Expressieherkenning met PCA en SVM

Ruben Muijrers

Inhoudsopgave

HOOFDSTUK 1

Introductie

1. Inleiding & Vraagstelling

1.1. Inleiding. In dagelijkse gesprekken wordt informatie uitgewisseld. Hoe deze informatie moet worden geïnterpreteerd wordt deels bepaald door de gezichtsuitdrukking erbij. Iemand die "Ik zie het niet meer zitten!" roept terwijl hij sip kijkt zal er wel overheen komen, maar iemand die dat roept met de tranen in zijn ogen is er erger aan toe. Mensen kunnen deze gezichtsuitdrukkingen over het algemeen zeer goed interpreteren binnen hun eigen omgeving, hoe wel er wel verschillen zijn qua interpretatie tussen culturen [**20**]. Het doel van dit onderzoek is het evalueren van technieken voor herkennen van emoties (gezichtsuitdrukkingen) met een computer. Er is binnen dit gebied al redelijk wat onderzoek gedaan en er worden herkenningsscores van tenminste 93% gehaald op de Cohn-Kanade expression dataset [**1**]. Wij hebben niet de ambitie om een recordbrekend algoritme te construeren, maar om wat te leren over de technieken en problemen binnen dit vakgebied.

1.2. Onderzoeksvraag. Het doel van dit onderzoek is het antwoord krijgen op de volgende vraag:

Is een combinatie van Principale Componenten Analyse en Support Vector Machines geschikt om expressies in het gezicht te herkennen?

Met *geschik* wordt hier bedoelt dat er een significant hoger correct herkenningspercentage gehaald wordt dan de gokkans. Ik gebruik hier 7 klassen dus de gokkans is ongeveer 14%. Hoewel 15% of 16% al boven de gokkans is zal hier vermoedelijk geen praktische toepassing voor zijn. Mijn doel is om ruim boven de 50% te komen.

2. Criteria

Om expressieherkenning te doen zullen we eerst moeten bepalen wat goede herkenning is. Dat blijkt niet zo triviaal als het lijkt.

Wat zijn de criteria voor een goede herkenning?

Gezichtsexpressies hebben betekenis en spelen een belangrijke rol in de communicatie tussen mensen. Vanuit de psychologie zijn gezichtsexpressies ook al vele jaren onderwerp van studie. Alleen is de vraag, vanuit psychologisch perspectief, iets ingewikkelder. Psychologie beperkt zich niet tot het simpelweg herkennen van expressies maar beschouwt ook de mogelijke invloeden, gevolgen en oorzaken van expressies.

De vraag of expressies (en de bijbehorende emoties) universeel zijn is een ingewikkelde. Veel onderzoek is gedaan met verschillende resultaten. Het artikel van James A. Russel [**20**] geeft een grondige review van voorgaand onderzoek en plaatst het in context door verschillende uitgangspunten aan te stippen in de onderzoeken. Op basis van dit artikel zijn onze aannames gedaan betreffende de maximale haalbare nauwkeurigheid en de juiste klassen in dit onderzoek.

Nauwkeurigheid. Het herkennen van expressies door mensen gebeurt niet foutloos. Fouten kunnen optreden door verkeerde interpretatie of een afwijkende expressie. Aan een afwijkende expressie is weinig te doen, deze fout zullen we ook terugzien in onze resultaten. Theoretisch gezien zou verkeerde interpretatie door een programma niet hoeven gebeuren, maar we vermoeden dat het niet realistisch is beter te presteren dan een gemiddeld mens. We nemen daarom de samengestelde resultaten van de review van Russel als uitgangspunt voor de nauwkeurigheid waarmee onze herkenner moet gaan werken. Onderaan de tabellen staat een mediaan, we nemen de gemiddelde mediaan als uitgangspunt.

Recognition Scores Fom Eight Studies With Literate Subjects

Culture	*n*	Facial expression "Happy"	"Surprise"	"Sadness"	"Fear"	"Disgust"	"Anger"
Western cultures							
American[a]	99	97	91	73	88	82	69
Brazilian[a]	40	97	82	82	77	86	82
American[b]	89	96.8	90.5	74.0	76.0	83.2	89.2
English[b]	62	96.2	81.0	74.5	67.0	84.5	81.5
German[b]	158	98.2	85.5	67.2	84.0	73.0	83.2
Swedish[b]	41	96.5	81.0	71.5	88.8	88.0	82.2
French[b]	67	94.5	84.2	70.5	83.5	78.5	91.5
Swiss[b]	36	97.0	85.5	70.0	67.5	78.2	91.8
Greek[b]	50	93.5	80.2	54.5	67.8	87.5	80.0
Chilean[c]	119	90.2	88.3	90.9	78	85	76
Argentine[c]	168	94.0	93	87.6	68	79.3	71.6
Estonian[d]	70	88.0	82.5	84.7	60.2	89.0	77.7
American[e]	53	96.7	85.9	72.6	69.8	71.7	64.6
American[g]	40	100	92.5	87.5	67.5	92.5	90.0
Estonian[h]	85	90	94	86	91	71	67
German[h]	67	93	87	83	86	61	71
Greek[h]	61	93	91	80	74	77	77
Italian[h]	40	97	92	81	82	89	72
Scottish[h]	42	98	88	86	86	79	84
American[h]	30	95	92	92	84	86	81
Median		96.4	87.5	80.5	77.5	82.6	81.2
M		95.1	87.4	86.5	77.3	81.1	79.1
Non-Western cultures							
Japanese[a]	29	87	87	74	71	82	63
Japanese[b]	60	93.8	79.2	66.8	58.2	55.8	56.8
African[b]	29	68.0	49.0	32.2	49.0	55.0	50.8
Kirghizian[d]	80	89.2	71.3	89.2	51.3	86.0	47.2
Malaysian[e]	30	95.8	69.8	66.4	45.6	59.2	49.8
Ethiopian[f]	100	86.8	50.5	52.0	58.8	54.8	37.3
Malaysian[g]	31	100	95	100	66.5	97.5	86
Chinese[h]	29	92	91	91	84	65	73
Japanese[h]	98	90	94	87	65	60	67
Sumatran[h]	36	69	78	91	70	70	70
Turkish[h]	64	87	90	76	76	74	79
Median		89.2	79.2	76.0	65.0	65.0	63.0
M		87.1	77.7	75.1	63.2	69.0	61.8

Note. Izard's (1971) term for "sadness" was *distress*, but it was defined as synonymous with *sadness*.
[a] Ekman, Sorenson, and Friesen (1969). [b] Izard (1971). [c] Ekman (1972). [d] Niit & Valsiner (1977). [e] Boucher and Carlson (1980); figures given are unweighted average across two stimulus sets. [f] Ducci, Arcuri, W/Georgis, and Sineshaw (1982). [g] McAndrew (1986). [h] Ekman et al. (1987).

[0]Delen van deze sectie zijn gebaseerd op werk van Albert Gerritsen.

Tijdsduur van herkenning. Dit criterium is vooral belangrijk voor realtime systemen. De algoritmen die we zullen behandelen zijn gemaakt om (na training) snel te kunnen classificeren. Het is dus geen onrealistische wens, het herkennen realtime te doen. Mocht realtime niet haalbaar zijn dan geldt: sneller is beter.

