MASTER'S THESIS

Voorspellen van rechterlijke beslissingen in het Nederlands belastingrecht door middel van Natural Language Processing en Machine Learning

van Amelsvoort, C. (Corbin)

Award date:
2019

Voorspellen van rechterlijke beslissingen in het Nederlands belastingrecht

door middel van Natural Language Processing en Machine Learning

Predicting judicial decisions in Dutch tax law

by Natural Language Processing and Machine Learning

Opleiding:	Open Universiteit, faculteit Management, Science & Technology Masteropleiding Business Process Management & IT
Programme:	Open University of the Netherlands, faculty of Management, Science & Technology Master Business Process Management & IT
Cursus:	IM0602 Voorbereiden Afstuderen BPMIT IM9806 Afstudeertraject Business Process Management and IT
Student:	Corbin van Amelsvoort
Identiteitsnummer:	
Datum:	22-8-2019
Afstudeerbegeleider	Marco Spruit
Meelezer	Stefano Bromuri
Versie nummer:	1.0
Status:	concept

Voorwoord

Voor u ligt de afstudeerscriptie “Voorspellen van rechterlijke beslissingen in het Nederlands belastingrecht”. Deze afstudeerscriptie is geschreven in het kader van mijn afstuderen aan de masteropleiding Business Process Management & IT aan de Open Universiteit en is het resultaat van drie jaar hard werken naast mijn reguliere baan.

Bij deze wil ik graag mijn begeleiders Marco Spruit, Stefano Bromuri en Ramon Ankersmit bedanken voor de ondersteuning tijdens dit traject. Ook wil ik mijn werkgever Topicus bedanken voor de kans, tijd en het vertrouwen.

Abstract

Machine learning technieken zijn de laatste jaren succesvol toegepast voor verschillende toepassingen en in verschillende domeinen. In dit onderzoek wordt er onderzocht of deze technieken toepasbaar zijn voor het voorspellen van rechterlijke beslissingen in het Nederlands belastingrecht. Voorgaand onderzoek laat zien dat het voorspellen van beslissingen in het legal domein mogelijk zijn met Machine Learning technieken. In dit onderzoek worden klassieke Machine Learning en Deep Learning technieken toegepast op jurisprudentie welke vastgelegd zijn in de database van de Rechtspraak.nl. De resultaten van dit onderzoek laten zien dat klassieke Machine Learning technieken de hoogste nauwkeurigheid behalen met een AUC van 0.789. Tenslotte worden de implicaties van de gevonden resultaten besproken.

Sleutelbegrippen

Voorspellen rechterlijke beslissingen, Machine Learning, Natural Language Processing, Support Vector Machine, RNN LSTM, Kunstmatige Intelligentie, belastingrecht, bestuursrecht

Samenvatting

Voorgaande onderzoeken laten zien dat de laatste jaren goede resultaten behaald zijn door Machine Learning technieken in verschillende domeinen. In dit onderzoek is er onderzocht hoe een aantal van deze technieken gebruikt kunnen worden om beslissingen te voorspellen in het Nederlands belastingrecht. Dit probleem is geclassificeerd als een binair classificatie probleem.

Via een experiment wordt aangetoond dat het mogelijk is om belasting rechtszaken van het procedure type ‘eerste aanleg’ te classificeren als ‘gegrond’ en ‘ongegrond’ Een eerste stap naar een complexere juridische bot die voorspellingen kan uitvoeren. De Rechtspraak.nl heeft grote database openbaar gesteld met jurisprudentie van verschillende rechtsgebieden. In het experiment worden twee Machine Learning classifiers vergeleken op basis van deze database. Een klassieke classifier, namelijk Support Vector Machine wordt vergeleken met een Deep Learning techniek, het Recurrent Neural Network. De beide modellen worden geëvalueerd via de AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve . De uitkomst van het experiment toont aan dat de Support Vector Machine met een AUC van 0.798 kan voorspellen of een zaak ‘gegrond’ of ‘ongegrond’ is. Het Recurrent Neural Network laat in dit onderzoek geen voorspellende mogelijkheden zien.

Summary

Previous studies show that good results have been achieved in recent years through Machine Learning techniques in various domains. This study investigated how a number of these techniques can be used to predict decisions in Dutch tax law. This problem is classified as a binary classification problem.

An experiment shows that it is possible to classify ‘first instance’ tax lawsuits as ‘grounded’ and ‘ungrounded’. A first step towards a more complex legal bot that can carry out predictions. The Rechtspraak.nl has made a large database public with cases from different areas of law. In the experiment two Machine Learning classifiers are compared based on the data from this database. A classic classifier, namely Support Vector Machine is compared with a Deep Learning technique, the Recurrent Neural Network. Both models are evaluated via the AUC (Area Under The Curve) ROC (Receiver Operating Characteristics) curve. The outcome of the experiment shows that the Support Vector Machine can predict whether a case is ‘founded’ or ‘unfounded’ with an AUC of 0.798. The Recurrent Neural Network does not show any predictive options in this study.

Inhoudsopgave

1. Introductie

1.1. Inleiding

De recente successen in Kunstmatige intelligentie (KI), zoals het verslaan van de beste menselijke speler in het spel Go door DeepMind's AlphaGo Zero (Silver et al., 2017) en autonoom rijdende voertuigen door onder andere het gebruik van Machine Learning, bieden mogelijk ook kansen in andere gebieden. In voorgaand onderzoek van Aletras et al. (2016) wordt aangetoond dat Natural Language processing en Machine Learning technieken het mogelijk maken om rechtelijke uitspraken van Europees Hof voor de Recht van de Mens te voorspellen met een nauwkeurigheid van 79%. In dit onderzoek wordt er onderzocht of en hoe het gebruik van Natural Language Processing (NLP) en Machine Learning (ML) algoritmen toepasbaar zijn bij het voorspellen van rechterlijke beslissingen in Nederland op basis van jurisprudentie.

In dit hoofdstuk wordt na de inleiding ingegaan op de gebiedsverkenning, aanleiding en de probleemstelling van het onderzoek. In hoofdstuk 2 wordt ingegaan op de theoretische onderzoeksvragen aan de hand van een literatuuronderzoek. In hoofdstuk 3 komt de methode aan bod die gebruikt is bij het uitvoeren van dit onderzoek. Hoofdstuk 4 bevat de resultaten en bevindingen van het onderzoek. Tot slot wordt er in hoofdstuk 5 een conclusie gepresenteerd, antwoord gegeven op de hoofdonderzoeksvraag en worden er aanbevelingen gedaan.

1.2. Gebiedsverkenning

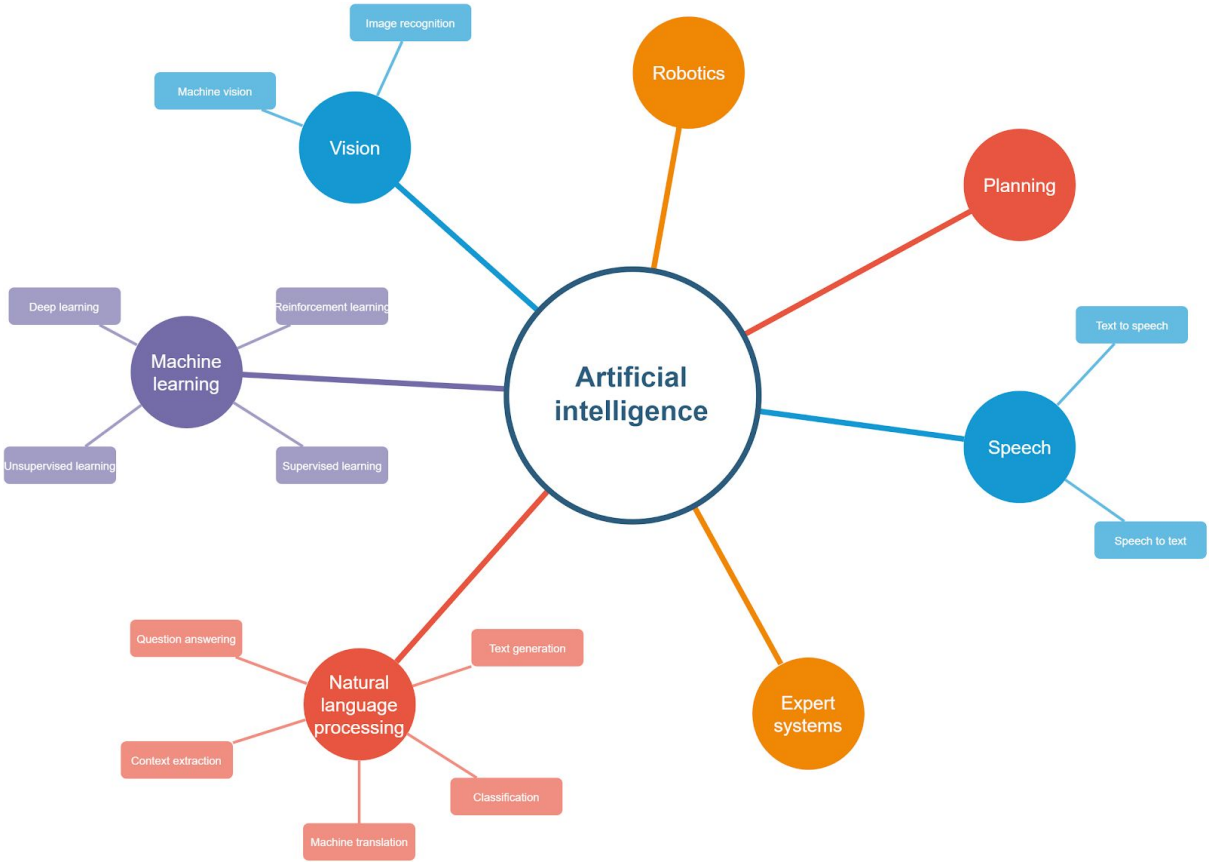
Dit onderzoek vindt plaats binnen de gebieden Machine Learning, Natural Language Processing en de Nederlandse rechtspraak. In deze paragraaf worden bovenstaande gebieden beschreven en de bijbehorende termen uitgelegd.

De rechtspraak in Nederland is opgedeeld in een aantal rechtsgebieden waarvan de belangrijkste zijn: civiel recht, strafrecht en bestuursrecht. Het civiel recht en strafrecht kennen al sinds de negentiende eeuw elk hun eigen vorm van handhaving. Het bestuursrecht heeft zich pas in de loop van de twintigste eeuw ontwikkeld. In art.2 van de Wet op de rechterlijke organisatie uit 1827 is bepaald dat de volgende gerechten tot de rechterlijke macht behoren: rechtbanken, de gerechtshoven en de Hoge Raad der Nederland. In totaal zijn er in Nederland negentien rechtbanken, vijf gerechtshoven en één Hoge Raad. De rechtspraak vindt plaats in enkelvoudige- of meervoudige kamers. In meervoudige kamers hebben drie rechters zitting, een voorzitter en twee gewone rechters. Alleen in grotere strafzaken is volgens art 268 lid 1 Sv rechtspraak een meervoudige kamer hoofdregel. In een enkelvoudige kamer wordt er gebruik gemaakt van een alleensprekende

rechter. De rechtspraak kent verschillende procedures. Als een zaak voor het eerst voor de rechter verschijnt en door hem wordt behandeld en beslist, spreken we van rechtspraak in de eerste aanleg. Deze procedure wordt uitgevoerd in de rechtbank. Als men het niet eens is met de uitspraak, kan er in hoger beroep worden gegaan. Dit wordt gedaan in het gerechtshof. Mocht er bezwaar zijn tegen de uitspraak van het gerechtshof, kan dit bij de Hoge Raad. Deze procedure noemt men cassatie (Verheugt, 2007).

Binnen dit onderzoek wordt er gebruik gemaakt van rechterlijke uitspraken, gedaan door Nederlandse rechters. Deze rechterlijke uitspraken worden in de juridische wereld en binnen dit onderzoek jurisprudentie genoemd. Een deel van deze uitspraken zijn terug te vinden in de database van Rechtspraak.nl. Elk uitspraak in de dataset van de Rechtspraak.nl is gekenmerkt met een uniek identificatienummer, ook wel Landelijk Jurisprudentie Nummer (LJN) genoemd.

Het gebied van kunstmatige intelligentie bevat meerdere gebieden met verschillende doelen en toepassingen, zie figuur 1.1 voor een overzicht van deze gebieden. Dit onderzoek focust alleen op de gebieden Machine Learning (ML) en Natural Language Processing (NLP).



Figuur 1.1 overzicht kunstmatige intelligentie gebieden

Het gebied Machine Learning houdt zich bezig met bouwen van computerprogramma's die zichzelf automatisch verbeteren op basis van ervaring (Mitchell, 1997). Het leren in Machine Learning kan gedefinieerd worden als ”Een computerprogramma leert van ervaring E met betrekking tot een aantal klasse van taken T en prestatiemaatstaf P als de prestatie ervan bij taken in T, gemeten door P, verbetert met ervaring E" (Mitchell, 1997). Binnen het gebied

van Machine learning zijn er verschillende subgebieden, namelijk Deep Learning, Supervised Learning, Unsupervised Learning en Reinforcement Learning.

Deep Learning (DL) is een term dat voor het eerst genoemd is in het onderzoek van Dechter (1986). Het is een ML techniek dat gebruikt maakt van algoritmen die proberen te leren in meerdere niveaus, corresponderend aan de verschillende niveaus van abstractie. De niveaus in deze statistische modellen corresponderen met verschillende concepten, waar concepten op hoog niveau worden gedefinieerd door concepten op een lager niveau. Deze laag niveau concepten kunnen helpen om veel hoog niveau concepten te definiëren (Deng & Yu, 2014). Het eerste echt werkende DL algoritme is gepubliceerd door Ivakhnenko & Lapa (1965). Dit algoritme bevat multilayer perceptrons die via supervised learning binaire classificaties kunnen uitvoeren. Een multilayer perceptron is een kunstmatig neuraal netwerk architectuur. Deep Learning wordt meestal gedaan via kunstmatige neurale netwerken (Deng & Yu, 2014). Een kunstmatig neuraal netwerk is een computationeel model geïnspireerd door de werking van het dierlijk brein (Gerven & Bohte, 2018). Kunstmatige neurale netwerken bieden een algemene, praktische methode voor het leren van real-valued, discrete-valued of vector-valued functies op basis van voorbeelden. Voor bepaalde problemen, zoals het leren om complexe real-world sensor data te interpreteren, zijn kunstmatige neurale netwerken een van de meest effectieve leermethoden die momenteel bekend zijn (Mitchell, 1997).

Supervised Learning (SL) is een manier van leren die gebruikt maakt van voorbeeld input-output paren en zichzelf een functie aanleert die de input aan de output koppelt (Russell & Norvig, 2010). De taak van supervised learning wordt door Russel & Norvig (2010) gedefinieerd als: “Gegeven een trainingsreeks van N voorbeeld invoer-uitvoer paren (x1, y1), (x2, y2),. . . (xN, yN), waarbij elke yj werd gegenereerd door een onbekende functie y = f (x), ontdek een functie h die de ware functie f benadert”. SL kan gebruikt worden om twee soorten problemen op te lossen, namelijk classificatie en regressie. We spreken van een classificatie probleem als de output van het input-output paar bestaat uit een eindige set van waarden, zoals bijvoorbeeld zonnig, regenachtig, bewolkt. Als de output bijvoorbeeld de temperatuur is, een continue waarde, en we willen de temperatuur van morgen voorspellen dan hebben we te maken met een regressie probleem.

Unsupervised Learning (UL) leert niet door een vooraf gedefinieerde training set en labels, zoals SL dit doet. In UL leert het algoritme patronen te herkennen in de input data zonder dat hiervoor expliciet feedback wordt gegeven (Russell & Norvig, 2010). Volgens Russel & Norvig (2010) is de meeste gebruikte UL taak clustering, het vinden van clusters in data.

Reinforcement Learning (RL) gaat in op de vraag hoe een autonome agent kan leren om de optimale acties te kiezen om zijn doel te bereiken (Mitchell, 1997). De agent neemt zijn omgeving waardoor sensoren en acteert hierop door actuatoren (Russell & Norvig, 2010). Door beloningen en straffen leert het algoritme onderscheid te maken tussen acties. AlphaGo Zero is een goed voorbeeld hoe reinforcement learning gebruikt kan worden om een complex probleem op te lossen, zoals het spelen van GO, en hierin te excelleren (Silver et al., 2017).

Natural Language Processing (NLP) is een gebied binnen de kunstmatige intelligentie dat zich bezighoudt met computersystemen die natuurlijke talen proberen te begrijpen en te produceren. De taken van NLP zijn o.a. tekst classificatie (categorisatie), informatie ophalen en information extraheren. De documenten (jurisprudentie) die gebruikt worden in dit onderzoek voor de tekstuele analyse, wordt corpus van documenten genoemd. Corpus is de term die gegeven wordt aan een verzameling van documenten (Russell & Norvig, 2010). Een van de simpelste taal modellen is de waarschijnlijkheidsverdeling van een tekenreeks, dat ook wel het N-gram model wordt genoemd (Russell & Norvig, 2010).

1.3. Aanleiding / relevantie

De rechtspraak in Nederland is al langer bezig met onderzoeken naar mogelijkheden voor kunstmatige intelligentie in de rechtspraak. Dit heeft tot op heden in Nederland nog niet geleid tot een zogeheten “robotrechter“ of nauwkeurige resultaten in het voorspellen van uitspraken (Dag van de Rechtspraak: Computer vs rechter, 2017). Ondanks het teleurstellende resultaat van het experiment van de Dag van de Rechtspraak toont internationaal onderzoek aan dat er wel degelijk mogelijkheden zijn. Het onderzoeksteam van Aletras et al. (2016) heeft aangetoond dat het voorspellen van juridische uitspraken, gedaan door het Europees Hof voor de Rechten van de Mens, voorspeld kan worden met een nauwkeurigheid van 79%. Ook het onderzoeksteam van Katz et al. (2017) toont aan dat het voorspellen van uitkomsten gedaan door het Hooggerechtshof van de Verenigde Staten te voorspellen zijn met een nauwkeurigheid van 70.2%. De uitkomst van het literatuuronderzoek toont echter aan dat het onderzoek van Katz et al. niet geschikt is voor dit onderzoek, omdat de dataset van rechtspraak uit grote ongestructureerde teksten bestaat. Het onderzoek van Aletras et al. (2016) laat veelbelovende resultaten zien en suggereren om vervolgonderzoek te doen met verschillende type data en bronnen. In dit onderzoek is gekozen voor het belastingrecht dat onderdeel is van het rechtsgebied bestuursrecht. Het formaat van een belastingrecht rechtszaak leent zich voor binaire classificatie en beschikt over vergelijkbare features als gebruikt in het onderzoek van Aletras et al. (2016). In “hoofdstuk 3 Methodologie” wordt dieper ingegaan op de keuze voor dit rechtsgebied. Dit onderzoek is tot op heden nog niet uitgevoerd in de Nederlandse rechtspraak en dit unieke onderzoek is alleen daarom al wetenschappelijk relevant. Het is een toevoeging aan het huis van de wetenschap.

De resultaten, mits positief, zijn van belang voor de rechtspraak in Nederland. De rechtspraak in Nederland heeft in 2017 een economisch zwaar jaar gehad en uit het jaarverslag blijkt dat 2017 het eerste jaar is dat is afgesloten met een negatief eigen vermogen (Bakker, 2017). Oorzaken hiervoor zijn het afnemen van het aantal rechtszaken, digitalisering en de kosten van de rechtspraak (Bakker, 2017). In het jaarverslag wordt aangegeven dat één van de mogelijke oorzaken voor het afnemen van rechtszaken is dat de hoge kosten van griffierechten en het starten van een procedure een afschrikwekkende werking hebben. Dit onderzoek kan hiervoor van belang zijn. Door het gebruik van kunstmatige intelligentie zou een jurist, advocaat of rechter potentieel efficiënter zijn werk kunnen doen wat als gevolg kan hebben dat de kosten van een rechtszaak lager worden.

1.4. Probleemstelling

Zoals hierboven is beschreven, laat eerder onderzoek van Aletras et al. (2016) en Katz et al. (2017) veelbelovende resultaten zien. In dit onderzoek wordt er onderzocht of het voorspellen van juridische beslissingen binnen de Nederlandse rechtspraak ook mogelijk is met ML en NLP. De probleemstelling van dit onderzoek is: “In hoeverre is het mogelijk om juridische beslissingen op basis van jurisprudentie te voorspellen in het Nederlands belastingrecht met behulp van Machine Learning en NLP?”.

1.5. Opdrachtformulering

De doelstelling van dit onderzoek is aantonen dat het voorspellen van juridische beslissingen in het Nederlands belastingrecht met behulp van Machine Learning en NLP mogelijk of niet mogelijk is. Om een antwoord te geven op de centrale onderzoeksvraag “In hoeverre is het mogelijk om juridische beslissingen op basis van jurisprudentie te voorspellen in het Nederlands belastingrecht met behulp van Machine Learning en NLP?” zijn de volgende deelvragen opgesteld:

- Wat is een geschikte analysetechniek voor het voorspellen van juridische beslissingen in het Nederlands belastingrecht?
- Hoe is de prestatie van de analysetechniek te meten?
- Wat is de nauwkeurigheid van de analysetechniek?

