Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar

Babák Botond

ENERGIASZOLGÁLTATÁSOK FELHASZNÁLÓINAK FOGYASZTÁSVIZSGÁLATA GÉPI TANULÁSI MÓDSZEREKKEL

KONZULENS

Dr. Toka László

BUDAPEST, 2021

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott **Babák Botond**, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2021. 05. 22.

.....

Babák Botond

Összefoglaló

Jelen szakdolgozat célja, hogy az energiaszolgáltatók által begyűjtött adatokból azok felhasználóinak fogyasztását vizsgáljam. A vizsgálat során célom az, hogy a fogyasztási adatokból minél pontosabban meghatározzam gépi tanulás segítségével azt, hogy a felhasználóbázisból mely fogyasztók károsítják meg potenciálisan a szolgáltató céget nem szabályos mintavételezés útján.

Ahhoz, hogy ezt elérjem több lehetséges módszer alkalmazok. Különböző algoritmusok hatásfokát vizsgálom megegyező adathalmazokon. Az adathalmaz tökéletesítését, azon végzett változtatások befolyását vizsgálom, és próbálom megtalálni a legalkalmasabb kombinációt. Mindezt az üres adatok kezelésével, a feleslegesnek tűnő adatok elvetésével, az adathalmaz méretének csökkentésével, adatbővítő algoritmusok használatával, illetve új adathalmaz létrehozásával érem el.

A megfelelő adathalmaz létrehozása után az általam vizsgált három algoritmus – logistic regression, random forest illetve XGBoost – optimalizálásával próbálok minél jobb eredményeket elérni, majd a legjobbnak tűnő modelleket összehasonlítani aszerint, hogy melyik lehet a legalkalmasabb a fraud detection feladatára az energiaszolgáltatók körében.

Abstract

The goal of this thesis is to inspect the data collected by energy suppliers about their users consumption. The main goal is to make accurate decisions from the consumption data via machine learning methods about our consumers being fradulent or not.

To reach this goal I am using different methods. To optimize the algorithms I am using, first I make modifications on the used dataset, and inspect the influence of these modifications on the results. On the dataset I try to achieve this by using different methods to fill NaN (Not a Number) values, deleting outliers and resizing the used dataset, using data augmentation methods and making a whole new dataset.

After I create the seemingly best dataset I am optimizing the three machine learning algorithms – logistic regression, random forest and XGBoost - I chose for the task to achieve better and better results and to decide which model seems the best for the task of fraud detection in the field of energy suppliers.

1. Bevezetés

1.1. Szabálytalan mintavételezés

Az energiaszolgáltatók mint víz-, elektromos áram- és gázszolgáltatók körében jelentős veszteséget okoznak az úgynevezett nem műszaki veszteségek csoportja, ami a nem a cég által a terjesztés/„szállítás" folyamán felmerülő műszaki és/vagy technikai probléma által keletkezett veszteségek. Ezek közé tartoznak többek között a hibás számlázások, a nem kiszámlázott szolgáltatás, az adott mérők hibás működése, az infrastruktúra hiánya, valamint amire a dolgozat koncentrál, a lopások, szolgáltatással kapcsolatos csalások, szabálytalan vételezési eljárások csoportjai.

A szabálytalan vételezés során az elkövető a szolgáltató cégnek szándékosan kárt, veszteséget okoz, és/vagy saját magának hasznot hoz az adott közüzemi termék „beszerzésének" folyamata során. Ez a szolgáltató cégnek, cégeknek nyilvánvalóan nem érdekük, ezért megakadályozásuk a cél, hogy maximalizálják profitjukat.

A gyakorlatban többféleképpen is lehetséges a szabálytalan vételezés. Leggyakoribbak közé tartozik a mérőóra teljes kikerülése, ami gáz- és vízszolgáltatások esetén szivárgáshoz vezethet, ezzel előbbi esetében akár veszélyes helyzetet is teremtve, valamint a szolgáltató cég berendezéseiben, illetve a környezetében is kárt téve. Egy másik módszer során a csalók nem kerülik ki az adott mérőórát, hanem azon keresztül folyik az elektromos áram, gáz vagy víz, de az egyén a mérőóra módosításával lelassítja annak mérőjét vagy teljesen megállítja azt, így a leolvasása során hamis adatokat mutat.

Ezek közös tulajdonsága, hogy mivel a fogyasztást minden felhasználó esetében ugyanúgy mérjük (egy csaló esetében is), az adatbázisba bekerülő számadatok alapján következtethetünk egy-egy felhasználó mivoltára. A technológia fejlődésével erre egyre több lehetőségünk van, a hasonló ellenőrzéseket gépekkel is elvégezhetjük, amire egyik megoldás a dolgozatban is vizsgált gépi tanulási módszerek segítségével lehetséges.

1.2. Gépi tanulás

A gépi tanulási eljárások a mesterséges intelligencia egy ágazata. Gépi tanulási algoritmusok során a gép az adatbázisban különböző mintákat azonosít. Ezeknek a felismert mintáknak a segítségével modelleket készít.[1]

A felállított modellek alapján a gépi tanulási módszerek képesek előrejelzést végezni, valamint ami a fraud detection terén segítségünkre lesz abban, hogy a különböző adatsorokat, amik helyzetünkben egy-egy felhasználót jelentenek, osztályokba tudjuk sorolni, ezzel a mi esetünkben megállapítva, hogy egy-egy felhasználóhoz tartozó fogyasztási adatok utalnak-e arra, hogy az illető csaló vagy sem.[1]

Mivel gépi tanulási módszerek között is vannak különbségek és egyes feladatokra különböző hatásfokkal rendelkeznek ezért a szakdolgozatom során a hasonló osztályozási feladatra leggyakrabban használt, illetve legígéretesebbnek tűnő algoritmusokkal elért eredményeket vizsgálom.

1.3. Adathalmaz

A témában az adott feladatra csekély számú adathalmaz található, ugyanis ahhoz, hogy a fraud detection feladatának megfeleljenek, illetve a modell betanítására is alkalmasak legyenek, és értékelhető eredményt és következtetést tudjunk levonni a tanítás után, több szempontnak is eleget kell tenniük. Az adathalmazunknak megfelelően nagy mennyiségű adatot kell tartalmaznia. Nem elég, hogy a felhasználóink sokan legyenek, de minden egyes fogyasztóhoz egy elég nagy időintervallumban kell fogyasztási adatokat rendelnünk.Mindezek mellett ahhoz, hogy döntéseket tudjon hozni, hogy a teszt adathalmazban melyik felhasználóhoz tartozó adatsor nevezhető a fogyasztási paraméterei alapján csalónak és melyik nem, szükségünk van egy bináris változóra, ami a tanító adathalmazban minden egyes fogyasztó esetében meghatározza ezt a tulajdonságot.

Az energiaszolgáltatásokkal kapcsolatos adathalmazokból rengeteg elérhető, viszont a fraud detection-höz szükséges bináris változó kevés adathalmazban található meg, ugyanis ehhez szükségünk van megfelelő számú, korábban igazolt szabálytalan mintavételezésekre. Mivel az éghajlat miatt szabályszerűen változnak a különböző energiafogyasztások, ezeknek az adatoknak egy régióból is kell kikerülnie.

A témában sokat használt adathalmaz egy a State Grid Corporation of China (SGCC) által kiadott dataset, ami 1035 napon keresztül, 2014. január 1-től 2016. október 31-ig tartalmazza 42372 felhasználónak a fogyasztását.[3]

A szakdolgozatban én is ezt az adathalmazt használtam, viszont mivel ez sem tökéletes, sok adat hiányzik belőle ezért többféleképpen átalakításokat végeztem rajta, hogy a hiányzó adatok ellenére is minél több és pontosabb eredményt érjek el a gépi tanulásos algoritmusokkal. Ezeket az átalakítási módszereket a későbbiekben részletezem. Ennek megfelelően az adathalmazunk a következőképpen néz ki a dátumok sorbarendezése után, ahol a FLAG oszlop a bináris változónk, ami 1 esetén pozitív eset, tehát a hozzá tartozó fogyasztó csaló, 0 esetén negatív eset, tehát nem csaló:

CONS_NO	FLAG	1/1/2014	1/2/2014	…	10/30/2016	10/31/2016
fogyasztói azonosító	1	fogyasztási adat	fogyasztási adat	…	fogyasztási adat	fogyasztási adat
fogyasztói azonosító	1	fogyasztási adat	fogyasztási adat	…	fogyasztási adat	fogyasztási adat
…	…	…	…	…	…	…
fogyasztói azonosító	0	fogyasztási adat	fogyasztási adat	…	fogyasztási adat	fogyasztási adat
fogyasztói azonosító	0	fogyasztási adat	fogyasztási adat	…	fogyasztási adat	fogyasztási adat

1.3.1. táblázat: A használt adatbázis felépítése

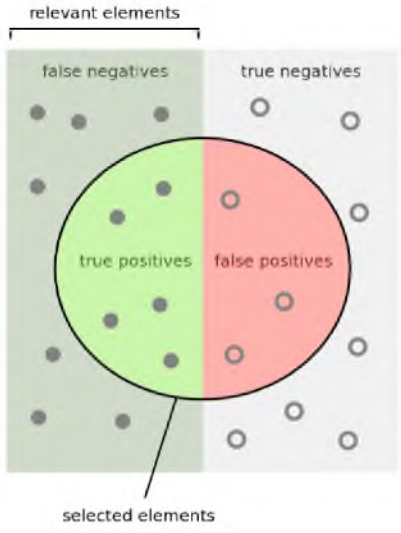
2. Osztályozási minőség mérőszámai

Egy osztályozási feladatnak több lehetséges mérőszáma is van, amiket kereshetünk, ugyanakkor egyes esetekben félre is vezethetnek minket. Ennek oka, hogy a feladat mivoltából adódóan, illetve a rendelkezésünkre álló adathalmaz egyes tulajdonságai alapján egy-egy érték annak ellenére magas értéket érhet el, hogy valójában nem funkcionál megfelelően az adott helyzetben. Ezeket az értékeket a következőkben angol neveiken fogom nevezni a könnyebb elkülönítés érdekében, mivel vannak értékek, amik a magyar nyelvben azonos jelentéssel bírnak, illetve a szaknyelvben is ritkán található magyarra fordítva.

2.1. Pontosság

Az egyik ilyen és talán legkézenfekvőbb érték a pontosság, vagy idegen nyelven accuracy. Bináris és többosztályos osztályozók pontosságának megállapítására is használható, könnyen értelmezhető és előállítható mérőszám. A pontosság kiszámítása a helyesen megjósolt eredmények összegének és az összes eredmény hányadosa, tehát:[4]

$$Accuracy = \frac{(True\ Positive + True\ Negative)}{(True\ Positive + True\ Negative + False\ Positive + False\ Negative)}$$



2.1.1. ábra: A pontosság egyszerűen szemléltetve[5]

Helyzetünkben a pontosság egy olyan érték, ami hamis eredményeket szolgáltat nekünk. Mivel a használt adathalmaz, és ha feltételezhetjük, hogy nagyságrendekkel többen járnak el

tisztességesen az energiaszolgáltatókkal szemben, mint ahány csaló van, a való életben is egy meglehetősen úgynevezett imbalanced, azaz kiegyensúlyozatlan adathalmazunk van. Mivel az egyik osztályunkból nagyságrendekkel kevesebb van, mint a másikból, így ha az modell minden adatot a true negative osztályba (esetünkben a tisztességesen eljáró fogyasztók) sorol, a pontosság 0,9, tehát 90% feletti lesz, mindeközben egyetlen csalót sem találtunk meg, amit egy későbbi példával szemléltetek.[6]

A legtöbb, alapvető beállításokkal futtatott modell 90-92% accuracy eredményeket ért el. Ezek közül példaként a logistic regression, 2014. január 1-től 2016. október 31-ig az üres adatokat az oszlopok átlagaival feltöltve 91,5%-ot, ami nagyon ígéretesnek tűnhet. Viszont ha megnézzük a számokat kiderül, hogy a 8475 felhasználót tartalmazó teszt adathalmazból 7671 nem csalót és 82 csalót sorolt a megfelelő osztályokba, viszont 46 nem csalót és 676 csalót rossz osztályokba helyezett el. Mindez azt jelenti, hogy modellünk a csalók mindössze 10,8%-át ismerte fel.

2.2. Megbízhatóság

A megbízhatóság, azaz precision nem az eredmények teljes halmazát veszi figyelembe, hanem csak a pozitív értékeket. Ennek segítségével könnyebben képet kaphatunk, hogy az adott osztályoknak hány százaléka ténylegesen pozitív eset.[4]

$$Precision = \frac{True\ Positive}{(True\ Positive + False\ Positive)}$$

A megbízhatóság már helyzetünkben egy használhatóbb értéket ad az pontossághoz képest, viszont mivel fő célunk a fraud detection, ezért leginkább a tényleges csalók csoportjára szeretnénk koncentrálni.

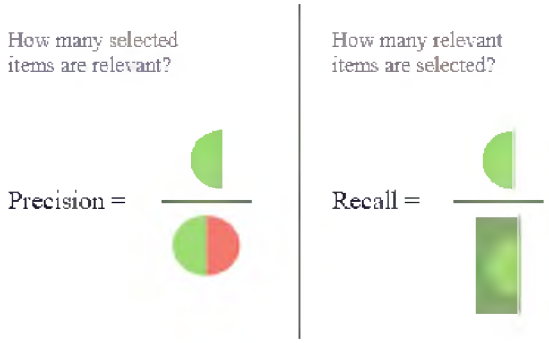
A csalófelderítés esetében a precision megadja, hogy a pozitív, tehát csaló osztályba sorolt felhasználóink hány százaléka került ténylegesen a megfelelő osztályba.

2.3. Érzékenység

A probléma megoldásához egyik legfontosabb érték az érzékenység, más néven recall lesz. A megbízhatósággal ellentétben az érzékenység nem csak a pozitív értékekre koncentrál, hanem azokra a fogyasztókra, akiket ki akarunk szűrni, tehát a csalók hány százaléka lett helyesen megjósolva.

Annak érdekében, hogy ezt elérjük több felhasználó ellenőrzése szükséges, ami egy nem tökéletes modell esetében több, ténylegesen nem csaló közelebbi megvizsgálását eredményezi, ami miatt a precision értékünk csökkenni fog, viszont ha a két értéket megfelelően egyensúlyba tudjuk hozni a szolgáltató cég rosszul kiszűrt felhasználók ellenőrzési költségei ellenére sem veszít a modellen.[4]

$$Recall = \frac{True\ Positive}{(True\ Positive + False\ Negative)}$$



2.3.1. ábra: A megbízhatóság és az érzékenység egyszerűen szemléltetve a 2.1.1.-es ábra alapján[7]

2.4. F-score

Az F-score (másnéven F1 score) a precision és a recall súlyozott átlaga. Mivel leginkább ezekre az értékekre koncentrálunk, az accuracyt mint mérőszámot elvethetjük, és ez egy jó mérték az összehasonlítás terén.[8]

$$F1\ score = \frac{2 * Precision * Recall}{(Precision + Recall)}$$

2.5. Receiver Operating Characteristic görbe

A Reveiver Operating Characteristic, innentől ROC görbe ugyan nem egy mérőszám, de az osztályozási feladatok szemléltetésére tökéletesen alkalmazható módszer, aminek segítségével egy a témában nem annyira jártas egyén is következtetéseket vonhat le. Ennek a számszerűsített változata az AUC (Area Under the Curve), ami a függvény alatt elhelyezkedő terület nagysága. Az Area Under the Curve, mint azt a nevéből is látni lehet egy általános fogalom, amit több adatelemzésre használatos görbével összefüggésben is használnak.[9]

A ROC bináris osztályozási problémák terén hasznos eredményeket tud mutatni. Az x tengelyen a false positive arányt, az y tengelyen pedig a true positive arányt mutatja.Kiegyensúlyozatlan adathalmazok esetén ugyan a ROC görbe tud félrevezető is lenni, ugyanakkor két görbe között egy modell fejlődése ennek ellenére jól látható.[9]

2.6. Precision-Recall görbe

A Precision-Recall görbe kiegyensúlyozatlan adathalmazok terén használandó szemléltetési módszer. A görbe a megbízhatóság és az érzékenység közötti egyensúlyt mutatja. Mint a ROC görbénél itt is a függvény alatti terület a számszerű meghatározása a modellünknek. Magas precision és magas recall esetén az AUC is magas lesz. [10]

2.7. Igazságmátrix

Mint az ezelőtt felsorolt két görbe sem egy mérőszám, úgy az igazságmátrix, azaz confusin matrix sem, viszont jól szemlélteti az eredményeink alakulását. Az igazságmátrix lényegében bináris osztályozás esetén egy kétszer kettes mátrix, amiben a következő értékek helyezkednek el:[11]

- true positive (valós pozitív), tehát azok a felhasználók, akiket a modell pozitív esetként osztályozott és valójában pozitív esetek voltak,
- false positive (hamis pozitív), tehát azok a felhasználók, akiket a modell pozitív esetként osztályozott, de valójában negatív esetek voltak,
- true negative (valós negatív), tehát azok a felhasználók, akiket a modell negatív esetként osztályozott és valójában negatív esetek voltak
- és false negative (hamis negatív), tehát azok a felhasználók, akiket a modell negatív esetként osztályozott, de valójában pozitív esetek voltak.