Toepasbaarheid. De toepasbaarheid van dit soort algoritmen is afhankelijk van veel factoren. We willen ons concentreren op het herkennen van expressies en de randvoorwaarden buiten beschouwing laten (aangezien het probleem al ingewikkeld genoeg is). Mogelijke randvoorwaarden die een rol spelen in een reallife situatie:

- Foto's genomen van verschillende standpunten
- Foto's die op een verschillende manier belicht zijn
- Foto's waar niet het hele gezicht op afgebeeld staat
- Foto's van verschillende kwaliteit

In onderzoek naar het menselijk brein zijn er diverse aanwijzingen dat het omgaan met deze randvoorwaarden los gebeurt van het herkennen van de eigenlijke expressies ([**10**], [**9**]). Ook zijn er diverse methoden in de Kunstmatige Intelligentie die het mogelijk maken om te gaan met deze randvoorwaarden (bijvoorbeeld [**3**]). Het is dus gerechtvaardigd om het preprocessen van de foto's als los deel te zien van het algoritme dat de expressies koppelt aan de foto's. In dit onderzoek wordt preprocessing gezien als een apart blok naast dimensionaliteit reductie en het uiteindelijke classificeren. Conclusies van dit onderzoek zullen dus niet zozeer gaan over het preprocessing gedeelte maar meer over of PCA en SVMs goed werken binnen deze context.

3. Klassen

De vraag die we in dit hoofdstuk willen beantwoorden is:

Wat zijn de klassen (expressies) die we willen onderscheiden?

Binnen het AI vakgebied wordt er in onderzoeksartikelen (over expressie herkenning) vaak niet vermeld waarom ze voor een bepaalde klassenverdeling gekozen hebben (zoals onder andere [**21**] en [**5**]). Opvallend is wel dat vrijwel in alle gevallen (een subset van) de volgende (vrij intuïtieve) expressies gebruikt wordt: blij (happiness), verrast (surprise), angst (fear), woede (anger) en verdriet (sadness).

Volgens [**20**] en verscheidene andere onderzoekers is er een set van universele expressies. De volledige set bestaat volgens Russell plus of min 2 uit de volgende expressies: blij (happiness), verrast (surprise), angst (fear), woede (anger), minachting (contempt), afkeer (disgust) en verdriet (sadness). Dit komt vrij goed overeen met de in onderzoeksartikelen gekozen set uit de vorige paragraaf.

Frequency of Free-Response Labels for Expressions Reported to Be of Anger and Contempt

"Anger" expression		"Contempt" expression	
Label	Freq	Label	Freq
Frustration	49	*Disgust*	16
Anger	41	*Bored*	10
Mad	18	*Disappointment*	9
Constipated	4	*Puzzled*	6
Upset	3	*Confusion*	6
Confusion	2	*Frustration*	5
Making a decision	2	*Indifference*	5
Perturbed	2	*Smug*	5
Perplexed	2	*Contempt*	3
Irritable	2	*Perplexed*	3
Doubt	2	*Pissed off*	3
Pissed off	2	*Cynical*	3
Scorn	2	*Disgruntled*	3
Idiosyncratic	27	*Sarcastic*	3
		Anger	3
		Stupid	2
		Depression	2
		Indecisive	2
		Impatient	2
		Dissatisfaction	2
		Pain	2
		Troubled	2
		Arrogant	2
		Disbelief	2
		Perturbed	2
		Amused	2
		Idiosyncratic	46
		Other	9

Note. $n = 160$ for each type of expression.

Uit hetzelfde artikel van Russell bleek helaas dat niet al deze 7 expressies even goed te herkennen zijn door mensen. Zo werd 'verdriet' vaak als 'woede' opgevat en lijkt 'minachting' voor veel mensen op 'afkeer'. 'minachting' werd door Engels sprekenden het slechtst herkend van de bovenstaande zeven. In een test waar een vrij gekozen label aan de expressie 'minachting' gekoppeld mocht worden kozen slechts 3 van de 160 onderzochten voor het label 'minachting'. De meningen waren nogal verdeeld over welk label nou bij de expressie hoorde. 'afkeer' werd het meest genoemd met 16 van de 160. De andere expressies worden over het algemeen door westerlingen vrij accuraat herkend met gemiddelde herkenningsscores van rond de 80 procent.

Wij willen voor ons onderzoek de universele set gebruiken, maar omdat het herkennen van expressies voor computers nog steeds moeilijker is dan voor mensen hebben we besloten om 'minachting' uit de set te halen en 'neutraal' toe te voegen. 'Minachting' is slecht herkenbaar en omdat niet iedereen altijd boos is of lacht, kan 'neutraal' als een veelvoorkomende expressie gezien worden. De uiteindelijke klassen met label zoals gebruikt in de code zijn te zien in tabel 1.

Klasse	Anger	Disgust	Fear	Happiness	Neutral	Sadness	Surprise
Label	ANG	DIS	FEA	HAP	NEU	SAD	SUR

TABEL 1. De gebruikte klassen en hun labels

HOOFDSTUK 2

Achtergrond

1. Eigenafbeeldingen & PCA

1.1. Eigenafbeeldingen. Het basis idee is vrij simpel, een afbeelding van X bij Y pixels wordt omgezet naar een vector van $X \times Y$ lang. Die vector wordt omgeschreven naar een combinatie van andere vectoren, deze noemen we hier de basisvectoren. Op deze manier hoeft men alleen de basisvectoren, die voor alle afbeeldingen hetzelfde zijn, en de veel kleinere vector van de combinatie op te slaan.

Voorbeeld. We nemen 3 vectoren

$$\begin{aligned} v_1 &= [1, 0, 1, 3] \\ v_2 &= [1, 0, 1, -4] \\ v_3 &= [0, 0, 0, 1] \end{aligned}$$

En 2 basisvectoren:

$$\begin{aligned} e_1 &= [0, 0, 0, 1] \\ e_2 &= [1, 0, 1, 1] \end{aligned}$$

Nu kunnen we de 3 vectoren beschrijven in de eigenvectoren:

$$\begin{aligned} v_1 &= 1e_1 + 2e_2 \\ v_2 &= 1e_1 + -5e_2 \\ v_3 &= 0e_1 + 1e_2 \end{aligned}$$

Om $v1$, $v2$ en $v3$ te representeren worden er nu vectoren van lengte 2 gebruikt in plaats van 4, namelijk de factoren van de basisvectoren. De dimensionaliteit is nu afgenomen zonder dat er informatieverlies is opgetreden. In de praktijk treedt er vaak wel informatieverlies op omdat niet alle basisvectoren gebruikt worden. Door het toestaan van informatieverlies kan het aantal benodigde basisvectoren flink gereduceerd worden en daarmee de dimensionaliteit van de resulterende vector. Bijvoorbeeld, bij 2000 afbeeldingen van 100 bij 100 pixels en 50 basisvectoren wordt de dimensionaliteit terug gebracht van $2000 \times 100 \times 100$ naar $2000 \times 50 + 50 \times 100$.

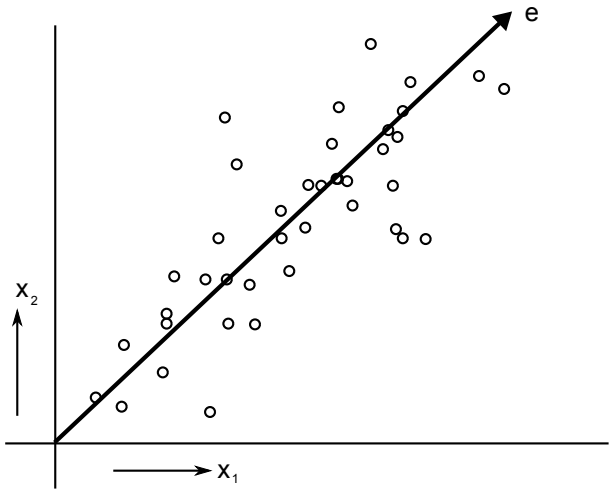
Voor de basisvectoren worden vaak de eigenvectoren van de dataset gekozen. Hiermee kan eenvoudig een klein aantal basisvectoren gekozen worden met zo min mogelijk informatie verlies. Eigenvectoren berekenen is computationeel vrij zwaar. Huidige algoritmen hebben een complexiteit van $O(K^3)$ met K gelijk aan het aantal attributen [**6**]. Als de eigenvectoren eenmaal bekend zijn kan een nieuwe afbeelding "snel" beschreven worden door voor iedere eigenvector het inproduct te berekenen. Dit kan in tijd $O(M \times K)$ met M het aantal eigenvectoren en K het aantal attributen (de lengte van de eigenvector).

