***	***n_estimators***	**Kaggle eredmény**	**GridSearch cv**	**GridSearch pontosság**
16	1	0.1	32	0.11290	2	0.084
32	0.01	5	32	0.12538	3	0.104
10	1	25	100	0.16935	4	0.102
9	1	10	75	0.16290	5	0.106
16	1	25	75	0.19689	6	0.112
100	10	5	75	0.19756	7	0.114
25	1	50	100	**0.20224**	8	0.115
50	1	50	100	0.19991	11	0.114
10	0.01	10	50	0.12565	12	0.116
9	10	5	75	0.16162	13	0.115
None	1	50	100	0.19986	15	0.119
16	1	25	100	0.19644	20	0.120

AdaBoostClassifier()

Ennél az algoritmusnál is használtam a GridSearch adta lehetőségeket, de csak kettő paramétert vizsgáltam, a *learning_rate* és az *n_estimators* paramétereket.

A *learning_rate* csökkenti minden osztályozónak a hozzájárulását az értékével. Szoros összefüggésben áll az *n_estimator* paraméterrel.

Az *n_estimators* a maximális becslések száma, amelynél az algoritmus stagnál. Amikor megtalálta a tökéletes állapotot a becslés hamarabb fejeződik be.

A *base_estimator* határozza meg, hogy az adaptív erősítés melyik algoritmust használja. Ez alapértelmezetten a *DecisionTreeClassifier(max_depth=1).* [51]

A táblázat tartalmazza az összes kipróbált lehetőséget. Látszik, hogy ez az egyik legrosszabb eredményeket adó algoritmus. Ezért nem is próbáltam ki többet, illetve a többi cv paraméter általában ugyanezeket a paraméter beállításokat javasolta.

learning_rate	***n_estimators***	**Kaggle eredmény**	**GridSearch CV**	**GridSearch pontosság**
1	100	0.08369	2	0.055
0.1	1	0.08867	3	0.053
1	25	**0.09097**	4	0.059
1	10	0.08318	8	0.056

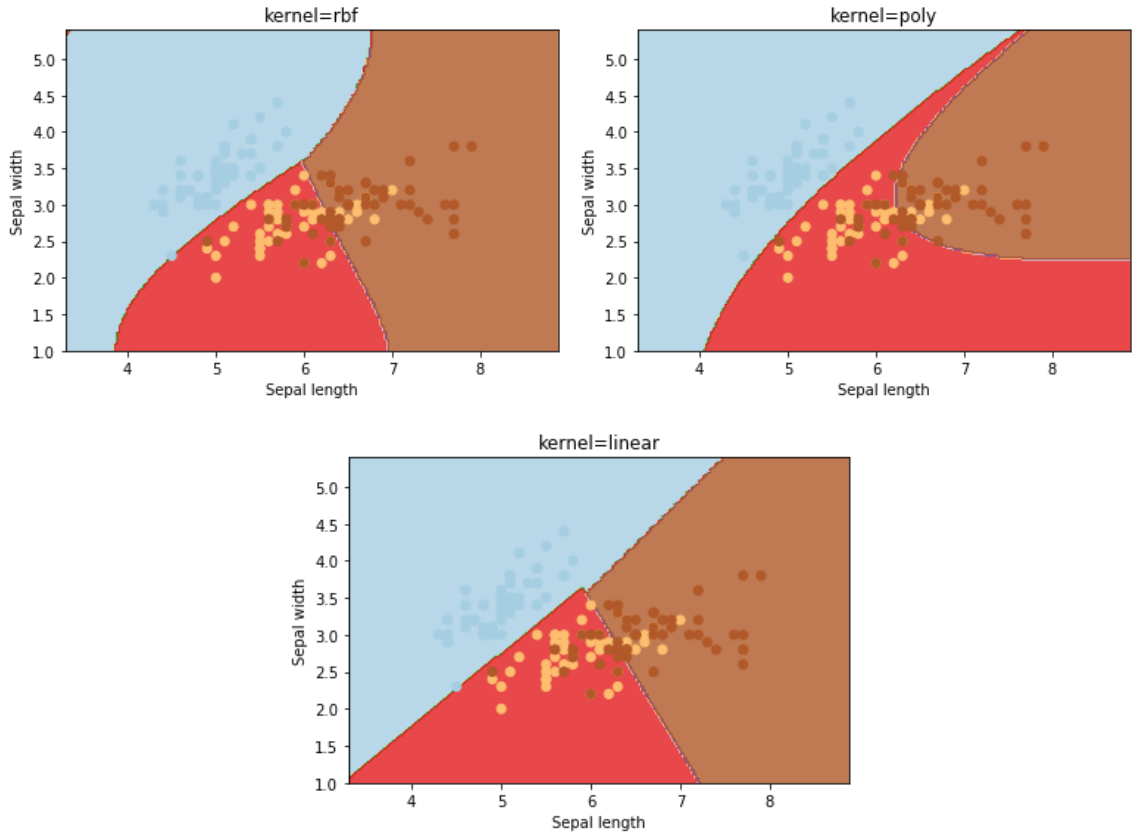
svm.SVC()

