TEXT CLASSIFICATION

Prof. Marco Viviani

marco.viviani@unimib.it

THE TASK

What classification is and what is not

- **Classification** (a.k.a. "categorization"): a ubiquitous enabling technology in data science.
  - Studied within pattern recognition, statistics, and machine learning.

- **Definition**: the activity of predicting to which, among a predefined finite set of groups ("classes", or "categories"), a data item belongs to.

- Formulated as the task of generating a hypothesis (or "classifier", or "model"):

$$h: D \rightarrow C$$

  where
  - $D = \{x_1, x_2, \ldots\}$ is a domain of data items.
  - $C = \{c_1, c_2, \ldots, c_n\}$ is a finite set of classes (the classification scheme).

What classification is and what is not

- The membership of a data item into a class must not be determinable with certainty.
  - E.g., predicting whether a natural number belongs to `Prime` or `NonPrime` is not classication).

- In **text classification**, data items are textual:
  - E.g., news articles, e-mails, tweets, product reviews, sentences, questions, queries, etc.

or partly textual:
  - E.g., Web pages.

Main types of classication

Binary classification

- $h: D \rightarrow C$ (each item belongs to exactly one class).
- $C = \{c_1, c_2\}$.
  - E.g., assigning emails to `Spam` or `Legitimate`.

Single-Label Multi-Class (SLMC) classification

- $h: D \rightarrow C$ (each item belongs to exactly one class).
- $C = \{c_1, c_2, \dots c_n\}$, with $n > 2$.
  - E.g., assigning news articles to one of `HomeNews`, `International`, `Entertainment`, `Lifestyles`, `Sports`.

Main types of classication

Multi-Label Multi-Class (MLMC) classification

- $h: D \rightarrow 2^C$ (each item may belong to zero, one, or several classes).
- $C = \{c_1, c_2, \ldots c_n\}$, with $n > 1$.
  - E.g., assigning computer science articles to classes in the ACM Classification System.
  - May be solved as n independent binary classification problems.

Ordinal classification (OC)

- As in SLMC, but for the fact that there is a total order $c_1 \preccurlyeq c_2 \preccurlyeq \cdots \preccurlyeq c_n$ on $C = \{c_1, c_2, \ldots c_n\}$.
  - E.g., assigning product reviews to one of `Excellent`, `Good`, `SoAndSo`, `Poor`, `Disastrous`.

Hard classification and soft classification

- The previous definitions denote "**hard classification**" (HC).

- "**Soft classication**" (SC) denotes the task of predicting a score for each pair (d, c), where the score denotes the {probability / strength of evidence / confidence} that d belongs to c.

Hard classification and soft classification

- **Hard classification** often consists of:
  1. Training a soft classifier that outputs scores $s(d, c)$.
  2. Picking a threshold t, such that:
     - $s(d, c) \geq t$ is interpreted as predicting c_1.
     - $s(d, c) < t$ is interpreted as predicting c_2.

- In **soft classification**, scores are used for ranking.
  - E.g., ranking items for a given class, ranking classes for a given item.

- HC is used for fully autonomous classifiers, while SC is used for interactive classifiers (i.e., with humans in the loop).

Dimensions of text classification

Text classification may be performed according to several dimensions ("axes") orthogonal to each other.

- By **topic**: by far the most frequent case, its applications are ubiquitous.
- By **sentiment**: useful in market research, online reputation management, customer relationship management, social sciences, political science.
- By **language** (a.k.a. "language identification"); useful, e.g., in query processing within search engines.

Dimensions of text classification

- By **type**: e.g., `AutomotiveNews` vs. `AutomotiveBlogs`, useful in website classification and others.
- By **author** (a.k.a. “authorship attribution”).
- By **native language** (“native language identification”).
- By **gender**: useful in forensics and cybersecurity.
- …

APPLICATIONS OF TEXT CLASSICATION

Example 1. Knowledge organization

- Long tradition in both science and the humanities.
  - The goal was organizing knowledge, i.e., **conferring structure** to an otherwise unstructured body of knowledge.

- The rationale is that using a structured body of knowledge is easier/more effective than if this knowledge is unstructured.

- Automated classification tries to automate the tedious task of assigning data items based on their content, a task otherwise performed by human annotators (or “assessors”, or “coders”).

Example 1. Knowledge organization

- **Scores of applications** (examples):
  - Classifying news articles for selective dissemination
  - Classifying scientific papers into specialized taxonomies
  - Classifying patents
  - Classifying "classified" ads
  - Classifying answers to open-ended questions
  - Classifying topic-related tweets by sentiment
  - ...

- **Retrieval** (as in search engines) could also be viewed as (binary + soft) classification into `Relevant` vs. `NonRelevant`, but mostly soft → Ranking.

Example 2. Filtering

- **Filtering** (a.k.a. "routing") refers to the activity of blocking a set of `NonRelevant` items from a dynamic stream, thereby leaving only the `Relevant` ones.

- E.g., when studying the sentiment of Twitter users towards `Donald Trump`, tweets that are not about `Donald Trump` must be "filtered out".

- Filtering is thus an instance of binary (hard) classification, and its applications are ubiquitous.

Example 2. Filtering – Applications

- **Spam filtering** is an important example of filtering, attempting to tell `Legitimate` messages from `Spam` messages.

- **Detecting unsuitable content** (e.g., violent content, racist content, cyberbullying, fake news) is also an important application, e.g., in PC filters or on interfaces to social media.

Example 3. Empowering other IR tasks

- Functional to improving the effectiveness of other tasks in IR or NLP.

- Some examples:
  - Classifying queries by intent within search engines
  - Classifying questions by type in QA systems
  - Classifying named entities
  - Word sense disambiguation in NLP systems
  - ...

- Many of these tasks involve classifying very small texts (e.g., queries, questions, sentences).

