Prototype selection for the nearest neighbour rule through proximity graphs

Mathijs Vos Ramon Janssen Petra van den Bos

26 maart 2012

Inhoudsopgave

Samenvatting

Het k-nearest-neighbour-principe kan geïmplementeerd worden door gebruik te maken van proximity graphs. Van de trainset wordt een graaf gemaakt waarbij de buren van een punt kunnen stemmen volgens het kNN-principe. Bij proximity graphs kan eerste orde en tweede orde editing toegepast worden, wat invloed heeft op welke punten kunnen stemmen in kNN.

In dit verslag worden Gabriel Graphs en Relative Neighbourhood Graphs besproken, en hoe hier door middel van editing en condensing een prototype van te maken is. De implementatie hiervan is getest op een aantal testsets en de resultaten hiervan worden genoemd en besproken. Tot slot worden de voor- en nadelen van dit algoritme besproken.

1 Inleiding

Een belangrijk probleem in data-analyse is dat van classificatie. Hoe deel je verschillende objecten of personen in, op basis van bepaalde kenmerken? Hiervoor is in de classificatie een algemeen stappenplan beschikbaar: van een aantal gevallen worden de kenmerken bepaald, en deze worden stuk voor stuk handmatig ingedeeld in een categorie. Een programma gebruikt deze gegevens als trainset, en verwerkt deze gegevens. Het programma kan daarna van onbekende gevallen de klasse voorspellen, als de kenmerken gegeven worden. Hierbij is te denken aan het classificeren van plaatjes op basis van hun pixels, het stellen van een diagnose op basis van medische kenmerken, en vele andere soorten problemen.

Een veelgebruikt criterium om punten te classificeren is het k-nearest-neighbour-principe. Hierbij wordt het k aantal punten bekeken dat het dichts bij het te testen punt ligt. Dit kan gedaan worden met de euclidische afstand van alle eigenschappen, maar dit verschilt per implementatie. De "buren"die zo gevonden worden, stemmen over welke klasse het nieuwe punt moet zijn. Het kNN-principe kan gemakkelijk worden toegepast op verschillende soorten datasets, doordat de afstand gemakkelijk onafhankelijk van het aantal punten en het aantal eigenschappen van de punten kan worden berekend.

2 Het Probleem

2.1 Condensing en editing

Het kNN-principe kan op een simpele manier op alle soorten gegevens worden toegepast. Door een nieuw punt simpelweg met alle punten in de trainset te vergelijken kan een goed resultaat worden verkregen. In de praktijk levert dit echter problemen op in performance, omdat een dataset al snel veel dimensies en punten bevat. Ook wordt ruis daarmee overgenomen, omdat van elk punt van de trainset wordt aangenomen dat deze correct gelabeld is. De complexiteit voor het testen van nieuwe punten wordt dan al snel veel te groot en daardoor zijn optimalisaties vereist.

Het doel van condensing is om de grootte van een dataset te reduceren door redundante informatie te schrappen. Dit richt zich voornamelijk op gebieden waar zich veel punten met dezelfde klasse bevinden: de punten in het midden van het gebied voegen niets toe ten opzichte van de punten op de rand. Alleen de randen worden dus gespaard. Hierbij zorgt ruis voor een probleem, omdat één (verkeerd gelabeld) punt in het midden van een gebied er anders voor zou zorgen dat het midden niet meer condensed kan worden.

Voor het weghalen van ruis is er editing. Dit haalt punten weg die nabij veel punten met een andere classificatie liggen, omdat dit soort punten meer kans hebben om ruis te zijn. Dit kan het beste vóór het condensen gebeuren, zodat de condensing minder last heeft van ruis.

Voor zowel condensing als editing kan de precieze werking verschillen. Het is afhankelijk van het algoritme hoeveel informatie er verwijderd wordt. Als er meer informatie verwijderd

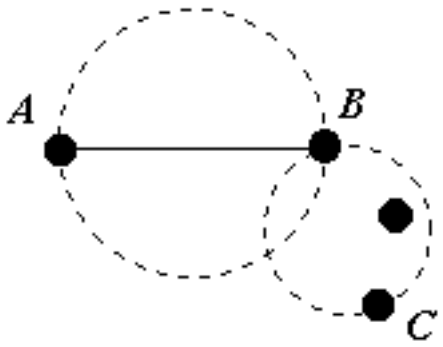
wordt, worden de resultaten gemiddeld slechter maar wordt de benodigde tijd voor het testen minder. Het verschilt ook per algoritme hoe effectief editing en condensing is onder verschillende omstandigheden, zoals de hoeveelheid dimensies en de vorm van de data.

2.2 De methode

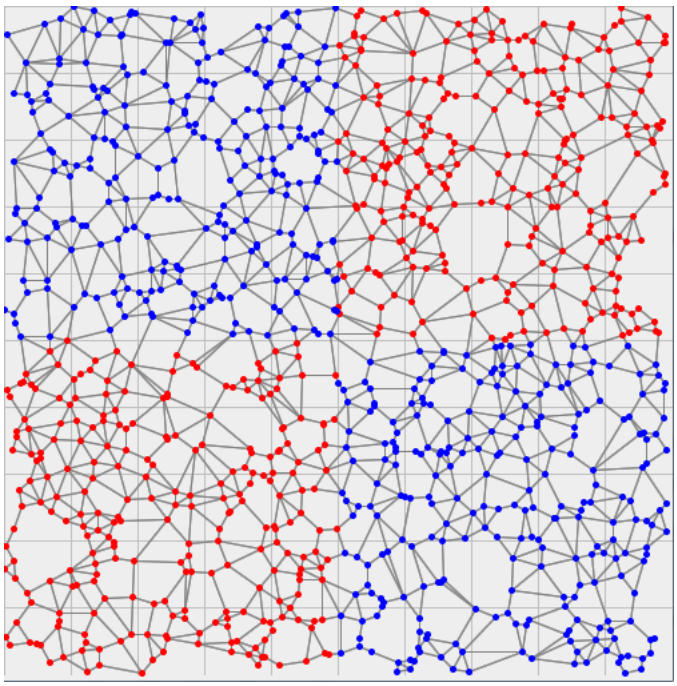
De titel van het artikel dat wij gekozen hebben is "Prototype selection for the nearest neighbour rulethrough proximity graphs". We hebben dit artikel gekozen omdat het ons wel leuk leek om grafen te gebruiken en omdat het artikel duidelijk was. Er worden twee soorten grafen beschreven: *Gabriel Graphs* en *Relative Neighbourhood Graphs*. Beiden zijn grafen waarin punten verbonden worden door lijnen die elkaar niet snijden. Als een punt A verbonden is met punt B, is A een buur van B en B een buur van A.

2.2.1 Gabriel Graph

In een Gabriel Graph is er een verbinding tussen twee punten, als binnen de cirkel, waarvan de verbinding de diameter is, geen andere punten liggen. Met de euclidische afstand kan dit formeel uitgedrukt worden: A en B hebben een verbinding als voor alle andere punten X geldt: $afstand^2(A,B) \leq afstand^2(A,X) + afstand^2(B,X)$. In Figuur 1 zijn A en B verbonden, maar B en C niet. In Figuur 2 is een gehele Gabriel Graph te zien.



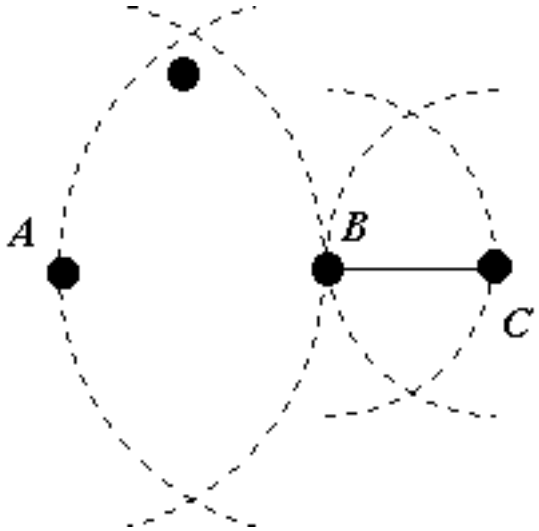
Figuur 1: Buren van punten in een Gabriel Graph



