Klassifikationsmetoder: Prototypemetoder og Nearest Neighbor

Peter N. Bakker

08-06-2023

Klassifikationsmetoder

- Klassifikationsmetoder: på baggrund af et datasæt med $(x_1, g_1), \ldots, (x_N, g_N)$ øvsker for en ny værdi af x at prædiktere den tilhørende g-værdi.
- Vi vil se på: K-means clustering, LVQ, Nearest Neighbor og DANN.
- Disse modeller er **ikke-parametriske metoder**. Der estimeres altså ikke parametre værdier, som ville kunne bruges til at sige noget generelt om datastrukturen.

K/M-means Clustering

- M-means clustering er en prototypemetode, der opdeler data i M forskellige klynger.
- Estimering af klynger:
  - Initialisering af M centroider i træningsdata.
  - Gentagende opdatering af centroiderne baseret på afstanden mellem datapunkter og centroider.

$$W(C) = \sum_{m=1}^{M} N_m \sum_{i:C(i)=m} \|\mathbf{x}_i - \bar{x}_m\|^2$$

    hvor N_m er antallet af observationer, der er havnet i klynge m ($\bar{x}_m$ er gennemsnittet).
  - Tildeling af datapunkter til den nærmeste centroid.
- Formel for estimering af centroiderne:

$$\bar{x}_m = \frac{1}{N_m} \sum_{i:C(i)=m} \mathbf{x}_i$$

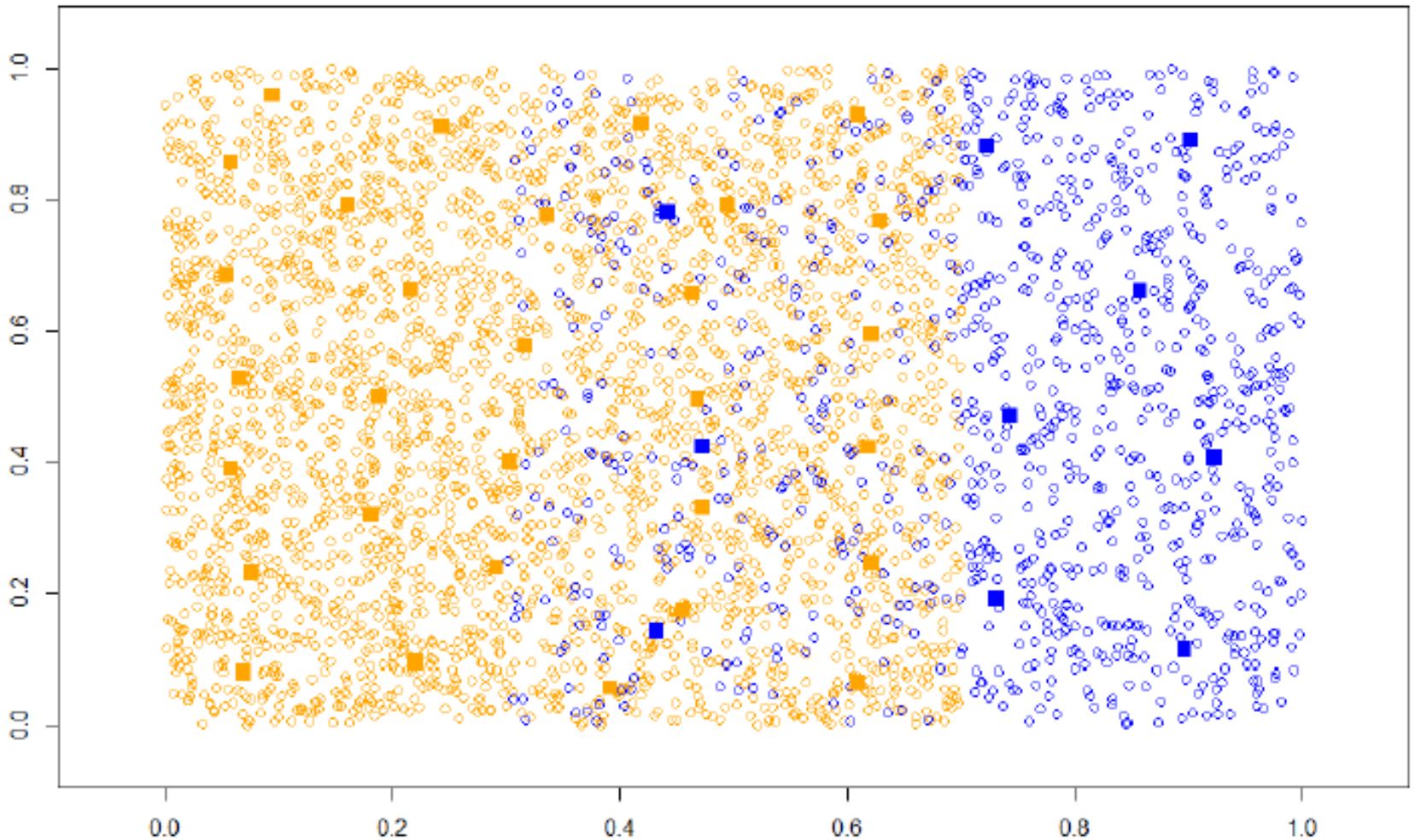
hvor $\bar{x}_m$ er centroiden for klyngen m, N_m er antallet af datapunkter i klyngen og $\mathbf{x}_i$ er det i'te datapunkt.