1.6. Aanpak in hoofdlijnen

In het vervolg van dit onderzoek wordt door middel van een literatuurstudie het theoretisch kader bepaald. De literatuurstudie geeft inzicht in de laatste ontwikkelingen van ML, NLP en voorgaand onderzoek over de toepassing hiervan in de juridische wereld. Op basis van de gevonden informatie in de literatuurstudie is de gebruikte methodiek en opzet van het model bepaald. Via een computationeel experiment is de performance van het model geanalyseerd. In hoofdstuk 4 zijn de resultaten van het prototype model te vinden. In hoofdstuk 5 zijn vervolgens conclusies getrokken op basis van de resultaten.

2. Theoretisch kader

In dit hoofdstuk wordt het theoretisch kader bepaald door middel van een literatuurstudie. In de volgende paragrafen worden de onderzoeksaanpak, uitvoering, resultaten en conclusies en het doel van vervolgonderzoek beschreven. Het literatuuronderzoek heeft geleid tot resultaten en conclusies. Op basis hiervan is het doel van het vervolgonderzoek gedefinieerd.

2.1. Onderzoeksaanpak

Het doel van deze literatuurstudie is inzicht krijgen in de laatste ontwikkelingen van ML, NLP en voorgaand onderzoek ten behoeve van de toepassing hiervan in de juridische wereld. Dit inzicht helpt mee in het selecteren van een architecturale opzet voor het voorspellende algoritme, verhoogt de interne validiteit en de uniekheid van dit onderzoek. Het literatuuronderzoek is uitgevoerd in vier stappen, het definiëren van vragen, de zoekstrategie, filteren van artikelen en de uitwerking.

De onderzoeksvragen geven antwoord op de mogelijke algoritmen die toegepast kunnen worden voor het voorspellen van rechterlijke beslissingen. Daarnaast wordt de huidige staat van ML en NLP in de Nederlandse rechtspraak onderzocht. Op basis hiervan zijn de volgende concrete vragen opgesteld en onderzocht:

1. Welke algoritmen zouden geschikt kunnen zijn voor binaire classificatie op basis van tekst?
2. Zijn er voorspellende algoritmen ontwikkeld en toegepast in de juridische wereld?
3. Wordt er in de Nederlandse rechtspraak gebruik gemaakt van ML en NLP om beslissingen te voorspellen?

Om de juiste bronnen te vinden die gebruikt kunnen worden voor het beantwoorden van de onderzoeksvragen zijn er een meervoud aan zoekcriteria opgesteld. Deze zoekcriteria zijn te vinden in onderstaande tabel 2.1. Voor Machine Learning, Kunstmatige Intelligentie en Natural Language Processing zijn zowel de volledige naam als de afkorting gebruikt in de zoekcriteria. De zoekcriteria zijn toegepast op de zoekmachine van Google Scholar (https://scholar.google.nl) en de digitale bibliotheek van de Open Universiteit (http://bibliotheek.ou.nl/). De zoekopdrachten in de digitale bibliotheek en Google Scholar worden gelimiteerd op artikelen met een publicatiedatum niet ouder dan drie jaar. Dit is gedaan omdat ML en NLP zich razendsnel ontwikkelingen en er state-of-the-art technologisch onderzoek benodigd is.

Vraag	**Zoekcriteria**
Welke algoritmen zouden geschikt kunnen zijn voor binaire classificatie op basis van tekst?	‘binary classification text’, ‘binary classification nlp’, ‘nlp machine learning classification’, ‘nlp text classification’, ‘text classification’

Zijn er algoritmen ontwikkeld die uitspraken voorspellen en toegepast zijn in de juridische wereld?	‘prediction judicial decisions’, ‘prediction judicial decision’, ‘predicting judicial decisions’, ‘predicting judicial decision’, ‘voorspellen rechterlijke uitspraken’, ‘voorspellen rechterlijke uitspraak’, ‘predicting court decisions’, ‘predicting court decision’, ‘prediction court decisions’, ‘prediction court decision’.
Wordt er in de Nederlandse rechtspraak gebruik gemaakt van ML en NLP om uitspraken te voorspellen?	‘rechtspraak ML’, ‘rechtspraak NLP’ , ‘rechtspraak AI’, ‘voorspellen uitspraken’, ‘voorspellen juridische uitspraken’, ‘voorspellen juridische uitspraak’, ‘voorspellen rechterlijke uitspraken’, ‘voorspellen rechterlijke uitspraak’, ‘predicting judicial decision rechtspraak’, ‘predicting judicial decision Netherlands’

Tabel 2.1 zoekcriteria

De zoekstrategie is als volgt, per zoekcriteria worden de onderstaande stappen doorlopen:

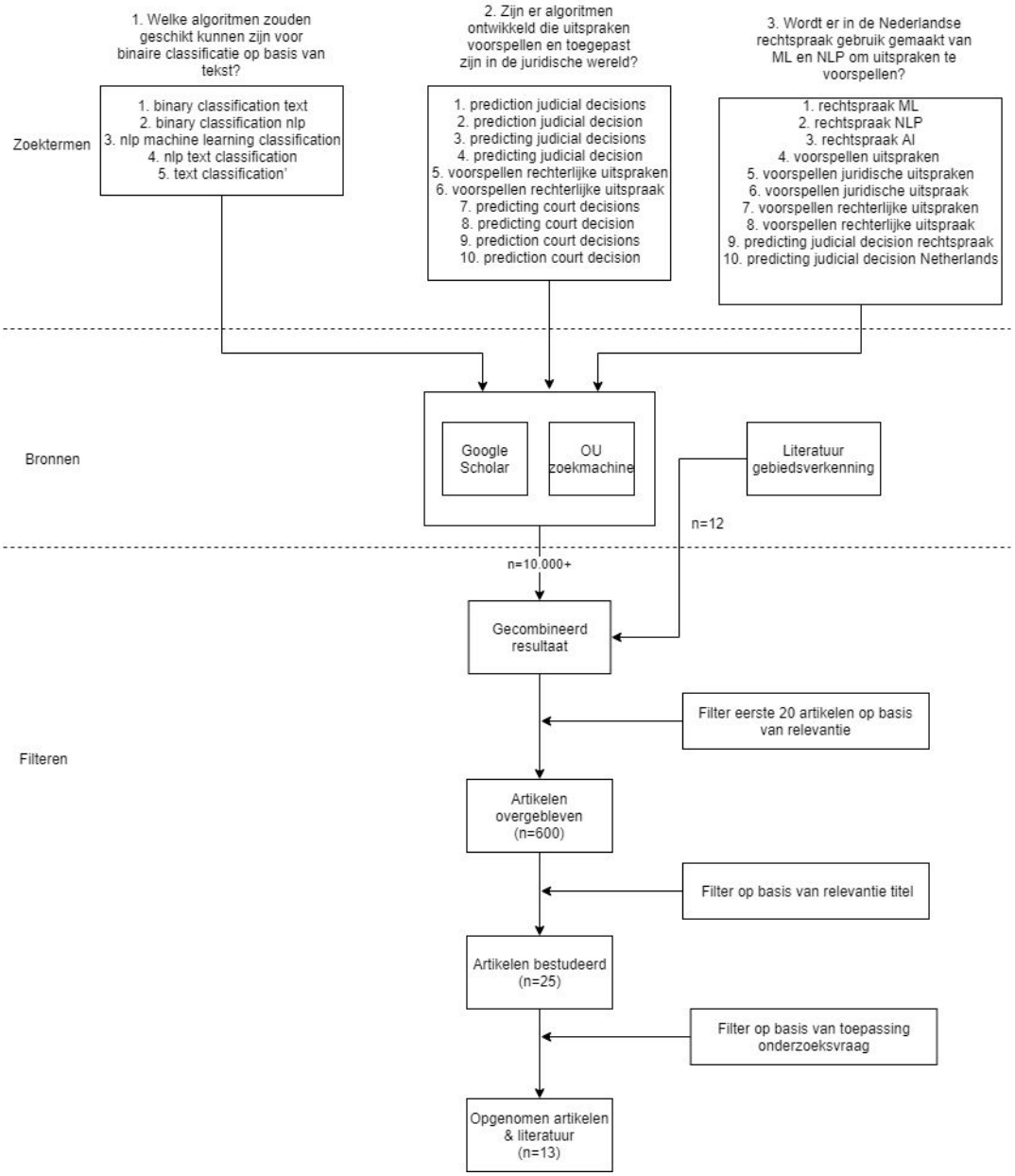
1. de zoekcriteria wordt toegepast op de zoekmachine van de OU en Google Scholar;
2. dit levert een n aantal artikelen op;
3. van elk artikel wordt de titel gelezen;
4. als er een vermoeden is op basis van de titel dat het artikel zou kunnen bijdragen aan het beantwoorden van de onderzoeksvraag, dan wordt vervolgens de abstract gelezen;
5. wanneer het artikel relevant lijkt te zijn op basis van de abstract wordt het eerste hoofdstuk van het artikel gelezen;
6. vervolgens wordt het hele artikel gelezen als voorgaande stappen aantonen dat het artikel relevant is voor het onderzoek.

.

2.2. Uitvoering

Tijdens het verzamelen van artikelen voor de eerste onderzoeksvraag werd geconstateerd dat de zoekcriteria te veel artikelen opleveren. De zoekcriteria behorend bij de eerste onderzoeksvraag leverden per criteria meer dan tienduizend artikelen op. Er is toen gekozen om voor het vervolg van het literatuuronderzoek per zoekcriteria de eerste twintig artikelen, gesorteerd op relevantie door zoekmachine, te gebruiken. Uiteindelijk heeft deze zoekstrategie in totaal geresulteerd in een zoekresultaat van zeshonderd artikelen, waarvan driehonderd in de digitale bibliotheek van de Open Universiteit en driehonderd in Google Scholar. De zoektermen hebben veel dezelfde artikelen opgeleverd waardoor het grote aantal artikelen snel gefilterd kon worden. Voor het beantwoorden van de literatuur onderzoeksvragen zijn van vijfentwintig artikelen de abstract gelezen en zijn er uiteindelijk acht artikelen gebruikt.

De zoekcriteria gebruikt voor de eerste onderzoeksvraag in combinatie met artikelen en literatuur vanuit de gebiedsverkenning fase, hebben geleid tot 9 artikelen. De artikelen van Breiman (2001), Joachims(1998, 2002) en Wang & Manning, 2012) gerefereerd in het onderzoek van Katz et al. (2017) en Aletras et al. (2016) zijn gebruikt voor het vinden van eigenschappen van de gevonden algoritmes. De tweede onderzoeksvraag is beantwoord door twee gevonden artikelen van Aletras et al. (2016) en Katz et al. (2017). Voor het beantwoorden van de derde onderzoeksvraag zijn ook artikelen gebruikt uit de gebiedsverkenning fase, omdat de zoekcriteria geen artikelen opleverde die relevant zijn voor dit onderzoek. Zie figuur 2.2 voor de schematische weergave van het selectieproces.



Figuur 2.2 filteren literatuur

2.3. Resultaten en conclusies

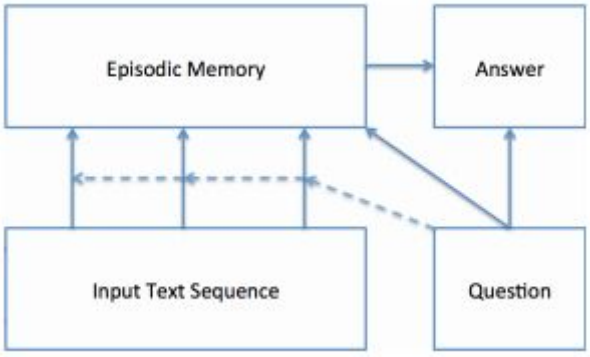
2.3.1 Welke algoritmen zouden geschikt kunnen zijn voor binaire classificatie op basis van tekst?

Een van de eenvoudigste maar ook een van de meest succesvolle vorm van machine learning is de decision tree (Russel & Norvig, 2010). Een decision tree is een weergave van een functie die op basis van een vector van attributen een enkele waarde teruggeeft (Russel & Norvig, 2010). De decision tree is een van de meeste populaire algoritmes voor inductieve gevolgtrekking en is onder andere succesvol toegepast in het diagnosticeren van medische zaken en het bepalen van kredietrisico bij leningen (Mitchell, 1997). Problemen die het meest geschikt zijn voor een decision tree hebben de volgende karakteristieken: de input value is een key-value pair, de output value is een binaire classificatie en de training set mag fouten bevatten (Mitchell, 1997). Een negatief punt van een decision tree is dat deze gevoelig is voor overfitting (Mitchell, 1997). Een random forest is een alternatief dat gebruikt maakt van meerdere decision trees om classificaties uit te voeren. Door de wet van grote aantallen heeft een random forest geen last van overfitting (Breiman, 2001). Een random forest wordt door Breiman (2001) gedefinieerd als: “Een random forest is een classifier die bestaat uit een verzameling boom gestructureerde classifiers (x, Θk), k = 1, ...} waarbij de {Θk} onafhankelijke identiek verdeelde willekeurige vectoren zijn en elke boom een eenheid stem voor de meest populaire klasse op ingang x” (p. 2). Door arbitraire input en arbitraire features worden goede resultaten behaald in classificatie en dit maakt de random forest een accurate classifier (Breiman, 2001).

Als een data scientist geen specialisatie kennis van het domein heeft is het support vector machine (SVM) framework de meest populaire manier voor “off-the-shelf” supervised learning (Russell & Norvig, 2010). Een support vector machine is een supervised learning algoritme dat binaire classificatie kan uitvoeren en geschikt is voor tekst classificatie, classificatie van afbeeldingen en het herkennen van handgeschreven karakters. SVMs hebben meerdere eigenschappen die ze aantrekkelijk maken:

- ze generaliseren goed, doordat de SVM een maximum margin separator construeert (Russell & Norvig, 2010);
- ze creëren een linear separatinghyperplane, maar kunnen ook data onderbrengen in een hogere dimensionale ruimte, de zogeheten kernel trick. De originele data is vaak niet lineair te scheiden en in een hogere dimensie is dit vaak makkelijker (Russel & Norvig, 2010);
- ze maken gebruik van de voordelen van de parametrische en niet-parametrische methode, waardoor ze flexibel zijn om complexe functies weer te geven, maar resistent zijn voor overfitting (Russel & Norvig, 2010);
- ze kunnen goed omgaan met veel input features door de resistentie voor overfitting (Joachims, 1998);
- ze zijn hebben goede resultaten laten zien in tekstclassificaties met kleinere datasets (Aletras et al, 2016; Joachims, 2002; Wang & Manning, 2012).

In onderzoek gedaan door Kumar et al. in 2016 wordt een neuraal netwerk architectuur voorgesteld die op basis van input en vragen, episodische herinneringen, een antwoord kan genereren. Dit model wordt het dynamic memory network (DMN) genoemd. Door het gebruik van herinneringen kan er geredeneerd worden over meerdere zinnen (Kumar et al., 2016) en leert het model van vorige iteraties. Het model bestaat uit vier modulen (zie figuur 2.3), een input module, vraag module, episodische herinnering module en antwoord module.



figuur 2.3 overzicht DMN modules (Kumar et al., 2016)

Het DMN is getest via de bAbI test dataset van Facebook en de resultaten zijn vergeleken met een traditioneel memory netwerk. De resultaten laten zien dat het model over de twintig verschillende taken uit de test dataset een hoger gemiddeld percentage, 93.6 %, heeft dan het traditionele memory network met 93,3 % (Kumar et al., 2016).

Een vorm van ML die recentelijk veel goede resultaten laat zien in NLP tegenover de klassieke vormen van ML zijn Recurrent Neural Networks (RNNs) (Menger et al., 2018). Dit heeft grotendeels te maken met de introductie van word2vec (Mikolov et al, 2013) en paragrah2vec (Le et al., 2014) algoritmen voor het leren van representaties van teksten (Menger et al., 2018). Menger et al. (2018) laten zien dat de RNN succesvol is toegepast op gestructureerde en ongestructureerde data en teksten in het elektronisch patiënten dossier in Nederland. RNNs, specifiek gebaseerd op Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) bereiken state-of-the-art resultaten op basis sequentiële data van verschillende sequentiële lengten. Toepassing van dit model zijn de omgang met NLP, image captioning, handschriftherkenning en genomische analyse (Lipton et al., 2015). Het onderzoek van Menger et al. (2018) gebruikt de Area Under Curve (AUC) of the Receiver Operator Curve om de verschillende ML technieken te vergelijken.

2.3.2 Zijn er algoritmen ontwikkeld die beslissingen voorspellen en toegepast zijn in de juridische wereld?

Er zijn onderzoeken gevonden waarin ML wordt gebruikt om voorspellingen te doen in de juridische wereld. Dit zijn de onderzoeken van Aletras et al. (2016) en Katz et al. (2017).

Het onderzoeksteam van Aletras et al. (2016) heeft een combinatie van NLP en ML gebruikt om de voorspellingen te doen. De onderzoeksopzet is als volgt, de dataset bestaat uit 584 zaken gerelateerd aan Artikel 3,6 en 8 van de Europese Conventie. De tekst in de dataset

zijn omgezet naar kleine letters en vervolgens zijn stopwoorden verwijderd. Per zaak worden tekstuele features geëxtraheerd uit verschillende (sub)secties. Deze features zijn N-gram features en woordclusters om onderwerpen te vinden. Er wordt gebruikt gemaakt van het BOW model (Salton, Wong & Yang, 1975; Salton & McGill, 1986) wat een populaire semantische representatie is van tekst in NLP (Aletras et al., 2016). Een document in het BOW model wordt gerepresenteerd als een multiset van unigrams of N-grams zonder rekening te houden met grammatica, syntaxis en woordvolgorde (Aletras et al., 2016). Dit resulteert in een vector space representatie waar documenten gerepresenteerd zijn als m-dimensionale variabelen over een set van m N-grams (Aletras et al., 2016). Voor elke zaak in de dataset worden de top-2000 meest voorkomende N-grams berekend, waar $N \in \{1,2,3,4\}$. Elke feature representeert de genormaliseerde frequentie van een N-gram in een zaak of een sectie van de zaak. Dit kan kan beschouwd worden als de feature matrix, $C \in R^{c \times m}$, waar c het aantal zaken is en m = 2000 (Aletras et al., 2016). Deze berekende N-grams dienen vervolgens als input om de onderwerpen te vinden door deze te clusteren en gebruik te maken van de verdelingshypothese die suggereert dat vergelijkbare woorden verschijnen in vergelijkbare contexten (Aletras et al., 2016). De C feature matrix is een distributie representatie (Turney & Pantel, 2010) van de N-grams waar de zaak als context dient en elke kolom vector van de matrix een N-gram representeert (Aletras et al., 2016). Via deze vector representatie van woorden kan N-gram gelijkenis worden berekend door gebruikt te maken van de cosinus maat en kan er een N-gram per N-gram gelijkheidsmatrix worden gemaakt (Aletras et al., 2016). Op deze gelijkheidsmatrix wordt vervolgens spectral clustering (von Luxburg, 2007) toegepast om dertig clusters van N-grams te verkrijgen. De verkregen onderwerpen zijn harde clusters waardoor een N-gram alleen deel kan uitmaken van één onderwerp. Een representatie van een cluster is afgeleid door te kijken naar de meeste voorkomende N-gram in het cluster (Aletras et al., 2016). Het belangrijkste voordeel voor het gebruik van onderwerpen (een set van N-grams) in plaats van enkele N-grams is dat het de dimensionaliteit van de feature space reduceert. Dit is essentieel voor feature selectie, omdat dit overfitting van de trainingsdata limiteert (Lampos et al., 2014; Preoțiuc-Pietro, Lampos & Aletras, 2015; Preoțiuc-Pietro et al., 2015; Aletras et al., 2016). Deze N-grams en onderwerpen worden vervolgens gebruikt om een SVM classifiers te trainen via een lineaire kernel functie. In elke zaak waar er sprake is van een overtreding wordt deze gelabeld met +1 en elke zaak zonder overtreding met -1. Hierdoor worden features met positieve gewichten geassocieerd met een overtreding en negatieve gewichten met zaken zonder overtreding (Aletras et al., 2016). De linear SVM heeft een regularisatie parameter welke getuned is via grid-search. Voor Artikel 6 en 8 wordt de data van Artikel 3 gebruikt voor parameter tuning en voor Artikel 3 wordt de data van Artikel 8 gebruikt. Via 10-voudige cross-validation is het model getest en dit levert een gemiddelde nauwkeurigheid op van 79 % zoals is te zien in figuur 2.3.

Feature Type		Article 3	Article 6	Article 8	Average
N-grams	Full	.70 (.10)	.82 (.11)	.72 (.05)	.75
	Procedure	.67 (.09)	.81 (.13)	.71 (.06)	.73
	Circumstances	.68 (.07)	.82 (.14)	.77 (.08)	.76
	Relevant law	.68 (.13)	.78 (.08)	.72 (.11)	.73
	Facts	.70 (.09)	.80 (.14)	.68 (.10)	.73
	Law	.56 (.09)	.68 (.15)	.62 (.05)	.62
Topics		**.78 (.09)**	.81 (.12)	.76 (.09)	.78
Topics and circumstances		.75 (.10)	**.84 (0.11)**	**.78 (0.06)**	**.79**

figuur 2.3 nauwkeurigheid van verschillende features per artikel (Aletras et al., 2016)

Het onderzoeksteam van Katz et al. (2017) gebruikt in plaats van een SVM een random decision forest. Kat et al. (2017) maken gebruik van data uit de Supreme Court Database (SCDB). Deze dataset bevat meer dan tweehonderd jaar aan hoge kwaliteit en door expert gecodeerde data (Kat et al., 2017). Elke zaak in de dataset kan wel tot tweehonderdveertig variabelen bevatten. De structuur van de data is verschillend, namelijk gestructureerde data door middel van key-value paren, in tegenstelling tot de gebruikte dataset in het onderzoek Aletras et al. (2016) waar ongestructureerde grote stukken tekst worden gebruikt. De onderzoeksopzet is als volgt: het onderzoeksteam heeft features geëxtraheerd uit de dataset en heeft zelf features gecreëerd. Er worden verder geen specifieke preprocessing stappen uitgevoerd op de dataset. De random forest wordt geconstrueerd door statische diverse trees. Deze trees worden gemaakt door bootstrap aggregatie toe te passen op arbitraire subsets van de training dataset. Het onderzoek van Kat et al. (2017) laat zien dat voor deze toepassing en dataset een random forest betere resultaten behaald dan een support vector machine en multilayer perceptron. Een F1 score is gebruikt om de nauwkeurigheid van het algoritme te meten en dit levert een nauwkeurigheid op van gemiddeld 70,2%.