SUPERVISED LEARNING AND TEXT CLASSICATION

Before supervised learning

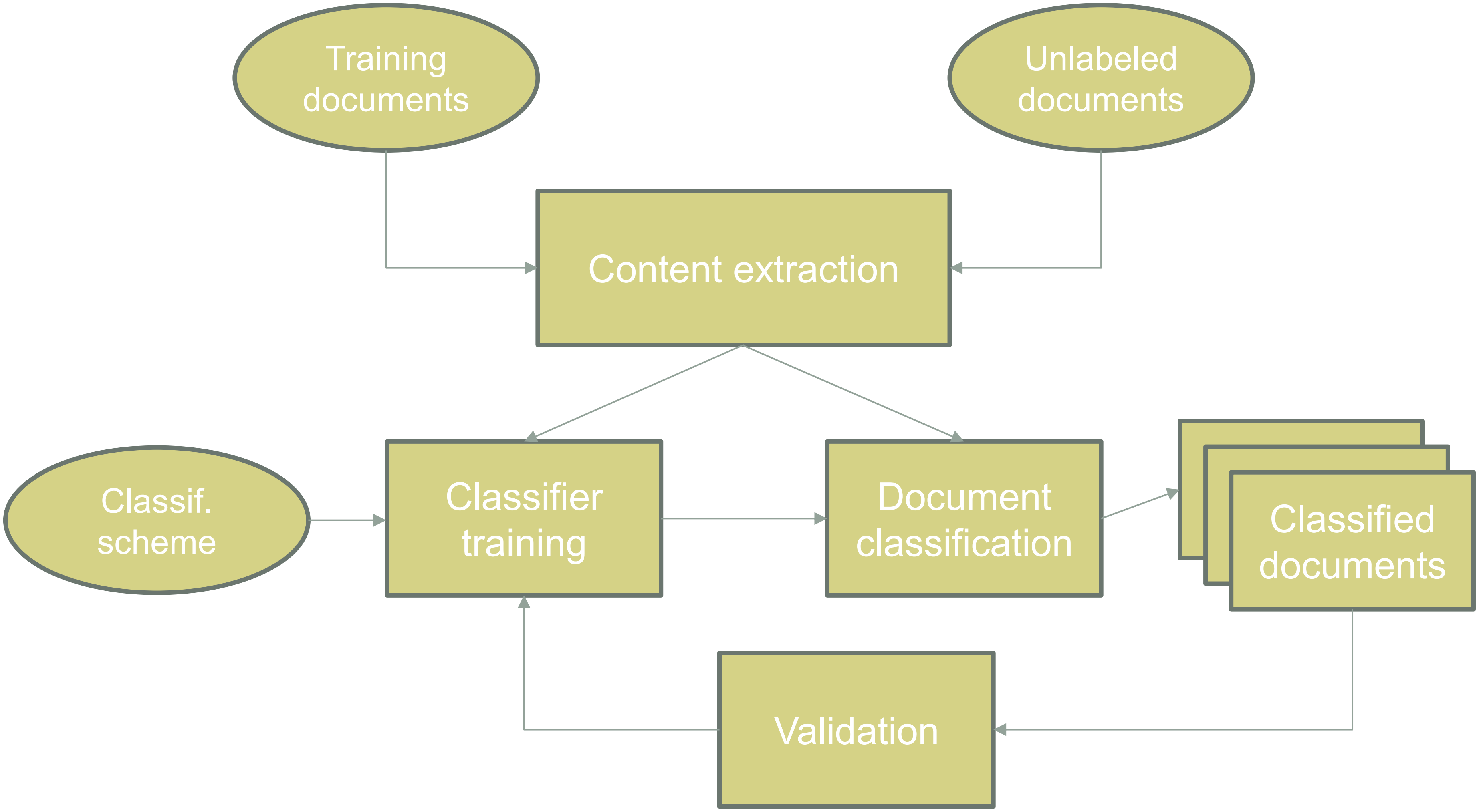
- An old-fashioned way to build text classifiers was via knowledge engineering, i.e., manually building **classification rules**.
  - E.g., (`Viagra` or `Sildenafil` or `Cialis`) → `Spam`
- Disadvantages: expensive to setup and to maintain.

Supervised learning and classification

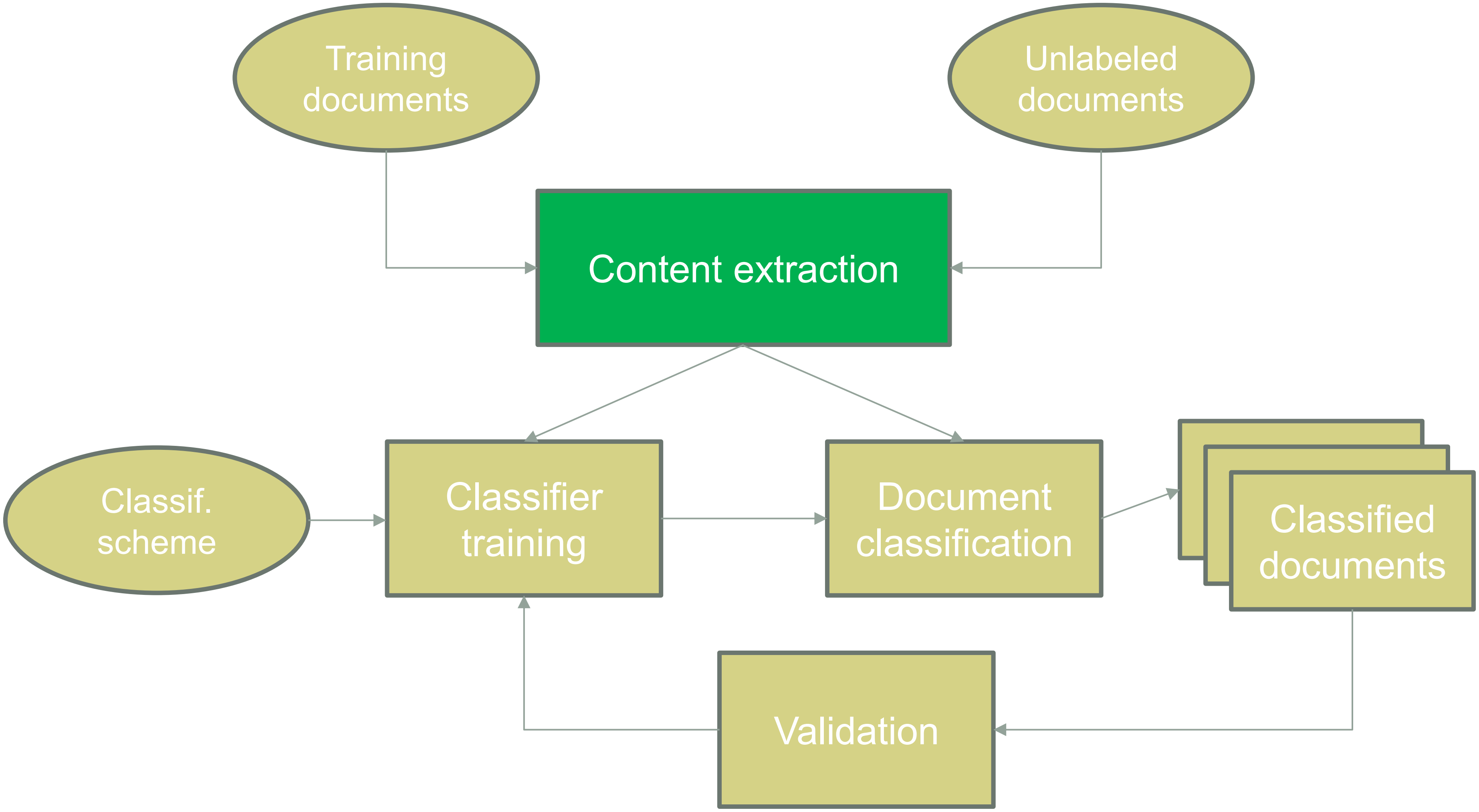
- **Supervised learning (SL)** approach:
  - A generic (task-independent) learning algorithm is used to train a classifier from a set of manually classified examples.
  - The classifier learns, from these training examples, the characteristics a new text should have in order to be assigned to class c.