M-means clustering til klassifikation

1. Opdel datasættet i K forskellige klasser ved hjælp af M-means clustering, hvor hver klasse har M klynger.
2. Udtræk de $M \cdot K$ klyngecentre, der fungerer som prototyper, fra de respektive klynger.
3. Tildel klassifikation til prototyperne baseret på den oprindelige klasse, de blev oprettet fra.
4. For at klassificere en ny inputvektor $\mathbf{x}$, find den nærmeste prototype til $\mathbf{x}$ baseret på en afstandsmåling.
5. Tildel inputvektoren $\mathbf{x}$ til den tilsvarende klasse for den nærmeste prototype.

Ved at bruge M-means clustering til klassifikation kan vi udnytte prototypernes repræsentative karakteristika for at tildele nye datapunkter til de korrekte klasser.

K/M-means Clustering Illustration



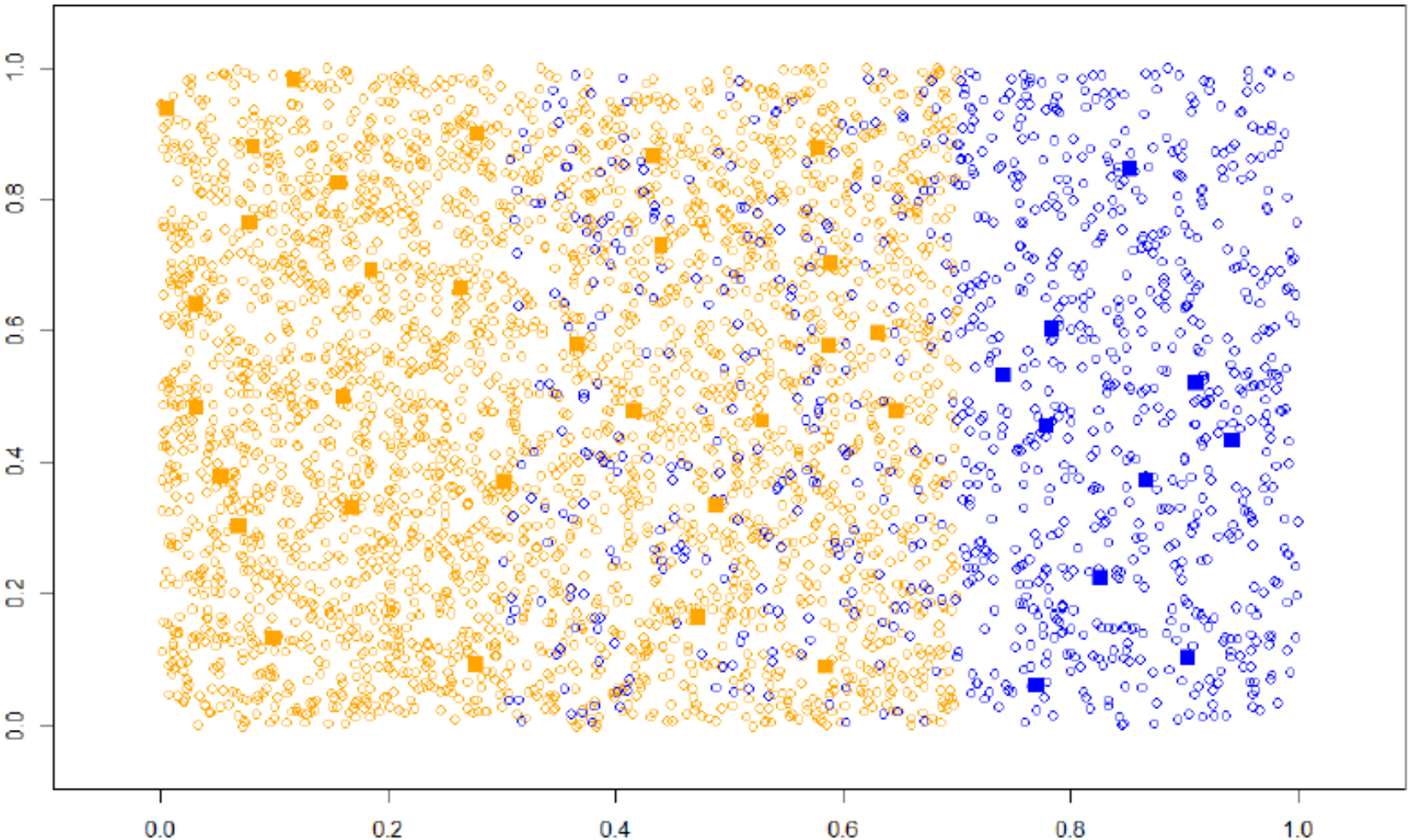
LVQ (Learning Vector Quantization)

En tilsvarende metode kommer ved at følge denne algoritme:

1. Vælg M startsprototyper for hver af de K klasser: Dvs. punkter i $\mathbb{R}^p$: $p_1(k), \ldots, p_M(k)$ for $k = 1, \ldots, K$.
2. Træk et observationspar $(\mathbf{x}_i, g_i)$ tilfældigt (med tilbagelægning) fra datasættet. Lad (m, k) være index for den nærmeste $\mathbf{x}_i$ prototype, dvs. $p_m(k)$.
3. Hvis $g_i = k$: Flyt prototypen $p_m(k)$ i retning mod $\mathbf{x}_i$ ved $p_m(k) \leftarrow p_m(k) + \epsilon(\mathbf{x}_i - p_m(k))$.
4. Hvis $g_i \neq k$: Flyt prototypen $p_m(k)$ væk fra $\mathbf{x}_i$ ved $p_m(k) \leftarrow p_m(k) - \epsilon(\mathbf{x}_i - p_m(k))$.
5. Gentag punkt 2 igen og igen, men hver gang gøres ϵ mindre ved $\epsilon = \frac{\alpha}{\text{nummer på iterationen}}$.

Ved denne algoritme ahænger prototyperne fra de forskellige klasser hinanden. LVQ er også bedre til at håndtere outliers og hvis klasserne er "viklet" sammen. Nogen vil mene at det er en forbedret version ift. M-means, dog er den mere kompleks at implementere. Metoden minimerer heller ikke noget optimeringsproblem.