1.1.1. *Gezichtsherkenning.* Nadat de set eigenvectoren gegenereerd is kunnen nieuwe afbeeldingen (bij benadering) beschreven worden op basis van deze eigenvectoren. Door de combinatievector te vergelijken met de vectoren uit een database kan men bepalen in welke mate gezichten overeen komen. Er zijn vrij hoge resultaten behaald met deze manier van gezichtsherkenning [**23**], hoewel verschillen in belichting en de hoek van de camera de resultaten erg naar beneden halen.

1.1.2. *Expressieherkenning.* De afbeeldingen uit de trainingsset worden eerst uitgedrukt in eigenvectoren. Met behulp van een classificatiealgoritme wordt gekeken welke eigenvectoren bepalend zijn voor klassen. Nieuwe afbeeldingen worden dan weer op basis van het getrainde classificatiealgoritme ingedeeld in een klasse.

1.2. PCA. Om de eigenvectoren te bepalen gebruiken we PCA zoals beschreven in [**17**]. PCA probeert een dataset met K dimensies af te beelden op M dimensies. Deze M dimensies worden zo gekozen dat ze zo min mogelijk data verlies veroorzaken.

Voorbeeld. In figuur 1 zijn een aantal objecten met 2 attributen geplot. De 2 attributen (de dimensies) staan langs de assen (x_1 en x_2). Om het aantal dimensies te reduceren kijken we naar de spreiding van de data. Een voor de hand liggende oplossing is het weggooien van de dimensie met de minste spreiding. In dit voorbeeld hebben x_1 en x_2 ongeveer evenveel spreiding en zou deze strategie onnodig veel data verlies veroorzaken. Beter is het om als het ware de dimensies x_1 en x_2 te vergeten en een totaal nieuwe dimensie creëren. In figuur 1 geeft vector e deze dimensie



FIGUUR 1. Een 2 dimensionale dataset e geeft de richting aan met de meeste spreiding

aan. In deze richting ligt de meeste spreiding van de data. Alle datapunten zijn nu te beschrijven als een combinatie van een vector parallel aan e en een vector loodrecht op e. Door de vectoren loodrecht op e weg te laten en alleen de parallele vectoren te gebruiken houden we hier 1 dimensie over.

Bij meerdere initiële dimensies wordt voor de eerste nieuwe dimensie de vector met de meeste spreiding berekend. Voor de tweede nieuwe dimensie wordt een vector die loodrecht op de eerste staat en de richting van de meeste overgebleven spreiding berekend. De derde nieuwe vector staat dan weer loodrecht op de tweede enzovoort, totdat er geen informatie verlies meer is. Dit levert maximaal evenveel vectoren op als dat er initiële dimensies waren. Deze nieuwe vectoren heten de al eerder genoemde eigenvectoren en de hoeveelheid spreiding die een eigenvector beslaat wordt uitgedrukt in de eigenwaarde van die eigenvector. Door de eigenvectoren met de laagste eigenwaarde weg te laten kan er makkelijk in dimensies gereduceerd worden met minimaal informatieverlies. De overgebleven eigenvectoren noemt men de principale componenten. PCA maakt geen gebruik van de klassenlabels en kan dus unsupervised draaien. Dit maakt het een zeer makkelijk toe te passen algoritme voor dimensionaliteit reductie.

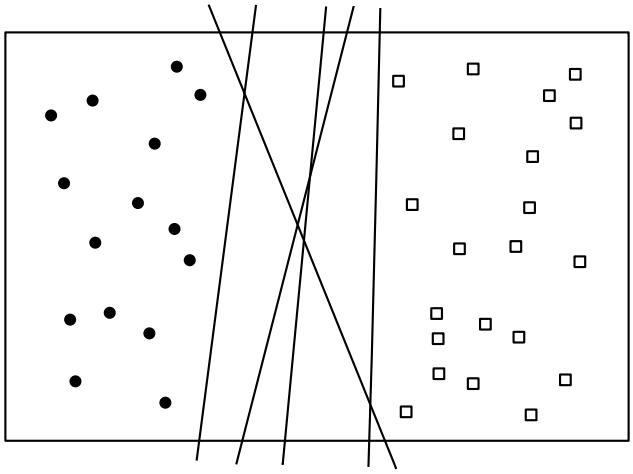
De eigenvectoren worden berekend aan de hand van de covariantiematrix $Q = [q_{ij}]$ met $i, j = 1, 2, 3, ...K$, deze wordt als volgt berekend:

$$q_{ij} = E((x_i - E(X_i))(x_j - E(X_j))),$$

met x_i de objecten vector voor attribuut i en $E(X_i)$ de verwachtingswaarde voor attribuut (of dimensie) i. Voor de verwachtingswaarden in deze formule wordt in het algemeen het gemiddelde van genomen. Met een eigenwaarde algoritme zoals bijvoorbeeld het *QR algorithm* [**6**] of de *power method* uitgelegd in [**12**] worden de eigenwaarden bepaald. De M eigenvectoren met de grootste eigenwaarden worden gekozen voor de nieuwe dimensies.

2. Support Vector Machines

Een tegenwoordig veel gebruikte classificatietechniek is de Support Vector Machine zoals beschreven in [**17**]. Het grote voordeel van deze techniek is dat deze weinig last heeft van hoogdimensionale data. Standaard voor SVMs is dat ze een binaire keuze maken. Ze kunnen dus bepalen of een datapunt bij klasse -1 of bij klasse $+1$ hoort. Implementaties van SVMs leveren typisch een reële waarde c tussen -1 en 1 op. Deze waarde kan dan opgevat worden als een maat van zekerheid, de klasse voor een datapunt is dan -1 als $c < 0$ en $+1$ als $c > 0$. De zekerheid van de classificatie is dan $abs(c)$. Bij 3 of meer klassen zijn er evenveel SVMs als klassen nodig. De eerste maakt het onderscheid tussen klasse A en niet klasse A, de tweede maakt het onderscheid tussen klasse B en niet klasse B, enz. Om de uiteindelijke klasse te bepalen kan de klasse met de hoogste zekerheid gekozen worden. Over het algemeen worden er nog betere resultaten gehaald als men de uiteindelijke keuze van de klasse door een boostingalgoritme of een Neuraal Netwerk laat bepalen.



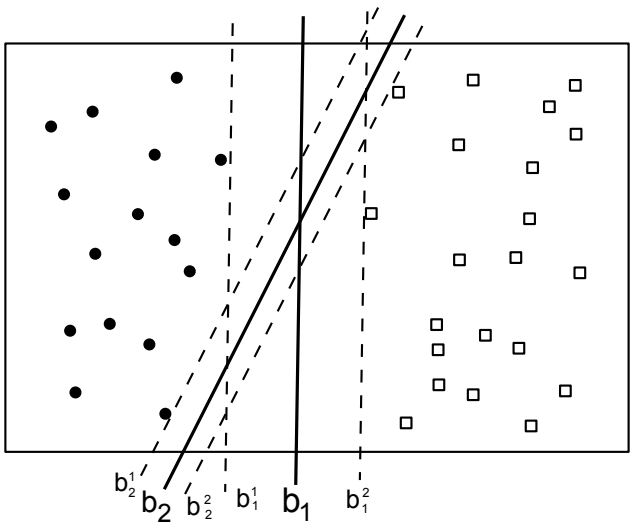
FIGUUR 2. Een lineair separeerbare dataset met mogelijke scheidingslijnen

2.1. Lineair Separeerbare Datasets. Gegeven een lineair separeerbare dataset, zoals in Figuur 2, kunnen we een lijn (in 2d, een vlak in 3d of een hypervlak in het algemeen) vinden zo dat alle witte vierkantjes aan een kant van de lijn liggen en alle zwarte rondjes aan de andere kant. Maar zoals aangegeven in de figuur zijn er oneindig veel mogelijke lijnen die aan die eigenschap voldoen. Hoewel ze voor deze dataset een perfecte score behalen is er geen garantie voor hoe ze het doen op ongeziene data.