- Advantages:
  - Generating training examples cheaper than writing classification rules.
  - Easy update to changing conditions (e.g., addition of new classes, deletion of existing classes, shifted meaning of existing classes, etc.).

Supervised learning and classification



Supervised learning and classification



Text representation

Representing text for classification purposes

- In order to be input to a learning algorithm (or a classifier), all training (or unlabeled) documents are converted into **vectors** in a common vector space.

- The dimensions of the vector space are called **features**, and the number k of features used is called the dimensionality of the vector space.

Representing text for classification purposes

In order to generate a vector-based representation for a set of documents D, the following steps need to be taken:

- **Feature Extraction**
- **(Feature Selection or Feature Synthesis)**
- **Feature Weighting**

Feature extraction

- Feature extraction in text classification is the process of converting raw text data into a numerical or categorical format that can be used as input for machine learning algorithms.

- Text data, unlike structured data, cannot be directly used for classification tasks because machine learning algorithms typically work with numerical data.

- Feature extraction is crucial in text classification as it transforms the text into a format that can be processed effectively by algorithms.

Feature extraction – Unigrams (1)

- In classification **by topic**, a typical (simplest) choice is to make the set of features coincide with the set of words that occur in the training set (unigram model, a.k.a. "Bag-of-Words").
  - This may be preceded by (a) stop words removal and/or (b) stemming or lemmatization; (b) is meant to improve statistical robustness.

- The dimensionality k of the vector space is the number of words (or stems, or lemmas) that occur at least once in the training set, and can easily be $O(10^5)$ or even $O(10^6)$.

Feature extraction – Unigrams (2)

- Each **document** usually contains $O(10^5)$ unique words!
  - If we indicate the absence of a word from a document by 0, this means that these vectors are usually very sparse.

- **Vector sparsity** and **high dimensionality** are possibly the <u>two most important characteristics</u> that distinguish text classification from other instantiations of classification (e.g., in data mining).

- The unigram representation renounces to encoding word order and syntactic structure.
  - Unigrams → ["the", "cat", "sat"]

Feature extraction – Bigrams

- **Word n-grams** (i.e., sequences of n words that frequently occur in D – a.k.a. “shingles”) may be optionally added; this is usually limited to $n = 2$ (unigram+bigram model).

- The higher the value of n, the higher the semantic significance and the dimensionality k of the resulting representation, but the higher the computational cost.

- The bigram representation can incorporate word order partially.
  - Bigrams $\rightarrow$ ["the cat", "cat sat"]

Feature extraction – Bigrams

Word Unigrams	$\boxed{\text{A}}$ swimmer likes swimming thus he swims A $\boxed{\text{swimmer}}$ likes swimming thus he swims A swimmer $\boxed{\text{likes}}$ swimming thus he swims A swimmer likes $\boxed{\text{swimming}}$ thus he swims ...
Word Bigrams	$\boxed{\text{A swimmer}}$ likes swimming thus he swims A $\boxed{\text{swimmer likes}}$ swimming thus he swims A swimmer $\boxed{\text{likes swimming}}$ thus he swims A swimmer likes $\boxed{\text{swimming thus}}$ he swims ...

Feature extraction – Character *n*-grams

- An alternative to the process above is to make the set of features coincide with the set of **character n-grams** (e.g., $n \in \{3,4,5\}$) that occur in D.
  - Useful especially for degraded text (e.g., resulting from OCR).

Character 5-grams	It was a dark and stormy night It was a dark and stormy night It was a dark and stormy night It was a dark and stormy night It was a dark and stormy night It was a dark and stormy night ...

Feature extraction – TF-IDF

- TF-IDF is a numerical statistic that reflects the importance of a term within a document relative to a collection of documents (corpus).

- It is calculated by multiplying the term frequency (TF) in a document by the inverse document frequency (IDF) of the term across the corpus.

- TF-IDF is used to represent **each document** as a vector of TF-IDF scores for each term.

- TF-IDF can be applied not only to unigrams, but also to bigrams, trigrams, or combinations of them.

Feature extraction – Word Embeddings (1)

- Word embeddings are dense vector representations of words that capture semantic meaning.

- Techniques like SVD, GloVe, word2vec, and FastText can be used to generate word embeddings.

- Words in a document can be averaged or concatenated to create **document embeddings**.
  - It is also possible to use Doc2Vec → Will be shown during the labs.

- Employ these document embeddings for classification w.r.t. a specific supervised classification technique.

Feature extraction – Word Embeddings (2)

- **Embeddings** (word, sentence, or document embeddings) capture semantic information → Words or documents appearing in similar contexts have similar vector representations.

- These representations do not just encode topics. They can also reflect, depending on the corpus:
  - Sentiment or tone of the text,
  - Syntactic roles of words,
  - Semantic relations between concepts,
  - Even stylistic or authorial features.

- In this case, the choice of the corpus and proper labeling are crucial for achieving good classification performance on aspects other than just topics.

Feature extraction – Contextualized WEs (1)

- Unlike traditional word embeddings like Word2Vec or GloVe, which provide fixed representations for words, **contextualized embeddings** adapt their representations based on the surrounding context in which the word appears.

- **Preprocessing**:
  - Tokenization: Break the text into sentences or words, depending on the model's requirements.
  - Special tokens: Add [CLS] and [SEP] tokens at the beginning and end of each text for BERT-style models.
  - Padding: Ensure that all sequences are of the same length by padding or truncating as needed.

Feature extraction – Contextualized WEs (2)

- **Obtain Embeddings**
  - Use a pre-trained BERT (or BERT-like) model to obtain contextualized word embeddings for your text data.
  - You can leverage pre-trained models available in libraries like Hugging Face Transformers or other sources.

- **Aggregation**
  - You can choose to aggregate the word embeddings to obtain a fixed-length representation for the entire document.
  - Common aggregation techniques include mean pooling, max pooling, or concatenating embeddings.
    - Mean pooling, also known as average pooling, calculates the average of all the embeddings in the sequence.
    - Max pooling calculates the maximum value for each dimension (feature) across all the embeddings in the sequence.

Feature extraction (1)

- The above is OK for classification by **topic**, but not necessarily when classifying by other dimensions!

- Examples
  - In classification by author, features such average word length, average sentence length, punctuation frequency, frequency of subjunctive clauses, etc., are used.
    - *"You have such a scar on your neck, Mr. Eden," the girl was saying. "How did it happen? I am sure it must have been some adventure."*

      (Martin Eden, by Jack London)
  - In classification by sentiment, Bag-of-Words is not enough, and deeper linguistic processing is necessary.

- The choice of features for a classification task (**feature design**) is dictated by the distinctions we want to capture and is left to the designer.

Feature extraction (2)

- When you use BoW, TF-IDF, or word embeddings for classification (even when it's not topic-based), you can extend the vector with **additional features** to improve the model's performance.

$$feature\_vector = [text\_representatio \mid\mid extra\_features]$$

- Be careful with the **type** of additional features!

