Searching and mining sequential data

**Disclaimer:** some of the images in slides 62-69 have been taken from UCR and Prof. Eamonn Keogh.
Outline

• Sequences
  ▫ Examples and analysis tasks

• Searching large sequences
  ▫ EBSM: Embedding-based time series subsequence matching
  ▫ DRESS: Dimensionality reduction for event subsequence matching

• Learning from time series
  ▫ Shapelets and shapelet trees
  ▫ Random shapelet forests

• Current challenges
Sequences

- Distinct events: e.g., text, DNA
- Variables over time: e.g., time series
- Interval-based events: e.g., sign language

```
TCTAGGGCA
```

```
head-shake

eye-brow raise

wh-word

wh-question
```
Analysis Tasks

- Classification
- Clustering
- Outlier Detection
- Frequent Pattern Mining

Similarity Search
Similarity search

- Time series matching / stream monitoring
  - ischemia
  - ECG of a patient

- DNA sequence alignment
  - possibility of cancer
  - human genome
  - TCTAGGGCA
  - GGATATTAAGAATAGGGATATA
Similarity search

- Query-by-humming

Music piece  $\rightarrow$ sequence of notes

![Diagram showing query-by-humming process with music piece and sequence of notes.](image-url)
Sequence classification

• Time series classification

![ECG of a patient](image1)

ECG of a patient → ischemia

• Gene classification

![DNA sequence](image2)

TCTAGGGCA → possibility of cancer
Two fundamental questions

- How to define an appropriate distance measure for the task at hand?

- How to speed up the search under that distance measure?
Time Series Similarity

- Given two time series

\[ X = (x_1, x_2, \ldots, x_n) \quad \text{and} \quad Y = (y_1, y_2, \ldots, y_n) \]

- Define and compute \( D(X, Y) \)

- Or better...
Time Series Similarity

- Given a time series database and a query $X$
- Find the best match of $X$ in the database

Why is that useful?
Examples

• Find companies with similar stock prices over a time interval
• Find products with similar sell cycles
• Find songs with similar music patterns
• Cluster users with similar credit card utilization
• Find similar subsequences in DNA sequences
• Find scenes in video streams
Elastic distance measures
(images taken from Eamonn Keogh)

- **Euclidean**
  - rigid

- **Dynamic Time Warping (DTW)**
  - allows local scaling

- **Longest Common SubSequence (LCSS)**
  - allows local scaling
  - ignores outliers

\[
D(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}
\]
Properties of DTW

- Warping path $W$:
  - set of grid cells in the time warping matrix
- DTW finds an optimal warping path $W$:
  - the path with the smallest matching score

Properties of a DTW legal path

I. Boundary conditions
   
   $W_1=(1,1)$ and $W_K=(n,m)$

II. Continuity
   
   Given $W_k = (a, b)$, then
   
   $W_{k-1} = (c, d)$, where $a-c \leq 1$, $b-d \leq 1$

III. Monotonicity
   
   Given $W_k = (a, b)$, then
   
   $W_{k-1} = (c, d)$, where $a-c \geq 0$, $b-d \geq 0$
Global Constraints

- Slightly speed up the calculations and prevent pathological warpings
- A global constraint limits the indices of the warping path
  \[ w_k = (i, j)_k \text{ such that } j-r \leq i \leq j+r \]
- Where \( r \) is a term defining allowed range of warping for a given point in a sequence
Speeding up search under DTW

...using embeddings

**EBSM**: subsequence matching for time series, SIGMOD 2008

**RBSM**: alignment of event sequences, VLDB 2009

**EBSM++**: subsequence and full sequence matching for time series, ACM TODS 2011

**EBESM**: full matching of temporal interval sequences, DAMI 2017
Strategy: identify candidate matches

Database X
Strategy: identify candidate matches

Database X

indexing structure
Strategy: identify candidate matches

Database X

indexing structure

query Q
Strategy: identify candidate matches

Database X

candidates

indexing structure

candidates

query Q

candidate: possible match

Main principle: use costly distance computation only to evaluate the candidates
Vector embedding

| Database | $x_1$ | $x_2$ | $x_3$ | $x_4$ | $x_5$ | $x_6$ | $x_7$ | $x_8$ | $x_9$ | $x_{10}$ | $x_{11}$ | $x_{12}$ | $x_{13}$ | $x_{14}$ | $x_{15}$ |
# Vector embedding