In figuur 3 zien we dezelfde dataset met twee lineaire scheidingslijnen. Beiden scheiden de data zonder fouten. Een SVM probeert bij het vinden van een scheidingslijn een zo groot mogelijke afstand tot het dichtstbijzijnde datapunt van iedere klasse te handhaven. Deze afstand noemen we de margin en is aangegeven met een stippellijn. In theorie geldt dat een scheidingslijn met een grotere margin waarschijnlijk beter generaliseert dan een met een kleine margin [**17**]. Intuïtief zou bij een kleine margin een kleine verschuiving al snel effect hebben op de classificatie, de datapunten liggen immers dichter bij de scheidingslijn. Bij een grote margin moet er een grotere "afstand" overbrugt worden voor de classificatie wordt beïnvloed. In dit geval is $\mathbf{b_1}$ dus een betere scheidingslijn dan $\mathbf{b_2}$.

Om de scheidingslijn en de margin te bepalen formaliseren we eerst de hierboven uitgelegde principes. Gegeven een dataset met N elementen die worden beschreven door hun attributenvector $\mathbf{x_i}$ en hun klasse $c_i \in -1, 1$ met $i \in 1 \ldots N$ kan het hypervlak dat de klassen scheidt als volgt geschreven worden:

$$\mathbf{w} \cdot \mathbf{x} + b = 0. \tag{1}$$



FIGUUR 3. Twee scheidingslijnen met hun marges voor deze dataset

$\mathbf{w}$ en b zijn hier parameters van het model; verderop wordt uitgelegd hoe men aan de waarden voor deze parameters komt. Voor ieder datapunt $\mathbf{x}$ dat op de scheidingslijn ligt geldt vergelijking 1. Als we van figuur 3 uitgaan met de zwarte rondjes als klasse -1, de witte vierkantjes als klasse $+1$ en hypervlak $\mathbf{b_1}$ dan geldt:

$$c_i = \begin{cases} +1, & \text{als } \mathbf{w} \cdot \mathbf{x}_i + b > 0, \\ -1, & \text{als } \mathbf{w} \cdot \mathbf{x}_i + b < 0. \end{cases} \tag{2}$$

Door $\mathbf{w}$ en b te schalen kunnen we de hypervlakken b_i^1 en b_i^2 als volgt schrijven:

$$b_i^1 : \mathbf{w} \cdot \mathbf{x}_i + b = -1,$$
$$b_i^2 : \mathbf{w} \cdot \mathbf{x}_i + b = +1.$$

Alles wat nu kleiner is dan -1 zal worden geclassificeerd als zwart rondje en alles wat groter is dan 1 als wit vierkantje (tussen 1 en -1 zitten bij constructie geen punten). De afstand tussen deze twee hypervlakken is de margin d. Door deze vergelijkingen van elkaar af te trekken en wat geometrie toe te passen krijgen we de volgende definitie voor de margin:

$$d = \frac{2}{||\mathbf{w}||}. \tag{3}$$

Om een zo groot mogelijke margin te vinden zoeken we dus een zo klein mogelijke $||\mathbf{w}||$. De volgende condities dwingen goede classificatie af:

$$\mathbf{w} \cdot \mathbf{x} + b \geq 1 \text{ als } c_i = +1, \tag{4}$$
$$\mathbf{w} \cdot \mathbf{x} + b \leq -1 \text{ als } c_i = -1. \tag{5}$$

We kunnen de vergelijkingen (3), (4) en (5) combineren en dit schrijven als een optimalisatie probleem:

$$\text{kies } \mathbf{w} \text{ en } b \text{ zo dat } ||\mathbf{w}|| \text{ zo klein mogelijk is}$$
$$\text{beperkt door } c_i \cdot (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 \geq 0 \text{ met } i \in 1, 2, ..., N.$$

In de bovenstaande vergelijking kan $||\mathbf{w}||$ vervangen worden door $\frac{1}{2}||\mathbf{w}||^2$ zonder dat dit gevolgen heeft voor de oplossing; $\mathbf{w}$ en b blijven hetzelfde. Als we het omschrijven naar de (Primaire) Lagrangiaan zonder constraints krijgen we:

$$L_P = \frac{1}{2}||\mathbf{w}||^2 - \sum_i \lambda_i(c_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1). \tag{6}$$

L_P wordt hier geminimaliseerd naar $\mathbf{w}$ en b en vervolgens gemaximaliseerd naar λ_i. De λ_i parameters heten de Lagrange multiplicatoren. Door van L_P de afgeleide te berekenen en deze gelijk te stellen met 0 kunnen we het minimum berekenen. De afgeleiden van $\mathbf{w}$ en b zijn:

$$\frac{\partial L_P}{\partial \mathbf{w}} = 0 \Longrightarrow \mathbf{w} = \sum_i \lambda_i c_i \mathbf{x}_i, \tag{7}$$

$$\frac{\partial L_P}{\partial b} = 0 \Longrightarrow \sum_i \lambda_i c_i = 0. \tag{8}$$

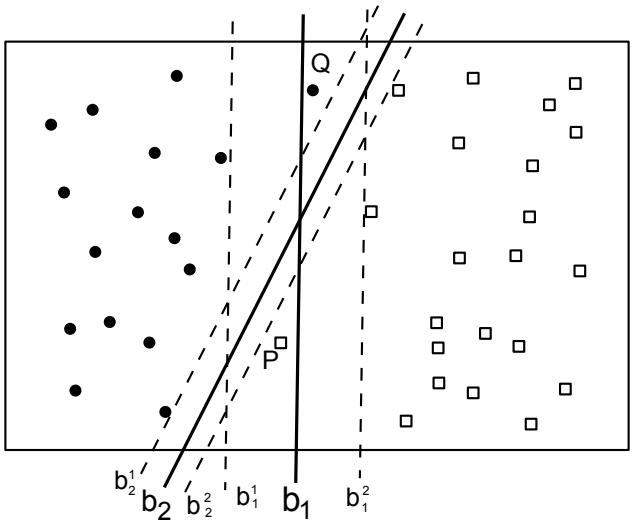
Doordat de constraints op het minimalisatie probleem ongelijkheden zijn is invullen niet zomaar mogelijk. Door ze te transformeren naar gelijkheden kan het wel maar ontstaan de volgende voorwaarden, ook wel Karush-Kuhn-Tucker (KKT) voorwaarden genoemd:

$$\lambda_i \geq 0,$$
$$\lambda_i[c_i(\mathbf{w}\cdot\mathbf{x}_i + b) - 1] = 0.$$

Nu kunnen we 7 en 8 invullen in vergelijking 6 en krijgen we de duale Lagrange vergelijking.

$$L_D = \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i\lambda_j c_i c_j \mathbf{x}_i \cdot \mathbf{x}_j \tag{9}$$

Hier is te zien dat het minimalisatie probleem nu alleen nog afhangt van de Lagrange multiplicatoren en de trainingsdata. De gewenste waarden voor $\mathbf{w}$ en b kunnen terug gerekend worden aan de hand van de vergelijkingen 7 en 8. Het oplossen van de duale Lagrange vergelijking wordt typisch gedaan door middel van *quadratic programming* met bijvoordeeld de *ellipsoid method* [**8**]. Hoe dit precies in zijn werk gaat, zal niet in dit verslag worden uitgelegd.



