Recommender systems voor nieuws

Misschien vindt u dit ook interessant

Een casestudy voor de regionale omroep NH

Abstract

Voor NH onderzoek ik welke features de beste classifiers opleveren. Hiervoor maak ik gebruik van PU Learning (Elkan & Noto, 2008). Daarnaast stel ik een manier voor om te beoordelen of lezers een gelezen artikel interessant vonden zonder het te vragen. Deze informatie kan vervolgens gebruikt worden om de classifier op te trainen.

Het blijkt dat de beste features de eerste alinea en de volledige tekst van het artikel zijn. Verder is er met de voorgestelde interessemaat een kleine verbetering zichtbaar in de resultaten van het classificatie algoritme.

Thomas van Zwol, 3609367

30-6-17

Begeleider: Tomas Klos

Opleiding: bachelor Kunstmatige Intelligentie, UU

7,5 ECTS

Voorwoord

Ik wilde in mijn scriptie mijn beide interesses combineren – journalistiek en KI – en ik ben blij om te zeggen dat dat gelukt is. Ik heb voor NH, de regionale omroep van Noord-Holland, het begin mogen maken aan een *recommender system*. Dit valt precies in het vakgebied van KI waarin ik mij later wil specialiseren: *machine learning*.

Ik had deze scriptie niet kunnen schrijven zonder de hulp van anderen en daarom wil ik de volgende mensen bedanken:

Daan Zonneveld, voor het bieden van de mogelijkheid om mijn scriptie te doen bij Vinson en de feedback tijdens het schrijven;

Max Knobbout, voor de kritische vragen en gesprekken over de verschillende mogelijkheden bij het programmeren van de *recommender*;

Marcel Verbeek, voor het beantwoorden van al mijn vragen over de app en het toegankelijk maken van de informatiestromen;

Anne Cuperus, voor het helpen toegang krijgen tot de google analytics data zodat het mogelijk was om modellen te trainen;

NH, voor het beschikbaar stellen van hun data;

Emma Hendriks, voor het geduldig aanhoren van mijn struggles en het koken voor mij tijdens de laatste loodjes zodat ik me volledig kon richten op het schrijven;

Carlo van Zwol en Wiljo Los, voor de steun en het vertrouwen dat ik ooit de juiste studie zou gaan doen waardoor ik nu deze scriptie heb kunnen schrijven;

En last but not least, Tomas Klos, voor de begeleiding, de opbouwende feedback, de goede adviezen en de tussentijdse deadlines waardoor ik op schema bleef.

1. Inleiding

Lokale en regionale nieuwsmedia hebben het lastig. Ze hebben een kleinere doelgroep dan landelijke media en dus ook minder advertentie-inkomsten. Daarnaast hebben de regionale media afgelopen jaar een bezuiniging van 17 miljoen euro voor de kiezen gekregen. Dit komt neer op gemiddeld 10% minder budget per regionale omroep. Veel omroepen hebben daarom werknemers moeten ontslaan of zelfs moeten reorganiseren (Roodheuvel, 2016).

Advertenties zijn een belangrijke bron van inkomsten (Mulder, Poort, Marlet, & van Woerkens, 2006). Traditioneel werden die vertoond dan wel laten horen rond tv- en radioprogramma's. Enige tijd geleden kwamen daar websites bij en sinds een paar jaar is de app die hoort bij de omroep ook een belangrijke plek voor het plaatsen van advertenties.

Voor deze advertenties wordt per view betaald door de adverteerders (Wang & Yuan, 2015). Het zou voor de regionale media goed zijn als ze het aantal views kunnen maximaliseren omdat op die manier meer geld wordt verdiend. Om het aantal views te maximaliseren is het belangrijk dat de traffic in de app toeneemt. Het laten toenemen van traffic kan op verschillende manieren: bijvoorbeeld met een beter ontwerp van de app, met meer relevante artikelen of door op het bestaan van de app te wijzen in de traditionele media-uitingen. Deze manieren zorgen ervoor dat mensen vaker de app openen, maar op een gegeven moment loop je daar tegen een maximum aan. Een andere manier om traffic te verhogen, is door mensen langer in de app te houden wanneer ze die eenmaal geopend hebben. Hiervoor zijn persoonlijke aanbevelingen van nieuws een belangrijk gereedschap. Uit onderzoek is gebleken dat *recommender systems* de lengte van een bezoek met een factor 2,5 kan laten toenemen (Garcin, et al., 2014).

1.1 Recommender systemen

Een *recommender system* doet aanbevelingen aan gebruikers voor artikelen die mogelijk ook interessant zijn om te lezen. Waarop het deze voorspelling baseert, is wat het onderscheid maakt tussen de verschillende typen *recommender systems*. Er zijn heel veel verschillende manieren om dit te doen, maar de meest gangbare zijn de volgende: *content based recommendation*, *collaborative filtering* en een hybride versie van een combinatie van deze twee (Said & Bellogín, 2014) (Lu, Wu, Mao, Wang, & Zhang, 2015).

Content based recommendation is het type aanbeveling waarbij gekeken wordt naar eerder gelezen artikelen van een gebruiker. Op basis van deze artikelen wordt een voorkeuren profiel opgesteld met bepaalde eigenschappen die de gebruiker interessant vindt. Vervolgens worden artikelen aanbevolen

die aansluiten bij het interesseprofiel (Lu, Wu, Mao, Wang, & Zhang, 2015). Door te specificeren naar hoeveel eerder gelezen artikelen er gekeken moet worden, kan er een tijdshorizon bepaald worden.

Collaborative filtering kan onderverdeeld worden in *user based* en *item based*. Bij *user based* krijgt een gebruiker aanbevelingen van artikelen die vergelijkbare gebruikers ook interessant vonden. Bij *item based* krijgt een gebruiker aanbevelingen van artikelen die lijken op artikelen die hij eerder interessant vond (Lu, Wu, Mao, Wang, & Zhang, 2015). Het verschilt *van content based classification* in dat het kijkt naar artikelen die lijken op individuele eerder gelezen artikelen, terwijl bij *content based classification* gekeken wordt naar een interesseprofiel dat opgebouwd is uit de eerder gelezen artikelen.

Bij *recommender systems* wordt gebruik gemaakt van een *classifier*: een functie die vectoren afbeeldt op klassen. Als input heeft een classifier een vector van eigenschappen (features) nodig van datgene waar hij voorspellingen over moet doen. Als output geeft de *classifier* vervolgens de klasse waartoe volgens de *classifier* deze input behoort. In het geval van een *recommender system* is de uitkomst hiervan binair: aanbevelenswaardig of niet. Ook is het mogelijk om de *classifier* zo in te stellen dat het als output de kans geeft waarmee het 'denkt' dat het tot een bepaalde klasse behoort. Tijdens het trainen en evalueren van de *classifier* zijn voorbeelden en labels nodig. Voorbeelden zijn de vectoren van eigenschappen, labels zeggen over de voorbeelden tot welke klasse ze behoren. Meestal is er sprake van een positieve klasse (met label 1) en een negatieve klasse (met label 0 of -1). Bij het trainen leert de *classifier* aan de hand van deze voorbeelden met bijbehorende labels welke eigenschappen indicatief zijn voor welke klassen en hoe hij nieuwe voorbeelden die hij nog niet heeft gezien moet classificeren. Bij het evalueren krijgt de *classifier* alleen voorbeelden te zien waarna het daarvan de klasse moet voorspellen. Deze voorspellingen worden vervolgens vergeleken met de labels.

Om de prestatie van een *classifier* te evalueren zijn er twee belangrijke maten: *precision* en *recall*. Precision is gedefinieerd als het deel van de positieve voorspellingen dat daadwerkelijk positief is (*true positives*). *Recall* is gedefinieerd als het deel van alle positieve voorbeelden waarvan de *classifier* voorspelt dat ze positief zijn. Er bestaat altijd een balans tussen *precision* en *recall*: als verder alles hetzelfde blijft en de *recall* omhoog gaat, dan gaat de *precision* naar beneden en vice versa. Dit komt doordat voor een hogere *recall* van meer artikelen voorspeld moet worden dat ze positief zijn, waardoor de kans op verkeerd voorspelde data toeneemt en daarmee de *precision* afneemt. Het is dus van belang om te beslissen waar de balans moet liggen tussen deze twee waarden wanneer een *recommender system* in gebruik genomen wordt.

