Bachelorscriptie
Informatica / Informatiekunde

Radboud Universiteit

Invloed van MetaCost op imbalanced classificatie problemen

Auteur:
Lars Kuijpers
s4356314

Inhoudelijk begeleider:
Prof. dr. Tom Heskes
tomh@cs.ru.nl

Tweede lezer:
Prof. dr. Elena Marchiori
elenam@cs.ru.nl

16 Augustus 2016

Samenvatting

Class imbalance is een probleem in Data Mining dat effect heeft op de prestaties van bepaalde algoritmes bij classificatie problemen. Omdat class imbalance vaak voorkomt in datasets is het interessant om te zoeken naar een oplossing, zodat de classificatie van deze datasets betrouwbaarder kan worden gemaakt. Een van de mogelijke oplossingen voor class imbalance is het gebruik van cost-sensitive classifiers. Bij cost-sensitive classifiers geldt dat bij de classificatie sommige misclassificaties slechter zijn dan andere. Dit in tegenstelling tot de normale situatie, waar alle misclassificaties als even erg worden gezien. In dit onderzoek kijken we naar MetaCost, een bepaalde manier om classifiers cost-sensitive te maken, en het effect hiervan op drie verschillende algoritmes (C4.5, Random Forests en SVM). Uit ons onderzoek blijkt dat zowel C4.5 als Random Forests een significante verbetering kunnen krijgen door het gebruik van MetaCost, mits de graad van class imbalance groot genoeg is. Bij SVM was echter geen significant verschil te merken.

Inhoudsopgave

Hoofdstuk 1

Inleiding

In dit onderzoek kijken we naar het probleem van class imbalance in de context van Data Mining. In Data Mining zijn er zogenaamde classificatie problemen. Bij dit soort problemen is het de bedoeling dat elk item in de bijbehorende dataset in een bepaalde klasse wordt gestopt. Een moeilijkheid bij sommige classificatie problemen is een hoge graad aan class imbalance. Hierbij is, in de optimale verdeling van de items in de dataset, een (deel) van de klassen veel kleiner dan de rest. Bij veel classificatie algoritmen zorgt dit voor slechte prestaties, omdat deze algoritmes geen rekening houden met deze imbalance.

Class imbalance is een probleem dat regelmatig voorkomt bij classificatie problemen, bijvoorbeeld in de context van de medische wereld. Bij problemen in deze context wil men vaak items van een normale situatie scheiden van degene die iets bijzonders betekenen (bijvoorbeeld de aanwezigheid van een bepaalde ziekte). Hierbij komt bijna altijd de normale situatie veel vaker voor dan de bijzondere situatie, wat tot class imbalance leidt. Als we beter weten om te gaan met class imbalance zal dat bij dit soort classificatie problemen leiden tot betrouwbaardere uitkomsten.

Een manier om het probleem van class imbalance op te lossen is het gebruik van cost-sensitive classifiers. Hoewel er al onderzoek gedaan is waarbij voor specifieke problemen wordt gekeken of cost-sensitive classifiers invloed hebben op de correctheid van de uitkomsten, is dit nog nooit op een globale manier gedaan. In dit onderzoek willen we kijken hoe goed het gebruik van cost-sensitive classifiers over het algemeen het probleem van class imbalance kan oplossen, door te kijken naar het verschil tussen gewone classifiers en cost-sensitive classifiers (met behulp van MetaCost) over meerdere datasets.

Hoofdstuk 2

Voorkennis

In dit hoofdstuk zullen we een aantal dingen beschrijven die nodig zijn om het hele onderzoek te begrijpen. Hieronder vallen beschrijvingen van de algoritmes die we gaan bekijken, waaronder de manier waarop we ze cost-sensitive gaan maken (met MetaCost). Ook bevat het een uitleg over de ten-fold cross-validation methode en het gebruik van AUC scores die in dit onderzoek zullen worden toegepast.

2.1 Classificatie algoritmes

De drie algoritmes die we bekijken in dit onderzoek zijn C4.5, Random Forests en SVM. Wat een classificatie algoritme eigenlijk doet is het creëren van een model. Dit doet men door het algoritme te 'trainen' op (een gedeelte van) een dataset. Hierbij maakt het algoritme een model dat de trainingdata goed indeelt in klassen. Het idee is dan om dit model te gebruiken voor nieuwe datapunten, zodat onbekende data geclassificeerd kan worden.