Feature extraction (3)

- The final feature vector is the concatenation of the text representation and the additional features of different kind:

$$final\_vector = [text\_vector \ \ || \ \ numeric\_features \\ || \ \ categorical\_features]$$

  - $text\_vector \rightarrow$ text representation (BoW, TF-IDF, or embedding)
  - $numeric\_features \rightarrow$ numerical features (e.g., text length, sentiment score)
  - $categorical\_features \rightarrow$ categorical features (e.g., author, language, source)

Feature extraction – Other features (1)

- **PoS**
  - Instead of just individual words, you can use Part-of-Speech tags as features.

- **Document length**
  - The length of a document can be used as a feature.
  - Short and long documents may have different characteristics in some classification tasks.

- **Topic Modeling**
  - Techniques like Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF) can be used to discover topics within a collection of documents, and the topic distribution for each document can be used as features.

Feature extraction – Other features (2)

- **Sentiment Analysis**
  - Sentiment scores or sentiment features derived from sentiment analysis can be used in sentiment classification tasks.

- **NER**
  - Named Entity Recognition is an NLP technique that identifies and classifies named entities, such as names of people, organizations, locations, dates, and more, in text.
  - Leveraging NER output as features can provide valuable information to improve text classification, especially when the entities are crucial for understanding the content or context of the text.

Feature extraction – How to treat them? (1)

