DATA SCIENCE FOR DIGITAL HUMANITIES 1

# INTRO TO MACHINE LEARNING

# TEXT CLASSIFICATION & CLUSTERING

PROF. DR. GORAN GLAVAŠ

# Refresher: machine learning basics

- **Supervised machine learning**
  - We have **labeled data** as input
  - Supervised ML algorithms learn the mapping between input representations and output labels
  - **Classification**: output – discrete label (no ordering between labels)
  - **Regression**: output – an integer or real value (obviously, there is ordering between labels)

- **Unsupervised machine learning**
  - We have no labels (i.e., we have unlabeled data) at input
  - **Clustering**: grouping instances by the similarity of their representations
  - **Outlier detection**: recognizing instances that are very dissimilar from all other instances in the dataset

# Refresher: machine learning basics

- **Supervised machine learning** models „learn" the mapping between input values and output values
  - A single input to the classifier is called an **instance** or **example** (denoted „$\mathbf{x}$")
  - An instance is represented as an n-dimensional feature vector

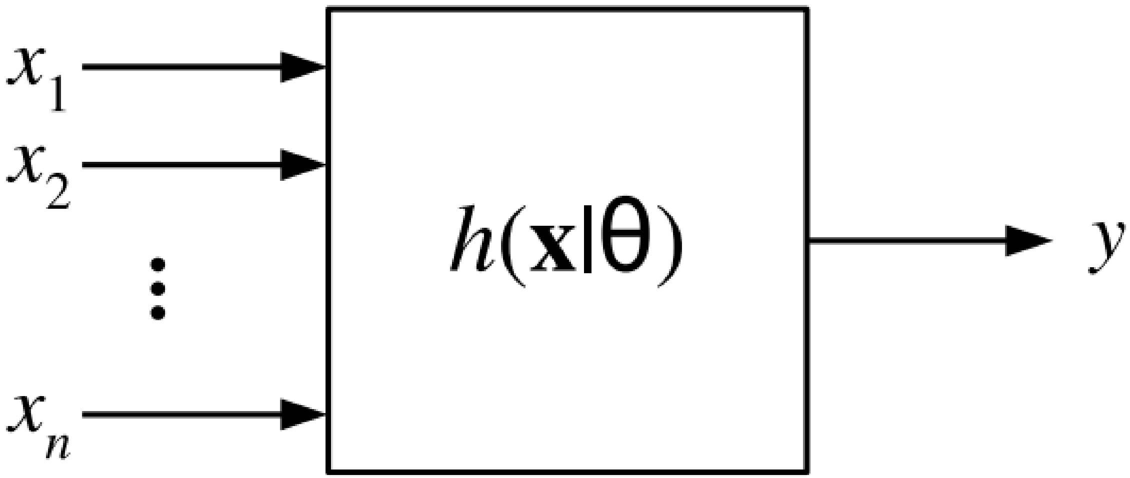
$$\mathbf{x} = (x_1, x_2, \ldots, x_n)$$

- The desired output is called the **target label** (or just label, denoted $y$)
- **A classifier** $h$ maps an instance $\mathbf{x}$ to a label $y$ – $h : \mathbf{x} \rightarrow y$

- „Learning" – model has parameters $\boldsymbol{\theta}$ (denoted $h(\mathbf{x}|\ \boldsymbol{\theta})$) whose values are optimized to maximize the prediction accuracy of the output labels, given instance

# Refresher: machine learning basics

## Supervised classification




- Binary classification: just two output labels (yes/no, 0/1)
- Multi-class classification: each instance has one of K labels
- Multi-label classification: an instance can have more than one label at once
- Sequence labeling: input a sequence of instances and output a sequence of labels

# Text Classification and Clustering

For both classification and clustering we **represent texts/documents** as **numeric vectors**:

1. **Sparse text/document representations**
   - Sparse Bag-of-words vectors
   - Term-frequency (TF) – Inverse Document Frequency (IDF) weighting