2.1.1 C4.5

C4.5 is een veelgebruikt algoritme gebaseerd op het principe van Decision Trees (Quinlan 1993 [19]). Bij Decision Tree algoritmes is het model een beslisboom, die bestaat uit een aantal knopen en zijden tussen die knopen. Elke knoop heeft een soort vraag en elk bijbehorend antwoord op die vraag heeft een geassocieerde zijde naar een andere knoop. Wanneer een datapunt aan dit model wordt gegeven, wordt bij de beginknoop gekeken wat het antwoord op de vraag is van deze knoop voor dit datapunt. Afhankelijk van dat antwoord wordt een pad naar een andere knoop gevolgd, waar weer een vraag gesteld wordt. Dit gaat door totdat een blad wordt bereikt, die zal aangeven tot welke klasse het datapunt behoort. Hoe de beslisboom precies wordt gebouwd verschilt erg per implementatie.

C4.5 werkt door bij elke stap te kijken hoe de overgebleven datapunten op de huidige knoop het best gesplitst kunnen worden. Dit gebeurt door het uitrekenen van de zogeheten Information Gain voor elk attribuut van de datapunten. Uiteindelijk wordt gesplitst op het attribuut met de hoogste Gain.

2.1.2 Random Forest

Net als C4.5 is het Random Forest algoritme gebaseerd op Decision Trees (Ho 1995 [11]). Echter, bij Random Forest wordt er in plaats van één boom vele bomen gecreëerd, een zogenaamd forest. Er wordt in dit forest gekeken naar al de gegenereerde bomen en deze worden gemiddeld naar de boom die uiteindelijk voor classificatie gebruikt kan worden. Het voordeel aan Random Forest is dat het beter omgaat met de hoge variantie van Decision Tree algoritmes, omdat het simpelweg het trainen van de boom herhaalt en een 'gemiddelde' boom er uit haalt.

2.1.3 SVM

SVM is een classificatie algoritme met een simpele intuïtie er achter, maar een complexe manier van berekenen (Cortes & Vapnik 1995 [4]). Oorspronkelijk werkte het alleen op binaire classificatie problemen. Het algoritme ging als volgt: alle datapunten in de training-data geplaatst in een n-dimensionaal spectrum, waarbij n het aantal attributen van de data is. Dan wordt er een (n-1)-dimensionaal vlak gevonden, dat de datapunten van de ene klasse perfect scheidt van de andere klasse.

De oorspronkelijke versie van SVM is sindsdien meerdere malen uitgebreid. Zo is er tegenwoordig de mogelijkheid om niet-lineaire scheidingen te gebruiken (met behulp van de zogenaamde kernel trick) en ook problemen met meer dan twee klassen kunnen op verscheidene manieren worden opgelost (Schölkopf, Tsuda & Vert 2004 [20]). De implementatie die wij gebruiken in dit onderzoek is Sequential Minimal Optimization, of SMO, met een polynomiale kernel. SMO is gemiddeld sneller en schaalt beter naar grotere problemen dan de originele implementatie voor SVM, maar werkt op hetzelfde principe (Platt 1998 [18]).

2.2 MetaCost

Een mogelijke manier om classifiers cost-sensitive te maken, is het MetaCost algoritme (Domingos 1999 [5]). Het voordeel van MetaCost is dat het toepasbaar is bovenop elk ander classificatie algoritme. Dit is dus zeer geschikt voor ons onderzoek.

MetaCost werkt door de training-set nieuwe klasse labels te geven die optimaal zijn gegeven de cost matrix. Deze nieuwe labels hoeven dus niet hetzelfde te zijn als de gegeven klasse labels. Dit wordt gedaan aan de hand van een geschatte kans voor elke klasse, welke aangeeft hoe groot de kans is dat een willekeurig item uit de set tot die klasse behoord. Om deze kansen te berekenen zijn twee mogelijkheden: de classifier bepaalt zelf deze kansen, of (als dit onmogelijk of onbetrouwbaar is) maakt MetaCost gebruik van een ensemble methode om deze uit te rekenen. Hoe de classifier deze kansen berekent verschilt per algoritme. Hier zullen we niet op in gaan.

De ensemble methode van MetaCost werkt als volgt: er worden een aantal 'bootstrap samples' van de training-set gegenereert en de gegeven classifier wordt hier op gerund. Aan de hand van deze samples wordt de geschatte kans voor elke klasse berekend, die aangeeft hoe groot de kans is dat een willekeurig item uit de set tot die klasse behoord.

Aan de hand van de informatie over de klasse-kansen, samen met de informatie uit de cost matrix, worden de nieuwe labels gemaakt voor de set. Dit gebeurt door voor elk item de klasse te kiezen met het laagste 'risico'. Dit risico wordt berekend aan de hand van de volgende formule:

$R(i|x) = \sum_j P(j|x)C(i,j)$

Hierbij is het risico om een item x te classificeren als klasse i gelijk aan de som over alle andere klassen van de kans op die klasse j (die we kregen uit de eerste runs van de classifier) maal de cost om j te misclassificeren als i (die uit de cost matrix komt). Aan de hand van deze risico waardes worden daarna de items in de oorspronkelijke set opnieuw gelabeld. Tot slot wordt dan de classifier opnieuw gerund op deze set met de nieuwe labels. De uitkomst hiervan is de uiteindelijke classificatie van de set.