1.2 Vraagstukken

Een belangrijk probleem bij het maken van een *recommender system* voor nieuws artikelen, is dat lezers niet expliciet aangeven dat ze artikelen niet interessant vinden: de labels voor de klasse 'niet-interessant' ontbreken. Hierdoor bestaat de trainingsdata voor het *recommender system* uit positieve (de gelezen artikelen) en ongelabelde (de ongelezen artikelen) data. Welke artikelen ze interessant vinden valt af te leiden uit welke artikelen ze geopend hebben en hoe lang, maar over oninteressante artikelen valt een stuk minder te zeggen. Niet openen kan, naast het ontbreken van interesse in het specifieke artikel, namelijk komen door verschillende redenen: het artikel niet voorbij hebben zien komen, geen tijd om het artikel op dit moment te lezen of op zoek zijn naar iets anders om er een paar te noemen. Voor een *recommender system* is het ontbreken van negatief gelabelde data (oninteressante artikelen) een probleem, omdat een *classifier* beide klassen nodig heeft om te leren. Als de *classifier* alle niet gelezen artikelen als oninteressant zou beschouwen, dan leert de *classifier* het verkeerde aan, want tussen de niet gelezen artikelen zitten artikelen die de gebruiker wel interessant had gevonden.

Het probleem van positieve en gelabelde data kan worden opgelost met Positive and Unlabeled (PU) Learning zoals voorgesteld door Elkan en Noto (2008). De oplossing baseert zich op de aanname dat de positieve voorbeelden willekeurig zijn geselecteerd uit alle voorbeelden en dat er tussen de niet gelabelde data ook positieve voorbeelden zitten. Deze aanname acht ik in deze situatie plausibel: van alle artikelen die een gebruiker interessant zou kunnen vinden, is er een deel dat hij leest. De keuze voor welke artikelen hij daadwerkelijk leest is geen bewuste. De aanname die hoort bij PU Learning komt bovenop de aanname dat de positieve voorbeelden gescheiden kunnen worden van de negatieve. Voor deze situatie is PU Learning een manier om om te gaan met het feit dat de data bestaat uit positieve en ongelabelde data. Uit het onderzoek van Elkan en Noto komt naar voren dat deze methode een verbetering zou moeten opleveren ten opzichte van Positive and Negative learning. Ik onderzoek of dat hier ook het geval is.

Het grootste vraagstuk bij het maken van een *recommender system*, is welke features gebruikt moeten worden (Forman, 2003). Daarnaast is het ook belangrijk of je gebruik maakt van alleen het openen van een artikel, of de tijd die gespendeerd wordt aan een artikel. Het is namelijk mogelijk dat de klasse van artikelen 'interessant voor gebruiker' vervuild raakt door het per ongeluk openen van artikelen en het openen van artikelen waarvan de gebruiker na het lezen van de eerste paar zinnen vindt dat het toch geen interessant artikel is. Ik onderzoek of een drempelwaarde voor de tijd die besteed is aan het artikel betere voorspellingen oplevert doordat op deze manier de vervuiling van de klasse interessant enigszins wordt tegen gegaan.

2. Probleemanalyse

Voor de app van NH zijn een aantal verschillende *recommendation engines* mogelijk. Deze kunnen grofweg opgesplitst worden in *content based recommendation* en *collaborative filtering*. Voor de NH app zijn op dit moment geen live statistieken mogelijk en is het alleen mogelijk om de data van gisteren en eerder op te vragen: er kan van artikelen die vandaag gepubliceerd worden niet gezien worden hoeveel mensen die hebben gelezen, van een gebruiker die vandaag de app opent kan niet gezien worden wat hij allemaal bekijkt et cetera. Op deze informatie moet een dag gewacht worden, voordat het zichtbaar is in Google Analytics. Hierdoor kan *user based collaborative filtering* op dit moment niet ingezet worden omdat van een *recommendation engine* verwacht wordt dat het alle artikelen kan aanbevelen die beschikbaar zijn, ook degene die vandaag gepubliceerd zijn. In het geval van *user based collaborative filtering* is dat niet mogelijk, omdat over de artikelen van de huidige dag niets bekend is.

Item based collaborative filtering is wel mogelijk, omdat van de artikelen wel alle features bekend zijn en het dus mogelijk is om voor elk artikel te bepalen welk artikel er het meest op lijkt.

Content based recommendation kan gedaan worden door de data van gisteren en eerder als trainingsdata te gebruiken en de artikelen die op dit moment in de app te zien zijn te gebruiken als testdata.

Een speciaal geval van *item based collaborative filtering* kan gebruikt worden om aanbevelingen te doen op basis van het artikel dat op een specifiek moment bekeken wordt door een gebruiker: door de grootte van de geschiedenis waarnaar gekeken wordt op 1 te zetten, wordt alleen naar het huidige artikel gekeken en worden op basis daarvan de artikelen aanbevolen die er het meest op lijken. Dit noem ik in het vervolg *current item based recommendation*.

De verschillende manieren van aanbevelingen doen kunnen los of gecombineerd worden ingezet in verschillende *recommender systems* die op verschillende plekken in de app aanbevelingen doen.

2.1 Current item based recommendation

Current item based recommendation is relatief eenvoudig te implementeren omdat er geen *machine learning* aan te pas komt: sorteer alle artikelen op gelijkenis ten opzichte van het huidige artikel en beveel de bovenste zoveel aan. Om deze aanbeveling te doen, moet er een mate van gelijkheid worden berekend tussen artikelen. De manier waarop dit gedaan wordt, bepaalt hoe goed de maat van gelijkheid is en dus hoe goed de aanbeveling is. De gelijkheidsmaat is de factor die bepaalt hoeveel rekenwerk het vergt om deze manier van aanbeveling te doen. Uit onderzoek van Thada en Jaglan komt naar voren dat de cosinusgelijkenis de beste maat is voor gelijkheid in tekstdocumenten

(2013). De waarde van deze functie ligt tussen -1 en 1, waarbij -1 betekent dat het volledig het tegenovergesteld is en 1 volledige gelijkheid. Aangezien de features waarop de cosinusgelijkheid wordt toegepast een waarde hebben tussen 0 en 1, zal de waarde van de cosinusgelijkheid ook tussen de 0 en 1 liggen. In deze context kan een waarde van 0 dus geïnterpreteerd worden als het ontbreken van overeenkomst. Bij de aanbeveling van artikelen is het nodig om een drempelwaarde te kiezen voor de cosinusgelijkheid waarboven de artikelen daadwerkelijk worden aanbevolen: een zeer kleine gelijkheid tussen de artikelen kan leiden tot het aanbevelen van artikelen die te weinig overeenkomen om interessant te zijn. Aangezien het vertrouwen in aanbevelingen door een *recommender agent* afhangen van de accuraatheid van de aanbevelingen (Chen & Pu, 2005), is deze drempelwaarde noodzakelijk. De notie van een drempelwaarde is ook bij de andere aanbevelingssystemen van belang, omdat ook daar de accuraatheid invloed heeft op het vertrouwen in de aanbevelingen.

2.2 Content based recommendation

De *content based recommendation* maakt de beleving van het gebruik van de app persoonlijker omdat bij deze vorm van aanbeveling de artikelen worden aanbevolen op basis van eerder gelezen artikelen en daarom aansluiten bij de persoonlijke interesse. Het model dat gebouwd wordt op deze eerder gelezen artikelen kijkt naar de eigenschappen van deze artikelen. De eigenschappen zijn bij de classificatie van tekst over het algemeen de woorden uit de tekst. Tijdens het bouwen van het model leert het model welke woorden belangrijk zijn en die iets zeggen over het onderwerp van de tekst. Het model op basis van de artikelen gelezen door een specifieke gebruiker kan gezien worden als een interesseprofiel van de gebruiker: welke woorden, en dus onderwerpen, zijn doorslaggevend bij het bepalen van wel of niet interessant.

Het is gebruikelijk om voor de labels voor het classificatie algoritme gebruik te maken van het al dan niet openen van een artikel, maar ik stel een andere manier voor om tot betere aanbevelingen te komen: omdat bij de gebruikelijke aanpak er alleen gekeken wordt naar wel of niet openen, gaat er waardevolle informatie verloren die iets kan zeggen over hoe interessant de gebruiker het artikel echt vond: als een artikel geopend wordt, maar de gebruiker verlaat het artikel alweer na het lezen van de eerste twee zinnen, dan vindt de gebruiker dit artikel zeer waarschijnlijk minder interessant dan een artikel waarvan alles gelezen wordt. Om een maat te hebben van hoeveel er gelezen wordt kan het aantal seconden per woord berekend worden voor een artikel. Dit is te berekenen door het aantal seconden dat de gebruiker besteed heeft aan het artikel te delen door de lengte van de tekst in woorden. Door dit te delen door het gemiddelde aantal seconden per woord voor deze gebruiker, kan voor een specifiek artikel een interessescore berekend worden: bij een interesse die gelijk is aan

het gemiddelde heeft deze deling als resultaat 1, bij bovengemiddelde interesse is het resultaat hoger dan 1 en bij een beneden gemiddelde interesse ligt de waarde lager dan 1.