FIGUUR 4. Een dataset met scheidingslijnen en ruispunten P en Q

2.2. Ruis en Niet-Lineair Separeerbare Datasets. Helaas is in de praktijk data niet altijd (goed) lineair separeerbaar. In figuur 4 is dezelfde dataset als in figuur 3 weergegeven maar er zijn 2 punten P en Q toegevoegd. Scheidingslijn $\mathbf{b_1}$ creëert geen foutloze scheiding meer en $\mathbf{b_2}$ wel. Toch is in dit geval $\mathbf{b_1}$ waarschijnlijk een betere keus dan $\mathbf{b_2}$ omdat P en Q ruis kunnen zijn en dus eigenlijk niks toevoegen aan de data. Om dit probleem op te lossen wordt gebruik gemaakt van een "soft margin". Deze methode staat kleine fouten toe en maakt een afweging tussen de breedte van de margin, het aantal fouten en de grootte van de fouten in de classificatie.

Om fouten toe te staan in de classificatie wordt de voorwaarde voor het minimalisatie probleem uitgebreid met zogenaamde slack variabelen:

$$c_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i \geq 0.$$

Met voor ξ_i een getal dat de afstand bepaalt tot de goede klasse, voor goed geclassificeerde objecten is dit dus 0. Voor objecten die binnen de margins liggen of voor objecten die geheel fout zijn geclassificeerd geldt $\xi_i > 0$. De minimalisatiefunctie wordt ook aangepast.

$$\text{Minimaliseer } ||\mathbf{w}||^2/2 + C(\sum_i \xi_i).$$

Hier is C vrij te kiezen voor de gebruiker, deze bepaalt de strafmaat voor foutieve classificaties. De primaire Lagrange vergelijking voor het minimalisatie probleem komt er als volgt uit te zien.

$$L_P = \frac{1}{2}||\mathbf{w}||^2 + C\sum_i \xi_i - \sum_i \lambda_i(c_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1 + \xi_i) - \sum_i \mu_i \xi_i \tag{10}$$

μ_i is hier gewoon een Lagrange multiplicator maar met een extra conditie, zie vergelijking 13. Door de afgeleide te berekenen en de ongelijkheden te transformeren naar gelijkheden krijgen we de volgende condities:

$$\xi_i \geq 0, \lambda_i \geq 0, \mu_i \geq 0, \tag{11}$$

$$\lambda_i[c_i(\mathbf{w} \cdot \mathbf{x_i} + b) - 1 + \xi_i] = 0, \tag{12}$$

$$\mu_i \xi_i = 0. \tag{13}$$

Met invullen ontstaat de volgende duale Lagrange vermenigvuldiging:

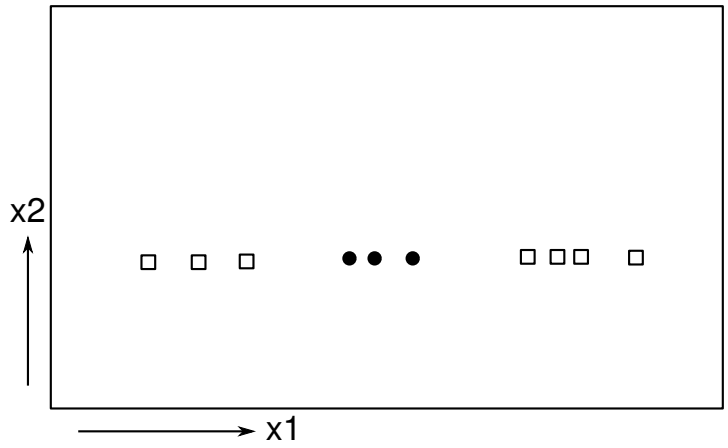
$$\begin{aligned} L_D &= \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j c_i c_j \mathbf{x_i} \cdot \mathbf{x_j} + C\sum_i \xi_i - \sum_i \lambda_i(c_i(\sum_j \lambda_j c_j(x_i) \cdot \mathbf{x_j} + b) - 1 + \xi_i) - \sum_i (C - \lambda_i)\xi_i \\ &= \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i \lambda_j c_i c_j \mathbf{x_i} \cdot \mathbf{x_j} \end{aligned}$$

Vanwege vergelijking (13) kan μ_i weggelaten worden. De Lagrange vermenigvuldiging is hetzelfde als de L_D voor het ruisvrije geval. De condities zijn echter wel anders in dit geval. Naast $\lambda_i \geq 0$ geldt ook $\lambda_i \leq C$. Ook hier brengt *quadratic programming* ons naar de uiteindelijke waarden voor $\mathbf{w}$ en b.

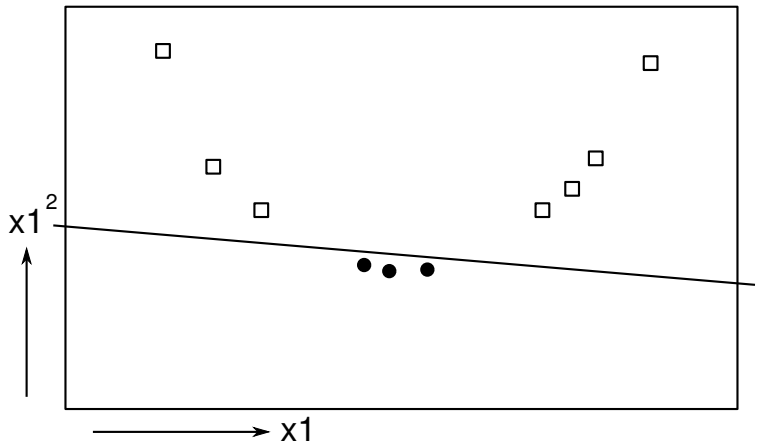
In figuur 5 is een dataset te zien die niet met een scheidingslijn geclassificeerd kan worden. Hoewel voor mensen dit niet een heel moeilijk probleem is, is dit onmogelijk foutloos te doen met de technieken die we tot nu toe hebben gezien. SVMs kunnen dit oplossen door de ruimte te transformeren, vaak naar hogere dimensies hoewel dat in dit geval niet nodig is. Het attribuut x_2 is voor alle datapunten gelijk en heeft dus geen onderscheidend vermogen. Het attribuut x_1 heeft dat wel en kan dus gebruikt worden voor de transformatie, we kiezen de volgende transformatie Φ:

$$\Phi : [x_1, x_2] \longrightarrow [x_1, x_1^2]$$

Het resultaat voor de datapunten is te zien in figuur 6.