Het gebruik van MetaCost heeft echter ook een groot nadeel. Omdat de classifier zo vaak wordt gebruikt (op de samples en weer op de uiteindelijke set), duurt het trainen met gebruik van MetaCost een stuk langer dan het trainen van de classifier op zichzelf. Dit kan een reden zijn om MetaCost niet te gebruiken. Uitzondering hierop is als de kansen van elke klasse op de dataset al bekend zijn, dan kan de set meteen opnieuw gelabeld worden en hoeft de classifier nog maar één keer te runnen. In dit onderzoek wordt dit probleem echter buiten beschouwing gelaten.

2.3 Ten-fold cross-validation

Ten-fold cross-validation (Devijver & Kittler 1982 [6]; Geisser 1993 [9]) is een veelgebruikte techniek in Data Mining die helpt met het voorkomen van overfitting van het model op de training-set. Dit maakt de schatting van hoe goed het algoritme zou presteren met soortgelijke data betrouwbaarder en minder afhankelijk van de training-set zelf (Kohavi 1995 [14]).

Bij ten-fold cross-validation wordt de dataset in 10 willekeurige stukjes verdeeld. De classificatie wordt dan 10 keer uitgevoerd, waarbij elke keer een ander stukje als test-set (de data waarop uiteindelijk de kwaliteit van het model wordt berekend) wordt gebruikt en de rest als training-set (de data waarop eerst het model getraind wordt). Het gebruiken van ten-fold cross-validation maakt het resultaat dus betrouwbaarder. Dit is nuttig voor ons onderzoek omdat we in meer algemene zin willen zien hoe goed de algoritmes zouden presteren op dit soort datasets.

2.4 ROC curves en AUC scores

ROC curves zijn grafieken die de performance van een binaire classifier (een classifier die werkt met maar 2 klassen) illustreert. De grafiek is uitgezet tussen de True-Positive Rate (TPR) en de False-Positive Rate (FPR). De TPR, ook wel sensitivity of recall genoemd, is het aantal correct geclassificeerde true-items (true positives) gedeeld door het totaal aantal items geclassificeerd als true (true positives en false negatives). De FPR, ook wel fall-out genoemd, is het aantal true-items die incorrect geclassificeerd worden als false (false positives) gedeeld door het totaal aantal items geclassificeerd als false (true negatives en false positives).

Een makkelijke manier om ROC curves te vergelijken is door te kijken naar de Area Under Curve, ook wel AUC genoemd (Fawcett 2006 [8]). De AUC geeft de proportie aan van het gedeelte van het vierkante vlak waarin de ROC curve wordt afgebeeld dat zich onder de curve bevindt. Anders gezegd geeft de AUC de kans aan dat de classifier een willekeurig gekozen true-item 'meer waar' inschat dan een willekeurig gekozen false-item. De AUC geeft dus een indicatie van hoe goed de performance van de classifier is op de gebruikte dataset.

De resultaten tussen de normale en cost-sensitive varianten van de algoritmes worden uiteindelijk vergeleken met de Hanley & McNeil methode (Hanley & McNeil 1982 [10]). Dit is een redelijk oude, maar nog steeds veelgebruikte, methode om de AUC scores van meerdere ROC curves te kunnen vergelijken en te kijken of het verschil significant is.

Hoofdstuk 3

Gerelateerd onderzoek

Er is al veel onderzoek gedaan op het gebied van class imbalance en hoe dit probleem kan worden opgelost. De meeste onderzoeken die op een globale manier naar het probleem kijken, vergelijken echter de effectiviteit van verschillende technieken om class imbalance tegen te gaan. Denk hierbij aan under- en oversampling of boosting, maar ook cost-sensitive learning (Japkowicz, 2000 [12]; Weiss 2004 [21]). Hierbij wordt echter vrijwel altijd maar één algoritme gebruikt en niet gekeken naar het verschil in effectiviteit van meerdere algoritmes.

De drie algoritmes die in dit onderzoek bekeken worden (C4.5, Random Forest en SVM) zijn gekozen op basis van andere onderzoeken. C4.5 komt voor in de meeste onderzoeken betreffend de effectiviteit van algoritmes. Het algoritme lijkt consistent gevoelig te zijn voor class imbalance (Brown & Mues, 2012 [2]). Ook diens opvolger, C5.0, dat gebaseerd is op hetzelfde principe maar enkele optimalisaties heeft, lijkt gevoelig te zijn voor class imbalance (Japkowicz, 2002 [13]). Uit ander onderzoek is ook gebleken dat de toepassing van een cost-sensitive variant van C4.5 beter kan presteren dan een niet cost-sensitive variant, maar dat dit afhankelijk is van de gebruikte dataset (Liu & Zhou, 2006 [15]).

