Mining of structures

Javier Béjar  BY:  $  ☕

CS - MAI

AMLT - 2016/2017
Outline

1 Introduction
   • Motivation

2 Sequences
   • Introduction
   • Clustering Sequences
   • Frequent patterns in sequences
   • Frequent Strings
   • Frequent transaction sets
   • Clustering of Data Streams

3 Application

4 Graph mining
   • Introduction
   • Mining large graphs
   • Mining sets of graphs
     • Clustering of graphs
     • Mining frequent Graphs
1 Introduction

2 Sequences

3 Application

4 Graph mining
Mining of structures

- There are some domains where patterns are more complex.
- In these domains instances are related to each other.
- Mining these relationships is more interesting than mining instances individually.
- For example:
  - Temporal domains
  - Relational databases
  - Structured instances (trees, graphs)
- Usually the methods used in this kind of domains are specific.
1 Introduction

2 Sequences

3 Application

4 Graph mining
Mining of sequences

- These domains suppose some kind of sequential relationship among instances (usually temporal)
- We can have a unique sequence or a set of sequences
- This kind of data can not be analyzed using classical techniques from time series analysis (ARIMA modeling, Kalman filter, ...)
- What makes different this kind of data?
  - Usually qualitative data
  - Very short series or a long series that has to be segmented
  - Interest in the relationships among series
  - Interest only in a part of the series (episodes, anomalies, novelty, ...)
Examples of sequence data

- Customer transactions over the time: Sets of transactions ordered by purchase time
- Web Data: Browsing activity of users in a web site
- Sensor data: History of events generated by a sensor (alarms, anomalies)
- Genome sequences: DNA/Protein sequences
Clustering of temporal series: Clustering algorithms applied to a set of short series

- Representation of the series, representation of the groups
- New distance measures (scale invariant, shape distances, ...)
- How to segment a unique series in a set of series? what parts are interesting?
Clustering Sequences - Data representation

- Segments (windows) of the series (overlapping/non overlapping)
  - Width of the window (series segmentation)

- Feature extraction: Generate informative features from the series windows:
  - Extreme points (maximum, minima, inflection points)
  - Frequency domain features (Fourier Transform, Wavelets, ...)
  - Chaos theory (time-lag transformation)
  - Probabilistic models (Hidden Markov Models)
Clustering Sequences - Data representation

Original Series

Discrete Fourier Coefficients

Discrete Wavelet Coefficients

1-Lag Series
Clustering Sequences - Distance Functions

- Usual distance functions ignore time dynamic
  - Euclidean, hamming, ...
- Patterns in series contain noise, time/amplitude scaling, translations
  - Dynamic Time Warping (DTW)
  - Longest Common Subsequence (LCSS)
  - Edit Distance with Real Penalty (ERP)
  - Edit Distance on Real Sequence (EDR)
  - Spatial Assembly Distance (SpADe)
Clustering Sequences - Distance Functions - DTW
Mining of sequences - Subpatterns

**Frequent sequence patterns:** Discovery of patterns inside the sequence

- The series does not need to have a temporal relationship (e.g., DNA sequences)
- The values can be continuous, discrete or structured (transaction data)
- Goals:
  - Discovery of subseries that reveal some causality of events
  - Discovery of abnormal/novel patterns (deviation from normal behavior)
  - Discovery of frequent patterns
Frequent Motifs

Keogh, Lonardi, Chiu *Finding Surprising Patterns in a Time Series Database In Linear Time and Space* (2002)

- Quantitative data, continuous time series
- The goal is to discover the most frequent patterns given a window length
- The trivial algorithm is $O(N^2)$
- To reduce computational complexity the windows are transformed to a simplified representation (Symbolic Aggregate Approximation/SAX)
- Distance over new representation is a lower bound of distance in original space
Frequent Motifs - SAX
**Frequent Motifs - Algorithm**

- A window of a specified length is used to segment the sequence (overlapped windows)
- The transformed sequence is stored in multiple hash tables using locality sensitive hashing
- Search for similar sequences:
  - Sequences appearing in the same or contiguous buckets in different hash tables are compared
  - First, compute the distance in the transformed series (cheap), if it is less than a threshold, then compute the exact distance
  - If sequences are near, store the sequences as candidate frequent motifs
- Return the candidates that appear more than a number of times
Frequent Strings