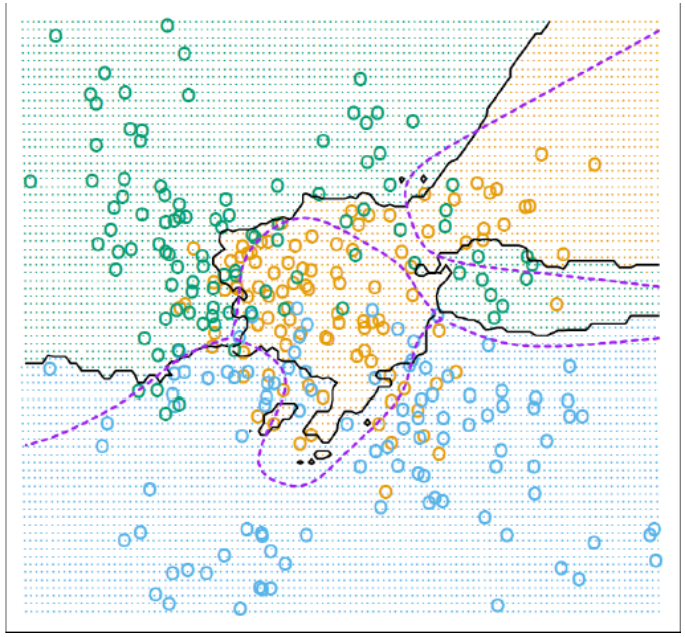
LVQ illustration

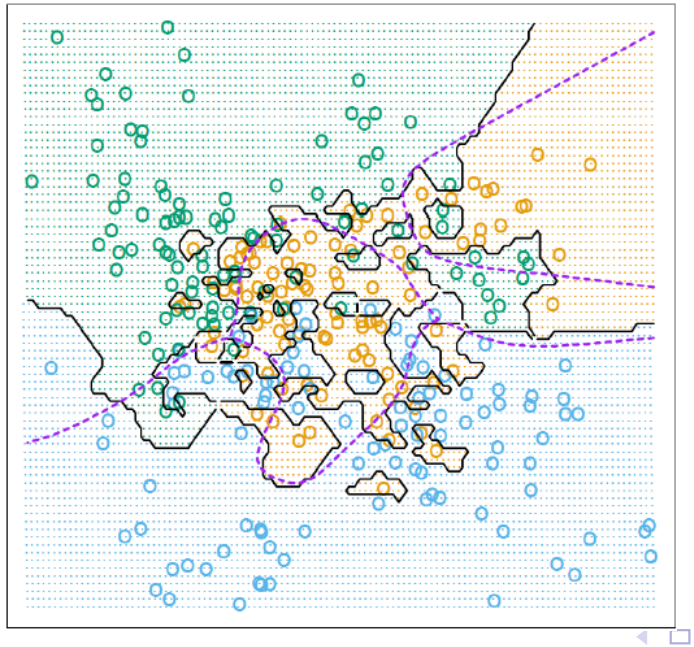


k/m-Nearest Neighbor

- Nearest Neighbor er en enkel klassifikationsmetode baseret på afstandsmålinger.
- Klassifikation af nye eksempler ved at finde den nærmeste nabo i træningsdataene. Vi kigger altså ikke længere hvor langt der er til den nærmeste prototype, men istedet bare på de nærmeste observationer.
- Vi definerer $N_m(x)$ som **naboområdet** omkring x af de m nærmeste x_i-værdier i træningsdatasættet.
- Vi definerer $N_m(x.k)$ som antallet af observationer tilhørende hver klasse i naboområdet.
- Klassifikationen bestemmes af den klasse der ses flest af i naboområdet. Dvs. vi prædikter g-værdien hørende til x som $\hat{G}(x) = argmax_{k \in 1,\ldots,K} N_m(x,k)$. Kan også estimeres som **klasse-sandsynligheder** som $\hat{P}(G = k|X = x) = \frac{N_m(x,k)}{m}$, hvor man kan skrue på treshold-værdier.
- m bestemmer bias/varians tradeoff. Nearest er meget beregningstung ift. prototyper.