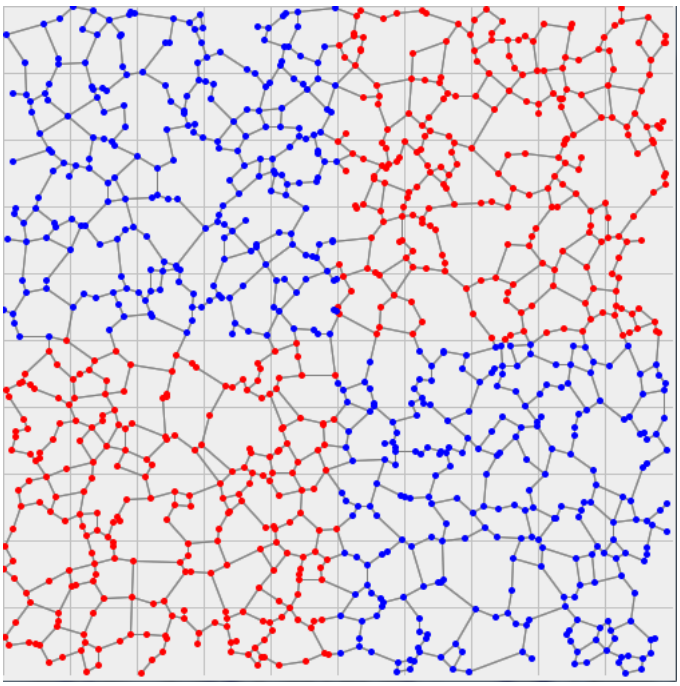
Figuur 2: Een Gabriel Graph

2.2.2 Relative Neighbourhood Graph

In een Relative Neighbourhood Graph hebben twee punten een verbinding als er geen andere punten binnen de 'lune' van die twee punten liggen, zie Figuur 3. Met de euclidische afstand kan dit formeel uitgedrukt worden: A en B hebben een verbinding als voor alle andere punten X geldt: $afstand(A, B) \leq max\{afstand(A, X), afstand(B, X)\}$. In Figuur 3 zijn B en C verbonden, maar A en B niet. In Figuur 4 is een gehele Relative Neighbourhood Graph te zien.



Figuur 3: Buren van punten in een Relative Neighbourhood Graph



Figuur 4: Een relative neighbourhood graph

Deze grafen kun je gebruiken om nieuwe data (de testset) te klassificeren aan de hand van data waarvan de classificatie al bekend is (de trainset). Een graaf bevat in eerste instantie dus alle punten uit de trainset. Om de performance te verbeteren en eventuele ruis te compenseren, wordt condensing en editing toegepast op de trainset. Hierbij wordt gekeken naar de punten die een verbinding hebben met een bepaald punt. Bij condensing wordt ieder punt weggehaald, dat alleen verbindingen naar punten met dezelfde klasse heeft. In het artikel werden twee soorten editing algoritmes besproken: een eerste orde editing algoritme en een tweede orde editing algoritme. Het eerste orde editing algoritme bepaalt voor elk punt of de meerderheid van de buren van dat punt tot een van de andere klasses behoort. De punten waarvoor dit geldt, worden verwijderd. Het tweede orde algoritme verwijdert alle punten als geldt dat de meerderheid van de buren tot een van de andere klasses behoort, en als ook de meerderheid van de buren, van de buren die van dezelfde klasse als dat punt zijn, tot een van de andere klasses behoort.

De testset wordt geklassificeerd op de volgende manier: Eerst wordt de graaf van alle punten van de trainset geconstrueerd. Vervolgens wordt eerste of tweede orde editing toegepast. Daarna wordt condensing toegepast. Met de bewerkte graaf die overblijft wordt een klasse aan elk van de punten in de testset toegewezen. Dit gebeurt met behulp van het k-nearest neighbour algoritme. Van elk punt in de testset wordt gekeken welke k punten het dichtstbij liggen. Deze k dichtstbijzijnde punten bepalen we door ofwel alleen naar de directe buren van het testsetpunt te kijken (eerste orde), ofwel naar de directe buren en buren van die directe buren te kijken (tweede orde). Als k groter is dan het aantal buren dat het testsetpunt heeft, worden alleen die buren gebruikt. Daardoor is k voor dat punt dus kleiner.

3 Experimenten

We hebben onze methode met verschillende datasets getest:

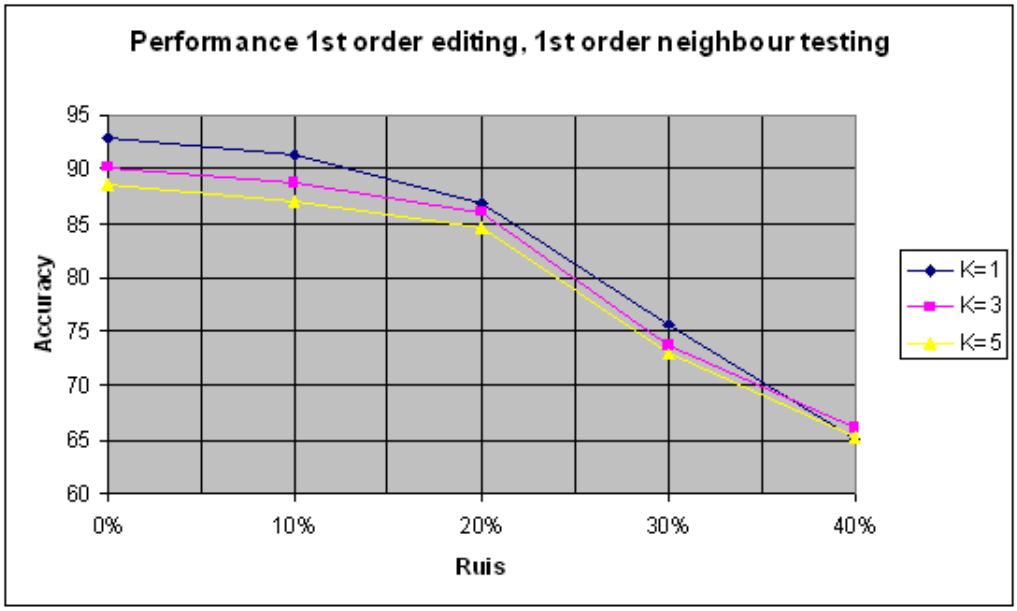
1. De XOR-dataset met 2 dimensies en 2 mogelijke klasses.
2. De MNIST-dataset met 196 dimensies en 10 mogelijke klasses.
3. De diabetes-dataset met 8 dimensies en 2 mogelijke klasses.

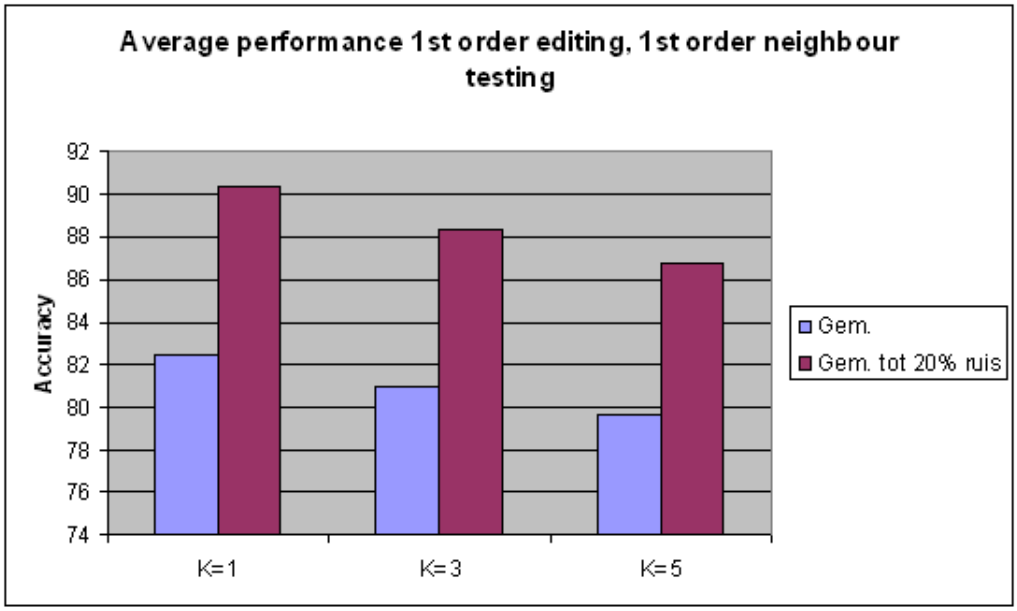
3.1 XOR

We hebben de gemiddelde nauwkeurigheid en reductie voor elk ruispercentage bepaald, door het gemiddelde te nemen van de score van alle testsets (dat waren er 40), toegepast op alle trainsets (er waren 10 trainsets). Hierbij hebben we Gabriel Graphs gebruikt. We hebben deze tests gedaan voor eerste orde en tweede orde editing, in combinatie met eerste orde en tweede orde (k-nearest-neighbour) tests op de punten van de testset. Hieronder staan de testresultaten weergegeven in een tabel en daarbij twee grafieken die de resultaten uit de tabel weergeven.

