

Primjena metoda strojnog učenja u otkrivanju financijskih prevara

Knezić, Ivan

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:217:347949>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-12-27**



Repository / Repozitorij:

[Repository of the Faculty of Science - University of Zagreb](#)



Primjena metoda strojnog učenja u otkrivanju financijskih prevara

Knezić, Ivan

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: <https://um.nsk.hr/um:nbn:hr:217:347949>

Rights / Prava: [In copyright](#) / [Zaštićeno autorskim pravom.](#)

Download date / Datum preuzimanja: **2024-06-18**



Repository / Repozitorij:

[Repository of the Faculty of Science - University of Zagreb](#)



SVEUČILIŠTE U ZAGREBU
PRIRODOSLOVNO–MATEMATIČKI FAKULTET
MATEMATIČKI ODSJEK

Ivan Knezić

**PRIMJENA METODA STROJNOG
UČENJA U OTKRIVANJU
FINANCIJSKIH PREVARA**

Diplomski rad

Voditelj rada:
Prof. dr. sc. Luka Grubišić

Zagreb, rujan, 2020

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Ovaj rad posvećujem obitelji i prijateljima koji su uvijek vjerovali u mene i pomogli mi da fizički i psihički završim studij. Posebna zahvala ide mom mentoru dr. sc. Luki Grubišiću bez kojeg ovaj rad ne bi bilo moguće napisati.

Sadržaj

Sadržaj	iv
Uvod	2
1 Općenito	3
1.1 O skupu podataka za otkrivanje kartičnih prevara	3
1.2 Evaluacija modela	6
2 Problemi s neuravnoteženim skupovima podataka	12
2.1 Neuravnoteženi skup podataka	12
2.2 Manjak informacija	13
2.3 Separabilnost klasa	14
2.4 Mali disjunki	15
3 Predobrada na neuravnoteženim skupovima podataka	17
3.1 Priprema podataka	17
3.2 Poduzorkovanje	22
3.3 Preuzorkovanje	30
3.4 Kombinacije metoda ponovnog uzorkovanja	39
4 Klasifikacija na neuravnoteženim skupovima podataka	45
4.1 O klasifikaciji	45
4.2 Logistička regresija	45
4.3 Stablo odluke	48
4.4 Bagging	53
4.5 Slučajne šume	54
4.6 Boosting	56
4.7 Metoda najbližih susjeda	59
4.8 Metoda potpornih vektora	61
4.9 Glasanje	63

5	Eksperimentalna procjena	64
5.1	Korištene metode	64
5.2	Učenje na originalnom skupu podataka	65
5.3	Učenje na poduzorkovanom skupu podataka	66
5.4	Učenje na preuzorkovanom skupu podataka	69
5.5	Učenje na kombiniranim tehnikama	73
6	Zaključak	77
A	Dodatak	78
	Bibliografija	80

Uvod

Još donedavno su krađe i prevare bile jednostavne. Kradljivac bi pomoću željezne poluge provalio u trgovinu tijekom noći ili bi zaprijetio trgovcu nekom vrstom oružja, kako bi ukrao novac ili druge dragocjenosti. No, kako se svijet razvija tako i kradljivci razvijaju modernije alate za krađu. Kako se u današnje vrijeme već većina transakcija izvršava elektronički, tako su i kradljivci razvili razne programe, odnosno uređaje koji im omogućuju pristup privatnim podacima (bankovnim računima, transakcijama, lozinkama, identifikacijskim brojevima, ...) korisnika koje koriste da bi im 'skinuli' novac s računa. Neke od tih modernijih metoda uključuju lažne mailove, prevarantske telefonske pozive, lažne nagradne igre, različite viruse, krađu preko javno dostupne (lozinkom nezaštićene) Wi-Fi mreže i slično. Od modernih uređaja postoje RFID (Radio-frequency identification), odnosno NFC (Near Field Communication) čitači koji se koriste da bi kradljivcima 'dohvatili' kartične podatke prolaznika ili primjerice putnika gradskog prijevoza. Kao što iz navedenih primjera vidimo elektroničke prevare su već sada jako veliki problem za banke i njihove korisnike, a sve vodi ka tome da bi mogao postati i veći budući da se danas za različite transakcije i plaćanja već mogu koristiti pametni mobiteli ili pametni satovi (dosta ljudi već koristi takve mogućnosti). Ako uz to još napomenemo da su i kriptovalute zadnjih godina postale dosta popularne (budući da su univerzalne vjerojatno će se u skoroj budućnosti i one koristiti za različite transakcije), možemo naslutiti da će vrlo brzo papirnati novac nestati s tržišta te će se praktično sve transakcije provoditi elektroničkim putem (navodno Švedska već do 2023. godine planira potpuno ukinuti plaćanje gotovinom). U ovom radu upravo proučavamo kako je moguće zaštititi ljude od transakcijskih prevara (konkretno, proučavamo kako je moguće koristiti strojno učenje da bismo odredili je li neka transakcija prevara ili nije). U vrijeme pisanja ovog rada banke ili već imaju neki algoritam strojnog učenja koji bi im 'olakšao' prepoznavanje transakcijskih prevara, ili intenzivno rade na njegovoj implementaciji. Da bismo prikazali kako je moguće upotrijebiti algoritme strojnog učenja u ovom radu koristimo Credit Card Fraud Detection Dataset, odnosno skup podataka iz creditcard.csv datoteke (dostupne na kaggleu na adresi <https://www.kaggle.com/mlg-ulb/creditcardfraud/>). Cilj je koristeći algoritme strojnog učenja trenirane na ranije prikupljenim podacima prepoznati je li neka transakcija prevara ili nije. Problem prepoznavanja prevara među transakcijama je klasifikacijski problem

zato što individualna transakcija može biti prevara (u ovom radu klasu prevara označavamo brojem 1) ili valjana transakcija (tu klasu označavamo brojem 0), odnosno output je kategorički (poprima konačno mnogo vrijednosti). Kao što i možemo pretpostaviti taj skup podataka je vrlo neuravnotežen, odnosno puno je više valjanih transakcija nego prevara, što bi moglo negativno utjecati na klasifikaciju, odnosno klasifikatori bi mogli prepoznati manje prevara zbog pristranosti u korist valjanih transakcija. Prirodno rješenje tog problema bi bilo da se prikupi još podataka, no to je ili nemoguće u većini slučajeva, ili bi zahtijevalo vrlo snažan hardver (obzirom na količinu podataka). Zato da bismo otklonili pristranost klasifikatora prema većinskoj klasi koristimo različite metode ponovnog uzorkovanja, konkretno različite metode poduzorkovanja i preuzorkovanja. Poduzorkovanjem smanjujemo broj primjera većinske klase, dok preuzorkovanjem povećavamo broj primjera manjinske klase. Spomenimo da se u literaturi manjinsku klasu (prevare) često naziva i pozitivnom klasom, dok se većinska klasa (valjane transakcije) naziva negativnom klasom. Problem neuravnoteženosti klasa također se pojavljuje u većini skupova podataka vezanih za stvarni svijet (uključujući prepoznavanje tumora na rendgenskim slikama, prepoznavanje govora, prepoznavanje izljeva ulja u oceane na satelitskim snimkama, prepoznavanje anomalija i još mnogo drugih) i u svakom od tih skupova podataka, kao i u ovom o transakcijskim prevarama, nam je zanimljivija manjinska (pozitivna) klasa. U nastavku ovog rada ćemo opisati različite metode poduzorkovanja i preuzorkovanja, kako ih je i kada moguće koristiti i zašto. Prilikom eksperimenta koristit ćemo različite klasifikatore kako bismo im naposljetku mogli usporediti performanse, te ćemo objasniti koje metode se koriste za evaluaciju modela i zašto. Nastavak rada je podijeljen na sljedeći način. U prvom poglavlju opisujemo skup podataka te koje metrike je najbolje koristiti u skupovima podataka kod kojih postoji problem neuravnoteženosti klasa (primjerice zašto *točnost* nije dobra metrika). U drugom poglavlju opisujemo na kakve točno izazove nailazimo kada se bavimo klasifikacijom na neuravnoteženim skupovima podataka. U trećem poglavlju opisujemo razne tehnike poduzorkovanja i preuzorkovanja koje koristimo kada se radi o klasifikaciji na neuravnoteženim skupovima podataka. U četvrtom poglavlju opisujemo klasifikatore koje koristimo u eksperimentu. U petom poglavlju ćemo predstaviti rezultate eksperimenta (na Credit Card Fraud Detection Datasetu). U šestom poglavlju ćemo donijeti zaključak na temelju eksperimenta. Rad također sadrži i dodatak u kojem je opisano kako pripremiti podatke i koje pakete je potrebno instalirati (i kako) da bismo koristili metode koje smo koristili prilikom eksperimenta.

Poglavlje 1

Općenito

1.1 O skupu podataka za otkrivanje kartičnih prevara

Sadržaj

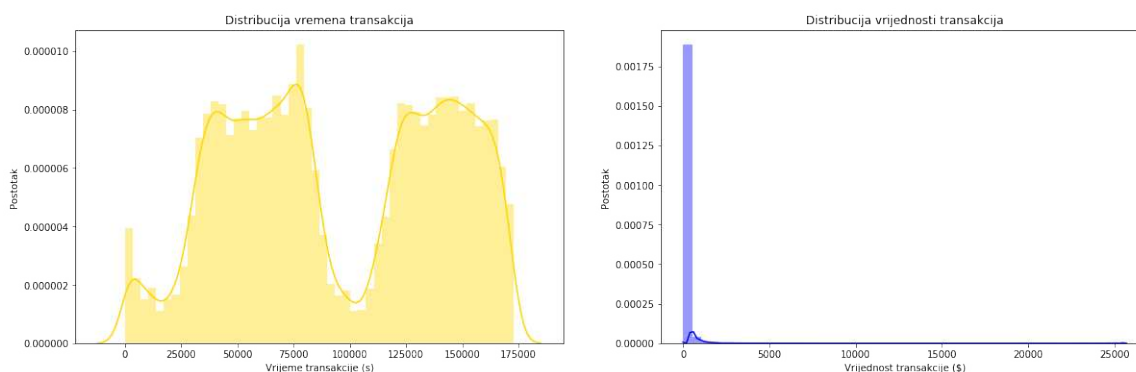
U uvodnom dijelu smo već spomenuli da ćemo za eksperiment koristiti skup podataka za prepoznavanje kartičnih prevara (Credit Card Fraud Detection Dataset) koji je javno dostupan na Kaggleu na adresi <https://www.kaggle.com/mlg-ulb/creditcardfraud/>. Podaci su prikupljeni tijekom istraživanja vezanog za prepoznavanje prevara koje su zajednički provodile *Worldline* (francuska firma koja pruža razne transakcijske usluge) i *Machine Learning Group* (<http://mlg.ulb.ac.be>) of *ULB (Université Libre de Bruxelles)*. Taj skup podataka sadrži informacije o transakcijama koje su provedene kroz 2 uzastopna dana tijekom rujna 2013. godine. Poznato je da svaka transakcija ima 31 atribut (značajku) pri čemu je posljednja (31.) značajka oznaka klase, odnosno 0 (ako je transakcija valjana) ili 1 (ako je transakcija prevara). Zbog sigurnosnih razloga (informacije vezane za kreditne kartice su povjerljive) poznato je jedino što 2 značajke (od ostalih 30) obilježavaju, a to su 'Time' i 'Amount'. Preostalih 28 je preimenovano te spremljeno pod nazivima V1, V2, V3, ... V28. Također napomenimo da je poznato da su sve varijable numeričkog tipa te je na anonimiziranim varijablama (V1, ... V28) provedena PCA (Principal Component Analysis, hrv. analiza glavnih komponentata) transformacija (na 'Time' i 'Amount' nije). Značajka 'Time' obilježava vrijeme koje je proteklo od provođenja prve transakcije (prva transakcija ima vrijednost varijable 'Time' jednako 0) u sekundama. Dakle, ako je za neku transakciju varijabla 'Time' jednaka 5000, tada je ta transakcija izvršena 1 sat, 23 minute i 20 sekundi nakon prve transakcije. S druge strane značajka 'Amount' obilježava kolika je novčana vrijednost transakcije, odnosno koliki je iznos neke vrste plaćanja karticom.

Analiza

Da bismo analizirali podatke koristili smo Jupyter Notebook (programski jezik Python) iz Anaconda okruženja zato što možemo koristiti već ugrađene funkcije za vizualizaciju i prikaz podataka. S obzirom na to da je većina značajki anonimizirana (ne znamo što predstavljaju) detaljnije možemo analizirati samo značajke 'Time' i 'Amount'.

Prvo što smo analizom otkrili je da ni za jednu transakciju niti jedna značajka ne prima *NULL* vrijednosti. To će biti značajno kod treniranja modela jer znamo da nema podataka koji su nepoznati ili nedostaju, to jest model neće gubiti nikakve informacije tijekom treniranja.

Skup podataka koji promatramo sadrži 284807 transakcije od kojih je 284315 valjanih transakcija i samo 492 prevare. To znači da valjane transakcije čine čak 99.83% ukupnih transakcija (svega 0.17% su prevare). Takvi podaci su i očekivani jer očekujemo da većina ljudi koristi kartice za svakodnevne transakcije. To znači da je skup podataka vrlo neuravnotežen što je ustvari najveći problem kojeg ćemo proučavati u narednim poglavljima ovog rada.



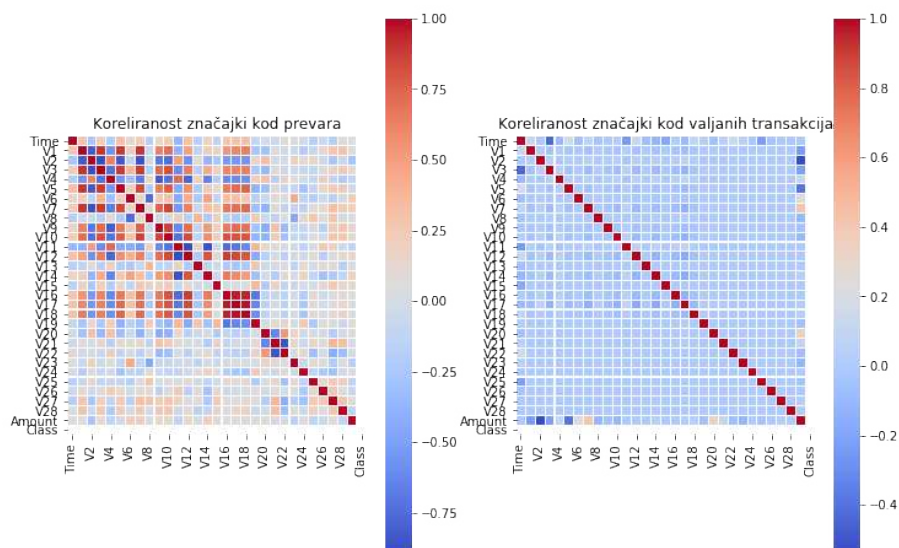
Slika 1.1: Distribucija vremena i vrijednosti transakcija

Prethodni dijagram (slika 1.1) prikazuje distribuciju vremena i vrijednosti transakcija, odnosno kada su se izvršavale transakcije i kolika je njihova novčana vrijednost. Iz dijagrama koji prikazuje distribuciju vremena (lijevo) izvršavanja transakcija možemo primijetiti da izgleda poput dvogrbe deve. Takav oblik vjerojatno poprma jer se radi o transakcijama tijekom dva uzastopna dana pa bismo mogli reći da 'nagli pad' između grba prikazuje noć, obzirom da tijekom noći nema ni približno jednako transakcija kao tijekom dana. Iz dijagrama koji prikazuje novčane vrijednosti transakcija (desni dijagram) možemo vidjeti da većina transakcija poprma iznose između 0\$ i 1000\$, ali da imamo i transakcije

koje poprimaju veće iznose. Razlog tome je vjerojatno korištenje kartica u svakodnevnim kupovinama kada računi za osnovne potrepštine nisu novčano visokih vrijednosti.

Zatim smo pomoću točkastog dijagrama pokušali prikazati razlike između valjanih transakcija i prevara, pri čemu smo za koordinatne osi postavili 'Time' i 'Amount'. Takvim prikazom nismo uspjeli uočiti postoje li neke razlike. Nakon što smo izvršili detaljniju analizu značajke 'Amount' uočili smo razlike među valjanim transakcijama i prevarama. Primjerice, prosječna vrijednost valjanih transakcija iznosi 88.29\$, dok je prosječna vrijednost prevara 122.21\$. Znači da je prosječna vrijednost transakcije koja je prevara oko 40% viša nego vrijednost valjane transakcija. Takva razlika bi mogla imati nekog značaja prilikom treniranja modela. Također primjećujemo veliku razliku između maksimalne vrijednosti prevara koja iznosi 2125.87\$ dok maksimalna vrijednost valjane transakcije iznosi čak 25691.16\$. Primijetimo da vrijednost valjane transakcije od 25691.16\$ nije toliko začuđujuća (budući da je korisnik kartice vjerojatno kupio novi auto ili platio modeliranje kuće ili stana). Isto ne možemo tvrditi za prevaru vrijednosti 2125.87\$, jer je to velik iznos ukraden samo jednom transakcijom. Da bismo detaljnije analizirali značajku 'Time' pretvorili smo vrijednosti značajke iz sekundi u sate i minute. Dobili smo da je prosječno vrijeme izvršavanja valjanih transakcija bilo u 14 sati i 3 minute, dok je prosječno vrijeme izvršavanja prevarantskih transakcija bilo u 11 sati i 39 minuta. Iz toga zaključujemo da su valjane transakcije provodile nešto kasnije u danu, no nakon što smo prikazali distribuciju značajke 'Time' (ovisno o klasi kojoj transakcija pripada) vidimo da se ne može uočiti značajna razlika. Kada razmislimo, intuitivno možemo zaključiti da kradljivci s kartica ne trebaju čekati sredinu noći da bi nekome ukrali novce, već samo prebace novac na svoj račun kada god misle da će transakcija biti neuočljiva. Iz tog razloga značajku 'Time' uopće nećemo koristiti za treniranje naših modela.

Naposljetku, pomoću distribucijskih dijagrama smo prikazali razlike između valjanih transakcija i prevara za preostalih 28 anonimiziranih značajki (V1, V2, ..., V28). Iz grafova smo uočili značajne razlike između valjanih transakcija i prevara kod sljedećih značajki: V3, V4, V9, V10, V11, V12, V14, V19. To znači da će prilikom treniranja modela te značajke najviše pridonositi modelu prilikom učenja (jer je na njima najlakše uočiti razlike pa je lakše odrediti o kakvoj se transakciji radi). Da bismo bolje prikazali koreliranost značajki koristili smo i korelacijsku matricu (na slici 1.2). Na toj slici vidimo razlike između korelacijske matrice za prevare (lijevo) i korelacijske matrice za valjane transakcije (desno). Iz tih matrica možemo primijetiti da su koeficijenti korelacije puno veći kod prevara nego kod valjanih transakcija (snažnija nijansa crvene pokazuje veću koreliranost, a u lijevoj matrici vidimo puno više crvenih kvadrata).



Slika 1.2: Korelacijske matrice za prevare i valjane transakcije

1.2 Evaluacija modela

Da bismo što bolje mogli usporediti različite modele za evaluaciju koristimo više različitih metrika. Jedna od standardnih metrika kojom određujemo performanse klasifikatora je konfuzijska matrica (prikazana na slici 1.3).

		Predicted	
		0	1
Actual	0	TN	FP
	1	FN	TP

Slika 1.3: Konfuzijska matrica

U konfuzijskoj matrici TP (True positives) predstavlja broj ispravno klasificiranih instanci pozitivne klase (u našem slučaju broj prevara koje je i model klasificirao kao prevare).

FP (False positives) predstavlja broj instanci negativne klase, a model ih je pogrešno klasificirao kao pozitivne (u našem slučaju broj valjanih transakcija koje je model klasificirao kao prevare), TN (True negatives) predstavlja broj instanci negativne klase koje je model ispravno klasificirao (u našem slučaju broj valjanih transakcija koje je model klasificirao kao valjane transakcije), a FN (False Negatives) predstavlja broj instanci pozitivne klase koje je model pogrešno klasificirao kao negativne (u našem slučaju broj prevara koje je model klasificirao kao valjane transakcije).

U ovom radu koristit ćemo konfuzijsku matricu da bismo mogli 'iščitati' konkretne brojčane vrijednosti. Primjerice koliko je stvarnih prevara model klasificirao kao valjane transakcije (FN - nalazi se u donjoj lijevoj kućici matrice). Također, da bismo dobili bolju intuiciju koliko model griješi koristimo i normaliziranu verziju konfuzijske matrice koja prikazuje relativne vrijednosti (odnosno u lijevoj donjoj kućici će biti postotak pogrešno klasificiranih prevara). Primijetimo da u normaliziranoj konfuzijskoj matrici zbroj brojeva i u gornjem retku i donjem retku iznosi 1.

Iz konfuzijske matrice možemo izvesti puno različitih evaluacijskih metrika. Jedna od najčešće korištenih je *točnost* (eng. *accuracy*). Točnost definiramo sljedećom formulom:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN} \quad (1.1)$$

Kao što iz formule vidimo *točnost* je definirana kao omjer ispravno klasificiranih primjera u odnosu na ukupan broj primjera. U našem slučaju (prepoznavanje kartičnih prevara) *točnost* ipak nije jedna od metrika koju ćemo koristiti. Za to postoje 2 razloga:

1. pozitivni primjeri su nam puno važniji od negativnih primjera (važnije nam je detektirati prevaru nego valjanu transakciju), a točnost daje jednaku važnost pozitivnim i negativnim primjerima.
2. iz prošlog odjeljka znamo da imamo puno više negativnih nego pozitivnih primjera (analizom smo dobili da skup podataka sadrži čak 99.83% negativnih primjera). To znači da bi model koji svaku transakciju klasificira kao valjanu postigao točnost od 99.83%, što je jako visoko. Problem je što taj model ne bi prepoznao niti jednu prevaru.

U nastavku ovog odjeljka ćemo definirati mjere koje ćemo ustvari koristiti za evaluaciju modela u ovom eksperimentu.

Preciznost i osjetljivost

Umjesto točnosti za evaluaciju koristimo *preciznost* (eng. *precision*) i *osjetljivost* (eng. *recall*). *Preciznost* i *osjetljivost* također izvodimo iz konfuzijske matrice, a zadane su

sljedećim formulama:

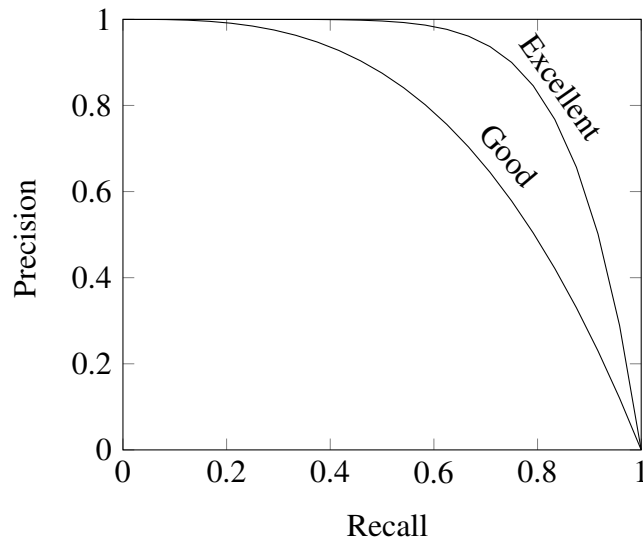
$$precision = \frac{TP}{TP + FP} \quad (1.2)$$

$$recall = \frac{TP}{TP + FN} \quad (1.3)$$

Vidimo da je *preciznost* definirana kao omjer ispravno klasificiranih primjera pozitivne klase u odnosu na ukupan broj primjera koje je model prepoznao kao pozitivne. Konkretno, u našem slučaju *preciznost* je omjer ispravno klasificiranih prevara u odnosu na broj svih transakcija koje je model prepoznao kao prevare. S druge strane *osjetljivost* je definirana kao omjer ispravno klasificiranih primjera pozitivne klase u odnosu na ukupan broj primjera koji stvarno pripadaju pozitivnoj klasi. Odnosno, u našem slučaju *osjetljivost* je omjer ispravno klasificiranih prevara u odnosu na sve transakcije koje su ustvari prevare.

Primijetimo da bismo željeli imati $recall \approx 1$, tj. želimo detektirati što više prevara (primijetimo da kada bismo svaku transakciju prepoznali kao prevaru bismo dobili $recall = 1$, ali bi preciznost bila jako niska, odnosno $precision < 0.01$). Dakle, idealno bismo željeli postići što višu osjetljivost bez da 'naštetimo' preciznosti. To je u praksi često nemoguće jer su te dvije mjere u konfliktu. Odnosno, kada bismo povećali broj ispravno klasificiranih prevara (TP), u većini slučajeva bi automatski i porastao broj pogrešno klasificiranih prevara (FP). Razlog tome je što će model općenito više transakcija prepoznati kao prevare, a malo je vjerojatno da će sve te 'novoprepoznate' prevare ustvari i biti prevare.

Da bismo grafički prikazali kompromis između osjetljivosti i preciznosti koristimo PR-krivulju (Precision Recall Curve). Pritom se na X-osi se nalaze vrijednosti koje poprima osjetljivost, dok se na Y-osi nalaze vrijednosti koje poprima preciznost. Možemo reći da je najbolji kompromis između preciznosti i osjetljivosti točka na krivulji koja se nalazi najbliže točki u desnom gornjem kutu (na koordinatama (1, 1)). Točka u desnom gornjem kutu je savršen slučaj, odnosno onaj u kojem su i osjetljivost i preciznost jednaki 1. Da bismo usporedili dvije PR-krivulje koristit ćemo AUPRC (Area Under Precision Recall Curve), odnosno površinu ispod PR-krivulje (veća površina znači bolji model). Na slici 1.4 možemo vidjeti razliku između PR-krivulja dobrog klasifikatora (označena s Good) i odličnog klasifikatora (označena s Excellent).



Slika 1.4: PR-krivulja

Za jednu PR-krivulju možemo reći da dominira nad drugom ako za svaku vrijednost Recall-a daje veći Precision (odnosno za istu vrijednost na X-osi, dominantna krivulja ima veću vrijednost na Y-osi). Uočimo da PR-krivulja označena s *Excellent* sa slike 1.4 dominira nad PR-krivuljom označenom s *Good*.

F_β -mjera

Sad kada smo definirali pojmove kao *preciznost* i *osjetljivost*, i uočili da obično kada pokušavamo povećati osjetljivost smanjujemo preciznost (ili obratno), možemo definirati F_β familiju mjera koje povezuju preciznost i osjetljivost. F_β -mjera je definirana formulom:

$$F_\beta = \frac{(1 + \beta^2) \cdot \textit{precision} \cdot \textit{recall}}{\beta^2 \cdot \textit{precision} + \textit{recall}} \quad (1.4)$$

U navedenoj formuli β je faktor koji koristimo da bismo preciznosti, odnosno osjetljivosti dodali 'težine'. Ovdje kao mjeru koristimo F_1 -mjeru koja se tradicionalno koristi kao savršeni balans između preciznosti i osjetljivosti (nijedna od te dvije mjere nije važnija). Napomenimo samo da bi F_2 mjera značila da je osjetljivost dvaput važnija od preciznosti, dok bi $F_{\frac{1}{2}}$ značila da je preciznost dvaput važnija od osjetljivosti. F_1 -mjeru dobivamo kada u općenitu formulu uvrstimo 1 umjesto β . Time dobivamo sljedeću formulu:

$$F_1 = \frac{2 \cdot \textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}} \quad (1.5)$$

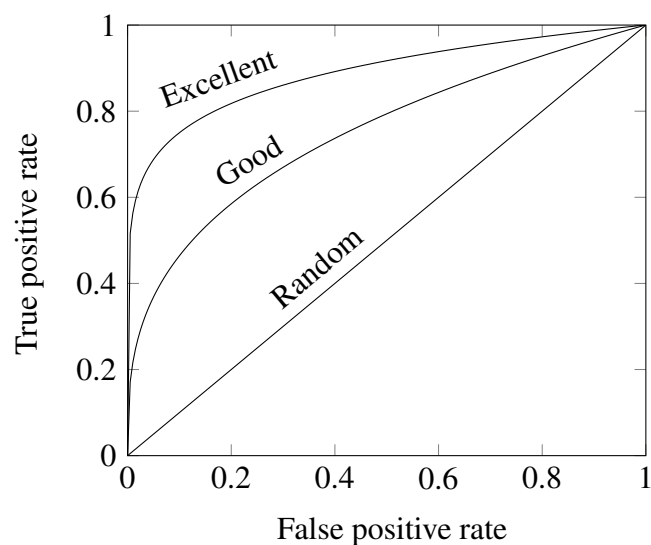
ROC-krivulja

Slično kao i kod PR-krivulje, ROC-krivulju (Receiver Operating Characteristic Curve) koristimo da bismo u ovom slučaju grafički prikazali TPR (True Positive Rate) u odnosu na FPR (False Positive Rate), pri čemu su TPR i FPR definirani sljedećim formulama:

$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \textit{recall} \quad (1.6)$$

$$\text{FPR} = \frac{\text{FP}}{\text{TN} + \text{FP}} \quad (1.7)$$

Kod ROC-krivulje na X-osi se nalaze vrijednosti koje poprima FPR, dok se na Y-osi nalaze vrijednosti koje poprima TPR. Za razliku od PR-krivulje gdje je idealna točka bila na koordinatama (1, 1) (gornji desni kut), kod ROC-krivulje je idealna točka na koordinatama (0, 1) (gornji lijevi kut). Primijetimo da u tom slučaju model ispravno prepoznaje sve pozitivne primjere (TPR = 1), te ne griješi prilikom klasifikacije nijednog primjera negativne klase (FPR = 0). Slično kao i kod PR-krivulja možemo reći kada jedna ROC-krivulja dominira nad drugom. Za jednu ROC-krivulju kažemo da dominira nad drugom ako za svaku vrijednost FPR-a daje veći TPR.



