Klassifisering

Notat for TMA4240/TMA4245 Statistikk*

Insitutt for matematiske fag,
NTNU

21. august 2012

Innen maskinlæring studerer man algoritmer som tillater datamaskiner å utvikle atferd på grunnlag av empiriske data. Læring betyr i denne sammenhengen at maskinens ytelse blir bedre jo mer data den har tilgjengelig. Ber vi f.eks. maskinen om å løse en oppgave, og svaret vi får tilbake enten er riktig eller galt, håper vi at andelen av gale svar vil bli mindre etter hvert som maskinen eksponeres for flere data.

Klassifiseringsproblemet

Anta at vi har et sett med n observasjoner $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$, av en m-dimensjonal forklaringsvariabel $\mathbf{X} = (x_1, \ldots, x_m)^T \in \mathcal{X}$ og en avhengig variabel, eller respons, $Y \in \mathcal{Y}$. Anta videre at mengden $\mathcal{Y}$ består av disjunkte klasser, slik at for hvert element $y \in \mathcal{Y}$, tilhører y nøyaktig én slik klasse.

Da kan vi se på observasjonene som eksempler som uttrykker sammenhengen mellom variablene X og Y. Vi betrakter så et nytt par $(\mathbf{x}_0, y_0)$ og ber maskinen løse følgende problem:

> Gitt dataene $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$ og punktet $\mathbf{x}_0 \in \mathcal{X}$, gi en prediksjon $\hat{y}_0$ av responsen y_0.

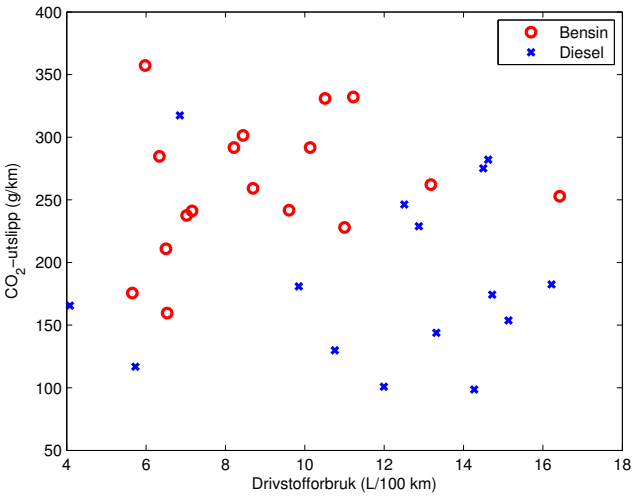
Vi sammenlikner deretter prediksjonen med det riktige svaret, og sier at prediksjonen var korrekt hvis $\hat{y}_0 = y_0$, eller feil dersom $\hat{y}_0 \neq y_0$.

Eksempel: Busker og trær

La x_i, $i = 1, \ldots, n$ være målinger av høyden til en rekke busker og trær i en skog, og la y_i være lik B for busk eller T for tre. Vi kan så måle høyden x_0 til en ny busk, og be om en prediksjon $\hat{y}_0$ av klassen. Vi vet at det er en busk vi har målt, altså er det riktige svaret $y_0 = B$. Dette kan vi bruke for å vurdere om prediksjonen er korrekt.

For mennesker er det vanligvis enkelt å skille mellom busker og trær, men baserer man seg på høyde alene vil man få problemer med spesielt høye busker eller lave trær. Når man gjenkjenner et bonsaitre som et lite tre snarere enn en busk, er det fordi man i tillegg til høyden også vurderer proporsjonene, og kanskje også andre egenskaper ved planten. I dette eksempelet er $m = 1$, dvs. den uavhengige variabelen har bare én dimensjon. Hadde

*Notatet er skrevet av Jacob Skauvold i samarbeid med Arild Brandrud Næss. Dersom du finner feil eller har forslag til forbedringer, ta kontakt med Arild Brandrud Næss, arild.ntnu@gmail.com.



Figur 1 – Spredningsplott av observasjoner av drivstoffforbruk og karbondioksidutslipp for 17 bensin- og 15 dieselbiler. Dataene er generert, og figuren laget, med `drivstoffdata.m`.

vi brukt en uavhengig variabel med flere dimensjoner (f.eks. kunne vi inkludert forholdet mellom bredde og høyde) kunne vi forventet bedre, mer nøyaktige prediksjoner. De fleste anvendelser av klassifisering bruker data med langt flere dimensjoner.

