MUSI-6201 — Computational Music Analysis
Part 4.3: Feature Post-Processing

alexander lerch
instantaneous features

overview

- **text book**
  - *Chapter 3: Instantaneous Features* (pp. 63–69)
- **sources**: slides (latex) & Matlab
  - github repository

lecture content

- derived features
- feature normalization
- feature aggregation, transformation, and dimensionality reduction
instantaneous features

overview

- **text book**
  - *Chapter 3: Instantaneous Features* (pp. 63–69)

- **sources**: slides (latex) & Matlab
  - [github repository](#)

- **lecture content**
  - derived features
    - feature normalization
    - feature aggregation, transformation, and dimensionality reduction
instantaneous features

overview

- **text book**
  - *Chapter 3: Instantaneous Features* (pp. 63–69)

- **sources**: slides (latex) & Matlab
  - [github repository](#)

- **lecture content**
  - derived features
  - feature normalization
    - feature aggregation, transformation, and dimensionality reduction
instantaneous features

overview

- **text book**
  - *Chapter 3: Instantaneous Features* (pp. 63–69)

- **sources**: slides (latex) & Matlab
  - [github repository](#)

- **lecture content**
  - derived features
  - feature normalization
  - feature aggregation, transformation, and dimensionality reduction
extracting multiple instantaneous features leads to
   → one feature vector per block, or
   → one feature matrix per audio file

\[ V = \begin{bmatrix}
    v(0) & v(1) & \ldots & v(N-1) \\
    v_0(0) & v_0(1) & \ldots & v_0(N-1) \\
    v_1(0) & v_1(1) & \ldots & v_1(N-1) \\
    \vdots & \vdots & \ddots & \vdots \\
    v_{F-1}(0) & v_{F-1}(1) & \ldots & v_{F-1}(N-1)
\end{bmatrix} \]

dimensions: \( F \times N \) (number of features and number of blocks, resp.)
what do we do with the feature matrix
what do we do with the feature matrix

multiple options that can be combined and depend both on the task and the classifier used

1. derive additional features
2. aggregate existing features (e.g., one feature vector per file)
3. reduce the number of features
4. ensure similar scale and distribution
feature post-processing
examples of derived features

- **diff**: use the change in value
  \[ v_{j,\Delta}(n) = v_j(n) - v_j(n - 1) \]

- **smoothed**: remove high frequency content
  - (anticausal) single-pole
  - moving average

- less common: non-linear combinations, e.g.
  \[ v_{jl}(n) = v_j(n) \cdot v_l(n) \]
feature post-processing

examples of derived features

- **diff**: use the change in value
  \[ v_{j,\Delta}(n) = v_j(n) - v_j(n - 1) \]

- **smoothed**: remove high frequency content
  - (anticausal) single-pole
  - moving average

- less common: non-linear combinations, e.g.
  \[ v_{jl}(n) = v_j(n) \cdot v_l(n) \]
feature post-processing
eexamples of derived features

- **diff**: use the change in value

  \[ v_{j,\Delta}(n) = v_j(n) - v_j(n - 1) \]

- **smoothed**: remove high frequency content
  - (anticausal) single-pole
  - moving average

- less common: non-linear combinations, e.g.

  \[ v_{jl}(n) = v_j(n) \cdot v_l(n) \]
feature post-processing
feature aggregation

calculate *summary features* from feature series: **subfeatures**

- *statistical descriptors*
  - mean, median, max, standard deviation
- *hand crafted*
  - anything that might be meaningful — periodicity, slope, . . .

- could be **hierarchical** process:
  1. *texture window*:
     - split feature series in overlapping blocks of a few seconds length
  2. compute subfeatures per block
  3. compute subfeatures of subfeature series

- note: also compare *pooling* operation in machine learning
calculate *summary features* from feature series: *subfeatures*

- *statistical descriptors*
  - mean, median, max, standard deviation
- *hand crafted*
  - anything that might be meaningful — periodicity, slope, . . .