2.3.3 Wordt er in de Nederlandse rechtspraak gebruik gemaakt van ML en NLP om beslissingen te voorspellen?

Er is geen wetenschappelijke literatuur gevonden waarin ML en NLP worden gebruikt om voorspellingen te doen in Nederlandse rechtsgebieden.

De rechtspraak in Nederland heeft in 2017 samen met het bedrijf LexIQ een experiment uitgevoerd waarin ML is gebruikt om een vonnis te maken op basis van eerdere uitspraken (Dag van de Rechtspraak: Computer vs rechter, 2017). Deze uitspraken zijn vervolgens vergeleken met de echte uitspraken en dit leverde tot nu toe geen goede resultaten op. In het experiment zijn twee zaken behandeld waarin het algoritme in de eerste zaak het verkeerde vonnis uitsprak. In de tweede zaak is wel in grote lijnen dezelfde uitspraak gedaan, echter is in het vonnis de vergoeding tien keer zo laag als wat de echte rechter heeft toegekend (Dag van de Rechtspraak: Computer vs rechter, 2017). Er is geen informatie gevonden over de algoritmen en technieken die zijn toegepast in het experiment. Prins & Roest (2018) geven aan dat een ‘robotrechter’ voor de afdoening van grote hoeveelheden standaardzaken aanstaande lijkt.

2.3.4 Conclusie

Het literatuuronderzoek laat zien dat er binnen de rechtspraak in Nederland nog geen gebruik wordt gemaakt van ML en NLP om rechterlijke beslissingen te voorspellen. Onderzoekers Prins & Roest (2018) geven desondanks aan dat de robotrechter aanstaande lijkt en recent buitenlands onderzoek van Aletras et al. (2016) en Katz et al. (2017) tonen veelbelovende resultaten in het voorspellen van uitspraken. De rechtspraak in Nederland heeft in combinatie met LexIQ geëxperimenteerd met ML, maar tot op heden geen goede resultaten behaald. Er is echter geen informatie te vinden over de manier waarop ML is toegepast om de uitspraken te voorspellen. Het literatuuronderzoek bevestigt dat er nog geen wetenschappelijk onderzoek is gedaan naar de toepassing van ML en NLP in de Nederlandse rechtspraak.

Het toegepaste algoritme van Katz et al. (2017) lijkt voor dit onderzoek niet een geschikt algoritme, omdat de dataset van de Nederlandse rechtspraak uit ongestructureerde grote stukken tekst bestaat en niet uit key-value paren. De decision tree en random decision forest algoritmen zijn het meest geschikt voor data bestaand uit key-value paren (Mitchell, 1997). De SVM lijkt een geschikt algoritme, omdat de dataset van de Rechtspraak uit grote stukken tekst bestaat, hierdoor zijn er veel features noodzakelijk en een SVM is geschikt voor veel features (Joachims, 1998). Ook is dit algoritme gebruikt in het onderzoek van Aletras et al. (2016). De RNN met een LSTM laag zou een geschikt algoritme kunnen zijn, omdat deze techniek om kan gaan met ongestructureerde tekst en dit andere gebieden heeft aangetoond (Menger et al., 2018; Lipton et al., 2015), De RNN en SVM lijken de meest geschikte algoritmen op basis van het literatuuronderzoek, omdat dit eerder is toegepast in andere onderzoeksgebieden. Een alternatief, dat mogelijk ook gebruikt kan worden, is het Dynamic Memory Network (DMN) ontwikkeld door Kumar et al. (2016). Er zijn geen artikelen gevonden waarin dit netwerk wordt toegepast in de juridische wereld, maar het lerende vermogen van dit netwerk en het redeneren over meerdere zinnen kan potentieel een geschikt algoritme zijn (Kumar et al., 2016). Omdat een DMN niet eerder is toegepast in vergelijkbare onderzoeken wordt deze niet meegenomen in dit onderzoek. In tabel 2.4 is het overzicht van de gevonden algoritmes te vinden.

Algoritme	**Mogelijkheid redeneren over ongestructureerde teksten**	**Eerder toegepast in het juridisch domein**	**Geschikt**
Decision tree	Nee	Ja	Nee
Random forest	Nee	Ja	Nee
SVM	Ja	Ja	Ja
DMN	Ja	Nee	Nee
RNN - LSTM	Ja	Nee	Ja

Tabel 2.4 overzicht gevonden algoritmes

2.4. Doel van het vervolgonderzoek

Het doel van het vervolgonderzoek is onderzoeken of het voorspellen van rechterlijke beslissingen in het Nederlands belastingrecht mogelijk is. De conclusie van het literatuuronderzoek toont aan dat er geen wetenschappelijk onderzoek is gedaan naar het voorspellen van rechterlijke beslissingen in Nederland. Het onderzoek van Aletras et al. (2016) toont aan dat het voorspellen mogelijk is, echter is dit geen garantie dat dit ook mogelijk is in de Nederlandse rechtspraak.

3. Methodologie

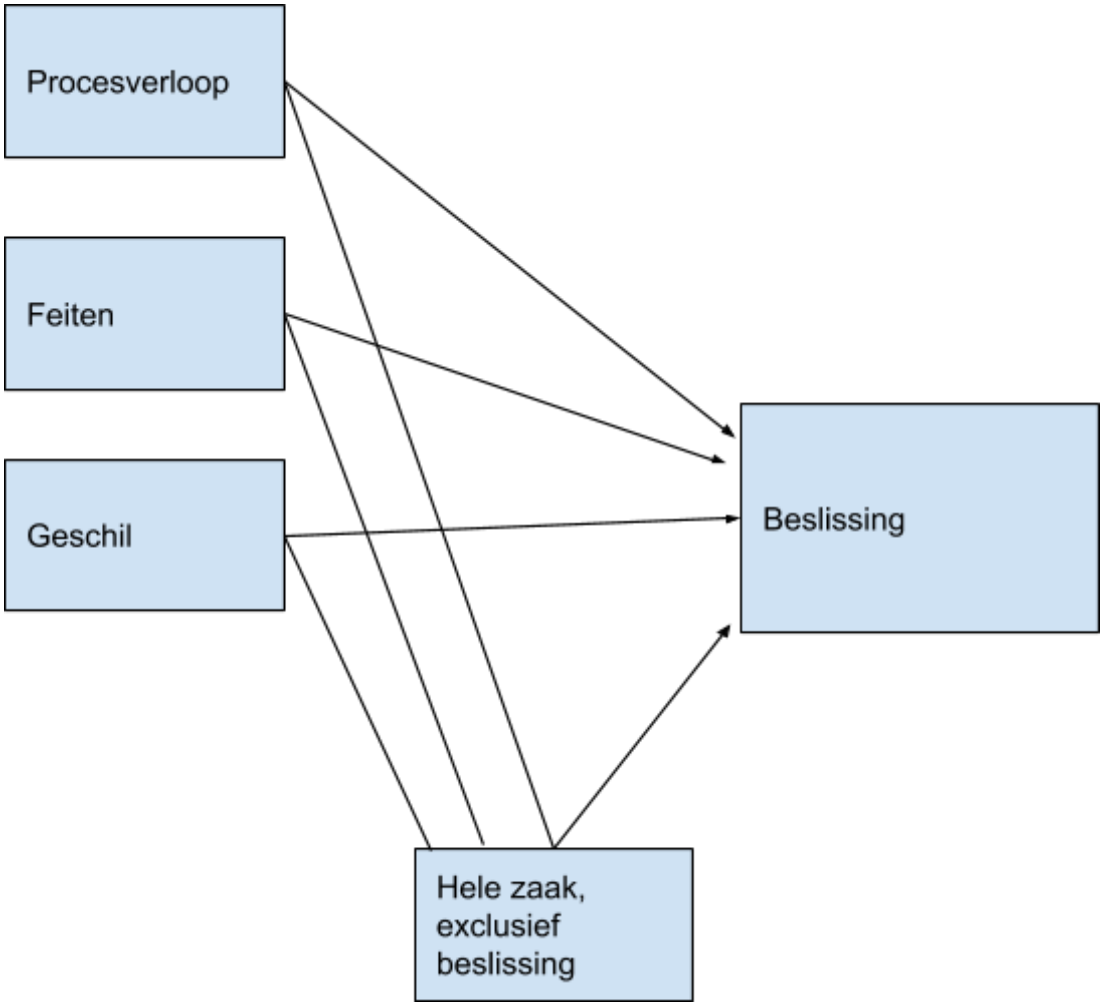
In dit hoofdstuk wordt de gebruikte methodologie beschreven door middel van een conceptueel ontwerp, technisch ontwerp en gegevensanalyse. Tevens wordt er gereflecteerd op de validiteit, betrouwbaarheid en ethische aspecten van het onderzoek.

3.1. Conceptueel ontwerp: keuze van onderzoeksmethode(n)

Het onderzoek is opgedeeld in drie deelvragen die gezamenlijk antwoord geven op de hoofdonderzoeksvraag. In deze paragraaf wordt per deelvraag onderbouwd wat er gedaan wordt en waarom.

De deelvragen geven antwoord op welke analysetechniek gebruikt kan worden voor het voorspellen van beslissingen, hoe de prestatie te meten is en hoe nauwkeurig de analysetechniek is. De informatie noodzakelijk voor het beantwoorden van deelvragen “Wat is een geschikte analysetechniek voor het voorspellen van beslissingen” en “Hoe is de prestatie van de analysetechniek te meten” is via literatuuronderzoek verkregen. Dit vormt de basis voor de beantwoording van de laatste deelvraag “Wat is de nauwkeurigheid van de analysetechniek”. In dit deel van het onderzoek wordt de gekozen analysetechniek beschrevenen wordt er een computationeel experiment opgezet. Er is gekozen voor een computationeel experiment als onderzoeksvorm, omdat dit als hoofddoel heeft om statische modellen te valideren (Wohlin et al, 2012). De nulhypothese die gedefinieerd is voor dit experiment is “via tekstuele features op basis van jurisprudentie in een belastingrecht zaak is het niet mogelijk om een beslissing te voorspellen” en de alternatieve hypothese is “via tekstuele features op basis van jurisprudentie in een belastingrecht zaak is het mogelijk om een beslissing te voorspellen”. Het doel van dit experiment is het meten van de prestatie van het model, het bevestigen of falsificeren van de hypothese en het meten van de hoeveelheid invloed per variabele op de uitkomst. Een uitspraak van een belasting rechtszaak bestaat uit een procesverloop, feiten, geschil, en beslissing. Deze variabelen zijn gebruikt in figuur 3.1

het conceptuele model, waarin procesverloop, feiten, geschil, en een combinatie van deze variabelen gedefinieerd zijn als onafhankelijke variabelen en de beslissing als afhankelijke variabele. In het experiment wordt duidelijk welke variabelen en hoeveelheid invloed deze variabelen hebben op de uiteindelijke beslissing.



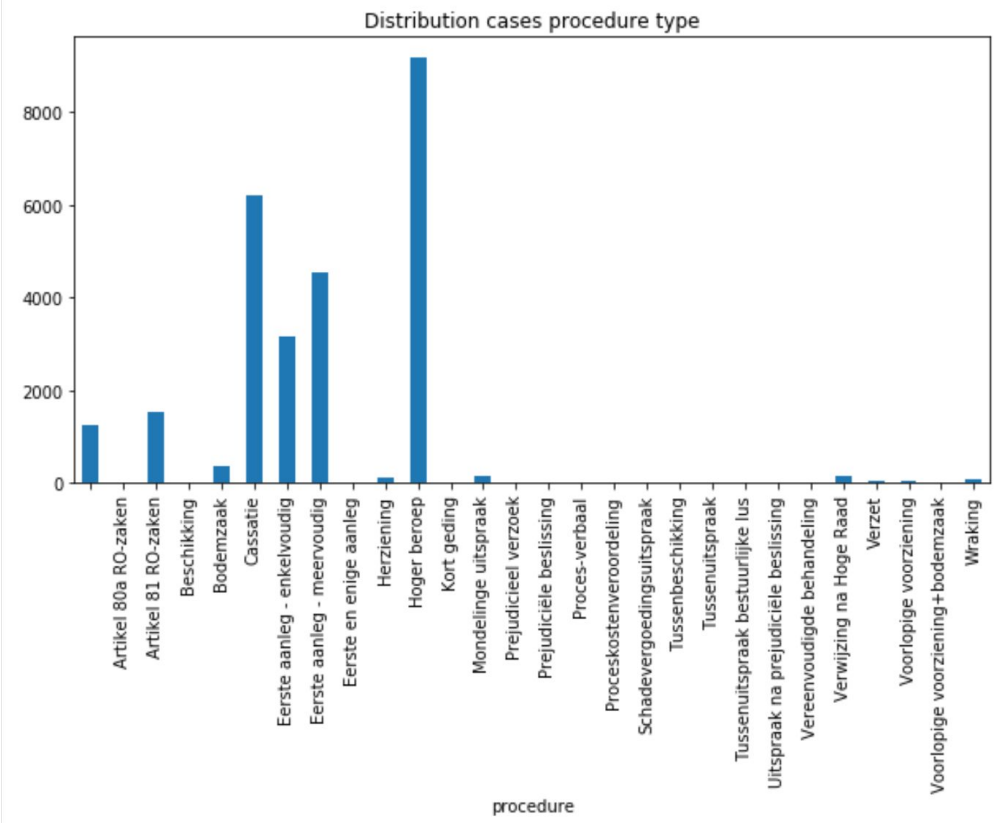
figuur 3.1 conceptueel model van het onderzoek

3.2. Technisch ontwerp: uitwerking van de methode

In deze paragraaf wordt het model gedefinieerd en brondata, tekstuele features en classificatiemodel beschreven. De analysetechniek die vanuit de literatuurstudie het meest geschikt lijkt om toegepast te worden is de combinatie van NLP, SVM en LSTM, zoals toegepast door Aletras et al. (2016) en Menger et al. (2018). De analysetechniek is toegepast op tekstuele documenten met een vergelijkbare structuur en door de bewezen acceptabele nauwkeurigheid (Aletras et al., 2016; Menger et al., 2018) vormt dit een goede basis voor de nieuwe modellen.

Data

In dit onderzoek wordt de dataset van de Rechtspraak.nl gebruikt als brondata voor het model. In deze dataset zijn Nederlandse rechtszaken te vinden in XML formaat. De dataset is via een zelf gemaakt Python programma (**Bijlage 2 - Uitvoer onderzoek Python notebook**) ingelezen waarin de data wordt gegroepeerd op rechtsgebied en procedure. De XML bestanden zonder inhoudelijke informatie over de zaak zijn gefilterd en dit heeft geresulteerd in een dataset van in totaal 296.902 rechtszaken. De dataset is vervolgens gefilterd op het rechtsgebied belastingrecht, wat heeft geresulteerd in 26.903 belasting rechtszaken. Deze 26.903 zijn vervolgens gegroepeerd op procedure, zie figuur 3.2.



figuur 3.2 verdeling zaken procedure

In dit onderzoek wordt gebruik gemaakt van de procedure eerste aanleg enkelvoudig en meervoudig. Deze procedure is binair classificeerbaar, namelijk gegrond of ongegrond. Ook bevat dit type procedure de noodzakelijke variabelen, in tegenstelling tot andere type procedures zoals de cassatieprocedure. Zie bijlage 1 voor een voorbeeld van een belasting rechtszaak.

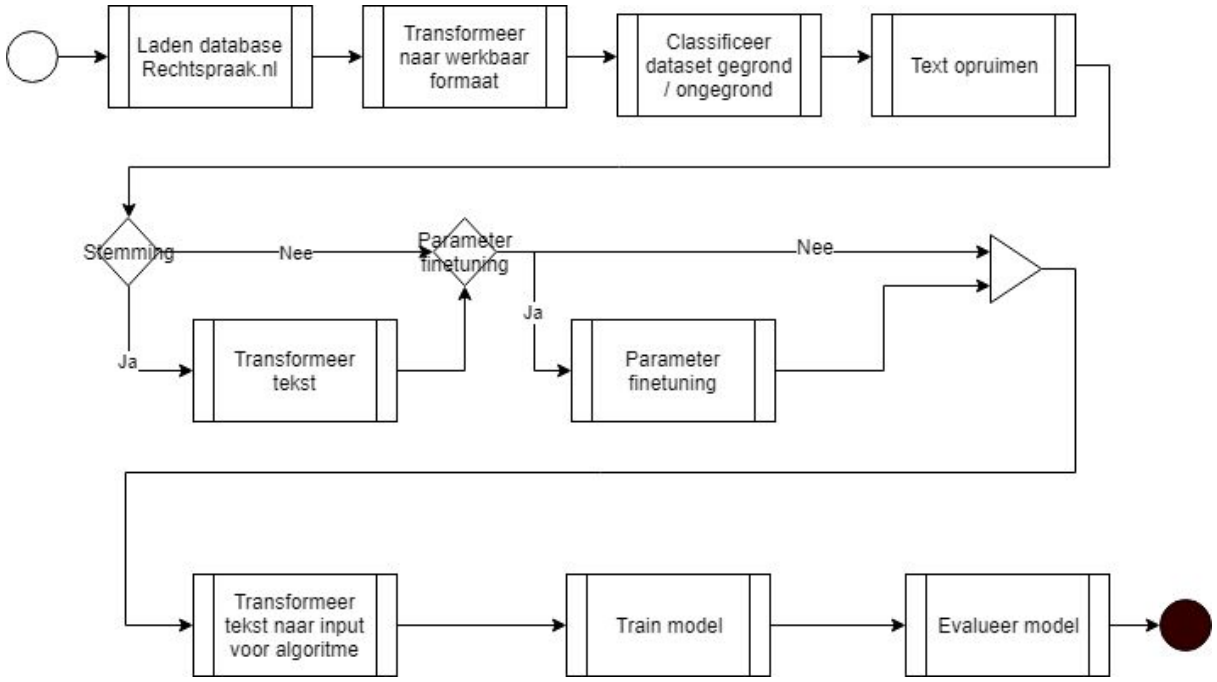
Uitwerking model

Het doel van het model is op basis een van belasting rechtszaak voorspellen of een beroep of verzet gegrond of ongegrond is. De taak van het model is het uitvoeren van binaire classificatie (gegrond / ongegrond). In het kort heeft elk model de volgende opzet:

- lees de data van de Rechtspraak.nl in en transformeer naar werkbaar formaat;

- classificeer elke zaak gegrond of ongegrond;
- tekst opruimen;
- optioneel, toepassen van stemming;
- transformeer tekst naar formaat dat algoritme kan verwerken;
- optioneel, toepassen van parameter finetuning;
- train algoritme;
- evalueer model;

Zie figuur 3.3 voor een schematische weergave van de volledige flow met de verschillende paden die mogelijk zijn.



figuur 3.3 Overzicht model opzet

Dit model wordt toegepast op twee verschillende ML methodieken waarmee de classificatie wordt uitgevoerd, namelijk de SVM en LSTM. In de onderstaande tabel worden de stappen van computationeel experiment in meer detail uitgelegd.

Activiteit	**Beschrijving**
Laden database Rechtspraak.nl	De database in xml structuur wordt ingeladen in een Python programma.
Transformeer werkbaar formaat	De data wordt getransformeerd naar een werkbare datastructuur en lege zaken worden eruit gefilterd.
Classificeer beslissing	Elke zaak in de testset wordt gelabeld op basis van de beslissing. Deze labels worden via de reguliere expressies uit een zaak geëxtraheerd. De beslissing wordt als volgt gelabeld , +1 waar gegrond beroep van toepassing is en -1 voor ongegrond beroep.