Ezek elrendezése az igazságmátrixban a következőképpen néz ki:[11]

true negative	false positive
false negative	true oositive

2.7.1. táblázat: Igazságmátrix celláinak jelentése

3. Algoritmusok

A feladat során egy-egy felhasználóhoz tartozó fogyasztási adatokról el kell döntenünk, hogy az adott egyén csaló-e vagy sem. Ennek egyértelműen két lehetséges kimenetele lehet, tehát egy bináris osztályozási problémáról beszélhetünk. Ennek megfelelően logikusan olyan algoritmusokat használtam, amik a bináris osztályozás feladatának megfelelnek, jól teljesíthetnek benne.

A gépi tanulási algoritmusokat csoportokra oszthatjuk tanulási módszereik alapján. Ez lényegében három csoportot eredményez:[12]

- felügyelt tanulás
- felügyelet nélküli tanulás
- félig felügyelt tanulás.

Ezek között a különbség az adatsorok címkéiben van. Míg a felügyelt tanulás esetében az adatsoraink, jelen esetben felhasználóink egytől-egyig fel vanna címkézve, mint fraud vagy nem fraud, a felügyelet nélküli tanulás esetén ez az adat nem áll rendelkezésünkre, a félig felügyelt tanulás esetében pedig a középút, egyes adatsorok rendelkeznek címkével, míg mások nem. Hasonló feladatok esetében leggyakrabban az első, felügyelt tanulási algoritmusok alkalmazása célszerű.

3.1. Logisztikus regresszió

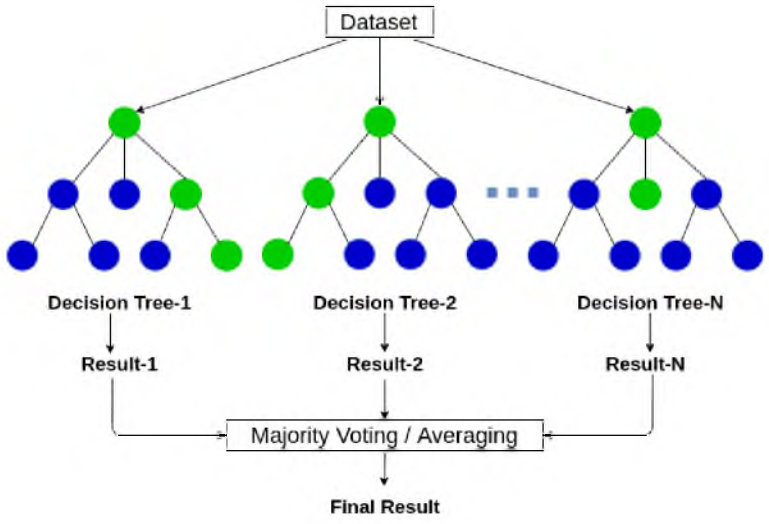
A logisztikus regresszió (logistic regression) annak ellenére, hogy nevéből adódóan regresszióra alkalmas, sokkal inkább klasszifikációs problémákra használt. A modell a lineáris regresszió (linear regression) egy továbbfejlesztett változata, ezért az elnevezés.[13]

A lineáris regresszióval ellentétben a logisztikus regresszió képes az kiugró értékekkel (outlierek) bánni. A logistic regression egy logisztikus függvényként ábrázolható, ami egy szigmoid függvény nulla és egy közötti értékekkel. A logsitic regression osztályozási algoritmus kifejezetten alkalmas bináris osztályozási feladatok esetében, ezért ilyen problémák során széles körben használt. [13]

3.2. Random Forest

A random forest algoritmus egy regresszióra és osztályozásra is alkalmas algoritmus. Általában pontos eredményt ad, és mivel egyszerűen alkalmazható, valamint sokoldalú így széles körben alkalmazzák mind osztályozási, mind regressziós feladatokra.[15]

A random forest működése során, mint azt a neve is sugallja, több döntési fát hoz létre. Ezeket a fákat párhuzamosan alkotja meg, majd miután a paraméterként megadott mennyiséget létrehozta, és elkészül az nevében szereplő „erdő", a már megalkotott fák összegzéseként egy új fát generál, ami az erdő által leggyakoribb döntések szerint épül fel, így optimalizálva a létrehozott fák eredményeit.[16]

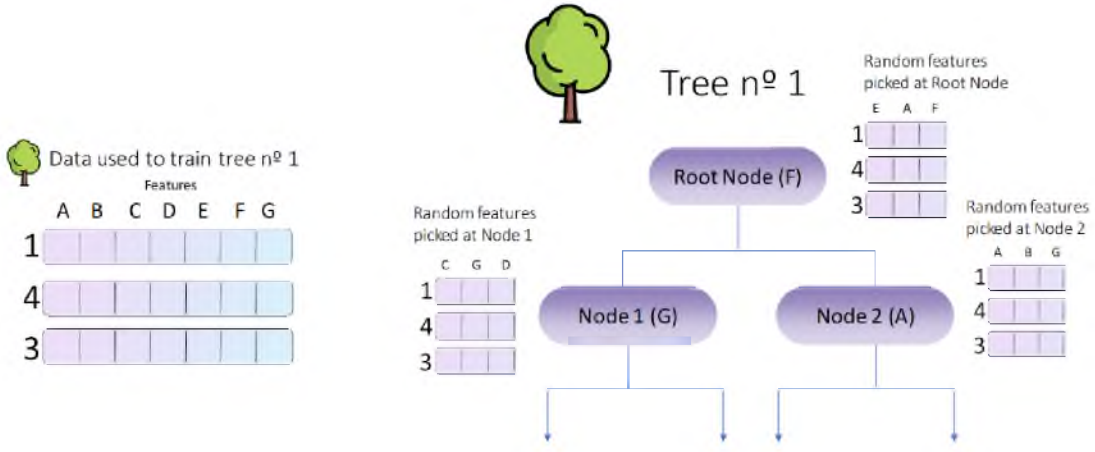


3.2.1. ábra: Random forest algoritmus működése egyszerűen[17]

A random forest a bagging (bootstrap aggregation) módszerét használja. A fák felállítása során véletlenszerűen választja ki, hogy adathalmazunk mely rekordjait használja fel az adott fához. Miután a fák létrejöttek a felépítéskor nem használt adatsorokat a fa tesztelésére használja fel.

3.2.2. ábra: Bagging működése egyszerűen[16]

Mélyebb, sok változóval rendelkező döntési fák gyakran a túltanulás (overfitting) csapdájába eshetnek. A random forest algoritmus annak ellenére, hogy döntési fákat használ, ezt is kezelni tudja, mert az egyes fákat nem minden változó használatával épít fel, hanem azok egy részhalmazával. Ennek köszönhetően a fák lehetnek nagyok is, ennek ellenére az esetek jelentős részében a modell túltanulását elkerülve.[16]



3.2.3. ábra: Random forest algoritmus faépítési folyamata különböző tulajdonságok felhasználásával[16]

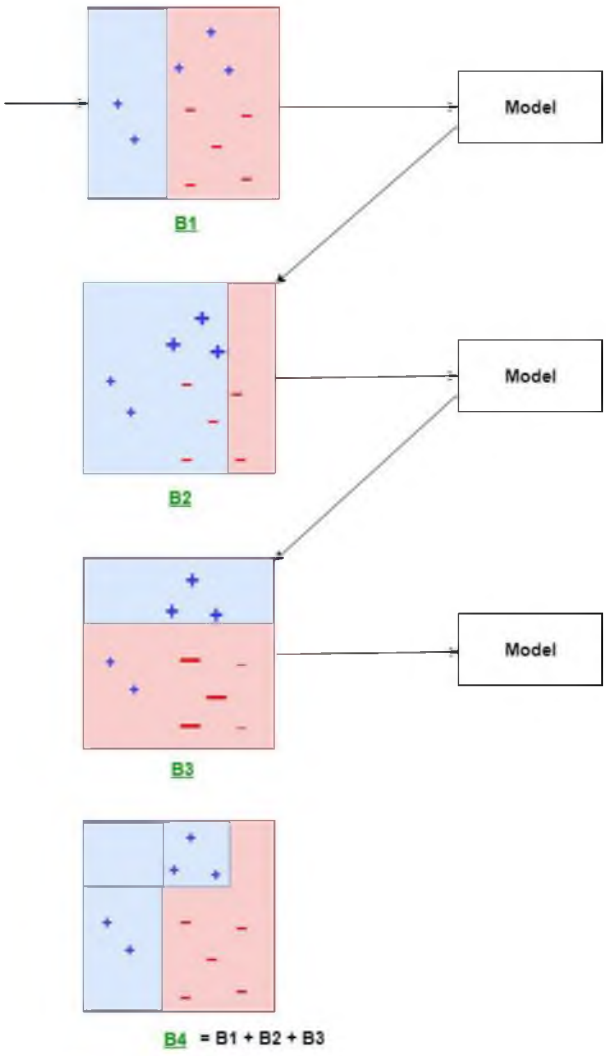
3.3. eXtreme Gradient Boosting

Az eXtreme Gradient Boosting, továbbiakban XGBoost valójában egy open-source könyvtár és algoritmus is, ami a gradient boosting módszer mindhárom legerősebb formáját (Gradient Boosting, Stochastic Gradient Boosting, Regularized Gradient Boosting) skálázhatóan

és pontosan implementálja. Az algoritmus a hatékony modellek mellett a számítási idő és a memóriahasználat optimalizálására lett kifejlesztve.[18]

Az XGBoost a Random Forest algoritmussal ellentétben nem a bagging, hanem a boosting módszerét alkalmazza az osztályozás és regresszió során. A Gradient Boosting során az algoritmus új modelleket hoz létre, de nem párhuzamosan, mint a korábbi algoritmus, hanem egymás után, egymásra építve őket, hogy a soron következő fa az előtte lévő hibáit kijavítva növelje saját hatékonyságát. Ezt addig folytathatja az algoritmus, amíg már nem tud javítani a döntési fákon tovább.[19]

Annak ellenére, hogy az XGBoost a boosting módszerét alkalmazza hatékonysága mégis a párhuzamos számítása miatt tűnik ki más Gradient Boosting megoldások közül. Az algoritmus egymásra épülő fákat hoz létre, viszont az egyes fák felépítése során párhuzamos számítást alkalmaz mind az egyes változókhoz tartozó statisztikák, mind pedig a fában lévő elágazások, döntések létrehozása során.[20]



3.3.1. ábra: A boosting működése egyszerűen[19]

4. Változtatások az adathalmazon

Az adathalmaz közel sem volt tökéletes, viszont hasonló adathalmazok hiánya miatt ennek tökéletesítése, fejlesztése volt a legkézenfekvőbb megoldás, ezért több fajta változtatást is eszközölni kellett rajta, hogy minél pontosabb eredményeket kapjak.

4.1. Értékek feltöltése

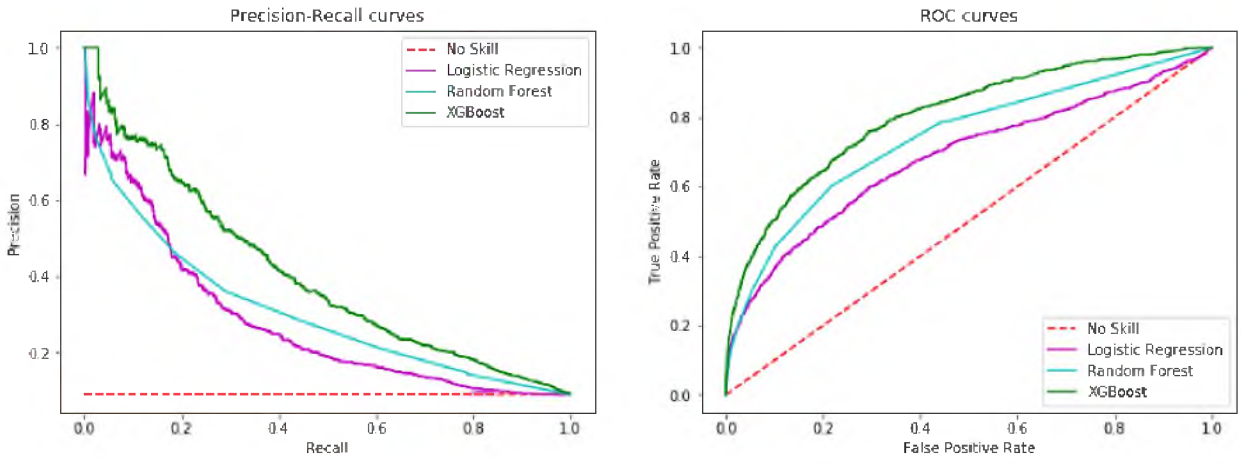
Az adathalmaz nagysága miatt egyértelműen előfordultak NaN (Not a Number) értékek. Mivel nem minden felhasználó fogyasztását egy dátumtól jegyezték, így az üres értékeket először el kellett tüntetni, hogy a választott algoritmusok gond nélkül lefuthassanak.

Mivel jelentős mennyiségű adat hiányzik az adathalmazból, ezért anélkül, hogy a meglévő adatokból vesztenénk, adatokat tudunk generálni az adathalmazunk NaN értékeinek helyére így nem eldobva meglévő adatsoraink nagyrészét. Ehhez első nekifutásra minden oszlop hiányzó adatait az adott oszlop átlagértékével töltjük fel.