Vilo **Discovering Frequent Patterns from Strings** (1998)

- The sequence is a string of characters (no assumptions about the relationship among the elements of the sequence)
- We look for frequent patterns of any length (complete, with equivalent classes, with wildcards)
- These algorithms use specialized data structures to obtain the patterns within a reasonable computational cost (for example suffix-trees)
- The construction of these structures are sometimes a preprocess (later we extract the patterns) or given a threshold frequency we obtain only the patterns we are looking for
Frequent Strings - Algorithm

A suffix-trie is obtained from the sequence, a given \((k)\) parameter sets the minimum frequency of the patterns. The structure is created with the empty string.

1. For each character that belongs to the alphabet of the sequence
   - Compute a list with the next position to the occurrence of the character
   - If the character appears more than \(k\) times we create a new descendant in the trie

2. Recursive call for each descendant created
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 10 11 2
ABACABBAACA$  K=2

A=2,4,6,9,10,12
B=3,7,8
C=5,11

λ{1,2,3,4,5,6,7,8,9,10,11,12}
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACAC$ K=2

A=2,4,6,9,10,12
B=3,7,8
C=5,11

A=10
B=3,7
C=5,11

A=4,9
B=8
C=

A=6,12
B=
C=

λ {1,2,3,4,5,6,7,8,9,10,11,12}
A B C {5,7} {2,4,6,9,10,12} {3,7,8}
Frequent Strings - Example

ABACABBAACA$  K=2

A=2,4,6,9,10,12
B=3,7,8
C=5,11

A=10
B=3,7
C=5,11

A\{2,4,6,9,10,12\}
B\{3,7,8\}
C\{5,7\}

A=4
B=8
C=

A=6,12
B=
C=

A=10
B=7
C=
Frequent Strings - Example

1 2 3 4 5 6 7 8 9 101112
ABACABBAACA$  K=2  {A,B,C,AB,AC,BA,CA,ACA}

A=2,4,6,9,10,12
B=3,7,8
C=5,11

λ{1,2,3,4,5,6,7,8,9,10,11,12}

A \{2,4,6,9,10,12\}  B \{3,7,8\}  C \{5,7\}

A=10
B=3,7
C=5,11

B \{3,7\}  C \{5,11\}

A=4
B=8
C=

A=6,12
B=
C=

A \{6,12\}
Frequent Transactions

- Sometimes the events in the sequence are sets of elements
- The definition of sequence has to be adapted:
  - Given an ordered list of elements:
    
    \[ s = \langle e_1 e_2 e_3 \ldots \rangle \]
  - Each element contains a collection of items
    
    \[ e_i = \{i_1, i_2, \ldots, i_k\} \]
- Also the definition of subsequence:
  - A sequence \( \langle a_1, a_2, \ldots, a_n \rangle \) is contained in another sequence \( \langle b_1, b_2, \ldots, b_m \rangle \) (\( m \geq n \)) if exists integers \( i_1 < i_2 < \ldots i_n \) such that
    
    \[ a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, a_n \subseteq b_{i_n} \]
Generalized Sequence Pattern (GSP)

Agrawal, Srikant, **Mining Sequential Patterns** (1995)
Srikant, Agrawal, **Mining Sequential Patterns: Generalizations and Improvements** (1996)

- An apriori approach to the mining of sequential patterns
- A similar monotonic property exists among a sequence and its subsequences:
  - Given a sequential pattern $s$ all its subsequences are also sequential patterns
  - Given a sequential pattern $s$ if $s'$ is a subsequence then $\text{support}(s) \leq \text{support}(s')$
- This means that generating all the frequent sequences for a database can be done using a sequential approach
Generalized Sequence Pattern (GSP)

- Each iteration the sequences of length \( k + 1 \) are generated by merging the frequent sequences of length \( k \).
- For the generation of candidates all items in each transaction are listed alphabetically.
- Each iteration the database is scanned to test for the candidates and compute their support.
- Sequences not frequent are pruned.
Generalized Sequence Pattern (GSP)

- Case 1:
  \[(ab)(ac)ab + (ab)(ac)ac \Rightarrow (ab)(ac)a(bc), (ab)(ac)abc, (ab)(ac)acb\]