   - For classification: traditional ML models/algorithms, e.g., ***logistic regression***
   - For clustering**:** we compare the sparse TF-IDF vectors of documents

2. **Dense text/document representations**
   - Document representations aggregated from dense ***word embeddings*** or generated with ***pre-trained deep neural encoders*** (e.g., BERT)

   - For classification: neural ML models, e.g., multi-layer perceptron, CNN, RNN
   - For clustering: we compare dense document vectors (embeddings)

# Sparse text representations

# Sparse text representations

- $V$ – vocabulary (set) of all words that we find in our collection of documents
- E.g., $V$ = {*„a”*, *„aachen”*, *„an”*, *„animal”*, ..., *„zuma”*, *„zygot”*}

- Sparse document vectors are **|*V*|-dimensional** vectors, in which each dimension corresponds to one word from the vocabulary V

$$d_j = [w_{1,j}, w_{2,j}, \ldots, w_{|V|-1,j}, w_{|V|,j}]$$

$$\begin{array}{ccccc} \uparrow & \uparrow & & \uparrow & \uparrow \\ \text{„a”} & \text{„an”} & \ldots & \text{„zuma”} & \text{„zygot”} \end{array}$$

- Weight $W_{i,j}$ captures the „importance” of the *i*-th vocabulary word for the j-th document in the collection
- If the *i-th* word does not appear in the *j-th* document, then $W_{i,j} = 0$

# Sparse text representations

- *V* – vocabulary (set) of all words that we find in our collection of documents
- E.g., *V* = {*„a”*, *„aachen”*, *„an”*, *„animal”*, ..., *„zuma”*, *„zygot”*}

$$d_j = [w_{1,j}, w_{2,j}, \ldots, w_{|V|-1,j}, w_{|V|,j}]$$

*„a”* *„an”* ... *„zuma”* *„zygot”*

- Weight $W_{i,j}$ captures the „importance” of the *i*-th vocabulary word for the j-th document in the collection

- If the *i-th* word does appear in the *j-th* document, what should $W_{i,j}$ look like?

# Sparse text representations

- If the *i-th* word does appear in the *j-th* document, what should $W_{i,j}$ look like?
- **Two assumptions**:

  1. The term is **more relevant for the document**, the **more frequently it appears in the document**

  2. The term is **more relevant for the document** the **less commonly it occurs across other documents**

- These two assumptions give rise to the popular and effective weighting scheme called **TF-IDF**:
  - Term frequency (TF, assumption 1)
  - Inverse document frequency (IDF, assumption 2)

# Sparse text representations

TF-IDF(*word*$_i$, *document*$_j$) = TF(*word*$_i$, *document*$_j$) * IDF(*word*$_i$)

$$\mathrm{TF}(\mathrm{word}_\mathrm{i}, \mathrm{document}_\mathrm{j}) = \frac{\mathrm{freq(wordi, documentj)}}{\mathrm{max.}\, word\ freq\ (documenti)}$$

$$\mathrm{IDF}(\mathrm{word}_\mathrm{i}) = \log \frac{|D|}{|\{d \in D : wordi \in d\}|}$$

TF(„a”, $d_j$) * IDF(„a”) ... TF(„zygot”, $d_j$) * IDF(„zygot”)

$$d_j = [w_{1,j},\ w_{2,j},\ \ldots,\ w_{|V|-1,j},\ w_{|V|,j}]$$

*„a”* *„an”* ... *„zuma”* *„zygot”*

# Traditional text classification

# Traditional Text Classification

**Traditional text classification**:

1. **Sparse text/document representations**
   - Sparse Bag-of-words vectors
   - Term-frequency (TF) – Inverse Document Frequency (IDF) weighting

2. **Traditional classification algorithms**:
   - **Logistic regression** (despite the name, a classification model)
   - (Linear) Support Vector Machines

