INF 5300
Introduction
Feature selection and principal component analysis

Lecturers:

- Asbjørn Berge
- Anne Schistad Solberg

Contact information

- Asbjørn Berge
  - On IFI wednesdays, room 4457
    Email: asbjorn.berge@sintef.no
- Anne Schistad Solberg
  - Room 4458
  - anne@ifi.uio.no , 22852435

Course hightlights

- Two main parts:
  - Computer vision
  - Pattern recognition
- One mandatory exercise
  - Individual themes
  - Can be linked to your master topic if requested
  - Deadline can be fitted to your schedule if it fits with the course schedule.
- New this year: lab exercise sessions
- Oral exam if less than 10 approx. students

Lecture plan

- 23.01: Feature selection and transforms (Anne)
- 30.01: Fisher's linear discriminant (Anne)
- 06.02: Lab exercise on feature transforms (Anne)
- 13.02: Classification and clustering in video (Asbjørn)
- 20.02: Randomized algorithms (Asbjørn)
  - RANSAC, forest algorithms, overcomplete feature sets.
- 27.02: Lab on video and randomized algorithms. (Asbjørn)
- 06.03: Motion cues from video (Asbjørn)
  - Following invariant features and calculating flow
- 13.03: Classification with Support Vector Machines (Anne)
- 20.03: Lab on Support Vector Machines (Anne)

Lecture plan cont.

- 03.04: Deformable contours and snakes (Anne)
- 10.04: More on snakes (Anne)
- 17.04: Geometry in video (Asbjørn)
  - Stereo and structured light
- 24.04: Lab on geometry/motion in video (Asbjørn)
- 08.05: Statistical tracking (Asbjørn)
  - Particle filters, predictive tracking
- 15.05: Lab or repetition. (Asbjørn)
- 22.05: Repetition

Curriculum for today

- The lecture is based on the following sections from ”Pattern Recognition” by S. Theodoridis and K. Koutroumbas:
- 5.1
- 5.2.2 Feature normalization
- 5.5.3 Scatter matrices
- 5.6 Feature subset selection
- 5.7 Fisher’s linear discriminant function (next lecture)
- 6.1-6.3 Principal component analysis
- See http://www.uio.no/studier/emner/matnat/ifi/INF5300/v12/undervisningsmateriale/inf5300_featsel_chap5.pdf
- http://www.uio.no/studier/emner/matnat/ifi/INF5300/v12/undervisningsmateriale/chap6PCAogAppendixB.pdf

Reminder - Basic classification principles

Classification task:

- Classify object $x = \{x_1,..., x_n\}$ to one of the R classes $\omega_1,...\omega_R$
- *Decision rule* $d(\mathbf{x})=\omega_r$ divides the feature space into R disjoint subsets K_r, r=1,...R.
- The borders between subsets K_r, r=1,...R are defined by R scalar *discrimination functions* $g_1(\mathbf{x}),....g_R(\mathbf{x})$
- The discrimination functions must satisfy:
  $g_r(\mathbf{x}) \geq g_s(\mathbf{x})$, $s \neq r$, for all $\mathbf{x} \in K_r$
- Discrimination hypersurfaces are thus defined by
  $g_r(\mathbf{x})-g_s(\mathbf{x})=0$
- The pattern x will be classified to the class whose discrimination function gives a maximum:
  $d(\mathbf{x})=\omega_r \Leftrightarrow g_r(\mathbf{x}) = \max_{s=1,...R} g_s(\mathbf{x})$

Reminder - Bayesian classification

- Prior probabilities $P(\omega_r)$ for each class
- Bayes classification rule: classify a pattern **x** to the class with the highest posterior probability $P(\omega_r|\mathbf{x})$
  $$P(\omega_r|\mathbf{x}) = \max_{s=1,...R} P(\omega_s|\mathbf{x})$$
- $P(\omega_s|\mathbf{x})$ is computed using Bayes formula
  $$P(\omega_s \mid x) = \frac{p(x \mid \omega_s)P(\omega_s)}{p(x)}$$
  $$p(x) = \sum_{s=1}^{R} p(x \mid \omega_s)P(\omega_s)$$
- **x** is a d-dimensional feature vector
- ω_s is one of K classes
- $p(\mathbf{x}|\ \omega_s)$ is the class-conditional probability density for a given class.