3.1.1 Eerste orde editing, eerste orde tests

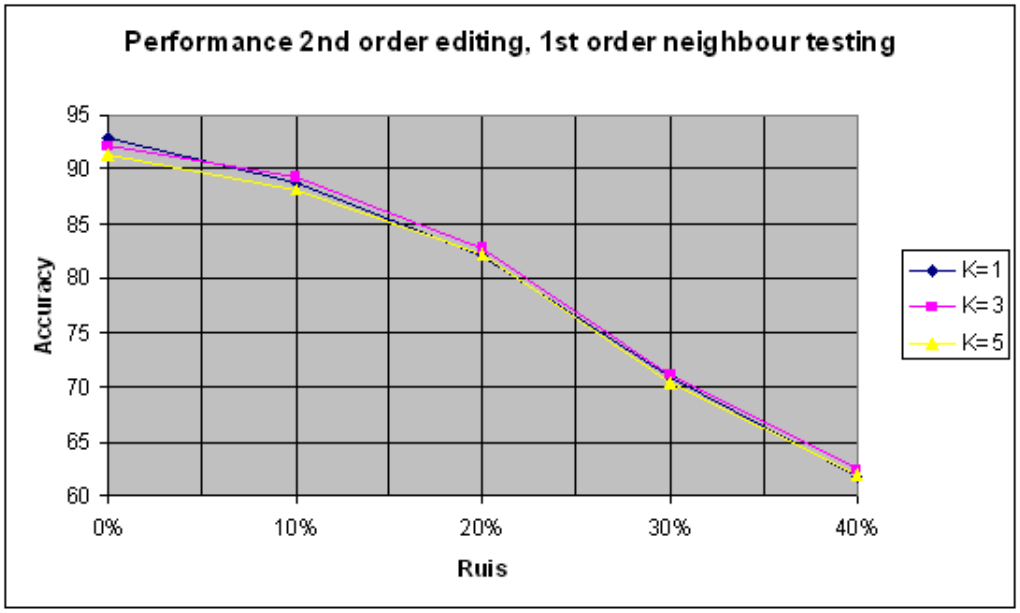
Ruis	K=1	K=3	K=5
0%	92,99	90,17	88,58
10%	91,37	88,69	86,94
20%	86,78	86,07	84,62
30%	75,71	73,76	72,91
40%	65,18	66,11	65,31
Gem.	82,41	80,96	79,67
Gem. tot 20% ruis	90,38	88,31	86,71

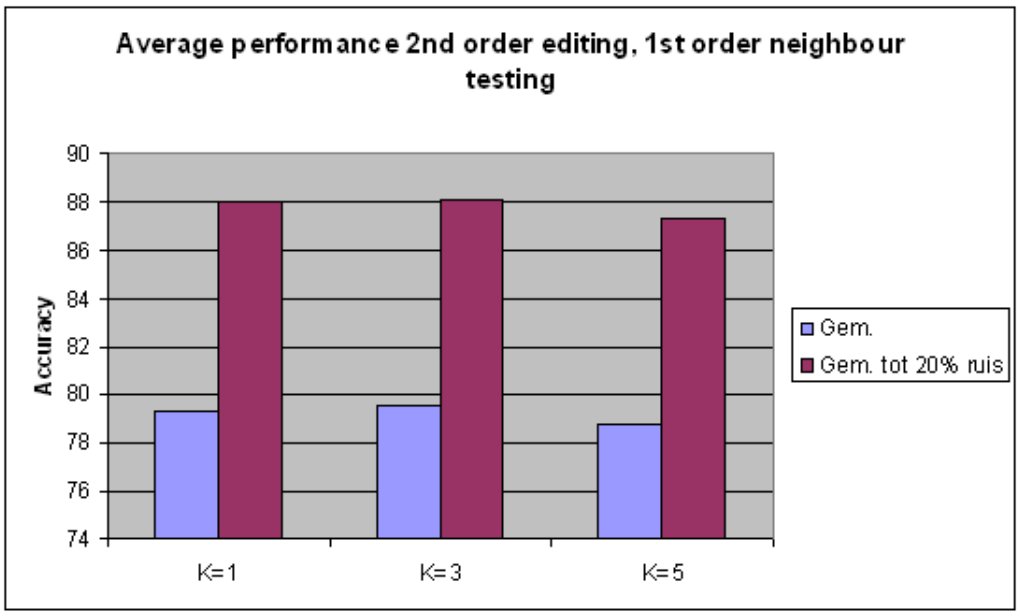




3.1.2 Tweede orde editing, eerste orde tests

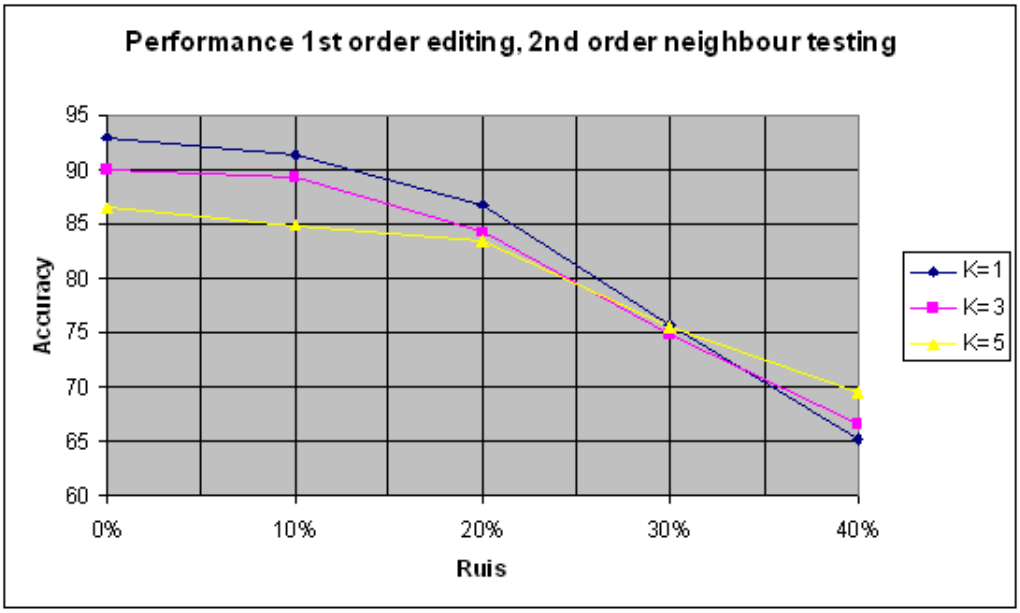
Ruis	K=1	K=3	K=5
0%	93,00	92,17	91,30
10%	88,78	89,20	88,21
20%	82,08	82,72	82,34
30%	70,84	71,06	70,4
40%	61,82	62,33	61,93
Gem.	79,30	79,5	78,83
Gem. tot 20% ruis	87,95	88,03	87,28