Text opruimen	De tekst wordt opgeruimd door de volgende stappen: - alle tekst te transformeren naar kleine letters; - stopwoorden te verwijderen; - interpunctie te verwijderen;	
Stemming	Een optionele stap in het model is de toepassing van stemming. Hiermee wordt een woord terugbracht naar de basisvorm, meestal door het laatste deel van het woord eraf te halen, zoals ‘en’. Dit kan potentieel invloed hebben op de nauwkeurigheid van het model.	
Parameter finetuning	Een optionele stap waarin de parameters gefinetuned worden. Dit kan potentieel invloed hebben op de nauwkeurigheid van het model.	
	SVM	**RNN - LSTM**
Transformeer tekst naar input voor algoritme (tekst representatie)	De representatie van tekst die gebruikt wordt voor de SVM zijn N-grams op basis van de secties procesverloop, feiten, geschil, de hele zaak en onderwerpen. We gebruiken in totaal de top-2000 N-grams. Er worden twee vectorizers gebruikt en vergeleken in dit onderzoek. De term frequency – inverse document frequency (TF-IDF) vectorizer en count vectorizer. De count vectorizer maakt een matrix op basis van de frequentie van een woord. De TF-IDF methode kijkt niet alleen de frequentie, maar ook naar de relevantie van een term in een document (Aletras et al., 2016; Menger et al., 2018).	Voor de RNN worden de secties procesverloop, feiten, geschil en de hele zaak omgezet naar sequenties en worden zinnen die minder karakters bevatten gevuld met 0 waarden, zodat elke sequentie dezelfde lengte heeft. Dit proces staat beter bekend als ‘padding’.
Train model	De SVM wordt door middel van de N-grams getraind. Er wordt gebruik gemaakt van een lineaire kernel functie waarmee het mogelijk wordt om belangrijke features te identificeren door naar de geleerde gewichten te kijken	De RNN architectuur is als volgt opgezet. De sequenties worden aan een ‘Embedding’ laag gevoerd en de resultaten hiervan worden aan een LSTM laag met 320 nodes doorgegeven. Deze wordt voor 20 epochs getraind. De

	van elke feature (Chang & Lin, 2008; Aletras et al. 2016).	parameters zijn gebaseerd op onderzoek van Menger et al., (2018).
Evalueer model	Voor elk model wordt er een evaluatie gedaan van de nauwkeurigheid. Dit wordt gedaan door middel van de AUC (Menger et al., 2018). Hiermee kan elk model vergeleken worden via dezelfde metriek.	

Evaluatie methodiek

Het gebruik van een computationeel experiment maakt het mogelijk om het voorgestelde statische model (Wohlin et al, 2012) te valideren. Door de onderzoeksvorm computationeel experiment te gebruiken is er net als in een experiment veel controle over de variabelen (Saunders, 2016). In de combinatie met empirische data zorgt dit ervoor dat het model gevalideerd kan worden met real life data in plaats van een traditioneel experiment dat gebruikt maakt van data in een experimentele setting. De N-gram features, als input van een SVM, laten in eerder onderzoek zien dat ze effectief zijn in supervised learning taken (Bamman, Eisenstein & Schnoebelen, 2014; Lampos & Cristianini, 2012; Aletras et al., 2016). Ook de RNN architectuur vertoond goede resultaten in voorgaand onderzoek (Menger et al., 2018). Naast de voorspellende beslissing van het model heeft het gebruikte model een bijkomend voordeel dat er geanalyseerd kan worden welke variabelen en onderwerpen van invloed zijn voor de beslissing. Door gebruik te maken van de AUC als evaluatiemethode is er een eenduidige manier om de nauwkeurigheid van de modellen te meten en te vergelijken.

3.3. Reflectie t.a.v. validiteit, betrouwbaarheid en ethische aspecten

In dit onderzoek wordt er gebruikt gemaakt van de openbare dataset van de Rechtspraak.nl waarin geanonimiseerde data wordt gebruikt. Er zijn geen namen van verdachten aanwezig in deze database, maar wel van rechters. In dit onderzoek wordt vanuit privacy oogpunt geen publicatie gedaan van gevonden correlaties die gekoppeld kunnen worden aan individuen. Het kan potentieel mogelijk zijn dat er verbanden gevonden worden tussen onderwerpen, variabelen, beslissingen en specifieke individuen. Deze verbanden zouden van gevoelige aard kunnen zijn, denk hierbij bijvoorbeeld aan discriminerende of racistische motieven.

De interne validiteit van het onderzoek wordt potentieel verhoogd door meerdere bronnen te combineren (Saunders, 2016), namelijk literatuur en een interview. Het onderzoeksvorm experiment heeft meestal een hogere interne validiteit door de controle van de variabelen (Saunders, 2016). Door de cross validatie methodiek kan er aangetoond worden dat het model de juiste classificatie uitvoert en de nauwkeurigheid hiervan. Hiermee kan op een accurate wijze de performance van het model worden gemeten op de bestaande dataset. Door de statische methodiek kan de invloed van onafhankelijke variabelen op afhankelijke variabele nauwkeurig worden gemeten. Een nadeel van het gebruikte model en ML in het algemeen is dat het lastig om te begrijpen wat er precies gebeurt in het model zelf, bijvoorbeeld wat betekent een specifieke waarde van een gewicht in het model.

In het onderzoek is gepoogd om alle keuzes te verantwoorden met bronnen om zo de interne validiteit verder te verhogen en het onderzoek reproduceerbaar te maken. Een punt van aandacht is de datakwaliteit van de Rechtspraak.nl. Door het grote aantal zaken is het niet mogelijk om deze allemaal handmatig te classificeren en de secties zijn niet in alle zaken goed opgedeeld in het XML bestand. Hierdoor is er een zelf geschreven Python programma gemaakt die hierbij ondersteunt (zie bijlage 2). Het programma classificeert via reguliere expressies de zaken als gegrond of ongegrond. Door een handmatige steekproef is gevalideerd dat de juiste classificatie is uitgevoerd.

Het model richt zich specifiek op het procedure type eerste aanleg in het rechtsgebied belastingrecht waardoor het model niet generaliseerbaar is voor alle type procedures en rechtsgebieden. Echter is het model wel generaliseerbaar voor alle eerste aanleg zaken in het Nederlands belastingrecht. In principe zou het model toegepast kunnen worden op andere rechtsgebieden waar de beslissing binair classificeerbaar is en de opzet van de zaak (onafhankelijke variabelen) overeenkomen. Er kunnen geen uitspraken gedaan worden over de nauwkeurigheid van het model in andere rechtsgebieden.

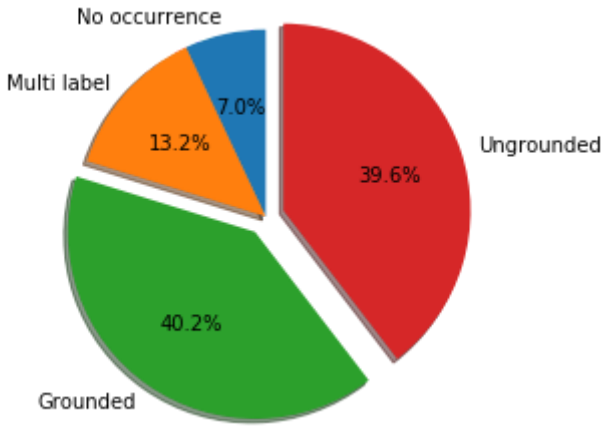
4. Resultaten

In dit hoofdstuk wordt beschreven hoe de uitvoering is verlopen, welke afwijkingen er zijn tijdens de uitvoering en de resultaten van de geteste modellen. De volledige uitvoering van het onderzoek, met ondersteunende Python code, is te vinden in bijlage 2.

Het oorspronkelijke plan van aanpak specificeert drie type modellen, maar tijdens de uitvoer van het experiment werd het duidelijk dat er meer configuraties mogelijk zijn. Denk hierbij aan verschillende parameters en configuraties. Dit heeft uiteindelijk geleid tot acht geteste modellen.

De eerste stap die is uitgevoerd is het filteren van de dataset van Rechtspraak.nl. De dataset is eerst gefilterd op de lege elementen: procesverloop, procedure, overwegingen en beslissing. Deze data is randvoorwaardelijk en rechtszaken zonder een van deze elementen worden niet verder gebruikt. Vervolgens zijn alleen de rechtszaken van het type ‘Belastingrecht’ en procedure type ‘Eerste aanleg’ gebruikt. Dit heeft geresulteerd in 6.538 rechtszaken.

De rechtszaken zijn daarna geclassificeerd op basis van de tekstuele sectie ‘Beslissing’. Waar het woord ‘gegrond’ voorkomt in deze sectie wordt de zaak geclassificeerd met de waarde 1 en ‘ongegrond’ met -1. Tijdens de uitvoering van deze actie kwam aan het licht dat er zaken zijn met meerdere beslissingen (zie figuur 4.1). Deze zaken zijn eruit gefilterd, omdat in dit onderzoek alleen gekeken wordt naar binaire classificatie en niet multi-label classificatie.



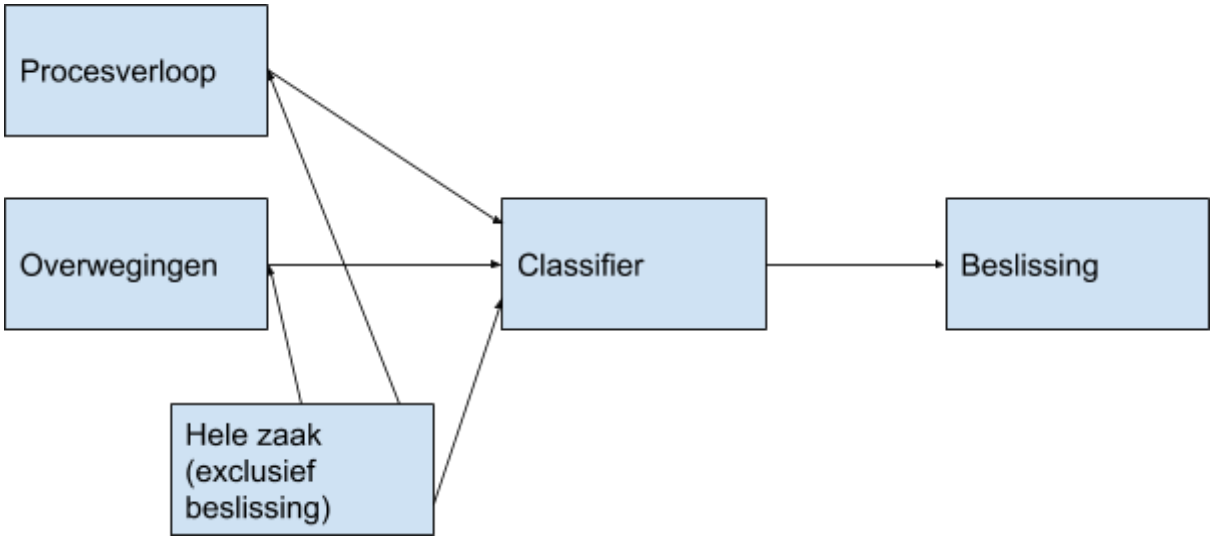
figuur 4.1 verdeling classificaties

Dit heeft uiteindelijk geresulteerd in een gebalanceerde dataset van 2.589 ongegronde rechtszaken en 2.628 gegronde rechtszaken.

Tijdens de feature engineering fase werd het duidelijk dat de beoogde features niet ondersteunt worden door de brondata. De secties ‘feiten’ en ‘geschil’ zijn niet in alle bronbestanden aanwezig en de xml structuur leent zich niet voor het onderscheiden van deze secties. De overkoepelende sectie ‘overwegingen’ is daarom gebruikt. Dit heeft

uiteindelijk geleid tot de volgende tekstuele secties die gebruikt worden om features te extraheren: procesverloop, overwegingen en de hele zaak zonder beslissing sectie.

In de oorspronkelijke model opzet was het de bedoeling om topics te gebruiken om zo de dimensionaliteit van de feature space te reduceren. Er is geprobeerd om de methodiek beschreven in het onderzoek van Aletras et al. (2016) te gebruiken, maar tijdens het opzetten van het model werd het niet mogelijk om deze features bruikbaar te krijgen. De datastructuur gegenereerd door mijn code komt niet overeen met datastructuur van Aletras et al. (2016) en er zijn ook geen voorbeelden online te vinden die dezelfde methodiek hanteren. Tijdens het testen met deze incorrecte features nam de accuraatheid van het model ook af. Deze topics zijn daarom niet meegenomen in de uiteindelijke modellen. Dit proces is verder beschreven in bijlage 2 ‘Uitvoer onderzoek Python notebook‘, paragraaf ‘Approach 1 - Vectorizing data’. Hieronder, in figuur 4.3 is de opzet van het uiteindelijke model te vinden.



figuur 4.3 vernieuwde opzet model

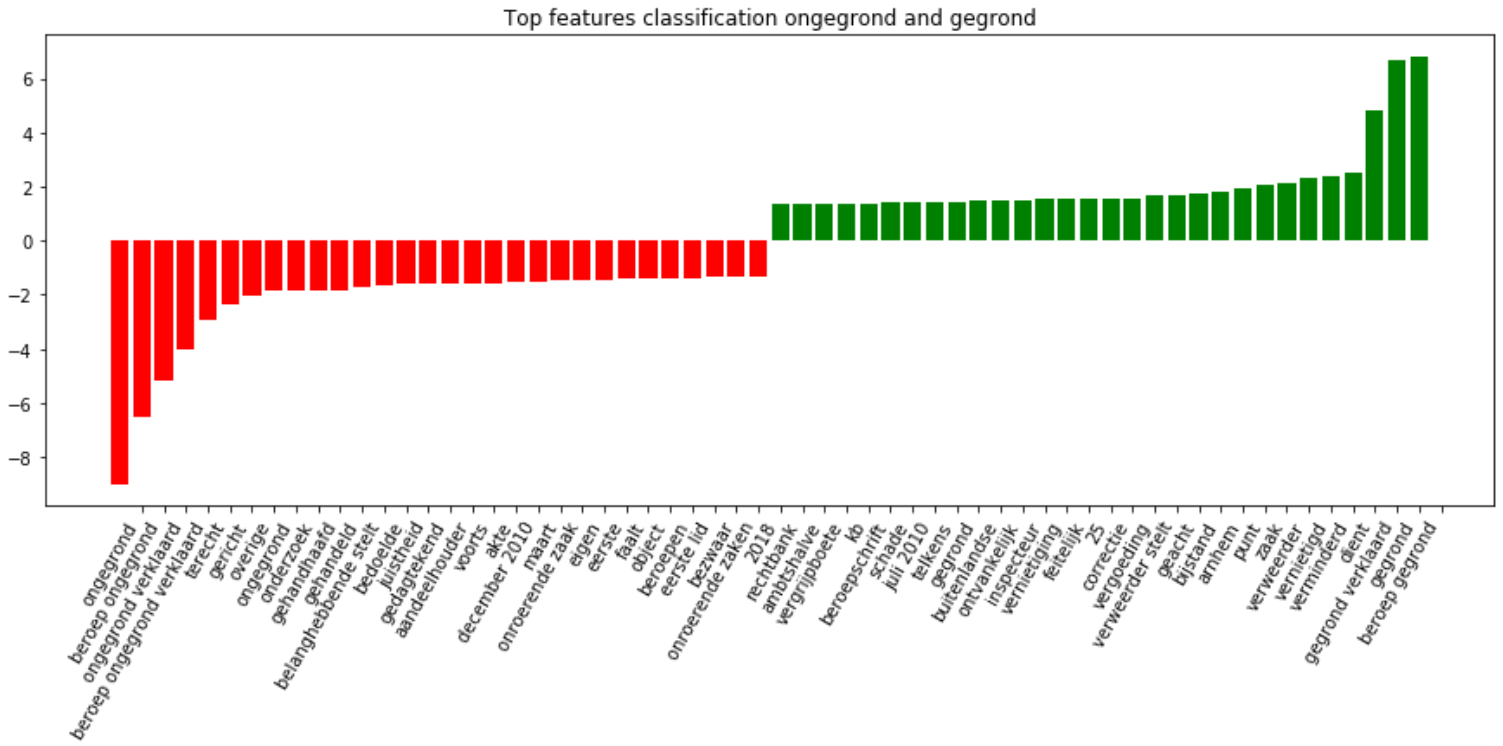
Het best scorende model heeft uiteindelijk via cross validatie een AUC van 0.789 bereikt. Zie hieronder in tabel 4.4 de verschillende modellen, configuraties en AUC scores.

	Classificatie methode	**Configuratie**	**AUC sectie ‘Procesverloop’**	**AUC sectie ‘Overwegingen’**	**AUC volledige zaak (exclusief beslissing)**
1.	SVM	Geen fine-tuning Geen stemming Tf-idf vectorizer N-gram range 4	0.646	0.765	**0.789**
2.	SVM	Geen fine-tuning Geen stemming Tf-idf vectorizer	0.646	0.765	**0.789**

		Bag-of-words			
4.	SVM	Count vectorizer N-gram range 4 Fine-tuning	0.594	0.747	**0.750**
3.	SVM	Tf-idf vectorizer N-gram range 4 Fine-tuning	0.596	0.727	**0.731**
4.	SVM	Geen fine-tuning Geen stemming Count vectorizer N-gram range 4	0.557	**0.729**	0.715
5.	SVM	Geen fine-tuning Geen stemming Count vectorizer Bag-of-words	0.564	**0.701**	0.696
6.	SVM	Count vectorizer N-gram range 4 Stemming	-	-	**0.687**
7.	SVM	Tf-idf vectorizer N-gram range 4 Stemming	0.633	0.647	**0.684**
8.	RNN	Embedding laag LSTM laag	0.5	0.5	-

tabel 4.4 resultaten

De top dertig hoogst scorende features voor gegronde en ongegronde rechtszaken zijn hieronder in figuur 4.5 te vinden.



figuur 4.5 top features gegrond / ongegrond

5. Conclusie, discussie en aanbevelingen, reflectie

In dit hoofdstuk wordt antwoord gegeven op de onderzoeksvraag “In hoeverre is het mogelijk om juridische beslissingen op basis van jurisprudentie te voorspellen in het Nederlands belastingrecht met behulp van Machine Learning en NLP?” en worden er conclusies getrokken.

5.1. Discussie

De resultaten van het experiment laten zien dat het model dat gebruikt maakt van een SVM als classifier en N-grams als tekst representatie de beste resultaten behaald. Specifiek de twee modellen die gebruik maken van de tf-idf vectorizer, zonder stemming en fine-tuning. Hiermee is de hoogste AUC behaald van 0.789 op basis van de combinatie van de secties ‘procesverloop’ en ‘overwegingen’. Zowel het bag-of-words model als N-gram (n=4) behalen dezelfde resultaten en hebben geen invloed op de AUC. Deze resultaten zijn vergelijkbaar met de resultaten die behaald zijn in het onderzoek van Aletras et al. (2016) waar het model op gebaseerd is. De tf-idf vectorizer laat in dit onderzoek betere resultaten zien dan de count vectorizer.

Het gemaakte RNN model heeft de slechte resultaten behaald met een AUC van 0.5. Deze waarde toont aan dat het model geen scheidingsmogelijkheden heeft en geen binaire classificatie kan uitvoeren met een hogere nauwkeurigheid dan willekeurig een waarde classificeren. In dit experiment laten de klassieke ML technieken veel betere resultaten zien dan de gebruikte Deep Learning techniek. In het onderzoek van Menger et al. (2018) laten de resultaten het tegenovergestelde zien. Een aantal potentiële verklaringen van deze resultaten zouden kunnen zijn:

- de architectuur van het Deep Learning model in dit onderzoek is incorrect;
- de sequenties gebruikt om het model te trainen zijn te lang;
- de maximale lengte van de sequentie is vastgezet op de gemiddelde lengte van alle sequenties en hierdoor zijn er details verloren gegaan;
- de RNN techniek is niet de juiste techniek voor dit classificatieprobleem.

In dit onderzoek is er geen oorzaak gevonden waarom de RNN niet leert op basis van de gemaakte features. Er is meer onderzoek nodig om de oorzaak van deze tegenstrijdige resultaten te vinden.

Een tekortkoming van het gemaakte model in dit onderzoek is de onwetendheid vanuit welke optiek de beslissing gegrond of ongegrond is. In een rechtszaak heb je te maken met een aanklager en een verweerder. Dit is in het model niet meegenomen en het model classificeert simpelweg op basis van de tekst als de beslissing gegrond of ongegrond is.

Het figuur 4.5 toont de top dertig hoogst scorende woorden voor de classificatie van gegronde en ongegronde zaken. Hierin is te zien dat woorden waar ‘gegrond’ of ‘ongegrond’ in voorkomen het hoogst wegen voor de classificatie. Deze resultaten zijn niet geheel onverwacht aangezien deze woorden vaak voorkomen in zaken die gegrond of ongegrond zijn, maar niet in alle zaken. De overige woorden geven niet direct aan waarom een beslissing gegrond of ongegrond is. Bijvoorbeeld de stad Arnhem of het woord ‘dient’ zijn sterk wegende woorden voor een gegronde zaken.

5.2. Conclusies

De hoofdvraag van het onderzoek is in hoeverre het mogelijk is om juridische beslissingen op basis van jurisprudentie voorspellen in het Nederlands belastingrecht. Het experiment toont aan dat dit mogelijk is met een relatief hoge nauwkeurigheid. Met een AUC van 0.789 zijn vergelijkbare resultaat behaald vergeleken met voorgaande onderzoeken. Het uitgevoerde experiment toont aan dat een SVM in combinatie met N-grams als tekst representatie een geschikte manier is om (binaire) juridische beslissingen in het Nederlands belastingrecht te voorspellen. De alternatieve hypothese “via tekstuele features op basis van jurisprudentie in een belastingrecht zaak is het mogelijk om een beslissing te voorspellen” is door het experiment bevestigd. In het conceptueel model waren er vier variabelen onafhankelijke variabelen gedefinieerd waarvan er drie zijn overgebleven. De ‘feiten’ en ‘geschil’ variabelen zijn samengevoegd tot een ‘overwegingen’ variabele. De variabele ‘hele zaak (exclusief beslissing)’ heeft de meeste invloed op een nauwkeurige classificatie, gevolgd door ‘overwegingen’. De ‘procesverloop’ variabele heeft de minste invloed op een nauwkeurige classificatie.