Deze informatie kan op verschillende manieren verwerkt worden waarbij er twee voor de hand liggen. De eerste is van gelezen artikelen het label 'gelezen' veranderen in 'ongelezen' als de interessescore niet boven een bepaalde drempelwaarde uitkomt. Hiermee verandert de data van positieve en ongelabelde data (PU) naar positieve, ongelabelde en negatieve data (PUN). Deze nieuwe labels kunnen vervolgens op twee manieren gebruikt worden: de negatieve voorbeelden kunnen herlabeld worden naar ongelabeld waardoor deze nieuwe labels gebruikt kunnen worden voor het trainen van een Support Vector Machine (SVM) met behulp van PU Learning. De drempelwaarde voor de interessemaat kan vervolgens zo gekozen worden dat het model de beste voorspellingen doet.

Daarnaast kunnen de nieuwe PUN labels ook gebruikt worden voor Label Propagation, een *semi-supervised* manier van *machine learning*. Label Propagation wordt in dit onderzoek niet onderzocht, maar is wel de moeite van het onderzoeken waard.

De tweede manier waarop de informatie verwerkt kan worden, is door te voorspellen hoeveel seconden per woord een gebruiker wil besteden aan een artikel. De voorspelling van het aantal seconden per woord kan gedaan worden door een regressie algoritme. Door vervolgens deze waarde te delen door het gemiddelde aantal seconden per woord krijg je weer een interessescore voor het artikel. Door deze score te vergelijken met een drempelwaarde kan vervolgens het artikel geclassificeerd worden als aanbevelenswaardig of niet. Deze aanpak wordt in dit onderzoek niet onderzocht, maar is zeker de moeite waard voor een vervolgonderzoek.

Naast deze nieuwe aanpak, is er ook een 'gewoon' classificatie algoritme gebruikt om een *content based recommender* te bouwen. Hiervoor is gebruik gemaakt van een Support Vector Machine met *soft margin* aangezien uit de literatuur naar voren is gekomen dat dit een van de beste algoritmes is voor tekst classificatie (Aggarwal & Zhai, 2012). Hierbij is gebruik gemaakt van PU Learning. Dit algoritme zal gebruikt worden om de hierboven genoemde alternatieve aanpakken mee te vergelijken.

2.3 Situatie specifieke keuzes

Welke features er gebruikt moeten worden om het *recommender system* op te baseren is het grootste vraagstuk en lastig te baseren op literatuur. Elk artikel bij NH heeft een aantal tags die meegegeven worden door de journalist. Dit zijn er in de regel niet meer dan zes: vrij weinig om een goede *recommendation* op te doen, maar wellicht genoeg in deze situatie. Een andere aanpak is het

gebruik van de volledige tekst en daar met TF-IDF een vector van het maken. Dit is computationeel een stuk zwaarder en zou dus een obstakel kunnen zijn voor gebruik bij NH. Een zo goed mogelijke aanbeveling is wenselijk, maar het moet ook *feasible* zijn: het moet niet zo zijn dat er uren besteed moeten worden aan het bouwen van een model als met een fractie van de tijd een model gemaakt kan worden dat iets minder goed is, maar dat nog steeds goed functioneert. Er zijn verschillende middenwegen mogelijk hierin: in plaats van het hele artikel kan ervoor gekozen worden om alleen de kop of de kop en de intro te gebruiken. Dit maakt het aantal woorden dat gevectoriseerd moet worden een stuk kleiner en daardoor sneller te berekenen. Verder kan het aantal woorden in het artikel gebruikt worden. Andere features die buiten de scope van dit onderzoek liggen, zijn de features van de foto's die bij de artikelen geplaatst worden.

Een andere afweging die gemaakt dient te worden is het aantal artikelen waarop het *recommender system* getraind moet worden. Een groter aantal artikelen kan zorgen voor meer accurate aanbevelingen wanneer interesses niet veranderen, een kleiner aantal artikelen vangt beter veranderende en tijdelijke interesses in de gebruikers. Daarnaast heeft dit aantal ook invloed op de tijd die het kost om een model te trainen. Naast het totale aantal artikelen waarop getraind wordt, is het ook van belang om een minimumaantal artikelen per gebruiker te specificeren: als de gebruiker te weinig artikelen heeft gelezen zullen de aanbevelingen ook van slechte kwaliteit zijn omdat er niet genoeg informatie is om de aanbevelingen op te baseren. Voor de *content based recommendation* wordt de NH-app op dit moment beperkt in de hoeveelheid artikelen door de artikelen die op dit moment te zien zijn in de app: maximaal 200. Vanwege de tijdspanne waarin dit onderzoek plaats vindt is ervoor gekozen om het corpus te beperken tot de artikelen die in deze tijdspannen gepubliceerd zijn: 1892 artikelen.

Het toepassen van *machine learning* is eenvoudig in Python door het bestaan van het scikit learn pakket waarin allerlei standaard *machine learning* technieken zijn opgenomen. Het toepassen van PU Learning wordt makkelijk gemaakt met het Python pakket PUAdapter[1]. Deze adapter is gemaakt op basis van de wiskunde zoals beschreven door Elkan en Noto (2008). Hiermee is het mogelijk om de standaard *machine learning* algoritmen uit scikit learn te gebruiken zonder dat de data daarbij verdeeld is in positieve en negatieve voorbeelden. De enige voorwaarde is dat de algoritmen een mogelijkheid hebben om de uitkomst weer te geven in de kans dat een voorbeeld volgens de *classifier* tot een bepaalde klasse hoort.

[1] https://github.com/aldro61/pu-learning

3. Methode

Voor dit onderzoek vergelijk ik verschillende dingen met elkaar. Allereerst kijk ik welke featuresets de beste resultaten opleveren. Vervolgens gebruik ik de beste drie featuresets om te kijken of PU Learning in deze situatie betere resultaten oplevert dan wanneer een gewone *classifier* gebruikt wordt en de niet gelezen artikelen als negatieve voorbeelden gezien worden. Daarna onderzoek ik of het herlabelen van gelezen artikelen naar ongelezen artikelen de resultaten van de *classifier* kunnen verbeteren. Ook onderzoek ik of de seconden per woord-maat zodanig te voorspellen is met een regressie algoritme dat aan de hand daarvan de artikelen geclassificeerd kunnen worden.

Het is te verwachten dat de *precision* van de modellen niet zeer groot zal zijn. Dit komt doordat de meeste gebruikers niet in staat zijn om alle artikelen te lezen die ze interessant vinden. De modellen zullen deze artikelen wel als positief classificeren, terwijl ze niet daadwerkelijk gelezen zijn. Dat betekent automatisch dat de *precision* omlaag gaat aangezien deze gedefinieerd is als het deel van de positief geclassificeerde artikelen dat daadwerkelijk positief (en in dit geval dus gelezen) is.

3.1 Data

De data die ik voor dit onderzoek heb bestaat uit een corpus van 1892 nieuws artikelen gepubliceerd door NH en de leesgegevens van gebruikers van de NH app. De artikelen zijn gepubliceerd in een periode van 42 dagen. Het aantal gebruikers waarvoor de modellen getest worden is 100. De 100 gebruikers zijn willekeurig gekozen uit de verzameling van gebruikers van de NH app met de voorwaarde dat ze minimaal 2% van de artikelen gelezen hebben. De reden voor een ondergrens is dat PU Learning een minimumaantal positieve voorbeelden nodig heeft. De keuze voor 2% is gemaakt omdat 2% neer komt op net iets minder dan 1 artikel per dag (een minimum van 38 artikelen).