A Support Vector Machine esetében három paramátert vizsgáltam a GridSearch segítségével, a *C, gamma* és *kernel* paramétereket. Ezeken felül sok más paramétere is van, de az én esetemben ezek voltak relevánsak és érdekesek.

A *gamma* paramétert a nemlineáris kernelekkel lehet használni. Minél nagyobb értéket vesz fel, annál pontosabban fog illeszkedni a tanuló adatkészlethez.

A *C* egy szabályozási paraméter, a szabályozás mértéke fordítottan arányos *C* értékével. Szigorúan pozitívnak kell lennie. Ha túl nagy értéket adunk neki az érzékenyen érinti a túltanulást.

A következő paraméter a *kernel*, ez öt sztring értéket vehet fel. Feladata kiválasztani az adatok elkülönítéséhez használt hipersík típusát. Az alapértelmezett *'rbf'* és a *'poly'* nevezetű érték egy nemlineáris hipersíkot képeznek, amíg a *'linear'* egy lineáris hipersíkot használ.



A 25. képen látható, hogyan osztanak fel a különböző kernelek egy minta adatkészletet. [52]

Én a kerneleknél csak az *'rbf'*-t használtam, mivel így is nagyon hosszú volt a futási idő, a többi esetében pedig még hosszabb idővel kellett volna számolni. Ez az algoritmus

különbözik a többitől abban, hogy nincs *predict_proba* függvénye, azaz nem tudtam valószínűség becslést végezni rajta, csak a sima becslést, ami egy eredményt ad. Ezért ez nem a legjobb algoritmus az ilyen típusú feladatokhoz. [53]

Ezzel az algoritmussal értem el a legrosszabb eredményeket. Ez annak is köszönhető, hogy az adatkészletemhez nem ez passzol a legjobban, illetve az is nagy szerepet játszott, hogy ehhez az algoritmushoz nem lehet a *predict_proba*-t használni.

C	***gamma***	**Kaggle eredmény**	**GridSearch CV**	**GridSearch pontosság**
0.1	0.1	0.03906	2	0.037
1	0.001	**0.04058**	3	0.05
10	0.05	0.03971	6	0.052
10	0.1	0.03947	10	0.062

MLPClassifier()

A Multi-layer Perceptron esetében is három paraméterre használtam a GridSearch adta lehetőségeket, az *alpha, hidden_layer_sizes, learning_rate* és *solver*.

A *hidden_layer_sizes* adja meg a neuronok számát a rejtett rétegekben. Ez egy számsort vár értékként.

Az *activation* paramétert a GridSearch mindegyik esetben *'relu'*-ra állította, ezért nem szerepeltettem külön a táblázatban. Ez a rejtett rétegek aktivitási függvénye. Négy értéket vehet fel. *'identitiy'*, ha inaktív az aktiválás, hasznos a lineáris implementációkhoz. Visszatérési értéke: f(x)=x. A következő a *'logistic',* ami szigmoid függvényt használ. Visszatérési értéke: f(x)=1/1+exp(-x)). A *'tanh'* egy hiperbolikus tangens függvényt használ, aminek a visszatérési értéke: f(x)=tanh(x). Végül az általam is használt és alapértelmezett *'relu'* egy finomított lineáris egységfüggvény, aminek a visszatérési értéke f(x)=max(0,x).