Een RNN volgens de voorgestelde opzet in dit onderzoek is geen geschikte analysetechniek voor het voorspellen van beslissingen. Met een AUC van 0.5 is dit vergelijkbaar met een willekeurige classificatie. Het is onduidelijk of dit komt doordat de RNN gebruikt in dit onderzoek foutief is geconfigureerd of dat een RNN niet om kan gaan met de bron data.

5.3. Aanbevelingen voor de praktijk

De resultaten van dit onderzoek laten zien dat binaire classificaties in het juridische domein mogelijk zijn. Echter is de generaliseerbaarheid naar andere rechtsgebieden laag, omdat het model voorgesteld in dit onderzoek niet bruikbaar is voor multi label classificatie. Veel rechtsgebieden en typen procedures hebben complexe uitspraken die niet binair classificeerbaar zijn. Het model zou gebruikt kunnen worden om jurisprudentie gebruikt in dit onderzoek bij de Rechtspraak.nl automatisch te labelen. Ook zijn de resultaten van dit onderzoek interessant voor andere onderzoekers en data scientists die gebruik willen gaan maken van de Rechtspraak.nl database.

5.4. Aanbevelingen voor verder onderzoek

In dit onderzoek is er alleen gekeken naar binaire classificatie waardoor er maar een klein deel van de database van de Rechtspraak.nl is gebruikt. Het is daarom voor vervolgonderzoek interessant om te kijken of multi label classificatie mogelijk is waardoor meer rechtsgebieden en typen procedures voorspelt kunnen worden. In dit onderzoek zijn er goede resultaten behaald met klassieke ML technieken. De Deep Learning techniek gebruikt in dit onderzoek heeft niet tot goede resultaten geleid. Meer onderzoek is nodig om de oorzaak hiervan te achterhalen. Onderzoek van Menger et al. (2018) toont aan dat Deep Learning technieken betere resultaten behalen vergeleken met klassieke ML technieken. De voorgestelde model opzet zou potentieel ook voor domeinen gebruikt kunnen worden waar een binaire classificatie op basis van NLP uitgevoerd kan worden.

5.5. Reflectie

In dit onderzoek is gebruik gemaakt van wetenschappelijke literatuur om de keuzes in het experiment te verantwoorden. De gemaakte modellen zijn methodisch opgebouwd en gebaseerd op voorgaand onderzoek waarin deze goede resultaten hebben behaald. De onderzoeksvorm computationeel experiment in combinatie met de gebruikte evaluatiemethodiek AUC zorgt ervoor dat er een nauwkeurige meting gedaan kan worden. We kunnen precies meten hoeveel invloed een onafhankelijke variabele heeft op de nauwkeurigheid van het model en de resultaten. Hierdoor kunnen we spreken van een hoge interne validiteit. Alle gebruikte classifiers, tekst representaties en configuraties zijn door dezelfde methodiek getraind en geëvalueerd. Hierdoor kan de evaluatie van elk model vergeleken worden met elkaar en kunnen we spreken van een correcte vergelijking.

Een tekortkoming in het onderzoek is de vergelijking van resultaten tussen de SVM en RNN als classifier. In het experiment is de alternatieve hypothese bevestigd, maar de vergelijking van resultaten van de SVM en RNN zijn minder nauwkeurig. Voorgaand onderzoek van Menger et al. (2018) toont aan dat deze ML vorm in theorie betere resultaten zou moeten tonen als de SVM. Echter zien we dat de RNN geen classificatie kan uitvoeren. Omdat het niet duidelijk is wat de oorzaak hiervan is kan hierover geen uitspraak worden gedaan. Andere zoektermen en meer papers omtrent RNN hadden er misschien voor gezorgd dat het RNN model beter was opgezet.

In het oorspronkelijke plan was het de bedoeling om een interview met een jurist in belastingrecht af te nemen om de resultaten te bespreken en te valideren. Via triangulatie zou dit de kwaliteit van het onderzoek ten goede komen. Helaas is het niet gelukt om contact kunnen leggen met een jurist in dit rechtsgebied.

Referenties

Aletras, N., Tsarapatsanis, D., Preoțiuc-Pietro, D., & Lampos, V. (2016). *Predicting Judicial Decisions of the European Court of Human Rights: A Natural Language Processing Perspective.* PeerJ in Computer Science*, 2*. e93.

Bakker, F. C. (n.d.). Jaarverslag 2017. Geraadpleegd van https://jaarverslagrechtspraak.nl/pagina/2017-in-beeld

Breiman, L., (2001). *Random Forests.* Berkeley: Statistics Department

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Heidelberg: Springer.

Chang Y-W, Lin C-J. (2008). *Feature ranking using linear SVM*. WCCI causation and prediction challenge, 53–64.

Dag van de Rechtspraak: Computer vs rechter. (2017, September 28). Geraadpleegd van https://www.rechtspraak.nl/Organisatie-en-contact/Organisatie/Raad-voor-de-rechtspraak/Nieuws/Paginas/Dag-van-de-Rechtspraak-computer-vs-rechter.aspx?pk_campaign=rssfeed&pk_medium=rssfeed&pk_source=Alle-landelijke-actualiteiten

Dechter, R. (1986). *Learning While Searching in Constraint-Satisfaction-Problems.. AAAI. 178-185.*

Deng, L. & Yu, D. (2014), *Deep Learning: Methods and Applications,* Foundations and Trends® in Signal Processing: Vol. 7: No. 3–4, pp 197-387.

Gerven, M., Bohte, S., (2018). *Artificial Neural Networks as Models of Neural Information Processing*. Lausanne: Frontiers Media.

Ivakhnenko, A. G., & Lapa, V. G. (1973). *Cybernetic predicting devices*

Joachims, T., 2002*. Learning to classify text using support vector machines: methods, theory and algorithms.* Kluwer Academic Publishers.

Joachims, T., 1998 *Text categorization with Support Vector Machinces: learning with many relevant features.* Proceedings of the 10th European Conference on Machine Learning: pp 137 - 142

Katz, D. M., Bommarito, M. J., & Blackman, J. (2017). *Predicting the Behavior of the Supreme Court of the United States: A General Approach. PLoS ONE,12(4). doi:https://doi.org/10.1371/journal.pone.0174698*

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulranjani, I., Zhong, V., Paulus, R., Socher, R., (2016). *Ask me Anything: Dynamic Memory Networks for Natural Language Processing.* MetaMind, Palo Alto, CA USA

Lampos V, Aletras N, Preoțiuc-Pietro D, Cohn T. 2014. *Predicting and characterising user impact on Twitter*. Proceedings of the 14th conference of the European Chapter of the Association for Computational Linguistics, 405–413.

Le, Q., Mikolov, T., (2014) *Distributed representations of sentences and documents*

Lipton, Z.C., Kale, D.C., Elkan, C. & Wetzel, R. (2018). *Learning to Diagnose with LSTM Recurrent Neural Networks*.

Menger, V., Scheepers, F., & Spruit,M. (2018). *Comparing Deep Learning and Classical Machine Learning Approaches for Predicting Inpatient Violence Incidents from Clinical Text.* Applied Sciences*, 8*(6), Data Analytics in Smart Healthcare, 981

Mikolov, T.; Corrado, G.; Chen, K.; Dean, J. (2013). *Efficient Estimation of Word Representations in Vector Space*.

Mitchell, T. M. (1997). *Machine learning*. Boston: McGraw-Hill.

Preoțiuc-Pietro D, Lampos V, Aletras N. (2015). *An analysis of the user occupational class through Twitter content.* Proceedings of the 53rd annual meeting of the Association for Computational Linguistics and the 7th international joint conference on natural language processing (Volume 1: Long Papers). 1754–1764.

Preoțiuc-Pietro D, Volkova S, Lampos V, Bachrach Y, Aletras N. 2015. *Studying user income through language, behaviour and affect in social media*. PLoS ONE 10

Prins, C., & Roest, J. (2018). *AI en de rechtspraak: Meer dan alleen de 'robotrechter'*. Nederlands Juristenblad, 93(4), 260-268.

Russell, S. J., & Norvig, P. (2010). *Artificial intelligence: A modern approach*(3rd ed.). Upper Saddle River: Pearson.

Salton G, McGill MJ. (1986). *Introduction to modern information retrieval*. New York: McGraw-Hill, Inc.

Salton G, Wong A, Yang C-S. (1975). *A vector space model for automatic indexing*. Communications of the ACM 18(11):613–620

Saunders, M., Lewis, P., & Thornhill, A. (2016). *Research Methods for Business Students* (7th ed.). Harlow, Essex: Pearson.

Silver, D., et al. (2017). *Mastering the game of Go without human knowledge*. Nature 550: 354.

Turney PD, Pantel P. (2010). F*rom frequency to meaning: vector space models of semantics*. Journal of Artificial Intelligence Research 37:141–188.

Von Luxburg U. (2007). *A tutorial on spectral clustering*. Statistics and Computing 17(4):395–416.

Verheugt, J. (2007). *Inleiding in het Nederlandse recht*(14th ed.). Den Haag: Boom Juridische uitgevers.

Wang S, Manning CD. (2012). *Baselines and bigrams: simple, good sentiment and topic classification*. Proceedings of the 50th annual meeting of the Association for Computational Linguistics: short papers-Volume 2. 9094.

Bijlage 1 - Voorbeeld belastingrecht zaak

RECHTBANK NOORD-NEDERLAND

Zittingsplaats Groningen
Bestuursrecht
zaaknummer: LEE 17/126
uitspraak van de enkelvoudige belastingkamer van 31 juli 2018 in de zaak tussen
[eiseres] , te [woonplaats] , eiseres
en
de heffingsambtenaar van de gemeente Dongeradeel (thans deWerkorganisatie DDFK-gemeenten), verweerder
(gemachtigde: [gemachtigde]).

Procesverloop

Verweerder heeft op 1 juli 2016 aan eiseres een naheffingsaanslag parkeerbelasting opgelegd ten bedrage van € 62,80.

Bij uitspraak op bezwaar van 27 december 2016 heeft verweerder het bezwaar van eiseres ongegrond verklaard.

Eiseres heeft tegen de uitspraak op bezwaar beroep ingesteld.

Verweerder heeft een verweerschrift ingediend.

Het onderzoek ter zitting heeft plaatsgevonden op 29 juni 2018. Eiseres is verschenen. Verweerder heeft zich laten vertegenwoordigen door zijn gemachtigde.

Overwegingen

Feiten

1. De rechtbank neemt de volgende, door partijen niet betwiste, feiten als vaststaand aan.

1.1

Eiseres was op 1 juli 2016 houder van het voertuig, een [auto] met het kenteken [kenteken] (de auto).

1.2

Een parkeercontroleur van de gemeente Dongeradeel heeft op 1 juli 2016 aan eiseres als houder van de auto een naheffingsaanslag parkeerbelasting opgelegd ten bedrage van

€ 62,80 (het bedrag van de parkeerbelasting ad € 2,80 verhoogd met de kosten van de naheffingsaanslag ad € 60) ter zake van het parkeren op voornoemde datum omstreeks 14:25 uur aan De Dijk te Dokkum, omdat het voornoemde kenteken van eiseres niet was aangemeld door middel van het zogenoemde

belparkeren (kort gezegd: het voldoen van de parkeerbelasting door het mobiel inloggen op een centrale computer via een parkeerprovider).

1.3

Ingevolge artikel 8 van de Verordening op de heffing en de invordering van parkeerbelastingen 2016 van de gemeente Dongeradeel (de Verordening), heeft het college van B & W van de gemeente Dongeradeel bij besluit van 1 december 2015 de parkeerplaatsen aan De Dijk te Dokkum (in het centrum) per 1 januari 2016 aangewezen als parkeerplaatsen waarbij de parkeerbelasting (met een maximale parkeerduur van 30 minuten) alleen kan worden voldaan door middel van belparkeren.

1.4

Op de onderhavige naheffingsaanslag parkeerbelasting staat onder andere het volgende vermeld:

“Omschrijving van het feitnummer:

Als bestuurder een voertuig parkeren op een parkeerterrein waar dit slechts met gebruikmaking van een ter plaatse aangebrachte parkeerautomaat is toegestaan anders dan voorzien van een door de parkeerautomaat afgegeven parkeerkaart, aangebracht op de voorgeschreven wijze

Toelichting

Belparkeren niet geactiveerd. ”.

Geschil en beoordeling

2. In geschil is het antwoord op de vraag of verweerder de naheffingsaanslag parkeerbelasting terecht heeft opgelegd. Eiseres beantwoordt deze vraag ontkennend, verweerder bevestigend.

2.1

Eiseres heeft - samengevat - aangevoerd dat er geen mogelijkheid was om te betalen, omdat de gemeente de parkeermeter waar je met muntgeld kon betalen, had weggehaald. Het bord waarop stond dat de parkeerbelasting alleen door middel van belparkeren kon worden betaald is voor automobilisten onzichtbaar en heeft zij niet waargenomen. Ook overigens acht zij de naheffingsaanslag onvoldoende duidelijk, omdat uit de tekst hiervan volgt dat zij geacht wordt een kaartje onder de voorruit te leggen van een parkeerautomaat die er niet is. Eiseres voert tevens nog aan dat het kwalijk is dat de uitspraak op bezwaar een half jaar heeft moeten duren.

2.2

Verweerder stelt zich - tevens samengevat - op het standpunt dat het voor eiseres, gelet op de situatie ter plaatse, duidelijk had moeten zijn dat belparkeren de enige manier van betalen voor het parkeren moest zijn. In het bijzonder wijst verweerder erop dat direct nabij de drie aanwezige parkeerplaatsen aan De Dijk duidelijk zichtbaar een blauw bord aanwezig was, waarop staat dat alleen door middel van belparkeren de parkeerbelasting kan worden voldaan.

3. De rechtbank stelt vast dat tussen partijen niet in geschil is dat de auto van eiseres geparkeerd stond zonder dat eiseres daarvoor de volgens de Verordening vereiste parkeerbelasting had voldaan. Tussen partijen is met name in geschil of voor eiseres kenbaar was dat zij, op de plek waar zij parkeerde, parkeerbelasting verschuldigd was.

4. De rechtbank overweegt dat de verplichting om parkeerbelasting te betalen voor het op een bepaalde plaats en een bepaalde tijd en gedurende een maximale tijdsduur parkeren van een voertuig kenbaar dient te zijn gemaakt op zo een wijze, dat omtrent de verschuldigdheid van parkeerbelasting voor dat parkeren redelijkerwijs geen misverstand kan bestaan. Van verweerder mag bovendien worden verwacht dat onder

andere door middel van duidelijke bebording bij de parkeerplaats of in de naaste omgeving daarvan, het ter plaatse geldende parkeerregime voldoende duidelijk is aangegeven en dat ook duidelijk is hoe de parkeerder de verschuldigde belasting kan voldoen (het zogenoemde kenbaarheidsvereiste, zie in deze zin Hoge Raad, 22 november 1995, nr. 30 141, ECLI:NL:HR:1995:AA3126). Hiervoor is niet vereist dat aan het begin en het eind van elke straat door middel van bebording wordt aangegeven dat het een straat betreft waar voor het parkeren van een auto parkeerbelasting verschuldigd is. Hier staat tegenover dat van een parkeerder mag worden verwacht dat hij zich op de hoogte stelt van de geldende regels met betrekking tot verschuldigdheid van parkeerbelasting in het gebied waar hij wenst te parkeren en de wijze waarop hij daar vervolgens aan moet voldoen (de zogenoemde onderzoeksplicht, zie hiertoe onder meer de uitspraak van gerechtshof Amsterdam van 23 januari 2018, ECLI:NL:GHAMS:2018:363).

5. Naar het oordeel van de rechtbank heeft verweerder, gelet op het verweerschrift, de overgelegde foto's en hetgeen hij ter zitting heeft toegelicht, aannemelijk gemaakt dat de regels omtrent de verschuldigdheid van parkeerbelasting door middel van belparkeren, zoals die op de locatie waar eiseres haar auto parkeerde van toepassing zijn, voor eiseres voldoende duidelijk waren dan wel hadden moeten zijn. Het blauwe bord met de tekst: "Betaald parkeren alleen via mobiel 9972" is op ongeveer twee meter hoogte, naast de RABO-bank, aangebracht in de zeer directe nabijheid van de parkeerplek aan De Dijk waar eiseres haar auto heeft geparkeerd. De rechtbank heeft hierbij in aanmerking genomen dat de gemachtigde van verweerder ter zitting heeft verklaard dat de parkeermeter er in 2016 nog wel heeft gestaan, maar dat deze was "onthoofd", maar dat er wel een waarschuwingsbordje bij was geplaatst met de mededeling, dat er enkel nog door middel van belparkeren kon worden betaald.

6. Hetgeen eiseres hiertegen heeft aangevoerd, namelijk dat zij, mede door haar bescheiden lengte, het blauwe bord niet heeft gezien en dat zij later, naar aanleiding van een eigen onderzoek en raadpleging van Google-Maps, meent dat het blauwe bord op een onlogische plaats is opgesteld, volgt de rechtbank niet (zie 5.). De rechtbank is van oordeel dat het voor eiseres, als zij aan haar onderzoeksplicht had voldaan, duidelijk had moeten zijn dat op De Dijk alleen geparkeerd kon worden via belparkeren. De stelling van eiseres dat de bebording op ooghoogte moet zijn aangebracht vindt geen steun in het recht. In het kader van de onderzoeksplicht kan, zeker in het geval van eiseres, die klein van stuk is, daarom niet worden volstaan met enkel een waarneming op ooghoogte. De veronderstelling van eiseres dat zij niet hoefde te betalen omdat er geen werkzame parkeermeter meer stond waar je met muntgeld kon betalen, dient voor rekening en risico van eiseres te komen.

7. De stelling van eiseres dat de tekst van de naheffingsaanslag (zie 1.4) onvoldoende duidelijk is, volgt de rechtbank niet. Weliswaar wordt in die tekst ten onrechte nog melding gemaakt van een parkeerautomaat en een parkeerkaart, maar uit de daaropvolgende toelichting volgt naar het oordeel van de rechtbank voldoende duidelijk dat sprake was van een situatie van belparkeren. Ook overigens zijn namens eiseres geen feiten of omstandigheden naar voren gebracht die tot de conclusie zouden moeten leiden dat verweerder de onderhavige naheffingsaanslag ten onrechte heeft opgelegd.

8. Ten aanzien van eiseres' stelling dat het kwalijk is dat de uitspraak op bezwaar een half jaar heeft moeten duren volstaat de rechtbank met de constatering dat verweerder uitspraak heeft gedaan binnen de wettelijke termijn van artikel 231, tweede lid, onderdeel b van de Gemeentewet, te weten voor het einde van het kalenderjaar waarin het bezwaarschrift is ontvangen.

9. Het beroep is ongegrond

10. Voor een proceskostenveroordeling bestaat geen aanleiding.

Beslissing

De rechtbank verklaart het beroep ongegrond.

Deze uitspraak is gedaan door mr. M. van den Bosch, rechter, in aanwezigheid van R.H. Wolfslag, griffier. De beslissing is in het openbaar uitgesproken op 31 juli 2018.

bron: https://uitspraken.rechtspraak.nl/inziendocument?id=ECLI:NL:RBNNE:2018:3015&showbutton=true

Bijlage 2 - Uitvoer onderzoek Python notebook

Introduction

In this document court cases from the 'Rechtspraak' database will be processed by means of Natural language processing (NLP) and Machine learning (ML). We try to predict if a decision in a Dutch tax case is grounded or ungrounded. We classify this problem as a binary classification problem.

Three different approaches will be examined:

- the first approach is a model that uses Tf-idf vectorized data in combination with a SVM to predict the outcome, this model is based on the model of Aletras et al. (2016).
- the second approach is mostly identical but with a different vectorizer.
- the third approach uses a RNN LSTM instead of a SVM.

Approach 1: Tf-idf vectorized data feeded to SVM

The model is created by the following steps:

1. load the database of Rechtspraak.nl.
2. filter cases by procedure, empty fields and case type.
3. filter decisions with a combination of ‘gegrond’ and ‘ongegrond
4. filter decisions without an occurrence of ‘gegrond’ or ‘ongegrond’
5. add classifier labels for 'ongegrond' and 'gegrond'
6. remove stopwords and transform to lowercase
7. vectorize dataset
8. train SVM with features
9. score performance with 10-fold cross validation

We start by importing the required modules.

Input:

In [2]:

```
%%time
import _pickle
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import nltk
```

```
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import svm
from sklearn.metrics import classification_report,roc_auc_score,roc_curve
from IPython.display import Markdown as md
from sklearn import model_selection
from sklearn.cluster import SpectralClustering
from sklearn.metrics.pairwise import cosine_similarity
```

Output:

```
Wall time: 0 ns
```

The first thing after this is loading the data frame with court cases from 2000 till 2018. We load an already pickled data frame so we don't have to process all the original xml files from the rechtspraak database again.