- **Numerical features**
  - They can be simply concatenated to the text vector.
  - Be careful with scale:
    - Text representations (especially TF-IDF or embeddings) usually have small, balanced values.
    - Numerical features with large magnitudes (e.g., “number of words = 1500”), can dominate the classifier’s input → Normalize or standardize numerical features before concatenation.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaled_numeric = scaler.fit_transform(numeric_features)
final_vector = np.concatenate((text_vector,
scaled_numeric), axis=1)
```

Feature extraction – How to treat them? (2)

- **Categorical features** (like author, language, source) cannot be added directly → they must be numerically encoded.

- Converts each category into a binary vector.
  - Example:
    - language = ["it", "en", "fr"] → [1, 0, 0], [0, 1, 0], [0, 0, 1].
  - Works well for features with few categories.
  - In neural models, categorical variables can be represented with small dense embeddings (e.g., 8–32 dimensions).

Feature selection

- Vectors of length $O(10^5)$ or $O(10^6)$ may result, esp. if word n-grams are used, in both "overfitting" and high computational cost.
- **Feature selection (FS)** has the goal of identifying the most discriminative features, so that the others may be discarded.
- The "filter" approach to FS consists in measuring (via a function φ the discriminative power $\varphi(t_k)$ of each feature t_k and retaining only the top-scoring features.

Feature selection – Binary classification

- For binary classification, a typical choice is **Mutual Information (MI)**.

- In probability theory and information theory, it is a measure of the mutual dependence between the two variables.

$$MI(t_k, c_i) = \sum_{c \in \{c_i, \overline{c_i}\}} \sum_{t \in \{t_k, \overline{t_k}\}} \Pr(t, c) \log_2 \frac{\Pr(t, c)}{\Pr(t) \Pr(c)}$$

- An alternative choice is **chi-square** feature selection.
  - The chi-square test measures how much the observed frequency of a feature (e.g., a word) in a given class differs from the expected frequency if the feature and the class were independent.

Feature selection – Tools

- **Mutual Information**
  - https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.mutual_info_classif.html
  - https://towardsdatascience.com/select-features-for-machine-learning-model-with-mutual-information-534fe387d5c8

- **Chi-squared**
  - https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html
  - https://towardsdatascience.com/chi-square-test-for-feature-selection-in-machine-learning-206b1f0b8223
  - http://ethen8181.github.io/machine-learning/text_classification/chisquare.html

Feature synthesis

- **Matrix decomposition** techniques (e.g., PCA, SVD, LSA) can be used to synthesize new features that replace the features discussed above.

- These techniques are based on the principles of distributional semantics, which states that the semantics of a word "is" the words it co-occurs with in corpora of language use.
  - Pros: synthetic features in the new vectorial representation.
  - Cons: computationally expensive, sometimes prohibitively.

- Therefore: Word Embedding/Contextualized Word Embedding, i.e., the "new wave of distributional semantics".

Feature weighting (1)

- **Feature weighting** means attributing a value w_{ki} to feature t_k in the vector $\vec{x}_i$ that represents document d_i: this value may be:
  - Binary (representing presence/absence of t_k in d_i).
  - Numeric (representing the importance of t_k for d_i). It can be obtained via feature weighting functions in the following two classes:
    - Unsupervised: e.g, $TF - IDF.$
    - Supervised: e.g., $TF * MI$, $TF * \chi^2$.

Feature weighting (2)