Solver paraméter se szerepel a táblázatomban, mivel az összesnél az *'sgd'* értéket adtam neki. Ez oldja meg a súlyoptimalizálást az algoritmusban. A Scikit-Learn-ben három lehetséges értéke létezik. Az alapértelmezett *'adam'*, ami egy sztochasztikus gradiens-alapú optimalizáló, nagy adatkészleten ezt érdemes használni. Az *'lbfgs'* egy Newton alapú optimalizáló, ez az *'adam'*-val ellentétben kis adatkészleten teljesít jobban. Az *'sgd'*, ami egy sztochasztikus gradiensre utal. Az *alpha* a szabályozási paraméter.

A *learning_rate* csak *'sgd'* esetén használatos, egyébként figyelmen kívül hagyja az algoritmus. Ez a tanulási terv a súlyfrissítésekhez. Ez lehet 'invscaling', ami fokozatosan csökkenti a tanulás sebességét. Lehet 'constant', ami állandó tanulási arányt jelent, amit a learning_rate_init-vel lehet beállítani. És lehet 'adaptive', ami a tanulási arányt konstans tartja, amíg a tanuló adatok vesztesége folyamatosan csökken. Én az utóbbi kettőt felváltva használtam. [54]

Ezek az eredmények az előzőhöz képest jobbak, de az döntési fákhoz és véletlen erdőkhöz képest nagyon rosszak lettek. A legjobb is csak 6,9%.

alpha	***hidde_layer_sizes***	***learning_rate***	**Kaggle eredmény**	**GridSearch CV**	**GridSearch pontosság**
0.0001	(50,100,50)	constant	0.06837	2	0.037
0.0001	(100,)	adaptive	**0.06913**	3	0.036
0.001	(50,50,50)	constant	0.06862	10	0.041
0.05	(50,50,50)	adaptive	0.06837	20	0.039

Fejlesztési lehetőségek

Mint minden területen, itt sem lehet teljesen 100%-os tudásra szert tenni és mindig van hova fejlődni, így a feladatomat is lehetne még tovább fejleszteni.

A legegyszerűbb fejlesztési lehetőség, megvizsgálni, hogyan alakulnak az eredmények több, vagy kevesebb oszlopnyi adattal. Megtalálni azt az összetételt, ami a legjobb az algoritmusoknak, akár minden egyes gépi tanulási módszernél változtatva. Ebbe beleértem azt is, hogy ha szükséges még több új oszlopot létrehozni. Vagy akár összesen két oszloppal próbálkozni.

A tanuló adatkészlet 37 millió sort tartalmaz. Én ebből csak 1 milliót használtam, mivel leredukáltam a 37 milliót azzal, hogy kitöröltem az olyan oszlopokat, amik null adatot tartalmaztak, és csak azokat hagytam meg, amiknél történt foglalás. Érdekes eredmények jöhetnek ki több sor meghagyásával. Például nem kitörölni a lyukas oszlopokat, hanem kitölteni az átlag adatokkal vagy meghagyni azokat a sorokat is, ahol nem történt foglalás.

Ezek után lehet még több paramétert vizsgálni, más és más kombinációkkal. A legideálisabb paramétereket lehet nagyobb intervallumban is keresni, akkor is javulhatnak az eredmények.

Ami az én eredményeimen változtathatott volna az a végső 5 darab eredmény kiválasztása. Mint később kiderült számomra nem csak az számít, hogy az ötben benne legyen a jó válasz, de az is, hogy hányadik helyen. Minél korábban szerepel a jó válasz, annál nagyobb pontosságot lehet elérni. Így az öt legjobb valószínűség kiválasztását lehet még finomítani azzal, hogy ha több ugyanolyan mértékű valószínűség jött ki, akkor még azokat is külön meg lehetne vizsgálni.