Eksempel: Bensin- og dieselbiler

Anta nå at vi har observasjoner av drivstofforbruk, x_1, målt i L/100km, og CO_2-utslipp, x_2, målt i g/km, for $n = 32$ biler, hvorav 17 tilhører klassen *Bensin*, og 15 tilhører klassen *Diesel*. Situasjonen er vist i figur 1.

Hvis vi ønsker å undersøke om variablene x_1 og x_2 egner seg for å predikere klassetilhørigheten (bensin/diesel) til biler, trenger vi

(i) en algoritme som, gitt et sett med observasjoner $(\mathbf{x_i}, y_i)$, $i = 1, \ldots, n$, og et testpunkt $\mathbf{x}_0 = (x_{01}, x_{02})^T$, returnerer en prediksjon $\hat{y}_0$, og

(ii) en partisjonering av de tilgjengelige dataene i et *treningssett*, som brukes til å trene opp algoritmen, og et *testsett* som brukes for å sjekke hvor ofte prediksjonene er korrekte.

Testpunktet $\mathbf{x}_0$ hentes fra testsettet. Jo flere ulike testpunkter man bruker, jo bedre kan man anslå hvor godt algoritmen vil fungere på nye data, som ikke er en del av det tilgjengelige datasettet. Oppdelingen av dataene er nødvendig fordi vi ikke kan bruke de samme dataene for å trene opp algoritmen, og for å estimere hvor ofte den gir korrekte prediksjoner. Det ville i så fall gi et for optimistisk feilestimat. Det er imidlertid mulig å bruke ulike partisjoneringer hver for seg, for så å kombinere resultatene for å oppnå et mer robust estimat, såkalt kryssvalidering.

k nærmeste naboer-algoritmen

Vi tenker oss nå at vi får et punkt $\mathbf{x}_0 = (x_{01}, x_{02})^T$ som representerer en ny bil, altså ikke en av de n bilene vi har observasjoner av fra før. Hvordan skal vi gå fram for å tildele $\mathbf{x}_0$

en prediksjon $\hat{y}_0$? Én idé er å se på de nærmeste naboene til punktet $\mathbf{x}_0$. Det kan vi gjøre så lenge vi har et begrep om avstand mellom punktene. Her kan vi bruke euklidsk avstand i planet, altså 2-normen $||\cdot||_2 : \mathbb{R}^2 \to \mathbb{R}$.

For hvert punkt $\mathbf{x}_1, \ldots, \mathbf{x}_n$ definerer vi nå avstanden $d(\mathbf{x}_0, \mathbf{x}_i)$ fra $\mathbf{x}_0$ til $\mathbf{x}_i$ som,

$$d(\mathbf{x}_0, \mathbf{x}_i) = ||\mathbf{x}_0 - \mathbf{x}_i||_2, \qquad i = 1, \ldots, n.$$

Om vi ordner avstandene i stigende rekkefølge, $d\left(\mathbf{x}_0, \mathbf{x}_{(1)}\right) \leq \ldots \leq d\left(\mathbf{x}_0, \mathbf{x}_{(n)}\right)$, får vi samtidig en ordning av punktene $\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(n)}$. Den nærmeste naboen til $\mathbf{x}_0$ er punktet $\mathbf{x}_{(1)}$. De k nærmeste naboene til $\mathbf{x}_0$ er elementene i mengden $\{\mathbf{x}_{(1)}, \ldots, \mathbf{x}_{(k)}\}$.

Ideen er nå å velge $\hat{y}_0$ utfra en "flertallsavgjørelse" blant de k nærmeste naboene til $\mathbf{x}_0$. En konsekvens er at $\hat{y}_0$ generelt vil avhenge av k. For eksempel kan to av de tre nærmeste naboene være dieselbiler, mens tre av de fem nærmeste er bensinbiler. Hvis ingen klasser har flest representanter blant de k nærmeste naboene, fordi flere klasser har like mange, velger vi én av disse klassene tilfeldig.

Tilpasning og validering

Siden prediksjonen $\hat{y}_0$ avhenger av verdien til k, vil også ytelsen til algoritmen gjøre det. Den beste verdien vi kan velge for k er den som gir minst mulig feilrate. Hvilken verdi som er best vil avhenge av dataene, derfor må algoritmen tilpasses et datasett. Samtidig ønsker vi å unngå overtilpasning, altså det at algoritmen oppnår god ytelse når den anvendes på treningsdataene, men vesentlig dårligere ytelse når den brukes på uavhengige data fra samme fordeling. Dette kan unngås ved å dele opp de tilgjengelige dataene, og bruke forskjellige deler av dataene for å trene algoritmen og for å anslå feilraten.

