

THE UNIVERSITY of EDINBURGH

## Text Technologies for Data Science

**INFR11145**

# Text Classification

Instructor:
**Björn Ross**

18-Nov-2026

1

## Pre-Lecture

- Today
  - Lecture 1 (Text classification): theory
  - Lecture 2: practical – building a tweet classifier
- This week
  - Lab 7 released; Coursework 2 part 3 builds on both
- Next week
  - Learning to rank; retrieval-augmented generation (RAG)
  - Coursework 3 announced



2

## Lecture Objectives

- Learn about text basics of text classification
  - Definition
  - Types
  - Methods and models



## Definition

## Text Classification

- **Text classification** is the process of classifying documents into predefined categories based on their content.

- Input: Text (document, article, sentence)
- Task: Classify into predefined one/multiple categories
- Categories:
  - Binary: relevant/irrelevant, spam .. etc.
  - Few: sports/politics/comedy/technology
  - Hierarchical: patents



## Classification is and is not

- Classification (a.k.a. "categorisation"): a common 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 $D = \{\mathbf{x}_1, \mathbf{x}_2, ...\}$ is a domain of data items and
$C = \{c_1, ..., c_n\}$ is a finite set of classes (the classification scheme)

## Classification is and is not

- Different from clustering, where the groups ("clusters") and their number are not known in advance
- Unsuitable when class membership can be determined with certainty (relatively easily)
  - e.g., predicting whether a natural number belongs to *Prime* or *Non-Prime* is not classification
- In text classification, data items are
  - **Textual**: e.g., news articles, emails, sentences, queries, etc.
  - **Partly textual**: e.g., Web pages



## Types of classification

## Types of Classification

- **Binary**:
  item to be classified into one of two classes
  $h : D \rightarrow C$, $C = \{c_1, c_2\}$
  - e.g., Spam/not spam, offensive/not offensive, rel/irrel
- **Single-Label Multi-Class** (SLMC)
  item to be classified into only one of *n* possible classes.
  $h : D \rightarrow C$, $C = \{c_1 \ldots. c_n\}$, where n>2
  - e.g., Sports/politics/entertainment, positive/negative/neutral
- **Multi-Label Multi-Class** (MLMC)
  item to be classified into none, one, two, or more classes
  $h : D \rightarrow 2^C$, $C = \{c_1 \ldots. c_n\}$, where n>1
  - e.g., Assigning CS articles to classes in the ACM Classification System
  - Usually be solved as *n* independent binary classification problems



## Methods and models

## Methods and models

- Rule-based classification
- Supervised-learning classification
  - Traditional features
  - Word embeddings
- Pre-trained language models
  - Supervised fine-tuning for classification
  - Zero-shot classification



## Rule-based classification

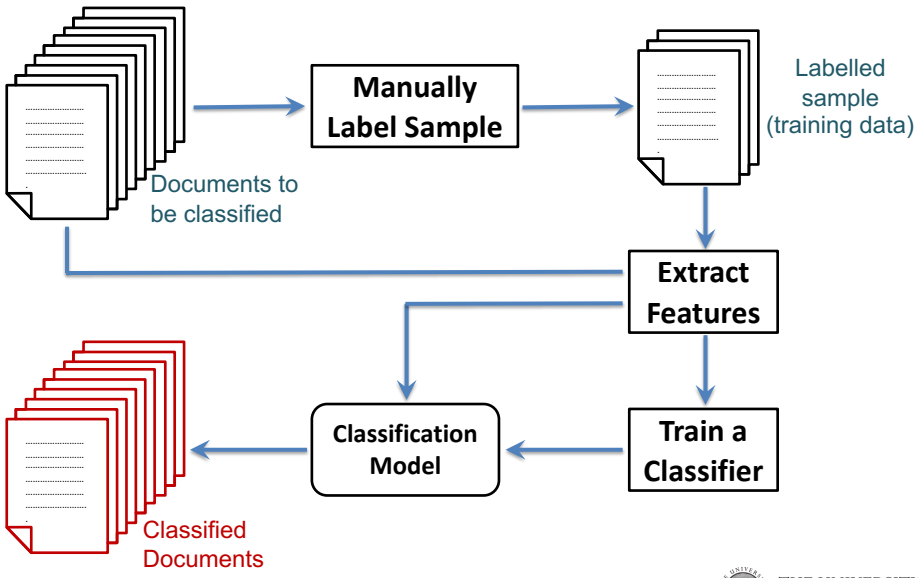
- An old-fashioned way to build text classifiers was via knowledge engineering, i.e., manually building classification rules
  - E.g., (Viagra or Sildenafil or Cialis) → Spam
  - E.g. (#MAGA or America great again) → support Trump
- Common type: dictionary-based classification
- Disadvantages:
  - Expensive to setup and to maintain
  - Depends on few keywords → bad coverage (recall)