A következő eredmények minden algoritmus alapbeállításaival történő futtatás során keletkeztek az 5. fejezet kezdetéig.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,55	0,78	0,57	7671	46
				676	82
Random Forest	0,56	0,76	0,58	7653	64
				667	91
XGBoost	0,55	0,84	0,57	7690	27
				675	83

4.1.1. táblázat: Elért relevánseredmények az üres értékek oszlopátlaggal való feltöltése után



4.1.1. ábra: ROC és Precision- Recall görbe az üres értékek oszlopátlaggal való feltöltése után

Mivel számunkra a recall érték fontosabb, mint a precision, és recallunk csak 0,55 körüli értéket ér el mindhárom algoritmus terén, ezért leginkább ennek fejlesztése a fő prioritás, hiszen a random forest algoritmus esetében például az igazságmátrixunk a következőképpen nézett ki:

True Negative	False Positive
False Negative	True Positive

7653	64
675	83

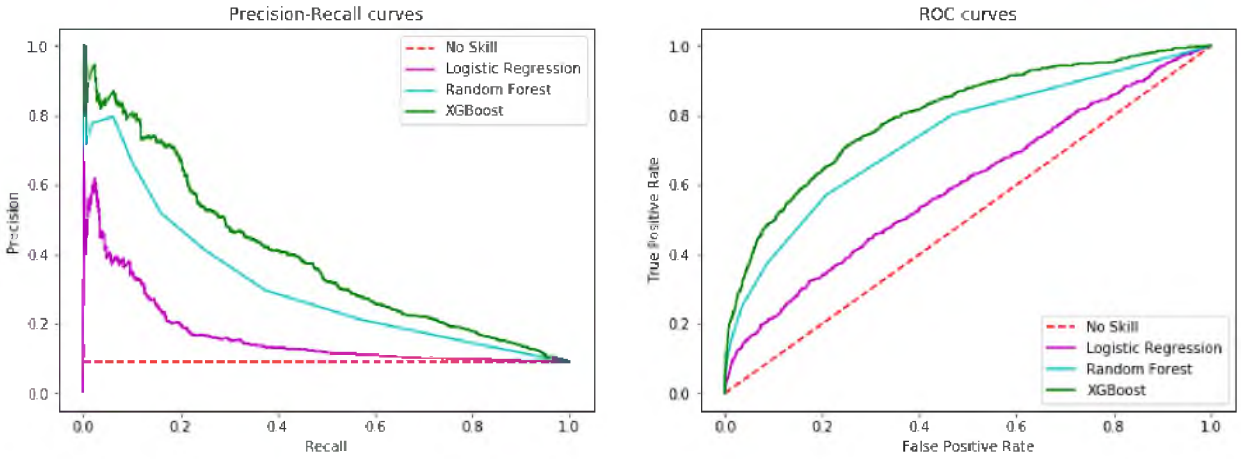
4.1.2. táblázat: A random forest algoritmushoz tartozó igazságtáblázatunk az üres értékek oszlopátlaggal való feltöltése után

Tehát a 758 csaló felhasználónk közül 83-at sikerült csak beazonosítania, ellenben a másik osztály terén a modell meglehetősen pontos, 7717 esetből 7653-at jól jósolt meg.

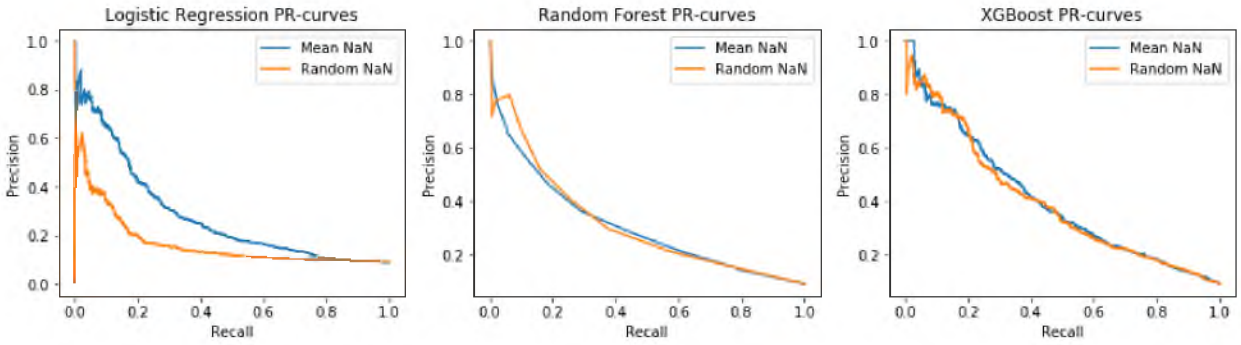
Az átlaggal való feltöltés nagyon sok egyforma adatot generál, ami az osztályozás terén problémát jelenthet, ezért a NaN értékeket egy másik módszerrel is feltöltöttem, hogy összehasonlíthassam az eredményeket. Minden dátumhoz tartozó oszlopnak van egy maximum illetve egy minimum értéke, amik közötti random számokat generálva az adataink változatosak lesznek, így biztosan nem befolyásolják az ugyanolyan értékek az osztályozás folyamatát.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,53	0,65	0,54	7618	99
				696	62
Random Forest	0,54	0,76	0,56	7667	50
				685	73
XGBoost	0,55	0,85	0,57	7694	23
				675	83

4.1.3. táblázat: Elért relevánseredmények az üres értékek minimum és maximum érték közötti random számmal való feltöltése után



4.1.2. ábra: ROC és Precision-Recall görbe az üres értékek minimum és maximum érték közötti random számmal való feltöltése után



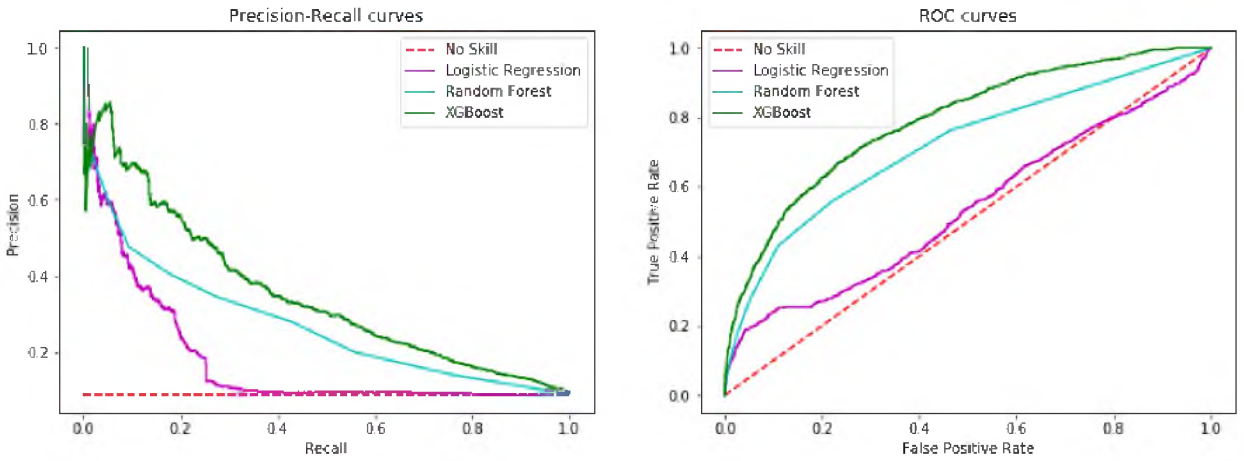
4.1.3. ábra: Az egyes algoritmusok oszlopátlaggal (kék) és random számokkal (narancssárga) elért Precision-Recall görbéinek összehasonlítása

Az adatok random feltöltése főleg a logisztikus regresszió terén produkált gyengébb eredményeket, ami leginkább a precision értékekben mutatkozik meg. Ennek ellenére, mivel a Random Forest illetve XGBoost ígéretesebbnek tűnik a Logistic Regression-höz képest, és előbbi két algoritmus során a véletlenszerű értékek néhol jobb, de általában is közel ugyanolyan eredményeket hoztak, tovább érdemes vizsgálni ezt az adathalmazt is.

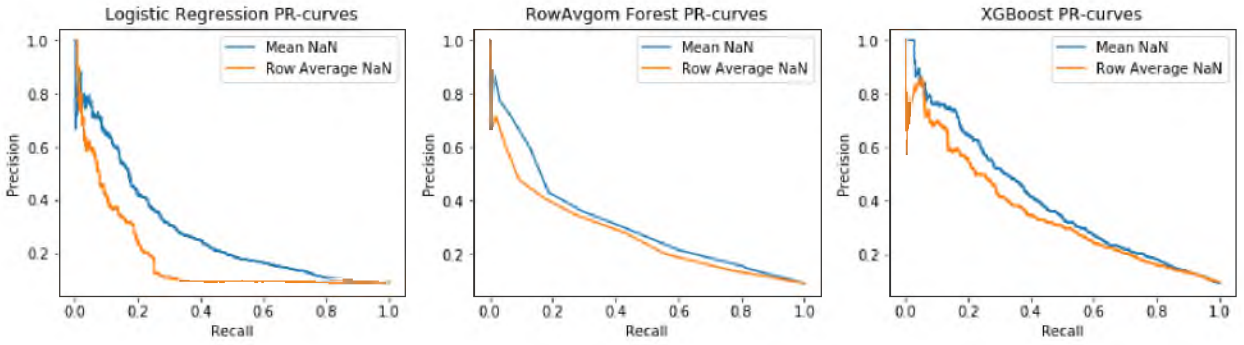
Egy harmadik megoldás lehet nem az oszlopaink átlagával feltölteni az üres értékeinket, hanem a sorátlagokat véve. Ezzel a módszerrel a következő eredményeket értem el.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,57	0,60	0,58	7340	377
				613	145
Random Forest	0,54	0,70	0,55	7639	78
				687	71
XGBoost	0,54	0,80	0,55	7689	28
				695	63

4.1.4. táblázat: Elért relevánseredmények az üres értékek sorátlagokkal való feltöltése után



4.1.4. ábra: ROC és Precision-Recall görbe az üres értékek sorátlagokkal való feltöltése után



4.1.5. ábra: Az egyes algoritmusok oszlopátlaggal (kék) és sorátlaggal (narancssárga) elért Precision-Recall görbéinek összehasonlítása

A sorátlaggal feltöltött üres értékek esetében láthatóan jobbanteljesített mind a random forest és az XGBoost algoritmusunk is, úgyhogy az adathalmaz további módosításai során is érdemes figyelembe venni ezt a lehetőséget.

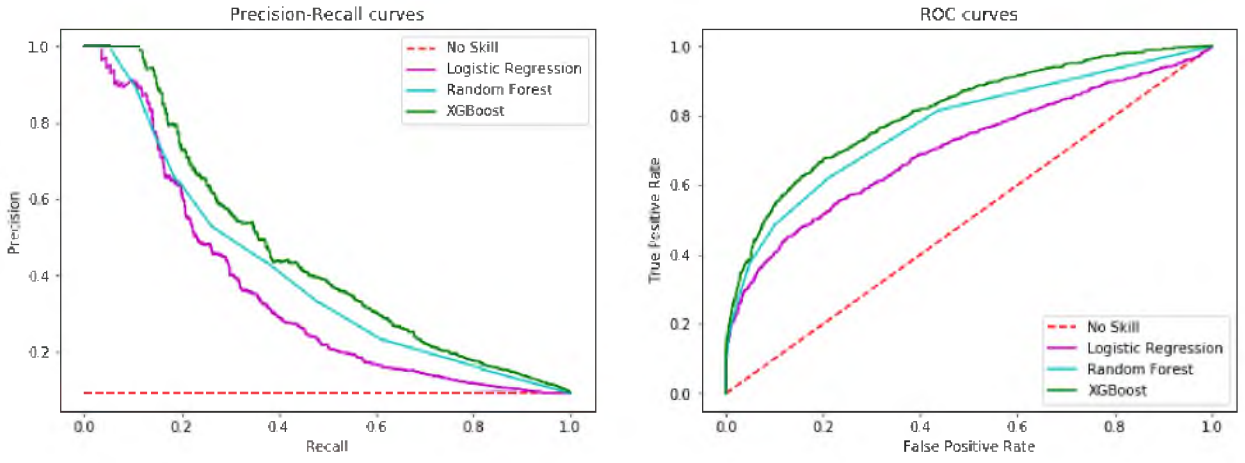
4.2. Adatok eldobása

4.2.1. Kiugró értékek

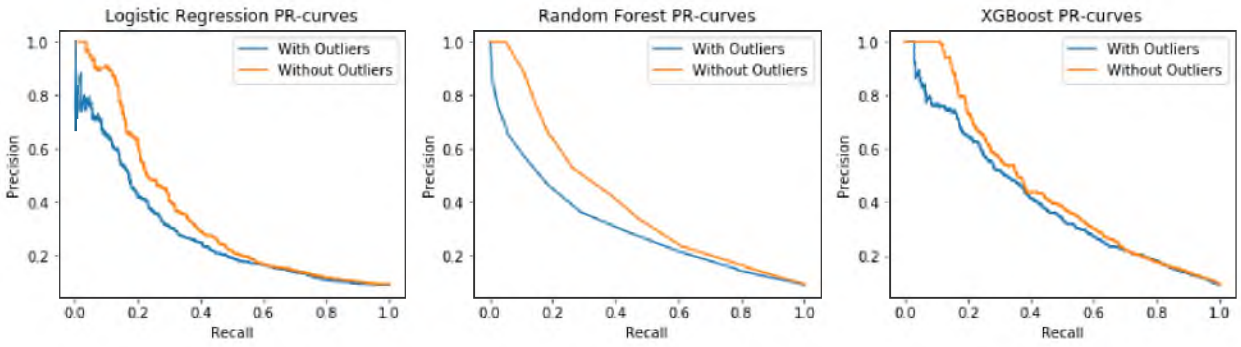
Ahhoz, hogy adathalmazunkat tovább fejlesszük néhány adat eldobása nagy segítségünkre lehet. Az outlierek, azaz kiugró értékek az osztályozás során nagyon könnyen elronthatják az algoritmusunkat, ezért mindenképp érdemes ezeket az adatsorokat eltávolítani. Ezzel a változtatással például a 2014. január 1-jei adatok átlagértéke 7,169-ről 5,327-re csökkentek.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,59	0,78	0,62	7107	75
				586	136
Random Forest	0,60	0,82	0,64	7124	58
				573	149
XGBoost	0,58	0,87	0,62	7156	26
				600	122

4.2.1. táblázat: Elért releváns eredmények az üres értékek oszlopátlaggal való feltöltése és az outlierek eldobása után



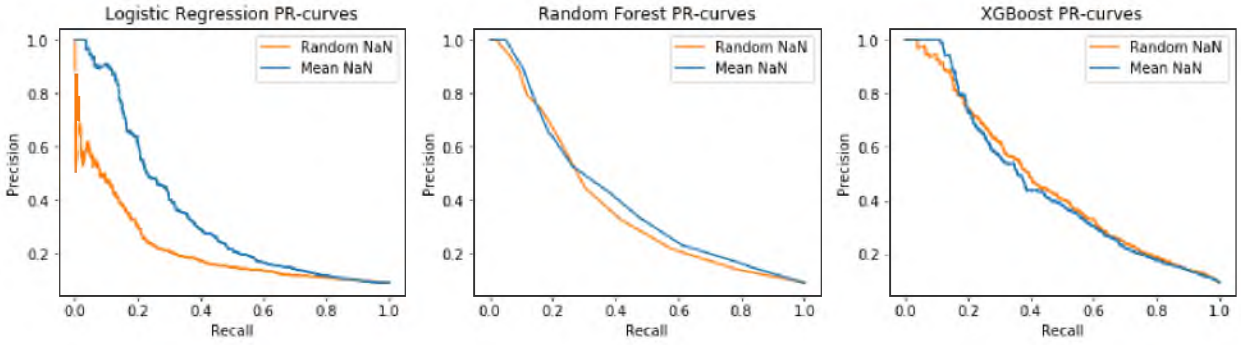
4.2.1. ábra: ROC és Precision-Recall görbe az üres értékek oszlopátlaggal való feltöltése és az outlierek eltávolítása után



4.2.2. ábra: Az egyes algoritmusok oszlopátlaggal elért Precision-Recall görbéinek összehasonlítása outlierekkel (kék) és outlierek nélkül (narancssárga)

Mint az fent az ábrán is látható, az új adathalmazunk kiugró értékek nélkül meglehetősen jobb eredményeket hozott mind az átlag értékekkel feltöltött, illetve a véletlenszerű adatokkal feltöltött adathalmazon is így a továbbiakban, mint az várható volt, a kiugró értékek nélküli adatokkal érdemes dolgozni.

Az alábbi diagramon látható, hogy az outlierek eltávolítása során lényegi változás nem keletkezett a két adathalmaz eredményei között. A logisztikus regresszió továbbra is sokkal gyengébben dolgozik véletlen számokkal, mint átlaggal, de emellett az XGBoost algoritmusnál a véletlenszerű halmaz görbéje az átlag halmaz görbéje fölött marad. Egyes esetekben érdemes lehet ezzel az adathalmazzal is további ellenőrzéseket folytatni.