Input:

In [3]:

```
 %%time
file_path =
'C:\\Users\\Corbin\\Documents\\machine-learning\\ML-Legal-Rechtspraak\\Legal-Rechtspraak\\input\\pickle\\dataf
rame_2000_2018_bestuursrecht.pkl'
with open(file_path, "rb") as data:
    data_frame = _pickle.load(data)
    print("Total number of cases in dataframe' " + str(len(data_frame)))
```

Output:

```
Total number of cases in dataframe: 296902
Wall time: 1.72 s
```

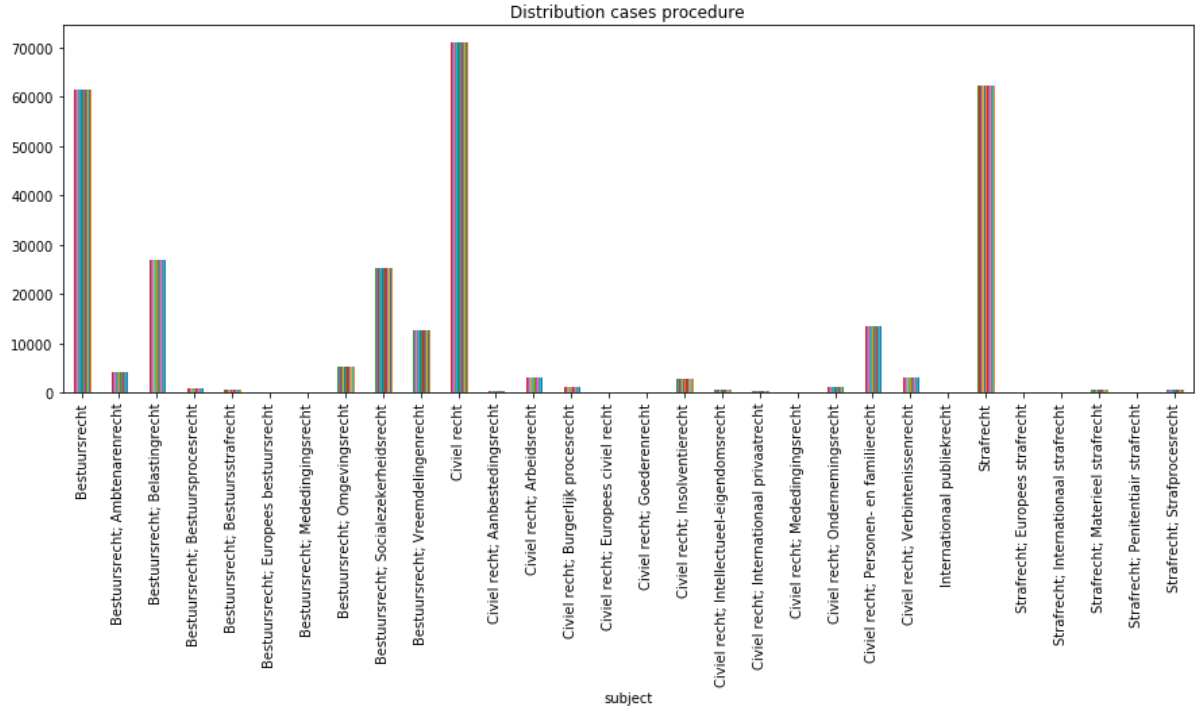
The full data frame contains a total of 296902 cases. These cases are distributed over different law areas. Below you can find the distribution of the cases in these areas.

Input:

```
%%time
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = [15, 5]
```

```
data_frame.groupby('subject').count().plot(kind='bar', legend=None, title="Distribution cases law area")
```

Output:



In this experiment we only use Dutch tax law. That's why the data frame will be filtered on law area 'Belastingrecht'.

Input:

```
%%time
filter_cases_type = 'Bestuursrecht; Belastingrecht'


data_frame = data_frame.loc[(data_frame["subject"] == filter_cases_type)]
cases_filtered_subject = len(data_frame)
print('Using case type {} cases with a total of {}'.format(filter_cases_type, str(cases_filtered_subject)))
```

Output:

```
Using case type Bestuursrecht; Belastingrecht cases with a total of 26903
CPU times: user 12.9 ms, sys: 954 µs, total: 13.9 ms
Wall time: 17.3 ms
```

The data frame contains a total of 26903 cases. This data frame contains all procedure types.

Input:

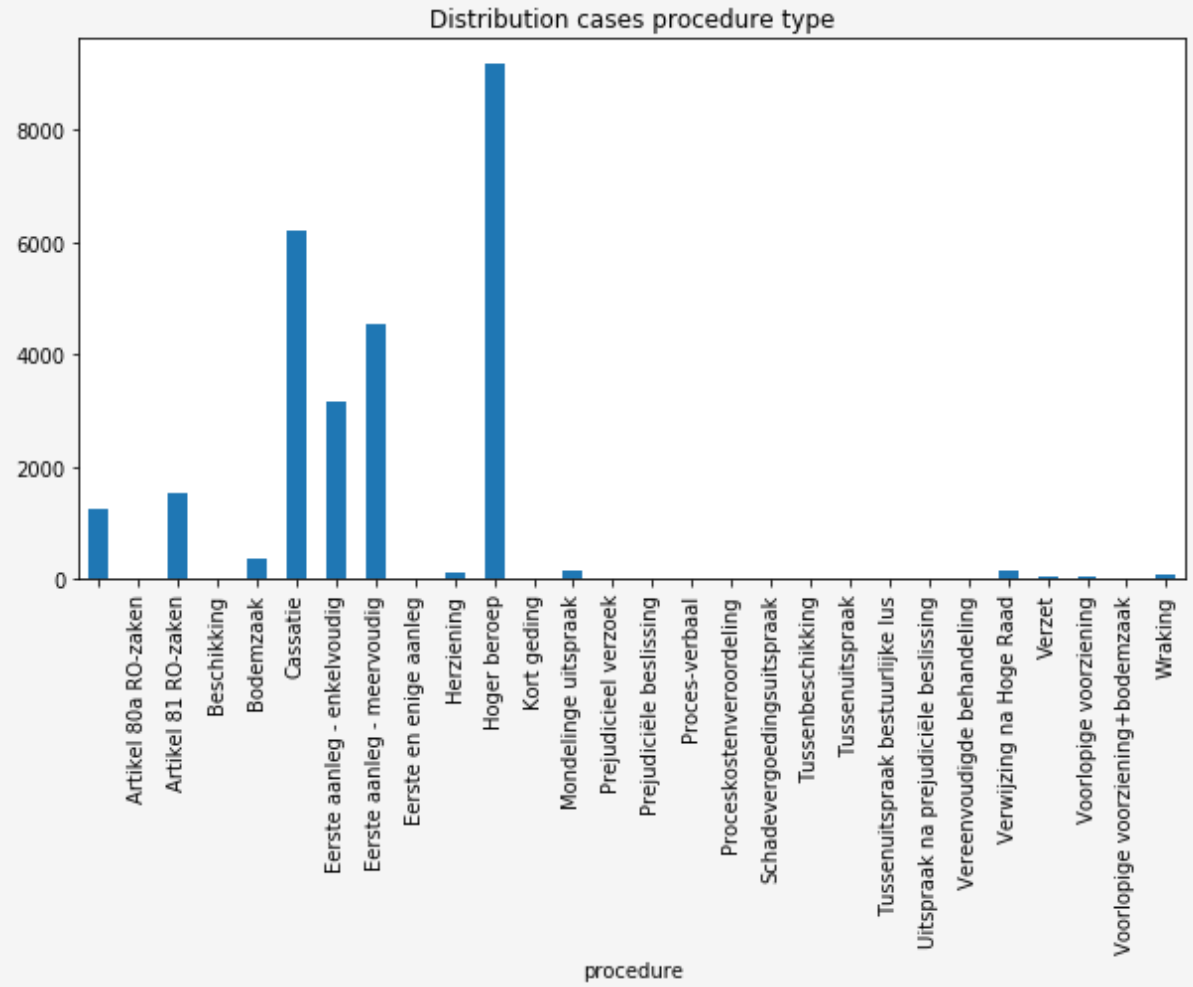
```
%%time
```

```
plt.rcParams['figure.figsize'] = [10, 5]
procedure_count = data_frame.groupby('procedure').size()
print(procedure_count)
procedure_count.plot(kind='bar', legend=None, title="Distribution cases procedure type")
```

Output:

```
procedure

Onbekend                          1252
Artikel 80a RO-zaken                 6
Artikel 81 RO-zaken               1537
Beschikking                          2
Bodemzaak                          376
Cassatie                          6200
Eerste aanleg - enkelvoudig       3173
Eerste aanleg - meervoudig        4527
Eerste en enige aanleg               3
Herziening                         129
Hoger beroep                      9165
Kort geding                          2
Mondelinge uitspraak               146
Prejudicieel verzoek                 4
Prejudiciële beslissing              7
Proces-verbaal                       6
Proceskostenveroordeling             4
Schadevergoedingsuitspraak          12
Tussenbeschikking                    7
Tussenuitspraak                      4
Tussenuitspraak bestuurlijke lus     7
Uitspraak na prejudiciële beslissing 5
Vereenvoudigde behandeling           6
Verwijzing na Hoge Raad            138
Verzet                              53
Voorlopige voorziening              42
Voorlopige voorziening+bodemzaak     7
Wraking                             83
dtype: int64
```

```
Wall time: 198 ms
```



We now filter this data frame on 'Eerste aanleg' procedure, because this procedure contains binary classifiable cases. To create features from our data frame we will also skip all cases that contain empty 'procesverloop', 'overwegingen', 'procedure' and 'beslissing' sections. These sections are mandatory for training our model.

Input:

```
%%time


filter_procedure_type = 'Eerste aanleg'

cases_after_procedure_type = len(data_frame)

data_frame = data_frame[data_frame['procedure'].str.contains(filter_procedure_type)]

print("Number of cases after filtering procedure {} {} ".format(filter_procedure_type,
str(cases_after_procedure_type)))
```

```
data_frame = data_frame.loc[(data_frame["procesverloop"] != "")]
cases_after_empty_procesverloop = len(data_frame)
print("Number of cases after filtering empty procesverloop {} ".format(str(cases_after_empty_procesverloop)))

data_frame = data_frame.loc[(data_frame["procedure"] != "")]
cases_after_empty_procedure = len(data_frame)
print("Number of cases after filtering empty procedure {} ".format(str(cases_after_empty_procedure)))

data_frame = data_frame.loc[(data_frame["overwegingen"] != "")]
cases_after_empty_overwegingen = len(data_frame)
print("Number of cases after filtering empty overwegingen {} ".format(str(cases_after_empty_overwegingen)))

data_frame = data_frame.loc[(data_frame["beslissing"] != "")]
cases_after_beslissing = len(data_frame)
print("Number of cases after filtering empty beslissing {} ".format(str(cases_after_beslissing)))
```

Output:

```
Number of cases after filtering procedure Eerste aanleg 6538
Number of cases after filtering empty procesverloop 6538
Number of cases after filtering empty procedure 6538
Number of cases after filtering empty overwegingen 6538
Number of cases after filtering empty beslissing 6538
CPU times: user 59.1 ms, sys: 27.7 ms, total: 86.8 ms
Wall time: 93.9 ms
```

After removing all the cases with empty sections and other filters, we have 6538 usable cases left.

Now the dataframe is filtered, we will add a classifier column based on the decision 'Beslissing'. When the word 'gegrond' occurs we classify it as 1, if the word 'ongegrond' occurs we classify it as -1. Before we do that we will apply another filter to exclude cases where multiple decisions, a combination of 'gegrond' and 'ongegrond', occur. We also filter cases where the words 'gegrond' or 'ongegrond' do not occur. Since we are doing binary classification, we don't handle cases without these labels.

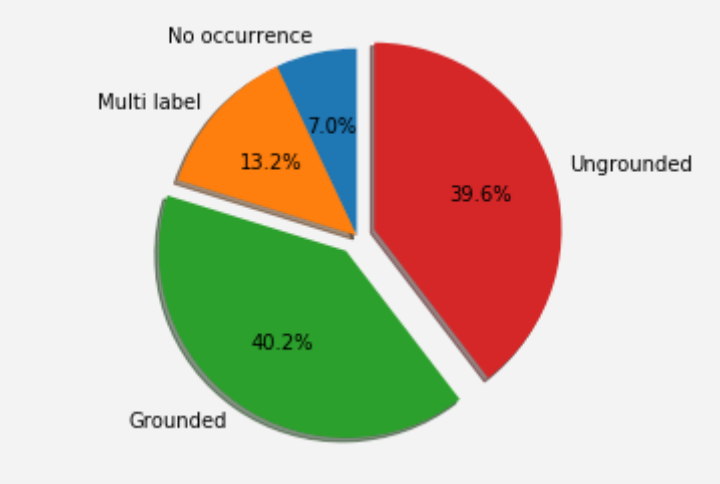
Input:

```
%%time

import matplotlib.pyplot as plt
```

```
total_count = len(data_frame)


data_frame = data_frame[~((data_frame['beslissing'].str.contains(' gegrond',case=False, regex=True)) &
(data_frame['beslissing'].str.contains(' ongegrond',case=False, regex=True)))]

cases_both_occurrence_count = total_count - len(data_frame)

cases_both_occurrence_pct = ((total_count - len(data_frame)) / total_count) * 100


data_frame_no_occurrence = data_frame[~((data_frame['beslissing'].str.contains(' gegrond',case=False,
regex=True)) | (data_frame['beslissing'].str.contains(' ongegrond',case=False, regex=True)))]


data_frame = data_frame[((data_frame['beslissing'].str.contains(' gegrond',case=False, regex=True)) |
(data_frame['beslissing'].str.contains(' ongegrond',case=False, regex=True)))]

no_occurrence_count = total_count - (len(data_frame) + cases_both_occurrence_count)

no_occurrence_pct = ((total_count - (len(data_frame) + cases_both_occurrence_count)) / total_count) * 100


data_frame['decision'] = data_frame['beslissing'].str.contains('ongegrond',case=False, regex=True)

data_frame['decision'] = data_frame['decision'].map({True: -1, False: 1})


usable_cases_count = data_frame['decision'].value_counts()


grounded_count = usable_cases_count.values[0]

grounded_pct = (usable_cases_count.values[0] / total_count) * 100

ungrounded_count = usable_cases_count.values[1]

ungrounded_pct = (usable_cases_count.values[1] / total_count) * 100


print('Multi label cases: ' + str(cases_both_occurrence_count))

print('No occurrence of grounded or ungrounded: ' + str(no_occurrence_count))

print('Grounded cases: ' + str(grounded_count))

print('Ungrounded cases: ' + str(ungrounded_count))


# Pie chart, where the slices will be ordered and plotted counter-clockwise:

labels = 'No occurrence', 'Multi label', 'Grounded', 'Ungrounded'

sizes = [no_occurrence_pct, cases_both_occurrence_pct, grounded_pct, ungrounded_pct]

explode = (0, 0, 0.1, 0.1)  # only "explode" the 2nd slice (i.e. 'Hogs')


fig1, ax1 = plt.subplots()

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',

        shadow=True, startangle=90)

ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.


plt.show()
```

Output:

```
Multi label cases: 863
No occurrence of grounded or ungrounded: 458
Grounded cases: 2628
Ungrounded cases: 2589
```




```
CPU times: user 1.35 s, sys: 75.8 ms, total: 1.43 s
Wall time: 1.37 s
```

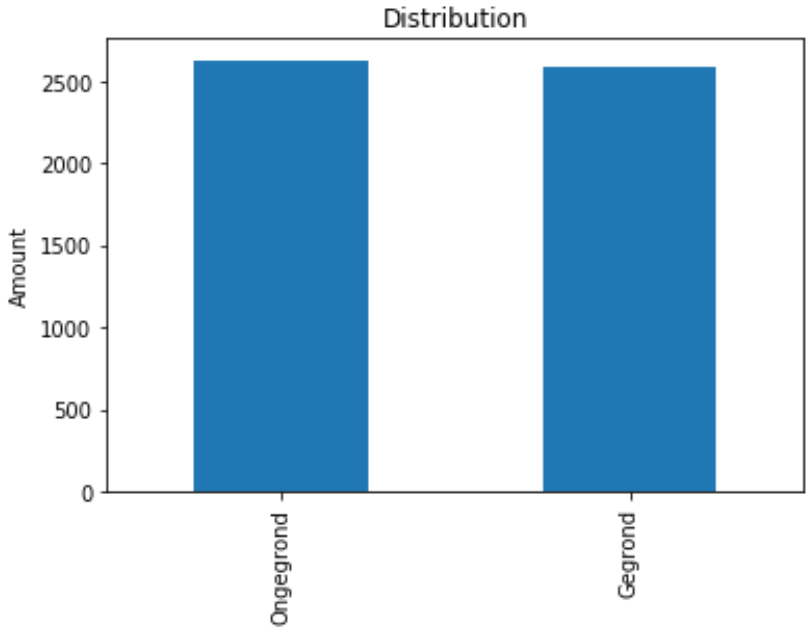
The data frame is balanced with a distribution of 2589 cases 'ongegrond' and 2628 cases 'gegrond'.

In [4]:

```
%%time

data_frame = data_frame[~((data_frame['beslissing'].str.contains(' gegrond',case=False, regex=True)) &
(data_frame['beslissing'].str.contains(' ongegrond',case=False, regex=True)))]
data_frame = data_frame[((data_frame['beslissing'].str.contains(' gegrond',case=False, regex=True)) |
(data_frame['beslissing'].str.contains(' ongegrond',case=False, regex=True)))]

data_frame['decision'] = data_frame['beslissing'].str.contains('ongegrond',case=False, regex=True)
data_frame['decision'] = data_frame['decision'].map({True: -1, False: 1})

bar_chart_distribution = data_frame['decision'].value_counts().plot(kind='bar', title="Distribution")
bar_chart_distribution.set_ylabel('Amount')
bar_chart_distribution.set_xticklabels(['Ongegrond', 'Gegrond'])
```

Output:

```
Name: decision, dtype: int64
Wall time: 3.04 s
```



Since we try to replicate the model from the Aletras et al. (2016), we will start off with 2000 features created with a tfidf vectorizer with n-gram range 1,4. The research of Aletras et al. (2016) talks about the bag-of-words model but specifies a n-gram range of 4. It looks like they made a mistake because the bag-of-worlds model has an n-gram range of 1. In this research we try both approaches.

Vectorizing data

We start by removing Dutch stop words and transform the data to lowercase for both overwegingen and procesverloop texts

Input:

In [4]:

```
%%time

stop_words_nl = nltk.corpus.stopwords.words('dutch')


tfidfVectorizerOverwegingen = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))

overwegingen_dataframe = pd.DataFrame(tfidfVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(), columns=tfidfVectorizerOverwegingen.get_feature_names())

tfidfVectorizerProcesverloop = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))

procesverloop_dataframe = pd.DataFrame(tfidfVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(), columns=tfidfVectorizerProcesverloop.get_feature_names())


# Merge dataframes
```

```
feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)
```

Output:

```
Wall time: 56.6 s
```

Now that we have the textual features, we try to get topics out of them by clustering them to together via Spectral Clustering (von Luxburg, 2007) and use these topics as extra features. The idea is to reduce the dimensionality of the feature space, which is essential for feature selection, it limits overfitting to training data (Lampos et al., 2014; Preoțiuc-Pietro, Lampos & Aletras, 2015; Preoțiuc-Pietro et al., 2015) and also provides a more concise semantic representation (Aletras et al, 2016).

Input:

In [5]:

```
%%time


similarity_matrix = cosine_similarity(feature_data_set, feature_data_set)

clustering = SpectralClustering(n_clusters=30, random_state=0, affinity='precomputed',
n_jobs=-1).fit(similarity_matrix)


print(clustering.labels_)
```

Output:

```
[15 15 15 ...  3  4  4]

Wall time: 11.1 s
```

I have tried to reverse engineer Aletras et al. (2016) way of finding topics but I couldn't find a way to create features out of them based on my knowledge and examples I found online.

I tried to add the output of the spectral clustering to the featureset but saw a decrease accuracy and I cannot validate that I correctly created the topics. Because of this, I have decided not include them in the feature set. This is inconvenient because the topics had a strong influence on the classification in the research of Aletras et al. (2016).

The results of the model with the topics can be found in the 'Results' section.

Input:

In [8]:

```
md('The feature matrix consists of '+ (str(len(feature_data_set.columns))) +' features and '+
str(len(feature_data_set)) + ' records.')
```

Output:

```
The feature matrix consists of 2000 features and 5217 records.
```

Train model

We can now use the feature matrix to train the Support Vector Machine (SVM). We start by splitting the dataframe into train and test data. We start with the ratio 80:20 (Pareto principle), where 80% is training data and 20% is test data.

In a later stage we try to get a better accuracy depiction of the model by using 10-fold cross validation.