Nearest neighbor illustration for 15 og 1 neighbors



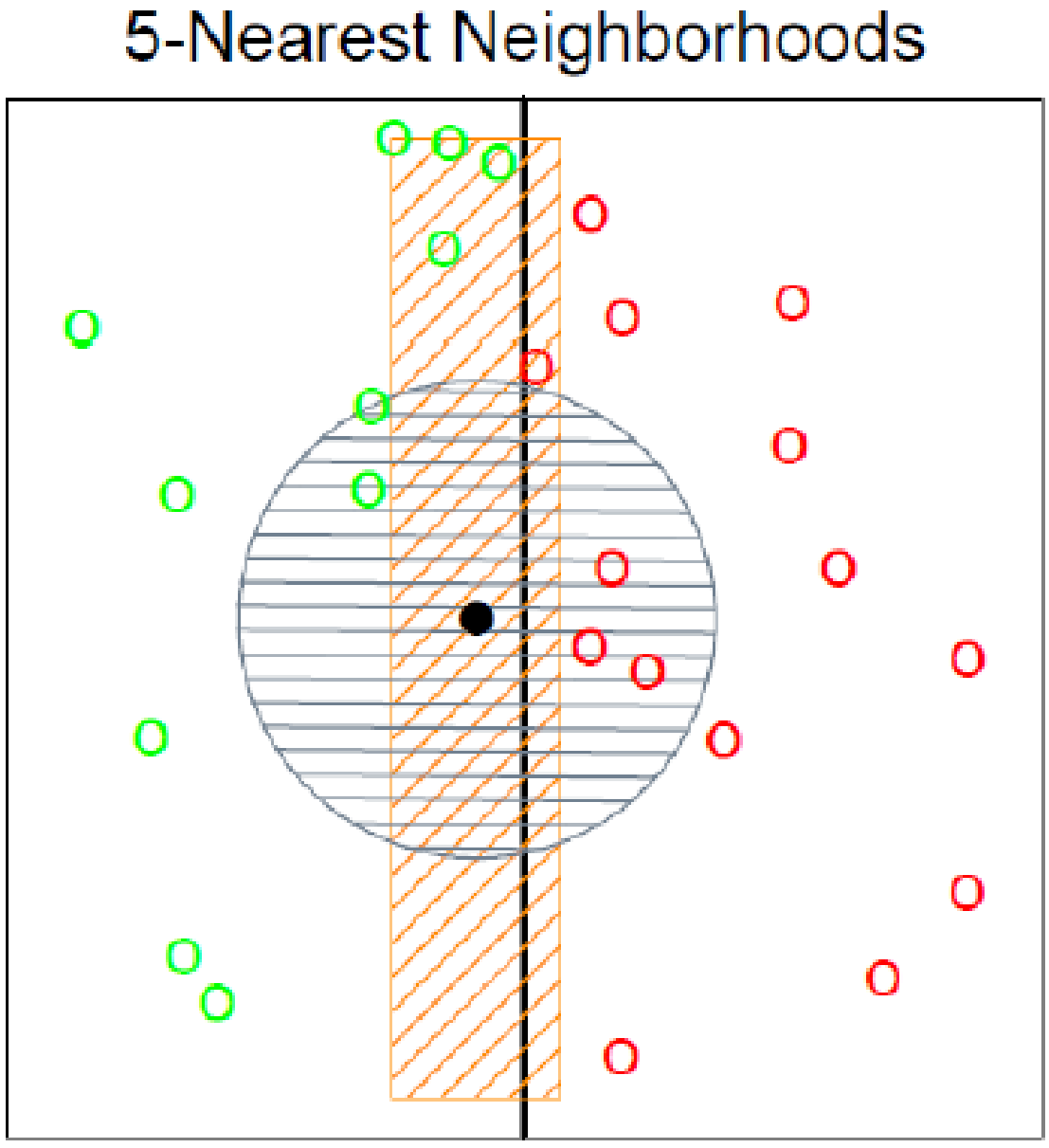


DANN (Discriminant Adaptive Nearest Neighbor)

Dimensionsudfordringer ved Nearest Neighbor:

- Nearest Neighbor-metoden kan have udfordringer i højdimensionelle rum (p er stort).
- Afstanden mellem datapunkter kan blive mindre meningsfuld i høje dimensioner. I nearest neighbor er afstanden kugle formet.
- Dette kan føre til tab af præcision og øget følsomhed over for støj og irrelevante dimensioner. Radiusen skal være meget stor.