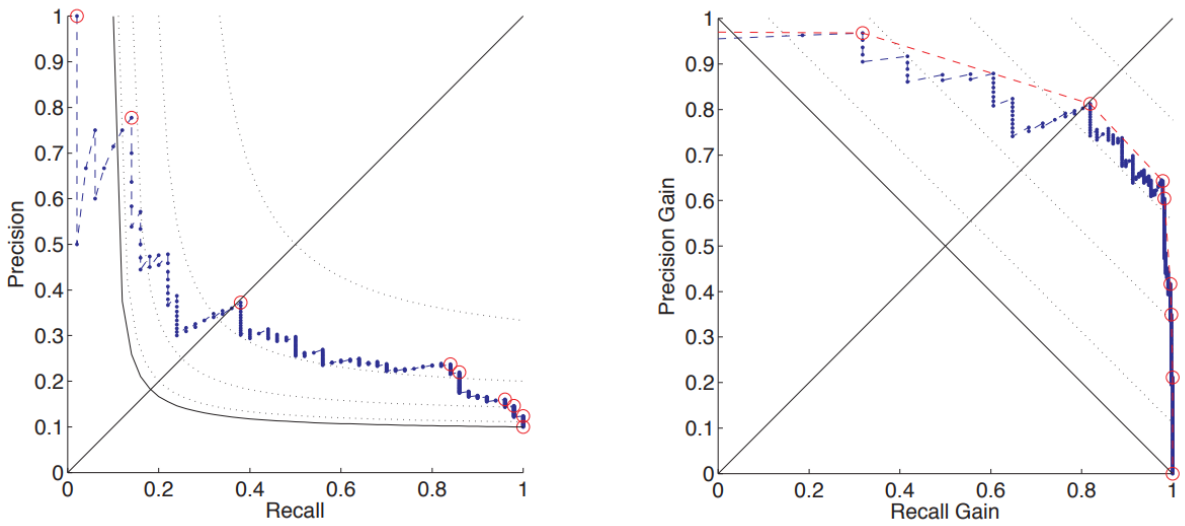
3.2 Evaluatie van modellen

Voor evaluatie van modellen wordt vaak gebruik gemaakt van de F_1 score: het harmonisch gemiddelde van *precision* en *recall*. Ik maak echter gebruik van Precision-Recall-Gain curves en de oppervlakte onder deze grafiek (AUPRG). De reden dat ik gebruik maak van Precision-Recall-Gain curves, is omdat van F-scores geen rekenkundig gemiddelde genomen kan worden omdat het een harmonische schaal is en omdat de oppervlakte van Precision-Recall curves verkeerde modellen aan kan wijzen als beste model omdat deze geplot worden op een lineaire schaal terwijl een harmonische schaal noodzakelijk is. Bij Precision-Recall-Gain curves speelt dit probleem niet, omdat die curves geplot worden op een harmonische schaal in plaats van een lineaire.

Receiver Operator Characteristic curves zijn hier ook niet geschikt, omdat daarvoor de *true positive rate* (*recall*) wordt geplot tegen de *false positive rate*. De *false positive rate* is in deze situatie lastig te berekenen omdat die gedefinieerd is als $\mathrm{FP}/(\mathrm{FP} + \mathrm{TN})$ waar FP het aantal *false positives* is en TN het aantal *true negatives*.

Zowel FP als TN is onduidelijk bij positieve en ongelabelde data: wat zijn precies de negatives? In het geval dat je ongelabelde data neemt als negatieve data, fluctueert de uitkomst van de breuk sterk wanneer je een aantal van de ongelabelde datapunten toch als positief beschouwt: hierdoor verandert zowel de teller als de noemer. Precision lijkt enigszins op de *false positive rate*: $\mathrm{FP}/(\mathrm{FP} + \mathrm{TP})$. Deze breuk houdt echter de noemer constant wanneer je een deel van ongelabelde data punten als positief ziet in plaats van negatief.

De F_1 score die verslagen dient te worden komt van het *always positive* model dat elk punt als positief classificeert. Vanwege deze eigenschap heeft dat model altijd een *precision* van *p* (waarbij *p* de kans is op een positief voorbeeld) en een *recall* van 1. Elk model met een *precision* of *recall* kleiner dan *p* heeft een lagere F_1 score en is daarmee minder goed dan het *always positive* model. Daarom wordt bij Precision-Recall-Gain voor *precision* en *recall* alleen gekeken naar het bereik [p, 1] en worden de assen harmonisch naar dit bereik geschaald. Dit resulteert in grafieken waarbij de oppervlakte onder de grafiek iets zegt over de fit van het model (Flach & Kull, 2015). Door gebruik te maken van de PRG-curve wordt ook automatisch de drempelwaarde voor positieve classificatie gevarieerd in de evaluatie.



Figuur 1: links een conventionele Precision Recall curve met hyperbolische F_1 isometrie (gestippelde lijnen) en de prestatie van de always positive classifier (ononderbroken hyperbolische lijn). Rechts een PRG-curve met de diagonaal tussen linksboven en rechtsonder als de prestatie van de always positive classifier, parallelle F_1 isometrie en een convex Pareto front. Afbeelding afkomstig uit Flach en Kull (2015).

Een gelijke F_1 score is in de PRG-curve weer te geven als een lijn met helling -1 waarbij de F_1 score van een *always positive classifier* de diagonaal is tussen linksboven en rechtsonder. De diagonaal van linksonder naar rechtsboven is de diagonaal waar *precision* gelijk is aan *recall*. Bij andere waarden van β voor F_β, is helling van de lijn die de score aangeeft $-\beta^2$. Zoals blijkt uit figuur 1 is het bij een PRG-curve veel eenvoudiger te zien wanneer punten een gelijke F-score hebben.

3.3 Voorwerk

In dit onderzoek maak ik gebruik van Support Vector Machines (SVM). Een SVM construeert een hypervlak in een hoog dimensionale ruimte met als doel de twee klassen te scheiden met dit hypervlak. Deze *classifiers* hebben parameters die bepalen hoe goed de classificatie gebeurt: de penalty term C en het type *kernel*. De penalty term geeft aan hoe erg het is als de *classifier* voorbeelden verkeerd classificeert: vaak is het onmogelijk om een hypervlak te construeren dat precies de klassen scheidt en zijn er een aantal voorbeelden die daardoor verkeerd worden geclassificeerd.

De *kernel* is de manier waarop er geprobeerd wordt de data van elkaar te scheiden. Door Hsu, Chang en Lin (2003) wordt aangeraden om als eerste een Radial Basis Function (RBF) *kernel* te gebruiken omdat een aantal andere *kernels* speciale gevallen zijn van de RBF *kernel*. Deze *kernel* is een maat van gelijkheid tussen telkens twee voorbeelden en wordt gebruikt om te leren welke voorbeelden op elkaar lijken. Bij deze *kernel* hoort nog een parameter gamma. Voor de parameter C zoek ik uit wat de beste resultaten geeft, gegeven dat de parameter gamma ingesteld wordt op 1 / n waar n het aantal features is. De keuze voor deze gamma wordt verderop toegelicht. De waarden die ik voor C probeer zijn: 1, 2, 4, 8, 16, 32, 64, 128, 256. Ik maak gebruik van een SVM zoals hieronder beschreven onder het kopje Featuresets met de featureset die bestaat uit alleen feature 4.

C	Gemiddelde (N = 9)	Standaarddeviatie
1	0,388	0,094
2	0,419	0,070
4	0,433	0,065
8	0,475	0,074
16	0,514	0,081
32	0,545	0,091
64	0,479	0,093
128	0,489	0,089
256	0,459	0,097

Tabel 1: gemiddelde AUPRG-scores van de folds en hun standaarddeviatie per waarde voor de parameter C

Uit dit korte vooronderzoek blijkt dat C = 32 de beste *classifiers* oplevert. In Tabel 1 staan de gemiddelde AUPRG-scores van de modellen en de standaarddeviatie. De uiteindelijke *classifiers*

kunnen beter gemaakt worden door *grid search* toe te passen over deze parameters, maar omdat het in dit onderzoek gaat om een vergelijking tussen featuresets en om een vergelijking tussen de verschillende doelwaarden gegeven een drempelwaarde, is een relatieve score ten opzichte van elkaar belangrijk en niet zozeer welke parameters de beste *classifiers* opleveren.

3.4 Featuresets

Om de beste featuresets te bepalen maak ik gebruik van een SVM met een RBF *kernel*, C = 32 en gamma = 1 / n waar n het aantal features is. Deze waarde voor gamma vloeit voort uit het gebruik van scikit learn waar ik de waarde voor gamma op 'auto' heb ingesteld, wat gedefinieerd is als 1 / n-features.

De voorbeelden worden gezien als behorend tot de klasse positief en ongelabeld. Hoeveel featuresets ik aanmerk als featuresets om de rest van de evaluaties mee te doen, is afhankelijk van de uitkomsten van dit onderdeel: liggen de waarden dicht bij elkaar zal ik meer featuresets meenemen bij de volgende testen, zijn er grote verschillen dan worden het er minder.

Voor elke featureset genereer ik voor elke gebruiker modellen. Dit doe ik met 10-fold kruisvalidatie. Uit de uitkomsten van de voorspellingen van elke *fold* worden een AUPRG-score berekend voor die *fold*. Met alle uitkomsten samen worden een PRG-curve geplot en een AUPRG-score berekend voor de combinatie van featureset en C-waarde.