- could be *hierarchical* process:
  1. *texture window*:
     - split feature series in overlapping blocks of a few seconds length
  2. compute subfeatures per block
  3. compute subfeatures of subfeature series

- note: also compare *pooling* operation in machine learning
calculate *summary features* from feature series: *subfeatures*

- *statistical descriptors*
  - mean, median, max, standard deviation
- *hand crafted*
  - anything that might be meaningful — periodicity, slope, ...

could be *hierarchical* process:

1. *texture window*:
   split feature series in overlapping blocks of a few seconds length
2. compute subfeatures per block
3. compute subfeatures of subfeature series

note: also compare *pooling* operation in machine learning
calculate *summary features* from feature series: subfeatures

- *statistical descriptors*
  - mean, median, max, standard deviation
- *hand crafted*
  - anything that might be meaningful — periodicity, slope, . . .

could be **hierarchical** process:

1. *texture window:*
   - split feature series in overlapping blocks of a few seconds length
2. compute subfeatures per block
3. compute subfeatures of subfeature series

note: also compare *pooling* operation in machine learning
raw features have
- different ranges and scaling factors
- possibly non-symmetric distributions

⇒ potential problems with vector distances and some classifiers

⇒ feature normalization
  - standard approach (zscore)
  - alternative

with

$$v_{j,N}(n) = \frac{v_j(n) - \mu_{v_j}}{\sigma_{v_j}}$$

$$v_{j,N}(n) = \frac{v_j(n) - Q_{v_j}(0.5)}{s_{v_j}}$$

$$s_{v_j} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (v_j(n) - Q_{v_j}(0.5))^2}$$
feature post-processing

normalization 1/2

- raw features have
  - different ranges and scaling factors
  - possibly non-symmetric distributions

⇒ potential problems with vector distances and some classifiers

⇒ feature normalization
  - *standard approach* (zscore)

\[ v_{j,N}(n) = \frac{v_j(n) - \mu_{v_j}}{\sigma_{v_j}} \]

- *alternative*

\[ v_{j,N}(n) = \frac{v_j(n) - Q_{v_j}(0.5)}{s_{v_j}} \]

with

\[ s_{v_j} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (v_j(n) - Q_{v_j}(0.5))^2} \]
Raw features have
- different ranges and scaling factors
- possibly non-symmetric distributions

⇒ potential problems with vector distances and some classifiers

⇒ feature normalization
- standard approach (zscore)

\[ v_{j,N}(n) = \frac{v_j(n) - \mu_{v_j}}{\sigma_{v_j}} \]

- alternative

\[ v_{j,N}(n) = \frac{v_j(n) - Q_{v_j}(0.5)}{s_{v_j}} \]

with

\[ s_{v_j} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} (v_j(n) - Q_{v_j}(0.5))^2} \]
feature post-processing

normalization 2/2

- alternative normalizations
  - normalize range to [0...1]

- symmetrize pdf shape
  - Box-Cox transform

\[

\nu(\lambda) = \begin{cases} 
\frac{\nu^{\lambda-1}}{\lambda}, & \lambda \neq 0 \\
\log(\nu), & \lambda = 0 
\end{cases}

\]

- numerical methods …
**alternative normalizations**
- normalize range to [0…1]

**symmetrize pdf shape**
- Box-Cox transform

\[ v^{(\lambda)} = \begin{cases} \frac{v^{\lambda} - 1}{\lambda}, & \lambda \neq 0 \\ \log(v), & \lambda = 0 \end{cases} \]