Random Forest is gekozen omdat het in de meeste onderzoeken als een van de algoritmes wordt beschouwd die goed bestand is tegen class imbalance (Brown & Mues, 2012 [2]; Burez & Poel, 2009 [3]). Dit maakt de mogelijke toename in prestatie tussen de normale en cost-sensitive varianten van Random Forest interessant, met name in vergelijking met hoe groot deze toename is bij C4.5.

Het laatste algoritme, SVM, is gekozen vanwege de tegenstrijdigheid tussen onderzoeken. In bepaalde onderzoeken komt naar voren dat SVM ongevoelig is voor class imbalance (Japkowicz 2002 [13]), maar in andere blijkt juist dat het heel erg gevoelig is voor class imbalance (Brown & Mues, 2012 [2]). Om mogelijk meer opheldering over dit probleem te krijgen, hebben we besloten ook SVM in ons onderzoek mee te nemen.

Hoofdstuk 4

Onderzoek

4.1 Opzet

In dit onderzoek wordt een soortgelijke opzet gebruikt als in het onderzoek van Brown en Mues [2]. We gebruiken hier drie datasets, die allemaal bij een binair classificatie probleem horen. Deze datasets zijn de MAGIC Gamma Telescope dataset [1], de Default of Credit Card Clients dataset [22] en de Thoracic Surgery dataset [17]. De drie gebruikte datasets worden artificieel aangepast en opgedeeld in 5 verschillende datasets met een uiteenlopende graad aan class imbalance. Dit zijn verhoudingen van 50/50, 75/25, 90/10, 95/5 en 99/1, waarbij een verhouding van x/y staat voor een dataset die voor x% bestaat uit items uit de meerderheidsklasse en y% van de items komen uit de minderheidsklasse.

Bij de MAGIC Gamma Telescope dataset was het creëren van deze aangepaste datasets simpel, omdat deze al een meerderheid bevat aan items uit de (normaal gesproken) minderheidsklasse. Dit komt omdat deze data specifiek is uitgekozen om een meerderheid te hebben aan de interessante punten, waar die normaal gesproken erg zeldzaam zou zijn. Bij deze dataset hebben we willekeurige items van de minderheidsklasse weggelaten om de verhouding te creëren die we willen.

Omdat de Credit Card en Thoracic Surgery datasets een minderheid bevatten aan data uit de minderheidsklasse, zoals dus ook in de echte situatie het geval zou zijn, werkte deze aanpak hier niet. In plaats daarvan hebben we voor de 50/50 en 75/25 verhoudingen willekeurige items verwijderd uit de meerderheidsklasse. Voor de overige verhoudingen hebben we op eenzelfde manier gewerkt als bij de MAGIC dataset, waarbij de meerderheidsklasse dus intact bleef.

Bij de Credit Card dataset zijn alle class labels (oorspronkelijk 0 en 1) vervangen (door a en b) omdat Weka problemen had met de oorspronkelijke labels. Ook zijn de id labels (het eerste attribuut) hieruit verwijderd, zodat deze geen invloed hebben op het classificatie proces.

Op elk van deze aangepaste datasets worden dan de drie algoritmes (C4.5, Random Forest en SVM) getraind. Hierbij worden de implementaties van deze algoritmes gebruikt uit Weka versie 3.6.13 met de standaard parameters. Dit zijn respectievelijk j48, RandomForest en SMO. Bij het trainen en testen wordt gebruik gemaakt van ten-fold cross validation, zoals is uitgelegd in het hoofdstuk Voorkennis. Dit gebeurt gestratificeerd, zodat in de test-set altijd een representatief aantal items zit van beide klassen.

Elk algoritme wordt twee keer op elke dataset toegepast, een keer in de normale variant en een keer met de toepassing van de MetaCost classifier uit Weka. Voor beide datasets wordt voor de MetaCost de volgende cost matrices aangevoerd, afhankelijk van de verhouding van de dataset:

Figuur 4.1: Cost matrices

(a) 50/50

geclassificeerd als → / echte waarde ↓	a	b
a	0	1
b	1	0

(b) 75/25

	a	b
a	0	3
b	1	0

(c) 90/10

	a	b
a	0	9
b	1	0

(d) 95/5

	a	b
a	0	19
b	1	0

(e) 99/1

	a	b
a	0	3
b	1	0

Hierbij staat a voor de minderheidsklasse[1] en b voor de meerderheidsklasse[2]. De cost om een item dat eigenlijk behoort tot de minderheidsklasse te classificeren in de meerderheidsklasse is dus groter dan andersom. Deze cost is berekend aan de hand van de verhouding van de klassen in die dataset, door ze te vereenvoudigen naar de vorm 1/x.