De verschillende features die ik gebruik zijn: tags gegeven door de journalist (1), de titel (2), de inleiding van het artikel (3), de volledige tekst van het artikel (4) en de lengte in woorden van het artikel (5). Met deze vijf verschillende features test ik alle verschillende combinaties van features om te zien hoe die vergelijken ten opzichte van elkaar (daarbij uiteraard 3 en 4 niet tegelijk gebruikend). Ook test ik feature 5 niet apart omdat deze feature niets zegt over het onderwerp van het artikel, maar wel van belang kan zijn voor de tijd die een gebruiker besteed aan het artikel.

Voor alle features die bestaan uit een *bag of words* maak ik gebruik van Latent Semantic Indexing (LSI) om de belangrijkste termen te bepalen in het corpus van artikelen. Latent Semantic Indexing wordt gebruikt om latente relaties tussen woorden te analyseren en daarmee op een hoger niveau het onderwerp van teksten vast te stellen. Het gaat uit van de aanname dat woorden die qua betekenis veel op elkaar lijken vaak samen in dezelfde tekst voorkomen. De reden dat ik gebruik maak van LSI naast TF-IDF, in plaats van het meer gangbare gebruik van alleen TF-IDF, is dat uit onderzoek blijkt dat LSI over het algemeen betere resultaten behaalt (Zhang, Yoshida, & Tang, 2011). Conceptueel is dit logisch: door naar een meer abstractere weergave van het onderwerp te kijken, in

plaats van naar alle woorden die voorkomen in een tekst, zijn artikelen beter in te delen in groepen waardoor de interesses van een gebruiker beter te vangen zijn.

De TF-IDF vector voor een artikel bereken ik voor elke feature apart en vervolgens bereken ik over de gehele feature vector de LSI vector. De LSI vector heeft als dimensie: het minimum van het aantal features in de vector -1 en 100. Het maximum van 100 dimensies is op aanraden van de makers van het scikit learn pakket (sklearn.svm.SVC - scikit-learn 0.18.2 documentation).

3.5 PU Learning tegenover PN Learning

Met de beste featuresets train ik vervolgens vijf keer SVM *classifiers* met een RBF *kernel*, gamma = 0.01 en C = 32. Ook hier maak ik gebruik van een corpus van 1892 artikelen, 10-fold kruisvalidatie en 100 gebruikers waarvoor ik de modellen train. De gemiddelde resultaten van de *classifiers* vergelijk ik met de uitkomsten van het vorige onderdeel waarbij wel PU Learning werd toegepast.

3.6 Herlabelen naar ongelezen

Vervolgens train ik met de beste featuresets SVM *classifiers*. Afhankelijk van wat de uitkomst is bij de vergelijking tussen PU Learning en PN Learning, gebruik ik wel of niet PU Learning hiervoor. Het verschil met de SVM *classifiers* uit bovenstaande onderdelen, is dat ik de doel-waarden aanpas. Hiervoor gebruik ik de interessemaat die ik in de sectie *Content based classifiers* heb uitgewerkt: seconden per woord / gemiddelde seconden per woord. Of een artikel herlabeld wordt naar ongelezen is afhankelijk van een drempelwaarde: als de interessemaat groter is dan de drempelwaarde wordt het artikel niet herlabeld en anders wel. Ik zal deze test uitvoeren voor de drempelwaarden: 0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7 en 0.8. Ook hier maak ik weer gebruik van hetzelfde corpus, dezelfde gebruikers en 10-fold kruisvalidatie. De scores geef ik wederom weer in AUPRG en vergelijk ik met de scores van verder dezelfde SMV *classifier* maar met de gewone doel-waarden.

4. Resultaten

Uit de verschillende runs is naar voren gekomen dat bij 10-fold kruisvalidatie op deze data, de scores van de eerste *fold* altijd ver van de daadwerkelijke score liggen. Daarom heb ik besloten de scores van de eerste *fold* niet te gebruiken voor de gemiddelde *fold* scores. De reden voor deze extreme afwijking is volgens mij dat bij de eerste *fold* de laatst gepubliceerde artikelen gebruikt worden om op te voorspellen. Van deze groep artikelen is het aannemelijk dat de gebruikers er verhoudingsgewijs minder van gelezen hebben, waardoor de *precision* slecht is en de AUPRG-score

als gevolg daarvan ook. Of het inderdaad zo is dat het percentage gelezen artikelen lager ligt en of dit een verklaring is voor de lage scores, zou onderzocht moeten worden. Ik heb ervoor gekozen om de scores uit de eerste *fold* te negeren aangezien de gemiddelde score van de andere *folds* dicht bij de scores van het gehele model lagen, waarbij de gemiddelde scores iets lager waren dan de model scores. Een overzicht van de volledige scores op basis waarvan ik deze keuze gemaakt heb is te vinden in de Appendix.

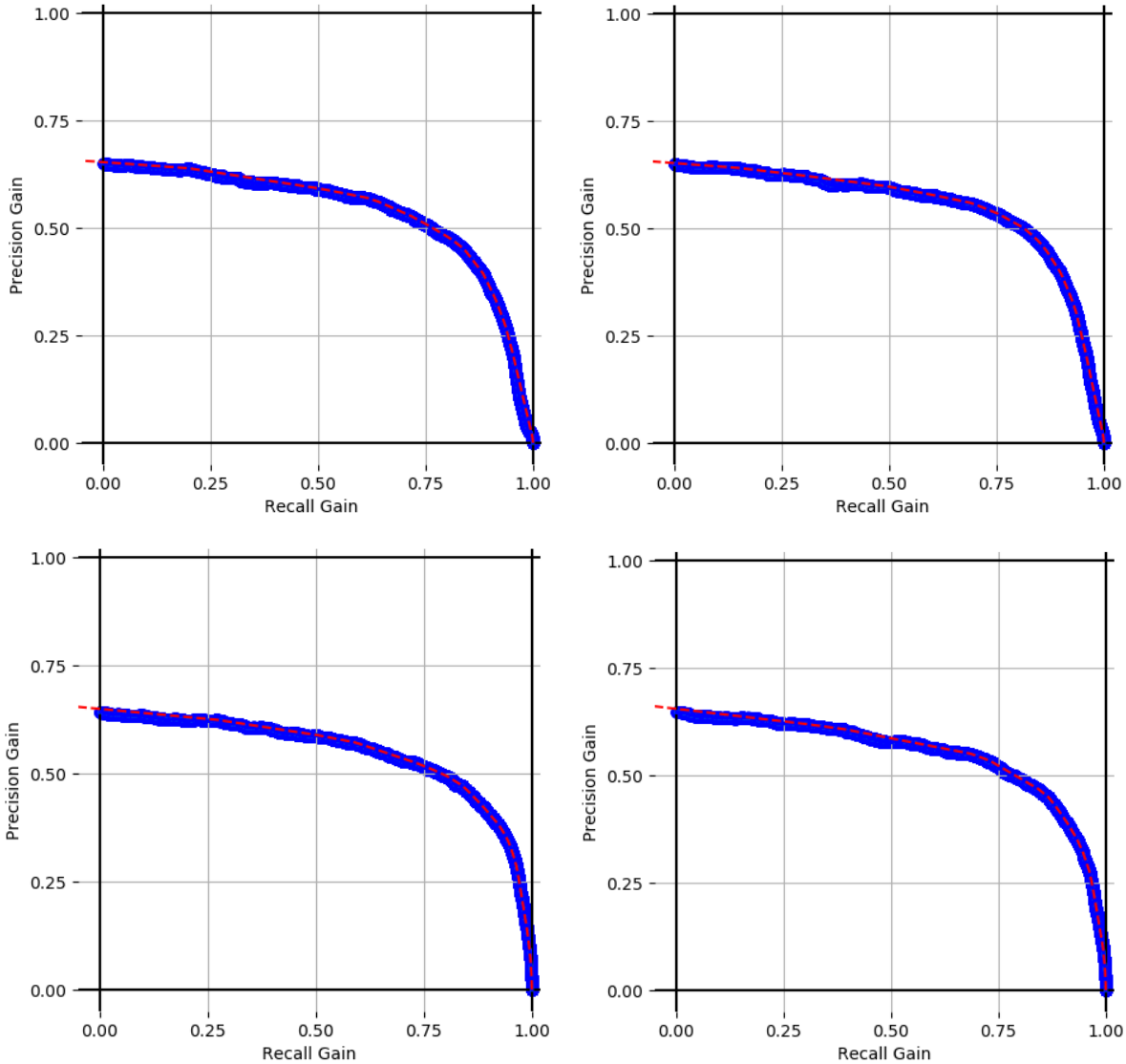
4.1 Featuresets

Als we kijken naar welke feature op zich de beste voorspeller en daarmee het meest veelzeggend is, dan is dat feature 3 (de eerste alinea van het artikel). De score van deze feature is hoger dan die van de andere individuele features en in de top vijf van beste featuresets is feature 3 in drie van de gevallen een element van de featureset. In tabel 2 valt te zien dat de scores bij verschillende featuresets relatief dicht bij elkaar liggen en dat bij andere featuresets de scores relatief verder uit elkaar liggen. Omdat in de gebruikte implementatie van de SVM een kans element zit in de vorm van een *random number generator*, heb ik besloten de beste de beste drie individuele features en de vijf beste gecombineerde featuresets te gebruiken voor verdere evaluatie. Hierdoor is er volgens mij voldoende gecontroleerd voor dit kans element.