Input:

In [10]:

```
%%time


X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)

clf = svm.SVC(kernel='linear')

clf.probability = True

model = clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

y_score = clf.decision_function(X_test)
```

Output:

```
Wall time: 5min 18s
```

Below you can find the top features that are used to classify 'gegrond' and 'ongegrond'

Input:

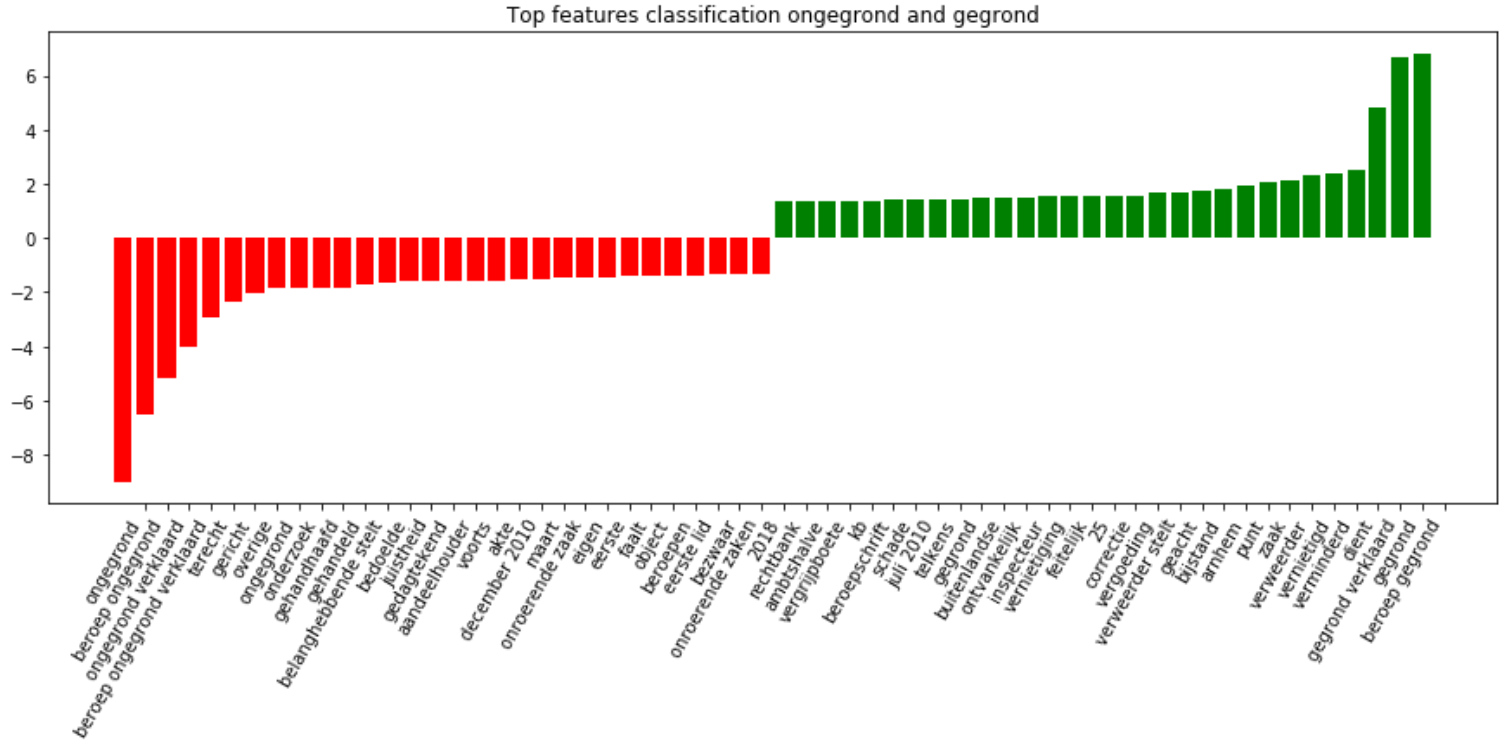
In [11]:

```
 %%time


top_features=30

feature_names = feature_data_set.columns


coef = clf.coef_.ravel()

top_positive_coefficients = np.argsort(coef)[-top_features:]

top_negative_coefficients = np.argsort(coef)[:top_features]

top_coefficients = np.hstack([top_negative_coefficients, top_positive_coefficients])


# create plot
```

```
plt.figure(figsize=(15, 5))
colors = ['red' if c < 0 else 'green' for c in coef[top_coefficients]]
plt.bar(np.arange(2 * top_features), coef[top_coefficients], color=colors)
feature_names = np.array(feature_names)
plt.xticks(np.arange(1, 1 + 2 * top_features), feature_names[top_coefficients], rotation=60, ha='right')
plt.title('Top features classification ongegrond and gegrond')
plt.show()
```

Output:

```
Wall time: 1.27 s
```



The top scoring features show that the words (beroep) gegrond for +1 and (beroep) ongegrond -1 are performing the best. This is not surprising because these words indicate a grounded or ungrounded case.

Results

With the fitted and scored SVM, we can now calculate the accuracy and area under the curve (AUC).

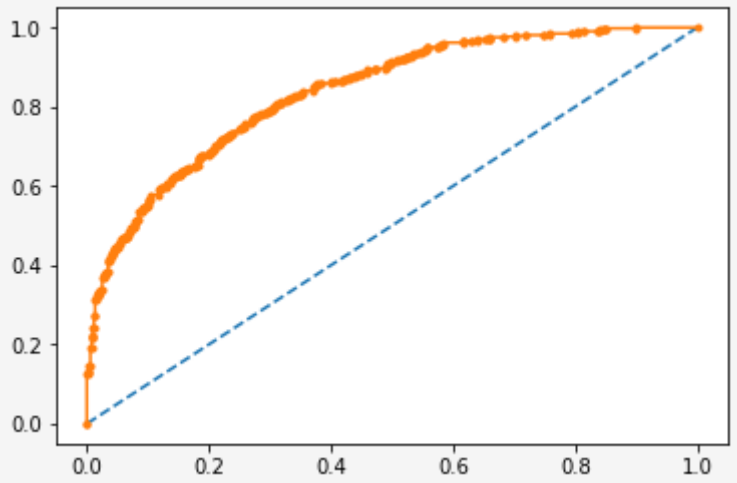
Input:

In [12]:

```
%%time

probabilities = clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]

print(classification_report(y_test,y_pred))

# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
```

Output:

```
              precision    recall  f1-score   support

          -1       0.73      0.78      0.75       518
           1       0.77      0.72      0.74       526

    accuracy                           0.75      1044
   macro avg       0.75      0.75      0.75      1044
weighted avg       0.75      0.75      0.75      1044

ROC curve AUC: 0.839
CPU times: user 9.96 s, sys: 1.97 ms, total: 9.96 s
Wall time: 9.97 s
```



ROC curve AUC: 0.830

Wall time: 10.5 s

We can see in the results that we got a mean accuracy of 75% with an AUC score of 0,839. This means that we have a 83% true positive rate. It's a good start compared to the model with 79% accuracy from Aletras et al. (2016).

To validate our results, let's try to use 10-fold cross validation method to see if AUC keeps up when using multiple folds to test the performance.

Input:

In [14]:

```
%%time


number_of_folds = 10


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds, scoring='roc_auc')

print('Procesverloop AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')

print('Overwegingen AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')

print('Full feature set AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
Procesverloop AUC ROC: 0.646 (+/- std: 0.06)
Overwegingen AUC ROC: 0.765 (+/- std: 0.11)
Full feature set AUC ROC: 0.789 (+/- std: 0.07)
CPU times: user 17min 11s, sys: 494 ms, total: 17min 12s
Wall time: 20min 56s
```

Results:

Component	**AUC**	**Standard Deviation**
Procesverloop	0.646	+/- 0.06
Overwegingen	0.765	+/- 0.11
Full feature set	**0.789**	+/- 0.07

The AUC of the cross validated full feature section is 0,789 compared to the 0,839 non-cross validated results. A loss of 0,050. As shown above, the 'Full feature' section scored the best with with 0.789 AUC ROC.

Let's try to see if the bag-of-words model (n-gram range = 1 instead of 4) scores better.

Input:

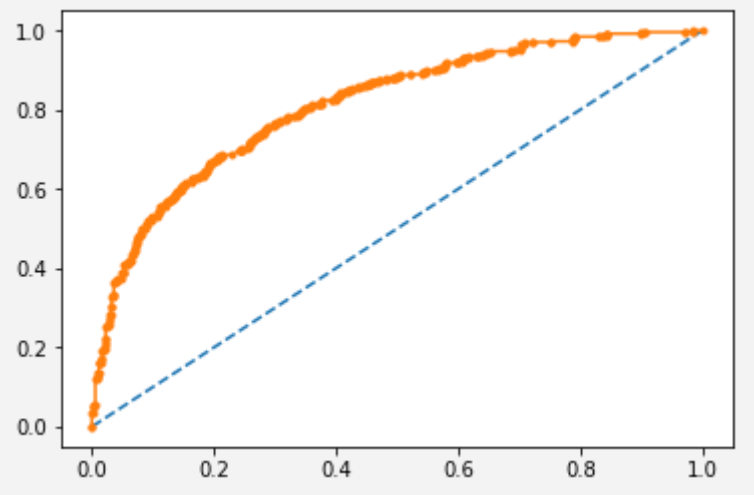
In [24]:

```
%%time

stop_words_nl = nltk.corpus.stopwords.words('dutch')


tfidfVectorizerOverwegingen = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 1))

overwegingen_dataframe = pd.DataFrame(tfidfVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(), columns=tfidfVectorizerOverwegingen.get_feature_names())

tfidfVectorizerProcesverloop = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 1))

procesverloop_dataframe = pd.DataFrame(tfidfVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(), columns=tfidfVectorizerProcesverloop.get_feature_names())


# Merge dataframes

feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)
```

```
X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)
clf = svm.SVC(kernel='linear')
clf.probability = True
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)

probabilities = clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]

print(classification_report(y_test,y_pred))

# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')
```

Output:

```
              precision    recall  f1-score   support

          -1       0.72      0.79      0.75       528
           1       0.76      0.68      0.72       516

    accuracy                           0.74      1044
   macro avg       0.74      0.74      0.73      1044
weighted avg       0.74      0.74      0.73      1044

ROC curve AUC: 0.813
CPU times: user 4min 32s, sys: 237 ms, total: 4min 33s
Wall time: 4min 33s
```



The bag-of-worlds approach show a tiny decrease in results compared to the n-gram 4 range approach. Let's validate the accuracy by 10-fold cross validation

Input:

In [29]:

```
%%time


number_of_folds = 10


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds, scoring='roc_auc')

print('Procesverloop AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')

print('Overwegingen AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')

scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')

print('Full feature set AUC ROC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
Procesverloop AUC ROC: 0.646 (+/- std: 0.06)
Overwegingen AUC ROC: 0.765 (+/- std: 0.11)
Full feature set AUC ROC: 0.789 (+/- std: 0.07)
CPU times: user 15min 57s, sys: 433 ms, total: 15min 58s
Wall time: 19min 21s
```

The cross validated AUC shows that the bag-of-words performance is exactly the same as the N-gram 4 range.

Bag-of-words

Component	**Accuracy**	**Standard Deviation**
Procesverloop	0.646	+/- 0.06
Overwegingen	0.765	+/- 0.11
Full feature set	**0.789**	+/- 0.07

N-gram 4

Component	**Accuracy**	**Standard Deviation**
Procesverloop	0.646	+/- 0.06
Overwegingen	0.765	+/- 0.11
Full feature set	**0.789**	+/- 0.07

Now we try if stemming improves the accuracy of the model.

Input:

In [55]:

```
%%time

import string
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer

def clean_text(text):

    ## Remove punctuation
    text = text.translate(string.punctuation)
```

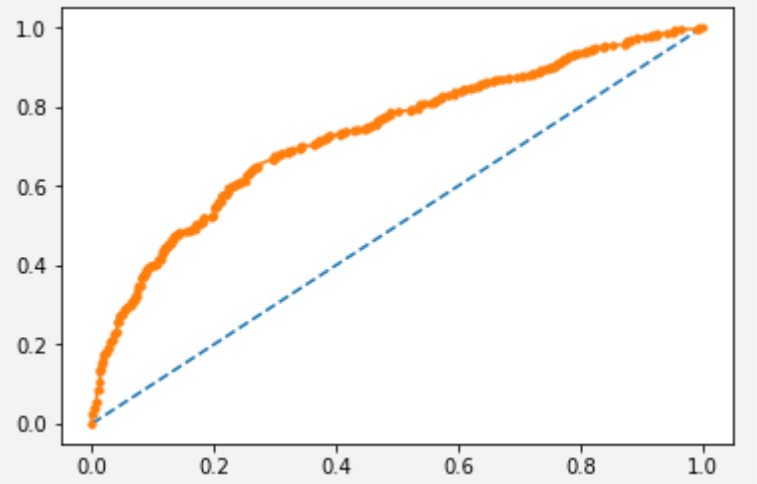
```
    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("dutch"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    ## Stemming
    text = text.split()
    stemmer = SnowballStemmer('dutch')
    stemmed_words = [stemmer.stem(word) for word in text]
    text = " ".join(stemmed_words)

    return text



stop_words_nl = nltk.corpus.stopwords.words('dutch')


tfidfVectorizerOverwegingen = TfidfVectorizer(max_features=1000, tokenizer=clean_text,
stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
overwegingen_dataframe =
pd.DataFrame(tfidfVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(),
columns=tfidfVectorizerOverwegingen.get_feature_names())
tfidfVectorizerProcesverloop = TfidfVectorizer(max_features=1000, tokenizer=clean_text,
stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
procesverloop_dataframe =
pd.DataFrame(tfidfVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(),
columns=tfidfVectorizerProcesverloop.get_feature_names())


# Merge dataframes
feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)


X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)
clf = svm.SVC(kernel='linear')
clf.probability = True
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)


probabilities = clf.predict_proba(X_test)
```

```
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]


print(classification_report(y_test,y_pred))


# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')


number_of_folds = 10


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds, scoring='roc_auc')
print('Procesverloop AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')
print('Overwegingen AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds, scoring='roc_auc')
print('Full feature set accuracy: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
 precision    recall  f1-score   support


        -1       0.65      0.75      0.70       515
         1       0.71      0.61      0.66       529


   accuracy                           0.68      1044
  macro avg       0.68      0.68      0.68      1044
```

```
weighted avg       0.68      0.68      0.68      1044


ROC curve AUC: 0.733
Procesverloop AUC: 0.633 (+/- std: 0.06)
Overwegingen AUC: 0.647 (+/- std: 0.11)
Full feature set accuracy: 0.684 (+/- std: 0.07)
CPU times: user 27min 56s, sys: 1.33 s, total: 27min 58s
Wall time: 31min 52s
```



Results:

Component	Accuracy	Standard Deviation
Procesverloop	0.633	+/- 0.06
Overwegingen	0.647	+/- 0.11
Full feature set	**0.684**	+/- 0.07

Stemming the words decreases the accuracy of the model. Stemming has the potential to increase accuracy by grouping words together and decreasing the total number of words. It seems that in this dataset the loss of specific details (words) causes a decrease in accuracy.

Let's try fine-tuning the parameters of our best performing model. We find these parameters by Grid search.

Input:

In [56]:

```
%%time

stop_words_nl = nltk.corpus.stopwords.words('dutch')
```

```
tfidfVectorizerOverwegingen = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))

overwegingen_dataframe = pd.DataFrame(tfidfVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(), columns=tfidfVectorizerOverwegingen.get_feature_names())

tfidfVectorizerProcesverloop = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))

procesverloop_dataframe = pd.DataFrame(tfidfVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(), columns=tfidfVectorizerProcesverloop.get_feature_names())


# Merge dataframes

feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)


X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values, test_size=0.20)

clf = svm.SVC(kernel='linear')


# Set the parameters by cross-validation

tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5],

                     'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]},

                    {'kernel': ['sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5],

                     'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]},

                    {'kernel': ['linear'], 'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]}

                   ]


scores = ['precision', 'recall']


for score in scores:


    clf = GridSearchCV(svm.SVC(C=1), tuned_parameters, cv=2, scoring='%s_macro' % score)

    clf.fit(X_train, y_train)


    print("Best parameters set found on development set:")

    print()

    print(clf.best_params_)
```

Output:

```
Best parameters set found on development set:


{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}

Best parameters set found on development set:
```

```
{'C': 1000, 'gamma': 0.001, 'kernel': 'rbf'}
CPU times: user 2h 22min 5s, sys: 1.75 s, total: 2h 22min 6s
Wall time: 2h 22min 7s
```

Let's try the best performing model again with these parameters.

Input:

In [57]:

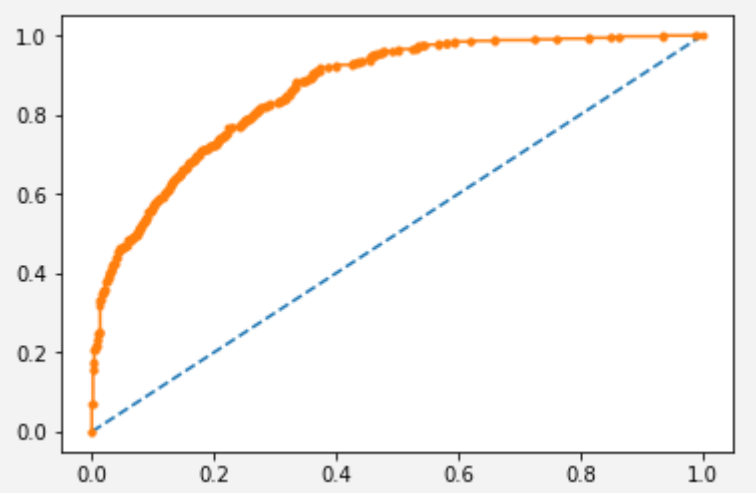
```
%%time

stop_words_nl = nltk.corpus.stopwords.words('dutch')

tfidfVectorizerOverwegingen = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
overwegingen_dataframe = pd.DataFrame(tfidfVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(), columns=tfidfVectorizerOverwegingen.get_feature_names())
tfidfVectorizerProcesverloop = TfidfVectorizer(max_features=1000, stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
procesverloop_dataframe = pd.DataFrame(tfidfVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(), columns=tfidfVectorizerProcesverloop.get_feature_names())

# Merge dataframes
feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)

X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values, test_size=0.20)
clf = svm.SVC(kernel='sigmoid', C=100, gamma=0.01)
clf.probability = True
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)

probabilities = clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]

print(classification_report(y_test,y_pred))

# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
```

```
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')


number_of_folds = 10


clf = svm.SVC(kernel='rbf', C=1000, gamma=0.001)
scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds)
print('Procesverloop AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='rbf', C=1000, gamma=0.001)
scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds)
print('Overwegingen AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='rbf', C=1000, gamma=0.001)
scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds)
print('Full feature set AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
  precision    recall  f1-score   support

        -1       0.76      0.77      0.77       517
         1       0.77      0.77      0.77       527

    accuracy                           0.77      1044
   macro avg       0.77      0.77      0.77      1044
weighted avg       0.77      0.77      0.77      1044

ROC curve AUC: 0.865
Procesverloop AUC: 0.596 (+/- std: 0.07)
Overwegingen AUC: 0.727 (+/- std: 0.10)
Full feature set AUC: 0.731 (+/- std: 0.08)
CPU times: user 20min 14s, sys: 1.26 s, total: 20min 15s
```

```
Wall time: 23min 42s
```



At first it looks like the fine-tuned configuration shows a better AUC but the cross validated AUC shows that the full feature scores 0.731 which is lower than 0.789 of the non fine-tuned configuration.

Below are the results for the feature set with topics. The performance of this model configuration is below 50% and is considered bad. This is because of the incorrect feature engineering of topics.

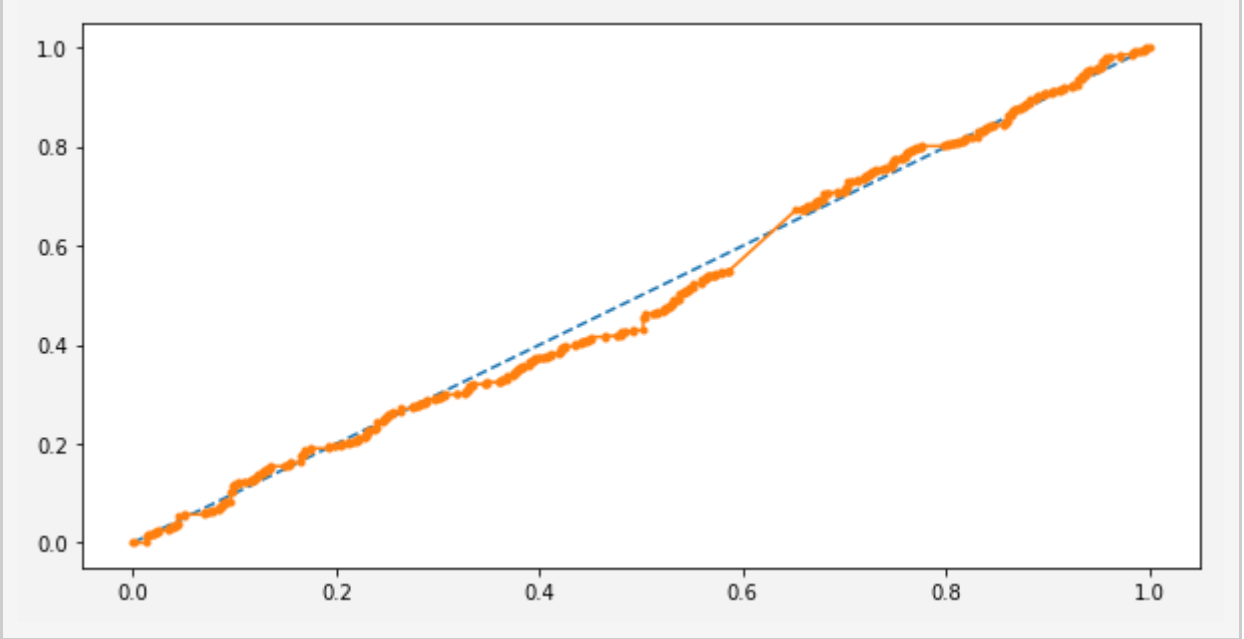
Input:

```
X_train, X_test, y_train, y_test = train_test_split(feature_data_set_with_topics.values,
data_frame['decision'].values, test_size=0.20)
clf = svm.SVC(kernel='sigmoid', C=100, gamma=0.01)
clf.probability = True
model = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)


probabilities = clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]


print(classification_report(y_test,y_pred))


# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
```

```
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')


number_of_folds = 10


clf = svm.SVC(kernel='sigmoid', C=100, gamma=0.01)
scores = model_selection.cross_val_score(clf, feature_data_set_with_topics.values,
data_frame['decision'].values, cv=number_of_folds)
print('Full feature set with topics accuracy: %0.2f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
 precision    recall  f1-score   support


        -1       0.48      0.46      0.47       520
         1       0.49      0.51      0.50       524


   micro avg       0.48      0.48      0.48      1044
   macro avg       0.48      0.48      0.48      1044
weighted avg       0.48      0.48      0.48      1044


ROC curve AUC: 0.493
Full feature set with topics accuracy: 0.50 (+/- std: 0.08)
```



Approach 2: Count vectorized data feeded to SVM

In the previous sections we used normalized term frequency (tf-df) to create features. Let's try another approach, the count vectorizer which only counts the frequency of a specific word.