<table>
<thead>
<tr>
<th>Database</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
<th>$x_{10}$</th>
<th>$x_{11}$</th>
<th>$x_{12}$</th>
<th>$x_{13}$</th>
<th>$x_{14}$</th>
<th>$x_{15}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>vector</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Vector embedding

### Database

<table>
<thead>
<tr>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
<th>$X_5$</th>
<th>$X_6$</th>
<th>$X_7$</th>
<th>$X_8$</th>
<th>$X_9$</th>
<th>$X_{10}$</th>
<th>$X_{11}$</th>
<th>$X_{12}$</th>
<th>$X_{13}$</th>
<th>$X_{14}$</th>
<th>$X_{15}$</th>
</tr>
</thead>
</table>

### Vector set

#### query

| $Q_1$ | $Q_2$ | $Q_3$ | $Q_4$ | $Q_5$ |
Vector embedding

Database

\[ \mathbf{X} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \]

vector set

query \quad Q_1 \quad Q_2 \quad Q_3 \quad Q_4 \quad Q_5

query vector
Embedding should be such that:

- Query vector is similar to match vector
Using vectors, we identify candidates much faster than brute-force search.
Using reference sequences

- Define a set of reference sequences

- Apply your favorite distance measure (e.g., DTW) to compute the cost of the best match of R and X

- Define $F^R(X)$ to be that cost

- $F^R$ is a 1D embedding
  - Each $F^R(X) \rightarrow$ single real number
Using reference sequences

- Apply the same matching computation against the query:
  - Matching cost of match R with Q
- Define $F^R(Q)$ to be that cost
Intuition about this embedding

- Suppose \( Q \) appears \textit{exactly} in \( X \)
- Then:
  - Warping paths are the same
  - \( F^R(Q) = F^R(X) \)

- Suppose \( Q \) appears \textit{inexactly} in \( X \)
- Then:
  - We expect \( F^R(Q) \) to be similar to \( F^R(X) \)
  - Why? \textit{Little tweaks} should affect \( F^R(X) \) little
  - No proof, \textit{but intuitive}, and lots of empirical evidence
Multi-dimensional embedding

- one reference sequence → 1D embedding
- two reference sequences → 2D embedding
Multi-dimensional embedding

- $d$ reference sequences $\rightarrow d$-dimensional embedding $F$
- Basic principle:
  - If $X_j$ is the best match of $Q$
  - $F(Q)$ should (for most $Q$) be more similar to $F(X, j)$ than to most $F^R(X, t)$
Filter-and-refine retrieval

**Offline step:**
- Compute $F^R(X)$ for each sequence in $X$

**Online steps - given a query $Q$:**
- Embedding step:
  - Compute $F^R(Q)$
- Filter step:
  - Compare $F^R(Q)$ to all $F^R(X)$
  - Select $p$ best matches $\rightarrow p$ candidate endpoints
- Refine step:
  - Use dynamic programming to evaluate each candidate
Filter-and-refine performance

Database X

candidates

• Accuracy:
  ✓ correct match must be among $p$ candidates, for most queries

• Trade-off:
  ✓ larger $p \Rightarrow$ higher accuracy, lower efficiency
Embedding Optimization

• How many reference sequence to choose?
• Which ones to choose?

• Solution:
  ▫ extract the set of reference sequences with the highest distance variance in the dataset
  ▫ choose those top $K$ that can guarantee the best trade-off between accuracy and $p$
  ▫ cross-validation (one-time pre-processing step)
  ▫ sample of queries (input to the algorithm)
  ▫ large pool of reference sequences
Performance

- 20 datasets: UCR Time Series Data Mining Repository
- Queries: 5,397
- Query lengths between 60 and 637
- 50% were used for embedding optimization

- EBSM can achieve over an order of magnitude speedup compared to brute-force search for ALL datasets!
  (2% to 5% of the database is examined) with accuracy of 99%
- And twice as fast as LB_PAA+LB_Keogh / UCR_Suite
Application: Query-by-humming

- Competitive performance on real queries
- Demo available: implementation in Matlab
Similarity search: event sequences

• Similarity search in large sequence databases
  → a frequently occurring problem

• Plethora of string matching methods

• Still a high demand for new robust, and scalable similarity search methods that can handle
  ▫ large query lengths
  ▫ large similarity ranges