- Case 2:
  \[(ab)(ac)(ab) + (ab)(ac)(ac) \Rightarrow (ab)(ac)(abc)\]

- Case 3:
  \[(ab)(abc) + (ab)(ab)a \Rightarrow (ab)(abc)a\]
Clustering of Data Streams

Data streams: Modeling an on-line continuous series of data

- Each item of the series is an instance (one value, a vector of values, structured data)
  - For instance, sensory data (one or multiple synchronized data), stream of documents (twitter/news)

- Data is generated from a set of clusters (stable or changing over time)
  - For example, states from a process or semantic topics

- The data is processed incrementally (model changes with time)
  - Only the current model
  - Periodic snapshots

- Different goals: Model the domain, detect anomalies/novelty/bursts, change (Concept drift)
Clustering of Data Streams - Micro clusters


- **On-line phase:**
  - Maintains/creates microclusters
  - Number of microclusters is larger than the actual number of clusters of the dataset (approximate the densities)
  - When new data arrives, it is incorporated to a microcluster or generates new microclusters
  - The number of microclusters is fixed, so microclusters are merged to maintain the number
  - Periodically the microclusters are stored

- **Off-line phase:**
  - Given a time window the stored clusters are used to compute the microclusters inside the time frame
  - A k-means algorithm is used to compute the clusters for the time window
Clustering of Data Streams - Density Based

Cao, Ester, Qian, Zhou, *Density-Based Clustering over an Evolving Data Stream with Noise* Proceedings of the Sixth SIAM International Conference on Data Mining, 2006

- **On-line phase:**
  - Core-micro-clusters (a weighted sum of close points)
  - The weight of a point fades exponentially with time (damping model)
  - New examples are merged and mc are classified as:
    - core-mc, sets of points with weight over a threshold
    - potential-mc
    - outlier-mc, sets of points with weight below a threshold
  - outlier-mc disappear with time

- **Off-line phase:** Modified version of DBSCAN
Clustering of Data Streams - Density Based
1 Introduction

2 Sequences

3 Application

4 Graph mining
Spinal Cord Signals Analysis

- Recorded signals over the lumbar vertebrae of an anesthetized cat (from L4 to L7)
- Different experimental conditions
- Study of the peaks generated spontaneously by groups of neurons
- Study of the synchronizations of the peaks in the different recording points
- Study of the sequential pattern behavior
Spinal Cord Signals Analysis

Peak Data
**Spinal Cord Signals Analysis**

**Shapes dictionary**

- **Goal:** Determine an structure for the peaks

- **Process**
  - Identify the peaks in the signals (peak finding algorithm)
  - The peaks extracted from each measuring point will be the first dataset
  - Clean/Transform the data to visualize the structure
  - Cluster the datasets to obtain patterns of peaks (k-means → determine the number of clusters)

- **Results:** A tentative dictionary of peaks shapes
Spinal Cord Signals Analysis

Clusters
Spinal Cord Signals Analysis

Synchronization Patterns

- **Goal:** Find patterns in the synchronizations of the peaks
- **Process**
  - Determine the synchronizations in the signal (synchronization algorithm)
  - Generate a database of transactions from the synchronizations
  - Apply a frequent transactions algorithm to the transactions database
- **Results:** A set of frequent sequences of synchronization
Spinal Cord Signals Analysis
Finding Synchronizations
Spinal Cord Signals Analysis

Synchronization sequences
1 Introduction

2 Sequences

3 Application

4 Graph mining
Mining of Structures

- There is a lot of information that has a relational structure.
- Methods and models used for unstructured data are not expressive enough.
- Sometimes structure can be flattened, but lots of interesting information is lost.
  - Relational database $\Rightarrow$ unique merged table
  - Attributes representing relations $\Rightarrow$ inapplicable attributes
  - Graph data $\Rightarrow$ strings based on graph traversal algorithms
  - Documents $\Rightarrow$ bag of words
Mining of Structures (WWW/Social networks)
Mining of Structures (Relational databases/Ontologies)
Mining of Structures (XML documents/Text)
Mining of Structures (Chemical compounds/Gene interactions)
Mining of Structures