Lehet még játszani a jellemzők és címkék kiválasztásával is. Ezekhez a változtatásokhoz a Scikit-learn oldalán nagyon sok segítő algoritmus található, mint az általam is használt GridSearch.

Természetesen fejlesztési lehetőség más módszereket is kidolgozni, kipróbálni. Láttam olyan megoldásokat is, amikor csak két oszlopot vizsgálva nagyon jó eredményt sikerült elérni. Azáltal, hogy azt a két oszlopnyi adatot jól csoportosította, ehhez évek tapasztalata szükséges és sok hasonló versenyben való részvétel.

Összefoglalás és konklúzió

Összességében a termékajánló rendszerek nagyon fontosak és a cégek bevételi forrását nagyban tudják befolyásolni, ha jó az algoritmus, amit használnak. Az egésznek a mesterséges intelligencia, azon belül pedig a gépi tanulás az alapja, amit ma már nagyon sok esetben alkalmaznak. A dolgozatban kifejtettem, hogy a termékajánló rendszerek mögött mi található, és hogyan lehet létrehozni. Aminek több ága is lehetséges, mint az együttműködési szűrők, a tartalom alapú szűrők és ezeknek is különböző lehetőségeit igyekeztem a tőlem telhető legjobban bemutatni.

A dolgozatom második részében kipróbáltam gépi tanulási algoritmusokat, több paraméterrel és azzal becsültem meg, hogy a felhasználó melyik hotel klasztert fogja választani.

A legjobb eredmény, amit elértem 20%-os pontosság lett. Így körülbelül az 1600. helyen végeztem volna a versenyen, közel 2000 versenyzőből. Az első helyezettnek 60% lett a pontossága.

A téma, amit boncolgattam nagyon érdekes ága az informatikának, örülök, hogy megismerhettem ezt a részét is, mivel ezelőtt nem igazán tudtam, hova tenni ezeket a rendszereket. Úgy voltam vele, mint ahogy gondolom sok más ember is, hogy láttam, hogy van és gondoltam, hogy működik, de nem tudtam, hogy mi lehet mögötte. Nagyon érdekes és hasznos cikkeket találtam a témával kapcsolatban, amik biztos vagyok benne, hogy a későbbiekben is hasznosak lesznek.

El tudom képzelni, hogy a jövőben én is használni fogom a fent említett módszereket, akár munkámban, akár saját fejlődés érdekében.

Örülök, hogy beleáshattam magam a gépi tanulás, és az ajánlórendszerek működésébe.

Irodalomjegyzék

[1] G. Takács, I. Pilászy, B. Németh és D. Tikk, „Scalable Collaborative Filtering Approaches,” 2009. [Online]. Available: http://www.jmlr.org/papers/volume10/takacs09a/takacs09a.pdf. [Hozzáférés dátuma: 23. április 2019.].

[2] R. Collobert, S. Bengio és J. Mariéthoz, „Torch: a modular machine learning software library,” 30. október 2002. [Online]. Available: http://www.idiap.ch/ftp/reports/2002/rr02-46.pdf. [Hozzáférés dátuma: 23. április 2019.].

[3] „Netflix Prize,” Netflic Inc., 1997-2009. [Online]. Available: https://www.netflixprize.com/leaderboard.html. [Hozzáférés dátuma: 23. április 2019.].

[4] R. Singh, K. Chuchra és A. Rani, „A Survey on the Generation of Recommender Systems,” 3. március 2017. [Online]. Available: https://pdfs.semanticscholar.org/7c38/fa13e9a843e73ede4a935a10565d2594a249.pdf. [Hozzáférés dátuma: 23. április 2019.].

[5] „kaggle.com,” [Online]. Available: https://www.kaggle.com/rounakbanik/movie-recommender-systems. [Hozzáférés dátuma: 14. április 2019].

[6] F. Ricci, L. Rokach, B. Shapira és P. Kantor B., Recommender Systems Handbook, Springer, 2010, p. 845.

[7] J. Beel, S. Langer, M. Genzmehr és A. Nürnberger, „Persistence in Recommender Systems: Giving the Same Recommendations to the Same Users Multiple Times,” 2013. [Online]. Available: http://docear.org/papers/persistence_in_recommender_systems_--_giving_the_same_recommendations_to_the_same_users_multiple_times.pdf. [Hozzáférés dátuma: 23. április 2019.].