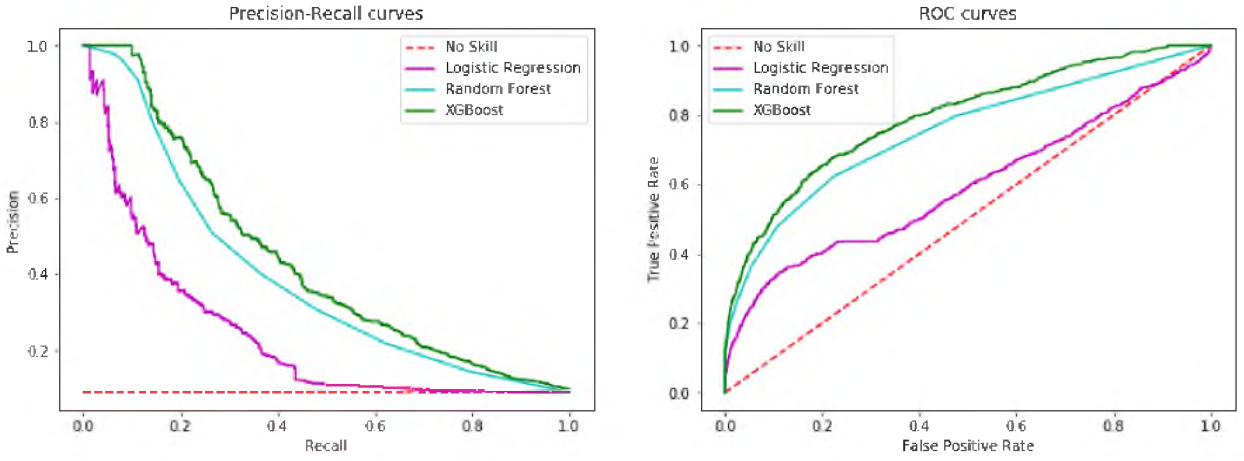


4.2.3. ábra: Az egyes algoritmusok oszlopátlaggal (kék) és random számokkal (narancssárga) elért Precision-Recall görbéinek összehasonlítása az outlierek eltávolítása után

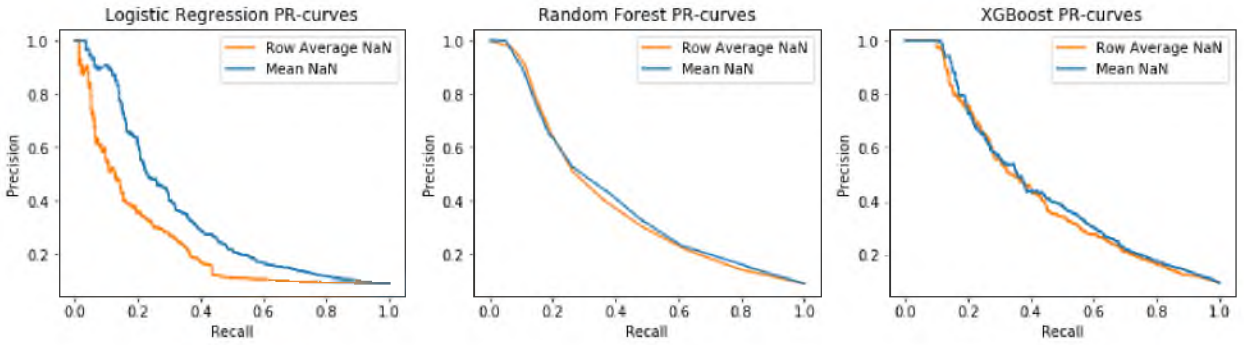
Miután a sorátlaggal feltöltött adathalmazból eltávolítottuk a kiugró értékeket, már közel sem akkora a különbség a random forest és XGBoost algoritmusok esetében sem, mint akkor, amikor az outlierek még az adathalmazban voltak.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,58	0,64	0,60	6930	252
				581	141
Random Forest	0,59	0,79	0,63	7105	77
				580	142
XGBoost	0,58	0,86	0,61	7152	30
				605	117

4.2.2. táblázat: Elért releváns eredmények az üres értékek sorátlaggal való feltöltése és az outlierek eldobása után



4.2.4. ábra: ROC és Precision-Recall görbe az üres értékek sorátlaggal való feltöltése és az outlierek eltávolítása után



4.2.5. ábra: Az egyes algoritmusok oszlopátlaggal (kék) és sorátlaggal (narancssárga) elért Precision-Recall görbéinek összehasonlítása az outlierek eltávolítása után

A logisztikus regresszió terén az oszlopok átlagaival feltöltött adathalmaz sokkal jobban teljesít, mint a sorok átlagával feltöltött halmaz, valamint az előzőekkel szemben már a random forest és XGBoost terén is utolérte a másik datasetet annak ellenére, hogy kiugró értékek eltávolítása előtt még jelentősen le volt maradva vele szemben.

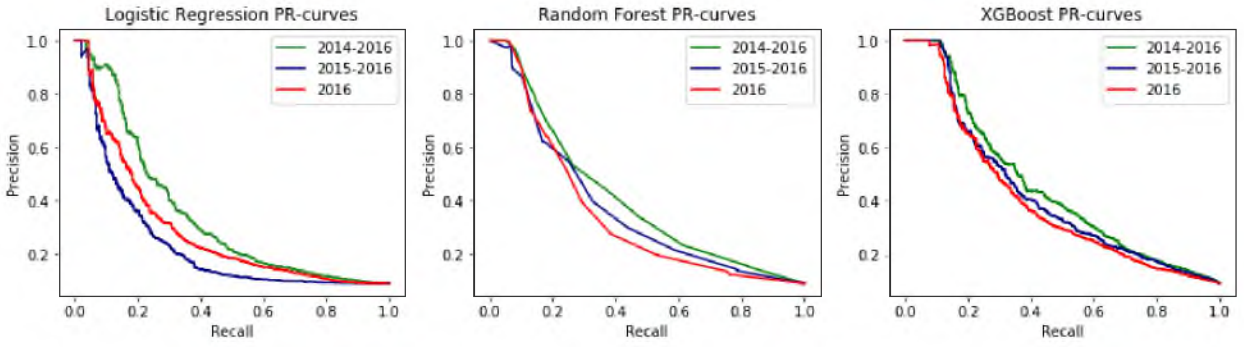
Mivel az outlierek eltávolításával egyértelműen jobb eredmények érhetőek el, így a továbbiakban minden eredmény a kiugró értékek eltávolítása után keletkezett.

4.2.2. Adatbázis megvágása

Egy másik lehetséges módszer az üres (NaN) értékek eltávolítására nem a feltöltésük, hanem az oszlopok, tehát dátumok eldobása. Az adathalmaz 2014. január 1-től 2016. október 31-ig tartalmazza a fogyasztási adatokat, viszont ezen időintervallumban a kezdeti dátumhoz képest egyre több és több fogyasztóhoz lett regisztrált adat, ami azt eredményezi, hogy az adathalmaz eleje az adatfeltöltés után, ha az átlaggal pótolt halmazt nézzük nagyon sokszor ugyanazt az értéket tartalmazza.

Ennek megfelelően lehetséges változtatás az, hogy az adathalmazunkból kisebb dataseteket hozunk létre a dátum alapján vágva azokat. Ezzel ugyan értékes adatokat veszthetünk, de a maradék adat nagyrészéről (még itt is előfordulhat, hogy néhány fogyasztási érték nincs regisztrálva, így ugyanúgy fel kell töltenünk NaN adatainkat) biztosan tudjuk, hogy egy valós, mért adat.

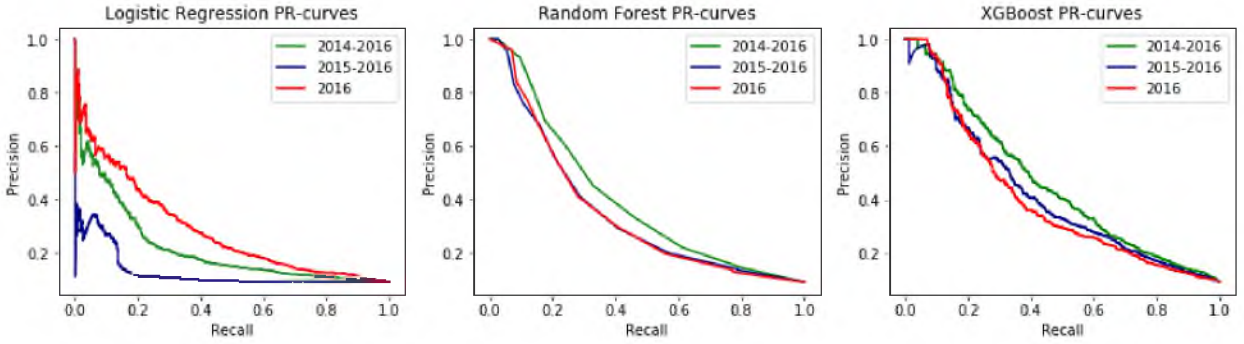
Az eredeti adathalmazból ezért a könnyebb követhetőség végett két új adathalmazt hoztam létre. Az egyik 2015. január 1-től 2016. október 31-ig tartalmazza az adatokat, míg a másik 2016. január 1-től az időintervallum végéig, 2016. október 31-ig. Ezzel ugyan potenciálisan értékes adatokat dobtam el az adathalmazokból, de a különböző esetekként vizsgáltam őket, mennyire befolyásolják eredményeimet az eszközölt változások. Mivel ennek ellenére ezekben is található volt NaN adat, az azzal kapcsolatos változásokat ezekre az adathalmazokra is alkalmazni kellett.



4.2.6. ábra: Az egyes algoritmusok oszlopátlaggal elért Precision-Recall görbéinek összehasonlítása az adatbázis dátum szerinti vágása alapján

Mint az az ábrákon is látszódik, az adathalmaz dátumok szerint kisebb szeletekre bontása az átlagokkal feltöltött adathalmazon nem változtatott érdemben az eredményeken, sőt, ha nem is

kiemelkedően, de továbbra is a teljes adathalmaz variációja eredményezi a legnagyobb görbe alatti területet.



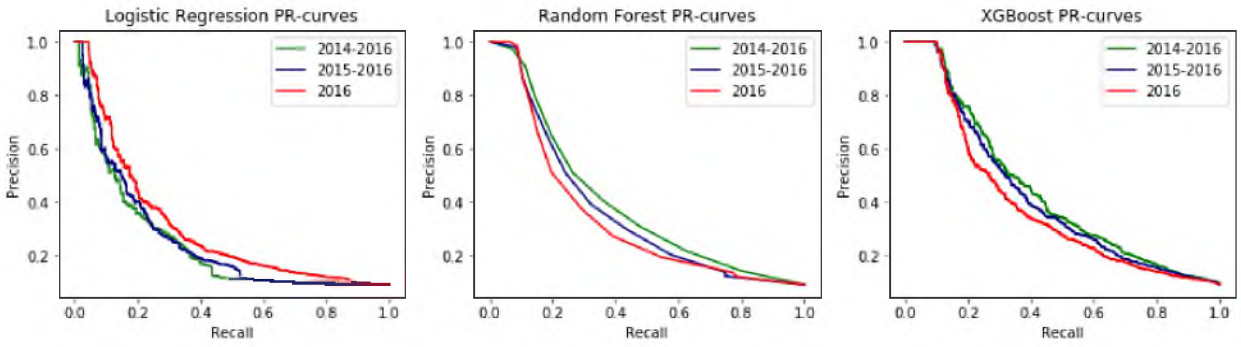
4.2.7. ábra: Az egyes algoritmusok random számokkal elért Precision-Recall görbéinek összehasonlítása az adatbázis dátum szerinti vágása alapján

Ellenben a randomizált NaN adatokon míg a Random Forest és XGBoost algoritmusok szinte megegyező eredményeket mutatnak, mint az átlaggal feltöltött NaN értékeken, a Logistic Regression a legrövidebb időintervallumon, ami csak a 2016-os adatokat tartalmazza a teljes intervallumra alkalmazott algoritmust nagyban felülmúlja. Ennek ellenére a mérési adatok továbbra sem előzik meg az átlagértékeket tartalmazó teljes adathalmazon elért eredményeket.

	Recall	Precision	Fscore	Igazságmátrix	
2014-2016 Mean NaN	0,55	0,78	0,57	7671	46
				676	82
2016 Random NaN	0,54	0,75	0,56	7134	48
				656	66

4.2.3. táblázat: A random számokkal elért legjobb logistic regression algoritmushoz tartozó értékek és az oszlopátlagokkal elért legjobb logistic regression algoritmushoz tartozó értékek összehasonlítása

A harmadik módszer, az üres értékek sorátlaggal való feltöltése hasonló eredményeket szül. A logisztikus regresszió esetében itt is a legkisebb, tehát a csak 2016-os adatokat tartalmazó adathalmaz hozza a legjobb eredményeket, bár itt már nem olyan kiemelkedően, mint a random adatok során tette, de a random forest és XGBoost terén továbbra is a teljes adathalmaz működik a legjobban.



4.2.8. ábra: Az egyes algoritmusok sorátlagokkal elért Precision-Recall görbéinek összehasonlítása az adatbázis dátum szerinti vágása alapján

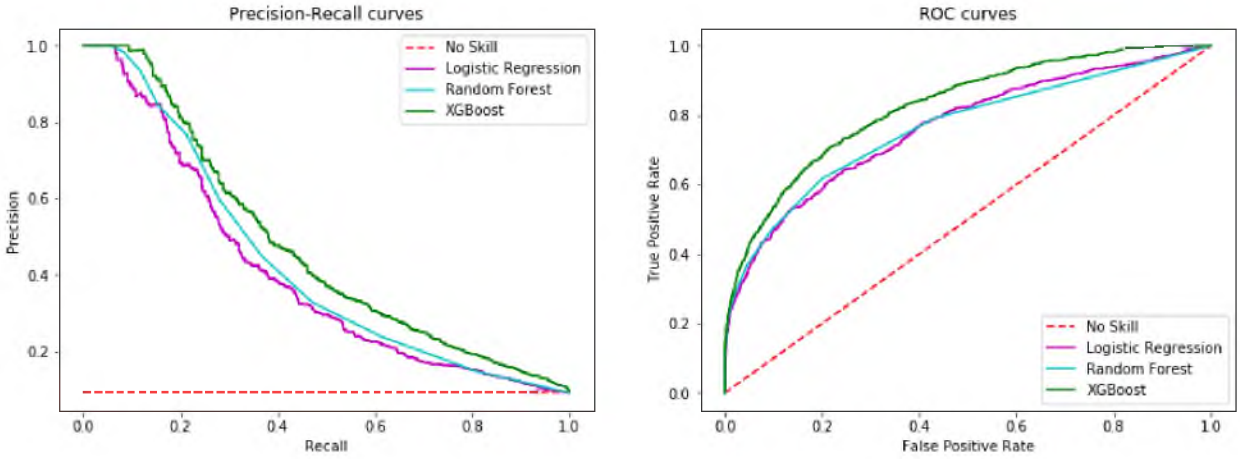
A sorátlagoknál és oszlopátlagoknál is látható, hogy a random forest és XGBoost algoritmus során is a teljes adathalmazok teljesítenek legjobban, ebből adódóan a továbbiakban is ezzel az adathalmazzal érdemes foglalkozni.