[1] label g bij MAGIC, 1/b bij Credit, t bij Thoracic
[2] label h bij MAGIC, 0/a bij Credit, f bij Thoracic

Dit is beredeneerd aan de hand van de formule die MetaCost gebruikt voor het berekenen van het risico van een bepaalde classificatie, zoals die gegeven is in het hoofdstuk Voorkennis:

$R(i|x) = \sum_j P(j|x)C(i,j)$

Omdat we in dit onderzoek alleen gebruikmaken van binaire classificatie problemen kunnen we de sommering hier versimpelen tot de volgende twee formules (waarbij a weer de minderheidsklasse is en b de meerderheidsklasse):

$(1)R(a|x) = P(b|x)C(a,b) + P(a|x)C(a,a)$
$(2)R(b|x) = P(a|x)C(b,a) + P(b|x)C(b,b)$

En omdat bij al onze cost matrices geldt dat $C(a,a) = 0$ en $C(b,b) = 0$ kunnen we dit nog verder versimpelen tot:

$(1)R(a|x) = P(b|x)C(a,b)$
$(2)R(b|x) = P(a|x)C(b,a)$

Omdat we onze datasets hebben gekozen zodat ze precies de gegeven verhoudingen hebben, weten we ook wat de klasse kans voor beide klassen is. Bij de 75/25 is dit respectievelijk 75% en 25% voor de meerderheids- en minderheidsklasse. Als we deze kansen invullen in de formules komen we uit op het volgende:

$(1)R(a|x) = 0.75 * C(a,b)$
$(2)R(b|x) = 0.25 * C(b,a)$

Nu hebben wij ervoor gekozen om, in een volledig willekeurig geval, de risico om een item te classificeren in de minderheidsklasse even groot te maken als om het te classificeren in de meerderheidsklasse. We willen dus dat geldt dat $R(a|x) = R(b|x)$. Dit zorgt ervoor dat er in de set met de nieuwe labels geen (of minder) class imbalance is, waardoor de algoritmes beter kunnen werken op deze set. Om dit te bereiken, hebben we de costs in de cost matrix zo gekozen dat deze het verschil in de klasse kansen opheffen. In dit geval is $C(a,b) = 1$ en $C(b,b) = 3$. Hierdoor ontstaan de volgende uitkomsten:

$(1)R(a|x) = 0.75 * 1 = 0.75$
$(2)R(b|x) = 0.25 * 3 = 0.75$

Nu geldt dat $R(a|x) = R(b|x)$ en dus wordt de set zo gelabeld dat er geen (of minder) class imbalance is. We hebben voor alle andere verhoudingen deze costs op eenzelfde manier berekend.

4.2 Resultaten

De AUC scores zijn verkregen van zowel de gewone en cost-sensitive varianten van elk algoritme, op elk van de drie datasets met elke verhouding. Ook is hierna de Hanley & McNeil methode toegepast om bij de p-waarde te komen [10]. Hierbij is gebruik gemaakt van een online tool om deze berekeningen uit te voeren [16]. Het betreft hier de tweezijdige test, zodat we kunnen zien of de resultaten significant beter of slechter zijn. We willen een p-waarde als significant zien als deze kleiner is dan 0.05. Echter, omdat we zoveel vergelijkingen doen, maken we gebruik van de Bonferroni-correctie (Dunn 1961 [7]). Zo weten we zeker dat de p-waardes echt significant zijn. Omdat we in totaal 45 testen doen, wordt de uiteindelijke grens voor een significante p-waarde $0.05/45 \approx 0.001$.

De iconen tussen haakjes geven aan of de AUC score van de MetaCost variant hoger (↑), lager (↓) of gelijk (=) is aan de AUC score van de normale variant. Alle significante runs zijn dikgedrukt.

Dataset	Verhouding	Algoritme	AUC gewoon	AUC cost-sensitive (verschil)	P-waarde
MAGIC	50/50	C4.5	0.858	0.854 (↓)	0.8598
		Random Forests	0.934	0.933 (↓)	0.7532
		SVM	0.716	0.719 (↑)	0.6295
	75/25	C4.5	0.834	0.838 (↑)	0.5050
		Random Forests	0.925	0.920 (↓)	0.1949
		SVM	0.500	0.500 (=)	1
	90/10	**C4.5**	**0.773**	**0.827 (↑)**	**<0.0001**
		Random Forests	0.911	0.900 (↓)	0.0521
		SVM	0.500	0.500 (=)	1
	95/5	**C4.5**	**0.715**	**0.810 (↑)**	**<0.0001**
		Random Forests	0.900	0.890 (↓)	0.2117
		SVM	0.500	0.500 (=)	1
	99/1	**C4.5**	**0.488**	**0.715 (↑)**	**<0.0001**
		Random Forests	0.830	0.873 (↑)	0.0551
		SVM	0.500	0.500 (=)	1