[8] M. Montaner, B. López és J. Lluís de la Rosa, „Developing Trust in Recommender Agents,” [Online]. Available: http://eia.udg.es/arl/ramonycajal/156-montaner.pdf. [Hozzáférés dátuma: 23. április 2019.].

[9] J. A. Konstan és J. Riedl, „Recommender systems: from algorithms,” Springer, 2012.

[10] D. Goncalves, M. Costa és F. M. Couto, „A Flexible Recommendation System for Cable TV,” 2016.

[11] D. Jannach, P. Resnick, A. Tuzhilin és M. Zanker, „Recommender Systems - Beyond Matrix Completion,” 2016.

[12] L. E. Molina Fernández, *Recommendation System for Netflix,* 2018.

[13] „How Netflix’s Recommendations System Works,” Netflix Inc., [Online]. Available: https://help.netflix.com/en/node/100639. [Hozzáférés dátuma: 23. április 2019.].

[14] „Kaggle.com,” Kaggle Inc., [Online]. Available: https://www.kaggle.com/netflix-inc/netflix-prize-data. [Hozzáférés dátuma: 23. április 2019.].

[15] C. C. Aggarwal, Recommender Systems The Textbook, Springer, 2016.

[16] F. Isinkaye, Y. Folajimi és B. Ojokoh, „Recommendation systems: Principles, methods and evaluation,” 2015.

[17] J. W. Grice, „Pearson’s correlation coefficient,” november 2013. [Online]. Available:

http://psychology.okstate.edu/faculty/jgrice/psyc5314/pearsons_r.pdf. [Hozzáférés dátuma: 23. április 2019.].

[18] L. Walker, „Amazon Gets Personal With E-Commerce," *Washington Post,* 8 november 1998.

[19] R. Burke, „Knowledge-based recommender systems," [Online]. Available: https://pdfs.semanticscholar.org/1b50/ff4e5d8420f3ffae7de2a060b5a6fd4b8023.pdf. [Hozzáférés dátuma: 23. április 2019.].

[20] A. Munoz, „Machine Learning and Optimization," [Online]. Available: https://pdfs.semanticscholar.org/7fbb/a79630b5a09dd66ab13f00c3aefaa56cf268.pdf. [Hozzáférés dátuma: 23. április 2019.].

[21] J. Hurwitz és D. Kirsch, Machine Learning For Dummies, John Wiley & Sons, Inc., 2018.

[22] „Felügyelt tanulás," Szegedi Tudományegyetem, [Online]. Available: http://www.inf.u-szeged.hu/~ihegedus/teach/04-KNN.pdf. [Hozzáférés dátuma: 23. április 2019.].

[23] N. Kumar, L. Zhang és S. Nayar, „What is a Good Nearest Neighbors Algorithm for Finding Similar Patches in Images," [Online]. Available: http://www1.cs.columbia.edu/CAVE/publications/pdfs/Kumar_ECCV08_2.pdf. [Hozzáférés dátuma: 23. április 2019.].

[24] „towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/knn-k-nearest-neighbors-1-a4707b24bd1d. [Hozzáférés dátuma: 14. április 2019].

[25] L. Rokach és O. Maimon, „Decision Trees," in *Data Mining and Knowledge Discovery Handbook*.

[26] „Decision Trees," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/tree.html. [Hozzáférés dátuma: 23. április 2019.].

[27] L. Breiman, „Random Forests," január 2001. [Online]. Available: https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf. [Hozzáférés dátuma: 23. április 2019.].

[28] C. Tomasi, „Random Forest Classifiers," [Online]. Available: https://pdfs.semanticscholar.org/3052/b67a414adecd7545dad635ca3c8495be9314.pdf . [Hozzáférés dátuma: 23. április 2019.].

[29] „medium.com," [Online]. Available: https://medium.com/@williamkoehrsen/random-forest-simple-explanation-377895a60d2d. [Hozzáférés dátuma: 25. április 2020.].