Slika 1.5: ROC-krivulja

Na slici 1.5 možemo vidjeti razliku između ROC-krivulja odličnog, dobrog i slučajnog klasifikatora. Primijetimo da ROC-krivulja označena s *Excellent* dominira nad drugom ROC-krivuljom označenom s *Good*. Kao što kod PR-krivulja koristimo AUPRC, kod ROC-krivulja

koristimo AUROC (Area Under Receiver Operating Characteristic). AUROC je korisna jer nam (isto kao i AUPRC) kao rezultat daje jednu brojčanu vrijednost po kojoj možemo uspoređivati klasifikatore.

Poglavlje 2

Problemi s neuravnoteženim skupovima podataka

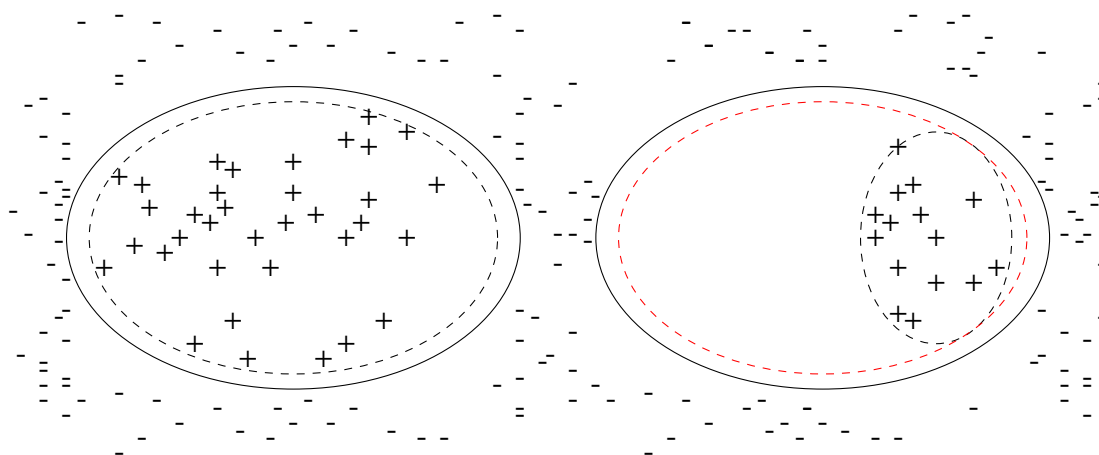
2.1 Neuravnoteženi skup podataka

U uvodu smo spomenuli da je neuravnoteženi skup podataka jedna od najvećih prepreka prilikom treniranja modela, odnosno skup podataka u kojem postoji puno više primjera koji pripadaju jednoj klasi u usporedbi s brojem primjera koji pripadaju drugoj klasi (ili klasama ako ih ima više). Spomenuli smo također da se problem neuravnoteženosti klasa javlja u svakodnevnim problemima, pa tako i u slučaju kada promatramo kartične prevare. Logično bi bilo prikupiti još podataka koji pripadaju manjinskoj klasi, no to je često ili nemoguće ili preskupo, a i novoprikupljeni podaci možda ne bi bili reprezentativni. Primjerice, u ovom skupu podataka imamo oko 280000 primjera negativne klase (valjanih transakcija) više nego primjera pozitivne klase (prevara) i trebalo bi dosta vremena da se prikupi dovoljno primjera pozitivne klase da bismo za učenje mogli koristiti uravnotežen skup. Osim toga što bi trebalo puno vremena (zbog problema očuvanja sigurnosti korisnika), bilo bi i skupo (skupoća automatski slijedi iz sigurnosnog problema), a podaci koje smo kasnije prikupili mogli bi se značajno razlikovati u odnosu na prethodno prikupljene (zbog mogućnosti da prevaranti razviju nove alate koje bi koristili za krađu). Iz upravo navedenih razloga potrebno je razviti algoritme koji postižu dobre rezultate i na neuravnoteženim skupovima podataka (algoritmi koje inače koristimo bi mogli pokazivati pristranost prema većinskoj klasi). Informaciju o neuravnoteženosti skupa podataka dobivamo iz *omjera neravnoteže* (IR - iz engleskog Imbalance Ratio). Ako je $IR = 1$, tada ima jednako primjera iz obje klase, no kod neuravnoteženih skupova podataka može se dogoditi da je $IR = 1 : 100$, pa čak i $IR = 1 : 10000$ ([12], [51]). Konkretno, za naš skup podataka vrijedi $IR \approx 1 : 578$. U [57] je eksperimentirano na tome kako IR utječe na rezultate klasifikacije te je pokazano da u približno balansiranim skupovima podataka ($IR \approx 1$) klasifikatori općenito daju nešto

bolje rezultate. Ipak, u [51] je istaknuto da nije sigurno koliki nerazmjer među klasama (koliko malen IR) mora biti da bi utjecao na (pogoršao) performanse modela. Također, ne možemo biti sigurni da je 1 : 1 najbolja moguća distribucija klasa u svakom slučaju. U našem slučaju savršena ravnoteža (1 : 1) klasa bi mogla dovesti do puno FP prilikom testiranja (upravo zbog jako niskog IR-a). Preciznije, nismo sigurni hoće li, zbog toga što su klase prirodno vrlo uneravnotežene, rezultati testiranja biti lošiji jer smo učili na uravnoteženom skupu podataka. U ostatku ovog poglavlja ćemo detaljno opisati koji su konkretno problemi kod rada s neuravnoteženim skupovima podataka.

2.2 Manjak informacija

Prvi problem kod klasifikacije u neuravnoteženim uvjetima nam predstavlja manjak informacija uzorkovan malim brojem primjera koji pripadaju pozitivnoj klasi (odnosno u našem slučaju manjak prevara u skupu za učenje). Taj manjak primjera pozitivne klase predstavlja problem jer se može dogoditi da algoritam koji koristimo za učenje možda ne uoči 'obrasce' koji su potrebni (specifični za pozitivnu klasu) da bi model ispravno donio odluku prilikom klasifikacije novih primjera.



Slika 2.1: Manjak informacija uzorkovan malim brojem primjera pozitivne klase

Na slici 2.1 možemo vidjeti kako manjak primjera pozitivne klase može loše utjecati na model, odnosno na određivanje granice između pozitivnih i negativnih primjera. Puna linija na slici označava stvarnu granicu između pozitivnih i negativnih primjera, dok iscrtkana linija označava granicu koju je model odredio. Vidimo da na lijevoj slici imamo dovoljno primjera pozitivne klase, pa model može lakše odrediti granicu upravo zato što učenjem na

više primjera pozitivne klase dobiva bolje razumijevanje obrazaca koji se u njoj pojavljuju. S druge strane, manjak primjera pozitivne klase (slika desno) onemogućava modelu prepoznavanje obrazaca, pa model ne može ispravno odrediti granicu između klasa, odnosno greška prilikom određivanja granice je veća na desnoj nego na lijevoj slici. Dakle, jednostavnom skicom smo pokazali da model može dobro naučiti granicu odluke onda kada ima adekvatan broj primjera pozitivne klase u skupu za učenje. Za razliku od toga, kada model nema dovoljno pozitivnih primjera u skupu za učenje ne može dovoljno dobro 'povući' granicu između klasa što rezultira puno većim brojem grešaka prilikom klasifikacije novih primjera. Odnosno, ako označimo s B stvarnu granicu (skup primjera koji su unutar elipse) između dviju klasa ($|B|$ označava broj primjera s unutarnje strane elipse), s B_L granicu koju je odredio model s lijeve slike, a s B_R granicu koju je odredio model s desne slike. Možemo vidjeti da vrijedi $B_R \subseteq B_L \subseteq B$ (da bismo lakše usporedili smo iscrtkanom crvenom crtom na desnoj slici nacrtana granica određena modelom s lijeve slike). Označimo s FN_L broj primjera pozitivne klase koje je model s lijeve slike prepoznao kao primjere negativne klase (*false negatives*), a s FN_R broj primjera pozitivne klase koje je model s desne slike prepoznao kao primjere negativne klase. Tada vrijedi:

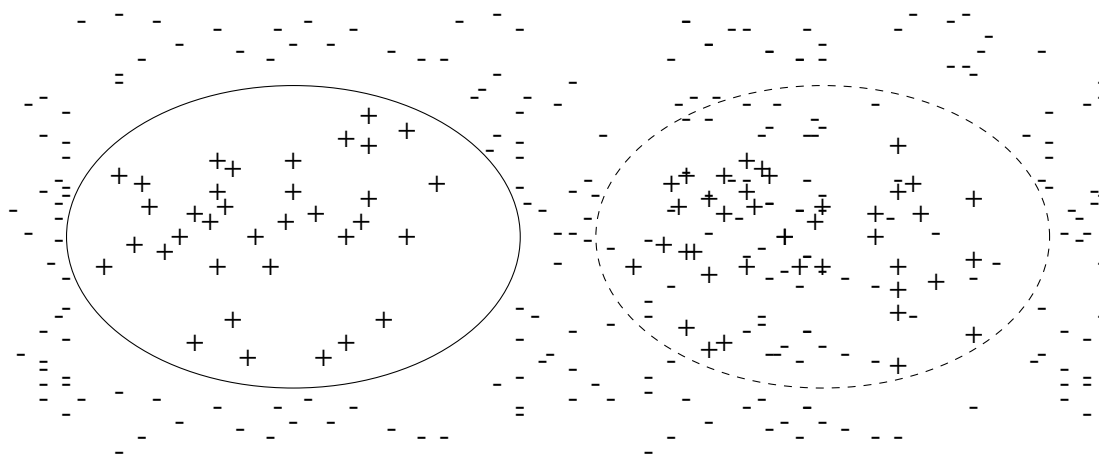
$$FN_L = |B| - |B_L| \leq \overbrace{|B| - |B_R|}^{B_R \subseteq B_L \Rightarrow |B_R| \leq |B_L|} = FN_R \quad (2.1)$$

Iz 2.1 vidimo da je $FN_L \leq FN_R$, odnosno model s desne slike pogrešno klasificira više primjera pozitivne klase (kao negativne) od modela s lijeve slike.

2.3 Separabilnost klasa

Sljedeći problem na koji nailazimo kada promatramo neuravnotežene skupove podataka je separabilnost klasa (također se koriste i termini preklopivost klasa te složenost klasa). Do problema separabilnosti klasa dolazi kada skup podataka sadrži primjere različitih klasa koji imaju slične ili iste karakteristike. Primjerice, zamislimo da kupujemo nešto preko interneta (redoviti smo kupac) koristeći elektronsku karticu. Svakako ćemo morati upisati PIN kako bismo potvrdili kupnju. U tom trenutku netko 'prisluškuje' tu transakciju i dohvća naš PIN te ga koristi kako bi na našu karticu naplatio i svoju kupnju. Te dvije transakcije mogu biti gotovo iste, s time da je razlika što je prevara izvršena par minuta nakon valjane transakcije koristeći drugi uređaj. Tada, zbog sličnosti transakcija, prevara može biti pogrešno klasificirana jer ima puno sličnosti s valjanom transakcijom, a pošto smo redoviti kupac, ta jedna transakcija može se naći među hrpom valjanih transakcija koje smo ranije (i kasnije) provodili. Vidimo da je u takvim slučajevima teško odvojiti (separirati) neke primjere manjinske klase od većinske, što automatski modelu otežava učenje obrazaca manjinske klase (pa time i otežava modelu stvaranje granice među klasama). U takvim situacijama model može pogrešno klasificirati pozitivni primjer (kao negativan),

jer je većinu sličnih značajki prilikom učenja vidio kod više tisuća negativnih primjera i možda svega par pozitivnih primjera (te pozitivne primjere je model vjerojatno shvatio kao 'šum' u podacima).



Slika 2.2: Problem preklapanja klasa

Sa slike 2.2 vidimo razliku između problema gdje je lako 'odijeliti' klase (slika lijevo) i problema gdje dolazi do preklapanja klasa (slika desno). Primijetimo da u slučaju kad nema preklapanja klasa (kao na slici lijevo) čak i jednostavan klasifikator može postići jako dobre rezultate (bez obzira na to što je broj negativnih puno veći nego broj pozitivnih primjera). Također možemo primijetiti da što je broj primjera pozitivne klase veći klasifikator će biti uspješniji. Preciznije, ako imamo veći broj primjera pozitivne klase, model ih neće nužno smatrati 'bukom', a i bit će mu dostupno više informacija. Tada model može pronaći detaljnije razlike među klasama. Ako se vratimo na raniji primjer s online kupovinom i u pretpostavimo da je prevarant izvršio više transakcija, model će možda uspjeti zaključiti da su prevarantske transakcije izvršene koristeći drugog operatera (jer bi ih bilo više pa bi imale veći utjecaj).

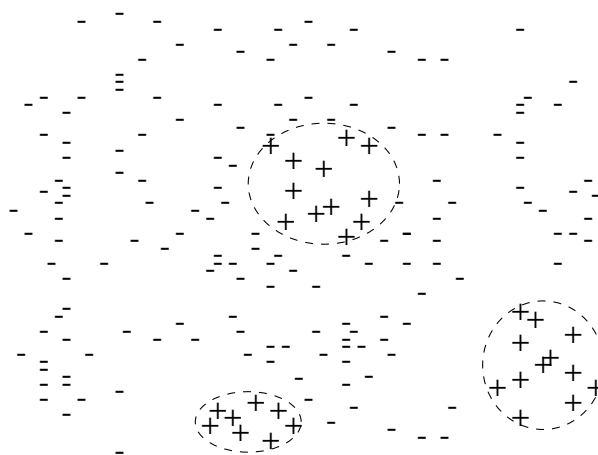
2.4 Mali disjunkt

Kod neuravnoteženih skupova podataka znamo da jedna klasa (negativna), sadrži puno veći broj primjera od druge klase (pozitivne). No, može se dogoditi da među samim primjerima pozitivne klase također postoji 'neuravnoteženost'. U tom kontekstu neuravnoteženost unutar same klase znači da se primjeri unutar te klase razlikuju (po karakteristikama) jedni od drugih. Odnosno, može se dogoditi da je skup pozitivnih primjera podijeljen na više

manjih skupova, gdje su svi primjeri unutar jednog od tih manjih skupova međusobno slični, ali su drugačiji od primjera u drugim manjim skupovima (svi manji skupovi su različiti jedan od drugoga). Takve manje skupove zovemo malim disjunktima. Iz [51] znamo da pojavljivanje malih disjunki na sljedeći način utječe na klasifikaciju:

1. prisustvo malih disjunki otežava učenje zbog veće složenosti skupa podataka
2. prisustvo malih disjunki je implicitno u većini slučajeva

Problem malih disjunki dodatno otežava problem neuravnoteženosti klasa zbog toga što većina pristupa pokušava riješiti problem neuravnoteženosti pozitivne i negativne klase, i pritom zanemaruju neuravnoteženost unutar same klase.



Slika 2.3: Mali disjunki

Na slici 2.3 vidimo grafički prikaz problema malih disjunki. Ilustrirajmo kako bi taj problem mogao izgledati na skupu podataka za prepoznavanje kartičnih prevara. U uvodu smo napomenuli da postoji više različitih metoda koje prevarantima omogućuju pristup kartičnim podacima korisnika (javna Wi-Fi mreža, lažni mailovi, prevarantski pozivi ...). Ti pristupi sami po sebi nisu slični, tako da će prevare koje su izvršene koristeći javnu Wi-Fi mrežu (za dohvaćanje podataka kartice) biti različitih karakteristika od onih koje koriste lažne nagradne igre. Također, sve prevare koje koriste javnu Wi-Fi mrežu će biti međusobno slične (isto vrijedi i za prevare korištenjem lažnih nagradnih igara), pa tako dobivamo više malih disjunki. Problem postaje još značajniji ako se još uz taj problem pojavljuje i neki drugi (npr. preklapanje klasa). Tada bi model jedan cijeli mali disjunkt mogao shvatiti kao 'šum'. U takvim situacijama bi bilo dobro iskoristiti neku od metoda preuzorkovanja pozitivne klase (da 'ojača' utjecaj primjera pozitivne klase prilikom učenja).

Poglavlje 3

Predobrada na neuravnoteženim skupovima podataka

3.1 Priprema podataka

Notacija

Označimo s $D = \{D_1, \dots, D_n\}$ neki skup podataka koji promatramo. S n označavamo broj primjera u tom skupu, odnosno vrijedi $|D| = n$. Svaki $D_i \in D$ je uređena $(m + 1)$ -torka za koju vrijedi $D_i = (\vec{x}_i, y_i)$, pri čemu je $\vec{x}_i = (x_{i_1}, \dots, x_{i_m})$ ulazni vektor gdje svaki $x_{i_j} \in \mathbb{R}$ označava jednu (j -tu) značajku (atribut) vektora \vec{x}_i , a $y_i \in \mathbb{C}$ ciljna varijabla (oznaka klase). Prepoznavanje kartičnih prevara klasifikacijski je problem, pa je vrijednost ciljne varijable kategorička, odnosno vrijedi $y_i \in \{0, 1\}$ obzirom da svaka transakcija može biti ili valjana (oznaka: 0), ili prevara (oznaka: 1). Također, u našem slučaju vrijedi $\vec{x}_i = (x_{i_1}, \dots, x_{i_{29}})$, obzirom da imamo 29 značajki (Amount-after-scaling, $T1, \dots, T28$). Primijetimo da značajku 'Time' ne koristimo za učenje modela (jer smatramo da vrijeme izvršavanja transakcija ne utječe na klasu). Značajku 'Amount' smo skalirali, budući da znamo da je nad preostalim značajkama provedena PCA transformacija. Cilj nadziranog učenja je koristeći dostupne podatke stvoriti model koji će predviđati oznake klasa još neviđenih (novih) primjera. Preciznije, cilj je naučiti nepoznatu funkciju f , takvu da vrijedi $y = f(\vec{x})$, koja će predviđati vrijednosti y za dotad neviđene primjere x . Doduše, funkciju f je u praksi nemoguće naučiti (f savršeno klasificira sve primjere), pa je najbolje što možemo naučiti funkciju h , takvu da $h(\vec{x}) \approx f(\vec{x})$. Funkciju h zovemo hipotezom funkcije f . Grešku koju funkcija h pravi prilikom klasifikacije obično mjerimo tako da 'prebrojimo' sve instance na kojima se vrijednost funkcije h (oznaka klase) razlikuje od vrijednosti funkcije f .

Učenje na neuravnoteženim skupovima podataka

U prethodnom poglavlju (2) spomenuli smo koje sve izazove možemo očekivati kada radimo s neuravnoteženim skupovima podataka. Od tih izazova najviše se ističe sama neuravnoteženost klasa (sjetimo se da skup podataka koji promatramo ima čak 99.83% negativnih primjera, odnosno samo 0.17% pozitivnih primjera). Zbog takvog omjera među klasama modeli će vjerojatno biti pristrani u korist negativne klase, pa će modeli prilikom klasifikacije novih primjera propustiti prepoznati velik broj prevara (velik broj *false negatives*). S obzirom na to da je cilj prepoznati prevare, takvi modeli nam neće biti od prevelike koristi. Da bi se taj problem pokušao riješiti, razne tehnike su razvijene. Te tehnike mogu biti podijeljene u 3 kategorije:

1. *Pristupi na razini podataka (ili eksterni pristupi)* - koriste *ponovno uzorkovanje* podataka (eng. *resampling*). Koristeći ovaj pristup pokušavamo smanjiti efekt neuravnoteženosti klasa (tako što balansiramo pozitivnu i negativnu klasu) i tako izbjeći veće modifikacije standardnih algoritama za učenje. Ponovno uzorkovanje se može postići tako da iz skupa za učenje uklonimo neke negativne primjere (taj postupak zovemo *poduzorkovanje*, eng. *undersampling*) ili da u skup za učenje dodamo pozitivne primjere (taj postupak zovemo *preuzorkovanjem*, eng. *oversampling*). Najveća prednost pristupa na razini podataka je nezavisnost predobrade podataka (ponovnog uzorkovanja) od modela, pa je stoga ovakav pristup lako prilagoditi mnogim različitim situacijama.
2. *Pristupi na razini algoritama (ili interni pristupi)* - pokušavaju prilagoditi modele tako da ih čine pristranijim u korist pozitivne klase. Da bi se primijenio algoritamski pristup potrebno je jako dobro poznavati klasifikator koji se koristi, ali i domenu na koju ga primjenjujemo (posebno je važno razumjeti zašto klasifikator postiže loše rezultate u slučaju neuravnoteženosti klasa). Možemo primijetiti da ovaj pristup 'naštimava' sam model, ali ne zahtijeva nikakvu predobradu podataka (za razliku od prethodnog).
3. *Troškovno-osjetljivi pristup* - hibridni pristup između pristupa na razini podataka i pristupa na razini algoritama. Kažemo 'hibridni' zato što uključuje neke karakteristike oba pristupa. Preciznije, na razini podataka dodjeljuje težine primjerima (pozitivni primjeri će imati veću težinu i tako smanjiti pristranost klasifikatora u korist negativne klase), dok na razini algoritama modificira klasifikatore tako da ubacuje primjerima pridijeljene težine u model kao faktor. Težine su zadane pomoću troškovne matrice koja sadržava cijene pogrešne klasifikacije. Konkretno, pošto u našem slučaju imamo samo dvije klase $Cost(1, 0)$ označava cijenu pogrešnog klasificiranja primjera pozitivne klase (kada model pozitivan primjer prepozna kao negativan), dok $Cost(0, 1)$ označava cijenu pogrešnog klasificiranja primjera negativne

klase (kada model negativan primjer prepozna kao pozitivan). Pošto je primjera pozitivne klase puno manje vrijedi $Cost(1, 0) > Cost(0, 1)$ - da bi se uklonila pristranost prema negativnoj klasi. Primijetimo da vrijedi $Cost(1, 1) = Cost(0, 0)$, odnosno ispravna klasifikacija se ne kažnjava.

Većinu ovog poglavlja posvećujemo raznim pristupima na razini podataka, odnosno na različitim tehnikama poduzorkovanja i preuzorkovanja.

Podjela podataka

Budući da nam je dostupan samo jedan skup podataka (*Credit Card Fraud Detection Dataset*), ne možemo ga cijelog iskoristiti za učenje modela (jer ne bismo mogli procijeniti performanse na neviđenim primjerima). Zato dijelimo skup podataka na dva dijela: skup za učenje i skup za testiranje, odnosno $D = D_{train} \cup D_{test}$ (D_{train} označava skup za učenje, a D_{test} skup za testiranje). Podatke dijelimo tako da 80% podataka stavljamo u D_{train} , a ostalih 20% u D_{test} . Prilikom takve podjele lako je napraviti velike greške čak i prije nego što uopće pokrenemo proces učenja.

1. **Podjela na skup za učenje i skup za testiranje** - U prethodnom odjeljku smo govorili o pristupima na razini podataka kod učenja na neuravnoteženim skupovima podataka. Jedna od većih grešaka prilikom podjele skupa podataka je podjela D na D_{train} i D_{test} **nakon** ponovnog uzorkovanja. Ilustrirajmo tu grešku na primjeru. Zamislimo da smo na skupu D proveli tehniku slučajnog preuzorkovanja (eng. random oversampling). Tada ćemo dobiti skup D_{os} koji ima jednak broj primjeraka pozitivne i negativne klase. Sjetimo se da za omjer neravnoteže (eng. imbalance ratio) na skupu podataka koji promatramo vrijedi $IR \approx 1 : 578$, što znači da je moguće da se prilikom provođenja slučajnog preuzorkovanja svaka prevara replicirati 577 puta. Pritom smo pretpostavili da se svaka prevara replicira jednak broj puta, što naravno ne mora biti slučaj. Kada bismo tek sada skup D_{os} podijelili na skup za učenje $D_{osTrain}$ i skup za testiranje D_{osTest} dobili bismo jako dobre rezultate (prilikom testiranja), ali ti rezultati ne bi bili od nikakve koristi. Razlog tome je što imamo 578 kopija svake prevare od kojih bi oko 100 bilo u testnom skupu. Tada bi sljedeća funkcija bila savršen klasifikator, odnosno vrijedilo bi: $f(\vec{x}) = h(\vec{x}) = y, \forall (\vec{x}, y) \in D_{osTest}$.

$$h(\vec{x}) = \begin{cases} 1, & \text{ako } (\vec{x}, 1) \in D_{osTrain} \\ 0, & \text{inače} \end{cases} \quad (3.1)$$

Primijetimo da će model h napamet naučiti sve prevare (jer postoje isti primjeri i u testnom i u trening skupu), dok će sve primjere koje dotad nije naučio klasificirati

kao valjane transakcije (time će ispravno klasificirati i sve prevare i sve valjane transakcije). Upravo zbog primjera kojeg smo upravo ilustrirali prvo dijelimo skup D na D_{train} i D_{test} , a tek onda provodimo tehnike ponovnog uzorkovanja. Napomenimo da tehnike ponovnog uzorkovanja provodimo samo na skupu za učenje (D_{train}) upravo da bi primjeri na kojima testiramo bili dotad neviđeni.

2. **k-struka unakrsna validacija** - Prilikom podjele skupa podataka kao tehniku probira koristimo k-struku unakrsnu validaciju (u našem primjeru smo uzeli $k = 5$). Tu tehniku koristimo kako bismo dobili nepristraniju procjenu greške nego što bi dobili najjednostavnijom podjelom na skupove za učenje i testiranje. Opišimo ukratko algoritam da bismo lakše objasnili zašto 'standardna' 5-struka validacija nije dovoljna u slučajevima kada radimo s neuravnoteženim skupovima podataka (pogotovo kada je omjer neuravnoteženosti velik kao sa skupom podataka s kartičnim prevarama).

Algorithm 1 k-fold cross validation

```

1: procedure K-FOLDERROR( $D$ )
2:    $D \leftarrow shuffle(D)$                                 ▶ promiješamo sve podatke u skupu podataka
3:    $(D_1, D_2, \dots, D_k) \leftarrow D$                     ▶ podijelimo na  $k$  skupova
4:   for  $i \leftarrow 1$  to  $k$  do
5:      $h_i \leftarrow trainModel(D \setminus D_i)$           ▶ treniramo model na svim osim  $i$ -tog skupa
6:      $Err_i \leftarrow calculateError(h_i, D_i)$            ▶ procjena greške modela  $h_i$  na skupu  $D_i$ 
7:   end for
8:    $\overline{ERR}_{k-fold} \leftarrow \frac{1}{k} \sum_{j=1}^k Err_j$        ▶ izračunaj prosječnu grešku za svih  $k$  modela
9:   return  $\overline{ERR}_{k-fold}$ 
10: end procedure
  
```

Takav postupak unakrsne validacije može dovesti do problema kada ga koristimo na neuravnoteženim skupovima podataka. Odnosno, može se dogoditi da ne bude željeni omjer prevara i valjanih transakcija u skupovima za učenje i testiranje, što bi moglo utjecati na performanse klasifikatora. Primjera radi, izveli smo eksperiment za 5-struku unakrsnu validaciju na skupu podataka za prepoznavanje kartičnih prevara (D_{train}) i dobili sljedeće rezultate:

k-fold	Trening skup		Test skup	
	Broj valjanih	Broj prevara	Broj valjanih	Broj prevara
1	181932	344	45496	73
2	181935	341	45493	76
3	181961	315	45467	102
4	181935	341	45493	76
5	181949	327	45479	90

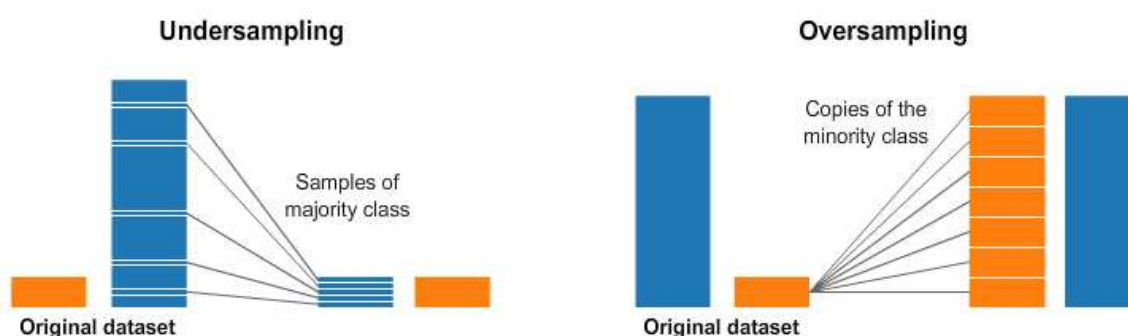
Zato umjesto klasične k -struke unakrsne validacije kod neuravnoteženih skupova podataka koristimo slojevitu k -struku unakrsnu validaciju (eng. stratified k -fold cross-validation). Slojevitom k -strukom unakrsnom validacijom dijelimo skup nasumično, ali tako da zadržimo jednaku distribuciju klasa u svakom podskupu. Kada izvršimo slojevitu k -struku unakrsnu validaciju na D_{train} dobivamo sljedeće rezultate:

strat k-fold	Trening skup		Test skup	
	Broj valjanih	Broj prevara	Broj valjanih	Broj prevara
1	181941	335	45487	82
2	181951	325	45477	92
3	181938	338	45490	79
4	181940	336	45488	81
5	181942	334	45486	83

Iz tablica vidimo da je distribucija klasa puno balansiraniija kada umjesto klasične k -struke unakrsne validacije koristimo slojevitu k -struku unakrsnu validaciju.

