Fast low-rank metric learning

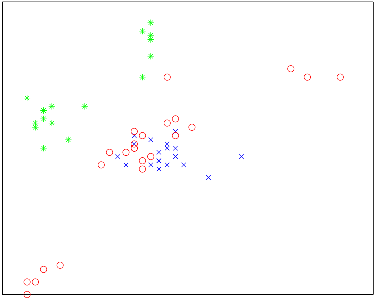
Dan-Theodor Oneaţă

July 25, 2011

k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

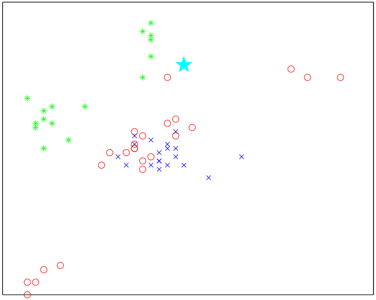
$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$



k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

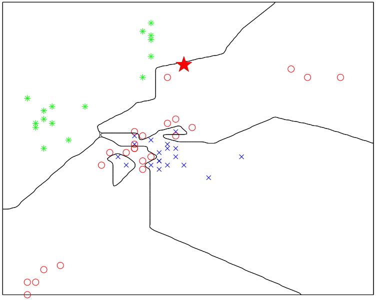
$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$



k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$

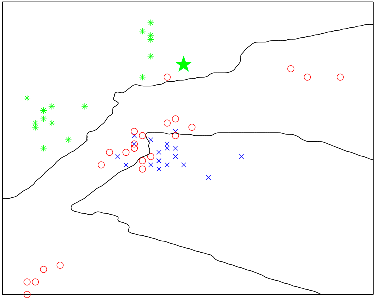


$k = 1$

k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$

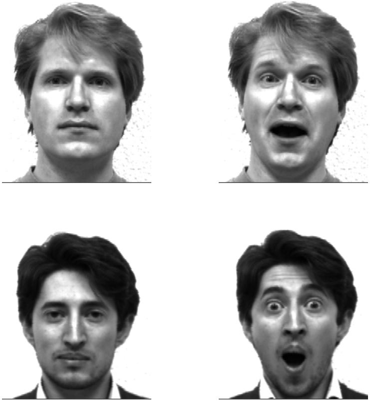


$k = 7$

k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

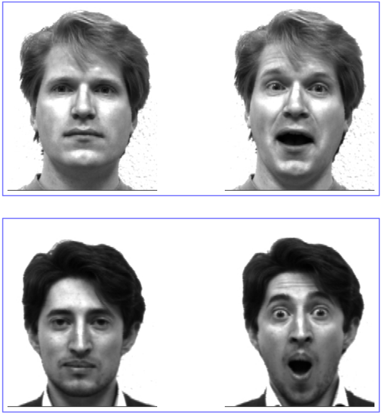
$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$



k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$



Face recognition

k nearest neighbours

- Simple, yet powerful classifier.
- Eucildean distance:

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}}(\mathbf{x}_i - \mathbf{x}_j)}$$

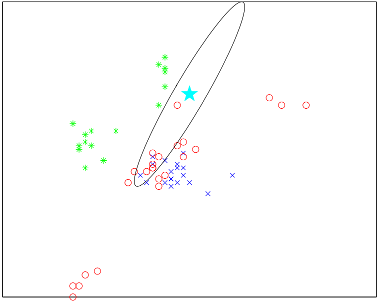
Expression recognition

Neighbourhood component analysis

- Learns a Mahalanobis metric

$$d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{S} (\mathbf{x}_i - \mathbf{x}_j)}$$

- Equivalent to a linear transformation: $d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathbf{I}}(\mathbf{A}\mathbf{x}_i, \mathbf{A}\mathbf{x}_j)$

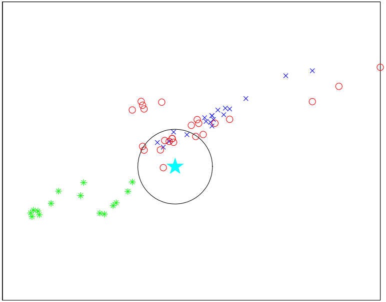


Neighbourhood component analysis

- Learns a Mahalanobis metric

$$d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^{\mathrm{T}} \mathbf{S} (\mathbf{x}_i - \mathbf{x}_j)}$$

