

# **EPL451: Data Mining on the Web – Lab 8**



University of Cyprus  
Department of  
Computer Science

---

Παύλος Αντωνίου

Γραφείο: B109, ΘΕΕ01



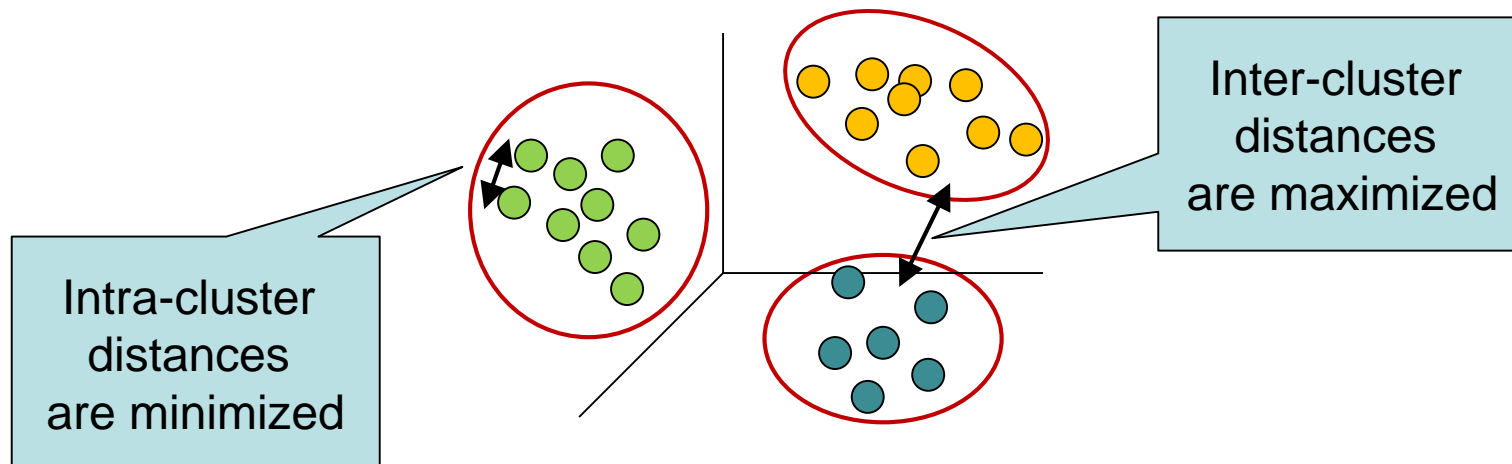
# **Text Clustering & Classification**

---

# Text Clustering Problem



- Input: a set of **text-based**, **un-labeled** documents (e.g. newspaper articles, emails, movie reviews, movie abstracts)
- Output: **partition** unlabeled unclustered docs **into** disjoint subsets – **clusters** – (hard clustering) such that:
  - Docs within a cluster are very similar
  - Docs in different clusters are very different



# Text Classification Problem

---



- Input: a set of **text-based**, **labeled** documents (e.g. newspaper articles, emails, movie reviews, movie abstracts)
  - Output: **choose correct class label** for each unclassified doc
-

# Text Clustering Applications

---



- Improving search applications
    - Improving search recall
      - When a query matches a document its whole cluster can be returned
    - Better user interface and navigation: search without typing
    - Speeding up vector space retrieval
      - Cluster-based retrieval gives faster search
  - Forensic data analysis
    - discover and analyze patterns of fraudulent activities in a large set of unstructured text files (emails, log files, etc.)
-

# Text Clustering Applications

---



- Detect the current hot topics on twitter
    - Find what people are talking about (generally or in a specific geographic area)
  - Find what topics people at a conference talk about, such as, what paper they liked most or who to network with as likes similar topics
  - Use the cluster to pre-populate suggest-box to autocomplete tags when users type
  - Cluster movies based on abstract and description and show related movies (augment recommendations)
-

# Preprocessing steps

---



- Obtain dataset
- Clean dataset
  - Remove unneeded information from documents e.g. html tags, numbers
  - Convert to lowercase
- Tokenization
  - Parse documents into smaller units (tokens) such as words and phrases (n-grams)
    - Token separators: whitespaces and punctuation
  - Create vocabulary
- Remove stop words
  - Stop words: frequently occurring words that don't carry much meaning e.g. and, of, in, ...