- numerical methods . . .
feature post-processing

dimensionality reduction — introduction

bullet problem
  - many ML cannot cope properly with large amounts of irrelevant features
  - ML algorithms might degrade in performance

bullet advantages
  - reducing storage requirements
  - reducing training complexity
  - defying the curse of dimensionality

bullet disadvantages
  - additional workload for reduction
  - adding an additional layer of model complexity
problem
- many ML cannot cope properly with large amounts of irrelevant features
- ML algorithms might degrade in performance

advantages
- reducing storage requirements
- reducing training complexity
- defying the curse of dimensionality

disadvantages
- additional workload for reduction
- adding an additional layer of model complexity
feature post-processing

dimensionality reduction — introduction

- **problem**
  - many ML cannot cope properly with large amounts of irrelevant features
  - ML algorithms might degrade in performance

- **advantages**
  - reducing storage requirements
  - reducing training complexity
  - defying the curse of dimensionality

- **disadvantages**
  - additional workload for reduction
  - adding an additional layer of model complexity
problems of high-dimensional data:

- increase in run-time
- overfitting
- curse of dimensionality
- number of required samples (training set size)

⇒ increasing number of input features may decrease classification performance
problems of high-dimensional data:

- increase in run-time
- overfitting
- curse of dimensionality
- number of required samples (training set size)

⇒ increasing number of input features may *decrease* classification performance
overfitting:
  - lack of training data
  - overly complex model

⇒ model cannot be estimated properly

- required training set size depends on
  - classifier and its parametrization
  - number of classes

https://en.wikipedia.org/wiki/Overfitting
overfitting:

- lack of training data
- overly complex model

⇒ model cannot be estimated properly

required training set size depends on
- classifier and its parametrization
- number of classes

overfitting:

- lack of training data
- overly complex model

⇒ model cannot be estimated properly
feature post-processing

dimensionality reduction — dimensionality issues 2/3

- **overfitting**:
  - lack of training data
  - overly complex model
  → model cannot be estimated properly

- required training set size depends on
  - classifier and its parametrization
  - number of classes
  - ...

**rule of thumb:**
don’t bother with training sets smaller than $F^2$
**curse of dimensionality:**
- increasing dimensionality leads to sparse training data
- neighborhoods of data points become less concentrated
- model tends to be harder to estimate in higher-dimensional space
- applies to distance-based algorithms

**example**
- uniformly distributed data
- identify region required for **1% of data**
  - 2-D: 10% of x-axis/y-axis
  - 3-D: 21.5% of x-axis/y-axis/z-axis
  - 10-D: 63%
  - 100-D: 95%
feature post-processing

dimensionality reduction — approaches

- **feature subset selection:**
  discard least helpful features
    - high “discriminative” or descriptive power
    - non-correlation to other features
    - invariance to irrelevancies

- **feature space transformation:**
  map feature space
feature post-processing

dimensionality reduction — approaches

- **feature subset selection**: discard least helpful features
  - high “discriminative” or descriptive power
  - non-correlation to other features
  - invariance to irrelevancies

- **feature space transformation**: map feature space
feature post-processing
manual feature selection

example scatter plots
of pairs of features in a multi-class scenario
feature post-processing
feature subset selection: introduction

1 wrapper methods:
   - description
     - use the “classifier” itself to evaluate feature performance
   - advantages
     - taking into account feature dependencies
     - model dependency
   - disadvantages
     - complexity
     - risk of overfitting

2 filter methods:
   - description
     - use an objective function
   - advantages
     - easily scalable
     - independent of classification algorithm
   - disadvantages
     - no interaction with classifier
     - no feature dependencies
feature post-processing
feature subset selection: introduction

1 **wrapper methods:**
   - **description**
     - use the “classifier” itself to evaluate feature performance
   - **advantages**
     - taking into account feature dependencies
     - model dependency
   - **disadvantages**
     - complexity
     - risk of overfitting

2 **filter methods:**
   - **description**
     - use an objective function
   - **advantages**
     - easily scalable
     - independent of classification algorithm
   - **disadvantages**
     - no interaction with classifier
     - no feature dependencies
feature post-processing
feature subset selection: introduction