• Our focus: Whole sequence matching
Motivation: large query lengths

- Expressed Sequence Tag (EST) databases:
  - common for representing large genomes
  - portions of genes expressed as mRNA
  - sequence length at least 500 – 800

- Large scale searches need to be performed against other genomic databases to determine locations of genes

- Searches can also target whole chromosomes, where the goal is to find chromosome similarities across different organisms
Motivation: large query lengths

- High evolutionary divergence \(\rightarrow\) task of identifying *distantly related gene or protein* domains by sequence search techniques not always trivial

- **Mutation process**: intermediate sequences may possess features of many proteins and facilitate detection of remotely related proteins

- Large range queries, aka *remote homology search*, in bioinformatics can be highly beneficial for searching proteins and genomes
Problem setting

- Given
  - a database $S$ and a query $Q$
  - a similarity range $r = \delta|Q|$

- Task:
  - retrieve all database strings $X$ such that $\text{ED} (Q, X) \leq r$
Existing work

- Global alignment [ED, Q-grams, RBE]
  - full-sequence matching approach for **global optimization**
  - identify the **min number of changes** that should be performed to one sequence so as to convert it to the other
  - forcing the alignment to **span the whole sequence**

- Local alignment [SW, BLAST, RBSA]
  - identify **local regions** within the compared sequences that are highly similar to each other, while they could be globally divergent
  - may **allow for gaps** in the alignment

- Short-read sequencing [SOAP, MAQ, WHAM]
The Edit distance

- Measures how **dissimilar** two strings are
- Computes the minimum cost of edit operations (**insertion**, **deletion**, **substitution**) to convert one string to the other

- Auxiliary matrix $\alpha$
  - $C_{\text{ins}}$: insertion cost
  - $C_{\text{del}}$: deletion cost
  - $C_{\text{sub}}$: substitution cost

\[
C(Q_j, X_i) = \begin{cases} 
C_{\text{sub}} & \text{if } Q_j \neq X_i \\
0 & \text{if } Q_j = X_i 
\end{cases}
\]

**initialization:**
\[
a^{0,0} = 0, a^{j,0} = a^{0,i} = \infty .
\]

**loop:**
\[
a^{j,i}(Q, X) = \min \left\{ a^{j-1,i}(Q, X) + C_{\text{ins}}, a^{j,i-1}(Q, X) + C_{\text{del}}, a^{j-1,i-1}(Q, X) + C(Q_j, X_i) \right\} (j = 1, \ldots, |Q|; i = 1, \ldots, |X|).
\]

**termination:**
\[
D(Q, X) = a^{|Q|,|X|}(Q, X).
\]
The Edit distance

Example:

\[ A = \text{ATC} \quad \text{and} \quad B = \text{ACTG} \]

\[ A = \begin{array}{ccc} A & - & T & C \end{array} \quad \text{ED} (A,B) = 2 \]

\[ B = \begin{array}{ccc} A & C & T & G \end{array} \]
Our contributions

- **DRESS**: a novel filter-and-refine approximate method for speeding up similarity search under edit distance:
  - no index training required
  - can handle large query lengths and similarity ranges

- Efficient string representation in a new space, based on a set of query-specific codewords \( \rightarrow \) distance computation between strings is significantly faster than in original space

- Extensive experimental evaluation against state-of-the-art on three large protein and two DNA sequence datasets
  - DRESS outperforms competitors with very competitive accuracy and retrieval runtime
The DRESS framework

- A three-step framework

1. **Given a query** \( Q \) **defined in the original string space.**
2. **Identify the** \( t \) **most common codewords of size** \( l \) **in** \( Q \).**
3. **Map** \( Q \) **to its mapped version** \( q \) **and each** \( X \) **to its mapped version** \( x \).**
4. **Perform range query search** in the new string space using \( \delta' \) under distance \( D \).**
5. **Identify a set of candidate matches.**
6. **Refine each candidate under** \( D \) **by validating whether it is a** \( r \)-range query result.
The mapping step: alphabet reduction

- This is an online query-sensitive step
- For each query $Q$, identify a set of codewords $E = \{E_1, \ldots, E_t\}$
  - $E$ contains the $t$ most frequent substrings of length $l$
- **IMPORTANT:** no pair in $E$ has an overlapping prefix-suffix
- $t$ and $l$ are identified after appropriate training
The mapping step: alphabet reduction