Metode ponovnog uzorkovanja

Kada smo ranije u ovom poglavlju govorili o učenju na neuravnoteženim skupovima podataka spomenuli smo 3 različite vrste pristupa koje pokušavaju ukloniti pristranost modela prema negativnoj klasi. To su bili *pristupi na razini podataka*, *pristupi na razini algoritama* i *troškovno-osjetljivi pristupi*. U ostatku ovoga poglavlja ćemo detaljno opisati metode koje se koriste kod ponovnog uzorkovanja. Kao što smo već i spomenuli metode ponovnog uzorkovanja se dijele na metode poduzorkovanja (eng. undersampling) i na metode preuzorkovanja (eng. oversampling). Na slici 3.1 (preuzeta s [1]) najlakše možemo primijetiti kakav utjecaj na skup podataka imaju metode ponovnog uzorkovanja.



Slika 3.1: Metode ponovnog uzorkovanja

Metode poduzorkovanja koristimo kada iz skupa podataka želimo ukloniti primjere negativne klase, a metodu preuzorkovanja kada u skup podataka želimo dodati primjere pozitivne klase. Obje vrste ponovnog uzorkovanja se koriste kako bismo balansirali skup podataka, odnosno uklonili pristranost modela prema negativnoj klasi. Bilo koja metoda ponovnog uzorkovanja se provodi u predobradi podataka, odnosno prije nego što podatke 'ubacimo' u klasifikator. Upravo je nezavisnost metoda ponovnog uzorkovanja od klasifikatora najveća prednost pristupa na razini podataka. Najveća mana ovog pristupa je što ne znamo točno kakav je omjer klasa najbolji za pojedini skup podataka. Preciznije, unaprijed nam nije poznato je li prilikom provođenja metoda ponovnog uzorkovanja najbolji omjer klasa 50 : 50 ili je ipak bolje da su podaci blago neuravnoteženi. U sljedećim odjeljcima ćemo detaljno opisati algoritme ponovnog uzorkovanja te navesti koje su im prednosti, odnosno mane.

3.2 Poduzorkovanje

Slučajno poduzorkovanje

Prva, a vjerojatno i najjednostavnija metoda poduzorkovanja koju spominjemo je *slučajno poduzorkovanje*. Slučajno poduzorkovanje je neheuristička metoda (ne koristi nikakav algoritam za rješavanje problema) kojoj je cilj balansirati distribuciju klasa tako što slučajno odabere primjere negativne klase i ukloni ih iz skupa za učenje (u slučaju kada imamo, kao i u skupu za prepoznavanje kartičnih prevara, samo 2 klase). U općenitom slučaju metoda može ukloniti samo primjere većinske klase ili ukloniti primjere iz svih klasa osim manjinske klase. Najveća mana ove metode je što potencijalno odbacuje puno korisnih podataka, tj. odbacuje podatke koji bi pomogli modelu da jasnije nauči razlike valjanih transakcija i prevara. Konkretno, kada bismo željeli balansirati skup podataka za prepoznavanje kartičnih prevara tako da omjer bude 1 : 1, odbacili bismo preko 220000 valjanih transakcija iz skupa za učenje (što je preko 80% cijelog skupa podataka). Najveća prednost ove tehnike je brzina, obzirom da za uklanjanje slučajnih primjera nije potreban nikakav algoritam. Nadalje, uklanjanje tako velikog broja valjanih transakcija iz skupa podataka drastično ubrzava i sam proces učenja (nakon slučajnog poduzorkovanja za učenje koristi oko 900 primjera, dok bi bez predobrade učio na više od 220000 primjera).

Metoda stisnutih najbližih susjeda

Metoda stisnutih najbližih susjeda (eng. condensed nearest neighbors - CNN) je metoda koju je prvi put u istraživačkom radu spomenuo P. E. Hart [30]. Hart je svoj rad objavio u svibnju 1968. godine kada su računala još bila jako spora (prvi mikroprocesor je proizveden tek 1970.) i cilj je bio ubrzati svaki algoritam koliko god je moguće (ako je skup podataka

na ulazu algoritma velik, tada će i izračun biti dug). Stoga je Hart predložio metodu CNN s ciljem da od originalnog neuravnoteženog skupa za učenje D stvori novi skup E , takav da $E \subseteq D$ i vrijedi da je skup E konzistentan sa skupom D . Pritom za skup E kažemo da je konzistentan sa skupom D ako skup E ispravno klasificira **svaki** primjer iz skupa D koristeći metodu 1-najbližeg susjeda. Cilj metode CNN je ustvari da se iz skupa podataka za učenje uklone svi suvišni primjeri. Suvišni primjeri su svi oni koji ne daju nikakve dodatne informacije prilikom učenja, odnosno oni koji su najudaljeniji od granice odluke.

Napomenimo da metoda CNN ne mora nužno naći najmanji konzistentan skup E za D , odnosno moguće je da postoji skup \bar{E} takav da je $\bar{E} \subseteq E \subseteq D$ i vrijedi da je \bar{E} konzistentan s D . Sljedeći dvodijelni algoritam prikazuje kako se provodi metoda CNN, odnosno kako dobiti skup E od skupa D koristeći poduzorkovanje.

Algorithm 2 Inicijalizacija

```

1: procedure INITIALIZECNN( $D$ )
2:    $S \leftarrow []$ 
3:    $G \leftarrow []$ 
4:    $x \leftarrow getRandomExample(D)$ 
5:    $S \leftarrow S \cup \{x\}$ 
6:    $D \leftarrow D \setminus \{x\}$ 
7:   while  $D \neq \emptyset$  do
8:      $d \leftarrow Front(D)$ 
9:      $nn \leftarrow NearestNeighbor(1, S, d)$ 
10:    if  $class(nn) = class(d)$  then
11:       $G \leftarrow G \cup \{d\}$ 
12:    else
13:       $S \leftarrow S \cup \{d\}$ 
14:    end if
15:     $D \leftarrow D \setminus \{d\}$ 
16:  end while
17:  return  $S, G$ 
18: end procedure

```

1. uzmemo prvi element (d) iz skupa D
2. nađemo mu najbližeg susjeda (nn) iz skupa STORE koristeći 1-NN metodu (nalazi samo 1 najbližeg susjeda, a znamo da u STORE već imamo bar 1 element)
3. Ako su oznake klasa primjera d i primjera nn jednake dodajemo d u skup GRABBAG, odnosno znamo da tada d možemo ispravno klasificirati. Ako oznake klasa primjera

d i primjera nn nisu jednake dodajemo d u STORE jer ga nismo mogli ispravno klasificirati koristeći ostale primjere iz STORE, što znači da d donosi dodatne informacije.

4. izbacimo d iz skupa D

U drugom dijelu algoritma (algoritam 3) pozivamo funkciju koju smo definirali u prvom dijelu da bismo inicijalizirali skupove STORE i GRABBAG. Sada želimo provjeriti može li se svaki element iz GRABBAG skupa ispravno klasificirati. Da bismo to provjerili definiramo sljedeće varijable:

1. `newAdditions` - varijabla koja poprima vrijednosti TRUE ili FALSE, te označava jesmo li ijedan primjer prebacili iz skupa GRABBAG u skup STORE (ako jesmo postavljamo varijablu na TRUE)
2. `count` - brojač kojim brojimo koliko smo elemenata skupa GRABBAG prošli
3. `gLength` - broj elemenata u skupu GRABBAG

Sada s početka skupa GRABBAG mičemo jedan po jedan element (označimo ga s g). Taj element ćemo ili vratiti nazad u skup GRABBAG ili ga prebaciti u skup STORE (tada ćemo `newAdditions` postaviti na TRUE). Ako je nakon toga skup GRABBAG ostao prazan (to se može dogoditi samo u slučaju da smo prebacili element g u skup STORE), onda smo gotovi. U tom slučaju (nazovimo taj scenarij slučajem 1) primijetimo da ustvari vrijedi $STORE = D$, odnosno poduzorkovanjem nismo izbacili niti jedan element. U suprotnom provjeravamo je li g prebačen u STORE, a da pritom GRABBAG nije ostao prazan. Ako jest (nazovimo taj scenarij slučajem 2), tada resetiramo varijable `newAdditions`, `count` i `gLength` na inicijalne vrijednosti (zato što će sada za prolaz kroz skup GRABBAG trebati manje koraka). Ako ne vrijede niti slučaj 1, niti slučaj 2, tada znamo da element g nismo prebacili u skup STORE, nego smo ga vratili na kraj skupa GRABBAG. Tada vrijedi jedno od sljedećeg:

1. ili smo prošli jednom kroz cijeli skup GRABBAG bez da smo prebacili ijedan element u STORE. Tada vraćamo skup STORE (jer nijedan primjer iz skupa GRABBAG više ne donosi nove informacije, tj. znamo da, kada bismo opet prošli kroz skup GRABBAG, opet ne bi bilo promjena)
2. ili smo samo napravili običan korak, stavivši element g s početka na začelje skupa GRABBAG

Iz algoritma za metodu stisnutih susjeda vidimo da ova metoda ima jednu značajnu prednost nad slučajnim poduzorkovanjem. Ta prednost je što ovom metodom iz skupa za učenje mičemo primjere koji ne donose nikakve dodatne informacije za učenje, dok

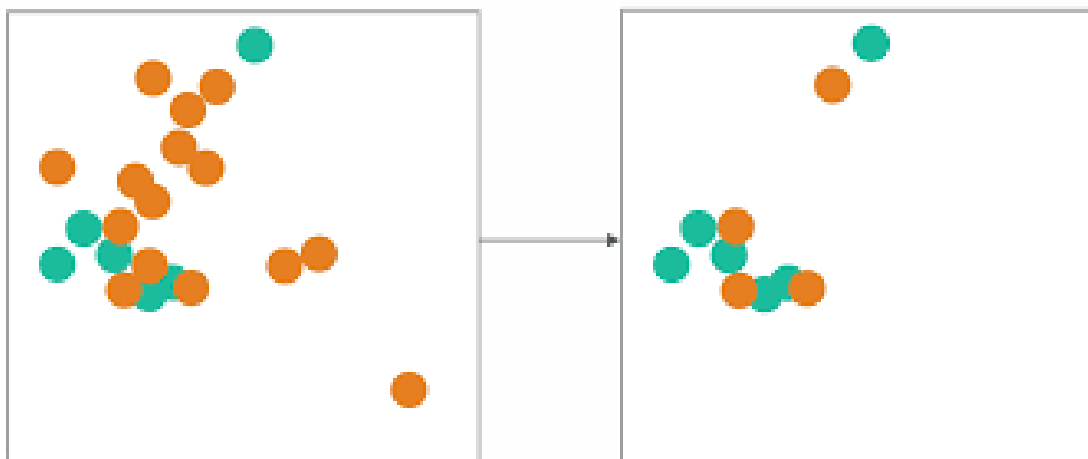
Algorithm 3 CNN

```

1: procedure CNN( $D$ )
2:    $S, G \leftarrow \text{InitializeCNN}(D)$ 
3:    $\text{newAdditions} \leftarrow \text{FALSE}$ 
4:    $\text{count} \leftarrow 0$ 
5:    $gLength \leftarrow \text{len}(G)$ 
6:   while TRUE do
7:      $g \leftarrow \text{Front}(G)$ 
8:      $G \leftarrow G \setminus \{g\}$ 
9:      $\text{count} \leftarrow \text{count} + 1$ 
10:     $nn \leftarrow \text{NearestNeighbor}(1, S, g)$ 
11:    if  $\text{class}(nn) = \text{class}(g)$  then
12:       $G \leftarrow G \cup \{g\}$ 
13:    else
14:       $S \leftarrow S \cup \{g\}$ 
15:       $\text{newAdditions} \leftarrow \text{TRUE}$ 
16:    end if
17:    if  $G = \emptyset$  then return  $S$ 
18:    end if
19:    if  $\text{newAdditions} = \text{TRUE}$  then
20:       $\text{newAdditions} \leftarrow \text{FALSE}$ 
21:       $\text{count} \leftarrow 0$ 
22:       $gLength \leftarrow \text{len}(G)$ 
23:    end if
24:    if  $\text{count} = gLength$  then return  $S$ 
25:    end if
26:  end while
27:  return  $S$ 
28: end procedure

```

slučajnim poduzorkovanjem mičemo slučajno odabrane primjere (koji mogu biti značajni za učenje). S druge strane, mana ovog pristupa je sam postupak uklanjanja primjera koje je za slučajno poduzorkovanje jako brzo, a ovdje se ipak treba izvršiti gore navedeni algoritam koji je znatno sporiji. Djelovanje metode najbližih susjeda ilustrirano je na slici 3.2 preuzetoj iz [44].



Slika 3.2: CNN

Tomekove veze

Ivan Tomek je u svome članku 1976. iznio 2 modifikacije CNN algoritma [54]. Druga od tih modifikacija je i danas jako popularna, te je njemu u čast nazvana *Tomekovim vezama* (eng. *Tomek links*). Tomek je primijetio da CNN u početku dosta nasumično uzima primjere koje stavlja u skup E , dok pretkraj ima tendenciju stavljati u E primjere koji su bliže granici koja odjeljuje klase. To dovodi do sljedećih problema:

1. E sadrži dosta 'unutarnjih' primjera (onih daleko od granice). Ti primjeri će ionako biti bliži primjerima iste klase (jer nisu na granici, a koristi se 1-NN) pa mogu biti eliminirani bez gubitka informacija. Bolje rečeno, skup E je veći nego što je potrebno.
2. E sadrži primjere koji oblikuju granicu za E , ali nisu oblikovali granicu za D . To je problem jer znači da se granica pomaknula što bi moglo prouzročiti pogrešnu klasifikaciju novih primjera.

Tomek je zato u svojoj modifikaciji povezivao primjere na sljedeći način. Neka su nam dana 2 primjera e_0 i e_1 takvi da e_0 pripada klasi 0, a e_1 klasi 1, te neka je $dist(e_0, e_1)$ udalje-

nost između primjera e_0 i e_1 . Tada par (e_0, e_1) tvore Tomekovu vezu ako ne postoji primjer e_2 takav da vrijedi: $dist(e_0, e_2) < dist(e_0, e_1)$ ili $dist(e_1, e_2) < dist(e_0, e_1)$, odnosno primjeru e_0 je najbliži susjed e_1 i obratno. Primijetimo da ako primjeri e_0 i e_1 tvore Tomekovu vezu vrijedi točno **jedno** od sljedećeg:

1. e_0 i e_1 su primjeri koji se nalaze na granici koja odjeljuje klase (jer po definiciji Tomekove veze svaki pripada drugoj klasi)
2. jedan od e_0, e_1 je 'šum' u podacima (ovaj slučaj se obično događa kada jedan od primjera pripada jednoj klasi, a posjeduje karakteristike druge klase).

Algorithm 4 Tomek Links

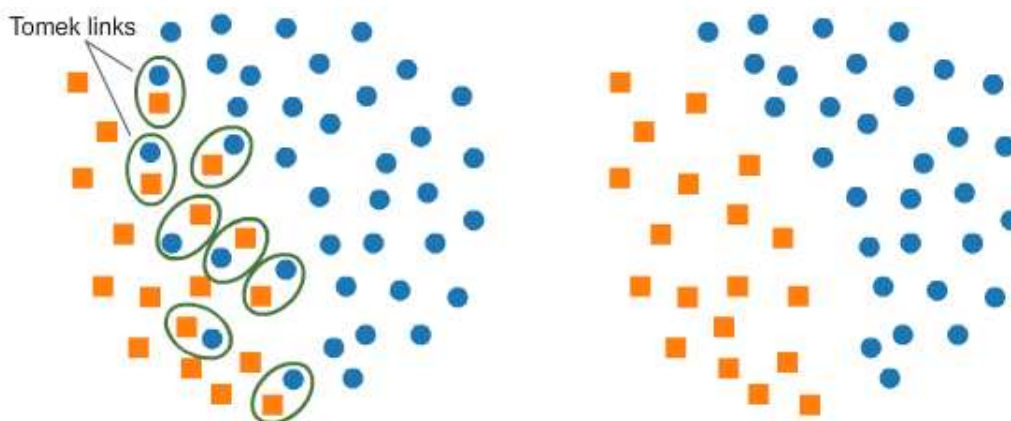
```

1: procedure TOMKLINKS( $D$ )
2:    $TL \leftarrow []$ 
3:   for each  $pair(e_i, e_j) \in D^2$  do
4:      $check = \text{TRUE}$ 
5:     if  $class(e_i) = class(e_j)$  then
6:       continue
7:     else
8:       for each  $e_k \in D$  do
9:         if  $e_k = e_i \vee e_k = e_j$  then
10:          continue
11:        else
12:          if  $dist(e_i, e_k) < dist(e_i, e_j) \vee dist(e_j, e_k) < dist(e_i, e_j)$  then
13:             $check = \text{FALSE}$ 
14:            break
15:          end if
16:        end if
17:      end for
18:      if  $check = \text{TRUE}$  then
19:         $TL \leftarrow TL \cup \{pair(e_i, e_j)\}$ 
20:      end if
21:    end if
22:  end for
23:  return  $TL$ 
24: end procedure

```

Algoritam 4 prikazuje pseudokod za pronalaženje Tomekovih veza. Primijetimo da metoda Tomekovih veza, osim za uklanjanje primjera većinske klase kod neuravnoteženih

podataka, može biti korištena i za čišćenje podataka (u tom slučaju se odbacuju primjeri obiju klasa). Također važna razlika između CNN-a i Tomekovih veza je što CNN-om dobivamo primjere koje ostavljamo u skupu, dok metodom Tomekovih veza dobivamo primjere koje brišemo iz skupa. Slika 3.3 (preuzeta s [1]) prikazuje djelovanje Tomekovih veza.



Slika 3.3: Tomek Links

Metoda uređenih najbližih susjeda

Posljednji 'jednostavni' algoritam za poduzorkovanje koji spominjemo zove se *metodom uređenih najbližih susjeda* (eng. *edited nearest neighbors*). Tu metodu je prvi predložio je Dennis L. Wilson [58] u svome radu 1972. godine. Wilson je smatrao da se većina praktičnih problema može svesti na sljedeće:

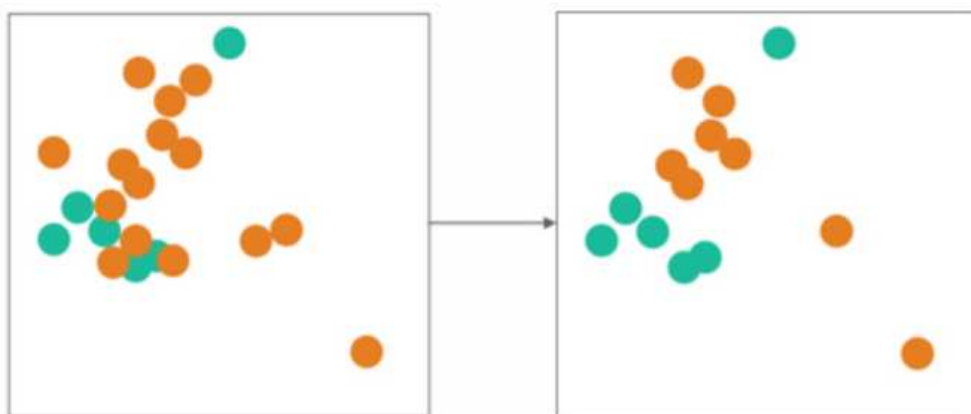
1. postoji primjer e koji želimo klasificirati
2. od ranije postoji skup već klasificiranih primjera $S = \{e_1, \dots, e_m\}$ koji pripadaju istom skupu podataka - njih treba koristiti da bi se donijela odluka za novi primjer e
3. nema nikakvih dodatnih informacija, tj. sve informacije koje koristimo za klasifikaciju primjera e dobivamo iz skupa otprije klasificiranih primjera S
4. moguće je 'nekako' (koristeći neku poznatu metriku) izmjeriti udaljenosti između svaka 2 primjera e_i i e_j

Na temelju toga osmislio je metodu uređenih najbližih susjeda koja prvo koristi 3-NN (3 najbliža susjeda) da bi pronašla pogrešno klasificirane primjere među ranije klasificiranim primjerima (tj. primjerima iz S). Ti pogrešno klasificirani primjeri se onda uklanjaju

iz skupa za učenje te se novi primjer e klasificira korištenjem 1-NN metode. Kod poduzorkovanja metodu ENN koristimo na sljedeći način:

1. za svaki primjer $d \in D$ izračunamo 3 najbliža susjeda
2. ako vrijedi da je d primjer negativne (većinske) klase ($d = (\vec{x}, 0)$) i da je d pogrešno klasificiran (korištenjem 3-NN metode iz prošlog koraka), onda izbacujemo d iz D
3. ako vrijedi da je d primjer pozitivne (manjinske) klase ($d = (\vec{x}, 1)$) i pogrešno je klasificiran (korištenjem 3-NN metode iz prošlog koraka, susjede označimo s d_1, d_2, d_3), onda iz D izbacujemo svakog od 3 susjeda koji pripada negativnoj (većinskoj klasi). Odnosno, ako vrijedi $d_k = (\vec{x}_k, 0)$ tada d_k izbacujemo iz D za $k \in \{1, 2, 3\}$

Kod algoritma za metodu uređenih najbližih susjeda korisno je primijetiti da će primjer pozitivne klase ($class(d) = 1$) biti pogrešno klasificiran ($predCls = 0$), ako barem 2 od 3 najbliža susjeda pripadaju negativnoj klasi. Wilson je u svome radu pokazao da kada primijenimo metodu uređenih susjeda prije nego što 1-NN metodom klasificiramo nove primjere dobivamo bolje rezultate, tako da je prirodno upitati se što ako ponovimo ENN metodu više puta [53], odnosno hoće li rezultati biti još bolji nakon što na skup D dvaput primijenimo ENN metodu? Ako hoće koliko puta bismo trebali ponavljati metodu bez da se rezultati pogoršaju? Odgovori na ta pitanja su nažalost i dalje nepoznati. Na slici 3.4 (preuzeta iz [44]), možemo vidjeti kako skup podataka izgleda prije i poslije ENN metode.



Slika 3.4: ENN

Algorithm 5 ENN

```

1: procedure ENN( $D$ )
2:   for each  $d \in D$  do
3:      $(d_1, d_2, d_3) \leftarrow \text{NearestNeighbors}(3, D, d)$ 
4:      $\text{predCls} \leftarrow \text{getMajority}(\text{class}(d_1), \text{class}(d_2), \text{class}(d_3))$ 
5:     if  $\text{class}(d) \neq \text{predCls}$  then
6:       if  $\text{class}(d) = 1$  then
7:         if  $\text{class}(d_1) = 1$  then
8:            $D \leftarrow D \setminus \{d_2, d_3\}$ 
9:         else
10:          if  $\text{class}(d_2) = 1$  then
11:             $D \leftarrow D \setminus \{d_1, d_3\}$ 
12:          else
13:            if  $\text{class}(d_3) = 1$  then
14:               $D \leftarrow D \setminus \{d_1, d_2\}$ 
15:            else
16:               $D \leftarrow D \setminus \{d_1, d_2, d_3\}$ 
17:            end if
18:          end if
19:        end if
20:      else
21:         $D \leftarrow D \setminus \{d\}$ 
22:      end if
23:    end if
24:  end for
25:  return  $D$ 
26: end procedure

```

3.3 Preuzorkovanje

Slučajno preuzorkovanje

Kao najjednostavniju metodu poduzorkovanja smo naveli slučajno poduzorkovanje. Isto tako kod preuzorkovanja postoji metoda zvana *slučajno preuzorkovanje* (eng. *random oversampling*). Slučajno preuzorkovanje također je (kao i slučajno poduzorkovanje) ne-heuristička metoda kojoj je cilj balansirati distribuciju klasa tako što slučajno odabere primjere pozitivne klase te ih replicira (moguće je jedan primjer replicirati više puta), odnosno nadoda kopije slučajno odabranih pozitivnih primjera u skup za učenje. Prednost ove metode nad slučajnim poduzorkovanjem je što ne gubimo nikakve informacije (jer ne

izbacujemo podatke iz skupa za učenje, nego samo dodajemo kopije). Da bismo na skupu podataka za prepoznavanje kartičnih prevara dobili omjer 1 : 1 potrebno je replicirati pozitivne primjere više od 220000 puta. Znači, ako pretpostavimo da će se svaki pozitivni primjer slučajno izabrati (za replikaciju) jednako puta, dobili bismo da se svaki pozitivni primjer replicira 577 puta. Upravo to je najveća mana ove metode, odnosno ubacili smo u skup za učenje jako puno podataka koji ne donose nikakve dodatne informacije, što znači da će i proces učenja biti znatno sporiji (algoritam za slučajno preuzorkovanje je brz, jer ne koristi nikakvu heuristiku - samo je proces učenja spor).

SMOTE

Vjerojatno najpoznatija tehnika preuzorkovanja (čak i ponovnog uzorkovanja) kada se radi o neuravnoteženim skupovima podataka zovemo *tehnikom sintetskog kreiranja primjera manjinske klase* (eng. synthetic minority oversampling technique, ili skraćeno SMOTE). SMOTE je prvi put predložio Chalwa [13] u svome radu 2002. godine. Možemo primijetiti da je ova metoda razvijena puno kasnije od metoda koje smo spomenuli u odjeljku o poduzorkovanju. Razlog tome je vjerojatno veliki napredak u razvoju računala, odnosno računala su 2000-tih već bila uređaji koji se nalaze u većini kućanstava, dok su industrijska računala postala dovoljno jaka da u sekundama izvrše algoritma za koji su se 1970-tih izvršavali satima, a neki čak i danima. Tada se umjesto izbacivanja podataka iz skupa za učenje (jer je brzina bila prioritet), počelo s nadodavanjem novih (s ciljem da se ne izgube informacije već nadodaju nove). Chalwa je na ideju došao kada je 2000. godine radio na svom diplomskom radu u kojem je pokušao detektirati piksele koji sadrže kancerogene stanice. On je primijetio da je, trenirajući jednostavan klasifikator (obično stablo odluke), dobio preko 97% točnosti, dok bi za klasifikator koji svaku stanicu klasificira kao nekancerogenu dobio 97.68% točnosti. Uočio je da klasifikator postiže jako loše rezultate za pozitivnu klasu. Nakon što je upotrijebio metodu slučajnog preuzorkovanja primijetio je da model bolje klasificira primjere pozitivne klase, ali da dolazi do overfittinga. Zapitao se kako riješiti taj problem i tada se dosjetio SMOTE tehnike (koja se pokazala daleko superiornijom od slučajnog preuzorkovanja).

Umjesto jednostavnog repliciranja primjera, SMOTE tehnika koristi "sintetski kreirane" primjere. SMOTE funkcionira tako da se odaberu primjeri koji imaju slične karakteristike (blizu su u "polju značajki"), te se povuče linija koja spaja te primjere. Tada SMOTE stvara sintetski kreirani primjer negdje na toj liniji. Preciznije, prvo se slučajno odabere neki primjer manjinske klase. Onda se pomoću k -NN algoritma pronađe njegovih k najbližih susjeda (u originalnom radu se koristilo $k = 5$) koji pripadaju pozitivnoj klasi. Nakon što su ti susjedi pronađeni slučajno biramo jednog od njih te se na dužini između primjera i njegovog slučajno odabranog susjeda odabire točka u kojoj se kreira sintetski primjer. Napomenimo

da prethodni postotak opisuje SMOTE kojim za svaki primjer manjinske klase stvaramo 1 sintetsko kreirani primjer (odnosno koeficijent preuzorkovanja je 100%). Kada bismo htjeli preuzorkovati s koeficijentom od 200% trebali bismo za svaki primjer slučajno odabrati 2 susjeda te na dužinama između primjera i ta 2 susjeda kreirati 2 sintetska primjera (svaki na jednoj dužini). S druge strane, ako bismo htjeli preuzorkovati s koeficijentom 50% bismo odabrali samo pola primjera pozitivne klase za koje bismo sintetski kreirali nove primjere.