# Supervised-learning classification

- 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 classification



## Extract Features

- 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
- In order to generate a vector-based representation for a set of documents *D*, the following steps need to be taken
  1. Feature Extraction
  2. Feature Selection or Feature Synthesis (optional)
  3. Feature Weighting



## Step 1: Feature Extraction

- What are the features that should be different from one class to another?
- Simplest form: Bag-of-words (BOW)
  - Each term in a document is a feature
  - Feature space size = vocabulary in all docs
  - Standard IR preprocessing steps are usually applied
    - Tokenisation, stopping, stemming

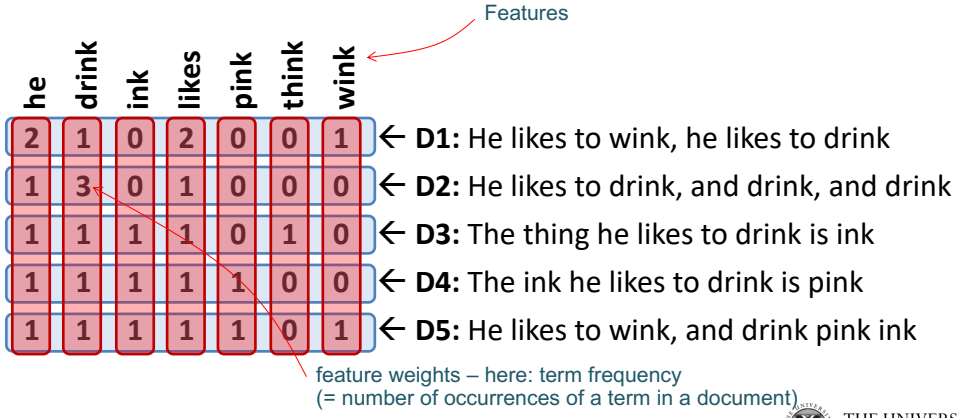
# Step 1: Feature Extraction

- Bag-of-words (BOW)
  - Recall from Indexing lecture how we represented **documents** and words as vectors



# Step 1: Feature Extraction

- Bag-of-words (BOW)
  - Recall from Indexing lecture how we represented documents and **words** as vectors



## Step 1: Feature Extraction

- Other simple features forms:
  - Word n-grams (bigrams, trigrams, ….)
    - Much larger + more sparse
  - Sometimes char n-grams are used
    - Especially for degraded text (OCR or ASR outputs)



## Step 1: Feature Extraction

- What other text features could be used?
- Sentence structure:
  - POS (part-of-speech tags)
  - Syntactic tree structure
- Topic-based features:
  - LDA topics
  - NEs (named entities) in text
  - Links / Linked terms
- Non-textual features:
  - Average doc\sentence\word length
  - % of words start with upper-case letter
  - % of links/hashtags/emojis in text

## Step 1: Feature Extraction

- What preprocessing to apply?
  - Case-folding? really vs Really vs REALLY
  - Punctuation? "?", "!", "@", "#"
  - Stopping? "he", "she", "what", "but"
  - Stemming? "replaced" vs "replacement"
- Other Features:
  - Starts with capital letter, all caps
  - Repeated characters "congraaaaaats" "help!!!!!!!!"
  - Scores from dictionaries and lexicons (e.g. LIWC)
- Which to choose?
  - Classification task/application



## Step 2: Feature Selection

- Number of distinctive features = length of feature vector
- Vector can be of length in the order of $10^6$, and might be sparse
  → High computational cost
  → Overfitting
- What are the most important features among those?
  - e.g. Reduce from $10^6$ to $10^4$
- For each class, find the top representative *k* features for it → get the Union over all classes → reduced feature space

# Step 2: Feature Selection Functions

- Document frequency
  - % of docs in class $c_i$ that contain the term $t_k$
  - Very basic measure. Will select stop words as features

$$\#(t_k, c_i) = P(t_k|c_i)$$

- Mutual Information
  - How much we learn from the presence or absence of term $t_k$ about whether or not a document is in class $c_i$
  - Often used in feature selection in text classification