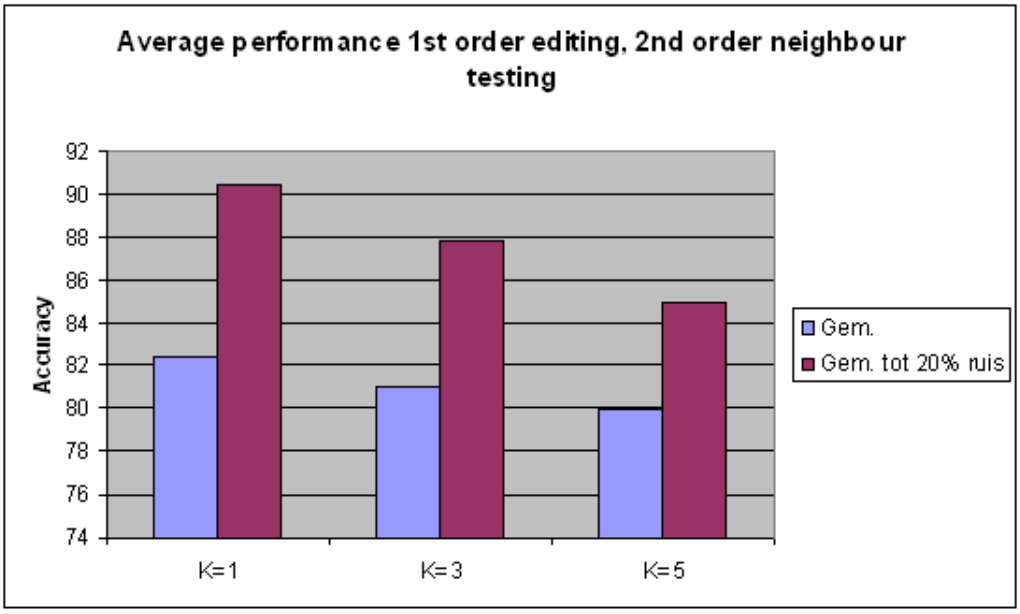




3.1.3 Eerste orde editing, tweede orde tests

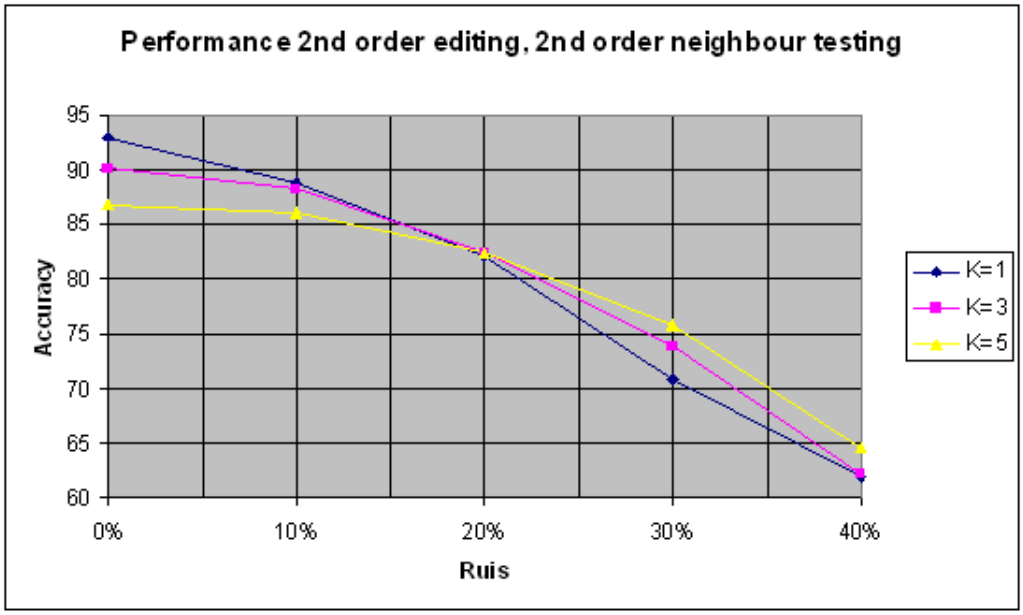
Ruis	K=1	K=3	K=5
0%	92,99	89,94	86,51
10%	91,37	89,29	84,86
20%	86,78	84,17	83,49
30%	75,71	74,83	75,49
40%	65,18	66,72	69,54
Gem.	82,41	80,99	79,98
Gem. tot 20% ruis	90,38	87,80	84,95

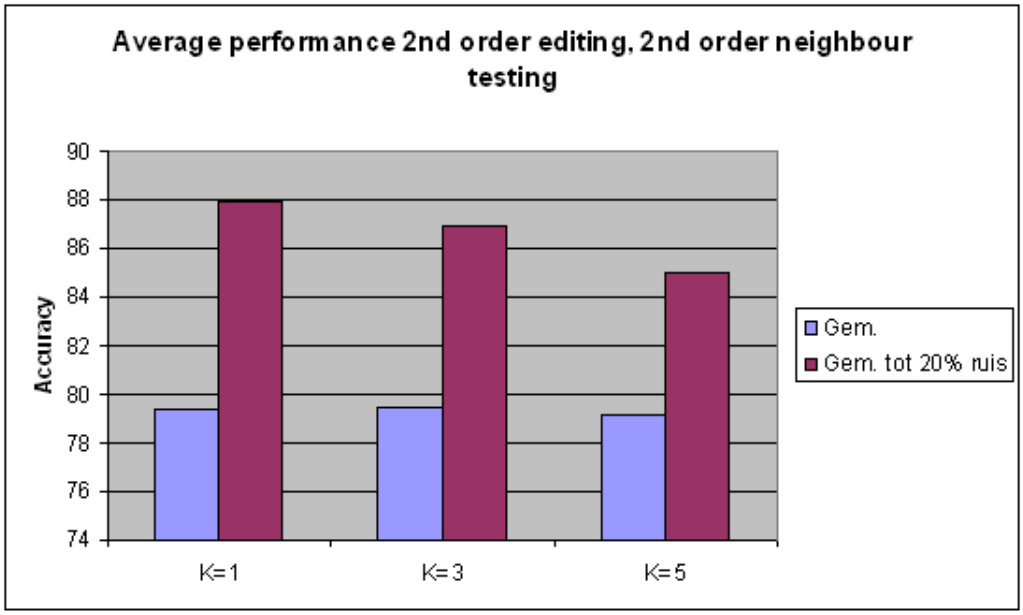




3.1.4 Tweede orde editing, tweede orde tests

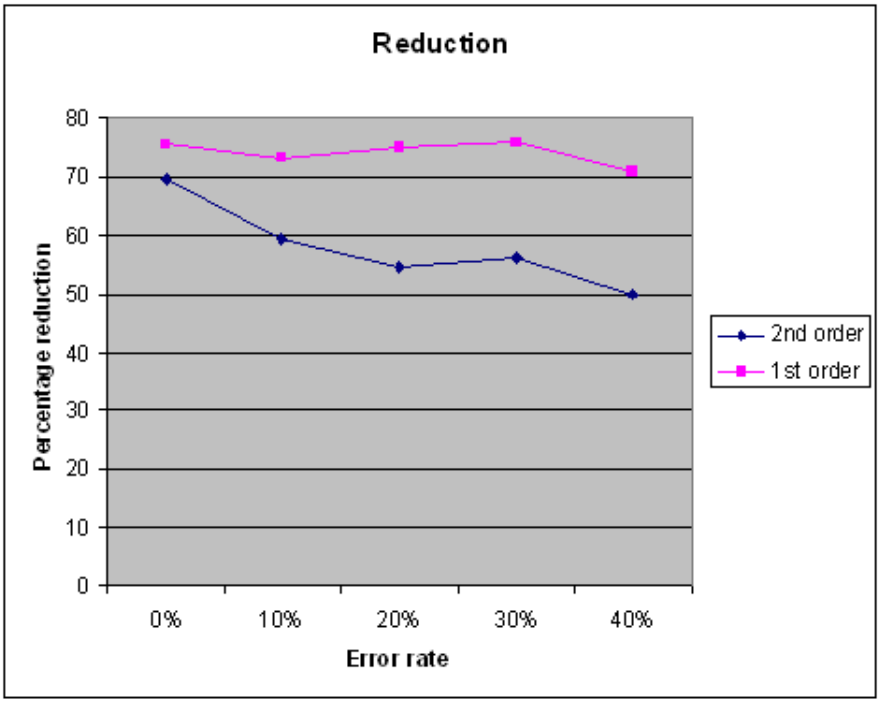
Ruis	K=1	K=3	K=5
0%	93,005	90,06	86,76
10%	88,78	88,25	85,98
20%	82,08	82,46	82,4
30%	70,84	73,93	75,73
40%	61,82	62,23	64,6
Gem.	79,30	79,38	79,09
Gem. tot 20% ruis	87,95	86,92	85,04





3.1.5 Reductie

	0%	10%	20%	30%	40%
2nd order editing	69,7	59,3	54,6	56	50
1st order editing	75,6	73,1	74,9	75,9	70,9