Opišimo sada preciznije kako se sintetski kreiraju novi primjeri:

1. odredimo 'razliku' između primjera i njegovog slučajno odabranog susjeda u "polju značajki"
2. razliku koju smo dobili pomnožimo s brojem $\theta \in [0, 1]$ te je dodamo na vektor od primjera do njegovog susjeda. Time ćemo dobiti neku točku koja se nalazi na dužini između primjera i njegovog susjeda, pri čemu vrijedi da će za $\theta = 0$ sintetski kreirani primjer ustvari biti kopija tog primjera, dok će za $\theta = 1$ sintetski kreirani primjer ustvari biti kopija njegovog susjeda.

Algorithm 6 Positive Examples

```

1: procedure GETPOSITIVEEXAMPLES( $D$ )
2:    $pos \leftarrow []$ 
3:   for each  $d \in D$  do
4:     if  $class(d) = 1$  then
5:        $pos \leftarrow pos \cup \{d\}$ 
6:     end if
7:   end for
8: end procedure

```

Algoritam 7 prikazuje prvi dio SMOTE tehnike. Prvo od svih primjera u skup podataka za učenje D izvučemo sve pozitivne primjere (spremamo ih u polje `pos`). Količinu primjera nad kojima treba provesti algoritam predstavlja varijabla `volPerc` koja je poznata jer je ulazni parametar. Ako je `volPerc` manji od 100% tada u polju `pos` sačuvamo `synNum` slučajnih primjera. Varijabla `vol` nam govori koliko puta od nekog primjera treba kreirati sintetski primjer (primijetimo da ako je `volPerc < 100` onda izbacimo neke pozitivne primjere, tj. od izbačenih primjera ne kreiramo sintetske primjere, ali postavljamo `volPerc` na 100 da bismo od preostalih primjera odabrali po jednog susjeda. Nakon toga u petlji prolazimo po svim (preostalim) pozitivnim primjerima te za svakog nalazimo k najbližih susjeda. U drugom koraku petlje pozivamo funkciju `createSyntheticSamples` definiranu algoritmom 8 kojom kreiramo sintetske primjere.

Algorithm 7 SMOTE

```

1: procedure SMOTE( $D, volPerc, k$ )
2:    $pos \leftarrow getPositiveExamples(D)$ 
3:    $synNum \leftarrow len(pos)$ 
4:   if  $volPerc < 100$  then
5:      $synNum \leftarrow \lfloor \frac{volPerc}{100} * synNum \rfloor$ 
6:      $volPerc \leftarrow 100$ 
7:      $pos \leftarrow keepRandom(pos, synNum)$ 
8:   end if
9:    $vol \leftarrow \lfloor \frac{volPerc}{100} \rfloor$ 
10:  for each  $p \in pos$  do
11:     $pNeighbors \leftarrow NearestNeighbors(k, pos, p)$ 
12:     $D \leftarrow D \cup \{createSyntheticSamples(vol, p, pNeighbors, pos)\}$ 
13:  end for
14: end procedure

```

Algorithm 8 Creating Synthetic Samples

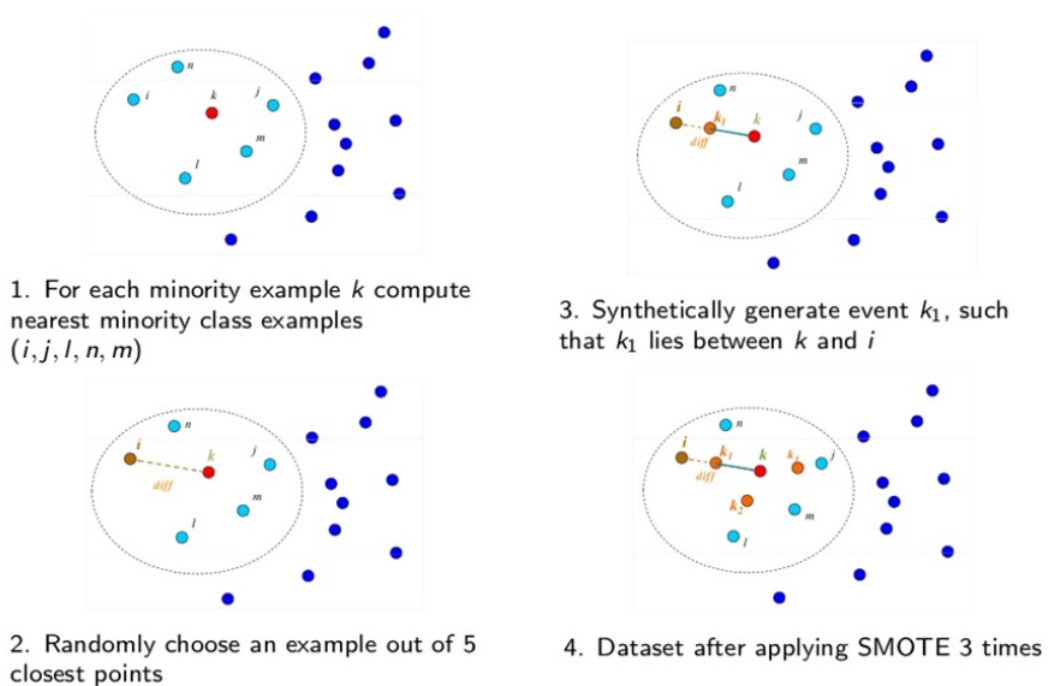
```

1: procedure CREATESYNTHETICSAMPLES( $vol, p, pNeighbors, pos$ )
2:    $syn \leftarrow []$ 
3:   for  $i = 1$  to  $vol$  do
4:      $neighbor \leftarrow pNeighbors[random.randint(1, len(pNeighbors))]$ 
5:      $pIndex \leftarrow index(p)$ 
6:      $neighborIndex \leftarrow index(neighbor)$ 
7:      $newSyn \leftarrow SyntheticExample()$ 
8:     for  $feature = 1$  to  $len(p) - 1$  do
9:        $diff \leftarrow pos[neighborIndex][feature] - pos[pIndex][feature]$ 
10:       $\theta \leftarrow random.uniform(0, 1)$ 
11:       $newSyn[feature] \leftarrow pos[pIndex][feature] + \theta \cdot diff$ 
12:    end for
13:     $newSyn[len(p)] \leftarrow 1$ 
14:     $syn \leftarrow syn \cup \{newSyn\}$ 
15:  end for
16:  return  $syn$ 
17: end procedure

```

Opišimo sad korak po korak kako se kreiraju sintetski primjeri. Polje syn na početku je prazno i postepeno ga punimo kreiranim sintetskim primjerima. Sljedeći korak je za svaki primjer kreirati vol sintetskih primjera (to možemo petljom od 1 do vol). Zatim slučajno odaberemo jednog od k susjeda. Varijabla $newSyn$ predstavlja jedan primjer (sjetimo se da je svaki primjer ustvari definiran kao kolekcija značajki, odnosno za primjer e vrijedi $e = (\vec{x}, y)$). U petlji prolazimo kroz sve značajke (osim posljednje) promatranog primjera p te računamo 'razliku' (po značajkama) između njega i susjeda. Također za θ odabremo slučajno generirani broj između 0 i 1 (sjetimo se da je to 'pomak' na dužini između promatranog primjera p i njegovog susjeda $neighbor$). U zadnjem koraku unutarnje petlje (po značajkama) kreiramo određenu značajku sintetskog primjera. Nakon što smo odredili sve značajke sintetskog primjera posebno mu postavimo posljednju značajku na 1 (ta značajka je ustvari oznaka klase obzirom da je primjer oblika (\vec{x}, y)).

Slika 3.5 (preuzeta s [45]) vrlo dobro prikazuje SMOTE tehniku korak po korak.



Slika 3.5: SMOTE

Borderline-SMOTE

SMOTE metoda za preuzorkovanje je u trenu postala vrlo popularna, stoga danas postoje mnoge varijante klasičnog algoritma opisanog u prošlom odlomku. Jedna od prvih i najpopularnijih je tehnika koja odabire one primjere koji su modelu teški za klasificirati (vjerojatnost pogrešne klasifikacije je velika) te samo njih razmatramo prilikom preuzorkovanja. Ideja za takvom metodom potječe iz činjenice da je primjere koji su blizu granice odluke najteže naučiti, te bi model postizao bolje performanse kada bi mu to uspjelo. Kao što iz naziva ove tehnike možemo zaključiti, preuzorkovanje se vrši slično kao i kod SMOTE-a, odnosno za primjer pozitivne klase se nalazi njegovih k najbližih susjeda, te se na dužinama između primjera i susjeda kreiraju sintetski primjeri. Razlika je što se u Borderline-SMOTE-u [29] preuzorkuju samo granični primjeri pozitivne klase.

Algorithm 9 Borderline SMOTE

```

1: procedure BORDERLINE SMOTE( $D, volPerc, m, k$ )
2:   ( $pos, DANGER$ )  $\leftarrow$  ( $getPositiveExamples(D), []$ )
3:   for each  $p \in pos$  do
4:      $mNeighbors \leftarrow NearestNeighbors(m, D, p)$ 
5:      $m \leftarrow len(mNeighbors)$ 
6:      $maj \leftarrow len(np.where(np.array(mNeighbors) == 0)[0])$ 
7:     if ( $m = maj$ )  $\vee$  ( $(0 \leq maj) \wedge (maj < \frac{m}{2})$ ) then
8:       continue
9:     else
10:      if ( $\frac{m}{2} \leq maj$ )  $\wedge$  ( $maj \leq m$ ) then
11:         $DANGER \leftarrow DANGER \cup \{p\}$ 
12:      end if
13:    end if
14:  end for
15:  for each  $b \in DANGER$  do
16:     $bNeighbors \leftarrow NearestNeighbors(k, pos, b)$ 
17:     $D \leftarrow D \cup \{createSyntheticSamples(\lfloor \frac{volPerc}{100} \rfloor, b, bNeighbors, pos)\}$ 
18:  end for
19:  return  $D$ 
20: end procedure

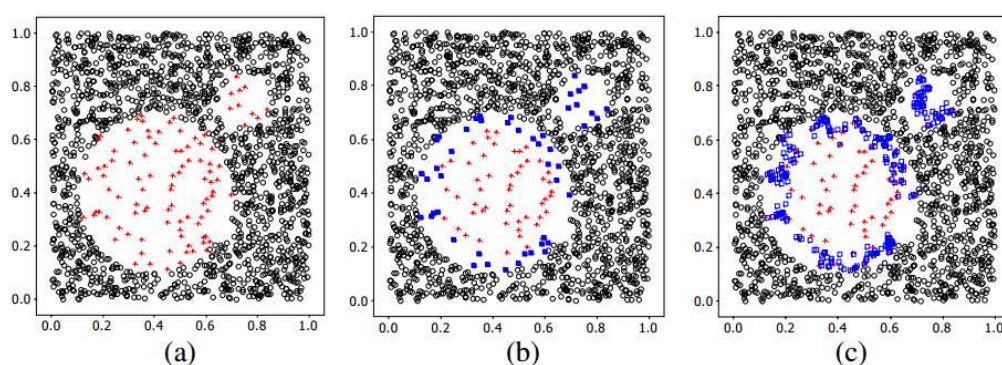
```

Opišimo malo detaljnije kako se provodi Borderline-SMOTE (detaljno je prikazano algoritmom 9).

1. Prvo, za svaki primjer pozitivne klase (označimo ga s p) nalazimo m najbližih susjeda (ti susjedi mogu pripadati i pozitivnoj i negativnoj klasi). Označimo s maj broj

susjeda koji pripadaju većinskoj klasi. Za maj vrijedi $0 \leq maj \leq m$.

2. Ako vrijedi $maj = m$ tada znamo da svih m najbližih susjeda od p pripada negativnoj klasi, pa za p pretpostavljamo da je šum u podacima (jer je pozitivan, a svi najbliži susjedi su mu negativni). Takav p ne razmatramo za preuzorkovanje.
3. Ako vrijedi $\frac{m}{2} \leq maj \leq m$ znamo da barem pola najbližih susjeda primjera p pripada negativnoj klasi, što znači da bi p vjerojatno bio pogrešno klasificiran. Preciznije, zamislimo da imamo već istreniran model te da trebamo predvidjeti oznaku klase primjera p , koji je dosad neviđen, te uzmimo da za klasifikaciju novih primjera model koristi metodu najbližih susjeda. Tada bi p vjerojatno bio pogrešno klasificiran kao negativan primjer jer mu više od pola susjeda pripada negativnoj klasi. Svaki takav p čuvamo u skupu DANGER i koristimo u ostatku algoritma (upravo ti primjeri su granični, odnosno oni koji su modelu najteži za klasificirati).
4. Ako vrijedi $0 \leq maj < \frac{m}{2}$, tada većina od m najbližih susjeda od p pripada pozitivnoj klasi. Takav p smatramo 'sigurnim' te ga ne razmatramo za preuzorkovanje.
5. U ovom koraku za svaki primjer $b \in \text{DANGER}$ računamo njegovih k najbližih susjeda iz **pozitivne klase**. Primijetimo da vrijedi $\emptyset \subseteq \text{DANGER} \subseteq \text{pos}$. Ako je $\text{DANGER} = \emptyset$, tada nemamo graničnih primjera i nećemo preuzorkovati nijedan primjer (ovaj slučaj je u praksi nevjerovatan zbog neuravnoteženosti podataka).
6. U posljednjem koraku od svakog primjera $b \in \text{DANGER}$ kreiramo l sintetskih primjera ($1 \leq l \leq k$) ovisno o količini preuzorkovanja, te onda na dužinama između b i njegovih l susjeda stvaramo sintetske primjere kao i u SMOTE metodi.



Slika 3.6: Borderline-SMOTE

Slika 3.6 (preuzeta s web-stranice [56]) prikazuje Borderline-SMOTE kroz 3 koraka. Slika označena s a) prikazuje skup podataka prije Borderline-SMOTE preuzorkovanja. Crvenom bojom su označeni primjeri pozitivne klase, a crnom primjeri negativne klase. Na slici b) vidimo da su neki primjeri pozitivne klase sada označeni plavom bojom. Upravo oni označavaju granične primjere koje ćemo preuzorkovati. Slika c) prikazuje kako izgleda skup podataka nakon preuzorkovanja. Vidimo da je sada blizu granica puno više primjera, što je i ideja Borderline-SMOTE metode.

ADASYN

U ovom odjeljku opisujemo ADASYN [31] metodu preuzorkovanja. ADASYN (Adaptive synthetic sampling) je, kao i Borderline-SMOTE, posebna varijanta klasične SMOTE tehnike. ADASYN je baziran na ideji da se sintetski kreiraju pozitivni primjeri u skladu sa svojom distribucijom. Bolje rečeno, više sintetskih primjera je kreirano za granične primjere (jer ih je teško naučiti), a manje za unutarnje primjere (jer se lakše uče). Možemo primijetiti da je ADASYN metoda dosta slična Borderline-SMOTE metodi. Obje metode se fokusiraju na preuzorkovanje teških primjera, dok ADASYN preuzorkuje i lakše primjere.

Opišimo sada ukratko ADASYN metodu preuzorkovanja (detaljan postupak moguće je vidjeti u algoritmu 10).

1. Prvo se računa već spomenuti omjer neravnoteže, odnosno $min : maj$ pri čemu je min broj primjera pozitivne klase, a maj broj primjera negativne klase.
2. Računamo koliko primjera pozitivne klase trebamo metodom kreirati (to je ulazni parametar $volPerc$). Ako je $volPerc = 100$ tada kreiramo točno onoliko primjera koliko je potrebno da nakon preuzorkovanja omjer neravnoteže bude 1 : 1.
3. Za svaki pozitivan primjer p_i računamo njegovih k najbližih susjeda, te omjer neravnoteže u 'susjedstvu' (označimo ga s r_i). Veći r_i znači da je u susjedstvu više primjera negativne klase, odnosno pozitivni primjeri u tim susjedstvima su 'teži za naučiti'. Nakon toga normaliziramo r_i tako da bi zbroj svih bio jednak 1 (to će nam biti potrebno za preuzorkovanje).
4. Za svako susjedstvo računamo koliko je sintetskih primjera potrebno kreirati (taj broj označavamo s g_i). Primijetimo da u susjedstvima u kojima je r_i veći kreiramo više sintetskih primjera.
5. U ovom koraku za svaki p_i kreiramo izračunati broj sintetskih primjera g_i slično kao i SMOTE metodom (na dužini između primjera p_i i slučajno odabranog susjeda koji također pripada **pozitivnoj** klasi kreiramo sintetski primjer).

Algorithm 10 ADASYN

```

1: procedure ADASYN( $D, volPerc, k$ )
2:    $pos \leftarrow getPositiveExamples(D)$ 
3:    $(posNum, negNum) \leftarrow (len(pos), len(D) - len(pos))$ 
4:    $IR \leftarrow \frac{posNum}{negNum}$ 
5:    $G \leftarrow (negNum - posNum) \cdot \lfloor \frac{volPerc}{100} \rfloor$ 
6:    $(r, i) \leftarrow ([], 0)$ 
7:   for each  $p_i \in pos$  do
8:      $p_i.Neighbors \leftarrow NearestNeighbors(k, D, p_i)$ 
9:      $maj \leftarrow len(np.where(np.array(mNeighbors) == 0)[0])$ 
10:     $r[i + +] \leftarrow \frac{maj}{k}$ 
11:  end for
12:   $(\hat{r}, g) \leftarrow ([], [])$ 
13:   $sumR \leftarrow \sum_{m=0}^{posNum-1} r[m]$ 
14:  for  $j = 0$  to  $posNum - 1$  do
15:     $\hat{r}[j] \leftarrow \frac{r[j]}{sumR}$ 
16:     $g[j] \leftarrow round(\hat{r}[j] \cdot G)$ 
17:     $p_j \leftarrow pos[j]$ 
18:    for  $l = 0$  to  $g[j] - 1$  do
19:      while  $class(neighbor) \neq 1$  do       $\triangleright$  Dok susjed ne bude iz pozitivne klase
20:         $neighbor \leftarrow p_j.Neighbors[random.randint(0, k)]$ 
21:      end while
22:       $(p_jIndex, neighborIndex) \leftarrow (index(p_j), index(neighbor))$ 
23:       $newSyn \leftarrow SyntheticExample()$ 
24:      for  $feature = 1$  to  $len(p_j) - 1$  do
25:         $diff \leftarrow pos[neighborIndex][feature] - pos[p_jIndex][feature]$ 
26:         $\theta \leftarrow random.uniform(0, 1)$ 
27:         $newSyn[feature] \leftarrow pos[p_jIndex][feature] + \theta \cdot diff$ 
28:      end for
29:       $newSyn[len(p_j)] \leftarrow 1$ 
30:       $D \leftarrow D \cup \{newSyn\}$ 
31:    end for
32:  end for
33:  return  $D$ 
34: end procedure

```

Najveća prednost ADASYN metode je to što preuzorkuje više sintetskih primjera u susjedstvima primjera koje je 'teško naučiti' (zbog \hat{r} koeficijenata). No ADASYN metoda ima i sljedeće 2 mane:

1. Rijetkost pozitivnih primjera - primjerice, ako neki pozitivan primjer p u susjedstvu nema niti jedan drugi pozitivan primjer. Tada će sintetski kreirani primjeri ustvari biti kopije primjera p .
2. Pogoršana preciznost - ADASYN sintetski generira više primjera u susjedstvima gdje je više negativnih primjera. Tada sintetski generirani pozitivni primjeri mogu biti vrlo slični primjerima negativne klase, što može prouzročiti veći broj *false negativesa*.

3.4 Kombinacije metoda ponovnog uzorkovanja

Kombinacija slučajnog preuzorkovanja i slučajnog poduzorkovanja

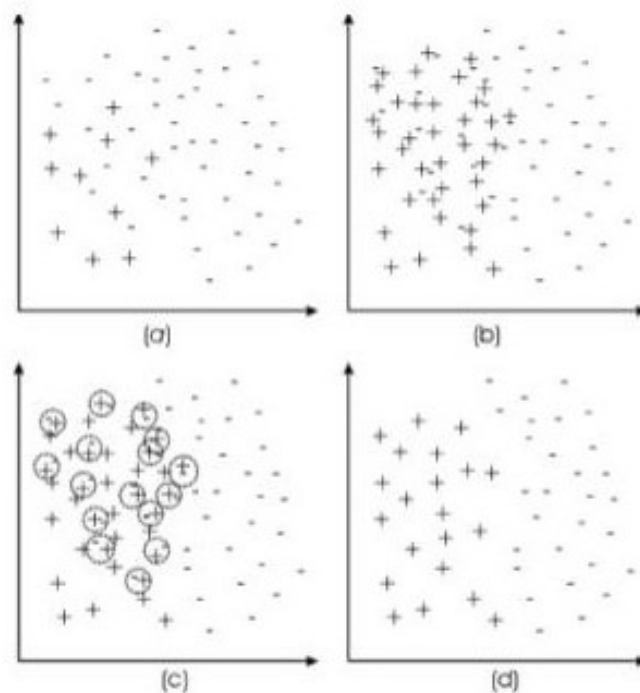
Prva metoda koju koristimo u ovom odjeljku je upravo ona najjednostavnija, tj. kombinacija slučajnog poduzorkovanja i slučajnog preuzorkovanja (obje metode smo već ranije opisali). Prisjetimo se da slučajno preuzorkovanje replicira slučajno odabrane primjere pozitivne klase, dok slučajno poduzorkovanje uklanja slučajno odabrane primjere negativne klase. Primijetimo da svaka od tih metoda djeluje na drugu klasu pa je svejedno kojim ih redom primjenjujemo. U našem slučaju ćemo prvo primijeniti slučajno preuzorkovanje kojim ćemo broj prevara povećati tako da bude jednako desetini broja valjanih transakcija ($minNum = 0.1 * majNum$), te nakon toga odbaciti polovinu valjanih transakcija. Iako su obje metode dosta jednostavne, njihova kombinacija može dati bolje rezultate nego kad se svaka metoda primijeni samostalno.

Kombinacija SMOTE-a i slučajnog poduzorkovanja

Sljedeća metoda također kombinira jednu metodu preuzorkovanja i jednu poduzorkovanja. U ovom slučaju su to: SMOTE za preuzorkovanje i slučajno poduzorkovanje. SMOTE kao što smo i spomenuli ranije sintetski kreira nove pozitivne primjere (algoritmi 7, 8) na dužini između slučajno odabranih pozitivnih primjera te njihovih slučajno odabranih susjeda pozitivne klase te time unosi dodatne informacije o pozitivnoj klasi. Kombinacija SMOTE-a i slučajnog poduzorkovanja se čini prirodna jer ne želimo sintetski kreirati previše primjera (jer bi neki od tih novokreiranih primjera mogli biti sličniji primjerima negativne klase nego što su primjerima pozitivne klase), ali ne želimo niti izbaciti previše primjera iz negativne klase (jer bismo izgubili puno korisnih informacija).

Kombinacija SMOTE-a i Tomekovih veza

Da bismo shvatili ovu metodu prisjetimo se kako obje metode funkcioniraju. SMOTE (algoritmi 7, 8) koristimo da bismo sintetski kreirali nove primjere pozitivne klase na dužini između odabranih pozitivnih primjera te njihovih slučajno odabranih susjeda (koji također pripadaju pozitivnoj klasi). Iako tom metodom možemo samostalno balansirati distribuciju klasa, stvorit ćemo i nove probleme. U stvarnom svijetu pozitivna i negativna klasa najčešće ne mogu biti jednostavno odijeljene, tj. postoje primjeri negativne klase koji su najbliži susjedi nekih pozitivnih primjera (odnosno postoje negativni primjeri u susjedstvima pozitivnih primjera). SMOTE sintetski stvara pozitivne primjere koji bi isto tako mogli biti kreirani u susjedstvima negativnih primjera. Pošto nakon SMOTE-a možemo dobiti tako opisanu situaciju ('miješanje' klasa) koristimo metodu Tomekovih veza da bismo uklonili takve primjere. Sjetimo se da smo Tomekove veze (algoritam 4) naveli kao metodu poduzorkovanja, ali da se također može koristiti i za čišćenje podataka. Upravo ovdje ćemo Tomekove veze koristiti za čišćenje podataka (odnosno umjesto brisanja primjera negativne klase iz skupa za učenje, brisat ćemo i negativne i pozitivne primjere) jer je već provedena SMOTE metoda da bi po volji uravnotežila skup podataka.



Slika 3.7: Kombinacija SMOTE-a i Tomekovih veza

Slika 3.7 (preuzeta iz [5]) prikazuje korake u kojima se provodi algoritam koji povezuje SMOTE i Tomekove veze.

1. slika a) prikazuje originalni skup podataka, odnosno prije nego što smo proveli neku metodu ponovnog uzorkovanja
2. slika b) prikazuje skup podataka dobiven nakon SMOTE metode za preuzorkovanje (vidimo više pozitivnih primjera)
3. slika c) prikazuje Tomekove veze (zaokruženi su parovi pozitivnih i negativnih primjera)
4. slika d) prikazuje kako skup podataka izgleda nakon uklanjanja Tomekovih veza (i pozitivnih i negativnih primjera)

Kombinacija SMOTE-a i ENN-a

Motivacija za korištenje kombinacije metoda SMOTE i ENN je slična kao i za kombinaciju SMOTE-a i Tomekovih veza. Odnosno prvo iskoristimo SMOTE (algoritmi 7 i 8) za preuzorkovanje manjinske klase da dobijemo ravnotežu klasa po volji. Prisjetimo se sada kako metoda uređenih najbližih susjeda funkcionira. Prvo bismo za svaki primjer $d \in D$ (i pozitivne i negativne) našli njegova 3 najbliža susjeda. Onda bismo promatrali bi li taj primjer bio pogrešno klasificiran koristeći oznake klase njegovih susjeda te bismo izbacivali:

Algorithm 11 Modificirani ENN

```

1: procedure MODIFIEDENN( $D$ )
2:   for each  $d \in D$  do
3:      $(d_1, d_2, d_3) \leftarrow \text{NearestNeighbors}(3, D, d)$ 
4:      $\text{predCls} \leftarrow \text{getMajority}(\text{class}(d_1), \text{class}(d_2), \text{class}(d_3))$ 
5:     if  $\text{class}(d) \neq \text{predCls}$  then
6:        $D \leftarrow D \setminus \{d\}$ 
7:     end if
8:   end for
9:   return  $D$ 
10: end procedure

```

1. ako je d primjer negativne klase i pogrešno klasificiran izbacili bismo d iz skupa za učenje
2. ako je d primjer pozitivne klase i pogrešno klasificiran izbacili bismo sve njegove negativne susjede

Ovdje modificiramo ENN metodu tako da u slučaju da je d pozitivan i pogrešno klasificiran kada izbacujemo njega umjesto njegovih susjeda (kao i u slučaju da je d negativan). Metodu ENN smo tako modificirali (prikazano algoritmom 11) obzirom da smo korak prije SMOTE metodom već preuzorčili podatke tako da ENN koristimo samo za čišćenje podataka.

Uzorkovanje jednostranim izborom

Za razliku od prethodnih metoda, koje su za predobradu podataka koristile kombinaciju jedne metode uzorkovanja i jedne metode poduzorkovanja, *uzorkovanje jednostranim izborom* (eng. *one-sided selection*, kratica: OSS) koristi 2 metode poduzorkovanja. Do takve ideje se došlo jer kreiranje kopija pozitivnih primjera ustvari ne donosi nikakve nove informacije u skup podataka, već samo usporava treniranje modela, dok kreiranje sintetskih primjera može stvoriti primjere koji su po karakteristikama sličniji primjerima negativne klase nego primjerima pozitivne klase. Zato se prilikom uzorkovanja jednostranim izborom pažljivo uklanjaju primjeri negativne klase, dok zadržavamo sve primjere pozitivne klase (čak i ako su neki od njih šum u podacima) jer ih ima jako malo, pa sadrže prevažne informacije da bismo ih izgubili. Kako bismo lakše prepoznali koje negativne primjere trebamo izbrisati iz skupa podataka možemo podijeliti primjere u 4 skupine:

1. šum u podacima - u ovu skupinu spadaju svi primjeri negativne klase koji se nalaze među primjerima pozitivne klase, odnosno s 'krive strane' granice odluke.
2. granični primjeri - u ovu skupinu svrstavamo sve negativne primjere koji se nalaze u blizini granice koja razdvaja primjere pozitivne klase od primjera negativne klase. Ti primjeri se smatraju nepouzdanim jer kada im malo promijenimo karakteristike mogu 'prijеći' na drugu stranu granice (među pozitivne primjere).
3. suvišni primjeri - suvišnim primjerima negativne klase smatramo onima koji ne donose dodatne informacije u skup za učenje. Preciznije, to su oni negativni primjeri koje možemo ispravno klasificirati koristeći ostale negativne primjere.
4. 'sigurni' primjeri - to su oni primjeri koji ne pripadaju niti jednoj od prethodnih skupina, odnosno primjeri koji sigurno pripadaju negativnoj klasi te ih treba koristiti prilikom učenja.