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

- Pearson's Chi-squared ($x^2$)
  - used more in comparisons between classes



# Step 2: Feature Selection Functions

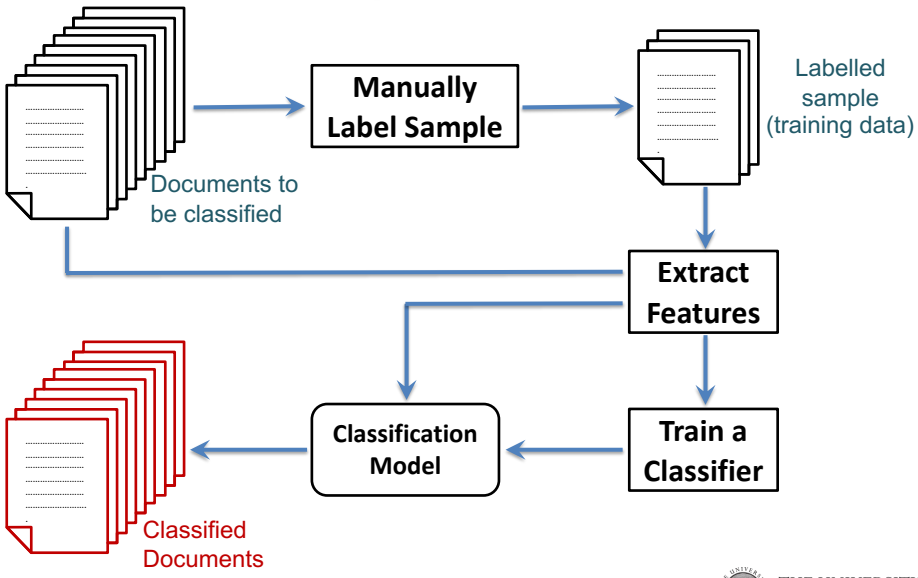
| Function | Denoted by | Mathematical form |
|---|---|---|
| *Document frequency* | $\#(t_k, c_i)$ | $P(t_k\|c_i)$ |
| *DIA association factor* | $z(t_k, c_i)$ | $P(c_i\|t_k)$ |
| *Information gain* | $IG(t_k, c_i)$ | $\sum_{c \in \{c_i, \bar{c}_i\}} \sum_{t \in \{t_k, \bar{t}_k\}} P(t, c) \cdot \log \frac{P(t, c)}{P(t) \cdot P(c)}$ |
| *Mutual information* | $MI(t_k, c_i)$ | $\log \frac{P(t_k, c_i)}{P(t_k) \cdot P(c_i)}$ |
| *Chi-square* | $\chi^2(t_k, c_i)$ | $\frac{\|Tr\| \cdot [P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)]^2}{P(t_k) \cdot P(\bar{t}_k) \cdot P(c_i) \cdot P(\bar{c}_i)}$ |
| *NGL coefficient* | $NGL(t_k, c_i)$ | $\frac{\sqrt{\|Tr\|} \cdot [P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)]}{\sqrt{P(t_k) \cdot P(\bar{t}_k) \cdot P(c_i) \cdot P(\bar{c}_i)}}$ |
| *Relevancy score* | $RS(t_k, c_i)$ | $\log \frac{P(t_k\|c_i) + d}{P(\bar{t}_k\|\bar{c}_i) + d}$ |
| *Odds Ratio* | $OR(t_k, c_i)$ | $\frac{P(t_k\|c_i) \cdot (1 - P(t_k\|\bar{c}_i))}{(1 - P(t_k\|c_i)) \cdot P(t_k\|\bar{c}_i)}$ |
| *GSS coefficient* | $GSS(t_k, c_i)$ | $P(t_k, c_i) \cdot P(\bar{t}_k, \bar{c}_i) - P(t_k, \bar{c}_i) \cdot P(\bar{t}_k, c_i)$ |

## Step 3: Feature Weighting

- Attributing a value to feature $t_k$ in 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 $d_i$ );
    obtained via feature weighting functions in the following two classes:
    - unsupervised: e.g., tfidf or BM25,
    - supervised: e.g., $tf * MI$, $tf * x^2$
- Similarity between two vectors may be computed e.g. via cosine similarity
- Scaling can be important!