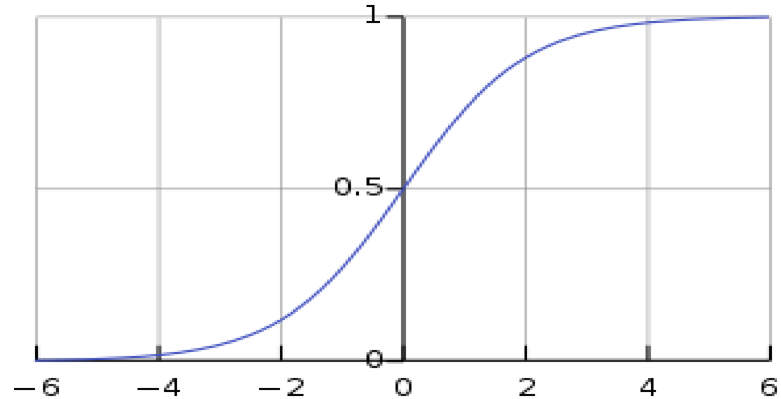
# Logistic regression

- Despite its name, **logistic regression** is a classification algorithm

- We will focus on binary classification – logistic regression computes the probability that some instance **x** belongs to some class ($y$ = 1)

$$h(\mathbf{x}|\ \boldsymbol{\theta}) = P(y = 1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})} = \sigma(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})$$

- Logistic regression is based on a logistic function: $\sigma(a) = 1 / (1 + e^{-a})$
- The logistic function maps the input value to the output interval [-1, 1]



# Logistic regression

- **LR for text classification**

- We will focus on binary classification – logistic regression computes the probability that some instance $\mathbf{x}$ belongs to some class ($y = 1$)

$$h(\mathbf{x}|\ \boldsymbol{\theta}) = P(y = 1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})} = \sigma(\boldsymbol{\theta}^{\mathrm{T}}\mathbf{x})$$

- In text classification, the instance $\mathbf{x}$, representing a single document, is the **TF-IDF vector of that document**,
  - So each component of $\mathbf{x} = [x_1, x_2, ..., x_{|V|}]$ **is the TF-IDF score** of one vocabulary word in that document
  - I.e., $x_i = w_{i,j} = TF(word_i, document_j) * IDF(word_i)$

# Logistic regression

- **LR for text classification**

$$h(\mathbf{x}|\ \boldsymbol{\theta}) = P(y = 1 \mid \mathbf{x}) = \frac{1}{1+\exp(-\boldsymbol{\theta}^{T}\mathbf{x})} = \sigma(\boldsymbol{\theta}^{T}\mathbf{x})$$

- Looking at the logistic regression formula (and the properties of log. function):

  $h(\mathbf{x}|\boldsymbol{\theta}) > 0.5$ (i.e., instance $\mathbf{x}$ belongs to the class) if and only if $\boldsymbol{\theta}^{T}\mathbf{x} > 0$
  $h(\mathbf{x}|\boldsymbol{\theta}) < 0.5$ (i.e., instance $\mathbf{x}$ doesn't belong to the class) if and only if $\boldsymbol{\theta}^{T}\mathbf{x} < 0$

- But what are the LR's parameters $\boldsymbol{\theta} = [\theta_1, \theta_2, ..., \theta_{|V|}]$?
  - One parameter for each vocabulary word
    - $\theta_1$ captures how indicative the *word*$_i$ is for the class of the document (class 1)
  - Their „optimal" values need to be learned on the „training set"

# Dense text representations and deep text classification

# Modern (Deep) Text Classification

**Modern text classification**:

1. **Dense text/document representations**
   - Documents represented as **sequences of word embeddings**

2. Deep learning **encoding / classification algorithms**:
   - **Convolutional neural networks**
   - Recurrent neural networks
   - Attention networks (so-called Transformers)

# Convolutional Neural Networks

- **Convolutional neural networks (CNNs)** use **convolution** instead of general matrix multiplication in at least one layer