4.3. Tulajdonságok felvétele

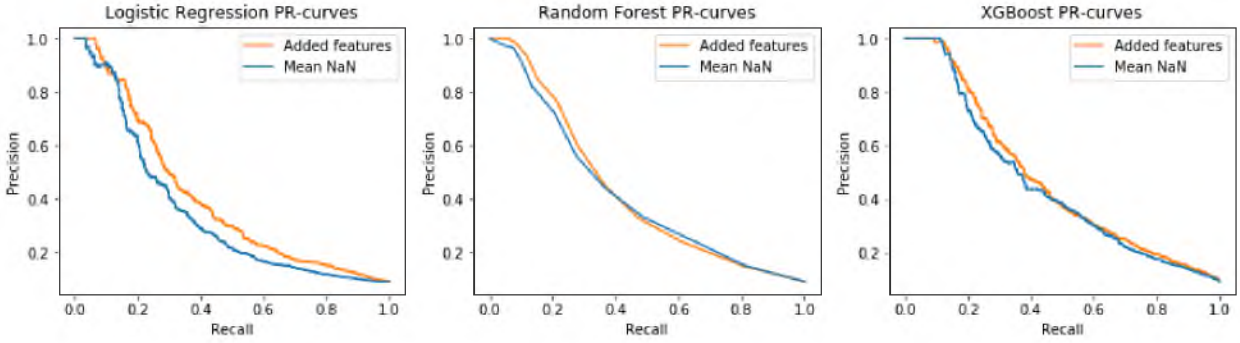
Mivel jelenlegi adathalmazunk csak mérési adatokat tartalmaz mint tulajdonság, azaz feature, érdemes ezekből néhány úgy oszlopot számítani a jobb teljesítmény érdekében. Ehhez a legkézenfekvőbb lehetőségek a sorok átlagát és szórását venni. Ennek segítségével szintén sikerült javítani a modellen kis mértékben.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,61	0,80	0,65	7104	78
				558	164
Random Forest	0,60	0,85	0,65	7135	47
				568	154
XGBoost	0,59	0,89	0,63	7158	24
				590	132

4.3.1. táblázat: Elért releváns eredmények az üres értékek oszlopátlagával való feltöltése és a sorátlag és szórás hozzáadása után

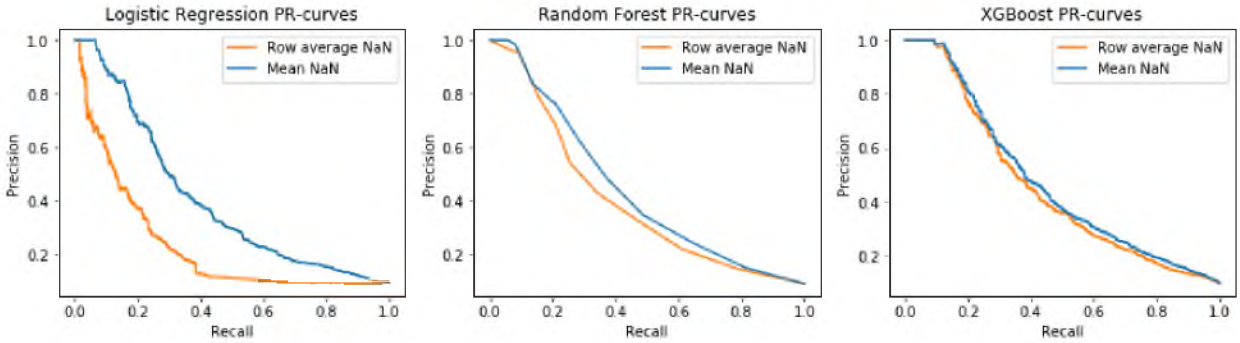


4.3.1. ábra: ROC és Precision-Recall görbe az üres értékek oszlopátlagával való feltöltése és a sorátlag és szórás hozzáadása után



4.3.2. ábra: Az egyes algoritmusok oszlopátlaggal feltöltés után hozzáadott sorátlag és szórással (narancssárga) és nélküle (kék)

A hozzáadott két új feature során az oszlopátlagokkal feltöltött adathalmaz, ha minimálisan is, de jobban teljesített, mint a sorátlagok használatával létrehozott adathalmaz.



4.3.3. ábra: Az egyes algoritmusok oszlopátlaggal (kék) és soráátlaggal (narancssárga) feltöltés után hozzáadott sorátlag és szórással

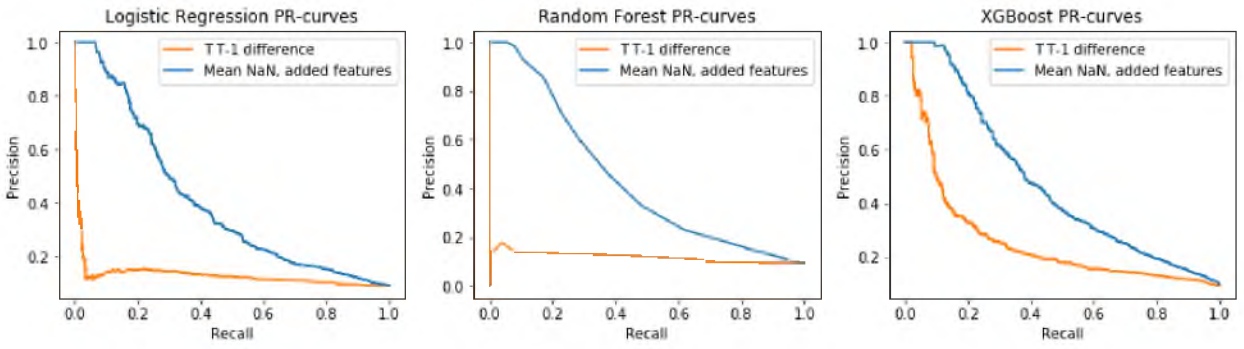
A hozzáadott két új oszlop segítségével jobb eredményeket sikerült elérni, ezért innentől kezdve minden eredmény, ami a fogyasztási adatokat tartalmazó adathalmazokból jött létre már ezt a két oszlopot is tartalmazza.

4.4. Új adathalmaz létrehozása

A számadatainkból létrehozhatunk egy új adathalmazt, ami nem a napi fogyasztást, hanem az aznapi és előtte lévő napi relatív különbséget tárolja. Az új adathalmazunkban ezzel különböző minták jöhetnek létre, amik eltérő eredményeket adhatnak osztályozás terén.

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,51	0,54	0,50	7062	120
				698	24
Random Forest	0,50	0,52	0,48	7150	32
				717	5
XGBoost	0,95	0,51	0,49	7182	0
				714	8

4.4.1. táblázat: Elért releváns eredmények az előző dátumtól való relatív eltérésekből álló adathalmazzal



4.4.1. ábra: Az egyes algoritmusok oszlopátlaggal feltöltés után hozzáadott sorátlag és szórással (kék) és az előző dátumtól való relatív eltérésekből álló adathalamazon

A számadatokból is, de főleg az ábrákból és az igazságmátrixból látszódik, hogy ez az adathalmazunk nagyon gyengén teljesít. A megjósolt csalók mennyisége minimális, így továbbra is a azzal az adathalmazzal érdemes foglalkozni, ami az oszlopátlagokkal feltöltött üres értékeket tartalmazza, valamint már a sorátlag és a szórás is hozzá van adva tulajdonságként.

4.5. Adatbővítés

Mivel az adathalmazunk kiegyensúlyozatlan (imbalanced), tehát a két lehetséges osztály mintái között jelentős számbeli különbség van, segítség lehet a kisebbségben lévő osztály,

esetünkben a csalók osztályát kiegészíteni több, a már meglévő adatsorok másolataival, de ez nem ad új adatot az adathalmazunkhoz. Hasonló feladatokra alkalmazhatóak ennek továbbfejlesztett változatai, az oversampling algoritmusok.[21]

Az oversampling algoritmusok úgy bővítik adathalmazunkat, hogy közben egymástól eltérő módszert használva hoznak létre új adatsorokat (felhasználókat), amivel kibővítik jellemzően a kisebbségben lévő osztály elemeit úgy, hogy egyenlő, vagy közel egyenlő legyen számuk, mint a többségben lévő osztályé.[21]

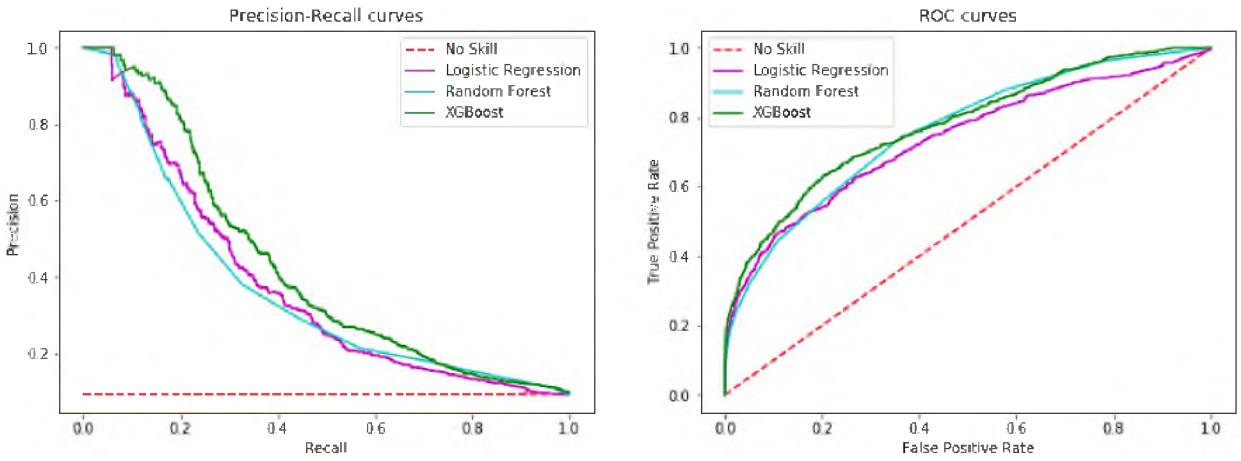
4.5.1. Synthetic Minority Oversampling Technique

A Synthetic Minority Oversampling Technique, továbbiakban SMOTE nem csak a meglévő adatsorok másolatait adja hozzá tanuló adathalmazunkhoz, hanem azokon kisebb módosításokat hajt végre.

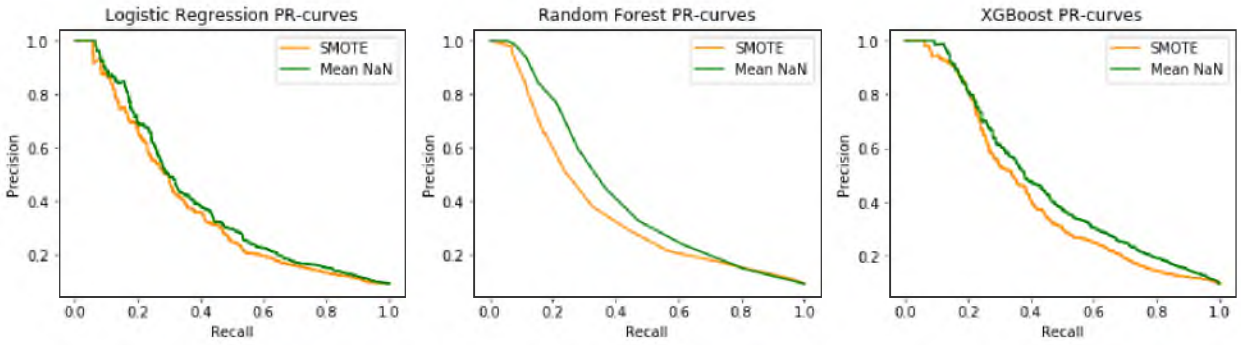
Az algoritmus működése során választ egy random mintát a meglévő pozitív eseteinkből, majd annak k legközelebbi szomszédjából kiválaszt egyet. A két kiválasztott minta a térben elhelyezhető és összeköthető egy egyenessel. Az új mintánk ezek összekötését szolgáló szakaszon helyezkedik el egy véletlenszerű helyen.[22]

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,67	0,58	0,58	5709	1473
				330	392
Random Forest	0,64	0,67	0,65	6813	369
				479	243
XGBoost	0,71	0,60	0,62	5920	1262
				292	430

4.5.1. táblázat: Elért releváns eredmények SMOTE adatbővítés után



4.5.1. ábra: ROC és Precision-Recall görbe SMOTE adatbővítés után



4.5.2. ábra: Az egyes algoritmusok SMOTE alkalmazása előtt (zöld) és után (narancssárga)

Ahogy látható, a három algoritmus görbéi a SMOTE adatbővítés után már sokkal közelebb találhatóak egymáshoz, mint az ezelőtti eredményeknél látszódott. Ellenben az adathalmazt, aminél az üres értékek helyére az oszlopok átlagait helyettesítettem be csak a logisztikus regresszió tudta megközelíteni SMOTE alkalmazása után, míg a random forest és az XGBoost alulmaradtak vele szemben.

Érdemes viszont jobban megvizsgálni a táblázatokat, azon belül is leginkább az igazságtáblázatainkat külön-külön. Mivel az XGBoost eredményei között volt a legnagyobb az eltérés vizsgáljuk meg a két különböző adathalmazon látható igazságmátrixokat, hogy a százalékos adatokhoz pontos számadatokat tudjunk rendelni.

True Negative	False Positive
False Negative	True Positive

Mean NaN	
7158	24
590	132

SMOTE	
5920	1262
292	430

4.5.2. táblázat: XGBoost algoritmus által elért igazságmátrixok SMOTE alkalmazása előtt és után

Ha átültetnénk ezeket az adatokat a való életbe, és a csalók ellenőrzéseként figyeljük a számokat az derül ki, hogy míg az átlagokkal feltöltött adathalmazunk során az algoritmus összesen 156 fogyasztó ellenőrzését szorgalmazta, amiből ugyan 132 ténylegesen csaló is volt, de emellett 590 esetben nem ismerte fel a pozitív osztályt, tehát a csalók 82%-át „futni hagyta", így fraud detection terén gyengén teljesítve.

Emellett láthatjuk, hogy a SMOTE adatbővítés után ugyanez az algoritmus, az XGBoost, az összes 7904 felhasználóból már 1692 fogyasztónál jelezte, hogy lehetséges csalóval van dolgunk, amiből 430 valójában is szabálytalan mintavételezést hajtott végre, és az algoritmus 292, azaz már csak a csalók 40%-át sorolta a negatív osztályba.

Ennek az eredménynek az eléréséhez ugyan tovább 1262 fogyasztót ellenőrizni kellett hiába, hiszen a valójában a negatív osztály tagjai voltak, de így is a megfigyelt felhasználók közül minden negyedik sikeresen kiszűrt fogyasztó. Ehhez ugyan a felhasználóink 21%-át meg kellett vizsgálnunk közelebbről.

Egy másik, más megközelítésű, de ígéretesnek tűnő algoritmus a SMOTE elvégzése után a random forest.

True Negative	False Positive
False Negative	True Positive

6813	369
479	243

4.5.3. táblázat: Random forest igazságmátrixa SMOTE alkalmazása után

Ez ugyan csak a csalók 34%-át találta meg és sorolta a pozitív osztályunkba, viszont közel sem került annyi negatív eset az ellenőrzendők közé, mint az XGBoost esetében. A rendszerünkben így továbbra bent marad a szabálytalanul vételező fogyasztók 66%-a, de az ellenőrzések hatásfoka nő, ugyanis minden tíz ellenőrzésből négy esetében ténylegesen csalót sikerül kiszűrnünk.