1 wrapper methods:
   • description
     • use the “classifier” itself to evaluate feature performance
   • advantages
     • taking into account feature dependencies
     • model dependency
   • disadvantages
     • complexity
     • risk of overfitting

2 filter methods:
   • description
     • use an objective function
   • advantages
     • easily scalable
     • independent of classification algorithm
   • disadvantages
     • no interaction with classifier
     • no feature dependencies
feature post-processing
feature subset selection: introduction

1 **wrapper methods:**
   - *description*
     - use the “classifier” itself to evaluate feature performance
   - *advantages*
     - taking into account feature dependencies
     - model dependency
   - *disadvantages*
     - complexity
     - risk of overfitting

2 **filter methods:**
   - *description*
     - use an objective function
   - *advantages*
     - easily scalable
     - independent of classification algorithm
   - *disadvantages*
     - no interaction with classifier
     - no feature dependencies
feature post-processing

feature subset selection: introduction

1. **wrapper methods:**
   - **description**
     - use the “classifier” itself to evaluate feature performance
   - **advantages**
     - taking into account feature dependencies
     - model dependency
   - **disadvantages**
     - complexity
     - risk of overfitting

2. **filter methods:**
   - **description**
     - use an objective function
   - **advantages**
     - easily scalable
     - independent of classification algorithm
   - **disadvantages**
     - no interaction with classifier
     - no feature dependencies
wrapper methods:
- **description**
  - use the “classifier” itself to evaluate feature performance
- **advantages**
  - taking into account feature dependencies
  - model dependency
- **disadvantages**
  - complexity
  - risk of overfitting

filter methods:
- **description**
  - use an objective function
- **advantages**
  - easily scalable
  - independent of classification algorithm
- **disadvantages**
  - no interaction with classifier
  - no feature dependencies
simple classifier — nearest neighbor

- **training:**
  - store feature vector (& class label) of each training sample
- **classification:**
  - for new file/feature vector, detect *closest training point*
  - choose closest point's class as result

![Scatter plot showing mean spectral centroid vs. std rms for music and speech](matlab/displayScatter.m)
simple classifier — nearest neighbor

- **training:**
  - store feature vector (\& class label) of each training sample

- **classification:**
  - for new file/feature vector, detect *closest training point*
  - choose closest point's class as result

![scatter plot with overlapping points for music and speech]

⇒ identify class of nearest point

---

**excursion**

music

speech

mean spectral centroid

std rms

⇒ identify class of nearest point

matlab source: matlab/displayScatter.m
feature post-processing

feature subset selection: wrapper methods 1/2

1. **single variable classification:**
   - **procedure**
     - evaluate each feature individually
     - choose the top $N$
   - **complexity**
     - subsets to test: $F$
   - **challenges**
     - inter-feature correlation is not considered
     - feature combinations are not considered

2. **brute force subset selection**
   - **procedure**
     - evaluate all possible feature combinations
     - choose the optimal combination
   - **complexity**
     - subsets to test: $2^F$
   - **challenges**
     - best solution, but
     - not feasible with large number of features
feature post-processing

feature subset selection: wrapper methods 1/2

1. **single variable classification:**
   - **procedure**
     - evaluate each feature individually
     - choose the top $N$
   - **complexity**
     - subsets to test: $\mathcal{F}$
   - **challenges**
     - inter-feature correlation is not considered
     - feature combinations are not considered

2. **brute force subset selection**
   - **procedure**
     - evaluate all possible feature combinations
     - choose the optimal combination
   - **complexity**
     - subsets to test: $2^\mathcal{F}$
   - **challenges**
     - best solution, but
     - not feasible with large number of features
feature post-processing

feature subset selection: wrapper methods 1/2

1. **Single variable classification:**
   - **Procedure**
     - evaluate each feature individually
     - choose the top \( N \)
   - **Complexity**
     - subsets to test: \( \mathcal{F} \)
   - **Challenges**
     - inter-feature correlation is not considered
     - feature combinations are not considered