FIGUUR 5. Een niet lineair separeerbare dataset



FIGUUR 6. De scheidingslijn in de getransformeerde ruimte

De dataset is nu wel lineair separeerbaar. Het transformeren van de ruimte garandeert dat we een lineaire scheidingslijn kunnen vinden, we kunnen immers zoveel dimensies toevoegen als we willen. Hoewel rekenen in hoogdimensionale ruimte veel tijd kost kan een SVM hier toch heel snel mee om gaan. Als we van een 2 dimensionale ruimte naar een 3 dimensionale ruimte willen transformeren en we kiezen de transformatie op een slimme manier kost het geen extra rekentijd. Een voorbeeld van een "snelle" transformatie is:

$$\Phi : [x_1, x_2] \longrightarrow [x_1^2, \sqrt{2}x_1x_2, x_2^2]$$

De duale Lagrange vergelijking voor de getransformeerde ruimte ziet er als volgt uit:

$$L_D = \sum_i \lambda_i - \frac{1}{2}\sum_{i,j} \lambda_i\lambda_j c_i c_j \Phi(\mathbf{x_i}) \cdot \Phi(\mathbf{x_j})$$

Het inproduct van $\Phi(\mathbf{x_i})$ en $\Phi(\mathbf{x_j})$ is computationeel erg duur in hoog dimensionale ruimtes, maar omdat Φ slim gekozen is kost het niet meer dan in de oorspronkelijke ruimte:

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y}) &= x_1^2y_1^2 + 2x_1x_2y1y_2 + x_2^2y_2^2 \\ &= (x_1y_1 + x_2y_2)^2 \\ &= (\mathbf{x} \cdot \mathbf{y})^2 \end{aligned}$$

Daar waar $\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$ staat kan nu dus $(\mathbf{x} \cdot \mathbf{y})^2$ worden ingevuld. Dit noemt men de *Kernel Trick*. Met als kernelfunctie $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^2$. Door het gebruik van zo'n kernelfunctie hoeven we niet precies te weten wat de transformatie is. Deze wordt immers nergens meer gebruikt. Een geschikte kernelfunctie rekent impliciet in een hogere dimensie, sommige kunnen zelfs in oneindig veel dimensies rekenen. Niet alle functies zijn geschikt als kernelfuncties, de belangrijkste voorwaarde voor een kernelfunctie is dat er een corresponderende transformatie bestaat die ervoor zorgt dat het resultaat van de kernelfunctie gelijk is aan het inproduct van de vectoren in de getransformeerde ruimte. Mercer heeft dit geformaliseerd:

THEOREM 2.1. Mercer's Theorem *A Kernel function K can be expressed as*

$$K(\mathbf{x}, \mathbf{y}) = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})$$

if and only if, for any function $g(x)$ *such that* $\int g(\mathbf{x})^2 d\mathbf{x}$ *is finite, then*

$$\int K(\mathbf{x}, \mathbf{y}) g(\mathbf{x}) g(\mathbf{x}) d\mathbf{x} d\mathbf{y} \geq 0$$

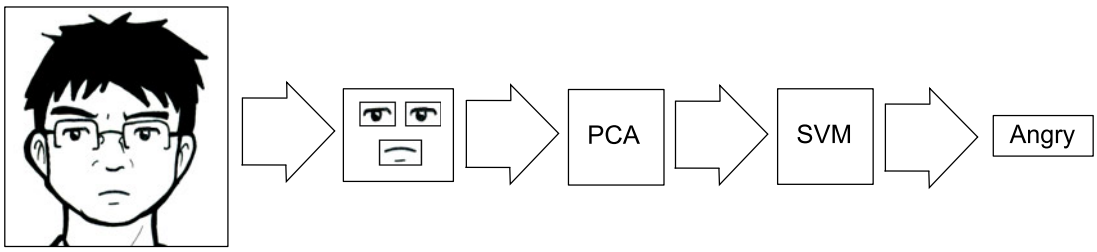
Twee veel gebruikte voorbeelden van kernelfuncties die aan Mercer's theorie voldoen zijn:

$$\begin{aligned} K(\mathbf{x}, \mathbf{y}) &= (\mathbf{x} \cdot \mathbf{y} + 1)^P, \\ K(\mathbf{x}, \mathbf{y}) &= e^{-||\mathbf{x}-\mathbf{y}||^2/(2\sigma^2)}. \end{aligned}$$

HOOFDSTUK 3

Methode

1. Algemene Opzet



FIGUUR 1. Schematische weergave van de classificatie methode

In figuur 1 is de algemene datastroom te zien. De afbeeldingen worden eerst in stukken gesneden, de *slices* en daarna genormaliseerd. Deze slices worden vervolgens als vector opgevat en achter elkaar gevoegd tot een grote vector die representatief is voor een originele afbeelding. Deze set van vectoren wordt in 10 gelijke folds (subsets) verdeeld, waarvan telkens 9 als trainingsset zullen dienen en 1 als testset. Op training- en testset combinatie wordt door middel van PCA dimensionaliteit reductie toegepast. Vervolgens wordt voor iedere expressie een SVM getraind. In de testfase wordt vervolgens voor iedere expressie de bijbehorende SVM toegepast en worden de resultaten samengevoegd tot de uiteindelijke classificatie.

2. Keuze parameters

2.1. Facial landmarks. Eén van de twee procedures gebruikt PCA om dimensionaliteit te reduceren. PCA kiest zijn principale componenten op basis van de covariantie matrix van de data en is unsupervised. Om het classificatieproces te sturen delen we het gezicht eerst op in verschillende facial landmarks en van deze reduceren we de dimensionaliteit. We hopen zo principale componenten te krijgen die gecorreleerd zijn met de bijbehorende expressies. Deze correlaties kunnen dan geleerd worden door het classificatiealgoritme in de volgende stap. Nu rest ons de vraag welke facial landmarks gekozen moeten worden. We hebben ons laten inspireren door [**4**]. Hierin worden een aantal overlappende segmenten gekozen rond enkele karakteristieke punten in gezicht: de wenkbrauwen, de onderste helft van de ogen en de mond. Dit levert de volgende segmenten (tussen haakjes staat de grootte van het plaatje in pixels):

- Linkeroog boven (80x55)
- Linkeroog onder (80x55)
- Rechteroog boven (80x55)
- Rechteroog onder (80x55)
- Linker mondhoek (50x55)
- Middelste deel mond (55x80)
- Rechter mondhoek (50x55)

Intuïtief zit in deze segmenten de meeste variatie bij een expressie in een gezicht. Andere mogelijke segmenten zijn het voorhoofd of delen van de wangen. In een vervolg studie zou ik wat meer onderzoek doen naar welke delen in het gezicht de meeste informatie bevatten. Mogelijk kan dat op een veel gedetailleerder nivau als relatief grote segmenten die we nu uitknippen.

2.2. PCA. De parameter keuze voor PCA is vrij eenvoudig. De enige vrijheidsgraad is het aantal eigenvectoren dat uiteindelijk gebruikt wordt. In plaats van een constante waarde hiervoor te kiezen hebben we dit gebaseerd op de variantie van de data. We kiezen precies zoveel eigenvectoren dat we een gegeven precisie kunnen halen als we de dataset met deze vectoren uitdrukken. Dit levert per fold (en uiteraard ook per slice) dus een andere waarde op maar dat is geen probleem voor de vervolg berekening omdat er per set nieuwe SVMs worden getraind die dus rekening houden met de lengte van de vectoren. We hebben gekozen voor de volgende varianties 95%, 85% en 75%.

Voor de vervolg berekening worden de eigenvectoren van de slices per sample achterelkaar gezet. Dit resulteert in vectoren van de volgende lengten (gemiddeld):

$$\begin{aligned} 95\% &\rightarrow 520 \\ 85\% &\rightarrow 182 \\ 75\% &\rightarrow 96 \end{aligned}$$

2.3. SVM. SVMs hebben vrij veel keuze in parameters. Om toch voor alle parameters een goede waarde te kiezen hebben we een grid search toegepast. Eerst kiezen we voor iedere parameter een grof interval en testen alle combinaties. De parameter combinaties die het hoogste scoren worden verfijnd met kleinere intervallen. We behandelen hier alleen de gekozen parameterwaarden voor de laatste iteratie. De Torch3 [**19**] implementatie heeft niet hetzelfde minimalisatie probleem geïmplementeerd zoals beschreven in de theorie hierboven. Hierdoor zijn niet alle parameters die in de theorie beschreven worden van belang. We hebben 2 soorten kernel functies gebruikt. Buiten kernel specifieke parameters zijn er ook globale:

(1) *Accuracy* geeft de maximale toegestane fout aan. Er wordt doorgerekend tot er onder deze waarde gescoord wordt.
(2) C is de C uit de theorie hierboven en geeft de striktheid aan waarmee er met foutieve classificaties omgegaan wordt

Voor de *Accuracy* zijn de waarden $0.01, 0.1, 1$ gebruikt. Voor de C parameter zijn de waarden $1, 10, 100, 1000$ gebruikt. Voor beide kernels is er nog één specifieke parameter. Voor de Dot-kernel ($K(\mathbf{x}, \mathbf{y}) = s * (\mathbf{x} \cdot \mathbf{y})$) zijn voor s de waarden $0.1, 0.01, 0.001$ getest. Voor de Gaussian-kernel ($K(\mathbf{x}, \mathbf{y}) = e^{-g||\mathbf{x}\cdot\mathbf{y}||^2}$) zijn voor g de waarden $10^{-4}, 10^{-5}, 10^{-6}$. De waarden voor g vallen bijzonder laag uit. Waarom deze kernel beter blijft scoren voor lage g is niet helemaal duidelijk geworden.