	Modelscore	**(N = 7)**	**Fold-scores**	**(N = 63)**
Featureset	Gemiddelde	Standaarddeviatie	Gemiddelde	Standaarddeviatie
(3, 5)	0,529671	0,013779	0,510911	0,060806
(3)	0,528	0,017844	0,511798	0,055205
(4)	0,518303	0,030475	0,497552	0,069969
(4, 5)	0,51039	0,053969	0,496594	0,081323
(2, 3)	0,486372	0,021632	0,460877	0,084059
(2, 3, 5)	0,477995	0,024031	0,458286	0,076802
(2, 4)	0,467547	0,039283	0,443453	0,09108
(2, 4, 5)	0,465846	0,049484	0,444615	0,099303
(2)	0,457542	0,023882	0,443438	0,070255
(2, 5)	0,44771	0,022243	0,434387	0,068954
(1, 2, 3)	0,438535	0,034373	0,424875	0,075656
(1, 2, 4)	0,437856	0,052495	0,422257	0,089883
(1, 2, 3, 5)	0,431558	0,051932	0,42132	0,090009
(1, 2, 4, 5)	0,429097	0,043919	0,417169	0,087741
(1, 3)	0,372614	0,03597	0,360325	0,091863
(1, 2)	0,362575	0,032926	0,348203	0,089232
(1, 2, 5)	0,35867	0,033279	0,347108	0,098456
(1, 4)	0,339773	0,065779	0,329786	0,111221
(1)	-0,06213	0,051589	-0,05514	0,117166
(1, 5)	-0,06716	0,04938	-0,06952	0,115146

Tabel 2: de gemiddelde AUPRG-scores van de SVM-classifier bij het toepassen van PU Learning voor de verschillend featuresets, gesorteerd van de hoogste gemiddelde modelscore naar de laagste.

Wat opvalt is dat bij elk van de featuresets de maximale *precision gain* niet veel hoger komt dan ongeveer 0.70. Dit maximum is vrij consistent tussen alle verschillende featuresets met voor enkele featuresets een lager maximum voor de *precision*.



Figuur 2: de Precision Recall Gain curves voor featureset (3) (linksboven), featureset (3,5) (rechtsboven), featureset (4) (linksonder) en featureset (4,5) (rechtsonder).

4.2 PU tegenover PN

De modellen waarbij geen PU maar PN Learning is toegepast scoren een stuk minder goed dan de modellen waarbij dit wel is toegepast. Alle gemiddelde scores liggen onder de -1. Scores van 0 en lager betekenen dat het model minder goed is in het doen van voorspellingen dan een model dat met een bepaalde kans een datapunt toewijst aan een bepaalde klasse. De precieze waarden vindt u in tabel 3.

Featureset	Modelscore (N = 5) Gemiddelde	Standaarddeviatie	Fold-scores (N = 45) Gemiddelde	Standaarddeviatie
(2)	-1,2475598	0,0383425	-1,35474	0,305188
(3)	-1,2306502	0,028599	-1,32322	0,284508
(4)	-1,3203562	0,0466986	-1,4101	0,308654
(2, 3)	-1,2783281	0,0291079	-1,39206	0,295449
(2, 4)	-1,3072547	0,0339576	-1,41769	0,323043
(3, 5)	-1,2064016	0,0303492	-1,31085	0,299937
(4, 5)	-1,3198007	0,0504927	-1,42349	0,335764
(2, 3, 5)	-1,2654071	0,0284092	-1,38048	0,314746

Tabel 3: de gemiddelde AUPRG-scores van de SVM-classifier voor de acht geselecteerde featuresets wanneer er geen gebruik wordt gemaakt van PU Learning.

4.3 Herlabelen

Het herlabelen lijkt de prestaties van de *classifier* te verbeteren, maar na een bepaalde drempelwaarde nemen de prestaties af. De drempelwaarde die gemiddeld de grootste verbetering oplevert is 0.3. Gemiddeld neemt de score van de *classifier* bij deze drempelwaarde met 0,004 toe als er gekeken wordt naar de modelscores (verbetering van 0,8%) en met 0,008 wanneer er gekeken wordt naar de gemiddelde *fold*-score (verbetering van 1,7%). Wanneer de drempelwaarde ingesteld wordt op 0.4 en hoger treedt er een verslechtering op ten opzichte van drempelwaarde 0.0. In tabel 4 zijn de exacte scores te zien.

Gemiddelde	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
Modelscore	0,4970	0,4812	0,4903	0,5007	0,4760	0,4488	0,4062	0,3318	0,2014
Fold-scores	0,4779	0,4661	0,4745	0,4857	0,4632	0,4377	0,3993	0,3250	0,1923

Tabel 4: gemiddelde scores voor de acht beste featuresets

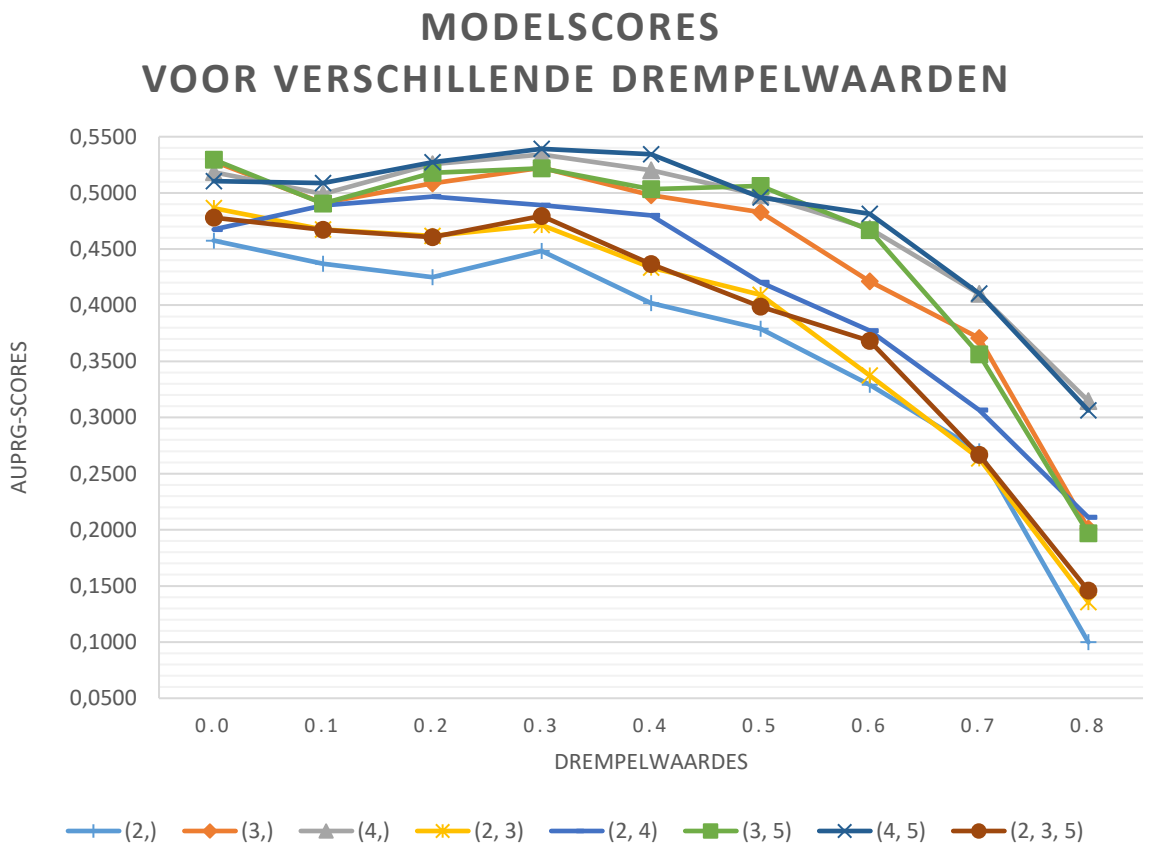
Vooral voor featureset (4), (2,4) en (4,5) levert het instellen van een drempelwaarde verbetering op ten opzichte van de modelscore wanneer er geen gebruik wordt gemaakt van de interessemaat (drempelwaarde 0.0). Andere featuresets hebben of geen profijt van de interessemaat of presteren zelfs slechter.