2. **Brute force subset selection**
   - **Procedure**
     - evaluate all possible feature combinations
     - choose the optimal combination
   - **Complexity**
     - subsets to test: \( 2^{\mathcal{F}} \)
   - **Challenges**
     - best solution, but
     - not feasible with large number of features
feature post-processing

feature subset selection: wrapper methods 1/2

1. **single variable classification:**
   - **procedure**
     - evaluate each feature individually
     - choose the top $N$
   - **complexity**
     - subsets to test: $\mathcal{F}$
   - **challenges**
     - inter-feature correlation is not considered
     - feature combinations are not considered

2. **brute force subset selection**
   - **procedure**
     - evaluate all possible feature combinations
     - choose the optimal combination
   - **complexity**
     - subsets to test: $2^\mathcal{F}$
   - **challenges**
     - best solution, but
     - not feasible with large number of features
feature post-processing

feature subset selection: wrapper methods 1/2

1 **single variable classification:**
   - **procedure**
     - evaluate each feature individually
     - choose the top $N$
   - **complexity**
     - subsets to test: $\mathcal{F}$
   - **challenges**
     - inter-feature correlation is not considered
     - feature combinations are not considered

2 **brute force subset selection**
   - **procedure**
     - evaluate all possible feature combinations
     - choose the optimal combination
   - **complexity**
     - subsets to test: $2^\mathcal{F}$
   - **challenges**
     - best solution, but
     - not feasible with large number of features
feature post-processing
feature subset selection: wrapper methods 1/2

1 single variable classification:
   - procedure
     - evaluate each feature individually
     - choose the top $N$
   - complexity
     - subsets to test: $\mathcal{F}$
   - challenges
     - inter-feature correlation is not considered
     - feature combinations are not considered

2 brute force subset selection
   - procedure
     - evaluate all possible feature combinations
     - choose the optimal combination
   - complexity
     - subsets to test: $2^\mathcal{F}$
   - challenges
     - best solution, but
     - not feasible with large number of features
**Sequential Forward Selection**

- **Procedure**
  1. **Init:** empty feature subset $V_s = \emptyset$
  2. Find feature $v_j$ maximizing objective function
  
  \[
  v_j = \arg\max_{\forall j \mid v_j \notin V_s} J(V_s \cup v_j)
  \]
  3. Add feature $v_j$ to $V_s$
  4. Go to step 2

- **Challenges**
  - In theory, the optimal solution may be missed

**Sequential Backward Elimination**

- **Procedure**
  1. **Init:** full feature set
  2. Find feature $v_j$ with the least impact on objective function
  3. Discard feature $v_j$
  4. Go to step 2

- **Challenges**
  - Complex with a large number of features
feature post-processing

feature subset selection: wrapper methods 2/2

4 **sequential forward selection**

- **procedure**
  1. init: empty feature subset \( V_s = \emptyset \)
  2. find feature \( v_j \) maximizing objective function
     \[
     v_j = \arg\max_{\forall j} J(V_s \cup v_j) \quad \forall j|v_j \notin V_s
     \]
  3. add feature \( v_j \) to \( V_s \)
  4. go to step 2

- **challenges**
  - in theory, the optimal solution may be missed

5 **sequential backward elimination**

- **procedure**
  1. init: full feature set
  2. find feature \( v_j \) with the least impact on objective function
  3. discard feature \( v_j \)
  4. go to step 2

- **challenges**
  - complex with a large number of features
feature post-processing
feature subset selection: wrapper methods 2/2

4 sequential forward selection
   procedure
   1 init: empty feature subset \( V_s = \emptyset \)
   2 find feature \( v_j \) maximizing objective function
     \[
     v_j = \arg\max_{\forall j \, |v_j \notin V_s} J(V_s \cup v_j)
     \]
   3 add feature \( v_j \) to \( V_s \)
   4 go to step 2
   challenges
   in theory, the optimal solution may be missed