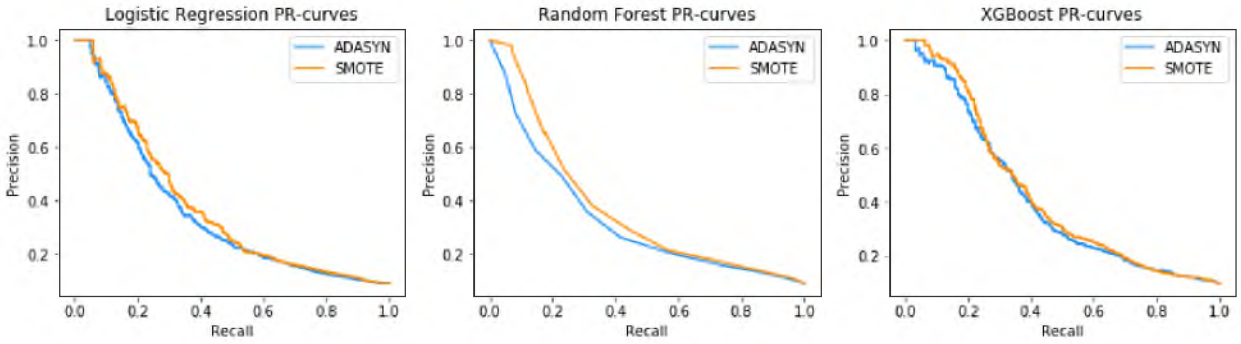
4.5.2. Adaptive Synthetic Sampling

Az Adaptive Synthetic Sampling, továbbiakban ADASYN a SMOTE-tal ellentétben nem teljesen random választja ki, hogy melyik mintánkat fogja másolni és módosítani. Az algoritmus leginkább azokra a mintákra koncentrál, amit az osztályozó algoritmusunknak nehéz megtanulnia.[21]

A másolni kívánt minta kiválasztása során azokat részesíti előnyben, amik hasonlítanak a negatív osztályunkban lévő mintáinkra is, tehát olyan mintákat választ, amik körül a térben a többi kisebbségben lévő minta elszórva található meg, viszont a többségben lévő mintából sok van.[21]

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,67	0,58	0,58	5548	1634
				307	415
Random Forest	0,63	0,65	0,64	6792	390
				501	221
XGBoost	0,70	0,59	0,59	5634	1548
				275	447

4.5.4. táblázat: Elért releváns eredmények ADASYN adatbővítés után



4.5.3. ábra: Az egyes algoritmusok ADASYN (kék) és SMOTE (narancssárga) alkalmazása után

Ahogy az a táblázatok és az ábrákat összehasonlítva is látszik, a SMOTE és az ADASYN eljárás logisztikus regresszió és XGBoost esetében szinte azonos, és random forest terén is a SMOTE kicsit erősebb eredményeket produkál.

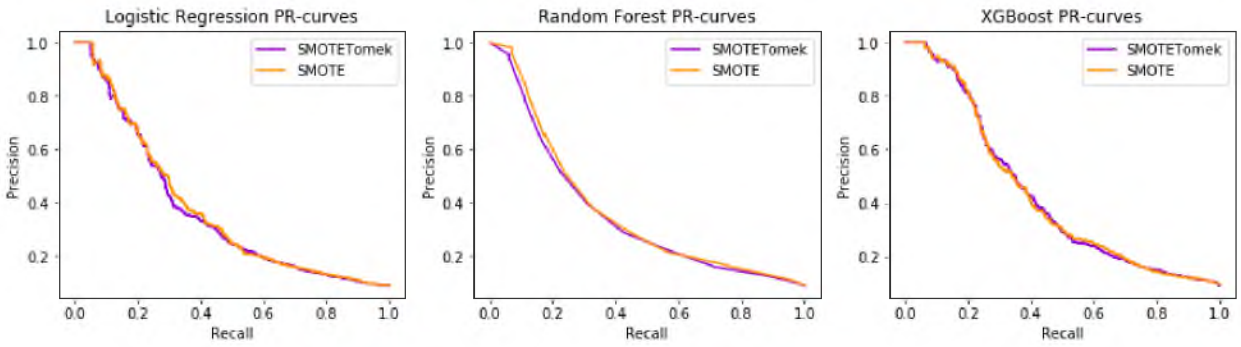
4.5.3. SMOTE + Tomek Links

Az oversampling mellett létezik úgynevezett undersampling is. Ezek a technikák nem új adatokat generálnak az adathalmazunkba, hanem bizonyos kritériumok alapján elvesznek belőlük, így kiegyensúlyozva az adathalmazunkat.

A SMOTE + Tomek Links egy hibrid technológia, mivel a SMOTE oversampling mellett használja a Tomek Links nevű undersampling technikát is. Miután az adatbővítés elvégezte a két osztály egyes elemei nagyon hasonlíthatnak egymásra, a csoportok összefolyhatnak, így túltanulást eredményezve. A Tomek Links emiatt az osztályok közötti határokat teszi jobban elkülöníthetővé azzal, hogy mindkét osztály mintáiból elvesz.[22]

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,68	0,58	0,58	5643	1539
				307	415
Random Forest	0,63	0,66	0,65	6832	350
				495	227
XGBoost	0,70	0,60	0,61	5851	1331
				296	426

4.5.5. táblázat: Elért releváns eredmények SMOTETomek adatbővítés után



4.5.4. ábra: Az egyes algoritmusok SMOTETomek (lila) és SMOTE (narancssárga) alkalmazása után

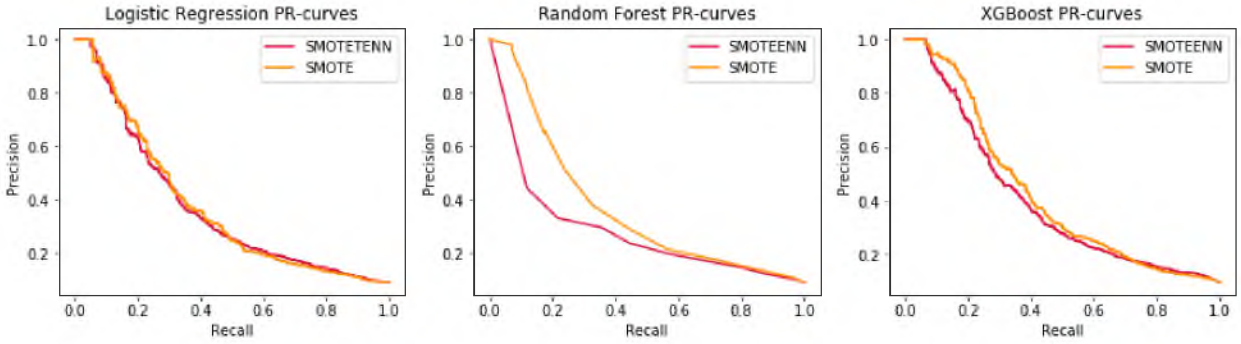
Mint az ábrákból látható, a SMOTE + Tomek links illetve a SMOTE elvégzése után az eredmények szinte azonosak, lényegi változás nincs a kettő között. Az oversampling mellett a Tomek links undersampling technika viszont nagyban megnöveli az adatbővítés idejét, ami a SMOTE 14 másodperces idejéhez képest 2973 másodperc alatt valósult meg változást nem hozva a mérőszámokban.

4.5.4. SMOTE + ENN

A SMOTE + ENN szintén egy undersampling technika, ami a k legközelebbi szomszéd alapján működik. Minden többségben lévő osztályba tartozó mintának veszi a szomszédjait, amik alapján eldönti, hogy a kiválasztott minta jó osztályba tartozik-e. Ha a mintának a kisebbségben lévő osztályban kéne kerülnie ezek alapján törli az adathalmazból.[22]

	Recall	Precision	Fscore	Igazságmátrix	
Logistic Regression	0,67	0,56	0,50	4277	2905
				181	541
Random Forest	0,67	0,57	0,57	5545	1637
				314	408
XGBoost	0,67	0,56	0,46	3630	3552
				115	607

4.5.6. táblázat: Elért releváns eredmények SMOTEENN adatbővítés után



4.5.5. ábra: Az egyes algoritmusok SMOTEENN (piros) és SMOTE (narancssárga) alkalmazása után

Az ábrákból ugyan az látszik, hogy a SMOTE + ENN szintén az undersampling nélküli technikával egyezik meg, ráadásul a bővítés ideje is rendkívül hosszú, ennek ellenére az XGBoost pozitív eseteinek érzékenysége újra egy az eddigiektől eltérő eredményt hozott, amit szintén érdemes jobban megvizsgálni.

Az algoritmus igazságmátrixa a következő:

True Negative	False Positive
False Negative	True Positive

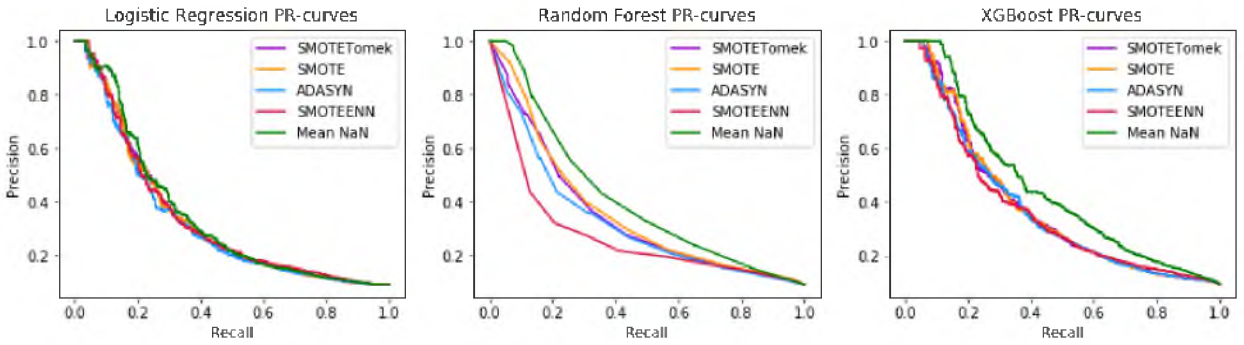
3630	3552
115	607

4.5.7. táblázat: XGBoost algoritmus igazságtáblázata SMOTEENN alkalmazása után

Ahogy az látszik, és a recallból is tudhatjuk, a modell a 722 pozitív esetünk 84%-át, azaz 607 esetet szűrt ki. Ez az eredmény az eddigi legjobb ezen a téren, viszont ennek ára a megbízhatóság visszaesése volt. Mindez azt jelenti, hogy ennek a kiszűréséhez a tesztadatok 53%-át osztályozta pozitív esetként a modell, így 7904 felhasználónkból 4159-et ellenőriztet a szolgáltatóval, viszont az ellenőrzöttek közül minden hetedik valóban szabálytalan vételezést hajtott végre.

4.5.5. Összefoglaló

Összességében az oversampling technikák számomra kedvező eredményeket hoztak. Annak ellenére, hogy a precision-recall görbéken továbbra is az az adathalmaz mutat a legjobban, amiben kiugró értékek már nem szerepelnek és random számokkal vannak feltöltve az üres értékek, -főleg a random forest és XGBoost esetében látható mindez, - az érzékenység értékek megnövelése volt a cél, amit sikerült elérni.



4.5.6. ábra: Az egyes algoritmusok négy adatbővítési módszerrel és adatbővítési módszer nélkül

Emellett nem szabad elfelejtkezni a tényről, hogy az adatbővítési technikáknak hosszabb ideig futnak, hiszen az új adatok generálása nyilvánvalóan időigényes feladat.

	Adatbővítési idő (sec)
SMOTE	14,20
ADASYN	168,35
SMOTE + Tomek	3121,75
SMOTE + ENN	3365,31

4.5.8. táblázat: Az adatbővítési ideje a négy data augmentation módszernek

Az oversampling eredményei után a továbbiakban a legjobb értékeket hozó kombinációkra fogom optimalizálni az algoritmusainkat. Ezek a következők igazságmátrixukkal:

- Oversampling: SMOTE

  Algoritmus: Random Forest

6813	369
479	243

- Oversampling: SMOTE

  Algoritmus: XGBoost

5920	1262
292	430

- Oversampling: SMOTEENN

  Algoritmus: XGBoost

3630	3552
115	607

Ezek az adatbővítő technikák és melléjük választott algoritmusok hoztak a legváltozatosabb eredményeket, amik erősségeinek kiaknázásával tovább lehet szűkíteni a kört és optimális esetben az eredményeken javítani is tudunk. Mindemellett a három alap algoritmust is vizsgálom tovább, ugyanis mint az a precision-recall görbéken látszódik f-scorejuk magasabb, mint az adatbővítés technikák utáni adathalmazokon, így, ha abban a formában nem is tökéletesek a fraud detection feladatára, optimalizálással más eredmények is elérhetőek lehetnek, amik felülmúlhatják az adatbővítés utáni alkalmazásukat.

5. Algoritmusok optimalizációja

A különböző algoritmusoknak különböző paraméterbeállításaik vannak. Ezen paraméterek segítségével tudjuk hangolni a modelljeinket egyes feladatokra, jelen esetben, hogy minél hatékonyabb bináris osztályozót hozzunk létre az adott adathalmazon, és így csaló felderítésre optimalizáljuk modellünket.

Ehhez továbbra is legfőbb mérőszámként az érzékenység és megbízhatóság értékeket vesszük, de mivel a modelleket alapvetően éles használatra szánjuk figyelembe kell venni azok tanítási idejét, illetve memóriahasználatát is.

A feladatra hatással lévő paramétereket és azok beállításait teszteléssel és többféle beállítás összehasonlításával sikerült kiválasztani.

Ugyanazon modellek eredményei minimálisan eltérhetnek az eddigiektől amit a modell újratanítása okozott.

5.1. Logisztikus regresszió

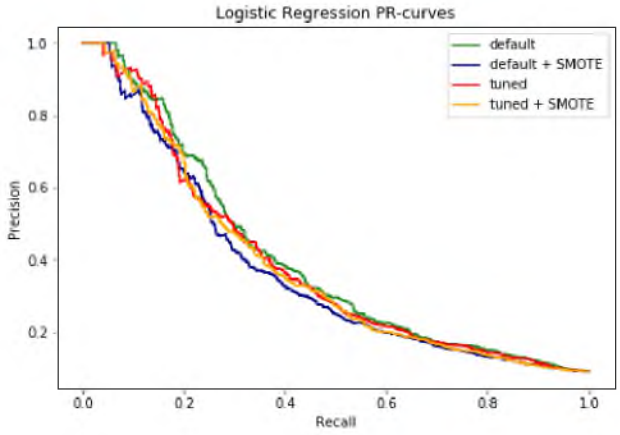
A logisztikus regresszió, mivel kevésbé komplex, mint másik kettő algoritmusunk, így nem rendelkezik rengeteg állítható paraméterrel, és ezeknek nagyrésze nem is befolyásolta érdemben eredményeinket a fraud detection problémájában. Az tesztelés során a befolyásoló tényezők a következők voltak:[23]

- class_weight
- max_iter

A beállítások, amikkel logisztikus regresszió esetében a legjobb eredményeket tudtam elérni így néztek ki:

```
lrc = LogisticRegression(solver='lbfgs', class_weight = 'balanced', max_iter = 50)
```

Mindezek ellenére a logisztikus regresszió továbbra sem bizonyult a legjobb választásnak a feladatra.



5.1.1. ábra: Logistic regression algoritmus alapbeállításokkal (zöld) majd SMOTE után (kék), hangolt beállításokkal (piros) majd SMOTE után (narancssárga)

Ugyan a Precision-Recall görbéken úgy néz ki, hogy nem sok minden változott az oszlopok átlagértékeivel feltöltött üres értékek adathalmazán a kezdeti állapottól, ahol a logisztikus regresszió alapbeállításait használtam (az ábrán default néven, zölddel jelölve), de a megbízhatóság illetve érzékenység nagyban változtak az alapbeállításokhoz képest.

	Recall	Precision	Fscore	Training time (sec)
default	0,61	0,80	0,65	4,54
default + SMOTE	0,68	0,58	0,59	9,76
tuned	0,69	0,59	0,60	3,34
tuned + SMOTE	0,68	0,58	0,59	5,23

5.1.1. táblázat: Elért releváns eredmények a 5.1.1. ábrán látható beállítások által

Mint az látható, a SMOTE és a hangolás szinte ugyanazt az értéket eredményezte, viszont az alap algoritmushoz képest mindhárom több csalót talált a meg, viszont ahhoz képest nagyon sok negatív osztályba tartozó felhasználót is csalónak minősített.