## Supervised-learning classification



## Training a Classifier

- For binary classification, essentially any supervised learning algorithm can be used for training a classifier; classical choices include
  - Support vector machines (SVMs)
  - Random forests
  - Naïve Bayesian methods
  - Lazy learning methods (e.g., k-NN)
  - Logistic Regression
  - ….
- 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
  - the sparse nature of the representations involved



## Training a Classifier

- For Multiclass classification, some learning algorithms for binary classification are "SLMC-ready"; e.g.
  - Decision trees
  - Random forests
  - Naive Bayesian methods
  - Lazy learning methods (e.g., k-NN)
  - Neural networks
- For other learners (notably: SVMs) to be used for SLMC classification, combinations / cascades of the binary versions need to be used
  - e.g. multi-class classification SVM
  - Could be directly used for MLMC as well

# Word embeddings

- More complex representation of words as vectors
  Recall the **term vectors** in traditional indexing or classification, using term frequency as weights:
  - Sparse (most values are 0)
  - Capture semantics only incidentally (similar vectors are terms that appear together)

| he | drink | ink | likes | pink | think | wink | |
|---|---|---|---|---|---|---|---|
| 2 | 1 | 0 | 2 | 0 | 0 | 1 | ← **D1:** He likes to wink, he likes to drink |
| 1 | 3 | 0 | 1 | 0 | 0 | 0 | ← **D2:** He likes to drink, and drink, and drink |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | ← **D3:** The thing he likes to drink is ink |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | ← **D4:** The ink he likes to drink is pink |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | ← **D5:** He likes to wink, and drink pink ink |



# Word embeddings

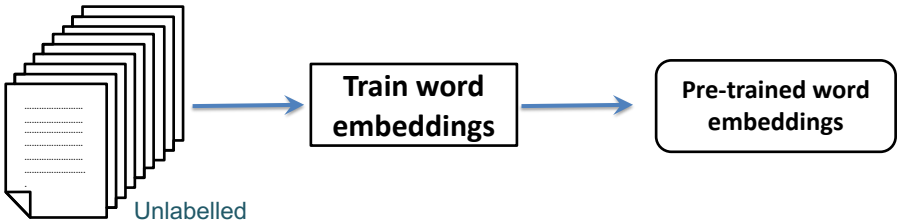
- More complex representation of words as vectors
  - Dense (all entries are non-zero)
  - Capture semantics (similar words have similar vectors)

| he | drink | ink |
|---|---|---|
| 0.123 | 0.521 | 0.313 |
| 0.451 | 0.987 | 0.812 |
| 0.938 | 0.141 | 0.411 |
| ... | ... | ... |

(many dimensions e.g. 300)

# Word embeddings

- Obtained through self-supervised learning to learn relationships between words
  - Predict a centre word given surrounding context words (CBOW)
  - Predict context words given target word (skip-gram)
- Can be done on a large unlabelled pre-training corpus
- Helps with out-of-vocabulary problems

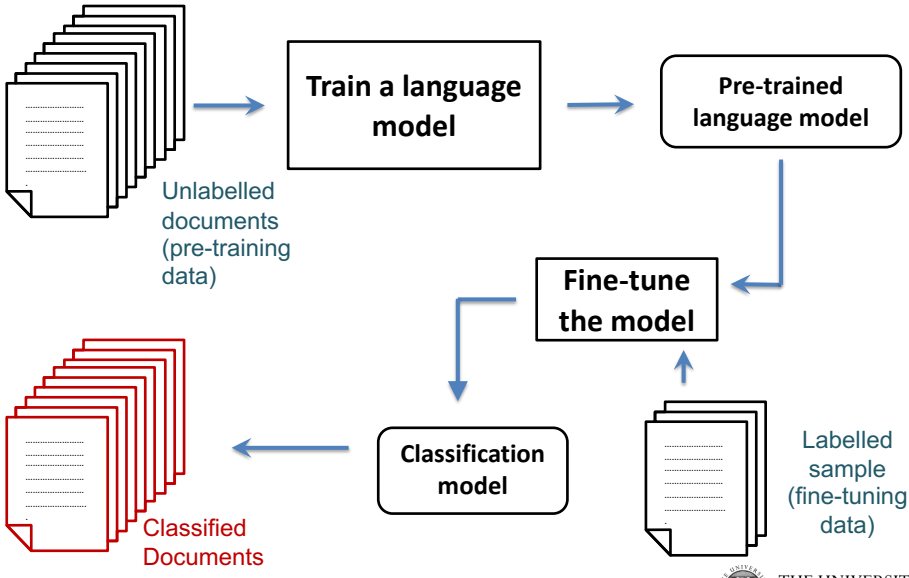





# Pre-trained language models

- Obtained through self-supervised learning
  - Predict the next word: The Queen of […] *(next token prediction)*
  - Predict a masked word: The […] of England. *(masked language modelling)*
  - …
- Also done on large unlabelled pre-training corpora
- Penultimate layer of network can be used to generate **contextualised word embeddings** for other language-based tasks
- Basis for many state-of-the-art text classifiers
  - e.g. BERT (DistilBERT, RoBERTa..), XLNet, etc.