5 sequential backward elimination
   procedure
   1 init: full feature set
   2 find feature \( v_j \) with the least impact on objective function
   3 discard feature \( v_j \)
   4 go to step 2
   challenges
   complex with a large number of features
feature post-processing

feature subset selection: wrapper methods 2/2

4 sequential forward selection

- procedure
  1. init: empty feature subset $V_s = \emptyset$
  2. find feature $v_j$ maximizing objective function

$$v_j = \arg\max_{v_j \mid v_j \notin V_s} J(V_s \cup v_j)$$

3. add feature $v_j$ to $V_s$
4. go to step 2

- challenges
  - in theory, the optimal solution may be missed

5 sequential backward elimination

- procedure
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2

- challenges
  - complex with a large number of features
feature post-processing

feature subset selection: wrapper methods 2/2

4. **sequential forward selection**
   - **procedure**
     1. init: empty feature subset \( V_s = \emptyset \)
     2. find feature \( v_j \) maximizing objective function
        \[
        v_j = \arg\max_{v_j \in V \setminus V_s} J(V_s \cup v_j)
        \]
     3. add feature \( v_j \) to \( V_s \)
     4. go to step 2
   - **challenges**
     - in theory, the optimal solution may be missed

5. **sequential backward elimination**
   - **procedure**
     1. init: full feature set
     2. find feature \( v_j \) with the least impact on objective function
     3. discard feature \( v_j \)
     4. go to step 2
   - **challenges**
     - complex with a large number of features
feature post-processing
feature subset selection: wrapper methods 2/2

4 sequential forward selection

- **procedure**
  1. init: empty feature subset $V_s = \emptyset$
  2. find feature $v_j$ maximizing objective function
     $$v_j = \arg \max \forall J(V_s \cup v_j) \forall j | v_j \notin V_s$$
  3. add feature $v_j$ to $V_s$
  4. go to step 2

- **challenges**
  - in theory, the optimal solution may be missed

5 sequential backward elimination

- **procedure**
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2

- **challenges**
  - complex with a large number of features
feature post-processing
feature subset selection: wrapper methods 2/2

4 sequential forward selection
- procedure
  1. init: empty feature subset $V_s = \emptyset$
  2. find feature $v_j$ maximizing objective function
     \[ v_j = \arg\max_{\forall j | v_j \notin V_s} J(V_s \cup v_j) \]
  3. add feature $v_j$ to $V_s$
  4. go to step 2
- challenges
  - in theory, the optimal solution may be missed

5 sequential backward elimination
- procedure
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2
- challenges
  - complex with a large number of features
feature post-processing

feature subset selection: wrapper methods 2/2

4 sequential forward selection

- procedure
  1. init: empty feature subset $\mathcal{V}_s = \emptyset$
  2. find feature $v_j$ maximizing objective function
     
     $$v_j = \arg\max_{\forall j | v_j \notin \mathcal{V}_s} J(\mathcal{V}_s \cup v_j)$$
  3. add feature $v_j$ to $\mathcal{V}_s$
  4. go to step 2

- challenges
  - in theory, the optimal solution may be missed

5 sequential backward elimination

- procedure
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2

- challenges
  - complex with a large number of features
feature post-processing

feature subset selection: wrapper methods 2/2

4 sequential forward selection

- procedure
  1. init: empty feature subset $V_s = \emptyset$
  2. find feature $v_j$ maximizing objective function
     $$v_j = \arg\max_{\forall j | v_j \notin V_s} J(V_s \cup v_j)$$
  3. add feature $v_j$ to $V_s$
  4. go to step 2

- challenges
  - in theory, the optimal solution may be missed

5 sequential backward elimination

- procedure
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2

- challenges
  - complex with a large number of features
4 **Sequential Forward Selection**