True Negative	False Positive
False Negative	True Positive

default	
7104	78
558	164

tuned + SMOTE	
5672	1510
312	410

5.1.2. táblázat: Az igazságtáblázatok alapbeállítások és hangol beállítások és SMOTE alkalmazása után

Az eddigiekben már láttunk ennél kedvezőbb eredményeket is az XGBoost illetve random forest algoritmusok esetében, ennek megfelelően a logisztikus regresszió módszert tovább vizsgálni nem érdemes.

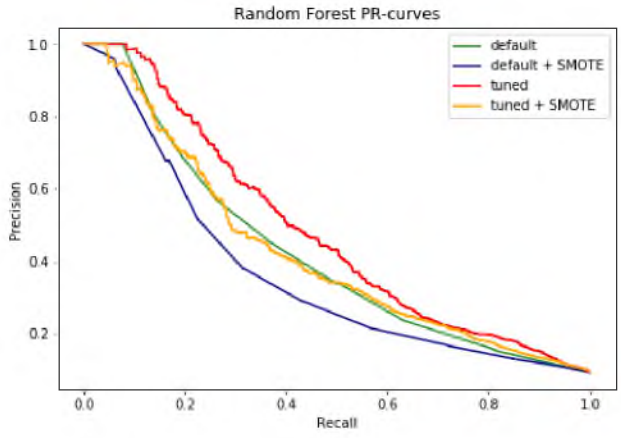
5.2. Random Forest

A random forest osztályozó esetében már több paraméter rendelkezésünkre áll, amin változtatni tudunk a jobb eredmények érdekében, de ezek közül sem mind hozott jelentős különbséget az eredményekben, így csak a következőket használtam:[24]

- n_estimators
- max_depth
- min_samples_split
- min_samples_leaf

A random forest esetében a legjobb eredményeket a következő beállításokkal sikerül elérnem:

```
rfc_tuned = RandomForestClassifier(n_estimators = 100, max_depth = 50,
min_samples_split = 5, min_samples_leaf = 2)
```



5.2.1. ábra: Random forest algoritmus alapbeállításokkal (zöld) majd SMOTE után (kék), hangolt beállításokkal (piros) majd SMOTE után (narancssárga)

Ahogy látható az ábrán is, az alapbeállítás és a hangolt beállítás között, valamint a két SMOTE alkalmazása utáni modell között is észrevehető különbségek vannak.

Ugyan a SMOTE nélküli esetek mindkét alkalommal (alap, illetve hangolt beállításokkal is) Precision-Recall görbén összesítve jobb eredményeket értek el, de mint azt már sokszor láthattuk ez nem feltétlenül a legkedvezőbb eset számunkra. A érzékenység értékek az adatbővítés után nagyban nőttek az eddig vizsgált esetekben, így most is.

	Recall	Precision	Fscore	Training time (sec)
default	0,59	0,80	0,62	26,19
default + SMOTE	0,63	0,66	0,65	38,42
tuned	0,59	0,89	0,63	173,99
tuned + SMOTE	0,65	0,70	0,67	201,46

5.2.1. táblázat: Elért releváns eredmények a 5.2.1. ábrán látható beállítások által

A táblázatból jól látszik, hogy annak ellenére, hogy a csak alapbeállítással, SMOTE nélkül futtatott modell és az optimalizált, adatbővítéssel futtatott modell esetében az utóbbinak érzékenység és F-score értékei főleg a kisebbségben lévő osztályt tekintve jóval magasabbak, és csak a megbízhatóság érték miatt tűnnek közel azonosnak a precision-recall görbén.

A két optimalizált modellünk között szintén az látszódik, hogy a érzékenység és F-score értékekben az adatbővítés után sokkal jobban teljesít a modell.

		Default + SMOTE		Tuned		Tuned + SMOTE	
True Negative	False Positive	6834	348	7158	24	6898	284
False Negative	True Positive	495	227	589	133	464	258

5.2.2. táblázat: Az igazságtáblázatok alapbeállítások és SMOTE, a hangolt beállítások és hangolt beállítások és SMOTE alkalmazása után

Az igazságtáblázatokból az is könnyebben látható, hogy a SMOTE továbbra is sokkal több fogyasztót sorol tévesen a csalók osztályába, ellenben kétszer annyi tényleges fraud esetet is talál meg, mint a sima, hangolt modellünk.

Mint az a két default + SMOTE és a tuned + SMOTE táblázatokból látszódik a modellünk minden tekintetben fejlődött a random forest algoritmus optimalizálásával. Mind a false negative és false positive értékek csökkentek, aminek eredményeképp true negative és true positive értékeink csak nőttek. A hangolás miatt a modell tanítási idejében növekedés látható, de az eredmények javulása kárpótolja ezt a változást.

A hangolt modell SMOTE-tal 258, azaz a csalók 36%-át találja meg, amihez további kicsit több, 284 fogyasztót sorol ebbe az osztályba, de ez az összes felhasználónk 4%-a. Ugyan nem ér el olyan eredményeket fraud detection terén, mint korábban látott XGBoost modellek, de nem is hibázik annyit a negatív osztály terén, így minimalizálva a téves fizikai ellenőrzéseket.

5.3. XGBoost

Az XGBoost esetében két esetet is vizsgálok. Az egyiknél a SMOTE adatbővítési módszer alkalmazása utáni eredményeket figyelem meg, másik esetben pedig a SMOTE oversampling után ENN undersampling használatával elért mérési adatokat összegzem. Az XGBoost optimalizálását főleg a következő paraméterek befolyásolták:[25]

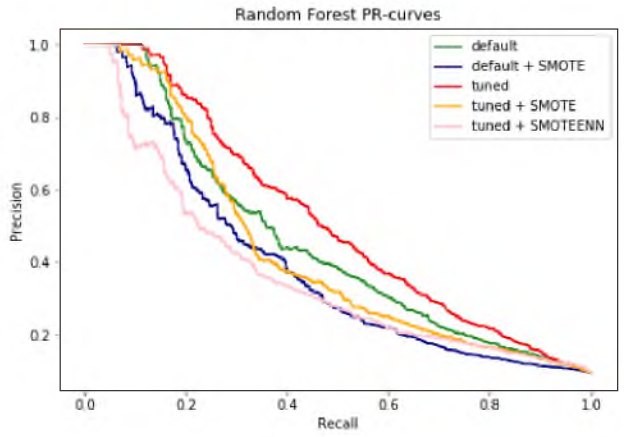
- max_depth
- min_child_weight
- max_delta_step
- scale_pos_weight

A SMOTE adatbővítéshez a legjobb eredményeket a következő beállításokkal étem el:

```
xgb_tuned = XGBClassifier(max_depth = 10, min_child_weight = 15, max_delta_step =
5, scale_pos_weight = 12)
```

A SMOTEENN alkalmazásához pedig a következő beállításokkal:

```
xgb_tuned = XGBClassifier(max_depth = 10, min_child_weight = 15, max_delta_step =
5, scale_pos_weight = 5)
```



5.3.1. ábra: XGBoost algoritmus alapbeállításokkal (zöld) majd SMOTE után (kék), hangolt beállításokkal (piros) majd SMOTE után (narancssárga) és hangolt beállítások és SMOTEENN után (rózsaszín)

Az ábrán is látható, hogy azok a modellek, ahol adatbővítést használtam nem mutatnak olyan jól, de ez a megbízhatóság érték esése miatt van így. Ellenben az látszik, hogy az optimalizált algoritmus magában igen jól teljesít a többihez képest.

	Recall	Precision	Fscore	Training time (sec)
default	0,59	0,89	0,63	155,28
default + SMOTE	0,70	0,60	0,61	293,88
tuned	0,69	0,73	0,71	468,05
tuned + SMOTE	0,72	0,59	0,59	1087,34
tuned + SMOTEENN	0,73	0,58	0,53	821,35

5.3.1. táblázat: Elért releváns eredmények a 5.3.1. ábrán látható beállítások által

Mind a SMOTE és SMOTEENN esetében sikerült a érzékenységet és a megbízhatóságot is növelni, de még mindig nagyon alacsonyak, ami az igazságmátrixból is jobban látható, hogyan is sikerül a csalók szűrése:

True Negative	False Positive
False Negative	True Positive

Default + SMOTE	
5902	1280
301	421

Tuned	
6888	294
414	308

Tuned + SMOTE	
5389	1793
219	503

Tuned + SMOTEENN	
4531	2651
123	599

5.3.2. táblázat: Az igazságtáblázatok alapbeállítások és SMOTE, a hangolt beállítások, hangolt beállítások és SMOTE és hangolt beállítások és SMOTEENN alkalmazása után

A modelleknél, ahol adatbővítést használtam a pozitív osztály jó részét már kiszűrik a modellek, viszont így sem tökéletesek az általuk adott eredmények. Annak ellenére, hogy a két SMOTE a csalók 58%-át, illetve 70%-át találja meg, valamint a SMOTEENN 83%-ukat helyezi a megfelelő osztályba, a negatív osztály rosszul felcímkézése problémát okoz.

6. Konklúzió

6.1. Adathalmaz fejlesztése

Ahhoz, hogy megfelelő eredményeket érjünk el a fraud detection problémája terén előszöris az adatokkal kell foglalkoznunk. Jelenleg csekély mennyiségű adat áll rendelkezésre az interneten a fogyasztási csalók vizsgálatához szükséges paraméterekkel, és azok minősége sem optimális. A rengeteg hiányzó adat, valamint a rengeteg kiugró érték a State Grid Corporation of China (SGCC) adathalmazában rendkívül megnehezíti a dolgot, és mint az az eredményekből is látható, az adathalmaz különböző optimalizálásával az értékek is nagyban javultak már az adatbővítási módszerek előtt is. Egy megfelelő adathalmaz, amiben elegendően sok, egy régióban élő felhasználó több éves fogyasztási adatait sikerülne összesíteni ténylegesen, minimális üres értékekkel valószínűleg nagyban javítaná a modellek által elért eredményeket.

6.2. Modellek választása

A gépi tanulási módszerek a fraud detection feladatára láthatóan alkalmazhatóak, viszont mint az az ábrákon is jól megfigyelhető, nem tökéletesek. Emellett mivel a különböző módszerek eltérő eredményeket képesek produkálni meg kell határozni, mi a prioritás a probléma elvégzése során.

Összegezzük a legjobban teljesítő kombinációkat. Ha jobban megnézzük a legjobb modellek végül az XGBoost algoritmus használatával jöttek létre, hiszen a random forest esetében az optimalizált algoritmus SMOTE adatbővítés alkalmazásával bizonyult a legjobbnak, aminél az XGBoost optimalizált algoritmusa kicsit jobb eredményeket ért el, mint az az igazságmátrixaikon látszódik:

True Negative	False Positive
False Negative	True Positive

Tuned RF + SMOTE	
6902	280
476	246

Tuned XGBoost	
6888	294
414	308

6.2.1. táblázat: Az igazságtáblázatok hangolt random forest algoritmus és SMOTE, valamint hangolt XGBoost algoritmus alkalmazása után

6.2.1. Jelmagyarázat a kördiagramokhoz

Vizsgáljuk meg jobban a maradék modellünkhöz tartozó igazságmátrixainkat. Kördiagramokon ábrázolva a különböző adatokat jobban szemléltethető az egyes osztályok eloszlása. Mind a négy kiválasztott modellhez három plusz egy darab kördiagramon szeretném szemléltetni, hogy melyik modell mire lehet alkalmas.

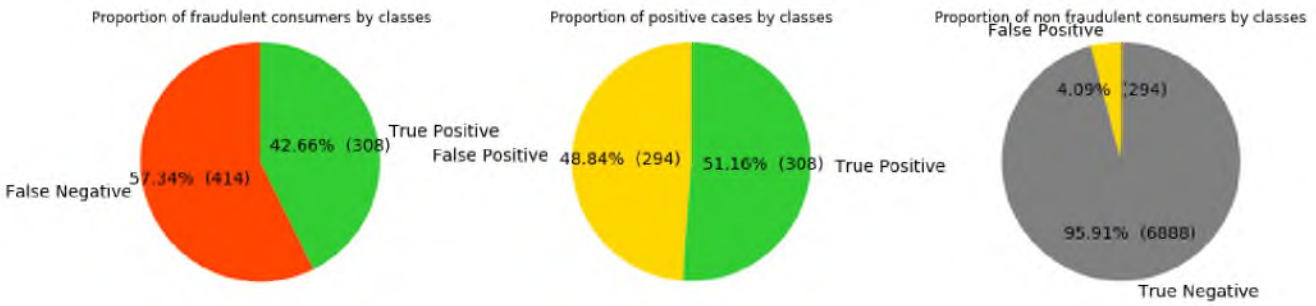
Minden diagramon a megegyező színek ugyanazt az adathalmazt takarják. Zöld szín jellemzi a csalókat, akiket sikerült jó osztályba elhelyeznünk, tehát a true positive eseteket. Piros színnel vannak jelölve azok a csalók, akiket a modellünk a negatív osztályba sorolt, tehát nem sikerült jól osztályozni őket, ezek a false negative esetek. Sárga színnel az olyan tisztességes fogyasztókat jelölöm, akik annak ellenére, hogy nem csalók a modellünk csaló osztályba sorolta őket, tehát a false positive esetek. Végül szürke a jó helyre sorolt nem csaló felhasználók, a true negative esetek.

Az első kördiagram az összes csalónk eloszlását jelzi aszerint, hogy pozitív vagy negatív osztályba lettek-e sorolva. A második diagram az összes olyan felhasználót ábrázolja, akiket a modellünk a csaló osztályba sorolt, míg a harmadik diagram az összes nem fraud fogyasztónkat aszerint, hogy melyik osztályba soroltuk őket.

A különálló kördiagram a teljes fogyasztóbázisunkat reprezentálja annak tekintetében, hogy a két osztályba összesen hány felhasználót sorolt a modellünk, ebből következtetve arra, hogy az elért eredmény után a fogyasztókhoz tartozó mérőórák fizikai ellenőrzésére hány esetben van szükség. A kék szín a negatív tehát nem csaló, míg a narancssárga a pozitív tehát csalóként osztályozott eseteket jelöli.

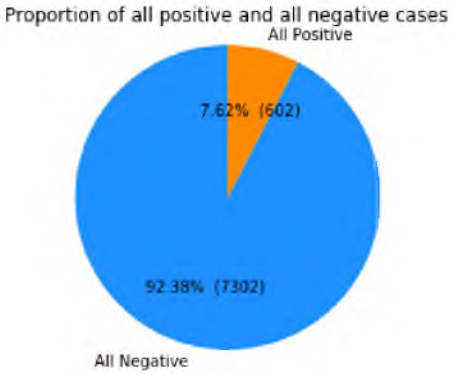
6.2.2. XGBoost hangolt beállításokkal

Az XGBoost optimalizált verzióján azt láthatjuk, hogy a nem csaló osztályt nagyon jól osztályozza, csak minimális része a hasonló felhasználóknak került a másik osztályba. Emellett a pozitív osztályunk 42,66%-át sikeresen megtalálja úgy, hogy a pozitív osztályba, tehát csalóként osztályozott felhasználóink több mint fele ténylegesen csaló. Összességében ez a modell nem nagy veszteséggel (kevés téves fizikai ellenőrzéssel) a közel felét megtalálja a csalóknak, ami ígéretes kezdésképp.