We start of the with bag-of-words approach

Input:

In [30]:

```
%%time


def train_model_countvectorizer(ngram_range):


    stop_words_nl = nltk.corpus.stopwords.words('dutch')


    countVectorizerOverwegingen = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, ngram_range))

    overwegingen_dataframe =
pd.DataFrame(countVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(),
columns=countVectorizerOverwegingen.get_feature_names())

    countVectorizerProcesverloop = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, ngram_range))

    procesverloop_dataframe =
pd.DataFrame(countVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(),
columns=countVectorizerProcesverloop.get_feature_names())
```

```
    # Merge dataframes
    feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)

    X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)

    clf = svm.SVC(kernel='linear')
    clf.probability = True
    model = clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    y_score = clf.decision_function(X_test)

    probabilities = clf.predict_proba(X_test)
    # keep probabilities for the positive outcome only
    probabilities = probabilities[:, 1]

    print(classification_report(y_test,y_pred))

    # calculate AUC
    roc_auc = roc_auc_score(y_test, probabilities)
    print('ROC curve AUC: %.3f' % roc_auc)
    # calculate roc curve
    fpr, tpr, thresholds = roc_curve(y_test, probabilities)
    # plot no skill
    plt.plot([0, 1], [0, 1], linestyle='--')
    # plot the roc curve for the model
    plt.plot(fpr, tpr, marker='.')



    number_of_folds = 10

    clf = svm.SVC(kernel='linear')
    scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds)
    print('Procesverloop AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))

    clf = svm.SVC(kernel='linear')
    scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds)
    print('Overwegingen AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

```
    clf = svm.SVC(kernel='linear')
    scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds)
    print('Full feature set AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

```
CPU times: user 10 µs, sys: 0 ns, total: 10 µs
Wall time: 15 µs
```

The bag-of-words (n=1) model first

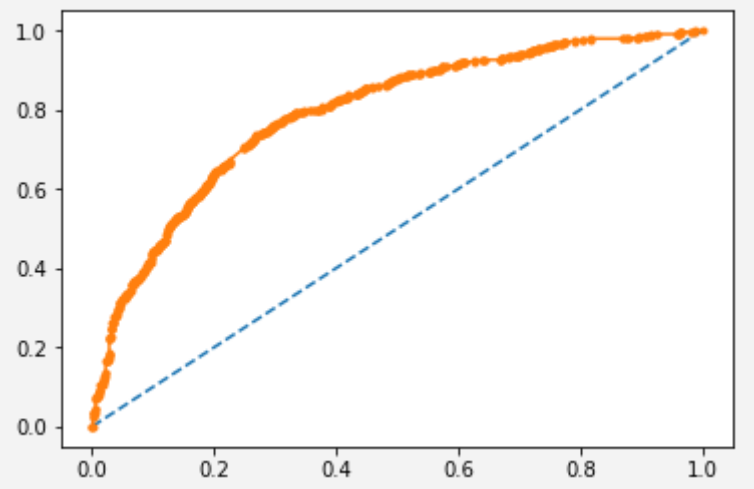
Input:

In [31]:

```
%%time


train_model_countvectorizer(1)
```

Output:

```
              precision    recall  f1-score   support

          -1       0.72      0.73      0.73       841
           1       0.74      0.74      0.74       881

    accuracy                           0.73      1722
   macro avg       0.73      0.73      0.73      1722
weighted avg       0.73      0.73      0.73      1722

ROC curve AUC: 0.792
Procesverloop AUC: 0.564 (+/- std: 0.06)
Overwegingen AUC: 0.701 (+/- std: 0.09)
Full feature set AUC: 0.696 (+/- std: 0.08)
CPU times: user 20min 20s, sys: 624 ms, total: 20min 21s
Wall time: 29min 39s
```



Count vectorizer with n-gram range 4

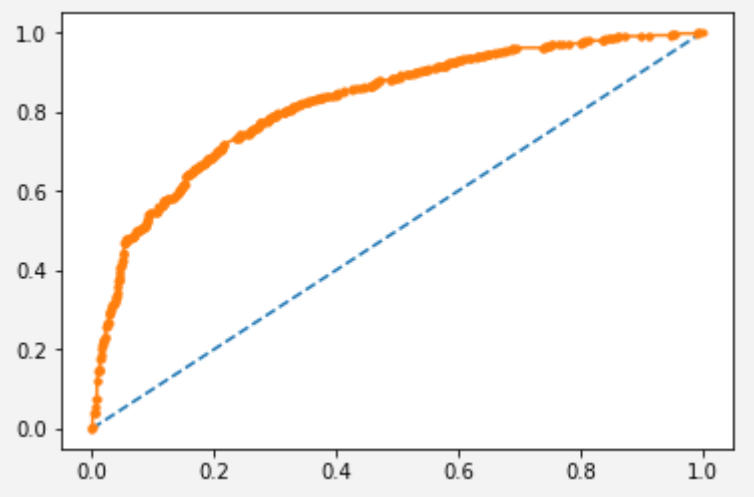
Input:

In [32]:

```
%%time

train_model_countvectorizer(4)
```

Output:

```
 precision    recall  f1-score   support

          -1       0.73      0.74      0.74       837
           1       0.75      0.74      0.75       885

    accuracy                           0.74      1722
   macro avg       0.74      0.74      0.74      1722
weighted avg       0.74      0.74      0.74      1722

ROC curve AUC: 0.822
Procesverloop AUC: 0.557 (+/- std: 0.05)
Overwegingen AUC: 0.729 (+/- std: 0.11)
Full feature set AUC: 0.715 (+/- std: 0.10)
CPU times: user 19min 38s, sys: 1.72 s, total: 19min 40s
Wall time: 31min 47s
```



The count vectorizer shows a small decrease in performance against the tf-idf vectorizer.

Let's try the best configuration with stemming.

Input:

In [32]:

```
%%time

import string
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer

def clean_text(text):

    ## Remove punctuation
    text = text.translate(string.punctuation)

    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("dutch"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    ## Stemming
    text = text.split()
```

```
    stemmer = SnowballStemmer('dutch')
    stemmed_words = [stemmer.stem(word) for word in text]
    text = " ".join(stemmed_words)

    return text



stop_words_nl = nltk.corpus.stopwords.words('dutch')


countVectorizerOverwegingen = CountVectorizer(max_features=1000, tokenizer=clean_text,
stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
overwegingen_dataframe =
pd.DataFrame(countVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(),
columns=countVectorizerOverwegingen.get_feature_names())
countVectorizerProcesverloop = CountVectorizer(max_features=1000, tokenizer=clean_text,
stop_words=stop_words_nl, lowercase=True, ngram_range=(1, 4))
procesverloop_dataframe =
pd.DataFrame(countVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(),
columns=countVectorizerProcesverloop.get_feature_names())


# Merge dataframes
feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)


X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)
clf = svm.SVC(kernel='linear')
clf.probability = True
model = clf.fit(X_train, y_train)


y_pred = clf.predict(X_test)
y_score = clf.decision_function(X_test)


probabilities = clf.predict_proba(X_test)
# keep probabilities for the positive outcome only
probabilities = probabilities[:, 1]


print(classification_report(y_test,y_pred))


# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
```

```
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')


number_of_folds = 10


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values, n_jobs=-1, cv=number_of_folds, scoring='roc_auc')
print('Procesverloop AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values, cv=number_of_folds, scoring='roc_auc')
print('Overwegingen AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='linear')
scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values, cv=number_of_folds, scoring='roc_auc')
print('Full feature set accuracy: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2)
```

Output:

In [32]:

```
                precision    recall  f1-score   support

          -1       0.61      0.66      0.63       492
           1       0.67      0.63      0.65       552

    accuracy                           0.64      1044
   macro avg       0.64      0.64      0.64      1044
weighted avg       0.64      0.64      0.64      1044

ROC curve AUC: 0.687
```

Somehow the above statements are executing extremely slow in Google Colab and after a while it timeouts. I don't have the cross validated data of this part, but we can already see that it performs worse with stemming. Same results as in our first approach.

Now let's do some fine-tuning on our best performing model.

Input:

In [32]:

```
%%time


stop_words_nl = nltk.corpus.stopwords.words('dutch')


countVectorizerOverwegingen = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, 4))

overwegingen_dataframe =
pd.DataFrame(countVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(),
columns=countVectorizerOverwegingen.get_feature_names())

countVectorizerProcesverloop = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, 4))

procesverloop_dataframe =
pd.DataFrame(countVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(),
columns=countVectorizerProcesverloop.get_feature_names())


# Merge dataframes

feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)


X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)

clf = svm.SVC(kernel='linear')


# Set the parameters by cross-validation

tuned_parameters = [{'kernel': ['rbf'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5],
                     'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]},
                    {'kernel': ['sigmoid'], 'gamma': [1e-2, 1e-3, 1e-4, 1e-5],
                     'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]},
                    {'kernel': ['linear'], 'C': [0.001, 0.10, 0.1, 10, 25, 50, 100, 1000]}
                   ]


scores = ['precision', 'recall']


for score in scores:


    clf = GridSearchCV(svm.SVC(C=1), tuned_parameters, cv=2, scoring='%s_macro' % score)

    clf.fit(X_train, y_train)


    print("Best parameters set found on development set:")

    print()

    print(clf.best_params_)
```

Output:

In [32]:

```
{'C': 50, 'gamma': 1e-05, 'kernel': 'rbf'}
Best parameters set found on development set:


{'C': 100, 'gamma': 1e-05, 'kernel': 'rbf'}
CPU times: user 1h 56min 4s, sys: 2.6 s, total: 1h 56min 6s
Wall time: 1h 56min 7s
```

Let's try the best performing model from approach 2 with fine-tuned parameters.

Input:

In [32]:

```
 stop_words_nl = nltk.corpus.stopwords.words('dutch')


  countVectorizerOverwegingen = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, 4))
  overwegingen_dataframe =
pd.DataFrame(countVectorizerOverwegingen.fit_transform(data_frame['overwegingen']).todense(),
columns=countVectorizerOverwegingen.get_feature_names())
  countVectorizerProcesverloop = CountVectorizer(max_features=1000, stop_words=stop_words_nl,
lowercase=True, ngram_range=(1, 4))
  procesverloop_dataframe =
pd.DataFrame(countVectorizerProcesverloop.fit_transform(data_frame['procesverloop']).todense(),
columns=countVectorizerProcesverloop.get_feature_names())


  # Merge dataframes
  feature_data_set = pd.concat([overwegingen_dataframe, procesverloop_dataframe], axis=1)


  X_train, X_test, y_train, y_test = train_test_split(feature_data_set.values, data_frame['decision'].values,
test_size=0.20)


  clf = svm.SVC(kernel='linear')
  clf.probability = True
  model = clf.fit(X_train, y_train)
  y_pred = clf.predict(X_test)
  y_score = clf.decision_function(X_test)


  probabilities = clf.predict_proba(X_test)
  # keep probabilities for the positive outcome only
  probabilities = probabilities[:, 1]
```

```
print(classification_report(y_test,y_pred))


# calculate AUC
roc_auc = roc_auc_score(y_test, probabilities)
print('ROC curve AUC: %.3f' % roc_auc)
# calculate roc curve
fpr, tpr, thresholds = roc_curve(y_test, probabilities)
# plot no skill
plt.plot([0, 1], [0, 1], linestyle='--')
# plot the roc curve for the model
plt.plot(fpr, tpr, marker='.')



number_of_folds = 10


clf = svm.SVC(kernel='rbf', C=100, gamma=1e-05)
scores = model_selection.cross_val_score(clf, procesverloop_dataframe.values, data_frame['decision'].values,
n_jobs=-1, cv=number_of_folds)
print('Procesverloop AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='rbf', C=100, gamma=1e-05)
scores = model_selection.cross_val_score(clf, overwegingen_dataframe.values, data_frame['decision'].values,
cv=number_of_folds)
print('Overwegingen AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))


clf = svm.SVC(kernel='rbf', C=100, gamma=1e-05)
scores = model_selection.cross_val_score(clf, feature_data_set.values, data_frame['decision'].values,
cv=number_of_folds)
print('Full feature set AUC: %0.3f (+/- std: %0.2f)' % (scores.mean(), scores.std() * 2))
```

Output:

In [32]:

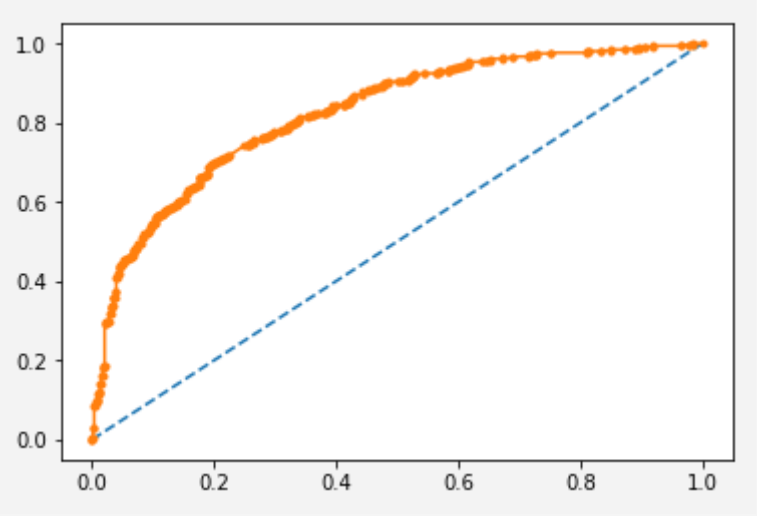
```
 precision    recall  f1-score   support


        -1       0.75      0.74      0.75       525
         1       0.74      0.74      0.74       519


    accuracy                           0.74      1044
   macro avg       0.74      0.74      0.74      1044
weighted avg       0.74      0.74      0.74      1044
```

ROC curve AUC: 0.825

Procesverloop AUC: 0.594 (+/- std: 0.06)

Overwegingen AUC: 0.747 (+/- std: 0.11)

Full feature set AUC: 0.750 (+/- std: 0.10)



Approach 3: Recurrent Neural Network LSTM

Another approach that is used frequently for text classification is the Recurrent Neural Network (RNN), specifically Long short-term memory network (LSTM). Let's try if we can get decent results via this approach.

The LSTM model is created based on following steps:

1. cleanup the text, by transforming the text to lowercase, removing stopwords, removing punctuations and a optional step of stemming the Dutch text.
2. tokenize the data.
3. pad the sequences
4. feed the data to the RNN - LSTM
5. score the performance.

We start by importing modules

Input:

In [6]:

```
 import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
import re
from nltk.corpus import stopwords
from nltk import word_tokenize
STOPWORDS = set(stopwords.words('dutch'))
from bs4 import BeautifulSoup
import plotly.graph_objs as go
import plotly.plotly as py
import cufflinks
from IPython.core.interactiveshell import InteractiveShell
import plotly.figure_factory as ff
InteractiveShell.ast_node_interactivity = 'all'
```

```
from plotly.offline import iplot
from sklearn.metrics import roc_auc_score
```

Output:

```
Using TensorFlow backend.
```

The first thing we will do is cleaning the text.

Input:

```
%%time

import string
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer

def clean_text(text):

    ## Remove punctuation
    text = text.translate(string.punctuation)

    ## Convert words to lower case and split them
    text = text.lower().split()

    ## Remove stop words
    stops = set(stopwords.words("dutch"))
    text = [w for w in text if not w in stops and len(w) >= 3]

    text = " ".join(text)

    return text

# apply the above function to data frame
data_frame['overwegingen'] = data_frame['overwegingen'].map(lambda x: clean_text(x))
data_frame['procesverloop'] = data_frame['procesverloop'].map(lambda x: clean_text(x))
```

Output:

```
CPU times: user 6.47 s, sys: 201 ms, total: 6.67 s
```

```
Wall time: 6.68 s
```

After cleaning the text, we need to tokenize the text and pad the sequences so they have an equal size.

Input:

```
MAX_WORDS=2000
# We use the mean length of the column because some text are over 80.000 characters
MAX_SEQUENCE_LENGTH=int(data_frame['overwegingen'].str.len().mean())
EMBEDDING_DIM=100


tokenizer = Tokenizer(num_words= MAX_WORDS)
X = tokenizer.texts_to_sequences(data_frame['overwegingen'].iloc[:1000].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)



print('Shape of data tensor:', X.shape)
```

Output:

```
Shape of data tensor: (1000, 4430)
```

Because of the size of texts, we cannot use the full text because a RNN consumes a lot of memory when you have over 80.000 words. The 12 gigabyte of memory available cannot handle these large texts. Because of this we use a max length of the mean of all string lengths in the data frame.

We create a model with a Word embedding layer and a LSTM layer with 320 nodes based on what is used in the research of Menger et al. (2018).

Input:

```
from keras import backend as K
import tensorflow as tf



def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)


# Build Model...
model = Sequential()
model.add(Embedding(MAX_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
```

```
model.add(LSTM(320, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(1, activation='softmax'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', auroc])
print(model.summary())
```

Output:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 4430, 100)         200000
_________________________________________________________________
lstm_6 (LSTM)                (None, 320)               538880
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 321
=================================================================
Total params: 739,201
Trainable params: 739,201
Non-trainable params: 0
_________________________________________________________________
None
```

Now we train the model for 20 epochs. We use an 'EarlyStopping' callback, this means that if the loss value stays the same it will stop earlier than 20 epochs because there is no learning improvement. This way we don't have to wait for a long for nothing.

Input:

```
epochs = 20
batch_size = 64


X_train, X_test, y_train, y_test = train_test_split(X, data_frame['decision'].iloc[:1000], test_size=0.20)


history = model.fit(X_train, y_train, epochs=epochs,
batch_size=batch_size,callbacks=[EarlyStopping(monitor='loss', patience=3, min_delta=0.0001)])
```

Output:

```
Epoch 1/20
800/800 [==============================] - 542s 678ms/step - loss: 14.1090 - acc: 0.5575 - auroc:
0.5000
Epoch 2/20
```

```
800/800 [==============================] - 539s 674ms/step - loss: 14.1090 - acc: 0.5575 - auroc: 0.5000

Epoch 3/20

800/800 [==============================] - 541s 677ms/step - loss: 14.1090 - acc: 0.5575 - auroc: 0.5000

Epoch 4/20

800/800 [==============================] - 541s 676ms/step - loss: 14.1090 - acc: 0.5575 - auroc: 0.5000
```

With the model trained, we can now calculate the AUC.

Input:

```
from sklearn.metrics import roc_auc_score

y_pred = model.predict_proba(X_test)

roc_auc_score(y_test, y_pred)
```

Output:

```
0.5
```

The RNN does not seem to be learning with an AUC of 0.5. An AUC score of 0.5 indicates the model has no class separation capabilities.

Let's see if the 'procesverloop' section performs differently.

Input:

```
MAX_SEQUENCE_LENGTH=int(data_frame['procesverloop'].str.len().mean())


X = tokenizer.texts_to_sequences(data_frame['procesverloop'].iloc[:1000].values)

X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)


X_train, X_test, y_train, y_test = train_test_split(X, data_frame['decision'].iloc[:1000], test_size=0.20)


def auroc(y_true, y_pred):
    return tf.py_func(roc_auc_score, (y_true, y_pred), tf.double)


# Build Model...

model = Sequential()

model.add(Embedding(MAX_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))

model.add(LSTM(320, dropout=0.2, recurrent_dropout=0.2))

model.add(Dense(1, activation='softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy', auroc])
```

```
print(model.summary())


epochs = 20

batch_size = 64


history = model.fit(X_train, y_train, epochs=epochs, batch_size=batch_size,
callbacks=[EarlyStopping(monitor='loss', patience=3, min_delta=0.0001)])
```

Output:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_8 (Embedding)      (None, 1682, 100)         200000
_________________________________________________________________
lstm_8 (LSTM)                (None, 320)               538880
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 321
=================================================================
Total params: 739,201
Trainable params: 739,201
Non-trainable params: 0
_________________________________________________________________
None
Epoch 1/20
800/800 [==============================] - 201s 252ms/step - loss: 14.4677 - acc: 0.5463 - auroc:
0.5000
Epoch 2/20
800/800 [==============================] - 200s 249ms/step - loss: 14.4677 - acc: 0.5463 - auroc:
0.5000
Epoch 3/20
800/800 [==============================] - 199s 249ms/step - loss: 14.4677 - acc: 0.5463 - auroc:
0.5000
Epoch 4/20
800/800 [==============================] - 200s 250ms/step - loss: 14.4677 - acc: 0.5463 - auroc:
0.5000
```

Calculate the AUC for 'procesverloop'.

Input:

```
from sklearn.metrics import roc_auc_score
```

```
y_pred = model.predict_proba(X_test)
roc_auc_score(y_test, y_pred)
```

Output:

```
0.5
```

Since the RNN on both 'procesverloop' and 'overwegingen' is not learning, it has not extra value to train it on the full case because that is a combination of both. The architecture of our RNN model or the input data is causing the model not to learn.