Reminder - Classification with Gaussian distributions

- Probability distribution for n-dimensional Gaussian vector:

$$p(x \mid \omega_s) = \frac{1}{(2\pi)^{d/2}|\Sigma_s|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu_s)^t \Sigma_s^{-1}(x-\mu_s)\right]$$

$$\hat{\mu}_s = \frac{1}{M_s}\sum_{m=1}^{M_s} x_m,$$

$$\hat{\Sigma}_s = \frac{1}{M_s}\sum_{m=1}^{M_s}(x_m-\hat{\mu}_s)(x_m-\hat{\mu}_s)^t$$

where the sum is over all training samples belonging to class s

- μ_s and Σ_s are not known, but they are estimated from M training samples as the Maximum Likelihood estimates

If we have many features – The curse of dimensionality

- Assume we have S classes and a d-dimensional feature vector.
- With a fully multivariate Gaussian model, we must estimate S different mean vectors and S different covariance matrices from training samples.
  - $\hat{\mu}_s$ has d elements
  - $\hat{\Sigma}_s$ has d(d-1)/2 elements
- Assume that we have M_s training samples from each class
- Given M_s, there is a maximum of the achieved classification performance for a certain value of d (increasing d beyond this limit will lead to worse performance after a certain).
- Adding more features is not always a good idea!
- If we have limited training data, we can use diagonal covariance matrices or regularization.

How do we beat the ”curse of dimensionality”?

- Use regularized estimates for the Gaussian case
  - Use diagonal covariance matrices
  - Apply regularized covariance estimation
- Generate few, but informative features
  - Careful feature design given the application
- Reducing the dimensionality
  - Feature selection
  - Feature transforms

Regularized covariance matrix estimation

- Let the covariance matrix be a weighted combination of a class-specific covariance matrix Σ_k and a common covariance matrix Σ:

$$\Sigma_k(\alpha) = \frac{(1-\alpha)n_k\Sigma_k + \alpha n\Sigma}{(1-\alpha)n_k + \alpha n}$$

  where $0 \leq \alpha \leq 1$ must be determined, and n_k and n is the number of training samples for class k and overall.
- Alternatively:

$$\Sigma_k(\beta) = (1-\beta)\Sigma_k + \beta I$$

  where the parameter $0 \leq \beta \leq 1$ must be determined.
- The effect of these are that we can use a quadratic classifier even if we have little training data/ill-conditioned Σ_k
- We still have to be able to compute Σ_k, but the only the regularized/more robust $\Sigma_k(\alpha)$ or $\Sigma_k(\beta)$ must be inverted.

Feature selection

- Given a large set of N features, how do we select the best subset of m features?
  - How do we select m?
  - Finding the best combination of m features out a N possible is a large optimization problem.
  - Full search is normally not possible.
  - Suboptimal approaches are often used.
  - How many features are needed?
- Alternative: compute lower-dimensional projections of the N-dimensional space
  - PCA
  - Fisher's linear discriminant
  - Projection pursuit and other non-linear approaches

Preprocessing - data normalization

- Features may have different ranges
  - Feature 1 has range $f1_{min}$-$f1_{max}$
  - Feature n has range fn_{min}-fn_{max}
  - This does not reflect their significance in classification performance!
  - Example: minimum distance classifier uses Euclidean distance
    - Features with large absolute values will dominate the classifier

Feature normalization

- Normalize all features to have the same mean and variance.
- Data set with N objects and K features
- Features x_{ik}, i=1...N, k=1,...K

Zero mean, unit variance:

$$\bar{x}_k = \frac{1}{N}\sum_{i=i}^{N} x_{ik}$$

$$\sigma_k^2 = \frac{1}{N-1}\sum_{i=i}^{N}(x_{ik} - \bar{x}_k)^2$$

$$\hat{x}_{ik} = \frac{x_{ik} - \bar{x}_k}{\sigma_k}$$

Softmax (non-linear)

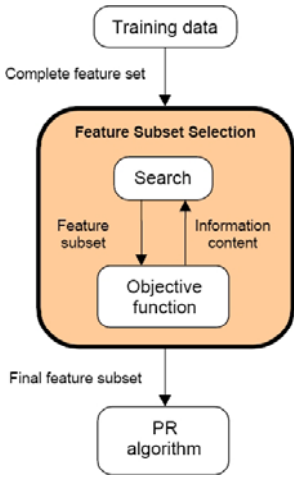
$$y = \frac{x_{ik} - \bar{x}_k}{r\sigma_k}$$

$$\hat{x}_{ik} = \frac{1}{1+\exp(-y)}$$

Remark: normalization may destroy important discrimination information

Feature selection

- How do we find the best subset of m out of n features.
- Search strategy
  - Exhaustive search implies $\binom{n}{m}$ if we fix *m* and 2^n if we need to search all possible *m* as well.
  - Choosing *10* out of *100* will result in 10^{13} queries to *J*
  - Obviously we need to guide the search!
- Objective function (J)
  - "Predict" classifier performance
  - Decides how good a subset if
  - We can use either a distance measure or the actual classification accuracy.