# Preprocessing steps

---



- Stemming and lemmatization
  - Different tokens might carry out similar information (e.g. tokenization and tokenizing)
  - Avoid calculating similar information repeatedly by reducing all tokens to its base form using various stemming and lemmatization dictionaries
- Feature extraction
  - Transform dataset in a format supported by machine learning algorithms → create features for each document
    - e.g. document to a numerical vector: [0,2,1,0,0,7]
  - **bag of words** model, **tf/idf** model (see appendix)
- Feature selection
  - singular vector decomposition (SVD), principal component analysis (PCA)



# Machine Learning Methods

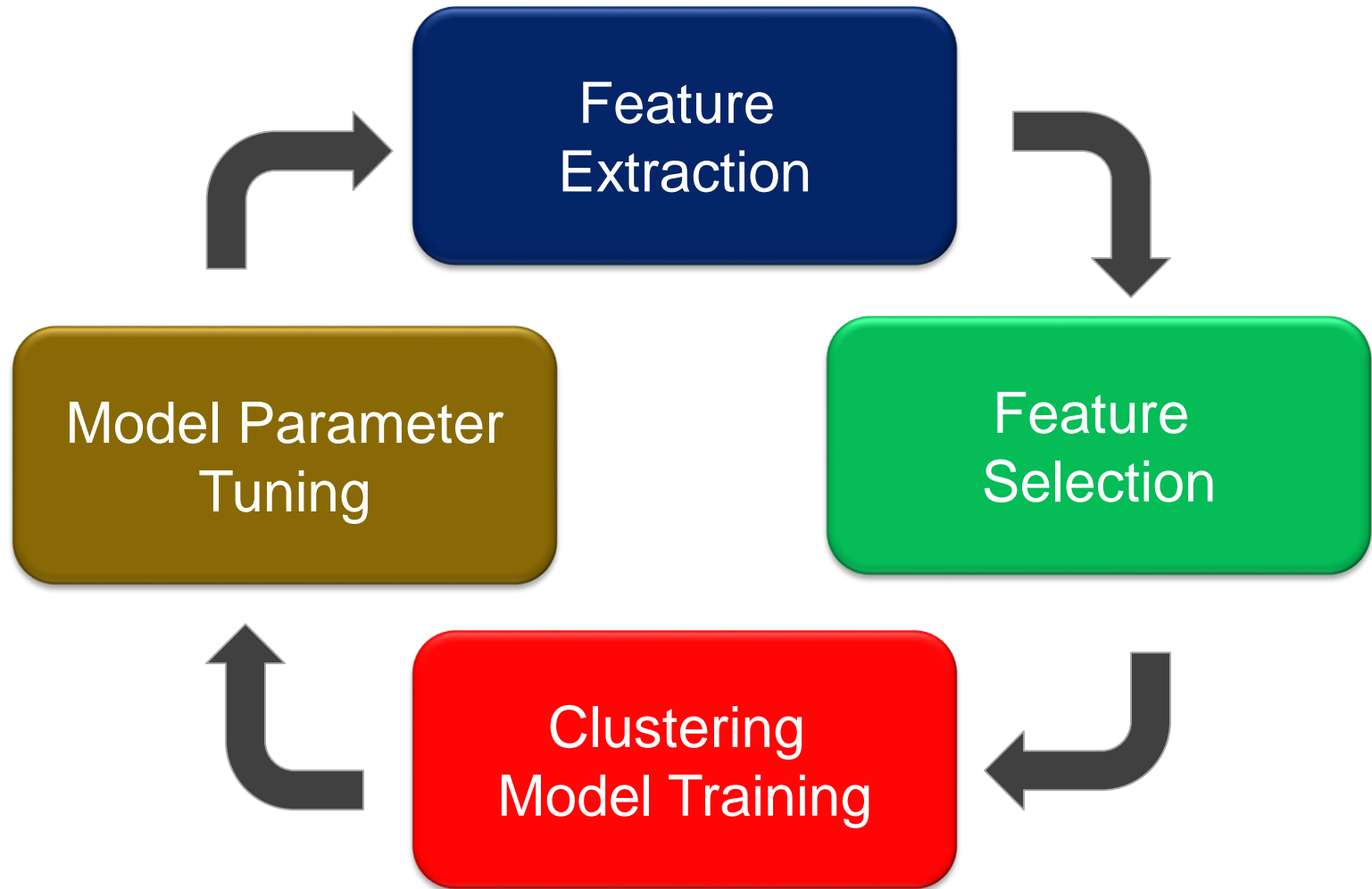
---



- Standard (text) clustering methods:
    - K-means
    - Bisecting K-means [N/A sklearn; download [here](#)]
    - Agglomerative Hierarchical Clustering
  - Standard (text) classification methods:
    - Support Vector Classifier
    - Random Forest (decision tree) Classifier
    - Naïve Bayes Classifier
-

# Text clustering/classification process

---



# Hands-on

---



- Install Natural Language Toolkit that involves tokenizers, stopwords, stemmers, datasets, etc