Dataset	Verhouding	Algoritme	AUC gewoon	AUC cost-sensitive (verschil)	P-waarde
Credit	50/50	C4.5	0.669	0.679 (↑)	0.1270
		Random Forests	0.773	0.776 (↑)	0.5997
		SVM	0.689	0.686 (↓)	0.6426
	75/25	C4.5	0.664	0.670 (↑)	0.2361
		Random Forests	**0.764**	**0.741 (↓)**	**<0.0001**
		SVM	**0.654**	**0.623 (↓)**	**<0.0001**
	90/10	**C4.5**	**0.605**	**0.692 (↑)**	**<0.0001**
		Random Forests	0.748	0.756 (↑)	0.1824
		SVM	0.500	0.500 (=)	1
	95/5	**C4.5**	**0.618**	**0.689 (↑)**	**<0.0001**
		Random Forests	0.744	0.766 (↑)	0.0070
		SVM	0.500	0.500 (=)	1
	99/1	**C4.5**	**0.499**	**0.668 (↑)**	**<0.0001**
		Random Forests	**0.686**	**0.741 (↑)**	**<0.0001**
		SVM	0.500	0.500 (=)	1
Thoracic	50/50	C4.5	0.492	0.506 (↑)	0.8398
		Random Forests	0.576	0.582 (↑)	0.9299
		SVM	0.579	0.571 (↓)	0.9068
	75/25	C4.5	0.484	0.560 (↑)	0.1730
		Random Forests	0.686	0.679 (↓)	0.8849
		SVM	0.550	0.574 (↑)	0.6610
	90/10	C4.5	0.473	0.517 (↑)	0.4985
		Random Forests	0.532	0.577 (↑)	0.4713
		SVM	0.499	0.510 (↑)	0.8651
	95/5	C4.5	0.479	0.441 (↓)	0.6836
		Random Forests	0.649	0.562 (↓)	0.2933
		SVM	0.500	0.494 (↓)	0.9478
	99/1	C4.5	0.200	0.328 (↑)	0.5233
		Random Forests	0.605	0.641 (↑)	0.8421
		SVM	0.500	0.499 (↓)	0.6426

4.3 Discussie

Het eerste wat opvalt aan de resultaten is dat SVM erg slecht presteert op alle datasets. Op alle datasets van de verhoudingen 90/10, 95/5 en 99/1 heeft SVM in beide varianten een AUC score van rond de 0.5. Het lijkt te zijn dat SVM voor de gebruikte datasets erg slecht omgaat met de class imbalance, zowel in de normale als de MetaCost variant.

Als we kijken naar de P-waardes zijn er weinig significante runs geweest. In de Thoracic dataset zijn zelfs helemaal geen significante runs. Dit komt omdat de Thoracic set behoorlijk klein was, wat de Standard Error (SE) van de test en daarmee ook de p-waarde groter maakt (wat de runs minder significant maakt).

In de runs op de MAGIC en Credit datasets zijn duidelijkere resultaten behaald. In beide datasets is bij de verhoudingen van 90/10, 95/5 en 99/1 een significant verschil te merken tussen de normale variant van C4.5 en de MetaCost variant. Hierbij presteert de MetaCost variant significant beter dan de normale variant met de gegeven cost matrices. In de Credit dataset heeft ook Random Forest op een van de verhoudingen, namelijk 99/1, een significante verbetering in de MetaCost variant tegenover de normale variant.

Opvallend is dat er twee significante runs zijn geweest waar een verslechtering heeft plaatsgevonden. Dit zijn de runs van SVM en Random Forests op de 75/25 verhouding van de Credit dataset. Dit kan komen doordat de cost matrix die we hebben meegegeven bij de 75/25 verhouding niet goed is. Misschien wordt hier beter gepresteerd met andere waarden in de cost matrix. Om dit te controleren hebben we nog twee runs uitgevoerd met aangepaste cost matrices op de datasets met deze verhouding. Deze cost matrices zien er als volgt uit:

Figuur 4.2: Aangepaste cost matrices

(a) 1/2

geclassificeerd als → / echte waarde ↓	a	b
a	0	2
b	1	0

(b) 1/3

	a	b
a	0	3
b	1	0

(c) 1/4

	a	b
a	0	4
b	1	0

Hierbij is cost matrix 1/3 dezelfde cost matrix als in eerste instantie voor de 75/25 verhouding is gebruikt. We gaan kijken of het verhogen of verlagen van de misclassificatie cost voor de minderheidsklasse naar de meerderheidsklasse een positief effect kan hebben op de AUC score, in beide varianten.