- All these types of data have in common that can be represented using graphs and trees
- Algorithms that use these data structures as input are needed
- Historically we can find different approaches to the discovery of patterns in graphs/trees:
  - Inductive logic programming: Structure is represented using logic formulas
  - Graph algorithms
    - Classic algorithms for detecting dense subgraphs (cliques)
    - Graph isomorphism algorithms
    - Graph partitioning algorithms
Mining of Structures: Computational issues

- Most of the problems used to discover structures in graphs are NP-Hard
  - Graph partitioning (Not for bi partitioning)
  - Graph isomorphism
- Other approaches for mining graphs
- Two different problems:
  - Mining large graphs (only one structure) ⇒ Partitioning, dense subgraphs
  - Mining sets of graphs ⇒ common substructures
Mining Large Graphs

- Some information can be described as a large graph (several instances connected by different types of relations)
  - For example: Social networks, Web pages,
- We are interested in discovering interesting substructures by:
  - Dividing the graph in subgraphs (k-way partitioning, node clustering)
  - Extracting dense substructures
Graph partitioning (2-way)

- The simplest partitioning of a graph is to divide the graph in two subgraphs.
- We assume that edges have values as labels (similarity, distance, ...)
- This problem is the minimum cut-problem:
  “Given a graph, divide the set of nodes in two groups so the cost of the edges connecting the nodes between the groups is minimum”
- This problem is related to the maximum flow problem that can be solved in polynomial time.
Graph partitioning (k-way)

- The general problem is NP-hard
- It can be solved approximately by local search algorithms (hill climbing)

Kerninghan-Lin Algorithm:
1. Start with a random cut of the graph
2. Interchange a pair nodes from different partitions that reduces the cut
3. Iterate until convergence

- There are different variations of this algorithm that changes the strategy for selecting the pair of nodes to interchange
Graph partitioning (k-way) - Clustering

Clustering algorithms can be adapted to obtain a graph partition:

- **K-means and K-medoids variations:**
  - Nodes of the graphs as prototypes
  - Objective functions to define node membership to clusters (geodesic distance)
  - Network structure indices

- **Spectral Clustering**
  - Define the Laplacian matrix from the graph
  - Perform the eigendecomposition
  - The largest Eigenvalues determine the number of clusters
Graph partitioning (k-way) - Clustering

- **Girvan-Newman Algorithm (Social Networks)**
  - Is based on the concept of *edge betweenness centrality*:
    \[
    B(e) = \frac{\text{NumConstrainedPaths}(e, i, j)}{\text{NumShortPaths}(i, j)}
    \]
  - *Random Walk Betweenness*: Compute how often a random walk starting on node *i* passes through node *j*
  - Detects bridges among dense components (edges that are not in the shortest paths between pairs of nodes)
- **Algorithm:**
  1. Rank edges by $B(e)$
  2. Delete edge with the highest score
  3. Iterate until a specific criteria holds (eg. number of components)
Dense subgraphs

- Not always a complete partitioning of the graph is necessary
- Dense subgraphs can represent interesting behaviors
- Different types of (pseudo)dense subgraphs:
  - **Clique**: All nodes are connected
  - **Quasi-clique**: Almost all nodes are connected (minimum density or minimum degree)
  - **K-core**: Every node connects to at least $k$ nodes
  - **K-plex**: Each node is missing no more than $k-1$ edges
  - ...
- Different goals: Minimum size, all or the best ranked, overlapping or not
Mining Sets of Graphs

- Some information can be described as a collection graph
  - For example: XML documents, chemical molecules

- We can use two approaches:
  - Cluster the graphs for common patterns and summarization
  - Find frequent substructures
Clustering of Graphs

- We look at a graph as a complex object
- We have to adapt the elements of clustering algorithms to this objects:
  - Distance measures to compare graphs
  - Summarization of graphs as prototypes
Clustering of Graphs - XProj

- This algorithm partitions a set of XML documents in $K$ classes.
- The summarization and similarity are based on frequent substructures.
- For each partition the frequent substructures of size $l$ are computed and used as prototype.
- The number of substructures of the prototypes in a graph is used as similarity.
- The more expensive part is the computation of the substructures (cost is reduced by linearizing the graphs).
## Clustering of Graphs - XProj