1. Set $\mathbb{E} = \emptyset$.
2. Find codeword $E$ with highest count, such that:
   - Codeword $E$ is still not an element of $\mathbb{E}$.
   - No suffix of any codeword of $\mathbb{E}$ is a prefix of $E$.
   - No prefix of any codeword of $\mathbb{E}$ is a suffix of $E$.
3. Insert $E$ to $\mathbb{E}$.
4. If $|\mathbb{E}| \neq t$, go to step 2, else we are done.
The mapping step: example

Let \( X = (babfcde) \) and \( E = \{ab, cd\} \)

Sequences are traversed from left to right!
The mapping step: faster than linear

- The process described above is linear to the database size

- **Inverted index**: precomputed for the database, where we store for each possible code-word of length $l$:
  - all the sequence indices and positions where it occurs
  - $\text{IND}[e][i]$: list of all positions where codeword $e$ occurs in the $i^{th}$ database sequence

- **NOTE**: The index is built in linear time
The mapping step: in detail

- Let $X_i$ be a database sequence and $x_i$ denote its $E$-mapping

1. Set $Pairs = \emptyset$
2. For each $E \in E$:
   - Set $e$ to the letter assigned to keyword $E$.
   - For each position $pos$ in $IND[E][i]$:
     - Insert pair $(e, pos)$ to $Pairs$.
3. Sort $Pairs$ in ascending order of the positions (i.e., the second elements of the pairs).
4. Set $x_i$ to be the empty string.
5. For each pair $(e, pos)$ in $Pairs$, considered in sorted order:
   - Insert letter $e$ to the end of $x_i$. 
The mapping step: complexity

- Complexity of the mapping step for a single DB sequence: $O(|x_i|\log|x_i|)$

- Good news:
  - Typically the length of $x_i$ is much smaller than the length of $X_i$
  - Hence, faster than scanning $X_i$
The filter-and-refine steps

- Query is mapped to the E-space
- Each DB sequence is mapped to the E-space
- Brute-force search under Edit Distance on the E-space
- Identify a set of candidates that are within a factor of $\delta'$ of the query length
- Refine the candidates in the original space under Edit Distance
The matching range in the E-space

- The matching range $\delta'$ in the E-space depends on the original $\delta$ and is set to be $\delta' = f \delta$

- Clearly, $\delta'$ should be set higher than $\delta$ to account for the cases where the distance in the E-space is a higher percentage of the mapped query length

- The latter can easily happen because the mapped strings are drastically shorter than the original strings

- **Hence:** while we expect similar strings to have similar mappings, the loss incurred by the mapping can lead to higher distances as % of the query length
Experimental setup

- Two real datasets
- Three methods
- Variable query sizes
- Several parameters
Experimental setup: Datasets

- **UniProt [Protein sequences]:** [http://www.ebi.ac.uk/uniprot/](http://www.ebi.ac.uk/uniprot/)
  - 530,264 strings
  - 25-letter alphabet

  - 249,250,621 bases available
  - 4-letter alphabet
Experimental setup: Datasets

<table>
<thead>
<tr>
<th></th>
<th>UniProt</th>
<th>DNA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$\text{Dataset}_{800}^{401}$</td>
<td>$\text{Dataset}_{1600}^{801}$</td>
</tr>
<tr>
<td>sequence length</td>
<td>$[401, 800]$</td>
<td>$[801, 1600]$</td>
</tr>
<tr>
<td>total # of sequences</td>
<td>130,962</td>
<td>28,155</td>
</tr>
<tr>
<td># of seq. in validation set</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td># of seq. in database</td>
<td>130,362</td>
<td>27,555</td>
</tr>
<tr>
<td># of seq. in test set</td>
<td>500</td>
<td>500</td>
</tr>
</tbody>
</table>
Experimental setup: Methods

- **DRESS**: our method
- **RBE**: an embedding-based method for sequence search
- **Q-grams**: uses inverted indices to find candidate matches
- Note that DRESS is an **approximate** method, while RBE and Q-grams are **exact**
Experimental results: retrieval cost

- Retrieval cost on UniProt [401,800]

- DRESS achieves up to 29 times lower retrieval cost against RBE and up to 75 times lower than Q-grams
Experimental results: retrieval cost

- Retrieval cost on UniProt [801, max]