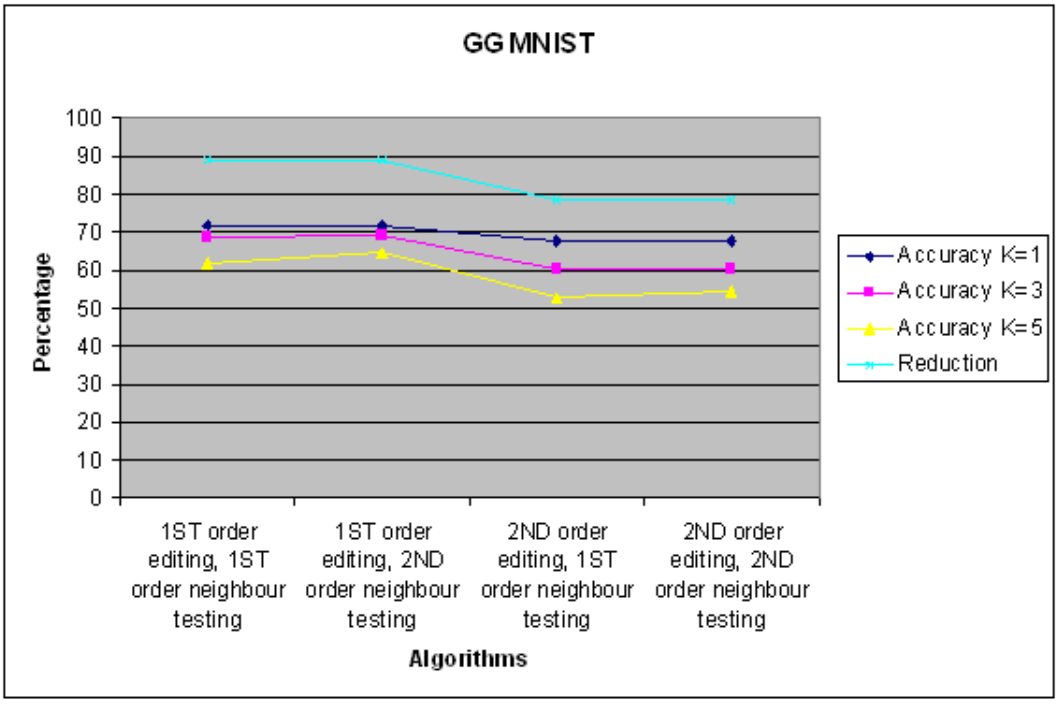


Uit de staafgrafieken blijkt dat voor de nauwkeurigheid in de xor-dataset gemiddeld het beste $k = 1$ gebruikt kan worden voor het k-nearest-neighbour algoritme, als er weinig ruis in de dataset zit. Als er veel ruis in de dataset zit, levert $k = 5$ gemiddeld een zelfde of beter resultaat. Eerste orde editing levert gemiddeld een iets beter resultaat, vooral voor $k = 1$. Eerste orde editing verwijdert meer punten, zeker bij meer ruis.

3.2 MNIST

Van de MNIST-dataset was een trainset en een testset beschikbaar. We kregen de volgende resultaten:

	Accuracy K=1	Accuracy K=3	Accuracy K=5	Reduction
1ST order editing, 1ST order neighbour testing	71,8	68,6	62	89,3
1ST order editing, 2ND order neighbour testing	71,8	69	64,4	89,3
2ND order editing, 1ST order neighbour testing	67,6	60,4	52,8	78,8
2ND order editing, 2ND order neighbour testing	67,6	60,4	54,6	78,8

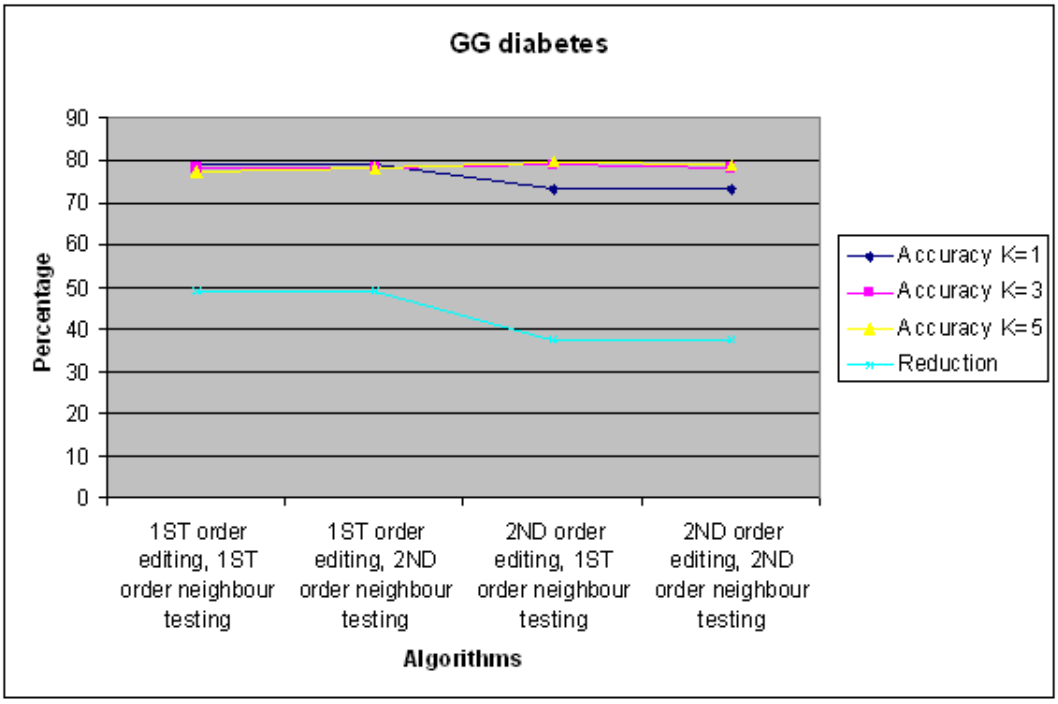


Voor deze dataset blijven de nauwkeurigheid en reductie vrij stabiel, tweede orde editing is iets slechter. Voor alle gevallen geldt dat $k = 1$ de beste keuze voor k is.

3.3 Diabetes

Van de diabetes-dataset was een trainset en een testset beschikbaar. We kregen de volgende resultaten:

	Accuracy K=1	Accuracy K=3	Accuracy K=5	Reduction
1ST order editing, 1ST order neighbour testing	78,9	78,1	77,3	49,2
1ST order editing, 2ND order neighbour testing	78,9	78,1	78,1	49,2
2ND order editing, 1ST order neighbour testing	73,4	78,9	79,7	37,5
2ND order editing, 2ND order neighbour testing	73,4	78,1	78,9	37,5



Voor deze dataset liggen de resultaten voor $k = 3$ en $k = 5$ erg dicht bij elkaar, afhankelijk van het algoritme dat je kiest is de een of de ander beter. $k = 1$ blijkt bij deze dataset juist een slechtere keuze te zijn.

3.4 Performance

De performance van het algoritme hangt sterk af van het aantal punten in de trainset. Door de implementatie van het zoeken van edges in een graaf, is de complexiteit van het maken van een graaf $\theta(d \cdot N^3)$, waarbij d het aantal dimensies is en N het aantal punten in de trainset. Voor elke combinatie van twee punten wordt nu namelijk gekeken of ze buren zijn, door te zoeken naar een derde punt dat er tussenin ligt. Het maken van een proximity graph zou in $\theta(d \cdot NlogN)$ moeten kunnen. Het maken van een prototype hoeft echter geen probleem te zijn: dit wordt eenmalig gedaan door een programma, waarna er onbeperkt geclassificeerd kan worden. De implementatie is nadeliger voor het testen van nieuwe punten. Dit heeft per nieuw punt complexiteit $\theta(d \cdot N^2)$, waarbij N de grootte van het prototype is. Hierdoor is de performance bij grote trainsets, zoals de MNIST-set, vrij slecht. De performance met trainsets en testsets met enkele honderden of minder punten is goed: dit gaat op een moderne computer in de orde van seconden. Om de performance van grotere sets wat te verbeteren is multithreading gebruikt, het algoritme leent zich goed voor een arbitrair aantal threads. Dit verlaagt de complexiteit echter niet en zou dus niet als significante verbetering van het algoritme gezien moeten worden.

4 Conclusie

Proximity graphs zijn geschikt voor het toepassen van het kNN-principe. De effectiviteit verschilde per testset en per parameter in het algoritme. Bij de XOR-set en de diabetes-set was de het percentage fout gelabelde punten ongeveer gelijk aan het percentage ruis. Het editing-algoritme lukt het dus niet om de nauwkeurigheid in dat opzicht te verbeteren, maar het zorgt wel voor een reductie en daarmee voor een snellere uitvoering van het algoritme. Bij de MNIST-set

was het percentage fouten 8% tot 28% hoger dan het percentage ruis, hierin was het editing-algoritme dus niet effectief. Een kleinere K gaf hierbij de beste resultaten. Eerste orde editing gaf in alle gevallen een gelijke of lagere nauwkeurigheid dan de tweede orde editing. De orde van testen heeft bij sommige datasets niet zo veel invloed, dit kan verklaard worden doordat het aantal buren van een punt vaak groter is dan k, waardoor tweede orde testing alleen tests doet met de directe buren, en dus vaak hetzelfde doet als eerste orde testing.