- The choice of feature weighting method depends on the characteristics of your text data and the nature of your classification task.

- It is often a good practice to experiment with different weighting strategies and select the one that performs best for your specific use case through cross-validation and evaluation metrics.

- Keep in mind that the effectiveness of these methods may vary depending on the size of your dataset, the complexity of the task, and the quality of your textual representation.

Feature weighting (3)

- **Attention Mechanisms**
  - Incorporate attention mechanisms into your model architecture.
  - The model can learn to assign different attention weights to words based on their importance in the context of the classification task.

- **Word Embedding Fine-Tuning**
  - Fine-tune the word embeddings themselves during the training of your classification model.
    - The embeddings will be updated to be more relevant for the specific task at hand → You fine-tune only the word vectors within your own classifier.

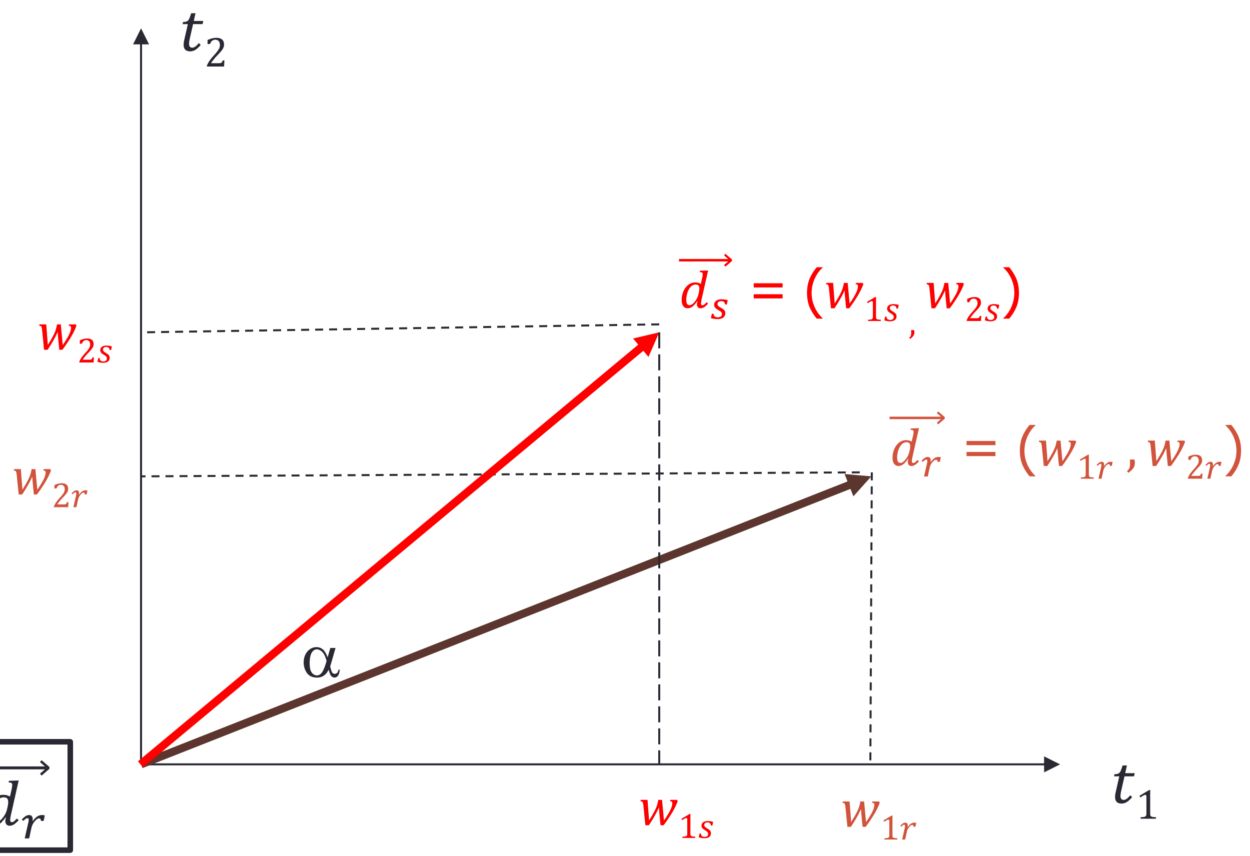
- **Pre-trained Model Fine-Tuning**
  - If you are using a pre-trained language model like BERT or GPT, you can fine-tune the model on your classification task.
    - The fine-tuning process adapts the model's internal representations to your specific task and classification criteria.

Document similarity

- Underlying the text classification process is the concept of **similarity** between formal representations of documents.
- To compute similarity among documents in a vector space, you can use various techniques and similarity metrics.
- The idea is to represent documents as vectors in a high-dimensional space, where each dimension corresponds to a unique feature, term, or word, and then measure the similarity between these document vectors.

Vector Space Representation

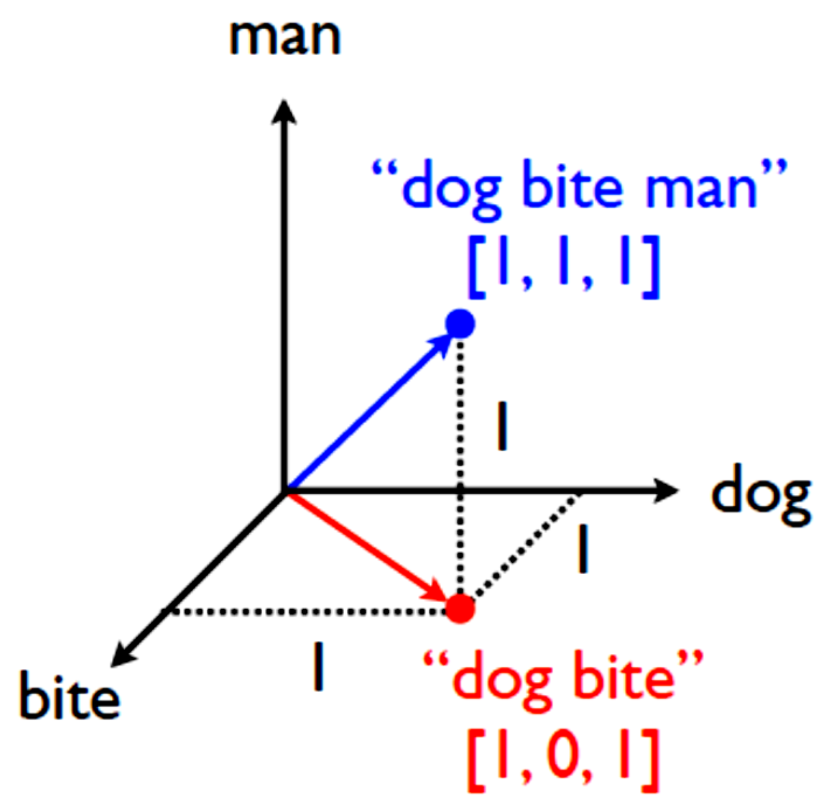
- Representation of documents in two-dimensional space defined by two terms



$$\boxed{\alpha = 0° \rightarrow \overrightarrow{d_s} \equiv \overrightarrow{d_r}}$$

Vector Space Representation (Binary)

	dog	*man*	*bite*
doc_1	1	1	1
doc_2	1	0	1



The Cosine Similarity

- **Similarity** between two vectors can be computed as follows:

$$sim(\vec{x}, \vec{y}\,) = \cos\alpha = \frac{(\vec{x} \bullet \vec{y})}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

- For two documents represented as two vectors

$$sim(\overrightarrow{d_j}, \overrightarrow{d_k}) = \frac{\overrightarrow{d_j} \bullet \overrightarrow{d_k}}{\|\overrightarrow{d_j}\| \cdot \|\overrightarrow{d_k}\|} = \frac{\sum_{i=1}^{n} w_{ij} w_{ik}}{\sqrt{\sum_{i=1}^{n} (w_{ij})^2 \sqrt{\sum_{i=1}^{n} (w_{ik})^2}}}$$

- If $w_{ij} > 0$ and $w_{ik} > 0 \rightarrow 0 \leq sim(\overrightarrow{d_j}, \overrightarrow{d_k}) \leq 1$

The Cosine Similarity – Example

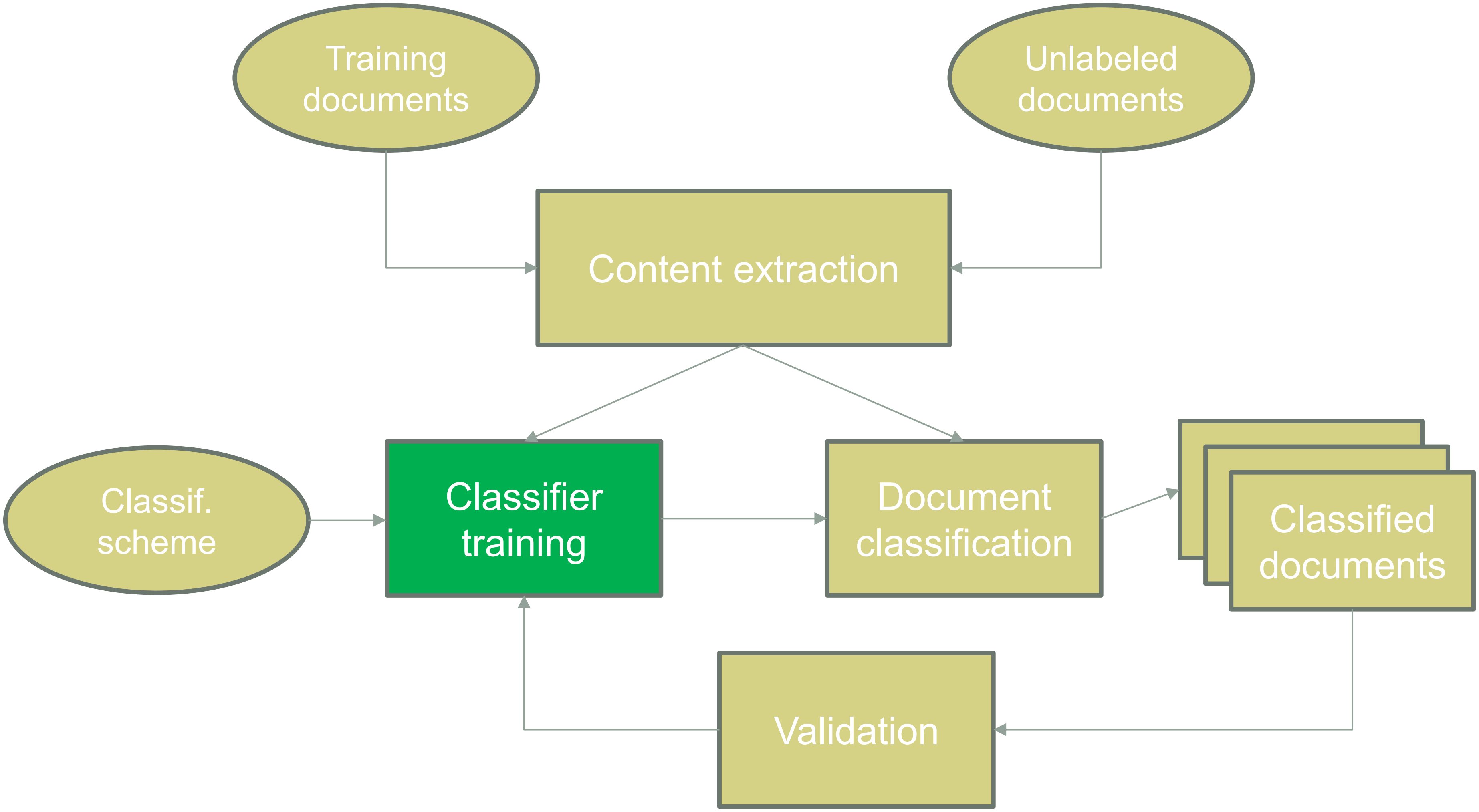
- Document A vector: $A = [2, 3, 1, 0, 1]$
- Document B vector: $B = [1, 2, 0, 1, 2]$

$$sim(A, B) = \frac{A \bullet B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} w_{ij} w_{ik}}{\sqrt{\sum_{i=1}^{n} (w_{ij})^2} \sqrt{\sum_{i=1}^{n} (w_{ik})^2}}$$

- Inner product:
  - $A \bullet B = (2 * 1) + (3 * 2) + (1 * 0) + (0 * 1) + (1 * 2) = 2 + 6 + 0 + 0 + 2 = 10$
- Eucledean norms:
  - $\| A \| = \sqrt{(2^2 + 3^2 + 1^2 + 0^2 + 1^2)} = \sqrt{14}$
  - $\| B \| = \sqrt{(1^2 + 2^2 + 0^2 + 1^2 + 2^2)} = \sqrt{10}$
- Cosine similarity: $\frac{10}{\sqrt{14} * \sqrt{10}} = 0.6793$

Supervised classification

Supervised learning and classification



SL for binary classification

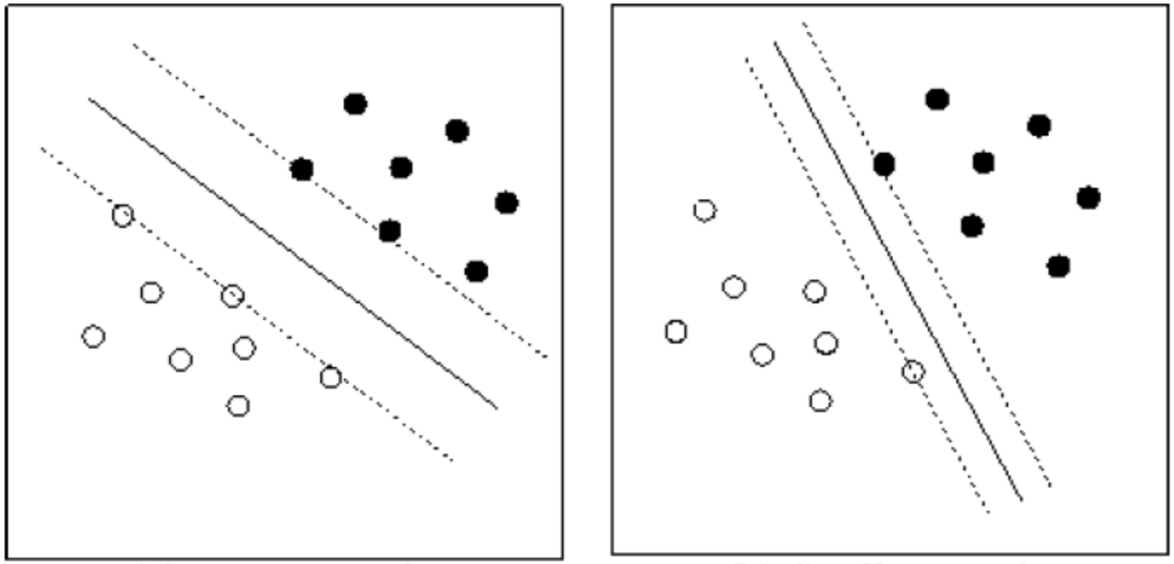
- For binary classification, essentially **any supervised learning algorithm** can be used for training a classier; popular choices include:
  - Support vector machines (SVMs)