- DRESS achieves up to 75 times lower retrieval cost against RBE and up to 461 times lower than Q-grams
Experimental results: runtime

- Protein datasets:

- DNA datasets:
Conclusions

- **DRESS**: novel method for whole sequence matching

- Supports large query lengths and matching ranges

- Experiments demonstrate that for higher values of search range, DRESS produces significantly lower costs and runtimes than the competitors

- **Loss** of guarantee of 100% recall

- This price can be an acceptable trade-off in several domains given the significant runtime savings
Future work

- Further improve the performance of DRESS by, e.g., implementing multiple filter steps
- Study the performance of DRESS on sequences from other domains, such as text
- Adapt DRESS for local alignment and short-read sequencing
- Explore alternative ways of producing codewords, e.g., learn synthetic words
Time Series classification

- Application: Finance, Medicine, Music

- 1-Nearest Neighbor
  - Pros: accurate, robust, simple
  - Cons: time and space complexity (lazy learning); results are not interpretable
Solution

• **Shapelets:**
  - time series subsequence
  - representative of a class
  - discriminative from other classes

• **Idea:**
  - Use shapelets as “attributes” or “features” for splitting a node in the decision tree
The Shapelet Classifier

Two steps:

- **Learn** a set of discriminative shapelets (typically a tree-like structure)

- **Predict** a class label for a previously unseen data series using the minimum distance to shapelets (following a path in the tree-structure model)
The Shapelet Classifier

Shapelet Dictionary

Shapelet Tree

Method | Accuracy | Time
--- | --- | ---
Shapelet | 0.720 | 0.86
Nearest Neighbor | 0.543 | 0.65
The utility of a Shapelet

- Collect all candidate shapeletes in a pool
- For each candidate: arrange the time series objects based on the distance from a candidate shapelet
- Find the optimal split point (maximal information gain)
- Pick the candidate achieving best utility as the shapelet
Extracting all Shapelets

\[
\sum_{l=\text{MINLEN}}^{\text{MAXLEN}} \sum_{T_i \in D} (|T_i| - l + 1)
\]

- **DB:** 200
- **Length:** 275
- **Candidates:** 7,480,200
- **Computation:** 3 days
Speeding up candidate generation

- **Main bottleneck:** candidate generation

  - **Reduce the time in two ways**
    - **Distance Early Abandon**
      - reduce the Euclidean distance computation time between two time series
    - **Admissible Entropy Pruning**
      - after calculating some training samples, use an upper bound of information gain to check against the best candidate shapelet
The Shapelet Classifier

Shapelet Dictionary

<table>
<thead>
<tr>
<th>Shapelet</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shapelet</td>
<td>0.720</td>
</tr>
<tr>
<td>Nearest Neighbor</td>
<td>0.543</td>
</tr>
</tbody>
</table>

Shapelet Tree
Alternative approaches

• Transformations + k-NN
  ▫ Improved subsequence searching and matching, using online normalization, early abandoning, and re-ordering
  ▫ Dimensionality reduction using symbolic aggregate approximation

• Synthetic shapelet generation
  ▫ Initialize using K-means clustering
  ▫ Learn synthetic Shapelets by following a Gradient Ascend approach
Alternative approaches

- **Feature-based**
  - Select the top $k$ most informative shapelets
  - Generate a new dataset $D'$ of $k$ columns and $|D|$ rows, where each element $(i,j)$ is the (minimum) distance from the shapelet $s_j$ to data series $d_i$
  - Learn any suitable classifier (e.g., SVM, Random Forest) using the transformed dataset

<table>
<thead>
<tr>
<th></th>
<th>$s_1$</th>
<th>$s_2$</th>
<th>...</th>
<th>$s_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>0.3</td>
<td>3.3</td>
<td>...</td>
<td>0.1</td>
</tr>
<tr>
<td>$d_2$</td>
<td>0.2</td>
<td>3.2</td>
<td>...</td>
<td>3.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$d_n$</td>
<td>3.1</td>
<td>0.9</td>
<td>...</td>
<td>9.6</td>
</tr>
</tbody>
</table>
Random Shapelet Forest

- A tree-structured ensemble based on the classic Random Forest and the Shapelet Tree classifier

- **Learn:**
  - Build $T$ random shapelet trees
  - Each tree is built from a random (with replacement) sample of time series in the database $D$
  - Inspect $r$ random shapelets at each node