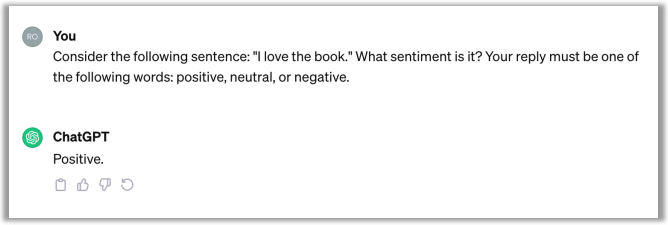
# Supervised fine-tuning



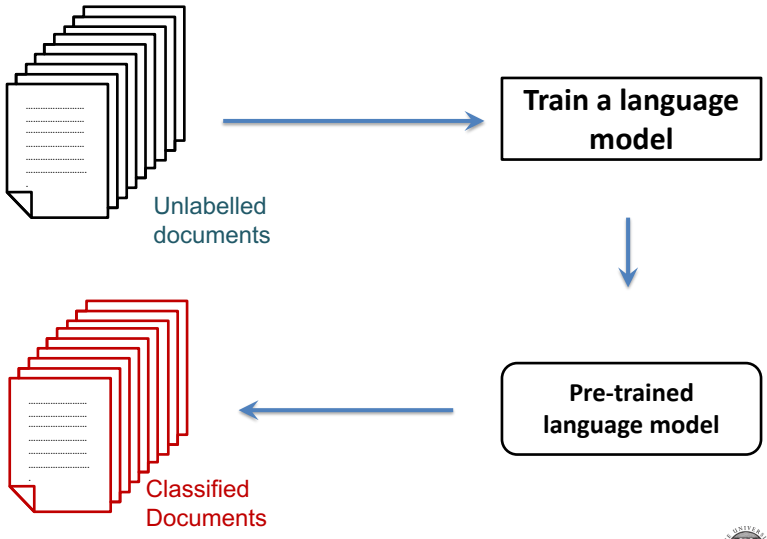



# Zero-shot classification

- Using language models for classification directly without giving training examples (= skipping the fine-tuning step)
  - A modern approach that can work well and requires little human effort
  - Depends highly on black-box models, limited opportunities for customisation and error analysis (but this is an active research area)
  - Often used as a baseline for performance comparisons



# Zero-shot classification






# Few-shot classification

- Variation of zero-shot classification: A few examples are given in the prompt
- Also skips task-specific fine-tuning
- Leverages models' capability at **in-context learning** (adapt to a task from examples in the input prompt)

Task: Identify the stance toward carbon tax in the given sentence.
Your reply must be one of: support, oppose, neutral.

Examples:

Sentence: "I think a carbon tax is necessary to fight climate change."
Target: carbon tax → support

Sentence: "Renewable energy is amazing, though I don't have strong feelings about a carbon tax."
Target: carbon tax → neutral

Now classify this sentence:
Sentence: "I support policies to reduce emissions, but I'm worried about how a carbon tax will affect me personally."
Target: carbon tax →

Target: **carbon tax → oppose**

# Four routes, one tweet

*"Kobe passes Wilt for 4th on all-time scoring list"* → **Sports**

| Route | What represents the tweet | What we fit on our labelled data | Labelled data used |
|---|---|---|---|
| **1. Sparse features + SVM** | counts of kobe, passes, wilt, 4th, … (unseen words: OOV) | vocabulary, idf, feature selection, SVM weights | training set (thousands) |
| **2. Frozen embeddings + classifier** | a sentence vector from a pretrained encoder (384 numbers for MiniLM) | logistic regression only | training set |
| **3. Fine-tuning** | the language model's own representation | all model weights + a classification layer | training set; GPU |
| **4. Prompting (zero-/few-shot)** | the prompt: label names (+ a few examples) | nothing; but label wording, examples and parsing rule are chosen | 0 to a few examples |