Omdat SVMs alleen binaire beslissingen kunnen nemen, wordt er voor iedere expressie een aparte SVM getraind. De uitkomsten van deze SVMs wordt gecombineerd door de expressie te kiezen met de hoogste mate van zekerheid.

3. Datasets

We hebben op diverse plaatsen datasets gevonden die in principe bruikbaar zouden kunnen zijn om ons classificatiealgoritme op te testen. Enkele voorbeelden:

FERET database: Een database opgezet door het NIST in de Verenigde Staten om onderzoek te ondersteunen naar automatische identificatie van mensen. Een deel van de files is ook geschikt om expressie herkenningsalgoritmen mee te testen. We hebben deze dataset aangevraagd maar nooit mogen ontvangen. Voor meer informatie zie [**15**].

RU BSI dataset: Een dataset opgesteld op de Radboud Universiteit door het Behavioural Science Institute. Proefpersonen zijn geïnstrueerd om bepaalde Facial Action Units te gebruiken en dit is vastgelegd. De beschrijving van de dataset klinkt veelbelovend, echter kwamen we te laat achter het bestaan van deze dataset. Zie [**16**].

MMIFaceDB: Dataset van Maja Pantic van het Imperial College London. We hebben toegang gekregen tot deze dataset en de data deels voorbewerkt. Het probleem met deze set is echter de kleine hoeveelheid verschillende sessies (175 samples), dus we zijn eerst verder gaan zoeken naar een grotere dataset. De website is op het moment van schrijven offline, zie [**18**].

Cohn-Kanade dataset: Deze dataset is ook opgezet als benchmark voor herkenningsalgoritmen. De plaatjes zijn geannoteerd met de Facial Action Units die de proefpersonen na instructie vertonen. In deze dataset zitten 487 samples dus we hebben ervoor gekozen met deze dataset verder te werken en de MMIFaceDB even te laten voor wat het is. Voor meer informatie zie [**2**].

4. Preprocessing

De herkenningsalgoritmen die we gaan testen nemen het volgende aan:

- De gezichtsfeatures zijn uitgelijnd
- De foto's hebben allemaal hetzelfde formaat
- De foto's zijn allemaal in grijswaarden en onder contante lichtomstandigheden
- Binnen de dataset is een evenwichtige klassendistributie
- De klasse van een foto's is bekend

De Cohn-Kanade dataset voldoet initieel aan geen van deze aanname en zal dus met preprocessing bijgeschaafd moeten worden.

De Cohn-Kanade set bestaat uit series van foto's die de overgang van neutraal naar een bepaalde combinatie van facial action units tonen. Voor dit onderzoek zijn alleen de laatste foto's van iedere serie gebruikt. Bij de dataset hoort een document dat combinaties van facial action units koppelt aan expressies. Deze koppeling is niet overal even duidelijk, daar waar onzekerheid was zijn de foto's handmatig gekoppeld aan een expressie.

Om gezichtsfeatures te detecteren was de eerste voorkeur om gebruik te maken van dezelfde gezichtsdetectiebibliotheek als in [**22**] gebruikt is (voor info zie [**11**]). Echter leverde deze toch minder goede resultaten als [**22**] suggereert en bovendien signaleerde deze tool regelmatig gezichten op plekken waar geen gezichten waren. Het algoritme levert alleen een bounding box van het gezicht op en we hoopten op basis hiervan de belangrijkste segmenten van het gezicht te kunnen onderscheiden (gebruik makend van enkele vaste verhoudingen). Echter is de opgeleverde bounding box niet altijd even groot ten opzichte van het te detecteren gezicht, wat het heel moeilijk maakt om met enige nauwkeurigheid segmenten vast te stellen. Na een tip van Xuelong Li zijn we een aantal handige toolkits op het spoor gekomen namelijk OpenCV [**7**] en Stasm [**14**]. Stasm is gebaseerd op OpenCV en levert een file op met daarin de coördinaten van de verschillende facial landmarks. Stasm gebruikt technieken beschreven in [**24**] om het gezicht te detecteren en vervolgens worden Active Shape Models gebruikt om de verschillende facial landmarks te lokaliseren (zie [**13**] voor meer informatie hierover).

FIGUUR 2. Afbeelding uitgesneden met Stasm en waarin de facial landmarks zijn aangegeven. Links origineel, rechts bewerkt.

In figuur 2 is te zien wat Stasm doet. De facial landmarks (in de rechter afbeelding) zijn met elkaar verbonden door een lijn, in totaal zijn er 68 facial landmarks. Op basis van de facial landmarks van Stasm snijden we de verschillende onderdelen van het gezicht uit, deze stukjes gezicht noemen we *slices*. Alle slices worden naar grijswaarden geconverteerd. Daarna wordt er voor het verschil in lichtintensiteit gecompenseerd door het bereik van waarden zó te verschuiven, dat het gemiddelde op de helft van de maximale waarde ligt (dus een translatie van waarden, geen schaling). Uiteindelijk worden alle slices naar het eerder genoemde formaat geschaald. Het resultaat is te zien in figuur 3.

FIGUUR 3. De genormaliseerde slices van de afbeelding in figuur 2

.

Om de klassendistributie gelijk te krijgen voegen we willekeurig gezichten dubbel toe. Hierdoor voorkomen we dat classificatiealgoritmen een bias ontwikkelen voor klassen die vaker voorkomen. Voor de evaluatie doen we 10 fold crossvalidatie waarbij we zorgen dat dubbele samples in dezelfde fold zitten (zodat de testset geen al eerder geziene elementen bevat). Iedere fold bevat ongeveer 70 gezichten.

HOOFDSTUK 4

Resultaten en Conclusie

1. Resultaten

De training van alle data duurde 2 nachten op een Intel Core 2 Duo E6750. Vooral het berekenen van de covariantie matrices nam veel tijd in beslag. De test runs duurde ongeveer 2 seconden per afbeelding, waarvan bijna alles gaat zitten in de berekening van Stasm, dus het preprocessen en het maken van de slices. Afhankelijk van de toepassing is de rest van het algoritme dus snel genoeg voor realtime toepassingen.

	ANG	**DIS**	**FEA**	**HAP**	**NEU**	**SAD**	**SUR**	**Average**
Dot	73.75	79.77	80.57	93.58	86.52	80.83	92.42	83.92
Gauss	78.93	86.10	88.64	94.83	87.42	84.75	94.86	87.93

TABEL 1. Herkenningspercentage voor de kernelfuncties per expressie

In tabel 1 kunnen we zien dat niet alle klassen even goed herkenbaar zijn. Happiness en Surprise scoren bijzonder hoog en Anger scoort vrij laag. Dat Anger moeilijker is dan Happiness en Surprise komt overeen met de resultaten van [**20**].