  - Boosted decision stumps (One-level decision tree)
  - Logistic regression
  - Naive Bayesian methods
  - Lazy learning methods (e.g., *k*-NN)

- The "No-free-lunch principle" (Wolpert, 1996): there is no learning algorithm that can outperform all others in all contexts.

- Implementations need to cater for:
  - The very high dimensionality typical of TC.
  - The sparse nature of the representations involved.

A supervised learning method: SVMs

- A **constrained optimization problem**: find the separating surface (e.g., hyperplane) that maximizes the margin (i.e., the minimum distance between the hyperplane and the training examples).



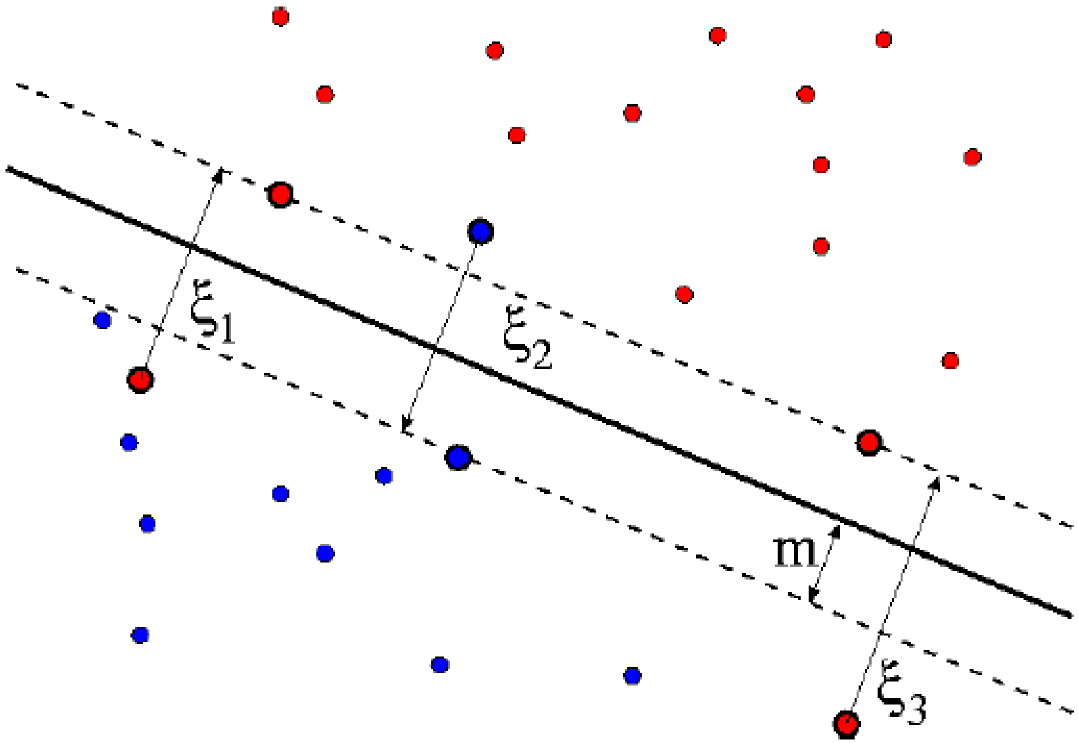
(a) Larger margin (b) Smaller margin

A supervised learning method: SVMs

- Margin maximization conducive to good generalization accuracy on unseen data.
- Theoretically well-founded + good empirical performance on a variety of tasks.
- Publicly available implementations optimized for high-dimensional, sparse feature spaces:
  - E.g., `SVM-Light`, `LibSVM`.

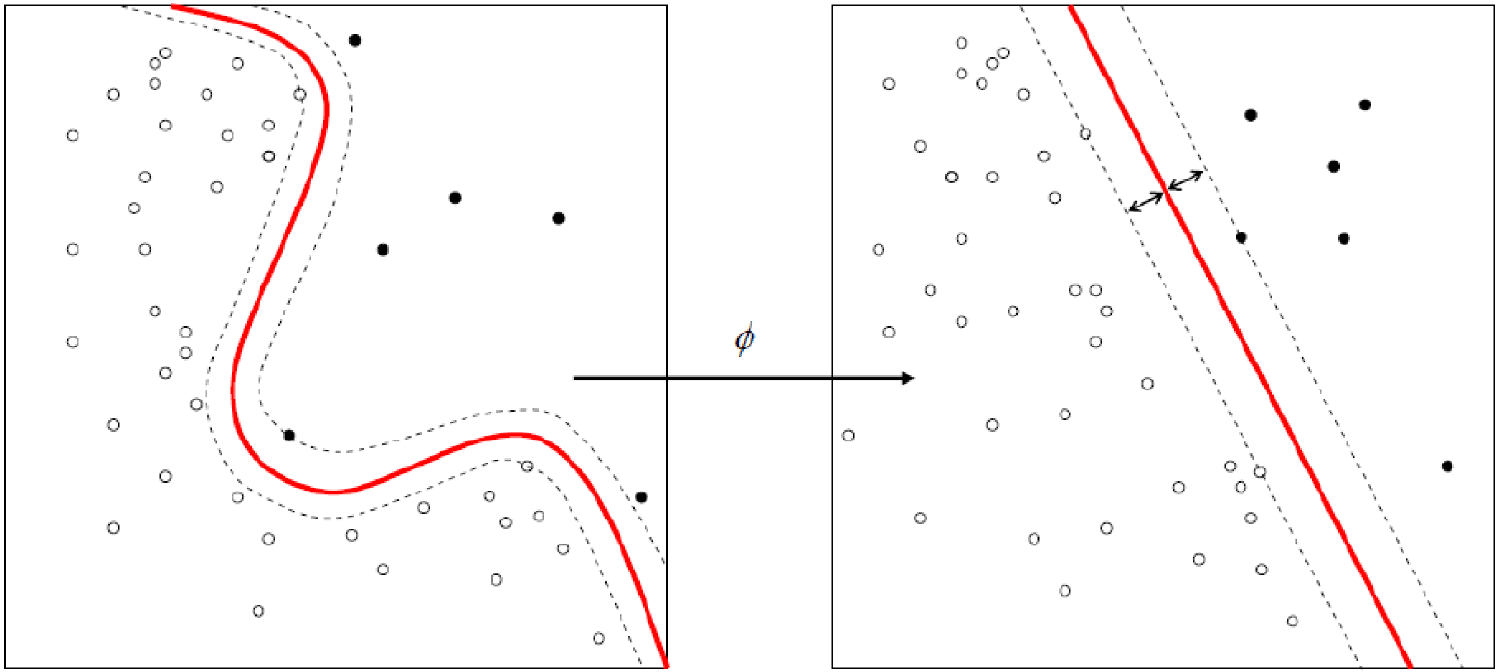
A supervised learning method: SVMs

- Classication problems are often not linearly separable (LS).



A supervised learning method: SVMs

- Non-LS problems can become LS once mapped to a high-dimensional space.



A supervised learning method: SVMs

- **Kernels** are similarity functions $K(\vec{x}_i, \vec{x}_j) = \rho(\vec{x}_i)\rho(\vec{x}_j)$, where $\rho(\cdot)$ is a mapping into a higher-dimensional space.

- SVMs can indeed use kernels instead of the standard dot product.

- Popular kernels are:
- $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i \cdot \vec{x}_j$ (the linear kernel)
- $K\left(\vec{x}_i, \vec{x}_j\right) = (\gamma \vec{x}_i \cdot \vec{x}_j + r)^d \; \gamma > 0$ (the polynomial kernel)
- $K\left(\vec{x}_i, \vec{x}_j\right) = \exp\left(-\gamma \left\|\vec{x}_i - \vec{x}_j\right\|^2\right) \; \gamma > 0$ (the RBF kernel)
- $K\left(\vec{x}_i, \vec{x}_j\right) = \tanh(\gamma \vec{x}_i \cdot \vec{x}_j + r)$ (the sigmoid kernel)

- However, the linear kernel is usually employed in text classication applications.

SL for non-binary classification

- Some learning algorithms for **non-binary classification** are «SLMC-ready», e.g.:
  - Decision trees
  - Boosted decision stumps
  - Logistic regression
  - Naive Bayesian methods
  - Lazy learning methods (e.g., *k*-NN)

- For other learners (notably: SVMs) to be used for SLMC classification, combinations / cascades of the binary versions need to be used.

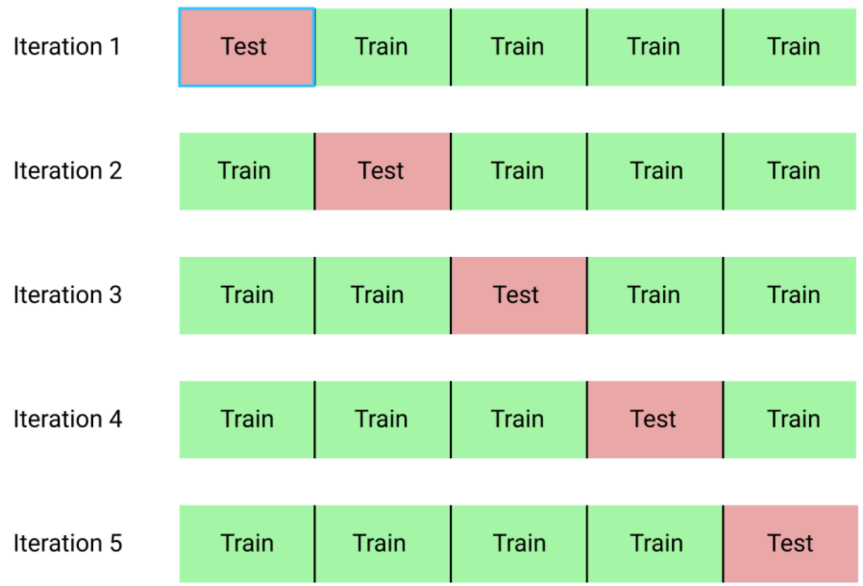
- For ordinal classification, algorithms customised to OC need to be used (e.g., SVORIM, SVOREX).

Yang, Y., Chen, B., & Yang, Z. (2019). An Algorithm for Ordinal Classification Based on Pairwise Comparison. *Journal of Classification*, 1-22

Parameter optimization in supervised learning

- The trained classifiers often depend on one or more parameters, e.g., γ, r, d parameters of non-linear kernels.

- These parameters need to be optimized, e.g., via ***k*-fold cross-validation** on the training set.



EVALUATIONS

Evaluating a classifier

- Two important aspects in the evaluation of a classifier are **efficiency** and **effectiveness**.

- **Efficiency** refers to the consumption of computational resources, and has two aspects:
  - Training efficiency (also includes time devoted to performing feature selection and parameter optimization).
  - Classification efficiency, usually considered more important than training efficiency, since classifier training is carried out: (*a*) offine and (*b*) only once.

- In text classification papers, it is good practice to report training costs and classification costs.

Evaluating a classifier