Note that $\binom{m}{l} = \frac{m!}{l!(m-l)!}$

Distance measures to compute the criterion function J

- Between two classes:
  - Distance between the closest two points?
  - Maximum distance between two points?
  - Distance between the class means?
  - Average distance between points in the two classes?
  - Which distance measure?
- Between K classes:
  - How do we generalize to more than two classes?
  - Average distance between the classes?
  - Smallest distance between a pair of classes?

Note: Often performance should be evalued in terms of classification error rate (e.g. on the training set or on a validation set). But this is slower than computing simple distance measures.

Class separability measures

- How do we get an indication of the separability between two classes?
  - Euclidean distance between class means $|\mu_r - \mu_s|$
  - Bhattacharyya distance
    - Can be defined for different distributions
    - For Gaussian data, it is

$$B = \frac{1}{8}(\mu_r - \mu_s)^T \left(\frac{\Sigma_r + \Sigma_s}{2}\right)^{-1} (\mu_r - \mu_s) + \frac{1}{2}\ln\frac{\left|\frac{1}{2}(\Sigma_r + \Sigma_s)\right|}{\sqrt{|\Sigma_r||\Sigma_s|}}$$

  - Mahalanobis distance between two classes:

$$\Delta = (\mu_1 - \mu_2)^T \Sigma^{-1} (\mu_1 - \mu_2)$$
$$\Sigma = N_1\Sigma_1 + N_2\Sigma_2$$

Divergence

- Divergence (see 5.5 in Theodoridis and Koutroumbas) is a measure of distance between probability density functions.
- Mahalanobis distance is a form of divergence measure.
- The Bhattacharrya distance is related to the Chernoff bound for the lowest classification error.
- If two classes have equal variance $\Sigma_1=\Sigma_2$, then the Bhattacharrya distance is proportional to the Mahalanobis distance.

Method 1 - Individual feature selection

- Each feature is treated individually (no correlation/covariance between features is consideren)
- Select a criteria, e.g. a distance measure
- Rank the feature according to the value of the criteria C(k)
- Select the set of features with the best individual criteria value
- Multiclass situations:
  - Average class separability or
  - C(k) = min distance(i,j) - worst case ← Often used
- Advantage with individual selection: computation time
- Disadvantage: no correlation is utilized.

Individual feature selection cont.

- We can also include a simple measure of feature correlation.
- Cross-Correlation between feature i and j: ($|\rho_{ij}| \leq 1$)

$$\rho_{ij} = \frac{\sum_{n=1}^{N} x_{ni} x_{nj}}{\sum_{n=1}^{N} x_{ni}^2 \sum_{n=1}^{N} x_{nj}^2}$$

- Simple algorithm:
  - Select C(k) and compute for all x_k, k=1,...m. Rank in descending order and select the one with best value. Call this x_{i1}.
  - Compute the cross-correlation between x_{i1} and all other features. Choose the feature x_{i2} for which

$$i_2 = \arg\max_j \left\{ \alpha_1 C(j) - \alpha_2 |\rho_{i_1 j}| \right\} \text{for all } j \neq i_1$$

  - Select x_{ik}, k=3,...l so that

$$i_k = \arg\max_j \left\{ \alpha_1 C(j) - \frac{\alpha_2}{k-1} \sum_{r=1}^{k-1} |\rho_{i_r j}| \right\} \text{for all } j \neq i_1$$

Method 2 - Sequential backward selection

- Select l features out of m
- Example: 4 features x_1, x_2, x_3, x_4
- Choose a criterion C and compute it for the vector $[x_1, x_2, x_3, x_4]^T$
- Eliminate one feature at a time by computing $[x_1, x_2, x_3]^T$, $[x_1, x_2, x_4]_T$, $[x_1, x_3, x_4]^T$ and $[x_2, x_3, x_4]^T$
- Select the best combination, say $[x_1, x_2, x_3]^T$.
- From the selected 3-dimensional feature vector eliminate one more feature, and evaluate the criterion for $[x_1, x_2]^T$, $[x_1, x_3]_T$, $[x_2, x_3]^T$ and select the one with the best value.
- Number of combinations searched:
  1+1/2((m+1)m-l(l+1))

Method 3: Sequential forward selection

- Compute the criterion value for each feature. Select the feature with the best value, say x_1.
- Form all possible combinations of features x1 (the winner at the previous step) and a new feature, e.g. $[x_1,x_2]^T$, $[x_1,x_3]^T$, $[x_1,x_4]^T$, etc. Compute the criterion and select the best one, say $[x_1,x_3]^T$.
- Continue with adding a new feature.
- Number of combinations searched: lm-l(l-1)/2.
  - Backwards selection is faster if l is closer to m than to 1.