- *procedure*
  1. init: empty feature subset $\mathcal{V}_s = \emptyset$
  2. find feature $v_j$ maximizing objective function

$$v_j = \arg\max_{\forall j \mid v_j \notin \mathcal{V}_s} J(\mathcal{V}_s \cup v_j)$$

3. add feature $v_j$ to $\mathcal{V}_s$
4. go to step 2

- *challenges*
  - in theory, the optimal solution may be missed

5 **Sequential Backward Elimination**

- *procedure*
  1. init: full feature set
  2. find feature $v_j$ with the least impact on objective function
  3. discard feature $v_j$
  4. go to step 2

- *challenges*
  - complex with a large number of features
feature post-processing
feature space transformation: PCA introduction

- **objective**
  - map features to new coordinate system

  \[ u(n) = T^T \cdot v(n) \]

  - \( u(n) \): transformed features (same dimension as \( v(n) \))
  - \( T \): transformation matrix \((F \times F)\)

\[
T = \begin{bmatrix}
    c_0 & c_1 & \ldots & c_{F-1}
\end{bmatrix}
\]

- **properties**
  - \( c_0 \) points in the direction of highest variance
  - variance concentrated in as few output components as possible
  - \( c_i \) orthogonal

  \[
c_i^T \cdot c_j = 0 \quad \forall \; i \neq j
\]

  - transformation is invertible

  \[
v(n) = T \cdot u(n)
\]
objective
- map features to new coordinate system

\[ u(n) = T^T \cdot v(n) \]

- \( u(n) \): transformed features (same dimension as \( v(n) \))
- \( T \): transformation matrix \((F \times F)\)

\[
T = \begin{bmatrix}
c_0 & c_1 & \ldots & c_{F-1}
\end{bmatrix}
\]

properties
- \( c_0 \) points in the direction of highest variance
- variance concentrated in as few output components as possible
- \( c_i \) orthogonal

\[ c_i^T \cdot c_j = 0 \quad \forall \ i \neq j \]
- transformation is invertible

\[ v(n) = T \cdot u(n) \]
**objective**
- map features to new coordinate system

\[ u(n) = T^T \cdot v(n) \]

- \( u(n) \): transformed features (same dimension as \( v(n) \))
- \( T \): transformation matrix \((\mathcal{F} \times \mathcal{F})\)

\[ T = \begin{bmatrix} c_0 & c_1 & \ldots & c_{\mathcal{F}-1} \end{bmatrix} \]

**properties**
- \( c_0 \) points in the direction of highest *variance*
- variance concentrated in as few output components as possible
- \( c_i \) orthogonal

\[ c_i^T \cdot c_j = 0 \quad \forall \ i \neq j \]

- transformation is invertible

\[ v(n) = T \cdot u(n) \]
feature post-processing

feature space transformation: PCA introduction

**objective**
- map features to new coordinate system
  \[ u(n) = T^T \cdot v(n) \]

  - \( u(n) \): transformed features (same dimension as \( v(n) \))
  - \( T \): transformation matrix (\( F \times F \))

\[ T = \begin{bmatrix} c_0 & c_1 & \ldots & c_{F-1} \end{bmatrix} \]

**properties**
- \( c_0 \) points in the direction of highest *variance*
- variance concentrated in as few output components as possible
- \( c_i \) orthogonal
  \[ c_i^T \cdot c_j = 0 \quad \forall \ i \neq j \]
- transformation is invertible
  \[ v(n) = T \cdot u(n) \]
feature post-processing

feature space transformation: PCA introduction

- **objective**
  - map features to new coordinate system
    \[ u(n) = T^T \cdot v(n) \]
    - \( u(n) \): transformed features (same dimension as \( v(n) \))
    - \( T \): transformation matrix (\( F \times F \))

```
T = [ c_0  c_1  ...  c_{F-1} ]
```