Si at vi har et datasett bestående av N par $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, N$ med ukjent fordeling. Vi ønsker å finne en god verdi av k for data fra denne fordelingen, og vi ønsker å anslå feilraten den tilpassede algoritmen oppnår når den brukes på slike data. Disse to målsetningene henger sammen; hvis vi først estimerer feilraten for en håndfull verdier av k, kan vi enkelt velge den beste av dem ved å ta den som gir den laveste feilraten.

Vi begynner med å dele inn datasettet i to deler. Den første delen, $(\mathbf{x}_i, y_i)$, $i = 1, \ldots, n$ kaller vi treningssettet, og resten, $(\mathbf{x}_i, y_i)$, $i = n+1, \ldots, N$, kaller vi testsettet. Foreløpig setter vi testsettet til side. Det skal brukes til slutt, når vi skal estimere feilraten som oppnås med den beste verdien av k.

Vi fortsetter ved igjen å dele treningssettet i to deler. Den ene av disse, med n_{dev} observasjoner, kaller nå vi utviklings- eller valideringssettet, mens vi refererer til de resterende $n - n_{\text{dev}}$ observasjonene som treningssettet. Oppdelingen av dataene er vist skjematisk i figur 2.

Figur 2 – Partisjonering av datasettet i et treningssett, et utviklingssett og et testsett.

Vi velger ut noen verdier av k, si $k_1, \ldots, k_t$ som vi vil prøve. For hver verdi k_j, $j = 1, \ldots, t$ gjør vi så følgende:

1. Bruk observasjonene i treningssettet som treningsdata, og bruk algoritmen, med $k = k_j$ til å predikere klassene til observasjonene i utviklingssettet.

2. For hver av de n_{dev} observasjonene $(\mathbf{x}_i, y_i)$ i utviklingssettet, definer indikatorvariabelen $I_{\mathbf{x}_i}$ slik at
$$I_{\mathbf{x}_i} = \begin{cases} 1 & \text{hvis } \hat{y}_i \neq y_i \\ 0 & \text{hvis } \hat{y}_i = y_i \end{cases}.$$

3. Definer den estimerte feilraten $\hat{p}_{k_j}$, når $k = k_j$, som gjennomsnittet av $I_{\mathbf{x}_i}$ når vi summerer over indeksene som svarer til utviklingssettet,
$$\hat{p}_{k_j} = \frac{\sum_i I_{\mathbf{x}_i}}{n_{\text{dev}}}.$$

Når vi er ferdige, kan vi sammenlikne estimatene $\hat{p}_{k_1}, \ldots, \hat{p}_{k_t}$. For å velge den beste verdien av k tar vi den verdien k^* som gir den laveste estimerte feilraten, altså
$$k^* = \arg\min_{k_j} \hat{p}_{k_j}, \qquad k_j \in \{k_1, \ldots, k_t\}.$$

Det gjenstår å bruke testsettet til å finne et estimat for feilraten når vi bruker den beste verdien av k. Vi setter derfor $k = k^*$, og betrakter observasjonene $(\mathbf{x}_i, y_i)$, $i = n+1, \ldots, N$ i testsettet. På samme måte som over definerer vi indikatorvariabelen $I_{\mathbf{x}_i}$ for hver av disse observasjonene, dvs. $I_{\mathbf{x}_i}$ har verdien 1 hvis prediksjonen er feil, eller 0 hvis prediksjonen er riktig. Estimatet $\hat{p}_{k^*}$ for feilraten med $k = k^*$ blir nå
$$\hat{p}_{k^*} = \frac{1}{N-n} \sum_{i=n+1}^{N} I_{\mathbf{x}_i}.$$

Kryssvalidering

Med inndelingen over ble det opprinnelige treningssettet delt opp i et nytt treningssett og et utviklingssett, og disse ble så brukt for henholdsvis trening og validering. En alternativ måte å partisjonere treningsdataene på, er å dele dem inn i K like store deler, bruke én del som utviklingssett, og la de øvrige $K-1$ delene utgjøre treningssettet, slik det er vist i figur 3. Dette gjentas K ganger, hver gang med et nytt valg av trenings- og utviklingssett,

Figur 3 – Partisjonering av datasettet i et treningssett og et testsett, og videre partisjonering av testsettet i K like store deler som hver inneholder n/K observasjoner. Én av delene brukes til validering, resten til trening.

slik at alle de K delene blir brukt til både trening og validering, og hver del brukes til validering nøyaktig én gang. Hver partisjonering gir opphav til et eget estimat av feilraten. Disse kan så kombineres til et mer robust estimat ved å ta gjennomsnittet av dem.