- Convolution = sum over elements of the element-wise product of two matrices
- Convolution between two M x N matrices, **A** and **B** is then:

$$\text{Conv}(A, B) = \sum_{i=1}^{M} \sum_{j=1}^{N} A_{ij} * Bij$$

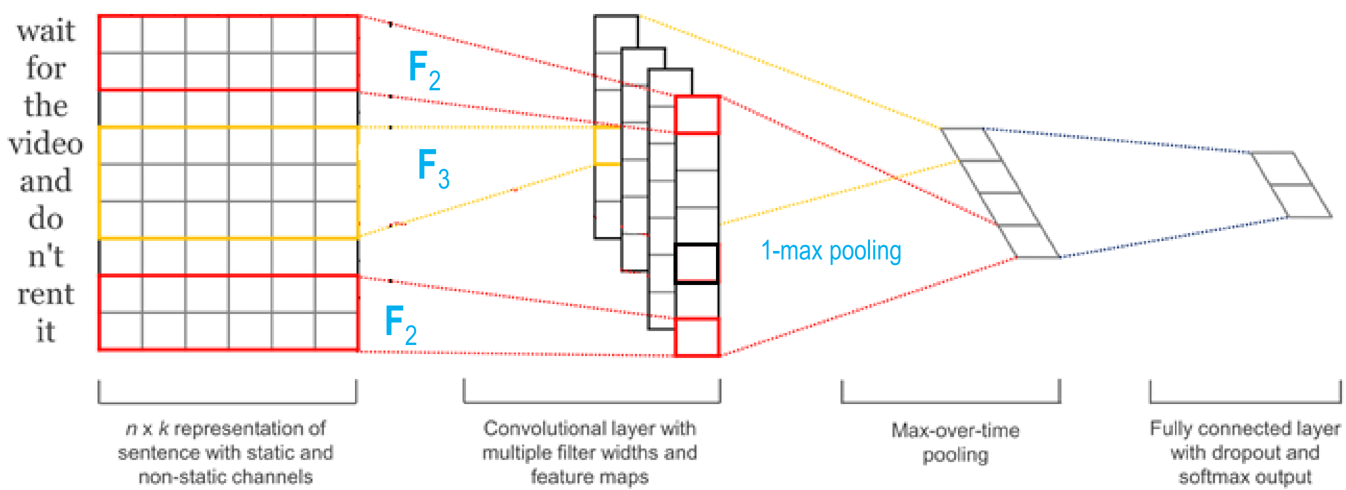
- Using convolutions instead of general matrix multiplication better models components of complex (hiearchical) structures

  - **NLP**: Texts → paragraphs → sentences → words
  - **CV**: Images → objects → regions → contures → ... → pixels

# Convolutional neural networks

- CNN are used to derive latent (dense) representations of larger portions of text by aggregating local sequences
    - Modelling local dependencies

- **Input:** Sequence of word embedding vectors
- **Parameters**: filter matrices (F) with which subsequences of input are convoluted



# Convolutional neural networks

- **Filters**: CNN parameter matrices of size $K \times d$, where $K$ is small number, typically between 3 and 5 and $d$ is the length of word embedding vectors
  - $K$ is called the size of the filter
  - One CNN typically has many filters, often of different sizes
  - E.g., 32 filters of size 3, 64 filters of size 4, and 32 filters of size 5

- **Convolution layer**:
  - Each filter strides down the input sequence and produces a convolution score with each input subsequence of size $K$
  - Let $\mathbf{F}_K$ be one filter (matrix) od size $K$ (i.e., dimensions $K \times d$)
  - Let $\mathbf{X}_{[a:b]}$ be the submatrix of the input matrix $\mathbf{X}$ consisting of rowa $a$ to $b$

  - **We then compute the vector of following convolutions**

$$\mathbf{C(F_K) =} [\mathrm{Conv}(\mathbf{X}_{[1:K]}, \mathbf{F}_K); \mathrm{Conv}(\mathbf{X}_{[2:K+1]}, \mathbf{F}_K); \mathrm{Conv}(\mathbf{X}_{\mathbf{[3:K+2]}}, \mathbf{F}_K); \ldots; \mathrm{Conv}(\mathbf{X}_{[N-K+1:N]}, \mathbf{F}_K)]$$