Cilj uzorkovanja jednostranim izborom je ukloniti sve negativne primjere koji pripadaju jednoj od prve 3 grupe (šum, granični, suvišni). Stoga jednostrani izbor možemo smatrati kombinacijom dviju metoda poduzorkovanja - Tomekovih veza i metode stisnutih najbližih susjeda (CNN). Granične primjere i šum možemo pronaći korištenjem Tomekovih veza (sjetimo se da se Tomekova veza sastoji od para pozitivnog i negativnog primjera koji su jedan drugom najbliži susjed, detaljno opisano algoritmom 4). S druge strane suvišne

primjere možemo pronaći tako da za skup D pronađemo konzistentan podskup E takav da se pomoću primjera iz E može ispravno klasificirati svaki primjer iz D , a upravo to je metoda CNN (opisana algoritmima 2, 3). Postupak za metodu jednostranog izbora je sljedeći:

1. neka je D originalni skup podataka
2. neka je C skup podataka koje želimo sačuvati, na početku u C stavimo sve pozitivne primjere i jedan slučajno odabrani negativan primjer
3. sljedeće klasificiramo svaki primjer $d \in D$. Ako smo d ispravno klasificirali ne radimo ništa, no ako smo ga pogrešno klasificirali dodajemo ga u C (kao u CNN metodi)
4. naposljetku izbacujemo iz skupa za učenje sve negativne primjere koji sudjeluju u Tomekovim vezama te tako eliminiramo sve granične primjere ili šum (primijetimo da ovdje Tomekove veze upotrebljavamo kao metodu poduzorkovanja, a ne za čišćenje podataka)

Metoda čišćenja susjedstva

Najveća mana metode jednostranog izbora je osjetljivost CNN metode na šum. Budući da će šumoviti primjeri vjerojatnije biti pogrešno klasificirani (sjetimo se algoritma 3), puno će ih biti zadržano u skupu STORE koji kasnije koristimo za učenje. Takvi negativni primjeri mogu prouzročiti pogrešnu klasifikaciju na skupu za testiranje. Iako OSS metoda nakon CNN metode koristi i Tomekove veze (algoritam 4) za uklanjanje šumovitih i graničnih primjera, skup koji dobijemo neće biti idealan upravo zato što je CNN metoda ostavila puno graničnih primjera (to možemo dobro vidjeti na slici 3.2). Ideja kod *metode čišćenja susjedstva* (eng. *neighbor cleaning rule*, skraćeno: NCL) je vrlo slična metodi jednostranog izbora, odnosno skup C čuvamo za učenje dok ostatak $O = D \setminus C$ reduciramo. Razlika je što NCL metodu koristimo primarno za čišćenje podataka, a ne za redukciju. Dva su osnovna razlog za takav pristup:

1. performansa modela ne mora nužno ovisiti o veličini klasa, odnosno postoje 'male' klase koje će model lako klasificirati (ako se, po karakteristikama, značajno razlikuju od drugih klasa), ali i 'velike' klase koje je teško klasificirati (puno primjera može značiti da su neki od tih primjera vjerojatno sličniji manjinskoj klasi nego ostalim primjerima većinske klase). Zato se ovim postupkom trudimo usredotočiti na ostale faktore (osim distribucije klasa) koji mogu pridonijeti pogrešnoj klasifikaciji (primjerice šumoviti podaci)

2. teško je zadržati performanse klasifikatora ako maknemo puno podataka iz skupa za učenje. Iako brisanjem primjera negativne klase poboljšavamo performanse klasifikatora na pozitivnoj klasi, metoda bi trebala biti u mogućnosti zadržati dosta dobre performanse i na negativnoj klasi (ne želimo prevelik broj *false positives*)

Zato NCL metoda koristi ENN metodu da bi se identificirali šumoviti primjeri $A_1 \subseteq O$ (sjetimo se da $O = D \setminus C$). ENN metodu (algoritam 5) koristimo za čišćenje podataka (modificiranu verziju koju smo opisali u odlomku o kombinaciji SMOTE metode i ENN metode), odnosno uklanjamo sve $d \in O$ koji bi bio pogrešno klasificiran metodom 3 najbliža susjeda. Primijetimo da tom metodom nećemo ukloniti previše primjera što je upravo ono što smo htjeli da bismo očuvali performanse klasifikatora na većinskoj klasi. Uz to, čistimo i susjedstva primjera $c \in C$. Ako 3 najbliža susjeda (c_1, c_2, c_3) pogrešno klasificiraju primjer c te pripadaju skupu O , onda ih spremamo u skup A_2 . Da bismo izbjegli uklanjanje primjera manjinske klase primjere spremamo u A_2 samo ako za klasu K , takvu da vrijedi $class(c) = K$, vrijedi $|K| \geq 0.5 \cdot |C|$ (primijetimo da za manjinsku klasu M vrijedi $|M| < 0.5 \cdot |C|$). Na kraju sve primjere iz A_1 i A_2 izbacujemo iz skupa za učenje D .

Sažeto zapisano:

1. Podijelimo D na skup primjera koje ćemo koristiti za učenje C i ostatak primjera $O = D \setminus C$
2. Identificiramo skup $A_1 \subseteq O$ koji predstavlja sve šumovite primjere dobivene modificiranim ENN metodom
3. Za svaku klasu $K \subseteq O$ provjeravamo klasificira li primjer $z \in K$ pogrešno neki primjer $c \in C$ (z je jedan od 3 susjeda od c). Ako da, tada $A_2 = A_2 \cup \{z\}$
4. $C = D \setminus (A_1 \cup A_2)$

Kombinacija Tomekovih veza i slučajnog poduzorkovanja

Posljednja metoda koju spominjemo u ovom odjeljku je kombinacija Tomekovih veza i slučajnog poduzorkovanja. Sjetimo se da Tomekove veze (opisane algoritmom 4) detektiraju šumovite odnosno granične primjere. To jest, Tomekova veza je par pozitivnog i negativnog primjera koji su jedan drugom najbliži susjedi. Ovdje koristimo Tomekove veze kako bismo uklonili isključivo primjere negativne klase. Nakon što uklonimo takve primjere možemo upotrijebiti metodu slučajnog poduzorkovanja. Primijetimo da će metoda slučajnog poduzorkovanja u takvoj situaciji ukloniti manji broj negativnih primjera (jer su već uklonjeni Tomekovim vezama), što je prednost jer smo ipak uklonili neke primjere za koje znamo da nam ne bi pomogli pri učenju modela (znamo da slučajno poduzorkovanje može ukloniti i važne i nevažne primjere jer ne koristi nikakvu heuristiku).

Poglavlje 4

Klasifikacija na neuravnoteženim skupovima podataka

4.1 O klasifikaciji

U prošlom poglavlju smo opisali predobradu podataka na neuravnoteženim skupovima podataka (konkretno metode koje se temelje na ponovnom uzorkovanju podataka), dok se u ovom poglavlju posvećujemo različitim tipovima klasifikatora. Početkom prethodnog poglavlja (odjeljak 3.1) u podsekciji 'Notacija' spomenuli smo što točno prepoznavanje kartičnih prevara čini klasifikacijskim problemom, a ne regresijskim (ciljna varijabla je kategorijska). Prisjetimo se da za svaki primjer vrijedi $D_i = (\vec{x}_i, y_i)$, pri čemu je $\vec{x}_i = (x_{i_1}, \dots, x_{i_{29}})$, gdje x_{i_1} označava značajku 'Amount' (nakon skaliranja), x_{i_2} označava značajku $T1$, ..., a $x_{i_{29}}$ značajku $T28$, dok y_i označava klasu i -tog primjera. Cilj nam je naučiti funkciju $h(\vec{x})$ koja 'što bolje' aproksimira funkciju f za koju vrijedi $f(\vec{x}) = y$. Da bismo odredili funkciju h koristit ćemo različite modele za učenje, odnosno različite tipove klasifikatora. U ostatku ovog poglavlja opisat ćemo učenje te donošenje odluka kod klasifikatora koje ćemo koristiti u eksperimentalnom dijelu.

4.2 Logistička regresija

Logistička regresija je vjerojatno najpoznatiji linearni klasifikacijski model, odnosno plohe koje dijele jednu klasu od druge su linearne (npr. pravac u dvodimenzionalnom prostoru). Za logističku regresiju koristimo takozvanu logističku funkciju:

$$P(X) = \frac{e^{\beta_0 + \beta X}}{1 + e^{\beta_0 + \beta X}} \quad (4.1)$$

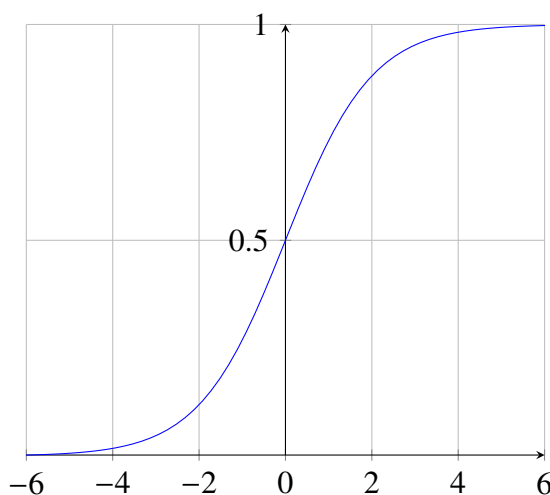
Takvu funkciju koristimo zato što vraća isključivo vrijednosti iz intervala $[0, 1]$. Napomenimo da β koristimo da bismo kraće zapisali vektor koeficijenata, odnosno vrijedi: $\beta = (\beta_1, \dots, \beta_m)$, pri čemu je vektor β jednako dug kao i vektor X . Konkretno, u našem slučaju (prepoznavanje kartičnih prevara) X se sastoji od 29 različitih atributa ('Amount', T1, ... T28) pa je $\beta = (\beta_1, \dots, \beta_{29})$. Za model koristimo metodu *maksimalne vjerodostojnosti* (eng *maximum likelihood*). Da bismo definirali metodu maksimalne vjerodostojnosti koristimo Bayesov model učenja. Prema Bayesovom pravilu za dosad dane podatke d , svaka hipoteza h_i ima **a posteriori** vjerojatnost:

$$P(h_i | d) = \frac{P(d | h_i) \cdot P(h_i)}{P(d)} = \alpha \cdot P(d | h_i) \cdot P(h_i) \quad (4.2)$$

Objasnimo oznake u formuli (4.2):

1. $P(h_i)$ - **a priori** vjerojatnost hipoteze h_i
2. $P(h_i | d)$ - izglednost hipoteze h_i , ako su nam dani podaci d
3. $P(d | h_i)$ - izglednost podataka d , ako nam je dana hipoteza h_i

Učenje maksimalne vjerodostojnosti se svodi na traženje hipoteze h_{ML} (ML = maximum likelihood) koja maksimizira $P(d | h_i)$, odnosno: $P(d | h_{ML}) = \max_{h_i \in H} P(d | h_i)$, pri čemu je H prostor svih hipoteza.



Slika 4.1: Krivulja logističke regresije

Nastavimo sada s logističkom regresijom. Graf logističke regresije (prikazan na slici 4.1) uvijek će poprimati oblik iskrivljenog slova 'S'. To je važno jer će prilikom predikcije

vjerojatnost da neki primjer pripada određenoj klasi uvijek biti između 0 i 1, ali nikad iznad 1 ili ispod 0 (vjerojatnost ne može biti manja od 0 ili veća od 1). Sada formulu (4.1) možemo zapisati kao:

$$\begin{aligned} P(X) &= \frac{e^{\beta_0 + \beta X}}{1 + e^{\beta_0 + \beta X}} \\ P(X) \cdot (1 + e^{\beta_0 + \beta X}) &= e^{\beta_0 + \beta X} \\ P(X) + P(X) \cdot e^{\beta_0 + \beta X} &= e^{\beta_0 + \beta X} \\ P(X) &= (1 - P(X)) \cdot e^{\beta_0 + \beta X} \\ \frac{P(X)}{1 - P(X)} &= e^{\beta_0 + \beta X} \end{aligned}$$

Lijevu stranu $\left(\frac{P(X)}{1-P(X)}\right)$ u konačno dobivenoj formuli nazivamo izgledima (eng. odds) te može poprimiti vrijednosti između 0 i ∞ . Izgledi se obično umjesto vjerojatnosti koriste u kladionicama jer daju intuitivniju informaciju. Kada na obje strane posljednje formule primijenimo log (logaritmiramo) dobivamo:

$$\log\left(\frac{P(X)}{1-P(X)}\right) = \beta_0 + \beta X \quad (4.3)$$

Lijeva strana formule (4.3) predstavlja takozvane log-izglede ili logit. Iz formule vidimo da je logit linearan u X .

Učenjem modela logističke regresije procjenjujemo koeficijente β_0 i β . Kao što smo i spomenuli za procjenu tih koeficijenata ćemo koristiti maksimalnu vjerodostojnost. Dakle, želimo pronaći koeficijente β_0 i β takve da se predikcija vjerojatnosti ($\hat{P}(x_i)$) oznake klase (promatramo za svaki x_i zasebno) podudara sa stvarnom vrijednosti klase (y_i). Stoga sljedeću funkciju koja formalizira taj postupak zovemo funkcijom vjerodostojnosti:

$$L(\beta_0, \beta) = \prod_{i: y_i=1} P(x_i) \cdot \prod_{i: y_i=0} (1 - P(x_i)) \quad (4.4)$$

Cilj je pronaći β_0 i β (tj. $\beta_0, \beta_1, \dots, \beta_{29}$) takve da funkcija vjerodostojnosti iz (4.4) postiže maksimalnu vrijednost (primijetimo da $P(x_i)$ dobijemo uvrštavanjem vrijednosti x_i u formulu (4.1)), pri čemu (x_i, y_i) predstavlja jedan primjer iz skupa za učenje. Označimo s $\hat{\beta}_0$ i $\hat{\beta}$ koeficijente koje smo dobili maksimiziranjem funkcije vjerodostojnosti, tada predikciju klase za novi (dosad neviđeni) primjer x dobivamo uvrštavanjem tog x u formulu:

$$\hat{P}(x) = \frac{e^{\hat{\beta}_0 + \hat{\beta}x}}{1 + e^{\hat{\beta}_0 + \hat{\beta}x}}$$

Logističku regresiju koristimo jer je prilično jednostavan i brz algoritam tako da po potrebi možemo bez problema podešavati parametre. U eksperimentu podešavamo sljedeće parametre:

1. C - regularizacijski parametar, inverzno proporcionalan λ regulatoru
2. 'kazna' - određuje koju regularizacijsku tehniku koristimo: lasso ili ridge. Kod lassa je 'kazna' apsolutna vrijednost veličine koeficijenata (formula (4.5)), dok je kod ridgea kazna kvadratna veličina koeficijenata (formula (4.6)).

Formula za lasso (kod logističke regresije):

$$lCost = \sum_{i=1}^n \left(y_i(\beta_0 + x_i\beta) - \log(1 + e^{\beta_0 + x_i\beta}) \right) + \lambda \sum_{j=1}^p |\beta_j| \quad (4.5)$$

Formula za ridge (kod logističke regresije):

$$rCost = \sum_{i=1}^n \left(y_i(\beta_0 + x_i\beta) - \log(1 + e^{\beta_0 + x_i\beta}) \right) + \lambda \sum_{j=1}^p \beta_j^2 \quad (4.6)$$

4.3 Stablo odluke

Stablo odluke vjerojatno je najjednostavniji model strojnog učenja jer je sličan načinu na koji čovjek razmišlja. Stablo odluke možemo zamisliti kao skup ugniježđenih **ako-onda** izjava. Pošto je problem prepoznavanja kartičnih prevara klasifikacijski, posvetit ćemo se klasifikacijskim stablima odluke (postoje i regresijska). Svako stablo odluke sadržava sljedeće entitete:

1. čvor - označava atribut (značajku) po kojoj se vrši podjela
2. veza - povezuje čvor s drugim čvorom ili listom, veza iz nekog čvora ustvari predstavlja određenu vrijednost koju atribut može poprimiti
3. list - označava terminalni čvor, odnosno čvor u kojem donosimo odluku (pridjeljujemo oznaku klase primjeru)

Originalan postupak, zvan ID3 (Iterative Dichotomiser) za generiranje stabla odluke iz podataka osmislio je Robert Quinlan. Opišimo ukratko kako ID3 (prikazan algoritmom 12) iz podataka generira stablo odluke. Dakle, na ulazu je skup za učenje D . Ako svi podaci imaju istu oznaku klase, tada je čvor u kojem se trenutno nalazimo list - označavamo ga s oznakom klase svih primjera. Ako nije, koristeći neku funkciju izaberemo neki atribut x_j . Konkretno, kod skupa podataka za prepoznavanje kartičnih prevara x_j može biti ili 'Amount' (poslije skaliranja) ili jedna od anonimiziranih značajki (T1, ..., T28). Nakon što smo izabrali atribut x_j stvaramo novi čvor (označen s x_j) u stablu. S D_k označimo skup svih podataka iz D koji za atribut x_j poprimaju vrijednost v_{jk} . Rekurzivno koristimo ID3

algoritam na podstablu koje na ulazu ima skup podataka D_k . Tada za svaku vrijednost v_{j_k} koju atribut može poprimiti stvaramo vezu (označenu s vrijednošću atributa v_{j_k}) između stabla D i svih podstabala D_k .

Algorithm 12 ID3

```

1: procedure ID3( $D$ )
2:    $c \leftarrow \text{getClasses}(D)$ 
3:   if  $\text{len}(c) = 1$  then                                     ▶ ako u  $D$  imamo samo jednu klasu
4:     return  $\text{Node}(c[0])$                                      ▶ vratimo list s oznakom klase
5:   end if
6:    $\text{splitter} \leftarrow \text{getSplittingAttribute}(D, f)$ 
7:    $DT \leftarrow \text{Node}(\text{splitter})$ 
8:   for each  $v_k \in \text{values}(\text{splitter})$  do
9:      $\text{indices}_k \leftarrow \text{np.where}(\text{np.array}(D) == v_k)[0]$ 
10:     $D_k \leftarrow \text{getExamples}(D, \text{indices}_k)$ 
11:     $DT_k \leftarrow \text{ID3}(D_k)$ 
12:     $\text{connect}(DT, DT_k)$ 
13:  end for
14:  return  $DT$ 
15: end procedure

```

U drugom redu algoritma 12 funkcija `getSplittingAttribute` služi za biranje atributa po kojem 'granamo' u trenutnom koraku. Prirodno se zapitati kako biramo taj atribut? Cilj je sigurno izgraditi što jednostavnije (manje) stablo zato što ćemo lakše donositi odluke prilikom klasifikacije budućih primjera. Stoga kažemo da je atribut dobar za grananje ako je svaki sljedeći čvor (onaj koji dobijemo grananjem atributa po vrijednostima), 'čist' koliko je god moguće, to jest ako se distribucija primjera u svakom sljedećem čvoru sastoji većinom od primjera jedne klase. Primijetimo da bi idealno bilo da se svaki sljedeći čvor sastoji isključivo od primjera jedne klase jer bi tada taj čvor ustvari bio list s oznakom te jedine klase koja se nalazi u čvoru (listu).

Dakle, želimo odrediti mjeru koja će za grananje preferirati attribute koji imaju visok stupanj 'reda'. Pritom za 'red' u jednom čvoru kažemo da je:

1. **maksimalan** - ako svi primjeri u tom čvoru pripadaju istoj klasi. Tada je taj čvor list jer sigurno znamo kako klasificirati takve primjere.
2. **minimalan** - ako u tom čvoru ima jednak broj primjera svake klase (u našem slučaju 2 klase). Tada je jednako vjerojatno da će primjer pripadati bilo kojoj od klasa.

Da bismo izmjerili količinu '(ne)reda' koristimo *entropiju*. Entropiju možemo smatrati mjerilom informacija u čvoru. Ako je maksimalan red u čvoru imamo sve potrebne infor-

macije (jer znamo koje su klase svi primjeri iste klase), a ako je u čvoru red minimalan tada nemamo nikakve informacije. Entropija je definirana sljedećom formulom:

$$E = - \sum_{k=1}^K p_k \cdot \log p_k \quad (4.7)$$

U formuli (4.7) K označava ukupan broj klasa u skupu podataka, dok p_k označava udio primjera koji pripadaju klasi k . Konkretno, u skupu podataka za kartične primjere imamo 2 klase pa formulu možemo zapisati kao:

$$E = -p_0 \cdot \log_2 p_0 - p_1 \cdot \log_2 p_1$$

Primijetimo da entropija mjeri kvalitetu podjele samo za pojedini podskup primjera (onaj koji odgovara jednoj vrijednosti atributa), a mi želimo dobiti kvalitetu cijele podjele (za sve atribute). Da bismo riješili taj problem izračunamo sve entropije i težinski ih usrednjimo. Taj postupak je prikazan sljedećom formulom:

$$I(D, x_i) = \sum_j \frac{|D_j|}{|D|} \cdot E(D_j) \quad (4.8)$$

U formuli (4.8) D predstavlja cijeli skup podataka, x_i predstavlja jedan od atributa koje koristimo za učenje (sjetimo se početka ovog poglavlja), dok $D_j \subseteq D$ predstavlja skup svih primjera kojima je vrijednost atributa x_i jednaka j . Sada možemo definirati *informacijsku dobit* atributa x_i kao:

$$InfoGain(D, x_i) = E(D) - I(D, x_i) = E(D) - \sum_j \frac{|D_j|}{|D|} \cdot E(D_j) \quad (4.9)$$

Atribut koji donosi maksimalan red je sada upravo onaj za koji je informacijska dobit najveća. Napomenimo da umjesto maksimiziranja informacijske dobiti možemo i minimizirati težinski usrednjenu entropiju (formula (4.8)) jer je $E(D)$ konstantno za sve atribute.

Problem koji se kod informacijske dobiti može pojaviti je da skup podataka sadrži atribute koji mogu poprimiti puno različitih vrijednosti, primjerice zamislimo da skup podataka za prepoznavanje kartičnih prevara sadrži identifikacijski broj transakcije. Pošto bi svaka transakcija imala jedinstveni identifikacijski broj podjela po tom atributu bi imala maksimalnu informacijsku dobit, ali bi u svakom listu bio samo 1 primjer, te bi bilo ukupno preko 280000 listova. Također, učenje po takvim atributima je loše jer dolazi do overfittinga (praktično je napamet naučeno). Zbog toga definiramo *unutarnju informaciju* atributa. Unutarnja informacija atributa predstavlja količinu informacija koju trebamo da bismo odredili kojoj grani atributa neki primjer pripada, te je zadana formulom:

$$IntInfo(D, x_i) = \sum_j \frac{|D_j|}{|D|} \log \left(\frac{|D_j|}{|D|} \right) \quad (4.10)$$

Oznake u formuli (4.10) su jednake kao i u formuli (4.8). Primijetimo da su atributi s višom unutarnjom informacijom manje korisni (jedan od takvih bi upravo bio identifikacijski broj transakcije). Sada kada znamo što je unutarnja informacija atributa možemo to povezati s informacijskom dobiti da bi algoritam bio manje pristran ka biranju atributa koji poprimaju mnogo vrijednosti. Stoga definiramo *udio dobiti* kao:

$$GainRatio(D, x_i) = \frac{InfoGain(D, x_i)}{IntInfo(D, x_i)} \quad (4.11)$$

Zbog ranije navedenog primjera *GainRatio* je bolje koristiti od *InfoGain*.

Postoji mnogo alternativnih mjera entropiji. Ona najpopularnija, koju i mi koristimo u eksperimentalnom dijelu se zove *gini indeks*. Gini indeks se također koristi kao mjera nečistoće te je zadan formulom:

$$Gini(D) = \sum_{k=1}^K p_k \cdot (1 - p_k) \quad (4.12)$$

Oznake imaju isto značenje kao i kod entropije. Izvedimo sada formulu za gini index kada imamo samo 2 klase:

$$Gini(D) = p_0 \cdot (1 - p_0) + p_1 \cdot (1 - p_1) \quad (4.13)$$

$$= p_0 - p_0^2 + p_1 - p_1^2 \quad (4.14)$$

$$= 1 - p_0^2 - p_1^2 \quad (4.15)$$

Kao i za entropiju možemo definirati težinski usrednjeni gini indeks kao:

$$Gini(S, x_i) = \sum_j \frac{|D_j|}{|D|} \cdot Gini(D_j) \quad (4.16)$$

Uglavnom se ili gini indeks ili entropija koriste da bismo dobili atribut za podjelu u funkciji `getSplittingAttribute` algoritma 12. No uz sve to možemo uočiti da *ID3* i dalje ima dosta problema. Neki od najbitnijih su sigurno:

1. numerički atributi - mogu poprimiti puno vrijednosti što bi za *ID3* algoritam bilo problematično (velik stupanj fragmentacije)
2. nedostatak vrijednosti - u stvarnim podacima često će vrijednosti nekih atributa za neke primjere biti nepoznate, pa je bitno da to ne ugrozi algoritam
3. robusnost na šum, overfitting - primijetimo da mala promjena u vrijednostima atributa može biti uzrok drukčije klasifikacije

Da bi se riješili ti problemi stvoren je C4.5 algoritam. C4.5 je unaprijeđeni ID3 koji rješava iznad navedene probleme. Opišimo kako C4.5 rješava svaki od tih problema.

1. numerički atributi - C4.5 koristi binarne podjele (npr. '*Amount*' < 100\$). Zato što takvi atributi mogu poprimiti jako puno vrijednosti u obzir uzimamo svaku moguću točku (za binarnu podjelu) te računamo informacijsku dobit (ili gini indeks, ovisi koju metriku koristimo). Nakon što prođemo kroz sve moguće podjele znat ćemo koja je točka najbolja i upravo tu točku odabiremo.
2. nedostatak vrijednosti - u najboljem slučaju atribut za koji nekim primjerima nedostaje vrijednost neće uopće biti uzet u obzir za podjelu, no ako hoće radimo sljedeće. Prvo podijelimo primjer na 'komadiće' tako da za svaku izlaznu granu (vezu) atributa imamo jedan komadić primjera. Svakom komadiću primjera pridijelimo težinu (suma težina svih komadića je 1) u skladu s popularnošću grane (što je grana popularnija, komadić primjera u toj grani poprima veću težinu).
3. robusnost na šum, overfitting - primijetimo da se kod stabla odluke za svaki primjer može naći grana u koju se primjer savršeno uklapa (to izlazi iz načina na koji se gradi stablo). Tako da jako često dolazi do overfittinga, odnosno mala promjena jednog atributa može pridonijeti pogrešnoj klasifikaciji. Da do toga ne bi došlo koriste se tehnike obrezivanja stabla: predobrezivanje (eng. pre-pruning) i naknadno obrezivanje (eng. post-pruning). Predobrezivanjem zaustavljamo rast grane ako informacije koje dobivamo nisu pouzdane (uglavnom ako grana postane presložena, pa ne može dobro generalizirati). Da bismo odredili kad zaustaviti rast koriste se statistički testovi (najčešće *chi-squared test*). Rast zaustavljamo kada ne postoji statistički značajna veza među atributom i oznakom klase. S druge strane naknadnim obrezivanjem izgradimo stablo klasičnim postupkom i onda naknadno neke čvorove mijenjamo listovima. U praksi se preferira naknadno obrezivanje zato što predobrezivanje može stati prerano (primjerice za *XOR* probleme).