- **Predict:**
  - Let each tree $t_1, \ldots, t_T$ vote for a class label
Experimental results

- Parameters:
  - number of shapelets \((r) = 100\)
  - number of trees \((T) = 500\)
  - min shapelet length \((l) = 2\)
  - max shapelet length \((u) = m\) (max time series length)
Friedman’s test: The observed differences in accuracy of RSF against the rest deviate significantly, i.e., p-value = 10^{-11}
## Multi-variate time series

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Parameters</th>
<th>Parameters</th>
<th>Accuracy</th>
<th>Accuracy</th>
<th>Accuracy</th>
<th>Accuracy</th>
<th>Accuracy</th>
<th>gRSF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$r$</td>
<td>cDTW</td>
<td>SMTS</td>
<td>LPS</td>
<td>UFS</td>
<td>gRSF</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ArabicDigits</td>
<td>100</td>
<td>0.908</td>
<td>0.964</td>
<td>0.971</td>
<td>0.964</td>
<td>0.975</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AUSLAN</td>
<td>500</td>
<td>0.762</td>
<td>0.947</td>
<td>0.754</td>
<td>0.972</td>
<td>0.955</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CharacterT.</td>
<td>500</td>
<td>0.960</td>
<td>0.992</td>
<td>0.965</td>
<td>0.993</td>
<td>0.994</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LIBRAS</td>
<td>500</td>
<td>0.800</td>
<td>0.909</td>
<td>0.903</td>
<td>0.849</td>
<td>0.911</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECG</td>
<td>10</td>
<td>0.850</td>
<td>0.818</td>
<td>0.820</td>
<td>0.862</td>
<td>0.880</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CMU_MOOCAP_S16</td>
<td>10</td>
<td>0.931</td>
<td>0.997</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uWaveGestureLibrary</td>
<td>10</td>
<td>0.929</td>
<td>0.941</td>
<td>0.980</td>
<td>0.929</td>
<td>0.929</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wafer</td>
<td>500</td>
<td>0.977</td>
<td>0.965</td>
<td>0.962</td>
<td>0.976</td>
<td>0.992</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Japanese Vowels</td>
<td>500</td>
<td>0.649</td>
<td>0.969</td>
<td>0.951</td>
<td>0.932</td>
<td>0.800</td>
<td></td>
<td></td>
</tr>
<tr>
<td>KickvsPunch</td>
<td>500</td>
<td>0.900</td>
<td>0.850</td>
<td>0.900</td>
<td>0.700</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Walkvssrun</td>
<td>500</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NetworkFlow</td>
<td>10</td>
<td>0.712</td>
<td>0.974</td>
<td>0.965</td>
<td>0.891</td>
<td>0.914</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PEMS</td>
<td>100</td>
<td>0.832</td>
<td>0.896</td>
<td>0.844</td>
<td>0.988</td>
<td>1.000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pendigits</td>
<td>10</td>
<td>0.912</td>
<td>0.917</td>
<td>0.931</td>
<td>0.919</td>
<td>0.932</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>0.866</td>
<td>0.939</td>
<td>0.925</td>
<td>0.927</td>
<td>0.949</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rank</td>
<td>-</td>
<td>4.177</td>
<td>3.036</td>
<td>2.964</td>
<td>2.964</td>
<td>1.857</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Future work

• Feature tweaking for shapelets:
  ▫ What is the minimum amount of changes we need to apply to a shapelet feature so that the classification result flips?

• Interval sequence features:
  ▫ Apply random shapelet interval sequences
Thank you for your attention!
References

**Subsequence matching**


References

Time series classification

- "Generalized Random Shapelet Forests". In the Data Mining and Knowledge Discovery Journal (DAMI) 2016


References

Sequences of temporal intervals

• "Indexing Sequences of Temporal Intervals". In the Data Mining and Knowledge Discovery Journal (DAMI), Accepted 2016

• "Finding the Longest Common Sub-Pattern in Sequences of Temporal Intervals". In the Data Mining and Knowledge Discovery Journal (DAMI), 29(5): 1178-1210, 2015

• "Mining Frequent Arrangements of Temporal Intervals". In Knowledge and Information Systems (KAIS), Vol. 21, Issue 2, pages 133–171, 2010
Thank you for your attention!

Panagiotis Papapetrou

panagiotis@dsv.su.se