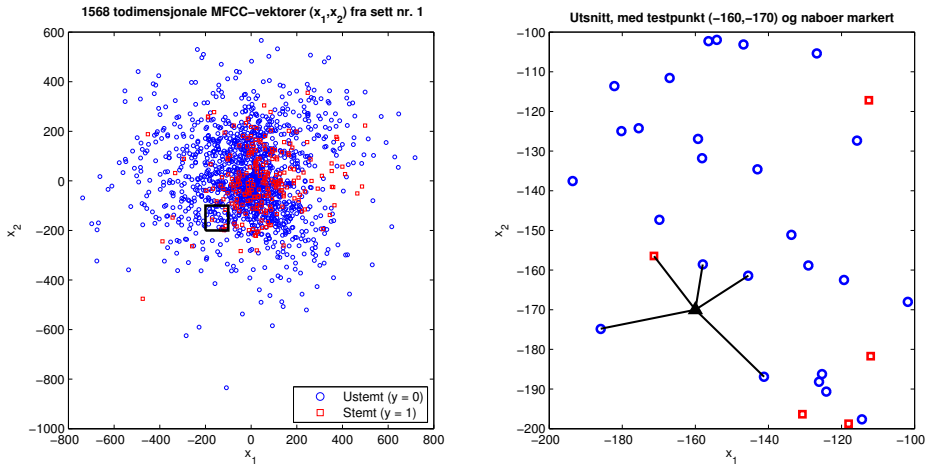
Talegjenkjenning

En av anvendelsene av klassifisering finner vi innen talegjenkjenning. Her er målet å analysere opptak av tale, og identifisere hva som blir sagt. I mappen `tale.zip` finnes eksempler på slike opptak og transkripsjoner av opptakene, både ord for ord og fonem for fonem.

Fonemene kan klassifiseres på forskjellige måter. Å identifisere hvilket fonem (blant 98 mulige) som høres i opptaket, er et vanskelig problem som vi kommer tilbake til. Vi begynner med et enklere problem, nemlig å klassifisere fonemer som stemte eller ustemte lyder. Vi har altså to klasser: 0 for ustemte lyder, og 1 for stemte lyder.

Som uavhengig variabel bruker vi 39-dimensjonale MFCC-vektorer (Mel-Frequency Cepstral Coefficients). Hver vektor representerer ett fonem. Det er også mulig å bruke kun to av de 39 komponentene.

I matlabfunksjonen `knn.m` er k nærmeste naboer-algoritmen implementert. Vi vil gjøre som beskrevet over, og kjøre algoritmen med forskjellige verdier av k, for å finne ut hvilken som er mest gunstig. Vi har tre sett med data, kalt sett nr. 1, 2 og 3. Disse er lagret i fila `taledata.mat`. Hvert sett består av et antall MFCC-vektorer, i både 2- og 39-dimensjonale utgaver, samt label-vektorer som inneholder klassene til hvert fonem, både for stemt/ustemt-problemet og for generell klassifisering av fonemer. Vi bruker sett nr. 1 som treningssett, nr. 3 som utviklingssett og nr. 2 som testsett. I figur 4 ses et spredningsplott av de todimensjonale MFCC-vektorene (x_{1i}, x_{2i}), $i = 1, \ldots, 1568$ i treningssettet. Man ser raskt at de to klassene ikke er lineært separerbare, dvs. det finnes ingen linje som skiller dem, slik at alle de ustemte lydene havner på den ene siden, og alle de stemte på den andre. Det er med andre ord vanskelig å skille klassene fra hverandre, men spredningsplottet viser også at punktene fra samme klasse ofte ligger nær hverandre i klynger. Dette gjør at k nærmeste naboer-algoritmen likevel fungerer på dette problemet.