Featureset		Modelscore voor verschillende drempelwaarden								
	(N = 5)	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
(2)	Gem.	0,4575	0,4370	0,4250	0,4482	0,4018	0,3791	0,3293	0,2698	0,0999
	SD	0,0162	0,0095	0,0217	0,0156	0,0363	0,0302	0,0296	0,0183	0,1050
(3)	Gem.	0,5280	0,4910	0,5083	0,5223	0,4979	0,4829	0,4213	0,3707	0,2010
	SD	0,0158	0,0203	0,0196	0,0278	0,0369	0,0429	0,0952	0,0783	0,1183
(4)	Gem.	0,5183	0,4990	0,5255	0,5341	0,5202	0,4979	0,4682	0,4101	0,3147
	SD	0,0249	0,0474	0,0364	0,0153	0,0414	0,0146	0,0223	0,0582	0,0881
(2, 3)	Gem.	0,4864	0,4675	0,4617	0,4715	0,4335	0,4092	0,3374	0,2637	0,1356
	SD	0,0236	0,0101	0,0350	0,0122	0,0470	0,0342	0,0354	0,0345	0,0913
(2, 4)	Gem.	0,4675	0,4889	0,4967	0,4890	0,4799	0,4203	0,3774	0,3067	0,2110
	SD	0,0488	0,0366	0,0277	0,0196	0,0377	0,0569	0,0396	0,0703	0,0547

(3, 5)	Gem.	0,5297	0,4906	0,5179	0,5218	0,5034	0,5063	0,4668	0,3561	0,1969
	SD	0,0117	0,0266	0,0151	0,0130	0,0459	0,0462	0,0529	0,0477	0,0690
(4, 5)	Gem.	0,5104	0,5088	0,5271	0,5392	0,5343	0,4959	0,4814	0,4104	0,3062
	SD	0,0540	0,0434	0,0302	0,0256	0,0300	0,0147	0,0448	0,0403	0,0561
(2, 3, 5)	Gem.	0,4780	0,4672	0,4605	0,4795	0,4366	0,3986	0,3682	0,2667	0,1458
	SD	0,0236	0,0173	0,0432	0,0184	0,0304	0,0298	0,0536	0,0608	0,0646

Tabel 5: de modelscore voor verschillende drempelwaarden van de interessemaat. Gem. staat voor gemiddelde en SD staat voor standaarddeviatie. De waarden in de kolom van drempelwaarde 0.0 zijn de waarden die gevonden zijn bij het onderzoeken van de featuresets.

Voor een meer grafische weergave van de ontwikkeling van de modelscores ten opzichte van de drempelwaarde, zijn de scores weergegeven in figuur 3.



Figuur 3: de ontwikkeling van de modelscores uitgezet tegen de verschillende drempelwaarden voor de interessemaat.

5. Conclusie

5.1 Beste features

De features die het beste werken voor het aanbevelen van artikelen in de NH app zijn de eerste alinea en de volledige tekst van het artikel. Het blijkt dat alleen de eerste alinea niet slechter is dan de volledige tekst. Dit is voor NH goed nieuws: het kost minder tijd en geheugen om de eerste alinea

om te zetten tot featurevector dan de volledige tekst. Daarnaast blijkt de lengte van de tekst een goede toevoeging te zijn, net als de titel, zij het in mindere mate.

Een verklaring voor de goede werking van de eerste alinea als feature, is dat het in de journalistiek gebruikelijk is de belangrijkste informatie in de eerste alinea te zetten, ook wel de lead genoemd. De rest van het artikel is over het algemeen een uitwerking van de eerste alinea. Dat betekent dat het onderwerp van het artikel uit alleen de eerste alinea al duidelijk kan worden, daarmee de noodzaak wegnemend voor analyse van het volledige artikel. Dit laat zien dat domeinkennis bij het selecteren van features zeer van belang kan zijn.

Dat de scores van de verschillende featuresets dicht bij elkaar liggen is, naar mijn idee, geen indicatie dat er daadwerkelijk weinig verschil zit tussen alle featuresets omdat de scores zijn gebaseerd op een groot aantal datapunten per featureset: 100 gebruikers, 7 training runs met elk 9 *folds* en 1892 artikelen. Zeker wanneer er gekeken wordt naar de standaarddeviatie blijkt dat voor veel featuresets de waarden duidelijk verschillen van ander featuresets.

5.2 Maximum precision gain

De reden dat de *precision gain* van de modellen niet boven de 0.70 uitkomt heeft volgens mij te maken met de interpretatie van de ongelabelde data als negatieve data voor het berekenen van de *precision gain*. Een deel van de ongelabelde data zou eigenlijk als positief gezien moeten worden. Doordat een deel van de als positief geclassificeerde data het verkeerde label heeft, zal de *precision* en daarmee de *precision gain* nooit 1 kunnen worden, zelfs wanneer de *classifier* perfect zou voorspellen. Een effect hiervan is dat een deel van de curve een F1 score heeft die lager ligt dan de F1 score van een *always positive classifier*, zie figuur 2.

Als de grafiek geschaald zou worden naar dit nieuwe bereik van *precision* ontstaat een heel nieuw beeld: de lijnen van de grafieken in figuur 2 lopen dan redelijk dicht langs de bovenrand en rechterrand van de grafiek. De ideale *classifier* heeft de lijn exact langs de bovenrand en rechterrand lopen: met meer *recall* wordt de *precision* niet slechter en vice versa. In het geval dat *precision* en *recall* even belangrijk zijn, is het beste punt voor de balans tussen *precision gain* en *recall gain* het punt waarbij de helling van de lijn -1 is. In deze situatie is dat rond een *recall gain* van 0.80.

5.3 PU tegenover PN

De resultaten van de *classifiers* zonder PU Learning hebben mij zeer verrast. Ik had verwacht dat ze slechter zouden presteren dan de *classifiers* met PU Learning, maar dat het verschil zo groot zou zijn had ik niet vermoed. Uit de AUPRG-scores in tabel 3 is duidelijk dat deze scores ver liggen van de

scores die behaald zijn met PU Learning. Uit deze resultaten blijkt naar mijn idee dat PU Learning hier zeker een toevoeging is.

Een verklaring die ik hiervoor kan bedenken is dat het aantal positieve voorbeelden bij veel van de honderd gebruikers te klein is geweest. Als ondergrens heb ik gesteld dat de gebruikers minimaal 2% van het corpus gelezen moesten hebben. Dit betekent dat er een zeer grote meerderheid van ongelezen en dus negatieve voorbeelden is. Support Vector Machines hebben relatief gelijkmatig verdeelde klassen nodig om goed te kunnen werken. Hierdoor zal de SVM sneller geneigd zijn de positieve voorbeelden toe te wijzen aan de negatieve klasse. Het is de moeite waard om deze test nog een keer uit te voeren maar dan met een groter minimumaantal van gelezen artikelen.

5.4 Herlabelen

Het valt op dat voor sommige featuresets het herlabelen wel een verbetering oplevert en voor andere niet. Daarnaast is het ook opvallend dat de waarden voor een aantal featuresets voor drempelwaarde 0.0 duidelijk hoger liggen dan de waarden voor drempelwaarde 0.1 en 0.2, terwijl er een duidelijke lijn zichtbaar lijkt van stijgende modelscores tussen 0.1 en 0.3.

Een mogelijke verklaring hiervoor is dat voor de drempelwaarde 0.0 de data gebruikt is van het onderzoek naar de beste featureset. Omdat er in de implementatie sprake is van een kans element kan het zijn dat het verschil in scores tussen de scores bij drempelwaarde 0.0 en de andere drempelwaarden voor een deel voortvloeit uit dit kans element. Nader onderzoek zou moeten uitwijzen of dit inderdaad het geval is.

Het herlabelen laat in mijn ogen potentie zien voor de verbetering van *classifiers* die normaal gesproken alleen uitgaan van het openen van een artikel, maar voor de precieze situaties waarin het een significant verschil maakt zijn de moeite van het onderzoeken waard.

6. Discussie & Aanbevelingen

Een belangrijke afweging die gemaakt diende te worden, was hoeveel gelezen artikelen er als minimum moest gelden voor de geselecteerde gebruikers. Idealiter wil je de gebruikers die het meest hebben gelezen omdat daarmee meer informatie beschikbaar is over interesses. Dat zou echter betekenen dat de modellen niet meer representatief zouden zijn voor de gehele groep van gebruikers, waardoor de resultaten lastig zouden kunnen worden toegepast bij de implementatie van het *recommender system* in de app. Aangezien je ook voor gebruikers voorspellingen wil doen die niet heel veel gelezen hebben is het noodzakelijk dat de *classifier* in deze situatie ook met een klein

aantal positieve voorbeelden goede voorspellingen kan doen. Daarom heb ik ervoor gekozen om een minimum van 2% te stellen. Dit kwam neer op een minimum van 38 artikelen in 42 dagen en daarmee op net iets minder dan één artikel per dag. Voor het gebruik van een nieuwsapp lijkt mij het een redelijke aanname dat een gemiddelde gebruiker dit aantal zouden halen.