Sada još ostaje pitanje kako procijeniti stopu pogreške prilikom obrezivanja? Kod C4.5 se pomoću skupa za učenje 'izvuku' intervali pouzdanosti (eng. confidence intervals) te se pesimistično pretpostavlja da je prava pogreška gornja granica intervala. Preciznije, ako je W od N primjera pogrešno klasificirano, možemo na to gledati kao na binomnu distribuciju (primjerice N puta bacamo novčić, W puta je pala glava). Uz to još uzimamo nivo pouzdanosti CF (u %), tada će gornja granica za grešku biti $Upper_{CF}(W, N)$. Za C4.5 ta pesimistična procjena greške u nekom čvoru se može zapisati formulom:

$$error = \frac{e_T + \frac{z^2}{2N} + z \sqrt{\frac{e_T}{N} - \frac{e_T^2}{N} + \frac{z^2}{4N^2}}}{1 + \frac{z^2}{N}} \quad (4.17)$$

Pritom u formuli (4.17) z izvučemo pomoću nivoa pouzdanosti (primjerice ako je $CF = 25\%$, tada je $z = 0.6915$ - iz tablice za normalnu distribuciju), e_T označava grešku na skupu za učenje, a N je broj primjera pokrivenih listom. Iz toga sada možemo procijeniti grešku za podstablo kao težinsku sumu procijenjenih grešaka u svim listovima tog podstabla (ako je u listu više primjera, težina je veća). Stablo podrezujemo u nekom čvoru kada je procjena greške u podstablu manja nego procjena greške u tom čvoru. Osnovni princip je znači sljedeći:

1. podijelimo skup za učenje u skup koji koristimo za stvaranje stabla (označimo ga s D_G) i skup koji koristimo za obrezivanje (označimo ga s D_P)
2. izgradimo stablo odluke koje ispravno klasificira svaki primjer $p \in D_G$
3. Dok god procjena greške na skupu D_P ne raste:
 - a) pokušamo zamijeniti svaki čvor s listom (pri čemu je oznaka lista jednaka oznaci većinske klase u podstablu gledanog čvora)
 - b) procijenimo grešku na novom stablu (onom koje je nastalo zamjenom čvora listom)
 - c) zamjenu čvora listom zadržavamo ako se greška ne povećava (u odnosu na grešku bez zamjene)
 - d) kada prođemo sve čvorove, stablo će biti promijenjeno (podrezano) na mjestima gdje smo smanjili grešku kad smo zamijenili čvor listom

4.4 Bagging

Kao što smo u prethodnom odjeljku vidjeli da su stabla odluke dosta intuitivna metoda, ali dosta problematična pa ne čudi činjenica da samo po sebi stablo odluke i nije najbolji klasifikator. No kada se koriste u ansamblima mogu biti poprilično jak alat. Prvi je to uočio Leo Breiman [7] 1996. godine. On je primijetio da se više prediktora može 'usrednjiti', te je taj postupak nazvao *baggingom* (skraćeno za Bootstrap AGGREGatING). Bagging je ansambl metoda strojnog učenja, a iz naziva možemo zaključiti da se algoritam sastoji od 2 ključna koraka: bootstrapping i agregiranje. Opišimo prvo bootstrapping. Bootstrapping je metoda ponavljano uzorkovanja (slično kao i metode iz poglavlja 3) samo s nešto drukčijim ciljem. Bootstrapping koristimo kada imamo već spreman skup za učenje D (u našem slučaju dobiven nakon metoda preuzorkovanja, odnosno poduzorkovanja) da bismo stvorili replike D_b koje koristimo da bismo dobili dodatne informacije o skupu podataka (pristranost i varijancu). Svaka od replika D_b se stvara tako da slučajno izvučemo $|D|$ primjera iz D s ponavljanjem, odnosno moguće je više puta izvući isti primjer iz D i staviti

ga u D_b (primijetimo da tako neki primjeri iz D uopće neće biti prisutni u D_b). Taj proces se ponavlja K puta (Breiman je u svome radu za klasifikacijske probleme ponavljao 50 puta), pri čemu nijedna replika ne ovisi o drugoj, odnosno dobivene su kao da se stvaraju paralelno. Zatim svaki od D_{b_i} , za $i = 1, \dots, K$, koristimo za učenje koristeći nepodrezana stabla odluke kao model (inače se može koristiti bilo koji kompleksniji model jer bagging eliminira varijancu, pa na njima najbolje djeluje). Nakon učenja agregiramo te modele metodom glasanja (za klasifikaciju). Proces je prikazan algoritmom 13.

Algorithm 13 Bagging

```

1: procedure BAGGING( $D, K, x$ )
2:   for  $i = 1$  to  $K$  do
3:      $D_{b_i} \leftarrow \text{Bootstrap}(D)$             $\triangleright$  Bootstrap uzorkovanje skupa  $D_{b_i}$  iz  $D$ 
4:      $h_i \leftarrow \text{DT}(D_{b_i})$ 
5:   end for
6:    $h_{\text{bag}}(x) \leftarrow \frac{1}{K} \sum_{i=1}^K h_i(x)$             $\triangleright$  Agregiranje za neki  $x$ 
7: end procedure

```

Opišimo klasifikaciju jednog primjera x iz skupa za prepoznavanje kartičnih prevara. Učenjem na bootstrap replikama dobivamo modele h_1, \dots, h_K . Za svaki $h_i(x)$ vrijedi da će poprimiti jednu od vrijednosti iz skupa $\{0, 1\}$ (sjetimo se da 0 predstavlja oznaku klase ako je transakcija valjana, a 1 ako je prevara). Tada će i usrednjena suma ($h_{\text{bag}}(x)$) iz šestog reda algoritma 13 biti broj između 0 ili 1. Ako vrijedi $h_{\text{bag}}(x) < 0.5$ tada će transakcija x biti prepoznata kao valjana transakcija, a ako vrijedi $h_{\text{bag}}(x) > 0.5$ tada će transakcija x biti prepoznata kao prevara. Bagging na nepodrezanim stablima odluke je vrlo uspješan zato što se na njima svaki primjer iz skupa za učenje može 'savršeno' klasificirati (sjetimo se odlomka o stablima odluke), ali su stabla građena na drukčijim replikama D_{b_i} što omogućava različite predikcije modela h_i .

4.5 Slučajne šume

Slučajne šume možemo shvatiti kao unaprijeđenu verziju bagginga na stablima odluke. Kao i kod bagginga koristimo bootstrapping da bismo dobili skupove D_{b_1}, \dots, D_{b_K} koje koristimo za izgradnju stabala. Razlika je što kod slučajnih šuma prilikom izgradnje stabala odluke, svaki put kada radimo podjelu u čvoru (po granama koja svaka predstavlja jednu vrijednost atributa), samo slučajno odabran skup atributa uzimamo u obzir prilikom podjele (od njih izaberemo najbolji). U svakom novom čvoru opet se slučajno odabire skup atributa koji se smiju koristiti prilikom nove podjele. Tipično uzimamo $q \approx \sqrt{m}$ atributa za taj slučajno odabran skup, pri čemu je m ukupan broj atributa. Konkretno u skupu podataka za prepoznavanje kartičnih prevara imamo $m = 29$ atributa ('Amount', T1, ..., T28), pa u

svakom koraku slučajno biramo skup od 5 ili 6 atributa od kojih jedan koristimo za podjelu. Algoritam 14 prikazuje proces građenja jednog stabla u slučajnoj šumi.

Algorithm 14 Izgradnja stabla u slučajnoj šumi

```

1: procedure RANDOMTREE( $D$ )
2:    $node \leftarrow Node(D)$  ▷ stvaramo čvor s podacima  $D$ 
3:    $c \leftarrow getClasses(D)$ 
4:   if  $len(c) = 1$  then
5:     return  $node$ 
6:   end if
7:    $features \leftarrow getRemainingFeatures(D)$  ▷ samo atributi koji nisu prije iskorišteni
8:   if  $features = \emptyset$  then
9:     return  $node$ 
10:  end if
11:   $q \leftarrow sqrt(len(features))$ 
12:   $randFeatures \leftarrow selectRandomFeatures(features, q)$ 
13:   $bestFeature \leftarrow getBestFeature(randFeatures)$  ▷ onaj koji daje najbolju podjelu
14:   $values \leftarrow getValues(bestFeature)$  ▷ sve vrijednosti koje značajka može poprimiti
15:  for each  $v \in values$  do
16:     $childNode \leftarrow Node(D, v)$  ▷ primjeri kojima je  $bestFeature = v$ 
17:     $RandomTree(childNode)$ 
18:  end for
19: end procedure

```

Isprva se zanemarivanje većine atributa (zanemarujemo 23 od 29) prilikom podjele čini dosta neintuitivno, ali je ustvari vrlo korisno. Objašnjenje za to je što u skupu podataka uvijek postoje bolji i lošiji atributi za podjelu i iako se na različitim bootstrap replikama D_{b_i} mogu izabrati različiti atributi, to često i nije slučaj. Razlog tome je što su bolji atributi i dalje bolji iako koriste malo drugačije podatke za izgradnju stabala, a ako izgradimo puno sličnih stabala glasovanje modela će isto biti slično pa se model dobiven baggingom neće uvelike razlikovati od običnog stabla odluke. Zbog toga bagging koji koristi nepodrezana stabla odluke i dalje može biti sklon overfittingu. Koristeći slučajne šume u svakom koraku biramo jedan atribut iz slučajno odabranog podskupa svih atributa. Vidimo da u toj situaciji čak i iz dva potpuno ista bootstrap skupa D_{b_i} i D_{b_j} možemo dobiti dva potpuno drugačija stabla odluke. Taj proces koji koriste slučajne šume zovemo dekorelacijom stabala. Primijetimo da je broj atributa q koje uzimamo u razmatranje jednak korijenu ukupnog broja atributa da bi mogli balansirati pristranost i varijancu. Ako uzmemo $q = len(features)$ slučajne šume se svode na klasični bagging na stablima odluke, za koji smo malo prije napomenuli da može stvarati slična stabla, pa biti sklon overfittingu. S druge strane, ako

uzmemo $q = 1$ tada praktično 'forsiramo' podjelu po slučajno odabranom atributu, pa ćemo dobiti veliku pristranost. Slučajne šume su zbog svega navedenog i danas jedan od najkorištenijih algoritama strojnog učenja, pa se i mi u eksperimentalnom dijelu posebno posvećujemo toj ansambl metodi (koristimo slučajne šume kao klasifikator u kombinaciji sa svim metodama ponovnog uzorkovanja).

4.6 Boosting

Boosting je, kao i bagging, ansambl metoda u kojoj kombiniramo puno slabih klasifikatora s ciljem da dobijemo jedan jači klasifikator. Pritom slabim modelima smatramo modele koji su nešto bolji od slučajnog klasifikatora. Prisjetimo se kako smo gradili stabla koristeći bagging. Prvo bismo stvorili bootstrap replike podataka D_{b_i} koje bismo koristili za učenje pojedinačnog modela h_i , te bismo tako stvorili K modela. Na kraju bismo ih sve agregirali u jedan model koji bi funkcionirao na temelju glasanja (za klasifikaciju), gdje je svaki model davao jedan glas prilikom klasifikacije dotad nepoznatog primjera x . Razlika boostinga u odnosu na bagging ansambl je što se kod boostinga modeli grade sekvencijalno, tj. jedan za drugim na modificiranim verzijama podataka (ranije izgrađenim stablima). Znači boosting ne koristi bootstrapping da bi dobio različita stabla, nego koristi stabla izgrađena u prethodnim koracima kako bi stvorio novo stablo. Krajnja predikcija je kombinacija predikcija pojedinačnih modela pri čemu je svakom modelu dodijeljen težinski faktor.

Općenita procedura kojom se boosting izvršava je kao i za bagging prilično jednostavna. Opišimo kako bi ta procedura izgledala kod problema binarne klasifikacije (jer je upravo to problem koji promatramo). Uzmimo da je skup za učenje D podijeljen na 3 jednaka dijela: D_1, D_2, D_3 te uzmimo slabi klasifikator takav da ispravno klasificira primjere iz D_1 i D_2 , ali da pogrešno klasificira primjere iz D_3 . Takav klasifikator griješi na 33% primjera (podijelili D na tri dijela upravo da bismo prikazali slabi klasifikator). Označimo taj 'početni' model s h_1 . Ideja boostinga je da stvorimo novi model h_2 s ciljem da taj model 'ispravi' greške koje je radio h_1 . To činimo tako da modifikacijom podataka 'naglasimo' greške od h_1 , pa će posljedično tome novi model biti fokusiraniji na primjere iz D_3 . Pretpostavimo da smo takvim postupkom dobili h_2 koji griješi na D_1 , ali dobro klasificira primjere iz D_2 i D_3 . Kombiniranjem modela h_1 i h_2 dobit ćemo model koji ispravno klasificira primjere iz D_2 , ali pravi nekoliko grešaka na D_1 i D_3 . Taj postupak se analogno nastavlja dok ne dobijemo zadovoljavajuće rezultate.

Najpoznatiji boosting algoritam zove se AdaBoost (Adaptive Boosting), a prvi su ga predstavili Freund i Schapire [22]. Adaboost funkcionira tako da prilikom izgradnje novog modela mijenja težine primjerima u skupu za učenje. To jest primjeri koji su pogrešno klasificirani prethodnim modelima će prilikom izgradnje novog modela imati veće težine (da se model fokusira na njih). Na kraju se glasa težinskom većinom (veći utjecaj prilikom glasovanja imaju točniji modeli). Pseudokod za Adaboost je dan algoritmom 15.

Algorithm 15 Adaboost

```

1: procedure ADABOOST( $D, K$ )
2:    $n \leftarrow \text{len}(D)$ 
3:    $w_1 \leftarrow \text{np.full}(n, \frac{1}{n})$ 
4:   for  $k = 1$  to  $K$  do
5:      $h_k \leftarrow \text{DecisionTree}(D, w_k)$ 
6:      $\text{err}_k \leftarrow P_{x \sim w_k}(h_k(x) \neq f(x))$ 
7:      $\alpha_k \leftarrow \frac{1}{2} \log\left(\frac{1 - \text{err}_k}{\text{err}_k}\right)$ 
8:     for  $i = 1$  to  $n$  do
9:        $w_{k+1}[i] \leftarrow w_k[i] \cdot \exp(\alpha_k \cdot |h_k(x_i) - y_i|)$ 
10:    end for
11:  end for
12:  return  $H(x) \leftarrow \text{sign}\left(\sum_{k=1}^K \alpha_k h_k(x)\right)$ 
13: end procedure

```

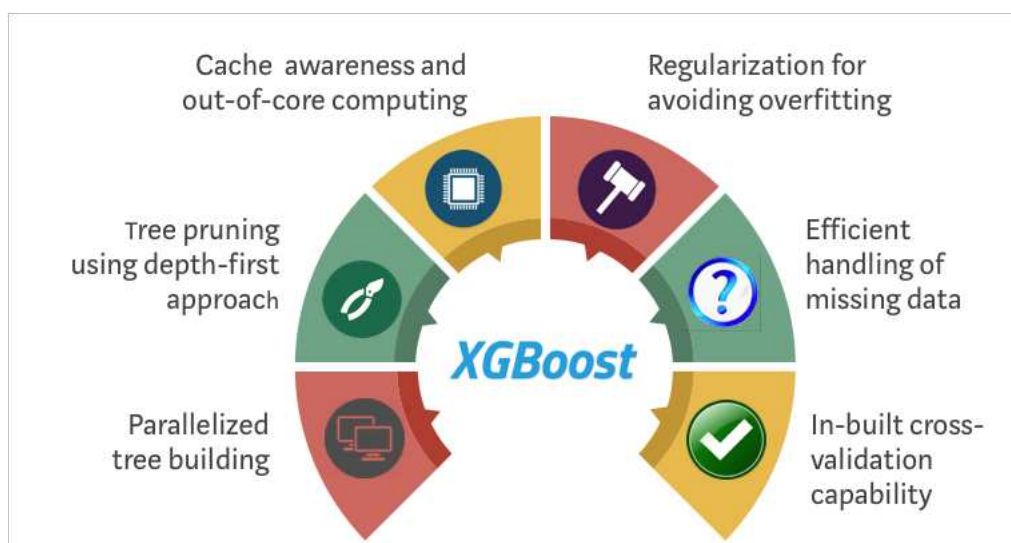
Opišimo malo detaljnije AdaBoost sada kada imamo pseudokod. Prvo inicijaliziramo sve težine na $\frac{1}{n}$, gdje je n broj primjera u skupu D (funkcija $\text{np.full}(n, \frac{1}{n})$ iz numpy biblioteke puni niz duljine n vrijednostima $\frac{1}{n}$). Onda u svakoj od K iteracija radimo sljedeće:

1. treniramo model h_k koristeći stablo odluke na skupu D s težinama w_k .
2. računamo grešku modela h_k (označenu s err_k). Funkcija $P_{x \sim w_k}(h_k(x) \neq f(x))$ broji koliko puta se naš model h_k razlikuje od stvarne vrijednosti $f(x) = y$ za primjer x
3. podešavamo faktor α koji koristimo kao 'korektor' težina
4. za svaki primjer podešavamo težine za sljedeću iteraciju koristeći težinu iz prethodne iteracije te eksponencijalnu funkciju u kojoj se kao faktor pojavljuje α iz prethodnog koraka u iteraciji

Boosting jako dobro radi na modelima s visokom pristranosti (jako jednostavni modeli) jer se kombinacijom takvih dobije dosta složeniji model (u svakoj iteraciji se pristranost smanjuje). No, pokazano je da boosting smanjuje i varijancu. Iako je konačni model H složen, margine između primjera različitih klasa se povećavaju. Pošto se margine povećavaju lakše je odrediti kojoj klasi pripada neki novi dotad neviđeni primjer - znači da se smanjuje varijanca.

Najveći problem Adaboosta je osjetljivost na outliere (jako šumovite podatke). Takvi primjeri će često dobivati sve veću težinu, ali i dalje neće biti ispravno klasificirani. Da bi se taj problem riješio osmišljeni su moderniji boosting algoritmi. Danas je najpopularniji XGBoost (eXtreme Gradient Boosting), kojeg je 2014 uveo Chen [14]. XGBoost je vrsta

ansambl algoritma strojnog učenja bazirana na stablima odluke, te kao što iz punog naziva može naslutiti koristi okvir za gradijentni boosting. Gradijentni boosting znači da se koristi metodom gradijentnog spusta da bi u narednim iteracijama boostinga minimizirao grešku (detaljno o gradijentnom boostingu je moguće pronaći na [25]). XGBoost je optimizirana verzija gradijentnog boostinga: koristi obrezivanje stabala, paralelnu obradu, rješava probleme vrijednosti koje nedostaju te ima dodatne regularizacijske parametre da bi se izbjegla pristranost i overfitting. Prednosti XGBoosta nad običnim gradijentnim boostingom mogu se vidjeti i na slici 4.2 (preuzeta s [43]).



Slika 4.2: Prednosti XGBoosta

Opišimo kako točno XGBoost koriste svoje prednosti:

1. **Paralelizacija** - XGBoost koristi paraleliziranu implementaciju prilikom procesa izgradnje sljedećih stabala. To je moguće jer su petlje koje se koriste za gradnju stabala zamjenjive. Vanjska petlja numerira listove, dok druga unutarnja petlja računa značajke. Kada su petlje tako ugniježdene paralelizacija je usporena jer korak vanjske petlje ne može početi bez da se izvrši unutarnja petlja (koja je računalno složenija). Stoga, da bi se algoritam ubrzao, poredak petlji je promijenjen pomoću inicijalizacije tako što prolazimo po svi primjerima, i sortiramo ih koristeći više drevi koje paralelno rade. Takva zamjena poboljšava performanse algoritma (u smislu brzine izvođenja) tako što neutralizira opće troškove paralelizacije.
2. **Obrezivanje stabala** - XGBoost za obrezivanje prvo koristi parametar `max_depth` koji služi za naknadno obrezivanje stabala (ako postanu preduga), a tek kasnije kri-

terij za zaustavljanje (vezan za porast greške prilikom obrezivanja) koji je klasično korišten za gradijentni boosting. Takvim rezanjem grana koje su postale 'preduge' značajno se ubrzava algoritam.

3. **Optimizacija hardvera** - Algoritam je dizajniran da bi učinkovito koristio dostupan hardver. Preciznije, koristi se 'cache awareness' (cache je najbrža vrsta memorije u računalu) da bi se alocirala memorija u spremnicima. Tu memoriju dretve koriste kako bi pohranile podatke o gradijentima. Također se koristi 'out-of-core' računanje s ciljem optimizacije diskovnog prostora kada se barata s velikim skupovima podataka koje se zbog veličine ne može čuvati u memoriji (sjetimo se da skup podataka za prepoznavanje kartičnih prevara sadržava podatke od nešto manje od 300000 transakcija, nakon preuzorkovanja još više).
4. **Regularizacija** - XGBoost koristi regularizacijske tehnike Lasso (L1) zadan formulom (4.5) ili Ridge (L2) zadan formulom (4.6).
5. **Vrijednosti koje nedostaju** - XGBoost 'pretpostavlja' vrijednosti značajke u primjerima gdje nedostaju tako što nauči najbolju vrijednost značajke (računanjem greške na trening skupu).
6. **Weighted Quantile Sketch** - XGBoost koristi distribuiranu verziju Quantile Sketch algoritma koristeći težine da bi efikasno pronašao najbolju točku za podjelu. Quantile Sketch algoritam kombinira vrijednosti dobivene s više računala (jer je skup podataka prevelik i jednom računalu bi trebalo predugo) da bi se napravio histogram. Taj histogram se onda koristi da bi se izračunali kvantili. XGBoost umjesto 'normalnih' kvantila (obično brojanje primjera - jednakobrojne podgrupe) koristi 'težinske' kvantile (brojanje primjera koristeći težine - grupe jednakih težina). Pritom se 'težine' računaju kao druga derivacija funkcije gubitka. Za klasifikacijski problem to znači da se težina dobije kao $Weight = PrevProba \cdot (1 - PrevProba)$.

Uz XGBoost i AdaBoost ćemo u eksperimentalnom dijelu koristiti i sljedeće boosting algoritme: LightGBM (Light Gradient Boosting Machine) [35], te CatBoost (Categorical Boosting) [47].

4.7 Metoda najbližih susjeda

Metodu najbližih susjeda već smo spominjali kada smo govorili o metodama ponovnog uzorkovanja. Kod poduzorkovanja se koristila da bi se izbacili primjeri koji bi bili pogrešno klasificirani koristeći 3 najbliža susjeda (CNN, ENN), ali i u Tomekovim vezama (gdje pozitivan i negativan primjer trebaju jedan drugom biti najbliži susjedi). S druge strane se kod

preuzorkovanja koristila kod varijacija SMOTE algoritama kada bi između pozitivnog primjera i jednog slučajno odabranog najbližeg susjeda (od 5) stvarali sintetski primjeri. Kao što iz tih primjena vidimo da metoda za ulaz ima ključan parametar k koji označava koliko najbližih susjeda je potrebno zapamtiti. Metodu najbližih susjeda predložio je Thomas Cover početkom 1960-ih. Metoda je prilično jednostavna:

1. želimo klasificirati novi primjer $e = (x, y)$ koristeći metodu $k - NN$
2. računamo udaljenosti između primjera e i svih ranije poznatih primjera $e_i = (x_i, y_i) \in D$ (gdje D označava skup za učenje)
3. spremamo udaljenost u sortirani niz na pravo mjesto
4. klasificiramo primjer e tako što mu dodijelimo najčešću klasu među k najbližih primjera

Glavni parametri za metodu najbližih susjeda su: broj susjeda (k) i snaga (p). Pritom snaga određuje koju metriku iz familije Minkowski koristimo. Te metrike su zadane formulom:

$$d_p(x_1, x_2) = \sqrt[p]{\sum_{j=1}^m (x_{1j} - x_{2j})^p} \quad (4.18)$$

Napomenimo samo da u formuli (4.18) m označava broj značajki od primjera x_1 , odnosno x_2 . Dvije najčešće korištene metrike su:

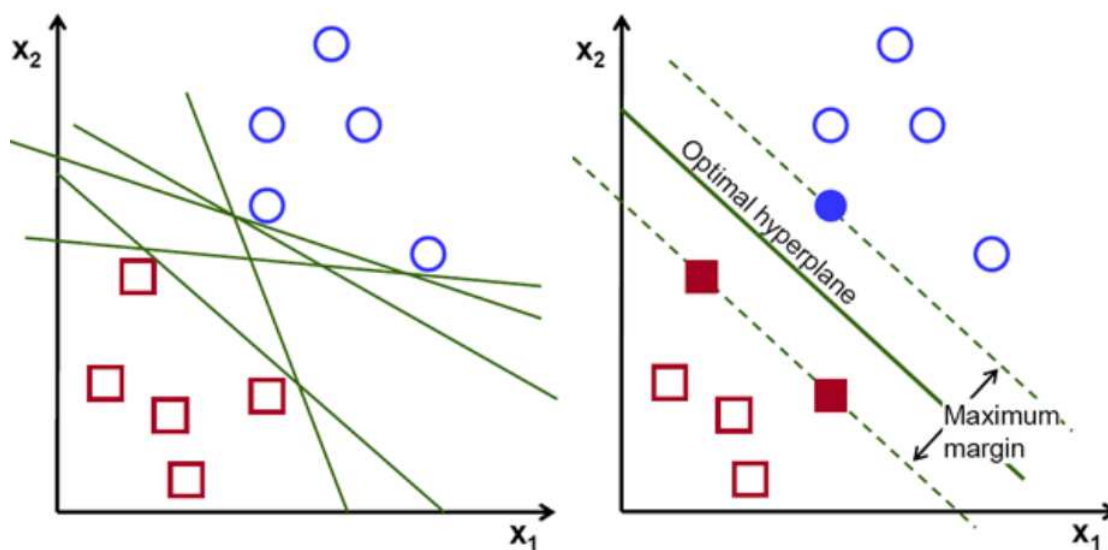
1. Euklidska ($p=2$) - tada je formula (4.18) oblika: $d_2(x_1, x_2) = \sqrt{\sum_{j=1}^m (x_{1j} - x_{2j})^2}$
2. Manhattan ($p=1$) - tada je formula (4.18) oblika: $d_1(x_1, x_2) = \sum_{j=1}^m |x_{1j} - x_{2j}|$

Pošto $k - NN$ metoda računa udaljenosti između primjera tako što izračuna udaljenosti za svaku značajku važno je skalirati sve značajke prije provođenja algoritma. U suprotnom bi mogla postojati jedna značajka za koju se udaljenosti između primjera razlikuju toliko da će to možda dovesti do krive prosudbe (iako su primjeri na ostalim značajkama slični). Odnosno jedan faktor $(x_{1z} - x_{2z})^2$ u sumi iz formule (4.18) može biti jako velik pa će dva primjera biti 'daleko' čak i ako se razlikuju u jednoj značajki. Također, možemo primijetiti i da su vrijednosti koje nedostaju problematične, tj. ne možemo izračunati udaljenost značajki dvaju primjera ako jednu ne znamo. U takvim situacijama možemo ili zanemariti značajku (odnosno računati bez nje) ili za vrijednost značajke uzeti neku vrijednost (primjerice aritmetičku sredinu te značajke kod ostalih primjera). Važno je znati i koji k upotrijebiti, odnosno koliko najbližih susjeda uzeti u obzir. Što je k veći granice

su 'glade', odnosno manja je mogućnost overfittinga. S druge strane, veći k znači da će se za klasifikaciju trebati koristiti udaljeniji primjeri (gubi se smisao lokalnosti), a samim time će i algoritam biti sporiji. Prednosti algoritma su jednostavnost i adaptivnost (jer se koriste samo lokalne informacije), dok su mane memorija (treba čuvati puno udaljenosti) te dugotrajnost klasifikacije (za svaki primjer moramo isponova računati udaljenosti).

4.8 Metoda potpornih vektora

Metoda potpornih vektora (eng. *support vector machine*, skraćeno SVM) je metoda razvijena 1990-ih (upravo za binarnu klasifikaciju, danas se može koristiti i za regresijske probleme) i od tada joj popularnost samo raste. Cilj metode je pronaći hiper-ravninu koja ima najveću marginu razdvajanja primjera različitih klasa. Margina je udaljenost između dviju najbližih točaka suprotnih klasa (one koje se nalaze blizu granice odluke). Primijetimo da je metoda vrlo intuitivna - ako je veća margina, veća je udaljenost između primjera različitih klasa. To znači da su točke obje klase udaljenije od granice odluke, pa je samim time lakše klasificirati nove primjere. Slika 4.3 (iz [33]) najbolje prikazuje kako metoda funkcionira.



Slika 4.3: Razlika između granica odluke i optimalne granice odluke za SVM

Lijeva slika prikazuje neke mogućnosti za granicu odluke između klasa, dok desna slika prikazuje optimalnu granicu odluke (kada je margina maksimalna). Iscrtkane pravce na desnoj slici nazivamo potpornim vektorima. Usporedbom slika vidimo da određivanjem

najveće margine dobivamo najveće udaljenosti primjera obje klase od granice odluke (sa svake strane je udaljenost do granice jednaka). To nam omogućava puno 'lakšu' klasifikaciju novih primjera (a i precizniju). Primjerice, da smo za granicu odluke odabrali najstrijmiji pravac (s lijeve slike) dobili bismo puno lošiji model. Odnosno, primjer koji bi na slici bio odmah pokraj najdesnijeg 'crvenog kvadrata' bio bi klasificiran kao 'plavi krug' (jer bi bio s druge strane granice), ali taj primjer bi očito imao karakteristike sličnije 'crvenim kvadratima' (kada bismo gledali najbliže susjede, svi bi bili crveni kvadrati).

No što ako problem nije linearno separabilan? Princip je ustvari isti, odnosno cilj je naći plohu koja što bolje razdvaja klase, dok se primjeri koji su sa suprotne strane **potpornog vektora** (ne granice odluke) penaliziraju (mekane margine). Da bismo matematički zapisali metodu koristit ćemo sljedeće oznake:

1. w - normala na hiperravninu razdvajanja (granicu odluke)
2. x_i - i -ti primjer
3. y_i - oznaka klase i -tog primjera ($y_i \in \{-1, 1\}$)
4. b - slobodni član (u dvodimenzionalnom koordinatnom sustavu predstavlja točku u kojoj pravac sječe os ordinatu)
5. klasifikator je određen funkcijom $f(x) = \text{sign}(w^T x_i + b)$
6. iz funkcije lako dobijemo marginu x_i , koja je određena s $y_i \cdot (w^T x_i + b)$

Ako pretpostavimo da se sve točke nalaze najmanje na udaljenosti $dist$ od granice odluke tada za svaki primjer $e_i = (x_i, y_i)$ iz skupa za učenje vrijedi:

1. $w^T x_i + b \geq dist$, ako je $y_i = 1$
2. $w^T x_i + b \leq -dist$, ako je $y_i = -1$
3. jednakosti u oba slučaja vrijedi za točke na marginama (potporne vektore)
4. udaljenost između nekog primjera i granice odluke je $r = \frac{w^T x_i + b}{\|w\|}$
5. iz prethodne točke onda znamo da za marginu vrijedi $\rho = \frac{2 \cdot dist}{\|w\|}$ (cilj metode je maksimizirati ρ). Problem se rješava korištenjem Lagrangeovih multiplikatora i prevođenjem u dualni problem.