**Algorithm: XProj**

Partition the documents in K random subsets
Compute the prototypes (frequent substructures length $l$) $S_k$

**repeat**

Assign each document to the most similar prototype ($S_k$)
/* let $M_1, \ldots M_k$ be the new partitions */

**foreach** $M_i$ do

Recompute the most frequent substructures of length $l$ of $M_i$

if frequency of substructures $> min_{sup}$ then

| Modify the prototype |

end

end

**until** convergence criteria
Mining frequent Graphs/Trees

- We look for frequent graphs/trees in a database
- Usually the structures are transformed to some kind of canonical representation (adjacency matrix, tree traversal)
- This representation gives a unique code for each different graph
- We can discover patterns in this code related to the patterns that appear in the original graphs
- Some of the approaches use the same properties used in association rules increasing the size of the patterns until no pattern is found
gSpan

Yan, Han **gSpan: Graph-Based Substructure Pattern Mining**
Proceedings of the IEEE International Conference on Data Mining 2002

- The graphs have labels in their edges and their vertices
- A canonical representation is used to reduce the cost to compute graph isomorphism
- This representation is based on the tree obtained by the depth first search of the graph and a lexicographical order among labels
- This representation transforms a graph into a string that contains the labels of the graph
- With this representation we can have an ordering over all the graphs that can be used to explore all possible subgraphs
**gSpan - DFS Lexicographic order**

- We assume an initial vertex ($v_0$) and an order among vertices ($i = 0 \ldots n$), many DFS traversals of a graph can obtained

- For a DFS traversal of a graph we define the forward edges (those with $i < j$, the DFS tree of the graph) and the backward edges (those with $i > j$)

- We define a linear order among edges, given $e_1 = (i_1, j_1)$ and $e_2 = (i_2, j_2)$
  - if $i_1 = i_2$ and $j_1 < j_2$ then $e_1 \prec_T e_2$; if $i_1 < j_1$ and $j_1 = i_2$ then $e_1 \prec_T e_2$; if $e_1 \prec_T e_2$ and $e_2 \prec_T e_3$ then $e_1 \prec_T e_3$

- Given a DFS tree for a graph and edge sequence can be defined based on $\prec_T$, this is the DFS code of the graph

- Given the DFS codes for a graph the linear order and a linear order for its labels $\prec_L$, a linear order among codes can be defined

- The *Minimum DFS code* is the **canonical label** for $G$
gSpan - DFS Lexicographic order

DFS DFS Code

0: (0,1,X,a,X)
1: (1,2,X,a,Y)
2: (2,0,Y,b,X)
3: (2,3,Y,b,Z)
4: (3,0,Z,c,X)
5: (2,4,Y,d,Z)
**gSpan - Algorithm**

**Initialization** \((D, \text{MinSup})\)
1. sort labels of the vertices and edges in \(D\) by frequency
2. remove infrequent vertices and edges
3. \(S_0 = \text{code of all frequent graphs with single edge}\)
4. sort \(S_0\) in DFS lexicographic order
5. \(S = S_0\)
6. for each code \(s\) in \(S_0\)
   1. \(\text{gSpan}(s, D, \text{MinSup}, S)\)
   2. \(D = D - s\)
   3. if \(|D| < \text{MinSup}\) then return

**gSpan\((s, D, \text{MinSup}, S)\)**
1. if \(s \neq \text{mincode}(s)\) then return
2. insert \(s\) into \(S\)
3. \(C = \emptyset\)
4. scan \(D\)
   - find every edge \(e\) such that \(s\) can be right-most extended to frequent \(s \ast e\)
   - insert \(s \ast e\) into \(C\)
5. sort \(C\) in DFS lexicographic order
6. for each \(s \ast e\) in \(C\) do
   1. \(\text{gSpan}(s \ast e, D, \text{MinSup}, S)\)
Graph mining

Mining of sets of graphs

gSpan - Example

Minsup = 2

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal

Not minimal
Python Notebooks

This Python Notebook has examples of dense subgraphs discovery and community discovery using geolocation information from Twitter for London, Paris and Barcelona

- Dense Subgraphs Notebook (click here to go to the url)

If you have downloaded the code from the repository you will able to play with the notebooks (run jupyter notebook to open the notebooks)