De reductie bij de simpele XOR-test lag tussen de 50% en 76%, waarbij met name de tweede orde editing niet veel reductie behaalde als er enige ruis aanwezig was. Dat eerste orde editing meer reductie behaalt komt doordat tweede orde editing test of het punt verwijderd zou worden met eerste orde editing, waarna het punt niet altijd verwijderd wordt als eerste orde editing dat zou doen, door de extra conditie van tweede orde editing. De reductie van de eerste-orde editing was voor alle hoeveelheden ruis vrij hoog, tussen de 70% en 76%. Ook bij de reductie van de MNIST-set en de diabetes-set was de eerste-orde editing ruim 10% effectiever dan de tweede-orde editing. De reductie van MNIST lag vrij hoog, tussen de 78% en 90%, maar bij de diabetes-set kon weinig reductie bereikt worden, tussen de 37% en 50%. In sommige gevallen levert de reductie dus weinig ruimtebesparing op.

Omdat de nauwkeurigheid van eerste- en tweede-orde editing niet veel uiteen lopen, maar eerste-orde wel consistent meer reductie behaalt, lijkt dit dus onder verschillende omstandigheden het betere testcriterium te zijn.

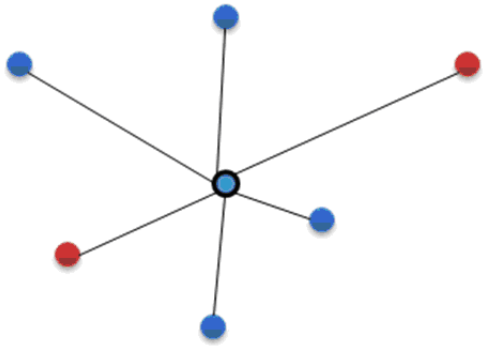
De performance van het algoritme is voor grote testsets slecht, doordat de implementatie een complexiteit van $\theta(d \cdot N^3)$ heeft. Dit zou verbeterd kunnen worden door een ander algoritme te gebruiken om de buren van een punt te bepalen. Ook de methode voor de editing/condensing zou verbeterd kunnen worden, er is in sommige gevallen slechts weinig reductie waardoor de benodigde CPU-tijd voor het testen ook toeneemt. De nauwkeurigheid op testsets verschilt, voor MNIST is deze redelijk laag en voor grotere sets zou de nauwkeurigheid dus verbeterd mogen worden. Voor de XOR-set en de diabetes-set is de nauwkeurigheid goed.

5 Aanpassingen

5.1 Methodes

5.1.1 Dynamic editing

Omdat eerste en tweede orde editing weinig reductie van de dataset opleverden, hebben we een nieuwe methode bedacht om meer punten te verwijderen: dynamic editing. Net zoals bij de andere soorten editing bepalen de buren van een punt of het punt mag blijven of niet. We gebruiken een “threshold” als eis om het punt te verwijderen. Als het aantal buren van een punt met dezelfde klasse, gedeeld door het totaal aantal buren, minder is dan de threshold, dan wordt dat punt verwijderd. Dus hoe hoger de threshold, hoe meer punten er verwijderd worden, dus hoe meer reductie. Voor de waarde van de threshold moet dus een getal tussen 0 en 1 gekozen worden. Hierbij worden er géén punten verwijderd bij een waarde van 0, en worden alle punten grenzend aan minstens één punt van een andere klasse verwijderd bij een waarde van 1. In Figuur 5 wordt het punt verwijderd als de threshold hoger is dan 2/3, omdat dit deel van zijn buren van dezelfde klasse is.



Figuur 5: Een punt en zijn buren

Dynamic editing is vooral gericht op een sterkere reductie dan eerste en tweede orde editing: het haalt punten sneller weg dan eerste orde editing. De reductie van eerste orde editing was erg laag (onder de 40%) zodra er meer ruis werd toegevoegd, met als gevolg dat ook condensing niet effectief was. Eerste orde editing bevat geen parameters om de editing krachtiger te maken, waardoor een nieuwe methode nodig was. De daadwerkelijke dicision boundary wordt ook vervormd door sterkere editing.

5.1.2 Adaptive nearest neighbour

Adaptive nearest neighbour is een variant op k-nearest neighbour. In plaats van de k dichtstbijzijnde punten gebruikt adaptive nearest neighbor alle buren van een nieuw punt om de klasse van dat nieuwe punt te bepalen. De klasse van het nieuwe punt wordt dus bepaald door de klasse die meerderheid van de buren van het nieuwe punt heeft. Op deze manier wordt gebruik gemaakt van de eigenschappen van de graaf die is opgebouwd uit de punten van de trainset. Dit ligt voor de hand, omdat er sowieso al een variant op kNN wordt toegepast door alleen de buren te beschouwen, in plaats van alle punten. Het doorlopen van alle neighbours kost ook veel minder performance dan het doorlopen van k nearest neighbours. **hoezo**? Voor het stemmen voor elk punt is wel meer informatie beschikbaar, dus dit zou een verbetering van de accuracy kunnen opleveren.

Het kNN-principe is bedoeld om te voorkomen dat er te veel punten invloed uit oefenen terwijl ze niet dicht bij een punt liggen, maar dit wordt al voorkomen door een proximity graph. De selectie van punten die voor het classificeren wordt gebruikt, wordt dan dus niet meer door k bepaald, maar puur door de criteria van de proximity graph.

5.1.3 Weighted Classification

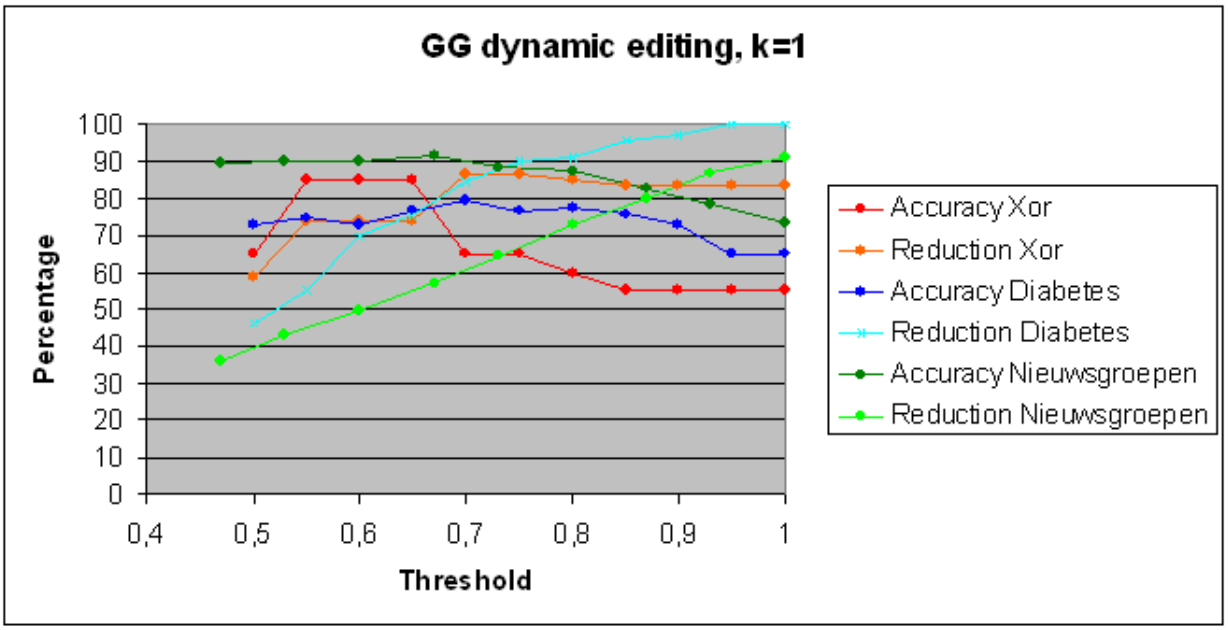
Weighted classification is bedoeld als een aanvulling op adaptive nearest neighbour, maar kan ook in andere combinaties gebruikt worden. Bij weighted classification telt de klasse van elke buur, die meedoet bij het stemmen, meer mee als hij dichter bij het punt ligt en minder als hij verder weg van het punt ligt. Als factoren hebben we 1/d en 1/(d+1) gebruikt. Hierbij is d de euclidische afstand van het punt tot de buur.