Za probleme koje nije moguće linearno separirati ni uz mekane margine koristimo nelinearni SVM. Nelinearni SVM koristimo tako da mapiramo podatke u još više-dimenzionalni prostor (u kojem će problem biti linearno rješiv). U tom više-dimenzionalnom prostoru će funkcija f biti oblika $f(x) = w^T x \cdot \phi(x) + b$. Problem je ako je taj novi prostor puno većih

dimenzija. Tada koristimo kernel trik. Kernel trikom izbjegavamo eksplicitno mapiranje podataka koje je potrebno prilikom 'prijelaza' u više-dimenzionalni prostor (kernel funkcija korespondira unutarnjem produktu u 'ekspandiranom' prostoru varijabli). Kerneli koje smo u eksperimentalnom dijelu uzeli u obzir su:

1. klasičan linearni: $K(x_i, x_j) = x_i^T x_j$
2. polinomni: $K(x_i, x_j) = (\gamma \cdot x_i^T x_j + r)^d$
3. rbf: $K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
4. sigmoid: $K(x_i, x_j) = \tanh(\gamma \cdot x_i^T x_j + r)$

4.9 Glasanje

Zadnja klasifikacijska metoda koju promatramo je klasifikacija *glasanjem*. Glasanje je vrlo učinkovita ansambl metoda jer koristi rezultate od više različitih klasifikatora (primjerice možemo koristiti slučajne šume, XGB i SVM). Dakle, pretpostavimo da imamo K istreniranih klasifikatora $\{h_1, h_2, \dots, h_K\}$ i neki dosad neviđeni primjer x . Cilj glasanja je agregirati rezultate svih klasifikatora i na temelju toga primjeru x dodijeliti oznaku neke klase. Najpopularnija metoda klasifikacije glasanjem je *glasanje većinom*. Neku vrstu glasanja većinom smo već susreli kod drugih algoritama strojnog učenja. Primjerice, kod metode najbližih susjeda (novi primjer dobiva oznaku klase koja je najzastupljenija među k najbližih susjeda) ili kod bagging ansambla (konačna predikcija je najbliže cijelo aritmetičke sredine predikcija svakog stabla treniranog na svojoj bootstrap replici skupa podataka). Upravo na isti način funkcionira i glasanje većinom, odnosno svaki od klasifikatora predloži oznaku klase koja bi trebala biti pridijeljena novom primjeru x , te se na temelju većine odredi kako će x biti klasificiran. Druga metoda klasifikacije glasanjem je *težinsko glasanje*, u kojem svakom klasifikatoru dodjeljujemo 'težinu glasa'. Tu metodu koristimo ako želimo nekom klasifikatoru pridati veću važnost (obično se kvalitetnijim klasifikatorima, npr. XGB-u daje veća težina u takvom procesu). Treća metoda koju ćemo spomenuti (i koju koristimo u eksperimentalnom djelu) naziva se *mekanim glasanjem*. Mekano glasanje funkcionira isto kao i glasanje većinom, jedina razlika je što svaki klasifikator umjesto da predloži oznaku klase za neki primjer x , predlaže kolika je vjerojatnost da x poprimi neku oznaku. Primjerice, ako je problem binarne klasifikacije kod glasanja većinom bi za svaki h_i vrijedili $h_i \in \{0, 1\}$, dok kod mekanog glasanja vrijedi $h_i \in [0, 1]$.

Metodama glasanja možemo dobiti jako dobre rezultate (jer možemo 'miješati' puno vrsta klasifikatora), no problem je što je potrebno posebno trenirati svaki klasifikator. Zbog toga tu metodu koristimo samo kod skupova podataka koji su dobiveni poduzorkovanjem.

Poglavlje 5

Eksperimentalna procjena

5.1 Korištene metode

U ovom poglavlju donosimo kakve smo rezultate dobili koristeći razne metode strojnog učenja (opisane u prethodnim poglavljima). Kao što već spomenuto, eksperiment je izvršen na skupu podataka za prepoznavanje kartičnih prevara. Poglavlje će biti podijeljeno u 4 dijela ovisno o metodama ponovnog uzorkovanja koja su upotrebljena prilikom predobrade podataka u skupu za učenje:

1. Učenje na originalnom skupu podataka - u ovom odlomku prikazujemo rezultate koje smo dobili kada nismo koristili nikakvu metodu za predobradu podataka, odnosno kada učimo na neuravnoteženom skupu podataka kakav je dobiven u datoteci.
2. Učenje na poduzorkovanom skupu podataka - u ovom odlomku prikazujemo rezultate koje smo dobili kada smo za predobradu koristili neku od sljedećih metoda poduzorkovanja:
 - a) Slučajno poduzorkovanje
 - b) Slučajno poduzorkovanje - glasanje
3. Učenje na preuzorkovanom skupu podataka - u ovom odlomku prikazujemo rezultate koje smo dobili kada smo za predobradu koristili neku od sljedećih metoda preuzorkovanja:
 - a) Slučajno preuzorkovanje
 - b) SMOTE
 - c) Borderline-SMOTE
 - d) ADASYN

4. Učenje na skupu podataka na kojem su izvršene barem 2 metode ponovnog uzorkovanja - u ovom odlomku prikazujemo rezultate koje smo dobili kada smo za predobradu koristili neku od sljedećih kombinacija metode za ponovno uzorkovanje:
- Slučajno preuzorkovanje + slučajno poduzorkovanje
 - SMOTE + slučajno poduzorkovanje
 - SMOTE + Tomekove veze
 - SMOTE + ENN
 - Jednostrani izbor (CNN + Tomekove veze)
 - Čišćenje susjedstva (CNN + ENN)
 - Tomekove veze + slučajno poduzorkovanje

Za eksperiment koristimo sve klasifikatore opisane u poglavlju 4. Pritom za sve metode koristimo samo sljedeća 3 klasifikatora: logistička regresija, slučajne šume i XGB, dok ostale koristimo samo za neke od navedenih metoda.

5.2 Učenje na originalnom skupu podataka

Rezultate dobivene eksperimentom prikazujemo tablično (najbolji rezultati su podcrtani). Klasifikatori su označeni kraticama - LR (logistička regresija), DT (stablo odluke), BAG (bagging na stablima odluke), RF (slučajne šume), 5-NN (5 najbližih susjeda), SVC (metoda potpornih vektora), XGB (extreme gradient boosting), LGBM (Light Gradient Boosting Machine), CAT (categorical boosting), ADA (adaptive boosting).

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	86.15%	57.14%	68.71%	0.976	0.761
DT	75.73%	79.59%	77.61%	0.898	0.777
BAG	95.06%	78.57%	86.03%	0.933	0.861
RF	97.47%	78.57%	87.01%	0.953	0.875
5-NN	93.83%	77.55%	84.92%	0.934	0.884
SVC	96.83%	62.24%	75.78%	0.811	0.796
XGB	93.02%	<u>81.63%</u>	86.96%	0.975	0.869
LGBM	40.82	61.22%	48.98%	0.768	0.463
CAT	<u>97.5%</u>	79.59%	<u>87.64%</u>	<u>0.989</u>	<u>0.887</u>
ADA	<u>87.72%</u>	73.47%	<u>78.26%</u>	<u>0.983</u>	<u>0.783</u>

Tablica 5.1: Bez ponovnog uzorkovanja

Iz tablice vidimo da CatBoost postiže najbolje rezultate u 4 od 5 kategorija (kategorije doduše ovise jedna o drugoj - sjetimo se odlomka 1.2). Jedina kategorija u kojoj CatBoost ne postiže najbolje rezultate je *osjetljivost* (tu je najbolji bio XGB), koja je možda i najvažnija obzirom da nam daje podatke o tome koliko je prevara model ustvari prepoznao. U skupu podataka imamo ukupno 98 prevara, od čega je XGB ispravno prepoznao 80, a CatBoost 78. Također, ako promotrimo valjane transakcije možemo uočiti da je XGB pogrešno klasificirao 6, a CatBoost je pogrešno klasificirao 2 valjane transakcije (klasificirane su kao prevare). Uzevši to u obzir, možemo čak i reći da je XGB postigao bolje rezultate. Da bismo to objasnili situaciju možemo promatrati iz perspektive inspektora. U slučaju da koristimo XGB uhvatit ćemo 80 prevaranta (2 više nego što bismo uhvatili koristeći CatBoost), dok ćemo detaljnije morati istražiti 6 slučajeva (4 više nego s CatBoostom). Preispitivanje 4 transakcije više je 'niska cijena' (ručno se može provjeriti u jako kratkom vremenu), ako tako možemo uhvatiti više prevaranata. Primijetimo veliku pristranost klasifikatora prema negativnoj klasi o kojoj smo i govorili kroz ovaj rad. Čak i kod XGB-a (najveća osjetljivost) nismo uspjeli prepoznati 18 prevara (prepoznali smo 80 od 98), dok s druge strane nismo uspjeli prepoznati samo 6 valjanih transakcija (njih je ukupno preko 57000). Klasificiranje korištenjem slučajnih šuma također je pokazalo dobre rezultate, no vrijeme potrebno za učenje je bilo dvostruko duže nego kod boosting metoda (XGB, CatBoost).

5.3 Učenje na poduzorkovanom skupu podataka

Slučajno poduzorkovanje

Kao i u prethodnom odlomku, rezultate prikazujemo koristeći tablicu.

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
DT	1.51%	<u>93.88%</u>	2.97%	0.917	0.477
BAG	4.04%	90.82%	7.74%	0.98	0.567
5-NN	5.47%	90.82%	10.31%	0.964	0.635
SVC	<u>10.03%</u>	88.78%	<u>18.03%</u>	0.984	0.677
LGBM	4.98%	92.86%	9.46%	0.981	<u>0.714</u>
CAT	5.88%	91.84%	11.06%	<u>0.985</u>	0.674
ADA	2.93%	91.84%	5.68%	0.976	0.678

Tablica 5.2: Slučajno poduzorkovanje

Slučajno poduzorkovanje je provedeno tako da omjer pozitivnih i negativnih primjera (omjer neuravnoteženosti) bude 1 : 1, dok smo za logističku regresiju, slučajne šume i XGB proveli dodatne 2 metode slučajnog poduzorkovanja: takve da dobijemo omjere neuravnoteže 1 : 2 te 1 : 3, odnosno pozitivni primjeri čine trećinu odnosno četvrtinu podataka

u skupu za učenje. Zbog preglednosti ćemo rezultate prikazati u 2 tablice: u prvoj će biti svi klasifikatori s omjerom neuravnoteženosti 1 : 1, u drugoj će biti oni klasifikatori za koje smo upotrijebili i druge omjere neravnoteže.

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	1 : 1	6.63%	92.86%	12.37%	0.982	0.704
	1 : 2	6.77%	91.84%	12.61%	0.982	0.717
	1 : 3	13.0%	89.8%	22.71%	<u>0.986</u>	0.71
RF	1 : 1	4.74%	<u>93.88%</u>	9.03%	0.982	0.79
	1 : 2	11.06%	89.8%	19.69%	0.977	0.78
	1 : 3	<u>17.16%</u>	88.78%	<u>28.76%</u>	0.978	<u>0.804</u>
XGB	1 : 1	5.19%	92.86%	9.83%	0.978	0.78
	1 : 2	10.37%	90.82%	18.62%	0.981	0.803
	1 : 3	14.15%	90.82%	24.48%	0.985	0.742

Tablica 5.3: Slučajno poduzorkovanje

Iz tablica 5.2 i 5.3 možemo vidjeti da su preciznost i f1-mjera puno manje u odnosu na tablicu 5.1, no osjetljivost je zato viša. Razlog za drastičan pad preciznosti (a samim time i f1-mjere) je gubitak informacija. Bolje rečeno, zato što je puno primjera negativne klase uklonjeno su izgubljene važne informacije koje bi pridonijele boljoj klasifikaciji primjera negativne klase. Također, u tablici 5.3 možemo uočiti da slično vrijedi i za različite omjere neravnoteže. Odnosno, kada su klase savršeno balansirane (1 : 1) dobivamo najveću osjetljivost, ali i najmanju preciznost. Uzmimo za primjer XGB (jer smo ga spomenuli i u prošlom odlomku) te promotrimo matricu konfuzije. U slučaju kada je omjer neravnoteže 1 : 1, model ispravno klasificira čak 91 prevaru, ali i pogrešno klasificira oko 1700 valjanih transakcija. Ako uzmemo omjer neravnoteže 1 : 3 model ispravno klasificira 89 prevara, dok pogrešno klasificira približno 540 valjanih transakcija. Usporedbe radi, bez ponovnog uzorkovanja je XGB ispravno klasificirao 80 prevara, ali je pogrešno klasificirao samo 6 valjanih transakcija. Kada gledamo iz perspektive inspektora, naravno da je bolje uhvatiti 10% prevara više, ali pitanje je koliko je ljudi potrebno da bi provjerili 540 ili čak 1700 pogrešno klasificiranih valjanih transakcija (klasificirane su kao prevare) i uvjerilo se da su to ustvari valjane transakcije. Također inspektori se trebaju odlučiti koliko poduzorkovanja misle da je optimalno, konkretno u ovoj situaciji: ima li smisla (i resursa) provjeravati 1100 transakcija više (razlika između slučajeva kada uzimamo omjere 1 : 1 i 1 : 3) da bismo uhvatili 2 prevare više.

Slučajno poduzorkovanje - glasanje

U ovom odlomku smo odlučili podesiti parametre kod modela kako bismo pokušali dobiti bolje rezultate. Također smo kombinirali te dobivene modele da bismo dobili različite mekane glasačke klasifikatore (označeni s VOT). Rezultati koje smo dobili koristeći modele s podešenim parametrima nisu prikazani (moguće ih je pronaći u jupyter bilježnici).

Model	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
VOT1	7.61%	91.84%	14.06%	0.986	0.783
VOT2	7.4%	91.84%	13.69%	0.985	<u>0.792</u>
VOT3	7.41%	91.84%	13.72%	0.986	0.782
VOT4	7.74%	92.86%	14.3%	0.985	0.76
VOT5	6.46%	<u>93.88%</u>	12.08%	0.98	0.746
VOT6	3.17%	<u>89.8%</u>	6.12%	0.98	0.768
VOT7	6.34%	90.82%	11.85%	0.982	0.787
VOT8	8.33%	90.82%	15.27%	0.98	0.78
VOT9	8.43%	91.84%	15.44%	0.979	0.752
VOT10	<u>8.8%</u>	90.82%	<u>16.05%</u>	0.983	0.764
VOT11	<u>5.99%</u>	93.88%	11.27%	<u>0.986</u>	0.773

Tablica 5.4: Slučajno poduzorkovanje - glasanje

Koristili smo 11 različitih glasačkih klasifikatora. Oni su prikazani sljedećom listom:

1. VOT1 - LR, XGB, RF
2. VOT2 - LR, XGB, RF, SVC
3. VOT3 - LR, XGB, RF, SVC, KNN
4. VOT4 - LR, LGBM, RF
5. VOT5 - LGBM, RF
6. VOT6 - BAG, DT
7. VOT7 - BAG, KNN
8. VOT8 - RF, KNN
9. VOT9 - RF, KNN, CAT
10. VOT10 - RF, SVC, ADA

11. VOT11 - XGB, LGBM, CAT, ADA

Kao što vidimo kombinacijom klasifikatora smo uglavnom dobili visoku osjetljivost kod svih glasačkih modela (preko 90% kod svih). Valjalo bi napomenuti da je eksperiment proveden više puta te da su svaki put rezultati bili drugačiji. Razlog tome je što slučajno poduzorkovanje nekad izbaci iz skupa za učenje 'nevažnije' podatke, a nekad važnije - tada model ne može dovoljno dobro naučiti negativnu klasu.

5.4 Učenje na preuzorkovanom skupu podataka

U ovom odlomku prikazujemo kakve smo rezultate dobili koristeći razne metode preuzorkovanja pozitivne klase (prevara). Pošto je skup podataka jako velik fokusirali smo se na 3 modela: logistička regresija, slučajne šume i XGB. Ta 3 modela smo odabrali iz sljedećih razloga:

1. XGB - jer smo dobili najbolje rezultate na poduzorkovanom skupu podataka
2. slučajne šume - jer smo također dobili jako dobre rezultate (najbolje od nekog modela koji nije boosting ansambl)
3. logistička regresija - model je jednostavan pa je učenje kratko, odnosno model je dobar za testiranje same metode preuzorkovanja

Napomenimo da za učenje nije bilo korišteno ništa osim lokalnih resursa, odnosno nije korišten vanjski GPU ili bilo kakav drugi hardver što bi vjerojatno drastično ubrzalo proces učenja.

Slučajno preuzorkovanje

Prvo predstavljamo rezultate koje smo dobili kada smo za preuzorkovanje koristili metodu slučajnog preuzorkovanja. Sjetimo se da ta metoda ne unosi nikakve dodatne informacije, ali da uklanja pristranost klasifikatora prema negativnoj klasi (valjane transakcije).

Iz tablice 5.5 možemo primijetiti da smo slučajnim preuzorkovanjem dobili puno bolje rezultate nego slučajnim poduzorkovanjem. Razlog za to je što praktično koristimo originalni skup podataka (samo su prevare višestruko replicirane). Cijena tih boljih rezultata je vrijeme potrebno za učenje. Primjerice, za učenje korištenjem slučajnih šuma je bilo potrebno preko 2 sata, dok bi na poduzorkovanom skupu podataka trebalo par sekundi. S druge strane, XGB je također postigao značajno bolje rezultate (u odnosu na tablicu 5.3), te je za to bilo potrebno između 20 i 30 minuta (sjetimo se (odjeljak 4.6) da smo kao jednu prednost XGB-a spomenuli upravo brzinu). Učenje logističkom regresijom nije donijelo jednako dobre rezultate kao slučajne šume ili XGB, ali je bilo jako brzo (otprilike 2 min),

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	1 : 1	6.12%	91.84%	11.48%	0.98	0.747
	1 : 2	12.61%	90.82%	22.14%	0.98	0.752
	1 : 3	17.25%	89.8%	28.95%	0.981	0.753
RF	1 : 1	97.5%	79.59%	87.64%	0.963	<u>0.89</u>
	1 : 2	<u>98.72%</u>	79.59%	88.14%	0.953	0.884
	1 : 3	97.56%	81.63%	<u>88.89%</u>	0.953	0.885
XGB	1 : 1	22.9%	<u>91.84%</u>	36.66%	<u>0.981</u>	0.823
	1 : 2	41.9%	89.8%	57.14%	0.978	0.828
	1 : 3	48.35%	89.8%	62.86%	0.978	0.801

Tablica 5.5: Slučajno preuzorkovanje

te je također pokazalo bolje rezultate nego koje smo dobili kada smo trenirali model na poduzorkovanom skupu podataka (a i nije bilo puno sporije).

SMOTE

U ovom dijelu predstavljamo rezultate dobivene na skupu preuzorkovanom korištenjem standardne SMOTE metode (kreiranje sintetskih primjera na dužinama koje spajaju pozitivan primjer i njegove pozitivne susjede).

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	1 : 1	5.82%	91.84%	10.94%	0.98	0.747
	1 : 2	12.15%	<u>91.84%</u>	21.45%	0.98	0.753
	1 : 3	16.14%	<u>88.78%</u>	27.32%	0.98	0.753
RF	1 : 1	88.17%	83.67%	85.86%	0.978	0.878
	1 : 2	90.11%	83.67%	86.77%	0.978	0.886
	1 : 3	<u>92.31%</u>	85.71%	<u>88.89%</u>	<u>0.995</u>	<u>0.887</u>
XGB	1 : 1	13.33%	89.8%	23.22%	0.985	0.775
	1 : 2	23.26%	88.78%	36.86%	0.986	0.813
	1 : 3	34.25%	88.78%	49.43%	0.987	0.821

Tablica 5.6: SMOTE

Zanimljivo je što SMOTE-om ni za logističku regresiju ni za XGB nismo dobili bolje rezultate u odnosu na slučajno poduzorkovanje (za XGB smo ustvari dobili očigledno lošije). No za slučajne šume smo možda dobili i najbolji model dosad (kod omjera neravnoteže 1 : 3). Taj model je ispravno klasificirao 84 od 98 prevara, i pritom je pogrešno klasificirao svega 7 valjanih transakcija, odnosno za samo 7 transakcija se potrebno ručno uvjeriti da

su ustvari valjane. Ako usporedimo model s onim iz tablice 5.5 možemo uočiti da ovaj ima veću osjetljivost, a manju preciznost (uz istu f1-mjeru). To je velika prednost jer ispravno prepoznamo čak 4 prevare više (uz 5 false positivesa više). Također površine ispod ROC-krivulje i PR-krivulje su veće kad za učenje koristimo preuzorkovanje SMOTE-om.

Podošavanjem parametara smo uspjeli dobiti rezultate prikazane tablicom 5.7.

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
RF	1 : 1	81.73%	86.73%	84.16%	<u>0.986</u>	<u>0.882</u>
	1 : 2	<u>84.69%</u>	84.69%	<u>84.69%</u>	0.967	0.854
	1 : 3	84.54%	83.67%	84.1%	0.957	0.868
XGB	1 : 1	13.41%	89.8%	23.34%	0.983	0.779
	1 : 2	25.88%	89.8%	40.18%	0.981	0.842
	1 : 3	43.56%	<u>89.8%</u>	58.67%	0.986	0.853

Tablica 5.7: SMOTE - parametri

Kod XGB-a posebno vidimo napredak rezultata i u preciznosti i u osjetljivosti.

Borderline-SMOTE

U ovom pododlomku prikazujemo rezultate koje smo dobili kada smo koristeći SMOTE metodu preuzorkovali samo one pozitivne primjere koji se nalaze blizu granice odluke.

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	1 : 1	11.23%	82.65%	19.78%	0.911	0.707
	1 : 2	14.31%	82.65%	24.44%	0.916	0.712
	1 : 3	16.6%	82.65%	27.65%	0.921	0.72
RF	1 : 1	93.1%	82.65%	87.57%	0.947	0.874
	1 : 2	<u>94.19%</u>	82.65%	<u>88.04%</u>	0.963	<u>0.882</u>
	1 : 3	<u>94.19%</u>	82.65%	<u>88.04%</u>	0.958	0.88
XGB	1 : 1	35.8%	88.78%	51.03%	0.968	0.815
	1 : 2	42.65%	<u>88.78%</u>	57.62%	0.965	0.784
	1 : 3	47.25%	87.76%	61.43%	<u>0.969</u>	0.832

Tablica 5.8: Borderline-SMOTE

Iz tablice 5.8 možemo uočiti da smo koristeći Borderline-SMOTE za preuzorkovanje dobili nešto lošije rezultate nego kada smo koristili SMOTE. Razlog za to može biti upravo 'velika količina' preuzorkovanja oko granice odluke. Odnosno, moguće je da se sintetski primjeri (pošto ih se kreira puno) stvaraju 'bliže' valjanim transakcijama nego prevarama.

Tada je modelu puno teže naučiti granicu odluke (jer su pozitivni i negativni primjeri oko nje izmiješani). No u tablici 5.8 možemo primijetiti još jednu zanimljivost. Vidimo da kada 'smanjimo' količinu sintetskih primjera koje treba stvoriti preuzorkovanjem dobivamo bolju preciznost, dok osjetljivost ostaje ista. Upravo to smo u početku i spomenuli kao cilj učenja na neuravnoteženim skupovima podataka (povećati preciznost bez da pogoršamo osjetljivost ili obratno). Također, pošto logistička regresija i slučajne šume imaju potpuno jednaku osjetljivost (82.65%) zbog goleme razlike u preciznosti (kod slučajnih šuma je 80% veća) možemo reći da su u ovom slučaju slučajne šume značajno bolji klasifikator.

ADASYN

Posljednja tehnika preuzorkovanja koju smo koristili je ADASYN. Sjetimo se da ADASYN preuzorkuje na temelju 'težih' i 'lakših' primjera za klasificiranje. Teži primjeri su oni u blizini granice odluke i njih se preuzorkuje više (da bi bolje naučili razliku), dok su lakši primjeri oni koje je prilično lako klasificirati.

Model	$P : N$	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	1 : 1	1.76%	94.9%	3.45%	0.978	0.739
	1 : 2	3.21%	93.88%	6.21%	0.979	0.743
	1 : 3	4.54%	91.84%	8.65%	0.979	0.744
RF	1 : 1	89.01%	82.65%	85.71%	0.989	0.868
	1 : 2	90.22%	84.69%	87.37%	0.979	0.873
	1 : 3	87.91%	81.63%	84.66%	0.984	0.876
XGB	1 : 1	4.29%	91.84%	8.2%	0.986	0.78
	1 : 2	7.3%	89.8%	13.5%	0.986	0.807
	1 : 3	9.93%	89.8%	17.89%	0.987	0.787

Tablica 5.9: ADASYN

Koristeći ADASYN za preuzorkovanje smo dobili nešto drugačije rezultate nego što bismo dobili drugim tehnikama. Odnosno, preciznost (pa tako i f1-mjera) su bili dosta niži nego kod drugih tehnika preuzorkovanja (posebno kod logističke regresije i XGB-a). Kod logističke regresije je osjetljivost bila jako visoka, ali zbog niske preciznosti ne možemo taj model smatrati 'dobrim'. Za slučajne šume smo dobili jako dobre rezultate za slučaj kada je omjer neravnoteže 1 : 2.

5.5 Učenje na skupovima podataka dobivenim nakon više tehnika ponovnog uzorkovanja

U ovom odjeljku predstavljamo rezultate dobivene kombiniranjem više metoda ponovnog uzorkovanja. Napomenimo da ćemo za računalno jednostavnije (brže) metode koristiti sve klasifikatore (kao i u odjeljku 5.2), dok ćemo za složenije metode koristiti 3 klasifikatora kao u odjeljku 5.4.

Kombinacija slučajnog preuzorkovanja i poduzorkovanja

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	12.38%	90.82%	21.79%	0.981	0.752
DT	46.71%	79.59%	58.87%	0.897	0.632
BAG	62.9%	79.59%	70.27%	0.912	0.733
RF	<u>89.13%</u>	83.67%	<u>86.32%</u>	0.975	<u>0.873</u>
RF-params	45.31%	88.78%	60.0%	0.981	0.804
5-NN	37.02%	88.78%	52.25%	0.943	0.751
SVC	34.5%	90.82%	50.0%	0.983	0.678
XGB	39.11%	89.8%	54.49%	0.981	0.828
LGBM	80.37	87.76%	83.9%	<u>0.989</u>	0.872
CAT	77.78%	85.71%	81.55%	0.983	0.863
ADA	14.0%	<u>92.86%</u>	24.33%	0.977	0.821

Tablica 5.10: Kombinacija slučajnog preuzorkovanja i poduzorkovanja

Iz dobivenih rezultata vidimo da kombinacijom preuzorkovanja i poduzorkovanja dobivamo generalno dobre rezultate. Prednost kombiniranja ovih metoda je sigurno brzina učenja, odnosno skup za učenje je manji nego kada smo samo koristili preuzorkovanje. Konkretno, učenje modela XGB na preuzorkovanom skupu podataka trajalo je oko 25 minuta, dok na skupu podataka dobivenim kombinacijom preuzorkovanja i poduzorkovanja traje oko 5 minuta (što je 5 puta brže). Možemo primijetiti također da je osjetljivost kod većine klasifikatora vrlo visoka, dok je i preciznost prilično visoka (primjerice LGBM). Upravo korištenjem LGBM-a ispravno klasificiramo čak 86 prevara, dok je svega 21 valjana transakcija klasificirana kao prevara.

Kombinacija SMOTE-a i slučajnog poduzorkovanja

U prethodnom pododjeljku smo vidjeli da kombinacija metoda preuzorkovanja i poduzorkovanja doprinosi dobrim rezultatima u kratkom vremenu. Ovdje, umjesto da samo replici-

ramo podatke, da bismo uklonili pristranost klasifikatora sintetski kreiramo nove primjere pozitivne klase (prevare).

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	11.72%	<u>91.84%</u>	20.79%	0.979	0.749
DT	15.75%	81.63%	26.4%	0.904	0.487
BAG	54.05%	81.63%	65.04%	0.95	0.763
RF	<u>76.99%</u>	88.78%	<u>82.46%</u>	0.987	<u>0.881</u>
RF-params	74.14%	87.76%	80.37%	0.991	0.878
5-NN	19.38%	88.78%	31.81%	0.948	0.652
SVC	28.57%	89.8%	43.35%	0.984	0.663
XGB	25.43%	89.8%	39.64%	0.987	0.832
LGBM	59.03	86.73%	70.25%	<u>0.993</u>	0.853
CAT	55.84%	87.76%	68.25%	0.979	0.873
ADA	10.66%	90.82%	19.08%	0.981	0.784

Tablica 5.11: Kombinacija SMOTE-a i slučajnog poduzorkovanja

Kada uz slučajno poduzorkovanje koristimo SMOTE umjesto slučajnog preuzorkovanja dobivamo zanimljive rezultate. Bolje rečeno, rezultati su se kod nekih klasifikatora popravili (npr. logistička regresija ili slučajne šume), a kod nekih pokvarili (npr. boosting metode). Promotrimo malo detaljnije upravo metodu slučajnih šuma (bez podešenih parametara). Taj model ispravno klasificira 87 prevara, dok svrstava svega 26 valjanih transakcija među prevare. To je vrlo dobar rezultat jer smo prepoznali gotovo 90% prevara, te pritom dobili samo 26 slučajeva koje 'ručno' trebamo osloboditi sumnje.