# Convolutional neural networks

- **Pooling layer**:
  - Each filter $\mathbf{F}_K$ will produce a vector of convolution scores $\mathbf{C}(\mathbf{F}_K)$ over the input sequence
  - We want to keep only the „most salient" local sequences
  - That's why we tipically select only $k$ largest values from the convolution vector of each filter – this is called **k-max pooling**
  - **Most often 1-max pooling is used (only the largest value is kept)**

- **Latent text representation**:
  - We concatenate the results of pooling for each of the filters into a single vector which is the latent representation of the text
  - This is the final representation $\mathbf{x}_{\mathbf{CNN}}$ of our document, which goes into the classifier

# Convolutional neural networks

- **Deep CNNs**
  - When we chain more than one convolutional layer
  - Outputs of one convolution layer become input for another convolution layer
  - Each convolution layer has its own set of filters

- **Classification**
  - CNN itself is not a classifier,
  - It merely builds an informative dense latent representation of the text (i.e., dense vector representing the input text)
  - To make a prediction, we couple CNN with a feed-forward classification network

# CNN-based Text Classification

- Let $\mathbf{x}_{CNN}$ be the latent vector produced by the CNN and $\mathbf{W}_{FF}$ and $\mathbf{b}_{FF}$ be the weight matrix and bias vector of the feed-forward classifier:

- The classification prediction is then given by:

$$\mathbf{y} = \mathrm{Softmax}(\mathbf{x}_{CNN}\,\mathbf{W}_{FF} + \mathbf{b}_{FF})$$

- The parameters of the CNN and the feed-forward network are then jointly optimized during training

# Text clustering

# Cluster Analysis („Clustering”)

- **Cluster analysis** (or, colloquially, **clustering**) is a multivariate statistical technique that allows automated generation of groupings in data

- **Components of clustering**:
  1. An **abstract representation** of an object using which the object is compared to other objects
  2. A **function** that measures the **distance or similarity** between the objects based on their abstract representations
  3. A **clustering algorithm** that groups the objects based on the similarities / distances computed from their representations
  4. (optional) **Constraints** with respect to cluster membership, cluster proximity, shape of the clusters, etc.

# Text Clustering

- **Representations of text** for clustering are usually the same as for text classification (only we lack the labels)
  - Sparse vectors (binary or weighted, e.g., using TF-IDF)
  - Dense vectors (latent or semantic representations, e.g., word embedding average)

- **Common distance/similarity functions**
  - **Cosine similarity/distance**
  - Euclidean distance, Jaccard coefficient, Kullback-Leibler divergence, ...

- **Clustering algorithms**:
  1. Sequential – e.g., **single pass clustering**
  2. Hierarchical – e.g., agglomerative clustering, divisive clustering
  3. Cost-function optimization clustering – e.g., **K-means**

# Single pass clustering

- **Simplest clustering algorithm**
  - The number of clusters does not need to be predefined

- **Algorithm**:
  1. Start by putting the first text $t_1$ into the first cluster $c_1 = \{t_1\}$
  2. For all other texts, $t_2, ..., t_n$, one by one

     I. Measure the distance/similarity with all existing clusters $c_1, ..., c_k$
     - The similarity with the cluster is avg/max of similarities with instances in cluster

     II. Identify the cluster $c_i$ with which the current text $t_j$ has the largest similarity (or smallest distance)

     III. If the similarity between $t_j$ and $c_i$ is above some predefined threshold $\lambda$, add the text $t_j$ to cluster $c_i$