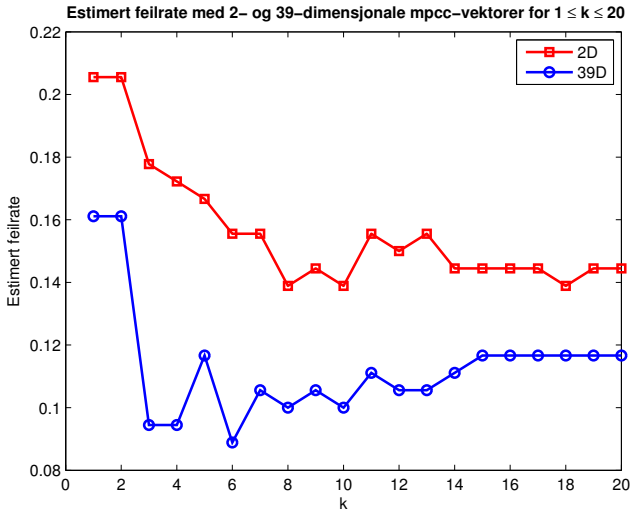
Figur 4 – Venstre: todimensjonale MFCC-vektorer fra sett nr. 1. Ustemte lyder er markert med blå sirkler, stemte lyder med røde firkanter. Høyre: Utsnitt av plottet til venstre. Punktet $(-160, -170)$ er markert med en svart trekant, og linjer er trukket ut til de fem nærmeste naboene. Begge plottene er laget med `makescatter.m`.

Ved å følge fremgangsmåten beskrevet under *Tilpasning og validering* kommer vi frem til de estimerte feilratene i tabell 1. Tabellen inneholder feilrater for både 2- og 39-dimensjonale MFCC-vektorer. I tillegg er feilratene plottet mot k i figur 5.

Vi legger merke til to ting ved plottet i figur 5. For det første oppnår vi jamt over lavere feilrater med 39-dimensjonale MFCC-vektorer enn med todimensjonale vektorer.

Tabell 1 – Estimert feilrate for verdier av k mellom 1 og 20, med 2- og 39-dimensjonale MFCC-vektorer. Estimatene er beregnet med `tryk.m`.

k	Feilrate (2D)	Feilrate (39D)
1	0.2056	0.1611
2	0.2056	0.1611
3	0.1778	0.0944
4	0.1722	0.0944
5	0.1667	0.1167
6	0.1556	0.0889
7	0.1556	0.1056
8	0.1389	0.1000
9	0.1444	0.1056
10	0.1389	0.1000
11	0.1556	0.1111
12	0.1500	0.1056
13	0.1556	0.1056
14	0.1444	0.1111
15	0.1444	0.1167
16	0.1444	0.1167
17	0.1444	0.1167
18	0.1389	0.1167
19	0.1444	0.1167
20	0.1444	0.1167



Figur 5 – Feilratene i tabell 1 plottet mot k. Begge kurvene avtar raskt i starten, og flater så ut når k øker.

Dette er som forventet, siden flere komponenter betyr mer informasjon om lydene som representeres av vektorene. Det at vi observerer en vesentlig forskjell mellom feilratene indikerer at algoritmen klarer å gjøre nytte av den ekstra informasjonen som er inneholdt i de 39-dimensjonale vektorene, sammenliknet med de todimensjonale.

For det andre har de to kurvene ganske lik oppførsel; i starten avtar feilraten raskt med k, men så flater kurven ut, og for større verdier av k er det liten eller ingen endring i feilrate når k økes. Siden beregningene blir mer ressurskrevende jo større k er, bør vi velge k i begynnelsen av det flate partiet av kurvene, altså i området 10 til 12 i begge tilfellene (altså for både 2 og 39 dimensjoner).

Hadde vi brukt kriteriet ovenfor, altså å velge den verdien av k som gir lavest feilrate, måtte vi valgt $k = 8$ eller $k = 10$ i det todimensjonale tilfellet, og $k = 6$ i det 39-dimensjonale tilfellet. Spørsmålet er om de små fluktuasjonene som gjør kurvene takkete i stedet for glatte, er et resultat av den underliggende populasjonen (som er hva vi ønsker å tilpasse algoritmen til) eller et resultat av støy, eller særegenheter ved akkurat disse datasettene. For enkelhets skyld, og for å unngå overtilpasning, velger vi heller $k = 11$ i begge tilfellene.