    - `conda install nltk`
  - Html parsing library
    - `conda install beautifulsoup4`
  - Download [lab tutorial](#) and go through the steps
-



---

Vector Space Model: Bag of words & tf/idf models

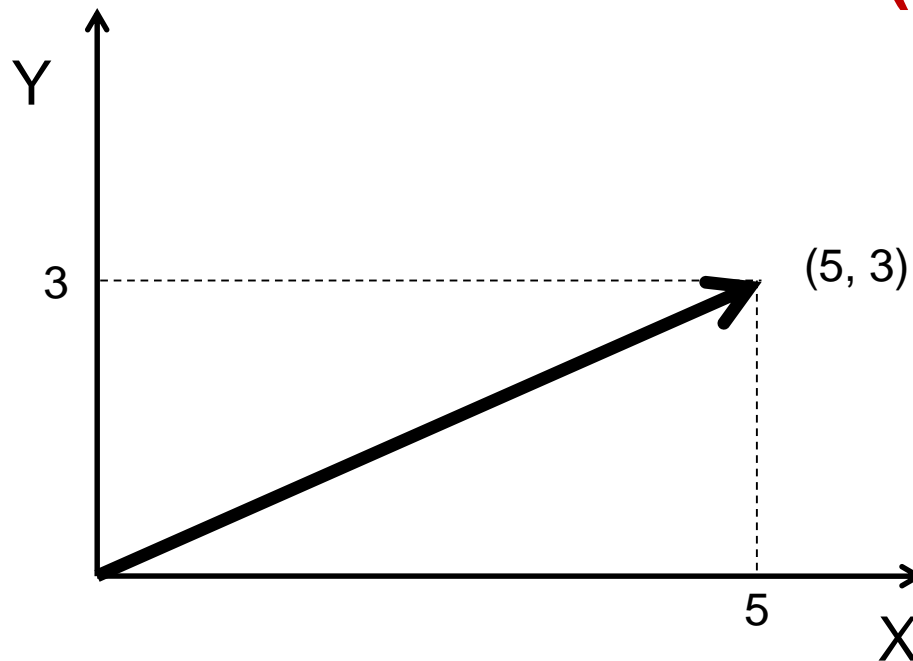
# Appendix

---

# Data as Vectors



- Machine learning text clustering/classification methods require vectors of numbers → **convert raw text documents into vectors (vector space model)**



- The vector denoted by point (5, 3) is simply  $5x + 3y$ 
  - weights
  - `Array([5, 3])` or `HashMap([0 => 5], [1 => 3])`

# Data as Vectors



- Document vector:  $v = a_1 * v_1 + a_2 * v_2 + \dots$ 
  - $a_1, a_2, \dots$  : weights
  - $v_1, v_2, \dots$  : components (terms in document vectors)
- The **weight** of a component of a document vector can be represented by **term frequency (tf)** and combination of term frequency and **inverse document frequency (idf)**
- Term Frequency denoted by  $tf$ , is the number of occurrences of a term  $t$  in the document  $D$ 
  - E.g. given document “hello world hello”,  
document vector =  $2 * \text{“hello”} + 1 * \text{“world”}$
  - This is the **Bag of words model**