Verder was een belangrijke afweging welke parameters van de SVM ik zou optimaliseren. Ik heb ervoor gekozen alleen de C parameter enigszins te optimaliseren omdat deze een grote invloed heeft op de resultaten. Omdat dit onderzoek uiteindelijk een vergelijking is tussen verschillende methoden, is het vooral van belang dat de parameters tussen de verschillende methoden gelijk blijven. Welke parameters de beste *classifier* opleveren binnen een methode is een onderzoek op zich waard.

Verder kan dit onderzoek aangevuld worden op verschillende manieren. Het zou goed zijn om specifieke onderdelen uitgebreider te testen: door meer runs van elke featureset te doen kan er duidelijker worden vastgesteld met welke featureset de beste voorspellingen gedaan kunnen worden. De waarden liggen dicht bij elkaar, door meer runs kan de standaarddeviatie omlaag gebracht worden waardoor er met meer zekerheid een ordening aangebracht kan worden. Ook zou de drempelwaarde nog beter onderzocht kunnen worden, zowel met een fijnmaziger bereik van waarden rond de 0.3 als met meer runs om ook hier de standaarddeviatie af te laten nemen.

Aanvullingen met andere algoritmen en methoden zijn ook zeker mogelijk en wenselijk. Er kan bijvoorbeeld ook nog onderzocht worden of met Label Propagation betere resultaten bereikt kunnen worden omdat door het herlabelen van de data van positieve en ongelabelde data, positieve, ongelabelde en negatieve data gemaakt wordt. Daarnaast kan ook onderzocht worden of met regressie voorspellingen gedaan kunnen worden zoals beschreven onder het kopje *Content based recommendation*.

Mijn uiteindelijke aanbevelingen aan NH zijn als volgt: gebruik de inleiding van het artikel als feature om de *content based recommender* op te trainen en herlabel de doelwaarden met 0.3 als drempelwaarde voor de interessemaat. Hiermee geeft de *recommender* de beste resultaten. Wat er dan alleen nog bepaald dient te worden is welke balans tussen *precision* en *recall* gewenst is. Als er gekeken wordt naar alle artikelen waarover voorspeld dient te worden, dan is *recall* belangrijker omdat *precision* nooit 1 zal zijn. Wordt er gekeken naar de artikelen die daadwerkelijk worden aanbevolen, namelijk de top *n* artikelen waarbij de artikelen gesorteerd zijn op de kans dat ze behoren tot de klasse ‘interessant’, dan is *precision* belangrijk: je wil van de top *n* artikelen dat ze allemaal horen tot de klasse ‘interessant’.

Geciteerde werken

Aggarwal, C. C., & Zhai, C. (2012). A survey of text classification algorithms. In *Mining text data* (pp. 163-222). Springer US.

Chen, L., & Pu, P. (2005). Trust building in recommender agents. In *Proceedings of the Workshop on Web Personalization, Recommender Systems and Intelligent User Interfaces at the 2nd International Conference on E-Business and Telecommunication Networks* (pp. 135-145).

Elkan, C., & Noto, K. (2008). Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 213-220). ACM.

Flach, P., & Kull, M. (2015). Precision-recall-gain curves: PR analysis done right. In *Advances in Neural Information Processing Systems* (pp. 838-846).

Forman, G. (2003). An Extensive Empirical Study of Feature Selection Metrics for Text Classification. *Journal of Machine Learning Research, 3*(Mar), 1289-1305.

Garcin, F., Faltings, B., Donatsch, O., Alazzawi, A., Bruttin, C., & Huber, A. (2014). Offline and online evaluation of news recommender systems at swissinfo.ch. In *Proceedings of the 8th ACM Conference on Recommender systems* (pp. 169-176). ACM.

Lu, J., Wu, D., Mao, M., Wang, W., & Zhang, G. (2015). Recommender system application developments: a survey. *Decision Support Systems*(74), 12-32.

Mulder, J., Poort, J., Marlet, G., & van Woerkens, C. (2006). Het economisch belang van reclame. *SEO Economisch Onderzoek/Stichting Atlas voor gemeenten*.

Roodheuvel, R. (Red.). (2016, november 17). *Regionale omroepen wostelen met bezuinigingen*. Opgeroepen op juni 6, 2017, van NVJ: https://www.nvj.nl/nieuws/regionale-omroepen-worstelen-bezuinigingen

Said, A., & Bellogín, A. (2014). Comparative recommender system evaluation: benchmarking recommendation frameworks. In *Proceedings of the 8th ACM Conference on Recommender systems* (pp. 129-136). ACM.

Thada, V., & Jaglan, V. (2013). Comparison of jaccard, dice, cosine similarity coefficient to find best fitness value for web retrieved documents using genetic algorithm. *International Journal of Innovations in Engineering and Technology, 2*(4), 202-205.

Wang, J., & Yuan, S. (2015). Real-time bidding: A new frontier of computational advertising research. In *Proceedings of the Eighth ACM International Conference on Web Search and Data Mining* (pp. 415-416). ACM.

Zhang, W., Yoshida, T., & Tang, X. (2011). A comparative study of TF* IDF, LSI and multi-words for text classification. *Expert Systems with Applications, 38*(3), 2758-2765.

Appendix

	Modelscore	**(N = 7)**	**Alle Folds**	**(N = 70)**		**Zonder Fold 1**	**(N = 63)**	
Featureset	Gemiddelde	Standaarddeviatie	Gemiddelde	Standaarddeviatie	Verschil	Gemiddelde	Standaarddeviatie	Verschil
(1,)	-0,06213	0,05159	-6,54530	26,47951	6,48317	-0,05514	0,11717	-0,00699
(1, 2)	0,36258	0,03293	-5,56787	21,04709	5,93044	0,34820	0,08923	0,01437
(1, 3)	0,37261	0,03597	-4,58816	17,83563	4,96077	0,36033	0,09186	0,01229
(1, 4)	0,33977	0,06578	-4,77957	23,27698	5,11934	0,32979	0,11122	0,00999
(1, 5)	-0,06716	0,04938	-3,91644	16,03298	3,84928	-0,06952	0,11515	0,00236
(1, 2, 3)	0,43853	0,03437	-5,15791	20,95550	5,59645	0,42488	0,07566	0,01366
(1, 2, 4)	0,43786	0,05250	-3,75983	17,27773	4,19768	0,42226	0,08988	0,01560
(1, 2, 5)	0,35867	0,03328	-4,29723	18,56953	4,65590	0,34711	0,09846	0,01156
(1, 2, 3, 5)	0,43156	0,05193	-1,60978	8,15615	2,04134	0,42132	0,09001	0,01024
(1, 2, 4, 5)	0,42910	0,04392	-5,63482	27,18462	6,06392	0,41717	0,08774	0,01193
(2,)	0,45754	0,02388	-3,39795	17,44713	3,85549	0,44344	0,07025	0,01410
(2, 3)	0,48637	0,02163	-0,90829	5,91006	1,39466	0,46088	0,08406	0,02550
(2, 4)	0,46755	0,03928	-2,21515	10,76627	2,68270	0,44345	0,09108	0,02409
(2, 5)	0,44771	0,02224	-5,54961	25,45926	5,99732	0,43439	0,06895	0,01332
(2, 3, 5)	0,47800	0,02403	-1,54994	8,75726	2,02793	0,45829	0,07680	0,01971
(2, 4, 5)	0,46585	0,04948	-2,52296	12,81628	2,98881	0,44461	0,09930	0,02123
(3,)	0,52800	0,01784	-0,45120	4,78709	0,97920	0,51180	0,05520	0,01620
(3, 5)	0,52967	0,01378	-1,61622	8,27606	2,14589	0,51091	0,06081	0,01876
(4,)	0,51830	0,03048	-1,04721	5,63811	1,56551	0,49755	0,06997	0,02075
(4, 5)	0,51039	0,05397	-0,98347	5,01421	1,49386	0,49659	0,08132	0,01380

Tabel 6: de gemiddelde waarden voor de modelscores, de gemiddelde scores over alle folds en de gemiddelde scores over alle folds behalve de eerste. Ter vergelijking zijn ook de standaarddeviatie en het verschil met de gemiddelde modelscores opgenomen.