5.2 Experimenten

5.2.1 Dynamic editing

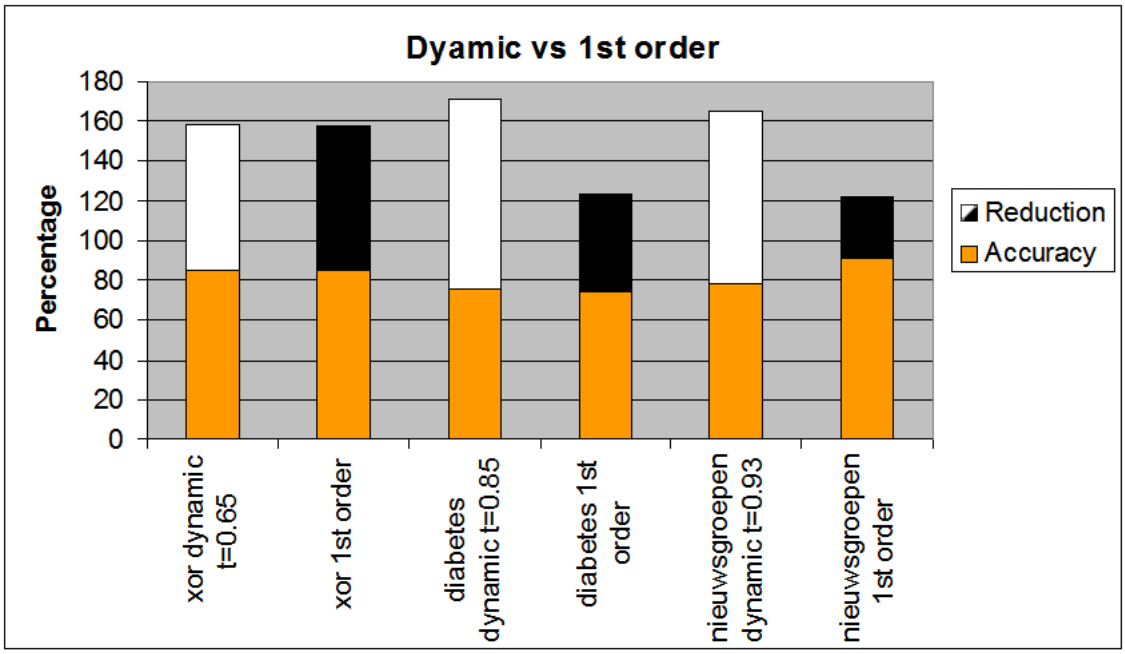
We hebben dynamic editing getest op verschillende datasets door 80% van de trainset te gebruiken om de graaf mee te construeren, en vervolgens de overige 20% van de trainset te gebruiken als

testset. Hieronder staat de bijbehorende grafiek. De tabellen met de meetgegevens zijn te vinden in de bijlage.



Om te bepalen hoe goed dynamic editing is hebben we voor elke dataset de threshold met de 'beste score' genomen uit bovenstaande grafiek. De beste score hebben we bepaald door voor iedere threshold de som van de accuracy en reduction te bepalen, en vervolgens de hoogste te nemen. Deze score hebben we vergeleken met eerste orde editing. Hieronder staat eerst de tabel met de resultaten van eerste orde editing en vervolgens de grafiek waarin dynamic en eerste orde editing worden vergeleken.

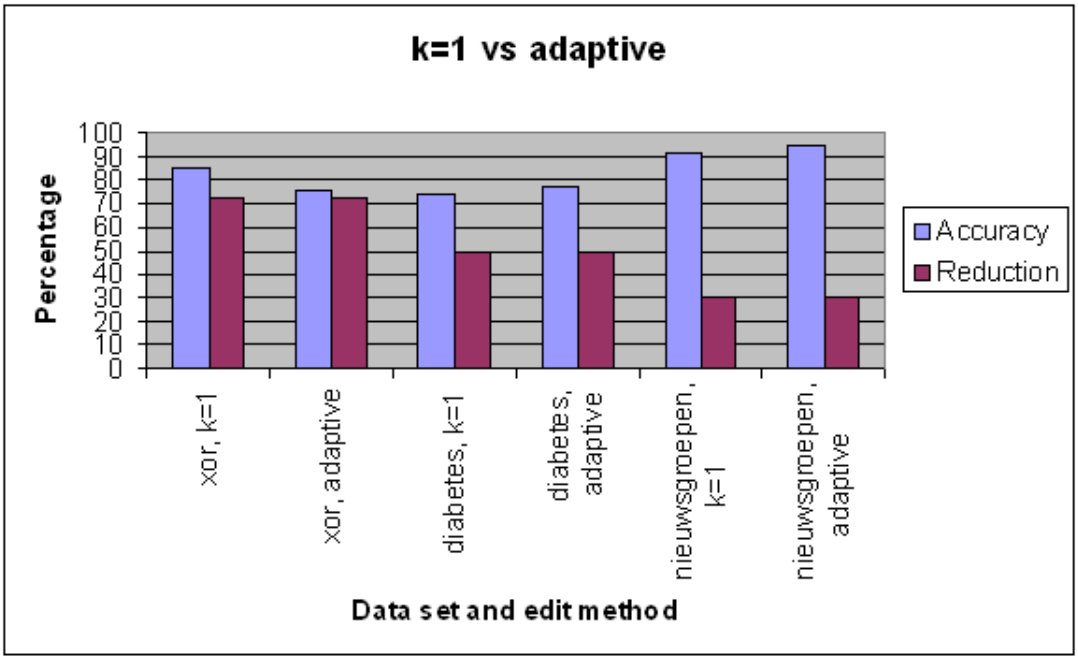
Dataset	Accuracy	Reduction
Xor	85	72,5
Diabetes	73,79	49,63
Nieuwsgroepen	91,5	31



5.2.2 Adaptive nearest neighbour

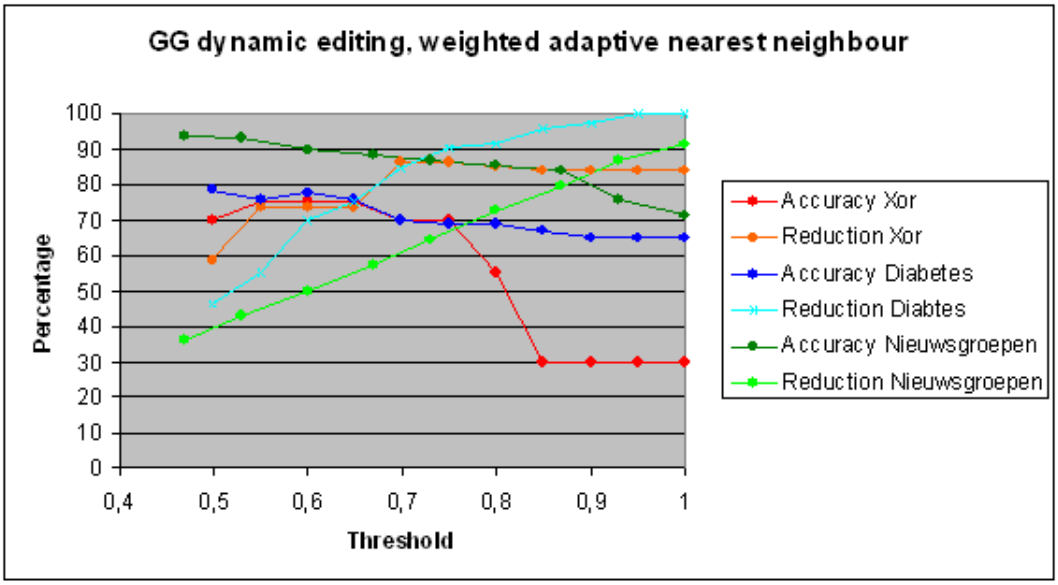
Adaptive nearest neighbour is getest door deze voor verschillende datasets te vergelijken met kNN. Hierbij is voor k=1 gekozen in de vergelijking, omdat deze gemiddeld het beste presteerde over de verschillende datasets. Hieronder zijn de resultaten weergegeven.

Dataset	Nearest neighbour	Accuracy	Reduction
Xor	K=1	85	72,5
Xor	Adaptive	75	72,5
Diabetes	K=1	73,79	49,63
Diabetes	Adaptive	77,67	49,63
Nieuwsgroepen	K=1	91,5	31
Nieuwsgroepen	Adaptive	94,5	31



5.2.3 Weighting

Voor het testen van Weighting hebben we dynamic editing en adaptive nearest neighbour gebruikt, omdat dit gemiddeld de beste resultaten leverde over de XOR-, diabetes- en nieuwsgroepen-datasets. Hieronder zijn de resultaten voor verschillende threshold-waarden weergegeven. Bijbehorende tabellen met de meetgegevens zijn te vinden in de bijlage.