		Classificatie						
		ANG	**DIS**	**FEA**	**HAP**	**NEU**	**SAD**	**SUR**
Werkelijke klasse	**ANG**	64	8	0	0	10	18	0
	DIS	35	51	4	0	8	2	0
	FEA	6	3	67	14	3	4	3
	HAP	1	0	10	87	2	0	0
	NEU	3	1	1	0	79	11	3
	SAD	14	2	0	0	19	62	3
	SUR	0	1	0	1	5	4	89

TABEL 2. Confusiematrix voor de Gaussian kernelfunctie

		Classificatie						
		ANG	**DIS**	**FEA**	**HAP**	**NEU**	**SAD**	**SUR**
Werkelijke klasse	**ANG**	49	11	6	0	14	20	0
	DIS	12	68	2	0	10	8	0
	FEA	6	4	56	20	5	9	0
	HAP	0	1	6	91	2	0	0
	NEU	3	1	2	1	77	12	2
	SAD	10	3	0	1	16	69	1
	SUR	0	2	1	2	3	8	84

TABEL 3. Confusiematrix voor de Dot kernelfunctie

In de confusiematrices in tabel 2 en tabel 3 is te zien dat Disgust vaak gezien wordt als Anger en dat Anger vaak gezien wordt als Sadness. In figuur 1 zijn twee gezichten te zien die door beide kernelfuncties fout geclassificeerd werden. De Stasm tool werkt hier prima op dus de fout zit in het PCA- en SVM-gedeelte. De klassen die het algoritme hiervoor vind zijn niet heel ver gezocht, het linker gezicht toont vrij weinig emotie en het rechter gezicht heeft de ogen erg ver geopend.

FIGUUR 1. Foutief geclassificeerde gezichten, klasse: Anger en Anger, resultaat: Neutral en Fear

Uit de confusiematrices zijn ook de uiteindelijke herkenningspercentages af te leiden, 71.46% voor de Gaussian kernelfunctie over en 70.77% voor de Dot kernelfunctie. Het grote verschil tussen de gemiddelde herkenning per expressie en de samengevoegde conclusie zit in het feit dat de klassen niet evengoed herkenbaar zijn. Anger scoort nooit een hoge zekerheid en Happy wel, op het moment dat deze twee expressies concurreren zal Happy vrijwel altijd winnen ook al scoort Anger heel hoog voor zijn doen. Dit probleem zou gereduceerd kunnen worden door als laatste stap niet voor de klasse te kiezen met de hoogste zekerheid maar de uitkomsten van de SVMs door een boostingalgoritme te halen of een neuraal netwerk. Deze kunnen rekening houden met de herkenbaarheid van de verschillende klassen.

2. Conclusie

We hebben Principale Componenten Analyse en Support Vector Machines gecombineerd en toegepast voor het herkennen van expressies in gezichten. We hebben hiermee een herkenningsscore behaald van ruim 71%. We vermoeden dat door middel van boosting deze score nog te verbeteren is. De score is ruim boven de statistiche kans van 14% met zeven klassen dus dit onderzoek kan als succesvol worden bestempeld. De herkenningsscores per expressie vertonen overeenkomsten met de scores uit het onderzoek van [**20**] dat de herkenningsscore van mensen onderzocht. De hier behaalde scores komen helaas niet in de buurt van de scores behaald door [**1**]. Dit kan deels verklaard worden door het feit dat [**1**] bewegende beelden gebruikt en daardoor meer informatie verkreeg over hoe gezichten er in neutrale positie uitzien.

Het algoritme zoals hier gebruikt is zou, afhankelijk van de toepassing, gebruikt kunnen worden voor realtime toepassingen. Het zoeken naar het gezicht in de foto en vervolgens de facial landmarks bepalen kost bijna 2 seconden en de vervolg classificatie kost enkele tientalle milliseconden. Er is niet gekeken naar het algoritme dat de facial landmarks bepaalt, mogelijk is dit nog sneller te doen.

Bibliografie

[1] MS Bartlett, G. Littlewort, C. Lainscsek, I. Fasel, and J. Movellan. Machine learning methods for fully automatic recognition of facial expressions and facial actions. 2004.

[2] Cohn and Kanade. Cohn-Kanade Dataset. Available from: `http://vasc.ri.cmu.edu/idb/html/face/facial_expression/index.html`.

[3] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(6):681–685, June 2001. `doi:10.1109/34.927467`.

[4] M.N. Dailey and G.W. Cottrell. PCA = Gabor for expression recognition. *UCSD CSE TR CS-629*, 1999.

[5] F. Dornaika and F. Davoine. Facial expression recognition using auto-regressive models. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 2, pages 520–523, 20-24 Aug. 2006. `doi:10.1109/ICPR.2006.539`.

[6] JGF Francis. The QR Transformation. *The Computer Journal*, 4(4):332–345, 1962.

[7] Adrian Kaehler et al Gary Bradski. OpenCV. Available from: `http://opencvlibrary.sf.net/`.

[8] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[9] Haxby, Hoffman, and Gobbini. The distributed human neural system for face perception. *Trends Cogn Sci*, 4(6):223–233, Jun 2000.

[10] G. W. Humphreys, N. Donnelly, and M. J. Riddoch. Expression is computed separately from facial identity, and it is computed separately for moving and static faces: neuropsychological evidence. *Neuropsychologia*, 31(2):173–181, Feb 1993.

[11] John Hershey Bret Fortenberry Josh Susskind Ian Fasel, Ryan Dahl and Javier R. Movellan. The Machine Perception Toolbox. Available from: `http://mplab.ucsd.edu/grants/project1/free-software/mptwebsite/API/`.

[12] E. Kreyszig and E. Kreyszig. *Advanced engineering mathematics*. Wiley New York, 1993.

[13] S. Milborrow. *Locating Facial Features with Active Shape Models*. Master's thesis. University of Cape Town (Department of Image Processing), 2007. Available from: `http://www.milbo.users.sonic.net/stasm`.

[14] S. Milborrow and F. Nicolls. Locating facial features with an extended active shape model. *ECCV*, 2008. Available from: `http://www.milbo.users.sonic.net/stasm`.

[15] NIST. FERET Database. Available from: `http://www.itl.nist.gov/iad/humanid/feret/`.

[16] Radboud University of Nijmegen. RU BSI Dataset. Available from: `http://www.ru.nl/wetenschapsagenda/editie_17_-_21_05_08/nieuws/vm/item_721253/beeld_duizenden/`.

[17] T. Pang-Ning, M. Steinbach, and V. Kumar. *Introduction to data mining*. Boston: Person Addison Wesley EducatioPress, 2005.

[18] Maja Pantic. MMIFaceDB. Available from: `http://www.mmifacedb.com/`.

[19] Johnny Marithoz Ronan Collobert, Samy Bengio. Torch3Vision. Available from: `http://www.torch.ch/`.

[20] James A. Russell. Is There Universal Recognition of Emotion From Facial Expression? A Review of the Cross-Cultural Studies. *Psychological Bulletin*, 115(1):102–141, 1994. Available from: `http://www2.bc.edu/~russeljm/publications/psyc-bull1994.pdf`.

[21] Caifeng Shan, Shaogang Gong, and Peter W. McOwan. Dynamic Facial Expression Recognition Using. A Bayesian Temporal Manifold Model. In *BMVC*, Edinburgh, September 2006. Available from: `http://www.macs.hw.ac.uk/bmvc2006/papers/099.pdf`.

[22] JM Susskind, G. Littlewort, MS Bartlett, J. Movellan, and AK Anderson. Human and computer recognition of facial expressions of emotion. *Neuropsychologia*, 2006.

[23] M. Turk and A. Pentland. Eigenfaces for Recognition. *Journal of Cognitive Neuroscience*, 3(1):71–86, 1991.

[24] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001. Available from: `citeseer.ist.psu.edu/viola01rapid.html`.