Kombinacija SMOTE metode i Tomekovih veza

Ovdje kao metodu ponovnog uzorkovanja koristimo kombinaciju SMOTE metode i Tomekovih veza. Pošto je učenje kada se koristi ta metoda (a i naredne metode) dosta sporije za klasifikaciju koristimo samo logističku regresiju, slučajne šume i XGB (isto kao i kod metoda preuzorkovanja).

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	44.74%	86.73%	59.03%	0.979	0.762
RF	<u>89.25%</u>	84.69%	<u>86.91%</u>	0.973	<u>0.884</u>
XGB	56.95%	<u>87.76%</u>	69.08%	<u>0.989</u>	0.84

Tablica 5.12: Kombinacija SMOTE-a i Tomekovih veza

Iz tablice 5.12 vidimo da smo dobili poprilično dobre rezultate koristeći kombinaciju SMOTE tehnike (za preuzorkovanje) i Tomekovih veza (za čišćenje podataka). Najveći problem ove metode je računalna složenost (čak i za logističku regresiju je učenje trajalo više sati). No valja napomenuti da se, uz vanjski hardver i dobro određenu količinu preuzorkovanja, mogu dobiti jako dobri rezultati.

Kombinacija SMOTE metode i uređenih najbližih susjeda

Ova metoda je, kao i prošla, računalno vrlo zahtjevna (jer zahtijeva pronalaženje susjeda na preuzorkovanom skupu podataka). Rezultate koje smo dobili ovom metodom su prikazani tablicom 5.13.

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	42.08%	86.73%	56.67%	0.979	0.765
RF	<u>82.35%</u>	85.71%	<u>84.0%</u>	<u>0.984</u>	<u>0.873</u>
XGB	<u>57.72%</u>	<u>87.76%</u>	69.64%	0.983	0.844

Tablica 5.13: Kombinacija SMOTE-a i ENN-a

Možemo primijetiti da smo, koristeći kombinaciju SMOTE-a i ENN-a (tablica 5.13), dobili slične rezultate kao i kada smo umjesto ENN-a koristili Tomekove veze (tablica 5.12). Bolje rečeno, za logističku regresiju i slučajne šume smo dobili malo slabije rezultate, dok smo za XGB dobili malo bolje rezultate. Ovu metodu, kao i prošlu, najbolje je koristiti ako je dostupna neka vrsta vanjskog hardvera (u slučaju da je skup podataka velik).

Jednostrani izbor

Ovdje prikazujemo rezultate koje smo dobili kada smo koristili jednostrani izbor, odnosno kombinaciju stisnutih najbližih susjeda (CNN) i Tomekovih veza.

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	86.36%	58.16%	65.51%	0.977	0.761
BAG	96.15%	76.53%	85.23%	0.918	0.841
RF	<u>96.3%</u>	79.59%	<u>87.15%</u>	0.953	0.874
5-NN	93.83%	77.55%	84.92%	0.934	<u>0.882</u>
XGB	93.02%	<u>81.63%</u>	86.96%	<u>0.977</u>	0.866

Tablica 5.14: Jednostrani izbor

U tablici 5.14 možemo vidjeti rezultate koje smo dobili koristeći jednostrani izbor (OSS). Najveći problem je što tom metodom očito nije uklonjeno dovoljno primjera, pa su

klasifikatori (kao i kada nismo primijenili tehnike ponovnog uzorkovanja) pristrani prema negativnoj klasi (valjanim transakcijama). To možemo vidjeti iz niske osjetljivosti (modeli prepoznaju svega oko 80% prevara).

Čišćenje susjedstva

U ovom odlomku prikazujemo rezultate koje smo dobili metodom čišćenja susjedstva (NCL), odnosno kada smo uz metodu stisnutih najbližih susjeda (CNN), umjesto Tomekovih veza, koristili metodu uređenih najbližih susjeda (ENN).

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	85.53%	66.33%	74.71%	0.977	0.767
RF	<u>90.8%</u>	80.61%	85.41%	0.953	<u>0.868</u>
XGB	90.0%	<u>82.65%</u>	<u>86.17%</u>	0.975	0.86

Tablica 5.15: Čišćenje susjedstva

Ako usporedimo rezultate iz tablice 5.14 i tablice 5.15 možemo uočiti da smo u slučaju kada smo koristili NCL dobili malo veću osjetljivost, ali smo zato izgubili malo na preciznosti. Obzirom da nam je bitnije prepoznati što više prevara možemo reći da je NCL metoda malo uspješnija.

Kombinacija Tomekovih veza i slučajnog poduzorkovanja

Posljednja metoda koju smo koristili je kombinacija dviju tehnika poduzorkovanja: Tomekovih veza i slučajnog poduzorkovanja.

Klasifikator	Preciznost	Osjetljivost	F1-mjera	AUROC	AUPRC
LR	10.77%	<u>91.84%</u>	19.27%	0.98	0.734
RF	<u>11.45%</u>	88.78%	<u>20.28%</u>	0.977	<u>0.785</u>
XGB	10.7%	91.84%	19.17%	<u>0.987</u>	0.783

Tablica 5.16: Kombinacija Tomekovih veza i slučajnog poduzorkovanja

Ovom metodom smo dobili slične rezultate kao što smo i dobili korištenjem samog slučajnog poduzorkovanja (tablica 5.3).

Poglavlje 6

Zaključak

Eksperimentom smo pokazali da kada koristimo neku metodu ponovnog uzorkovanja dobivamo dosta bolje rezultate nego što smo dobili kada smo klasifikatore trenirali na originalnom neuravnoteženom skupu podataka. Napomenimo da smo bolje performanse dobili kada smo koristili metode preuzorkovanja (konkretno varijante SMOTE tehnike) umjesto metoda poduzorkovanja. Mana metoda preuzorkovanja je što učenje traje jako dugo. Kombiniranjem više različitih metoda ponovnog uzorkovanja možemo precizno balansirati skup podataka te dobiti kvalitetne modele relativno brzo (učenje traje puno manje nego kada samo koristimo preuzorkovanje). Uočili smo da modeli koji su bazirani na ansamblima (konkretno XGB i slučajne šume) postižu najbolje performanse. Također smo eksperimentirali i s raznim omjerima neravnoteže te zaključili da omjer 1 : 1 nije nužno najbolji (to možemo vidjeti primjerice u tablici 5.6). Kao algoritam za prepoznavanje kartičnih prevara bih predložio sustav baziran na XGBoostu te SMOTE metodi preuzorkovanja. Razlog za to je što banke i ostale sigurnosne institucije imaju jak hardver što uvelike ubrzava učenje. Zbog prednosti XGBoosta (koje smo objasnili u 4.6) također možemo jednostavno unaprijediti sustav kada je to potrebno (obzirom da se i načini izvršavanja prevara isto moderniziraju). Prilikom implementacije takvog sustava važno je i razumjeti kako koristiti metodu SMOTE. Prije svega važno je dobro odabrati koliki omjer neravnoteže koji ćemo koristiti. U eksperimentu se pokazalo da XGBoost postiže najbolje rezultate kada smo omjer neravnoteže postavili na 1 : 3 (koristili smo omjere 1 : 1, 1 : 2, 1 : 3). Stoga vidimo da je moguće da bismo dobili još bolje rezultate kada bismo za omjer odabrali 1 : 4. Osim toga bitno je i razumjeti podatke, odnosno sličnosti između prevara i valjanih transakcija. Ako su razlike male SMOTE bi mogao sintetske primjere kreirati tako da budu sličniji (po karakteristikama) valjanim transakcijama nego prevarama. Ako primijetimo da je to slučaj dobro je kombinirati SMOTE metodu s nekom metodom za poduzorkovanje da bi 'očistili' podatke (primjerice Tomekove veze ili ENN).

Dodatak A

Dodatak

U ovom dodatku navodimo hardver, softver i dodatne pakete koje smo koristili prilikom provođenja eksperimenta.

Hardver:

1. Procesor: Intel(R) Core(TM) i7 – 8550U CPU @1.80 GHz - 8 virtualnih jezgri
2. Radna memorija: 8GB
3. L1 Cache: 256 KB
4. L2 Cache: 1 MB
5. L3 Cache: 8 MB

Softver:

1. Operativni sustav: Win10 Home
2. Okruženje: Anaconda Navigator (64bit) - dostupno na:
<https://www.anaconda.com/products/individual>
3. Programski jezik: Python 3.7 (unutar Jupyter Notebook)

Standardni paketi (dobiveni su u sklopu Anaconda Navigator okruženja):

1. **pandas** - biblioteka za rad sa strukturama podataka (učitavanje podataka iz datoteke, transformacija podataka i slično)
2. **numpy** - biblioteka koja olakšava rad s poljima (posebno višedimenzionalnim)
3. **matplotlib** - alat koji omogućava grafički prikaz podataka

4. **seaborn** - alat sa sličnom svrhom kao i matplotlib, lakše ga je koristiti za neke vrste grafova (npr. za distribucije)
5. **scikit-learn** - biblioteka koja nam omogućuje korištenje raznih klasifikacijskih algoritama te potrebne metode za treniranje i testiranje

Dodatni paketi (potrebno ih je instalirati pomoću anaconda prompta):

1. **imbalanced-learn** - biblioteka koju koristimo za rad s neuravnoteženim skupovima podataka. Instalacija: u anaconda prompt upisati:
conda install -c conda-forge imbalanced-learn
2. **py-xgboost** - biblioteka koja omogućava korištenje XGBoost-a. Instalacija: u anaconda prompt upisati:
conda install -c conda-forge py-xgboost
3. **lightgbm** - biblioteka koja omogućava korištenje LightGBM-a. Instalacija: u anaconda prompt upisati:
conda install -c conda-forge lightgbm
4. **catboost** - biblioteka koja omogućava korištenje CatBoost-a. Instalacija: u anaconda prompt upisati:
conda install -c conda-forge catboost
5. **scikit-learn** - potrebno je instalirati novu verziju (mi koristimo: 0.23.1). Instalacija: u anaconda prompt upisati: *conda update conda*, pa nakon toga:
conda install -c scikit-learn=0.23.1

U slučaju da koristimo drugi operativni sustav (Linux, macOS ili ako se radi na 32-bitnom operativnom sustavu) sve potrebne pakete možemo pronaći upisivanjem naziva paketa u anaconda prompt naredbom: npr. *anaconda search -t conda imbalanced-learn* (za biblioteku imbalanced-learn) te među ponuđenim bibliotekama potražiti kompatibilnu verziju. Napomenimo još da je detaljne podatke o svim navedenim paketima koje koristimo je moguće pronaći upisivanjem u internet tražilicu. Programski kôd, odnosno jupyter bilježnice (te razni linkovi) su javno dostupne na githubu na adresi: <https://github.com/occamsrazor81/creditCard>.

Bibliografija

- [1] R. Alencar, *Resampling strategies for imbalanced datasets*, 2017, <https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets>, (listopad 2020.).
- [2] A. Ali, S. M. Shamsuddin i A. Ralescu, *Classification with class imbalance problem: A review*, Int. J. Advance Soft Compu. Appl **7** (2015), br. 3, 176–205 (hrvatski).
- [3] E. Alpaydın, *Voting over Multiple Condensed Nearest Neighbors*, Artificial Intelligence Review **11** (1999), 115–132 (hrvatski).
- [4] G. E. A. P. A. Batista, A. C. P. L. F. Carvalho i M. C. Monard, *Applying One-Sided Selection to Unbalanced Datasets*, MICAI 2000: Advances in Artificial Intelligence **1793** (2006), 315–325 (hrvatski).
- [5] G. E. A. P. A. Batista, R. C. Prati i M. C. Monard, *A Study of the Behavior of Several Methods for Balancing machine Learning Training Data*, ACM SIGKDD Explorations Newsletter **6** (2004), br. 1, 20–29 (hrvatski).
- [6] P. Branco, L. Torgo i R. Ribeiro, *A Survey of Predictive Modelling under Imbalanced Distributions*, ACM Computing Surveys **49** (2015), 1–49, <https://arxiv.org/abs/1505.01658> (hrvatski).
- [7] L. Breiman, *Bagging predictors*, Mach. Learn. **24** (1996), 123–140 (hrvatski).
- [8] F. Carcillo, A. Dal Pozzolo, Y A. Le Borgne, O. Caelen, Y. Mazzer i G. Bontempi, *Scarff: a scalable framework for streaming credit card fraud detection with Spark*, Information fusion; Elsevier **41** (2018), 182–194 (hrvatski).
- [9] F. Carcillo, Y A. Le Borgne, O. Caelen i G. Bontempi, *Streaming active learning strategies for real-life credit card fraud detection: assessment and visualization*, International Journal of Data Science and Analytics **5** (2018), br. 4, 285–300 (hrvatski).

- [10] F. Carcillo, Y A. Le Borgne, O. Caelen, F. Oblé i G. Bontempi, *Combining Unsupervised and Supervised Learning in Credit Card Fraud Detection*, Information Sciences **1** (2019), 1–26 (hrvatski).
- [11] N. V. Chalwa, *Data Mining for Imbalanced Datasets: An Overview*, Springer, 2005, https://doi.org/10.1007/0-387-25465-X_40 (hrvatski).
- [12] N.V. Chalwa, N. Japkowicz i A. Kotcz, *Editorial: special issue on learning from imbalanced data sets*, SIGKDD Explor. Newsl. **6** (2004), br. 1, 1–6 (hrvatski).
- [13] N. V. Chawla, K. W. Bowyer, L. O. Hall i W. P. Kegelmeyer, *SMOTE: Synthetic Minority Over-sampling Technique*, Journal of Artificial Intelligence Research **16** (2002), 321–357 (hrvatski).
- [14] T. Chen i C. Guestrin, *XGBoost: A Scalable Tree Boosting System*, KDD '16: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining **1** (2016), 785–794 (hrvatski).
- [15] D. A. Cieslak, N. V. Chawla i A. Striegel, *Combating imbalance in network intrusion datasets*, 2006 IEEE International Conference on Granular Computing **1** (2006), 732–737 (hrvatski).
- [16] A. Dal Pozzolo, *Adaptive Machine Learning for Credit Card Fraud Detection*, ULB MLG, 2015 (hrvatski).
- [17] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi i G. Bontempi, *Credit card fraud detection: a realistic modeling and a novel learning strategy*, IEEE transactions on neural networks and learning systems **29** (2018), br. 8, 3784–3797 (hrvatski).
- [18] A. Dal Pozzolo, O. Caelen, G. Bontempi i Y A. Le Borgne, *Learned lessons in credit card fraud detection from a practitioner perspective*, Expert Systems with Applications **41** (2014), br. 10, 4915–4928 (hrvatski).
- [19] A. Dal Pozzolo, O. Caelen, R. A. Johnson i G. Bontempi, *Calibrating Probability with Undersampling for Unbalanced Classification*, 2015 IEEE Symposium Series on Computational Intelligence (SSCI) **1** (2015), 159–166 (hrvatski).
- [20] A. T. Elhassan, M. Aljourf, F. Al-Mohanna i M. M. Shoukri, *Classification of Imbalance Data using Tomek Link (T-Link) Combined with Random Under-sampling (RUS) as a Data Reduction Method*, Global Journal of Technology and Optimization **1** (2017), 1–11 (hrvatski).

- [21] A. Fernández, S. García, F. Herrera i N. V. Chawla, *SMOTE for Learning from Imbalanced Data: Progress and Challenges, Marking the 15-year Anniversary*, Journal of Artificial Intelligence Research **61** (2018), 863–905 (hrvatski).
- [22] Y. Freund i R.E. Schapire, *A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting*, Journal of Computer and System Sciences **55** (1997), 119–139 (hrvatski).
- [23] ———, *A Short Introduction to Boosting*, Journal of Japanese Society for Artificial Intelligence **14** (1999), 771–780 (hrvatski).
- [24] J. H. Friedman, *Stochastic gradient boosting*, Computational Statistics and Data Analysis **38** (2002), 367–378 (hrvatski).
- [25] J.H. Friedman, *Greedy Function Approximation: A Gradient Boosting Machine*, The Annals of Statistics **29** (2001), 1189–1232 (hrvatski).
- [26] J. Friedman, T. Hastie i R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2008 (hrvatski).
- [27] X. Furnkranz, *Decision-Tree Learning*, <http://www.ke.tu-darmstadt.de/lehre/archiv/ws0809/mldm/dt.pdf>, (rujan 2020.).
- [28] M. Galar, A. Fernandés, E. Barrenechea i H. B. Sola, *A Review on Ensembles for the Class Imbalance Problem: Bagging-, Boosting-, and Hybrid-Based Approaches*, IEEE Transactions on Systems Man and Cybernetics Part C (Applications and Reviews) **42** (2012), br. 4, 463–484 (hrvatski).
- [29] H Han, W.Y. Wang i Mao B.H., *Borderline-SMOTE: A New Over-Sampling Method in Imbalanced Data Sets Learning*, Advances in Intelligent Computing. ICIC **3644** (2005), 878–887 (hrvatski).
- [30] P.E. Hart, *The condensed nearest neighbor*, IEEE Transactions on Information Theory **14** (1968), 515–516 (hrvatski).
- [31] H. He, Y. Bai, E. A. Garcia i S. Li, *ADASYN: Adaptive synthetic sampling approach for imbalanced learning*, 2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence) **1** (2008), 1322–1328 (hrvatski).
- [32] H. He i Y. Ma, *Imbalanced Learning: Foundations, Algorithms, and Applications*, Springer, 2013 (hrvatski).

- [33] P. P. Ippolito, *SVM: Feature Selection and Kernels*, 2019, <https://towardsdatascience.com/svm-feature-selection-and-kernels-840781cc1a6c>, (listopad 2020.).
- [34] G. James, D. Witten, T. Hastie i R. Tibshirani, *An Introduction to Statistical Learning with Applications in R*, Springer, 2017 (hrvatski).
- [35] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye i T.Y. Liu, *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, Advances in Neural Information Processing Systems 30 (NIPS 2017) **1** (2017), 3149—3157 (hrvatski).
- [36] X. Ke, *Learning With Imbalanced Data*, <https://web.maths.unsw.edu.au/~qlegia/Learning%20with%20imbalanced%20data.pdf>, (rujan 2020.).
- [37] S. Kotsiantis, D. Kanellopoulos i P. E. Pintelas, *Handling imbalanced datasets: A review*, GESTS International Transactions on Computer Science and Engineering **30** (2006), 1–12 (hrvatski).
- [38] M. Kubat i S Matwin, *Addressing the Curse of Imbalanced Training Sets: One-Sided Selection*, In: Fisher, D.H. (ed.): Proceedings of the Fourteenth International Conference in Machine Learning **1** (1997), 179–186 (hrvatski).
- [39] M. Kuhn i K. Johnson, *Applied Predictive Modeling*, Springer, 2013 (hrvatski).
- [40] J. Laurikkala, *Improving Identification of Difficult Small Classes by Balancing Class Distribution*, AIME '01: Proceedings of the 8th Conference on AI in Medicine in Europe: Artificial Intelligence Medicine **1** (2001), 63–76 (hrvatski).
- [41] B. Lebigot, Y A. Le Borgne, L. He, F. Oblé i G. Bontempi, *Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection*, INNSBDDL 2019: Recent Advances in Big Data and Deep Learning **1** (2019), 78–88 (hrvatski).
- [42] XY. Liu, J. Wu i ZH. Zhou, *Exploratory Undersampling for Class-Imbalance Learning*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) **39** (2009), 539–550 (hrvatski).
- [43] V. Morde, *XGBoost Algorithm: Long May She Reign!*, 2019, <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be> (listopad 2020.).
- [44] A. More, *Survey of resampling techniques for improving classification performance in unbalanced datasets*, Mathematics, Computer Science - ArXiv (2016), 1–7, <https://arxiv.org/abs/1608.06048> (hrvatski).

- [45] A. R. Obispo, *Improving the Response Rate of Non-Customer Campaigns with SMOTE Algorithm*, 2017, <https://www.bbvadata.com/improving-response-rate-non-customer-campaigns-smote-algorithm/>, (listopad 2020.).
- [46] P. Pavithra i S. Babu, *Data Mining Techniques for Handling Imbalanced Datasets: A Review*, IJCSMC **2** (2019), br. 3, 1–5 (hrvatski).
- [47] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush i A. Gulin, *CatBoost: unbiased boosting with categorical features*, NIPS'18: Proceedings of the 32nd International Conference on Neural Information Processing Systems **1** (2018), 6639–6649, <https://arxiv.org/abs/1706.09516> (hrvatski).
- [48] S. J. Russell i P. Norvig, *Artificial Intelligence A Modern Approach*, Alan Apt, 1995 (hrvatski).
- [49] S. Singer, *Umjetna inteligencija - matrijali*, 2015, <http://degiorgi.math.hr/~singer/ui/>, (listopad 2020.).
- [50] S. J. Stolfo, W. Lee, A. Prodromidis i P. K. Chan, *Credit Card Fraud Detection Using Meta-Learning: Issues and Initial Results*, AAAI-97 Workshop on Fraud Detection and Risk Management **1** (1998), 1–13 (hrvatski).
- [51] Y. Sun, A. K. Wong i M.S. Kamel, *Editorial: special issue on learning from imbalanced data sets*, International Journal of Pattern Recognition and Artificial Intelligence **23** (2009), br. 4, 687–719 (hrvatski).
- [52] L. Targo, *Data Mining with R: Learning with Case Studies*, Chapman & Hall/CRC, 2016 (hrvatski).
- [53] I. Tomek, *An Experiment with the Edited Nearest-Neighbor Rule*, IEEE Transactions on Systems, Man, and Cybernetics **6** (1976), 448–452 (hrvatski).
- [54] ———, *Two Modifications of CNN*, IEEE Transactions on Systems, Man, and Cybernetics **6** (1976), 769–772 (hrvatski).
- [55] L. van der Maaten i G. Hinton, *Visualizing data using t-SNE*, Journal of Machine Learning Research **9** (2008), 2579–2605 (hrvatski).
- [56] R. Walimbe, *Handling imbalanced dataset in supervised learning using family of SMOTE algorithm*, 2017, <https://www.datasciencecentral.com/profiles/blogs/handling-imbalanced-data-sets-in-supervised-learning-using-family>, (listopad 2020.).

- [57] G.M. Weiss i F. Provost, *Learning when training data are costly: The effect of class distribution on tree induction*, Journal of Artificial Intelligence Research **19** (2003), 315–354 (hrvatski).
- [58] D. L. Wilson, *Asymptotic Properties of Nearest Neighbor Rules Using Edited Data*, IEEE Transactions on Systems, Man, and Cybernetics **3** (1972), 1–14 (hrvatski).
- [59] J. M. Zaki i Wagner. Meira Jr., *Data mining and analysis: Fundamental Concepts and Algorithms*, Cambridge University Press, 2014 (hrvatski).
- [60] ZH. Zhou, *Ensemble Methods Foundations and Algorithms*, Chapman and Hall/CRC, 2012 (hrvatski).
- [61] T. Šmuc, *Strojno učenje - materijali*, 2020, <https://web.math.pmf.unizg.hr/nastava/su/materijali/>, (listopad 2020.).

Sažetak

Potreba za prepoznavanjem kartičnih prevara u današnje vrijeme jedan je od najvećih problema s kojim se svijet susreće. Količina elektroničkih transakcija je iz godine u godinu sve veća, pa čak i tolika da je vremenski nemoguće 'ručno' provjeravati sve transakcije. Za eksperiment smo koristili skup od oko 285000 transakcija koje su prikupljene u svega 2 dana u Europi. Onda možemo samo zamisliti koliko transakcija se događa svaki dan u svijetu koristeći razne platforme. Da bi banke prepoznale prevare među svim tim transakcijama koriste razne algoritme strojnog učenja. Cilj je 'izgraditi' što precizniji model koji će prepoznavati velik broj prevara (odnosno ne želimo da model prepoznaje puno prevara, ali i da puno valjanih transakcija prepoznaje kao prevare). Problem koji se javlja u takvim situacijama je neuravnoteženost klasa, odnosno puno je više valjanih transakcija od prevara. U odjeljku 1.2 smo objasnili zašto točnost nije dobra mjera kada mjerimo performanse modela kod prepoznavanja kartičnih prevara. Također smo spomenuli metrike koje je dobro koristiti u slučaju neuravnoteženih klasa te smo pojasnili koje su prednosti tih metrika. U poglavlju 2 smo ukratko opisali izazove koje susrećemo prilikom klasifikacije na neuravnoteženim skupovima podataka te koje probleme ti izazovi predstavljaju prilikom treniranja modela. U poglavlju 3 smo se posvetili različitim metodama ponovnog uzorkovanja. Detaljno smo opisali više različitih metoda poduzorkovanja i preuzorkovanja te se osvrnuli na mane i prednosti svake od njih. Prilikom eksperimenta smo koristili razne klasifikatore (opisani u poglavlju 4). Rezultate dobivene različitim kombinacijama metoda ponovnog uzorkovanja i klasifikatora smo predstavili u poglavlju 5. Na temelju dobivenih rezultata smo donijeli zaključke (poglavlje 6). Sustav za prepoznavanje kartičnih prevara trebao bi biti dio sigurnosnog sustava kod svake banke te bi se trebao svakodnevno unaprjeđivati upravo zato što se i nove tehnike koriste za krađu. Odnosno, kada prevaranti shvate da banka ima dobar sustav za prepoznavanje prevara će i oni promijeniti svoje tehnike tako da i sustav treba aktualizirati. Obzirom na sav tak razvoj metode ponovnog uzorkovanja možda neće biti dovoljno dobre da bismo dobili zadovoljavajuće modele. Napomenimo da metode ponovnog uzorkovanja nisu jedini način za borbu s 'neravnotežom klasa'. Postoje i troškovno-osjetljivi pristupi za koje nije potrebno precizno prilagođavati omjer neravnoteže (sjetimo se da je jako teško ispravno odabrati omjer neravnoteže prilikom ponovnog uzorkovanja), ali zahtijevaju bolje razumijevanje modela i podataka.

Summary

The need to identify credit card fraud is one of the biggest challenges the world is facing nowadays. The number of electronic transactions is increasing year by year, even so much that it is basically impossible to 'manually' check all of them. For this experiment we used data set that consist of about 285000 transactions that are collected in just 2 days in Europe. Using that information we can only imagine how many transactions take place every day in the world using various platforms. Banks use many different machine learning algorithms to detect fraudulent transactions among non fraudulent ones. The goal is to 'build' a model that is precise and sensitive as possible, meaning that we don't want a model that is excellent at detecting frauds, but also recognizes many valid transactions as fraudulent. The major problem we are dealing with is the class imbalance, ie. there are many more valid transactions than fraudulent ones (in credit card fraud detection data set for each fraudulent transaction there is 578 valid ones). In section 1.2 we explained why accuracy is not good performance metric when it comes to imbalanced data sets such as credit card fraud detection data set. Then we mentioned which measures should be used instead and why are we using those other metrics. In chapter 2 we described challenges that we encounter when dealing with class imbalance and how to deal with problems that arise from those challenges during training phase. Chapter 3 is dedicated to exploring resampling methods. We present in detail description of several different undersampling and oversampling methods and which advantages or disadvantages those methods bring. For the experiment we used many different classifiers which we describe in chapter 4. The results obtained by different combinations of resampling methods and classifiers are presented in the chapter 5. Based on the obtained results, we drew conclusions (chapter 6). The credit card fraud detection system should be a part of every bank's security system and improving that system requires constant research and development (because new fraud techniques are also being developed). In other words, fraudsters realize when the bank have good anti-fraud system in place so they alter their techniques meaning that bank will have to update their system to adapt. Taking all that into consideration, resampling methods may not be good enough to stop fraudsters. We mentioned that resampling methods are not the only way to 'fight' class imbalance. We can also use cost-sensitive approaches which don't require adjusting imbalance ratio, but they demand better model and data understanding.

Životopis

Rođen sam 17.10.1996. godine u gradu Zagrebu. Godine 2003. krećem pohađati Osnovnu školu Augusta Šenoje u Zagrebu, koju s odličnim uspjehom završavam 2011. godine. Na jesen 2011. godine upisujem 11. gimnaziju u Zagrebu. Za vrijeme srednjoškolskog obrazovanja primjećujem da mi matematika ide bolje od ostalih predmeta pa na temelju toga nastupam na školskim i županijskim natjecanjima. Nakon predavanja o 'fakultetima' uviđam da se može izgraditi dobra karijera studiranjem matematike. Stoga 2015. godine upisujem preddiplomski studij Matematika (inženjerski smjer) na Prirodoslovno-matematičkom fakultetu Sveučilišta u Zagrebu. Nakon 3 godine uspješno završavam preddiplomski studij te upisujem diplomski studij Računarstvo i matematika na istom fakultetu.