6.2.1. ábra: A hangolt XGBoost algoritmushoz tartozó kördiagramok a 6.2.1. fejezet második és harmadik bekezdésében leírtak alapján

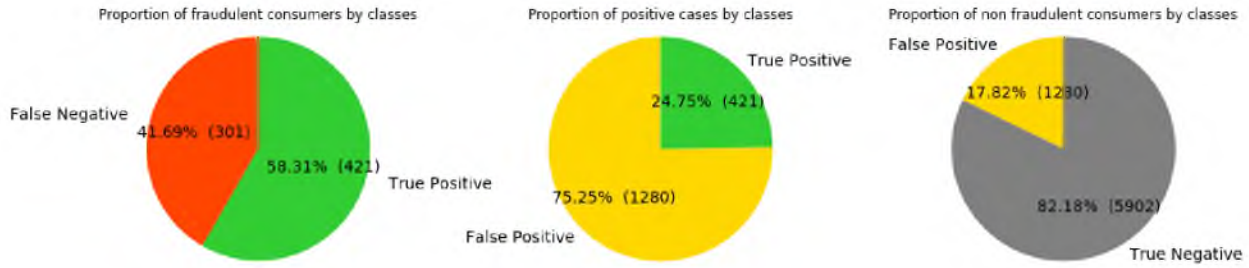
Ezekhez az eredményekhez egészen kedvező, csak a fogyasztóink 7,62%-t kell ellenőriznünk, mint gyanús eseteket, hogy megbizonyosodjunk modellünk jó osztályba sorolta-e őket.



6.2.2. ábra: A hangolt XGBoost algoritmushoz tartozó kördiagram a 6.2.1. fejezet negyedik bekezdésében leírtak alapján

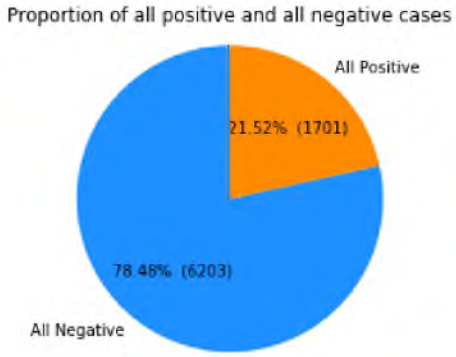
6.2.3. XGBoost alapbeállításokkal és SMOTE adatbővítéssel

Ugyan az XGBoost alapbeállításaival és SMOTE adatbővítéssel elért eredményeken látszódik, hogy több csalót sikerült kiszűrnie a modellnek, viszont ehhez már szükséges volt az is, hogy a csalóként osztályozott fogyasztóinknak csak majdnem negyede legyen pozitív eset igazából.



6.2.3. ábra: Az alapbeállításokkal és SMOTE-tal rendelkező XGBoost algoritmushoz tartozó kördiagramok a 6.2.1. fejezet második és harmadik bekezdésében leírtak alapján

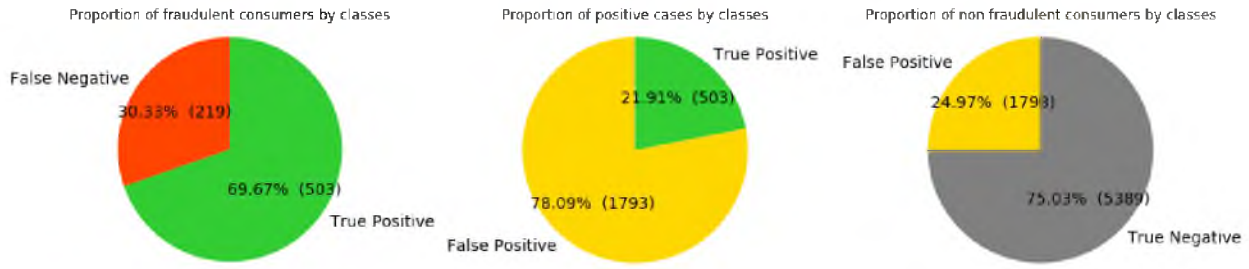
Ezeknek az eredményeknek az eléréséhez a teljes felhasználóbázisunk 21,52%-át további ellenőrzéseknek kell alávetnünk, hogy megbizonyosodjunk fradulent viselkedéséről.



6.2.4. ábra: Az alapbeállításokkal és SMOTE-tal rendelkező XGBoost algoritmushoz tartozó kördiagram a 6.2.1. fejezet negyedik bekezdésében leírtak alapján

6.2.4. XGBoost hangolt beállításokkal és SMOTE adatbővítéssel

Ellenben, ha optimalizáljuk az algoritmust és utána a SMOTE adatbővítés alkalmazzuk rá a következő eredményeket kapjuk:

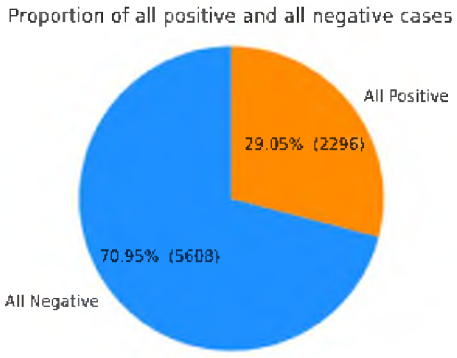


6.2.5. ábra: A hangolt beállításokkal és SMOTE-tal rendelkező XGBoost algoritmushoz tartozó kördiagramok a 6.2.1. fejezet második és harmadik bekezdésében leírtak alapján

Láthatjuk, hogy ez a modell már az első kördiagramon, tehát az összes csaló esetében sokkal jobban teljesít, hiszen több mint kétharmadát sikeresen kiszűri az adathalmazból. Ellenben

ennek megvan az ára is, hiszen második és harmadik diagramon látszódik, hogy sokkal több, valójában nem csaló fogyasztót sorol a pozitív osztályba, ami azt jelenti, hogy sokkal több felhasználót kell fizikailag is ellenőrizni, és ezek négyötöde feleslegesnek bizonyul.

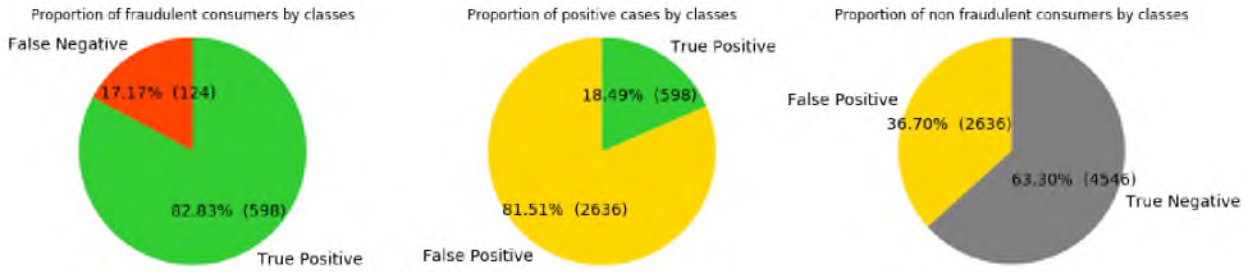
Ez azt jelenti, hogy ennek az eredménynek az eléréséhez a teljes fogyasztóbázisunk 29,05%-át további ellenőrzésnek kellett kitennünk, ami egy nagy szolgáltatócég esetében rengeteg erőforrás és idő, ami nem feltétlenül éri meg.



6.2.6. ábra: A hangolt beállításokkal és SMOTE-tal rendelkező XGBoost algoritmushoz tartozó kördiagram a 6.2.1. fejezet negyedik bekezdésében leírtak alapján

6.2.5. XGBoost hangolt beállításokkal és SMOTEENN adatbővítéssel

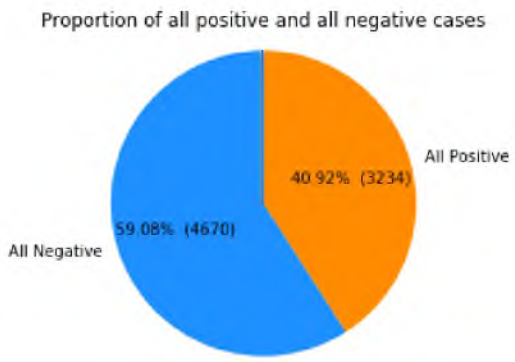
Az optimalizált algoritmus SMOTEENN adatbővítés után pedig ezeket az eredményeket produkálta:



6.2.7. ábra: A hangolt beállításokkal és SMOTEENN-al rendelkező XGBoost algoritmushoz tartozó kördiagramok a 6.2.1. fejezet második és harmadik bekezdésében leírtak alapján

A csalók kiszűrése itt már szinte hibátlan, 17,17%-ukat „hagyja futni”, ami elég jó eredmény, viszont itt is ugyanaz a probléma, mint a SMOTE esetében is volt. Rengeteg, nem csaló fogyasztót osztályoz csalóként, ami sok fizikai ellenőrzéshez vezet, ami egy szint felett már

kevésbé költséghatékony, mintha kevesebb csalót találna meg a modell, de kevesebb negatív osztályba tartozó fogyasztót is sorolna rossz helyre.



6.2.8. ábra: A hangolt beállításokkal és SMOTEENN-al rendelkező XGBoost algoritmushoz tartozó kördiagram a 6.2.1. fejezet negyedik bekezdésében leírtak alapján

A felhasználóink 40%-ának ellenőrzésének erőforrásigénye már valószínűleg túllépi az összes csalónk által okozott veszteségeket, viszont egy a feladatra alkalmas adathalmaz építéséhez egy megfelelő méretű, nem túl nagy fogyasztószámmal rendelkező régiót tesztcsoportként használva megfelelő modell lehet.

6.2.6. Összegzés

A négy modell tanítási idők terén és a modell illesztése alatti legmagasabb memóriahasználat terén a következőképpen néz ki specifikációkkal együtt:

Intel64 Family 6 Model 61 Stepping 4, GenuineIntel
Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz

	Training time (sec)	Peak memory usage (MB)
default + SMOTE	293,88	717,68
tuned	468,05	654,22
tuned + SMOTE	1087,34	717,67
tuned + SMOTEENN	821,35	546,49

6.2.2. táblázat: A négy jobban vizsgált modell tanítási ideje és a legmakasabb memóriahasználata a tanulás során

Az alapalgoritmus SMOTE-tal alkalmazva és a hangolt algoritmus között nincs nagy különbség sem tanítási idő, sem pedig memóriahasználat terén, ellenben a hangolt algoritmus SMOTE-tal már négyszer olyan hosszú tanítási idővel rendelkezik, mint az előbbi. Az egyetlen modellem, amin a SMOTEENN adatbővítést használtam a többihez képest alacsonyabb a legmagasabb memóriahasználat értéke, de nem szabad elfelejteni, hogy míg a SMOTE esetében

az adatbővítés csak 15 másodperc körüli idő alatt jött létre, addig a SMOTEENN esetében több mint 3000 másodpercre volt szükség.

Összességében a négy algoritmus közötti választás leginkább a prioritásokon, a fizikai ellenőrzésekhez szükséges költségeken és a szolgáltató cég által becsült nem szabályos mintavételezés által elszenvedett veszteségeken múlik.

Már a legkevésbé radikális modell, a hangolt XGBoost is képes kiszűrni a csalók közel felét, mindeközben még mindig csak a felhasználók 7,62%-át szükséges ellenőriznünk, ami elég ígéretes és értékeit tekintve talán a legjobb modellünk.

A maradék három modell esetében a megbízhatóság értékek annyira lecsökkennek, hogy az optimalizált algoritmus és SMOTE alkalmazása esetén 28,2% míg az optimalizált algoritmus és SMOTEENN alkalmazása esetén 41,12%-a az adathalmazunknak csalónak van titulálva a modell által, amik olyan magas értékek, hogy valószínűtlen, hogy költséghatékonyan kivitelezhető az összes fogyasztó ellenőrzése.

Az XGBoost alap algoritmusa SMOTE segítségével még ígéretes lehet, az előbb említett modellek mellett sok felhasználó kerül a pozitív osztályba annak ellenére, hogy a negatív osztályba kéne tartozniuk, de közel sem annyi, mint az előzőeknél.

Mindent összegezve, az első modell, tehát az XGBoost hangolt modellje tűnik a legjobb választásnak a feladatra, hiszen tanítási ideje és memóriahasználata sem kiugró, emellett az elért eredmények is ígéretesek. Azt tekintve, hogy a fraud detection az energiaszolgáltatók körében hasonló módszerekkel még nem elterjedt és kevés adatbázis áll rendelkezésünkre ezen a téren a modellünk igen hatékony, és a további adatgyűjtések alatt megállja a helyét.

Irodalomjegyzék

[1] Microsoft Azure: Mi a gépi tanulás?, https://azure.microsoft.com/hu-hu/overview/what-is-machine-learning-platform/

[2] henryRDlab: Electricity Theft Detection, adathalmaz, https://github.com/henryRDlab/ElectricityTheftDetection

[3] State Grid Corporation of China https://www.sgcc.com.cn

[4] Koo Ping Shung: Accuracy, Precision, Recall or F1?, https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9/ (2018. március 15.)

[5] Chris I.: Evaluating ML Models: Precision, Recall, F1 and Accuracy, https://medium.com/analytics-vidhya/evaluating-ml-models-precision-recall-f1-and-accuracy-f734e9fcc0d3/ (2019. szeptember 2.)

[6] Jason Brownlee: Tour of Evaluation Metrics for Imbalanced Classification, https://machinelearningmastery.com/tour-of-evaluation-metrics-for-imbalanced-classification/ (2020. január 8., utoljára frissítve 2021. május 1.)

[7] Ankit Bhatia: Accuracy will not give the correct picture always….., https://medium.com/@ab9.bhatia/the-simplest-model-evaluation-metric-for-classification-models-is-accuracy-it-is-the-percentage-of-75290a9aa126/ (2018. június 25.)

[8] Scikit Learn: *slearn.metrics.f1_score függvény dokumentációja,* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

[9] Jason Brownlee: How to Use ROC Curves and Precision-Recall Curves for Classification in Python, https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/ (2018. augusztus 31.)

[10] Scikit Learn: *Precision-Recall dokumentációja,* https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_curve.html

[11] Scikit Learn: *sklearn.metrics.confusion_matrix dokumentációja*, https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html

[12] Jason Brownlee: A Tour of Machine Learning Algorithms, https://machinelearningmastery.com/a-tour-of-machine-learning-algorithms/ (2019. augusztus 12. utoljára frissítve 2020. augusztus 14.)

[13] Thomas W. Edgar, David O. Manz, in Research Methods for Cyber Security: Logistic Regression https://www.sciencedirect.com/topics/computer-science/logistic-regression/ (2017)

[14] Dr Neil Gerry: Data Science: Logistic Regression Case Stud, http://www.masteringdatascience.co.uk/CaseStudy_LogisticRegression.html (2017-18)

[15] Niklas Donges: A complete guide to the random forest algorithm, https://builtin.com/data-science/random-forest-algorithm/ (2019. június 16., utoljára frissítve 2020. szeptember 3.)

[16] z_ai: Rendom Forest Explained, https://towardsdatascience.com/random-forest-explained-7eae084f3ebe/ (2020. szeptember 16.)

[17] Abhishek Sharma: Decision Tree vs. Random Forest – Which Algorithm Should you Use?, https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/ (2020. május 12.)

[18] Jason Brownlee: A Gentle Introduction to XGBoost for Applied Machine Learning, https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ (2016. augusztus 17.)

[19] raman_257: Boosting in Machine Learning | Boosting and AdaBoost, https://www.geeksforgeeks.org/boosting-in-machine-learning-boosting-and-adaboost/ (2019. május 06.)

[20] Vishal Morde: XGBoost Algorithm: Long May She Reign!, https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d/ (2019. április 8.)

[21] Jason Brownlee: SMOTE for Imbalanced Classifivation with Python, https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/ (2020. január 17., utoljára frissítve 2021. március 17.)

[22] Happy95: Overcoming Class Imbalance using SMOTE Techniques, https://www.analyticsvidhya.com/blog/2020/10/overcoming-class-imbalance-using-smote-techniques/ (2020. október 6.)

[23] Scikit Learn: *sklearn.linear_model.LogisticRegression dokumentációja*, https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[24] Scikit Learn: *sklearn.ensemble.RandomForestClassifier dokumentációja*, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[25] XGBoost: *XGBoost parameters dokumentációja,* https://xgboost.readthedocs.io/en/latest/parameter.html

[26] A szakdolgozat során használt kódok: https://github.com/botibabak/thesis