Illustration af dimensionsproblemer ved nearest neighbor



DANN - Løsning på dimensionsudfordringer

DANN (Discriminant Adaptive Nearest Neighbor):

- En variant af nearest neighbor-metoden, der håndterer dimensionsudfordringer.
- Fokuserer på at vælge og tilpasse de mest relevante dimensioner i beslutningsrummet.
- Ved at vægte dimensionerne baseret på deres diskriminante evne, kan DANN forbedre præcisionen og reducere følsomheden over for støj.

DANN Algoritme

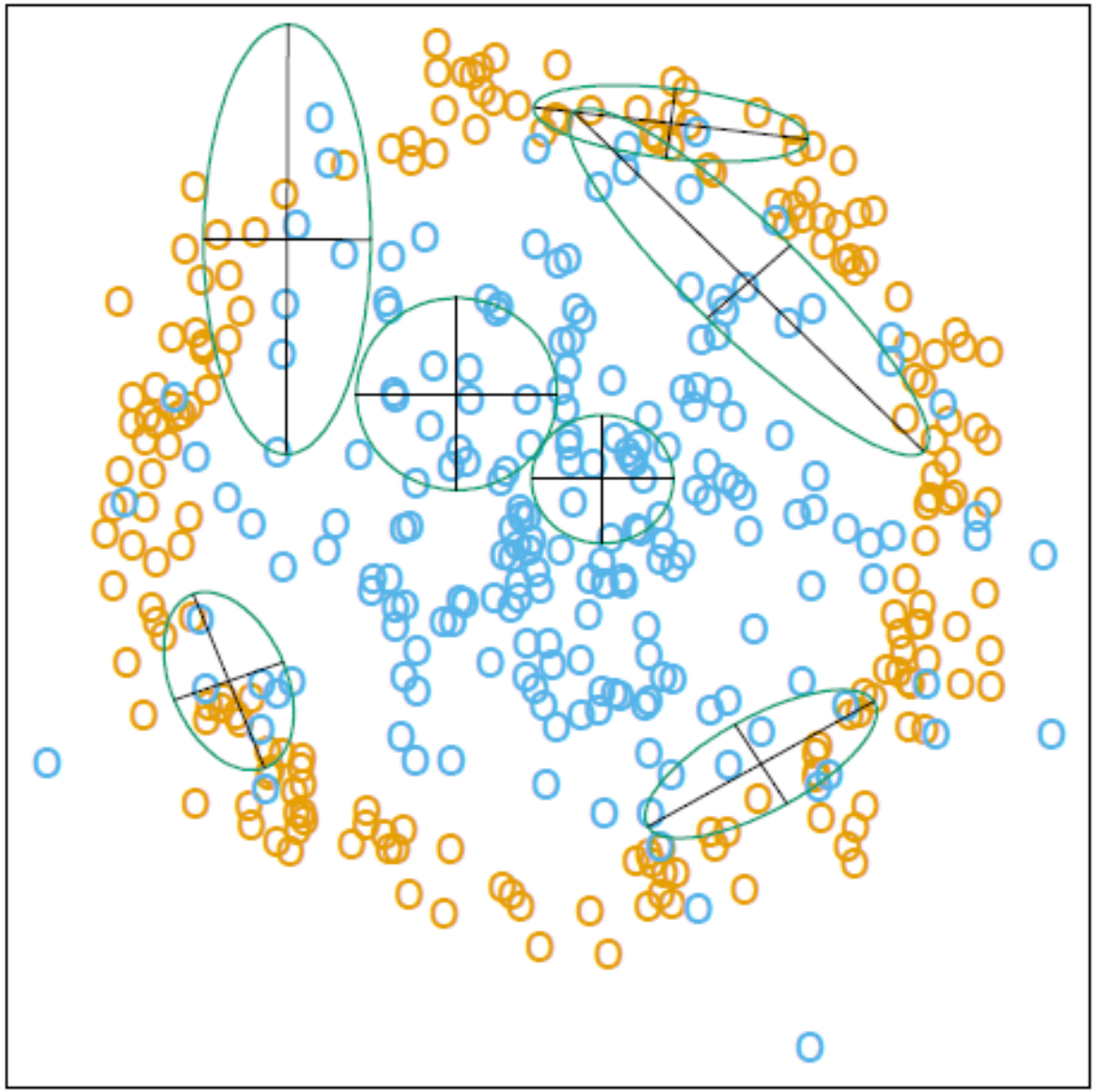
DANN Algoritme:

1. Vælg et passende antal dimensioner til brug i DANN.
2. Anvend dimensionel vægtning til de valgte dimensioner baseret på deres diskriminante evne.
3. Afstandsmålet reduceres ved $D(x, x') = (x - x')^T \Sigma_\epsilon (x - x')$, hvor Σ_ϵ er en matrix udregnet fra en punktsky på typisk 50 punkter, og ϵ er en tuning-parameter, der afgør hvor runde naboområderne bliver.
4. Foretag klassifikation af nye datapunkter ved at tilpasse nearest neighbor-metoden til de vægtede dimensioner.

Fordele ved DANN:

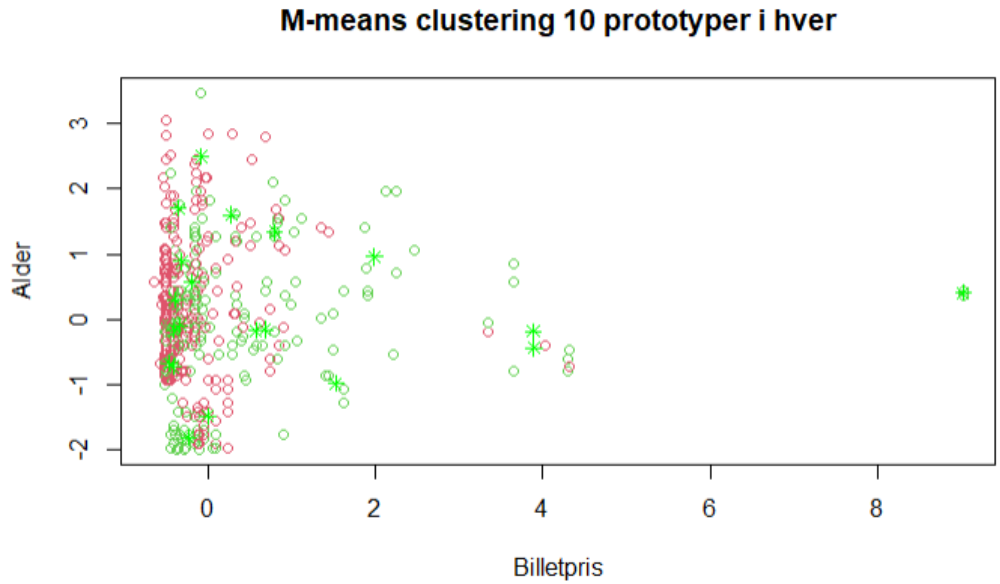
- Håndterer dimensionsudfordringer og reducerer følsomheden over for irrelevante dimensioner. Formen bliver mere eliptisk.
- Forbedrer præcisionen og robustheden af nearest neighbor-metoden i højdimensionelle rum.

Illustration DANN



Dataeksempel Titanic

- Vi vil prædiktere overlevelse ved at fokusere på Alder og Billetpris.
- Jeg benytter m-means clustering med 10 prototyper i hver klasse. Jeg deler datasættet op med 500 i træningsdata og 212 i test.
- Jeg får en fejlrate på 41,98 pct.
- Jeg gør det samme, men med LVQ istedet. Som bør være bedre til at håndtere outliers. Her får jeg en fejlrate på 33,96 pct.



Opsummering af metoder

- Generelt er alle metoderne relativt beregningstunge ved store datasæt.
- Nearest-neighbor-metoderne kan være mere beregningstunge ved store datasæt.
- Hvis antallet af dimensioner (p) er stort, kan alle metoderne blive ret ustabile, da afstandene bliver store til både naboer og prototyper.
- DANN er den mest beregningstunge metode, men klarer til gengæld højere dimensioner (større p) bedre.