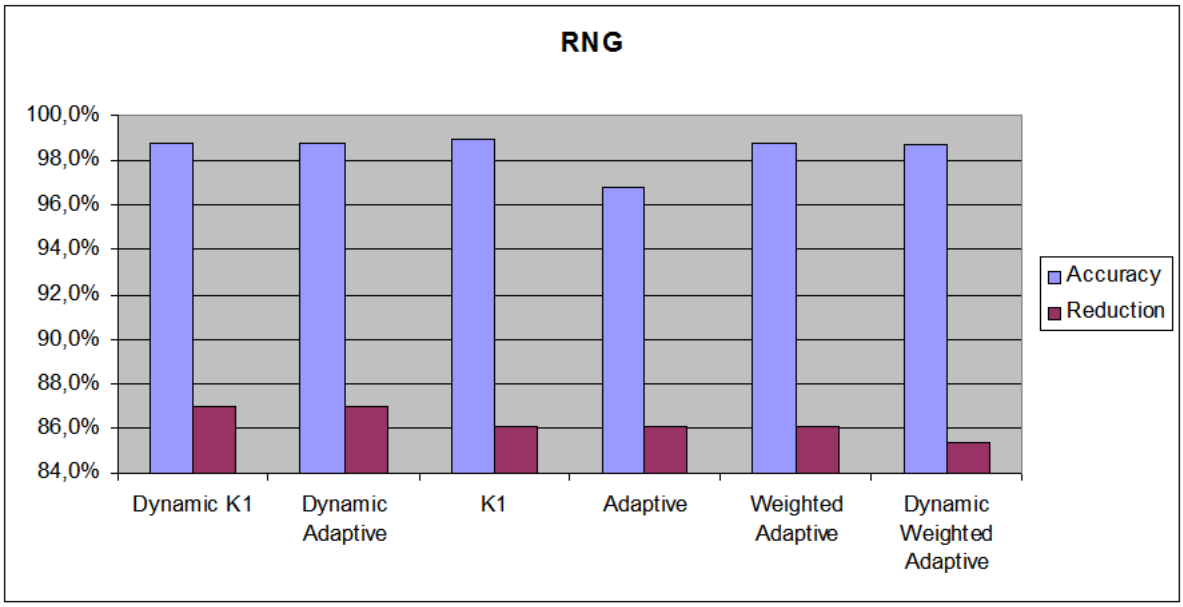


5.3 Karakterherkenning

In de voorgaande experimenten zijn vooral de XOR-, diabetes-, en nieuwsgroepen-datasets gebruikt, omdat het lastig is om voor grotere datasets veel herhalingen uit te voeren. Omdat resultaten kunnen variren voor verschillende soorten gegevens, hebben we voor één grotere dataset, die van karakterherkenning, enkele verschillende methodes toegepast. Omdat deze dataset erg groot was hebben we RNG gebruikt. Het duurt namelijk minder lang om een RNG te maken dan een GG. Dit komt doordat de eis voor het hebben van een edge strenger is, waardoor er eerder een punt wordt gevonden waardoor de eis niet geldt, waardoor dus eerder gestopt wordt met zoeken.

Hieronder staan de gegevens in een tabel en daaronder de bijbehorende grafiek. Standaard (als er niks staat) wordt eerste orde editing gebruikt en geen weighting toegepast bij het classificeren.

Instellingen	Accuracy	Reduction
Dynamic K=1	98.8	87.0
Dynamic Adaptive	98.8	87.0
K=1	98.9	86.1
Adaptive	96.8	86.1
Weighted Adaptive (1/(d+1))	98.7	86.1
Weighted Adaptive (1/d)	98.8	86.1
Dynamic Weighted	98.7	85.4



5.4 Conclusies

5.4.1 Dynamic editing

Dynamic editing zorgde voor een verbetering in prestaties van de diabetes- en nieuwsgroepen- en karakter-dataset. Bij de XOR-dataset waren de verschillen minimaal. Mogelijk was deze dataset zo klein dat de desicion boundary te veel werd aangepast, waardoor de accuracy bij een hoge threshold onder de 65% kwam. Bij de andere datasets bleef de accuracy tot een threshold van ongeveer 0,8 op een acceptabel niveau: die lag rond de accuracy die ook behaald werd met eerste orde editing. Voor thresholds tussen de 0,6 en 0,8 lag de accuracy in veel gevallen zelfs hoger dan

met eerste orde editing. Dit is te verklaren doordat er meer ruis wordt verwijderd, dan dat er punten bij de decision boundary worden verwijderd. De reductie lag voor hoge thresholds altijd boven die van eerste orde editing. Dit is logisch omdat dynamic editing simpelweg een strengere eis is. Bovendien zorgt het verwijderen van ruis er ook voor dat er meer punten met condensing verwijderd kunnen worden.

5.4.2 Adaptive nearest neighbour

In alle geteste datasets waren verschillen tussen kNN en adaptive NN klein, namelijk binnen enkele procenten. Bij de XOR-set en bij de karakter-set met eerste orde editing was kNN accurater, bij de andere tests was adaptive NN accurater. De verschillen tussen kNN en adaptive NN waren bij de karakter-set vrij klein: voor zowel accuracy als reductie waren deze minder dan één procent. Het verschilt dus per dataset welke methode een beter resultaat geeft. Het kNN-principe heeft als voordeel dat punten die ver uit elkaar liggen geen invloed op elkaar hebben. Punten zullen sneller binnen een boundary liggen als ze dichtbij elkaar liggen. Adaptive NN maakt daarentegen beter gebruik van de eigenschappen van proximity graphs: punten die ver weg liggen, worden alleen verbonden als er geen punten tussenin liggen. Een proximity graph zegt daardoor meer over welke klassen er zijn in de verschillende richtingen van een punt, in plaats van binnen een bepaalde straal van een punt. Beide eigenschappen blijken dus voordelig te kunnen zijn.

5.4.3 Weighting

Ook voor weighted adaptive NN geldt dat de resultaten verschillen per dataset. Bij de XOR-set is kNN enkele procenten (rond 10%) accurater, bij de diabetes- en nieuwsgroepen-datasets is weighted adaptive NN tot enkele procenten (rond 4%) accurater. Het is lastig om precies te beredeneren waarom dit beter of slechter is in bepaalde sets. Voor het aantal punten dat beschouwd wordt bij het stemmen valt weighted adaptive NN tussen gewone adaptive NN en kNN in: er worden wel meer punten bekeken dan in kNN met k=1, maar de buitenste punten van adaptive NN worden minder zwaar meegeteld. De voor en nadelen van adaptive NN en kNN zullen dus bij weighted ook een rol spelen.

5.5 Bijlagen

Tabellen bij dynamic editing (5.2.1):
Xor

Threshold	Accuracy	Reduction
0,5	65	58,75
0,55	85	73,75
0,6	85	73,75
0,65	85	73,75
0,7	65	86,25
0,75	65	86,25
0,8	60	85
0,85	55	83,75
0,9	55	83,75
0,95	55	83,75
1	55	83,75

Diabetes

Threshold	Accuracy	Reduction
0,5	72,82	46,21
0,55	74,76	55,01
0,6	72,82	69,68
0,65	76,7	75,06
0,7	79,61	84,35
0,75	76,7	90,22
0,8	77,67	91,2
0,85	75,73	95,84
0,9	72,82	97,31
0,95	65,05	100
1	65,05	100

Nieuwsgroepen

Threshold	Accuracy	Reduction
0,47	89,5	36
0,53	90	42,88
0,6	90	49,62
0,67	91,5	57,12
0,73	88,5	64,5
0,8	87,5	72,88
0,87	82,5	79,75
0,93	78,5	86,88
1	73,5	91

Tabellen bij GG dynamic editing, weighted adaptive nearest neighbour (5.2.3):

Xor

Threshold	Accuracy	Reduction
0,5	70	58,75
0,55	75	73,75
0,6	75	73,75
0,65	75	73,75
0,7	70	86,25
0,75	70	86,25
0,8	55	85
0,85	30	83,75
0,9	30	83,75
0,95	30	83,75
1	30	83,75

Diabetes

Threshold	Accuracy	Reduction
0,5	78,64	46,21
0,55	75,73	55,01
0,6	77,67	69,68
0,65	75,73	75,06
0,7	69,9	84,35
0,75	68,93	90,22
0,8	68,93	91,2
0,85	66,99	95,84
0,9	65,05	97,31
0,95	65,05	100
1	65,05	100

Nieuwsgroepen

Threshold	Accuracy	Reduction
0,33	93	23,62
0,4	94	29,25
0,47	93,5	36
0,53	93	42,88
0,6	90	49,62
0,67	88,5	57,12
0,73	87	64,5
0,8	85,5	72,88
0,87	84	79,75
0,93	75,5	86,88
1	71	91