# Data as Vectors

---



- Problem: all terms are equally important
  - certain terms have little or no discriminating power in determining relevance on a query
    - e.g. collection of documents on the auto industry is likely to have the term auto in almost every document
- Inverse Document Frequency of a term  $t$ , denoted by  $\text{idf}$ , is  $\log(N/\text{df})$ , where:
  - $N$ : total number of documents in the space,
  - $\text{df}$ : total number of documents that contain the term  $t$
- small  $\text{idf}$ :
  - a term occurs in many documents
- high  $\text{idf}$ :
  - a term occurs in a small number of docs

# Vectors: the tf-idf model

---



- The combination of tf and idf is the most popular weight used in case of documents similarity exercises
  - $\text{tf-idf}_{t,d} = \text{tf}_{t,d} * \text{idf}_t$
  - Weight is the highest, when term t occurs many times within a small number of documents
  - Weight is the lowest, when term t occurs fewer times in a document or occurs in many documents
-



# tf-idf example



- D1 = “Shipment of gold damaged in a fire”
- D2 = “Delivery of silver arrived in a silver truck”
- D3 = “Shipment of gold arrived in a truck”

Document vector of D1

| Terms    | tf <sub>i</sub> |    |    | df <sub>i</sub> | N/df <sub>i</sub> | IDF <sub>i</sub> | Weights = tf <sub>i</sub> * IDF <sub>i</sub> |        |        |
|----------|-----------------|----|----|-----------------|-------------------|------------------|--|--------|--------|
|          | D1              | D2 | D3 |                 |                   |                  | D1   | D2     | D3     |
| a        | 1               | 1  | 1  | 3               | 1                 | 0                | 0.0000                                       | 0.0000 | 0.0000 |
| arrived  | 0               | 1  | 1  | 2               | 1.5               | 0.1761           | 0.0000                                       | 0.1761 | 0.1761 |
| damaged  | 1               | 0  | 0  | 1               | 3                 | 0.4771           | 0.4771                                       | 0.0000 | 0.0000 |
| delivery | 0               | 1  | 0  | 1               | 3                 | 0.4771           | 0.0000                                       | 0.4771 | 0.0000 |
| gold     | 1               | 0  | 1  | 2               | 1.5               | 0.1761           | 0.1761                                       | 0.0000 | 0.1761 |
| fire     | 1               | 0  | 0  | 1               | 3                 | 0.4771           | 0.4771                                       | 0.0000 | 0.0000 |
| in       | 1               | 1  | 1  | 3               | 1                 | 0                | 0.0000                                       | 0.0000 | 0.0000 |
| of       | 1               | 1  | 1  | 3               | 1                 | 0                | 0.0000                                       | 0.0000 | 0.0000 |
| shipment | 1               | 0  | 1  | 2               | 1.5               | 0.1761           | 0.1761                                       | 0.0000 | 0.1761 |
| silver   | 0               | 2  | 0  | 1               | 3                 | 0.4771           | 0.0000                                       | 0.9542 | 0.0000 |
| truck    | 0               | 1  | 1  | 2               | 1.5               | 0.1761           | 0.0000                                       | 0.1761 | 0.1761 |

# Improving search recall



- *Cluster hypothesis* - Documents in the same cluster behave similarly with respect to relevance to information needs
- Therefore, to improve search recall:
  - Cluster docs in corpus a priori
  - When a query matches a doc  $D$ , also return other docs in the cluster containing  $D$
- Hope if we do this: The query “car” will also return docs containing *automobile*
  - Because clustering grouped together docs containing *car* with those containing *automobile*. Why?



# Speeding up vector space retrieval

---



- Using vector space retrieval model, documents closest to the query need to be found
- Calculate the similarity of the query to each document in the corpus
  - Very slow!
- Cluster docs in corpus a priori
  - Calculate similarity of the query to centroids
  - Return