Als er in de runs met deze nieuwe cost matrices ofwel minder significant slechte runs zijn, ofwel meer significante goede runs, dan is dat een teken dat de cost matrix voor deze verhouding niet optimaal is. Dit kan betekenen dat met een andere cost matrix MetaCost wel een positieve invloed zou kunnen hebben op de prestaties bij deze verhouding. Net zoals bij de vorige runs gebruiken we de Bonferroni-correctie zodat we zeker kunnen zijn van de significantie van onze runs. Omdat we 27 runs doen een significantie-level van 0.05 willen behalen, is onze uiteindelijk significantie grens $0.05/27 \approx 0.002$. Net als in de vorige tabel geven de iconen in de haakjes het verschil in AUC aan en zijn alle significante runs dikgedrukt.

Dataset	Algoritme	Cost	AUC gewoon	AUC cost-sensitive (verschil)	P-waarde
MAGIC	C4.5	1/2	0.834	0.825 (↓)	0.1426
		1/3	0.834	0.838 (↑)	0.5050
		1/4	0.834	0.837 (↑)	0.6178
	Random Forests	1/2	0.925	0.924 (↓)	0.7923
		1/3	0.925	0.920 (↓)	0.1949
		1/4	0.925	0.916 (↓)	0.0216
	SVM	1/2	0.500	0.500 (=)	1
		1/3	0.500	0.500 (=)	1
		1/4	0.500	0.500 (=)	1

Dataset	Algoritme	Cost	AUC gewoon	AUC cost-sensitive (verschil)	P-waarde
Credit	C4.5	1/2	0.664	0.674 (↑)	0.0477
		1/3	0.664	0.670 (↑)	0.2361
		1/4	**0.664**	**0.645 (↓)**	**0.0002**
	Random Forests	1/2	0.764	0.757 (↓)	0.1037
		1/3	**0.764**	**0.741 (↓)**	**<0.0001**
		1/4	**0.764**	**0.726 (↓)**	**<0.0001**
	SVM	1/2	0.654	0.643 (↓)	0.0338
		1/3	**0.654**	**0.623 (↓)**	**<0.0001**
		1/4	**0.654**	**0.610 (↓)**	**<0.0001**
thoracic	C4.5	1/2	0.484	0.529 (↑)	0.4237
		1/3	0.484	0.560 (↑)	0.1730
		1/4	0.484	0.602 (↑)	0.0317
	Random Forests	1/2	0.686	0.686 (=)	1
		1/3	0.686	0.679 (↓)	0.8849
		1/4	0.686	0.669 (↓)	0.7271
	SVM	1/2	0.550	0.550 (=)	1
		1/3	0.550	0.574 (↑)	0.6610
		1/4	0.550	0.574 (↑)	0.6610

Het verschil tussen de runs van verschillende costs is erg klein. Alleen bij Credit zijn er significante runs geweest en deze waren allemaal negatief. Het valt wel op dat bij de 1/2 cost matrix er minder (significant) negatieve runs zijn en bij de 1/4 cost matrix meer. Het lijkt er echter op dat de drie algoritmes verschillend reageren op de verandering in cost matrices. C4.5 krijgt over het algemeen betere resultaten naarmate de cost matrix meer richting de 1/4 gaat. Random Forests en SVM krijgen juist slechtere resultaten naarmate de cost matrix deze kant op gaat.

Hoofdstuk 5

Conclusies

Hoewel er weinig harde conclusies zijn te trekken uit een kleinschalig onderzoek als dit, blijkt uit onze resultaten wel een aantal patronen. Zo lijkt C4.5 redelijk gevoelig te zijn voor class imbalance en Random Forests redelijk ongevoelig. Dit bevestigt wat ook al bleek uit ander onderzoek. SVM bleek bij ons nog slechter om te gaan met class imbalance dan C4.5, maar het is onduidelijk of dat ligt aan de gebruikte datasets, of dat het algoritme van zichzelf hier slecht mee omgaat.

Verder lijkt MetaCost, met onze cost matrices, een positieve invloed te hebben op de prestatie van C4.5 op datasets waarbij de minderheidsklasse 10% of minder uitmaakt van de volledige dataset. Ook bij Random Forests zorgde MetaCost soms voor een significante verbetering, al lag de grens van verbetering hier wel lager dan bij C4.5 (rond de 1% minderheidsklasse).

Dit komt waarschijnlijk doordat het Random Forests algoritme diepere bomen toelaat dan C4.5, omdat ze uiteindelijk met alle andere bomen in het forest worden gemiddeld. Dit zorgt er voor dat Random Forests over het algemeen beter in staat is om takken naar de minderheidsklasse te creëren dan C4.5. Hierdoor gaat Random Forests beter om met class imbalance dan C4.5. Dit leidt er toe dat MetaCost pas bij een hogere graad aan class imbalance een positieve invloed op de resultaten van Random Forests.