[30] R. E. Schapire, „Explaining AdaBoost," [Online]. Available: http://rob.schapire.net/papers/explaining-adaboost.pdf. [Hozzáférés dátuma: 23. április 2019.].

[31] „towardsdatascience.com," [Online]. Available: https://towardsdatascience.com/understanding-adaboost-2f94f22d5bfe. [Hozzáférés dátuma: 25. április 2020.].

[32] A. Ben-Hur és J. Weston, „A User's Guide to Support Vector Machines," [Online]. Available: http://pyml.sourceforge.net/doc/howto.pdf. [Hozzáférés dátuma: 23. április 2019.].

[33] „cs.nyu.edu," [Online]. Available: https://cs.nyu.edu/faculty/davise/ai/linearSeparator.html. [Hozzáférés dátuma: 4. április 2019].

[34] A. Krenker, J. Bešter és A. Kos, „Introduction to the Artificial Neural Networks," [Online]. Available: http://cdn.intechweb.org/pdfs/14881.pdf. [Hozzáférés dátuma: 23. április 2019.].

[35] „Kaggle.com," Alphabet Inc., [Online]. Available: https://www.kaggle.com/c/expedia-hotel-recommendations/overview/description. [Hozzáférés dátuma: 18. április 2020.].

[36] „Kaggle.com," Alphabet Inc., [Online]. Available: https://www.kaggle.com/c/expedia-hotel-recommendations/overview/evaluation. [Hozzáférés dátuma: 18 április 2020.].

[37] „Kaggle.com," Alphabet Inc., [Online]. Available: https://www.kaggle.com/c/expedia-hotel-recommendations/data. [Hozzáférés dátuma: 18. április 2020.].

[38] „anaconda.com," Anaconda Inc., [Online]. Available: https://www.anaconda.com/distribution/. [Hozzáférés dátuma: 21. április 2020.].

[39] „jupyter.org," Project Jupyter, [Online]. Available: https://jupyter.org/. [Hozzáférés dátuma: 21. április 2020.].

[40] „pandas.pydata.org," [Online]. Available: https://pandas.pydata.org/. [Hozzáférés dátuma: 21. április 2020.].

[41] „numpy.org," [Online]. Available: https://numpy.org/. [Hozzáférés dátuma: 21. április 2020.].

[42] „matplotlib.org," [Online]. Available: https://matplotlib.org/. [Hozzáférés dátuma: 21. április 2020.].

[43] „seaborn.pydata.org," [Online]. Available: https://matplotlib.org/. [Hozzáférés dátuma: 21. április 2020.].

[44] „azure.microsoft.com," Microsoft Corporation, [Online]. Available: https://azure.microsoft.com/en-us/free/students/. [Hozzáférés dátuma: 24. február 2020.].

[45] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html. [Hozzáférés dátuma: 22. április 2020.].

[46] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/neighbors.html. [Hozzáférés dátuma: 21. április 2020.].

[47] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier.kneighbors. [Hozzáférés dátuma: 2020.].

[48] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html. [Hozzáférés dátuma: 21. április 2020].

[49] „medium.com," [Online]. Available: https://medium.com/@mohtedibf/indepth-parameter-tuning-for-decision-tree-6753118a03c3. [Hozzáférés dátuma: 21. április 2020].

[50] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. [Hozzáférés dátuma: 21. április 2020.].

[51] „scikit-learn.org," [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html. [Hozzáférés dátuma: 21. április 2020.].

[52] „medium.com," [Online]. Available: https://medium.com/all-things-ai/in-depth-

parameter-tuning-for-svc-758215394769. [Hozzáférés dátuma: 8. április 2020.].

[53] „scikit-learn.org,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC. [Hozzáférés dátuma: 21. április 2020.].

[54] „scikit-learn.org,” [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html. [Hozzáférés dátuma: 21. április 2020.].

[55] P. Langley és S. Sage, „Induction of Selective Bayesian Classifiers,” [Online]. Available: https://arxiv.org/ftp/arxiv/papers/1302/1302.6828.pdf. [Hozzáférés dátuma: 23. április 2019.].