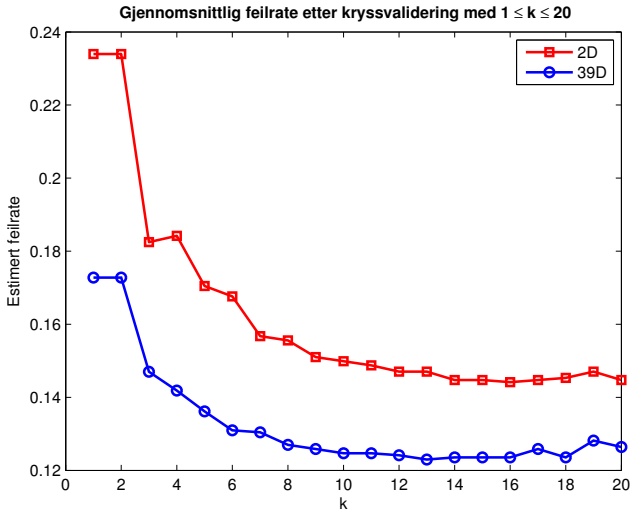
For å gjennomføre kryssvalidering kombinerer vi først sett nr. 1 (1568 observasjoner) og sett nr. 3 (180 observasjoner) til et større testsett med 1748 observasjoner. Vi deler så dette kombinerte settet opp i $K = 5$ deler, fire deler med 350 observasjoner, og én del med 348 observasjoner. Resultatet av kryssvalideringen er oppsummert i tabell 2, og vist i figur 6.

Tabell 2 – Resultat av kryssvalidering med kombinerte data fra sett nr. 1 og 3 delt inn i $K = 5$ deler. Estimert feilrate for 2- og 39-dimensjonale data (jf. tabell 1). Estimatene er regnet ut med `kryssvalider.m`.

k	Feilrate (2D)	Feilrate (39D)
1	0.2340	0.1728
2	0.2340	0.1728
3	0.1825	0.1470
4	0.1842	0.1419
5	0.1705	0.1361
6	0.1676	0.1310
7	0.1567	0.1304
8	0.1556	0.1270
9	0.1510	0.1258
10	0.1499	0.1247
11	0.1487	0.1247
12	0.1470	0.1241
13	0.1470	0.1230
14	0.1447	0.1236
15	0.1447	0.1236
16	0.1442	0.1236
17	0.1447	0.1259
18	0.1453	0.1236
19	0.1470	0.1281
20	0.1447	0.1264

Sammenliknet med plottet i figur 5, er denne kurven atskillig glattere. Kryssvalideringen gir oss estimater som er mindre preget av det enkelte datasettet, slik at det er enklere å unnga overtilpasning. $k = 11$ ser fortsatt ut til å være et godt valg.

Vi vil nå bruke testsettet, altså sett nr. 2, til å finne et endelig estimat for feilraten når algoritmen brukes på andre, uavhengige data fra samme populasjon som treningsdataene. Vi har to valg av treningsdata. Vi kan enten bruke kun sett nr. 1, som er det naturlige



Figur 6 – Feilratene i tabell 2 plottet mot k. Feilratene som estimeres er de samme som i figur 5. Figuren er laget med `kryssvalider.m`.

valget når vi ikke bruker kryssvalidering, eller vi kan bruke kombinasjonen av sett nr. 1 og 3. Vi prøver begge deler, og får resultatene i tabell 3.

Tabell 3 – Estimerte feilrater for to kombinasjoner av test- og trenintgssett, med $k = 11$. Estimatene er beregnet med `testk.m`.

Treningssett	Testsett	Feilrate (2D)	Feilrate (39D)
1	2	0.1643	0.1127
1 og 3	2	0.1455	0.1174

Normalisering

MFCC-vektorene vi har brukt til nå har ikke vært normaliserte. Å normalisere dem vil si å beregne empirisk forventningsverdi og standardavvik for hver komponent, trekke fra forventningsverdien og dele på standardavviket. Hvis $\mathbf{x}_i = (x_{1i}, \ldots, x_{mi})$, $i = 1, \ldots, n$ er MFCC-vektorer som ikke er normalisert, så vil

$$\mathbf{z}_i = \left(\frac{x_{i1} - \bar{x}_1}{s_1}, \ldots, \frac{x_{im} - \bar{x}_m}{s_m} \right), \qquad i = 1, \ldots, n$$

med

$$\bar{x}_j = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \quad \text{og} \quad s_j = \frac{1}{n-1} \sum_{i=1}^{n} (x_{ij} - \bar{x}_j)^2, \qquad j = 1, \ldots, m$$

være normaliserte vektorer for de samme dataene. Vi ønsker å sjekke om normalisering har noen innflytelse på feilraten. Vi beregner derfor estimater for feilraten med data som vi normaliserer i forhold til treningssettet, altså sett nr. 1. Vi bruker fortsatt $k = 11$. Resultatene er vist i tabell 4.