Bij de wat meer gebalanceerde datasets (met 25% minderheidsklasse of meer) had MetaCost soms een (significant) negatieve invloed op de prestatie van de algoritmes. Nadat we dit verder onderzocht hebben, door te kijken naar de invloed van het veranderen van de cost matrices bij de 75/25 verhouding, blijkt dat de algoritmes hier verschillend op reageren.

Zo presteert C4.5 beter naarmate de misclassificatie cost omhoog gaat. Random Forests en SVM presteren in dit geval juist slechter. Echter, als de misclassificatie cost omlaag gaat presteert C4.5 slechter en Random Forests en SVM juist beter.

Dit komt waarschijnlijk doordat bij deze verhouding door de gedefinieerde costs er teveel in de originele dataset opnieuw wordt gelabeld, waardoor de algoritmes een item te snel classificeren als de minderheidsklasse. Waarom C4.5 hier geen last van lijkt te hebben is onduidelijk. Mogelijk komt dit omdat C4.5 meer gepruned wordt dan Random Forests en SVM. Het zou kunnen dat C4.5 daarom niet overfit op deze (door MetaCost mogelijk onrepresentatief gelabelde) dataset. Het is onduidelijk vanaf hoeveel imbalance in een dataset MetaCost effectief kan worden ingezet.

Vervolgonderzoek kan kijken naar waar deze grens precies ligt, zodat duidelijk wordt op welke datasets, in het algemene geval, MetaCost effectief kan worden toegepast. Ook kan het nuttig zijn om het onderzoek nogmaals uit te voeren op andere datasets, specifiek voor SVM. Zo wordt misschien duidelijk of de slechte prestatie van SVM komt door de gebruikte datasets, of dat het echt gerelateerd is aan de gevoeligheid van het algoritme voor class imbalance.

Bibliografie

[1] Bock, R. K. (2007). MAGIC Gamma Telescope Data Set. Verkregen van `archive.ics.uci.edu/ml/datasets/MAGIC+Gamma+Telescope`

[2] Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. Expert Systems with Applications, 39(3), 3446-3453.

[3] Burez, J., & Van den Poel, D. (2009). Handling class imbalance in customer churn prediction. Expert Systems with Applications, 36(3), 4626-4636.

[4] Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine learning, 20(3), 273-297.

[5] Domingos, P. (1999, Augustus). Metacost: A general method for making classifiers cost-sensitive. In Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 155-164). ACM.

[6] Devijver, P. A., & Kittler, J. (1982). Pattern recognition: A statistical approach. Prentice hall.

[7] Dunn, O. J. (1961). Multiple comparisons among means. Journal of the American Statistical Association, 56(293), 52-64.

[8] Fawcett, T. (2006). An introduction to ROC analysis. Pattern recognition letters, 27(8), 861-874.

[9] Geisser, S. (1993). Predictive inference (Vol. 55). CRC press.

[10] Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. Radiology, 143(1), 29-36.

[11] Ho, T. K. (1995, August). Random decision forests. In Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on (Vol. 1, pp. 278-282). IEEE.

[12] Japkowicz, N. (2000, Juni). The class imbalance problem: Significance and strategies. In Proc. of the Int'l Conf. on Artificial Intelligence.

[13] Japkowicz, N., & Stephen, S. (2002). The class imbalance problem: A systematic study. Intelligent data analysis, 6(5), 429-449.

[14] Kohavi, R. (1995, August). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Ijcai (Vol. 14, No. 2, pp. 1137-1145).

[15] Liu, X. Y., & Zhou, Z. H. (2006, December). The influence of class imbalance on cost-sensitive learning: An empirical study. In Data Mining, 2006. ICDM'06. Sixth International Conference on (pp. 970-974). IEEE.

[16] Lowry, R. (zonder datum). Significance of the Difference between the Areas under Two Independent ROC Curves. Verkregen van `vassarstats.net/roc_comp.html`

[17] Lubicz, M., Pawelczyk, K., Rzechonek, A., & Kolodziej, J. (2013). Thoracic Surgery Data Data Set. Verkregen van `archive.ics.uci.edu/ml/datasets/Thoracic+Surgery+Data`

[18] Platt, J. (1998). Sequential minimal optimization: A fast algorithm for training support vector machines.

[19] Quinlan, J. R. (1993) C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers.

[20] Schölkopf, B., Tsuda, K., & Vert, J. P. (2004). Kernel methods in computational biology. MIT press.

[21] Weiss, G. M. (2004). Mining with rarity: a unifying framework. ACM SIGKDD Explorations Newsletter, 6(1), 7-19.

[22] Yeh, I. C. (2016). default of credit card clients Data Set. Verkregen van `archive.ics.uci.edu/ml/datasets/default+of+credit+card+clients`