# K-Means

- Arguably the most famous and widely used clustering algorithm
- Requires the number of clusters $k$ to be predefined – K clusters, $\mathbf{S} = \{S_1, S_2, ..., S_k\}$, represented by mean vectors $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, ..., \boldsymbol{\mu}_k$
- **K-means** clusters instances ($\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n$) by finding the partition **S** that minimizes the within-cluster distances (maximizing the within-cluster similarities):

$$\underset{\mathbf{S}}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x} \in S_i} \left\| \mathbf{x} - \boldsymbol{\mu}_i \right\|^2$$

- **Q:** How to find optimal clusters (minimize the sum of within-cluster distances)?
- **A: Using iterative optimization**

# K-Means

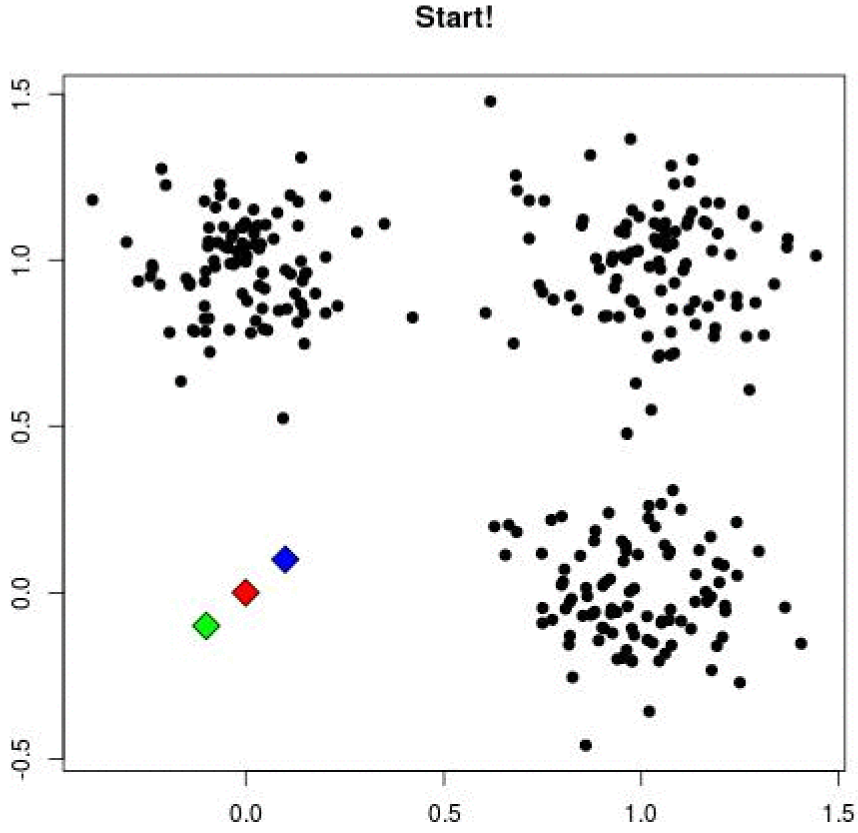
- Algorithm for learning the centroids:
  1. Randomly pick K mean vectors $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, ..., $\boldsymbol{\mu}_k$ in the same space (i.e., of same dimensionality) as instance vectors $\mathbf{x}$
     - **K-means++** is an extension that more intelligently chooses the initial mean vectors
  2. Iterate the following two steps **until convergence**:
     I. Assign each instance $\mathbf{x}_j$ to the cluster with the closest mean vector $\boldsymbol{\mu}_i$ :

$$S_i^{(t)} = \left\{ \mathbf{x}_j : \|\mathbf{x}_j - \boldsymbol{\mu}_i^{(t)}\|^2 \leq \|\mathbf{x}_j - \boldsymbol{\mu}_j^{(t)}\|^2, \forall j, 1 \leq j \leq k \right\}$$

     II. For each cluster, update the mean vector of a cluster
        › Set the mean vector to the mean of the instances in the cluster

$$\boldsymbol{\mu}_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$$

# K-Means




**Example from**: https://www.projectrhea.org/rhea/index.php/SlectureDavidRunyanCS662Spring14