- **Effectiveness** (a.k.a., accuracy) refers to how frequently classification decisions taken by a classifier are “correct”.
- Usually considered more important than efficiency, since accuracy issues “are there to stay”.
- Effectiveness tests are carried out on one or more datasets meant to simulate operational conditions of use.
- The main pillar of effectiveness testing is the evaluation measure we use.

Evaluation measures for classification

- Each type of classification (binary/SLMC/MLMC/ordinal) and mode of classification (hard/soft) requires its own measure
- For binary (hard) classification, given the contingency table Λ

		true	
		YES	NO
predicted	YES	TP	FP
	NO	FN	TN

  the standard measure is F_1, the harmonic mean of precision ($\pi = \frac{TP}{TP+FP}$) and recall ($\rho = \frac{TP}{TP+FN}$), i.e.,

$$F_1 = \frac{\pi\rho}{\pi+\rho} = \frac{2TP}{2TP+FP+FN}$$

- F_1 is robust to the presence of imbalance in the test set

Evaluation measures for classification

- For multi-label multi-class classification, F_1 must be averaged across the classes, according to
  1. microaveraging: compute F_1 from the "collective" contingency table obtained by summing cells (e.g., $TP = \sum_{c_i \in \mathcal{C}} TP_i$)
  2. macroaveraging: compute $F_1(c_i)$ for all $c_i \in \mathcal{C}$ and then average
- Micro usually gives higher scores than macro ...
- For single-label multi-class classification, the most widely used measure is ("vanilla") accuracy

$$A = \frac{\sum_{c_i \in \mathcal{C}} \Lambda_{ii}}{\sum_{c_i, c_j \in \mathcal{C}} \Lambda_{ij}}$$

where Λ_{ij} is the number of documents in c_i which are predicted to be in c_j

Some “classic” datasets for evaluating text classication

	Total examples	Training examples	Test examples	Classes	Hierarchical	Language	Type
Reuters-21578	≈ 13,000	≈ 9,600	≈ 3,200	115	No	EN	MLMC
RCV1-v2	≈ 800,000	≈ 20,000	≈ 780,000	99	Yes	EN	MLMC
20Newsgroups	≈ 20,000	—	—	20	Yes	EN	MLMC
OHSUMED-S	≈ 16,000	≈ 12,500	≈ 3,500	97	Yes	EN	MLMC
TripAdvisor-15763	≈ 15,700	≈ 10,500	≈ 5,200	5	No	EN	Ordinal
Amazon-83713	≈ 83,700	≈ 20,000	≈ 63,700	5	No	EN	Ordinal

Some more recent datasets

- https://imerit.net/blog/17-best-text-classification-datasets-for-machine-learning-all-pbm/

- Text Classification Dataset Repositories
  - Including TREC

- Sentiment Analysis and Review Datasets
  - Social media content, reviews (also from Amazon)

- Online Content Evaluation Datasets
  - Including hate speech detection dataset

Tools to experiment with text classication

- Several publicly available environments where to play with text preprocessing routines, feature selection functions, feature weighting functions, learning algorithms, etc., such as:
  - `scikit-learn` (http://scikit-learn.org/): Python-based, features various classification, regression and clustering algorithms including SVMs, random forests, gradient boosting, k-means (...), and is designed to interoperate with the Python numerical and scientific libraries `NumPy` and `SciPy`.
  - `Weka` (https://www.cs.waikato.ac.nz/ml/weka/): Java-based, features various algorithms for data analysis and predictive modeling.
  - `keras` (https://keras.io/): Keras is an open-source library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow library.
    - https://keras.io/examples/nlp/text_classification_from_scratch/

Current trends

- Text Classification with Word Embeddings
  - https://www.tensorflow.org/text/guide/word_embeddings
  - https://medium.com/analytics-vidhya/text-classification-using-word-embeddings-and-deep-learning-in-python-classifying-tweets-from-6fe644fcfc81

- Text Classification with BERT
  - https://www.analyticsvidhya.com/blog/2021/06/why-and-how-to-use-bert-for-nlp-text-classification/
  - https://www.tensorflow.org/text/tutorials/classify_text_with_bert