Method 4: Plus-L Minus-R Selection (LRS)

If $L > R$, LRS starts from the empty set and repeatedly adds L features and removes R features
If $L < R$, LRS starts from the full set and repeatedly removes R features followed by L feature additions

Algorithm

1. If $L > R$ then start with the empty set $Y = \emptyset$ else start with the full set $Y = X$ goto step 3
2. Repeat SFS step L times
3. Repeat SBS step R times
4. Goto step 2

LRS attempts to compensate for weaknesses in SFS and SBS by backtracking

Method 5: Bidirectional Search (BDS)

- Bidirectional Search is a parallel implementation of SFS and SBS
  - SFS is performed from the empty set
  - SBS is performed from the full set
- To guarantee that SFS and SBS converge to the same solution, we must ensure that
  - Features already selected by SFS are not removed by SBS
  - Features already removed by SBS are not selected by SFS
  - For example, before SFS attempts to add a new feature, it checks if it has been removed by SBS and, if it has, attempts to add the second best feature, and so on. SBS operates in a similar fashion

Method 6: Floating search methods

- Problem with backward selection: if one feature is excluded, it cannot be considered again.
- Floating methods can reconsider features previously discarded.
- Floating search can be defined both for forward and backward selection, here we study forward selection.
- Let $X_k=\{x_1,x_2,...,x_k\}$ be the best combination of the k features and Y_{m-k} the remaining m-k features.
- At the next step the k+1 best subset $X_{k+1}$is formed by 'borrowing' an element from Y_{m-k}.
- Then, return to previously selected lower dimension subset to check whether the inclusion of this new element improves the criterion.
- If so, let the new element replace one of the previously selected features.

Algorithm for floating search

- Step I: Inclusion

  $x_{k+1}=\text{argmax}_{y\in Ym-k}C(\{X_k,y\})$ (choose the element from Y_{m-k} that has best effect of C when combined with X_k).

  Set $X_{k+1}= \{X_k, x_{k+1}\}$.

- Step II: Test

  1. $x_r= \text{argmax}_{y\in X_{k+1}}C(\{X_{k+1}-y\})$ (Find the feature with the least effect on C when removed from X_{k+1})
  2. If r=k+1, change k=k+1 and go to step I.
  3. If $r\neq k+1$ AND $C(\{X_{k+1}-x_r\})<C(X_k)$, goto step I. (If removing x_k did not improve the cost, no further backwards selection)
  4. If k=2 put $X_k= X_{k+1}- x_r$ and $C(X_k)=C(X_{k+1}- x_r)$. Goto step I.

Algorithm cont.

- Step III: Exclusion

  1.$X_k'=X_{k+1}-x_r$ (remove x_r)

  2.$x_s= \text{argmax}_{y\in X_k'}C(\{X_k'-y\})$ (find the least significant feature in the new set.)

  3.If $C(X_k'- x_s)<C(X_{k-1})$ then $X_k= X_k'$ and goto step I.

  4.Put $X_{k-1}'=X_k'-x_s$ and k=k-1.

  5.If k=2, put $X_k=X_k'$ and $C(X_k)=C(X_k')$ and goto step I.

  6.Goto step III.

Floating search often yields better performance than sequential search, but at the cost of increased computational time.

Optimal searches and randomized methods

- If the criterion increases monotonically $J(x_{i1}) \leq J(x_{i1}, x_{i2}) \leq J(x_{i1}, x_{i2}, \ldots, x_{in})$, one can use graph-theoretic methods to perform effective subset searches. (I.e. branch and bound or dynamic programming)
- Randomized methods are also popular, examples would be sequential searching with random starting subsets, simulated annealing (a random subset permutation where the randomness cools off) or genetic algorithms.

Sequential Floating Search (SFFS and SFBS)

- Extension to the LRS algorithms with flexible backtracking capabilities
- Rather than fixing the values of L and R, these floating methods allow those values to be determined from the data: The size of the subset during the search can be thought to be "'floating"'
- Sequential Floating Forward Selection (SFFS) starts from the empty set
  - After each forward step, SFFS performs backward steps as long as the objective function increases
- Sequential Floating Backward Selection (SFBS) starts from the full set
  - After each backward step, SFBS performs forward steps as long as the objective function increases

Feature transforms

- We now consider computing new features as linear combinations of the existing features.
- From the original feature vector x, we compute a new vector y of transformed features

  $y=A^T x$

  y is l-dimensional, x is m-dimensional, A is a l×m matrix.
- y is normally defined in such a way that it has lower dimension than x.