Vi vil teste om det er noen signifikant forskjell mellom feilratene med og uten normalisering. Vi betegner feilraten uten normalisering med p, og feilraten med normalisering med

Tabell 4 – Estimerte feilrater med normaliserte data, for to kombinasjoner av test- og trenintgssett, med $k = 11$. Estimatene er beregnet med `testk.m`.

Treningssett	Testsett	Feilrate (2D)	Feilrate (39D)
1	2	0.1408	0.1362
1 og 3	2	0.1502	0.1268

p'. Fra tabell 3 og 4 har vi fire par $(\hat{p}, \hat{p}')$ av estimater. Hvis vi betrakter antall feilaktige prediksjoner som realiseringer av binomialfordelte variable med forventningsverdi $n_{\text{test}} \cdot p'$ og $n_{\text{test}} \cdot p$, kan vi bruke at ifølge kap. 9.11 i W.M.M.Y. er et tilnærmet $100(1-\alpha)\%$-konfidensinetrvall for differansen $p' - p$ gitt ved

$$\left(\hat{p}' - \hat{p}\right) - z_{1-\alpha/2}\sqrt{\frac{\hat{p}'(1-\hat{p}') + \hat{p}(1-\hat{p})}{n_{\text{test}}}} \leq p' - p \leq \left(\hat{p}' - \hat{p}\right) + z_{1-\alpha/2}\sqrt{\frac{\hat{p}'(1-\hat{p}') + \hat{p}(1-\hat{p})}{n_{\text{test}}}}$$

hvor $z_{1-\alpha/2}$ er $100(1-\alpha/2)\%$-persentilen til standard normalfordelingen. Siden vi bruker sett nr. 2 som testsett for alle estimatene, er n_{test} hele tiden lik antall MFCC-vektorer i sett nr. 2, altså 213. Vi velger $\alpha = 0.05$ og regner ut 95 prosent-konfidensintervaller for de fire differansene. Intervallene står oppført i tabell 5.

Tabell 5 – 95%-konfidensintervaller for forskjellene $p' - p$ med de to ulike valgene av treningssett, og for 2- og 39 dimensjonale MFCC-vektorer. Grensene for intervallene er regnet ut med `testk.m`.

Treningssett	Testsett	Konfidensintervall (2D)	Konfidensintervall (39D)
1	2	$(-0.0917, 0.0448)$	$(-0.0392, 0.0861)$
1 og 3	2	$(-0.0627, 0.0721)$	$(-0.0528, 0.0716)$

Alle konfidensintervallene inneholder null. På signifikansnivå $\alpha = 0.05$ har vi altså ikke grunnlag for å konkludere med at normalisering gir noen forskjell i feilrate, hverken i positiv eller negativ retning.

Klassifisering av fonemer

Vi tar nå for oss problemet med å identifisere hvert fonem som ett av 98 mulige. Sammenliknet med stemt/ustemt-problemet har vi altså en 49-dobling av antall klasser. Det gjør dette til et langt vanskeligere problem, og vi kan ikke forvente like lave feilrater som før. Fremgangsmåten for å finne den beste verdien av k er likevel den samme; først kryssvalidering med trenings- og utviklingssettene, deretter estimering av feilrate med testsettet. Vi bruker de samme MFCC-vektorene som før, og bytter kun ut labels-vektorene. Siden de todimensjonale vektorene gir svært høy feilrate (ca. 95 prosent) velger vi å konsentrere oss om de 39-dimensjonale MFCC-vektorene.

Vi tester ytelsen til algoritmen for ulike verdier av k mellom 1 og 20 på to forskjellige måter; validering med sett nr. 1 som treningssett og sett nr. 3 som utviklingsset, og kryssvalidering med kombinasjonen av sett nr. 1 og 3 som treningsssett, delt inn i fem like store deler, som før. Resultatet av valideringen er ført i tabell 6 og plottet i figur 7.