- Equivalent to a linear transformation:
  $d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_j) = d_{\mathbf{I}}(\mathbf{A}\mathbf{x}_i, \mathbf{A}\mathbf{x}_j)$



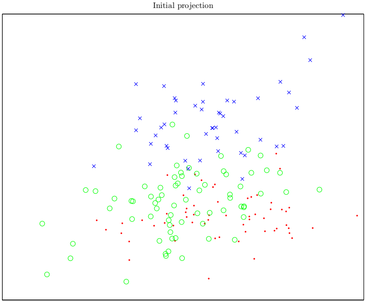
Neighbourhood component analysis

1. Find $\mathbf{S}$ that maximizes leave-one-out cross-validation score.
2. Soft version:
   $p(\mathbf{x}_i \in \text{class } c) = \frac{\sum_{j \in c} \exp\{-d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_j)\}}{\sum_k \exp\{-d_{\mathbf{S}}(\mathbf{x}_i, \mathbf{x}_k)\}}$

   Maximize $f(\mathbf{S}) = \sum_i p(\mathbf{x}_i \in \text{true class of } \mathbf{x}_i)$.

Optimizing $f(\mathbf{S})$

- Use $\nabla_{\mathbf{S}} f(\mathbf{S})$ for an optimization algorithm: *e.g.*, gradient ascent, conjugate gradients.
- How to initialise? Use random $\mathbf{S}$ or most discriminative projections given by PCA, LDA or logistic regression.

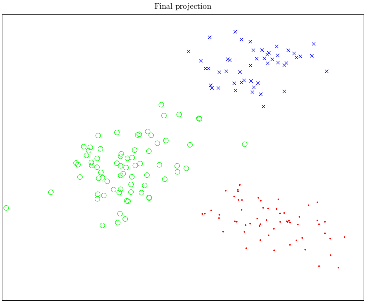


`A=randn(d,D)`

Data set: `wine`.

Optimizing $f(\mathbf{S})$

- Use $\nabla_{\mathbf{S}} f(\mathbf{S})$ for an optimization algorithm: *e.g.*, gradient ascent, conjugate gradients.
- How to initialise? Use random $\mathbf{S}$ or most discriminative projections given by PCA, LDA or logistic regression.

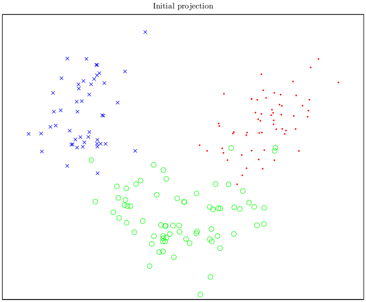


`A=minimize('nca',A)`

Data set: `wine`.

Optimizing $f(\mathbf{S})$

- Use $\nabla_{\mathbf{S}} f(\mathbf{S})$ for an optimization algorithm: *e.g.*, gradient ascent, conjugate gradients.
- How to initialise? Use random $\mathbf{S}$ or most discriminative projections given by PCA, LDA or logistic regression.

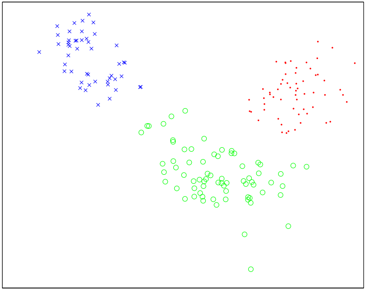


`A=eig(X*X'/N)`

Data set: `wine`.

Optimizing $f(\mathbf{S})$

- Use $\nabla_{\mathbf{S}} f(\mathbf{S})$ for an optimization algorithm: *e.g.*, gradient ascent, conjugate gradients.
- How to initialise? Use random $\mathbf{S}$ or most discriminative projections given by PCA, LDA or logistic regression.



`A=minimize('nca',A)`

Data set: `wine`.

Speeding the computations

1. Sub-sample the data set.
2. Use mini-batches:
    - Choose them randomly
    - Use cheap clustering method.

Speeding the computations

1. Sub-sample the data set.
2. Use mini-batches:
   - Choose them randomly
   - Use cheap clustering method.

Approximate computations

Exact computations

Future work

Conclusions