- **properties**
  - \( c_0 \) points in the direction of highest variance
  - variance concentrated in as few output components as possible
  - \( c_i \) orthogonal
    \[ c_i^T \cdot c_j = 0 \quad \forall \ i \neq j \]
  - transformation is invertible
    \[ v(n) = T \cdot u(n) \]
feature post-processing
feature space transformation: PCA visualization

calculation of the transformation matrix

1. compute covariance matrix $R$

$$R = \mathbb{E}\{(V - \mathbb{E}\{V\})(V - \mathbb{E}\{V\})\}$$

2. choose eigenvectors as axes for the new coordinate system
feature post-processing

feature space transformation: PCA visualization

calculation of the transformation matrix

1. compute covariance matrix $R$

$$R = \mathcal{E}((V - \mathcal{E}\{V\})(V - \mathcal{E}\{V\}))$$

2. choose eigenvectors as axes for the new coordinate system
PCA example

**pca input**

![PCA Input Diagram](image-url)
feature post-processing
PCA example

pca output
feature post-processing
PCA example

**pca eigenvalues**

![Eigenvalue plot for PCA example](matlab/displayPcaExample.m)
PCA example

**pca transformation matrix**

\[
\begin{bmatrix}
-0.4187 & 0.3467 & -0.4569 & 0.4143 & -0.1271 & -0.5549 \\
-0.3908 & 0.1815 & 0.8136 & -0.0289 & 0.2060 & -0.3304 \\
-0.4516 & 0.3384 & 0.0859 & 0.2413 & -0.2919 & 0.7285 \\
-0.4337 & 0.1699 & -0.3337 & -0.7243 & 0.3747 & 0.0816 \\
0.3802 & 0.5599 & -0.0381 & 0.2808 & 0.6622 & 0.1524 \\
0.3679 & 0.6245 & 0.0956 & -0.4071 & -0.5267 & -0.1495 \\
\end{bmatrix}
\]
PCA example

**pca transformation matrix**

\[
\begin{bmatrix}
-0.4187 & 0.3467 & -0.4569 & 0.4143 & -0.1271 & -0.5549 \\
-0.3908 & 0.1815 & 0.8136 & -0.0289 & 0.2060 & -0.3304 \\
-0.4516 & 0.3384 & 0.0859 & 0.2413 & -0.2919 & 0.7285 \\
-0.4337 & 0.1699 & -0.3337 & -0.7243 & 0.3747 & 0.0816 \\
0.3802 & 0.5599 & -0.0381 & 0.2808 & 0.6622 & 0.1524 \\
0.3679 & 0.6245 & 0.0956 & -0.4071 & -0.5267 & -0.1495 \\
\end{bmatrix}
\]
feature post-processing
feature space transformation: PCA exercise

**matlab exercise:** compute the principle components

1. extract 3 features: Spectral Centroid, Spectral Flux, and RMS
2. normalize the features
3. compute the principle components (`pca()` statistics toolbox)
4. analyze the transformation matrix and the variances — what can you learn about the input features
1. name examples for typical ways of computing derived features
2. why is more features not always better
3. what is the difference between feature selection and feature mapping
4. describe two ways of selecting features
5. what is PCA and what are the advantages
1. name examples for typical ways of computing derived features
2. why is more features not always better
3. what is the difference between feature selection and feature mapping
4. describe two ways of selecting features
5. what is PCA and what are the advantages
1. Name examples for typical ways of computing derived features.
2. Why is more features not always better?
3. What is the difference between feature selection and feature mapping?
4. Describe two ways of selecting features.
5. What is PCA and what are the advantages?
1. Name examples for typical ways of computing derived features
2. Why is more features not always better
3. What is the difference between feature selection and feature mapping
4. Describe two ways of selecting features
5. What is PCA and what are the advantages
1. Name examples for typical ways of computing derived features
2. Why is more features not always better
3. What is the difference between feature selection and feature mapping
4. Describe two ways of selecting features
5. What is PCA and what are the advantages