I figur 7 gjenkjenner vi tendensen fra figur 5 og 6, nemlig at kryssvalidering gir en glattere kurve med mindre fluktuasjoner. Også her ser $k = 11$ ut til å være et godt valg. Vi fikserer

Tabell 6 – Resultat av validering av k nærmeste naboer-algoritmen anvendt på fonemklassifiseringsproblemet. Kun 39-dimensjonale MFCC-vektorer er brukt. For kryssvalideringen (gjennomsnittlig feilrate) er kombinasjonen av sett nr. 1 og 3 delt inn i fem deler. For de øvrige estimatene er sett nr. 1 brukt som treningssett, og sett nr. 3 som utviklingssett. Estimatene er beregnet med `validerFonem.m`.

k	Feilrate	Gjennomsnittlig feilrate
1	0.8611	0.8369
2	0.8611	0.8369
3	0.8444	0.8283
4	0.8444	0.8175
5	0.8000	0.8112
6	0.7833	0.8049
7	0.7944	0.8026
8	0.7889	0.7957
9	0.7889	0.7986
10	0.7889	0.7969
11	0.7722	0.7997
12	0.7833	0.7963
13	0.7833	0.7917
14	0.7833	0.7997
15	0.7667	0.7957
16	0.7778	0.7986
17	0.8111	0.8014
18	0.8111	0.8020
19	0.8278	0.8077
20	0.8222	0.8049

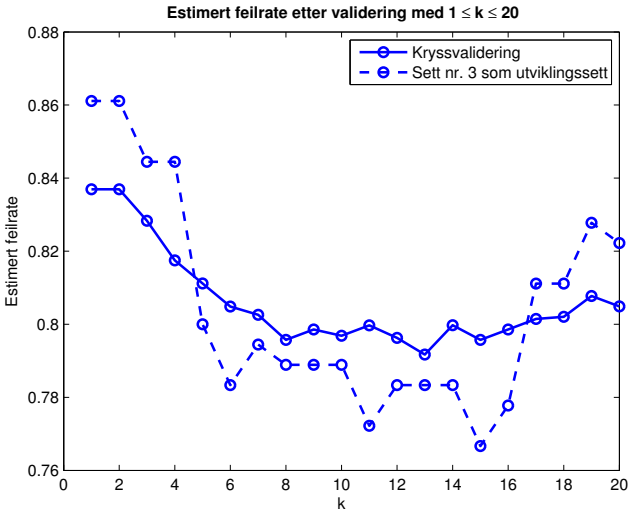
derfor k, og fortsetter ved å teste algoritmen med sett nr. 2 som testsett, både med og uten normalisering av MFCC-vektorene. Resultatene av testen er oppsummert i tabell 7.

Tabell 7 – Estimerte feilrater uten ($\hat{p}$) og med ($\hat{p}'$) normaliserte MFCC-vektorer. Dataene er normalisert i forhold til treningssettet, sett nr. 1. Estimatene er beregnet med `testPhone.m`.

Treningssett	Testsett	$\hat{p}$	$\hat{p}'$
1	2	0.7887	0.8357
1 og 3	2	0.7981	0.8967

Til slutt finner vi 95%-konfidensintervaller for differansene $p' - p$ for begge valgene av test- og treningssett. Disse er vist i tabell 8.

Intervallet vi får når kun sett nr. 1 brukes som treningssett inneholder null. I dette tilfelle finner vi altså på signifikansnivå $\alpha = 0.05$ ingen signifikant effekt av å normalisere dataene. Intervallet som svarer til å bruke kombinasjonen av sett nr. 1 og 3 som treningsdata ligger over null. Her kan vi dermed slå fast, med 95% konfidens, at normaliseringen gir en høyere feilrate. Vi merker oss at disse konfidensintervallene gjelder normalisering av dataene i forhold til set nr. 1, dvs. empirisk forventningsverdi og standardavvik er beregnet utfra



Figur 7 – Feilratene i tabell 6 plottet mot k. Gjennomsnittelig feilrate fra kryssvalidering er markert med heltrukken linje, feilrate fra validering med sett nr. 1 som treningssett og sett nr. 3 som utviklingssett er markert med stipla linje. Figuren er laget med `validerFonem.m`.

Tabell 8 – 95%-konfidensintervaller for differansene $p' - p$ mellom feilrate med og uten normalisering av data, beregnet med `testPhone.m`.

Treningssett	Testsett	Konfidensintervall
1	2	$(-0.0271, 0.1210)$
1 og 3	2	$(0.0309, 0.1662)$

dataene i sett nr. 1. Om vi i stedet brukte kombinasjonen av sett nr. 1 og 3 til dette formålet, ville vi kanskje få et annet resultat.