**All four routes still need a labelled development set and test set.**



# Evaluation

# Evaluation

- Effectiveness (e.g. accuracy, precision, recall, F1):
  - Global effectiveness measures
  - Per class effectiveness measures
- Efficiency:
  - Speed in learning
    - SVM with linear kernel is known to be fast
    - DNNs are known to be much slower (specially with large # layers)
  - Speed in classification
    - K-NNs are known to be one of the slowest
  - Speed in feature extraction
    - BOW vs POS vs Link analysis features
- Importance of baselines

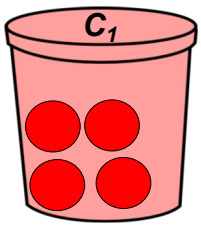


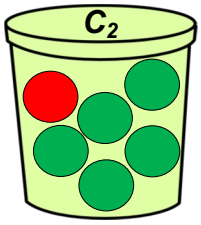
# Evaluation: Baselines

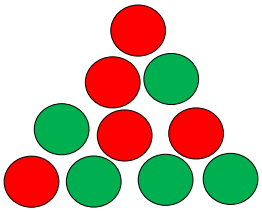
- There are standard methods for creating baselines in text classification to compare your classifier with
- Most popular/simplest baselines
  - Random classification
    - Classes are assigned randomly
    - How much better is the classifier doing than random?
  - Majority class baseline
    - Assign all elements to the class that appears the most
    - How much better you are doing than if you always picked the same thing output regardless of input?
  - Simple algorithm, e.g. BOW
    - Usually used when you introduce new interesting features
  - Prompted LLMs (zero-/few-shot)
    - Can sometimes outperform fine-tuned models

# Evaluation: Binary Classification

- Accuracy:
  - How many of the samples are classified correctly?
- A = (4+5)/10 = 0.9



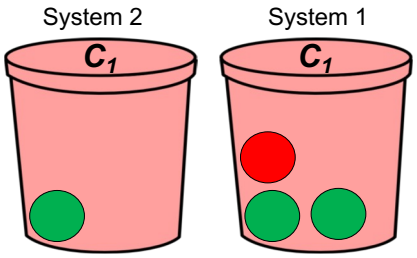



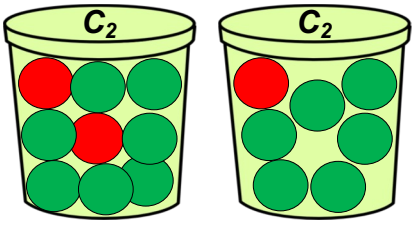



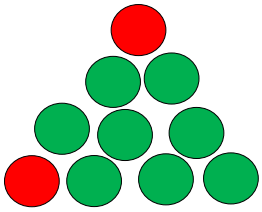


# Evaluation: Binary Classification

- A = (1+6)/10 = 0.7 System 1
- A = (0+7)/10 = 0.7 System 2
- When classes are highly unbalanced
  - Precision/recall/F1 for the **rare class**
  - e.g. Spam classification (detection)

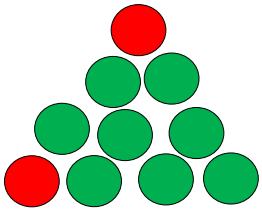


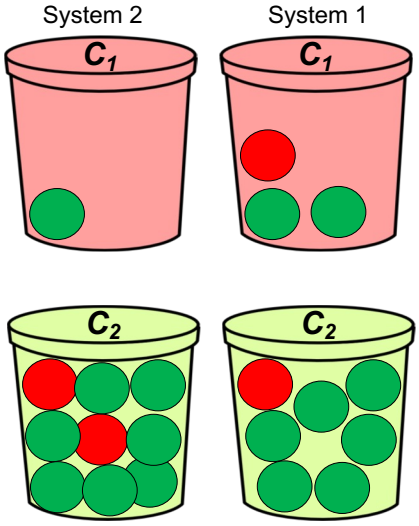



# Evaluation: Binary Classification

| | System 1 | System 2 |
|---|---|---|
| Precision | 1/3 = 0.33 | 0/1 = 0 |
| Recall | 1/2 = 0.5 | 0/2 = 0 |
| F1 | 0.4 | 0 |





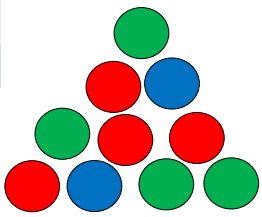


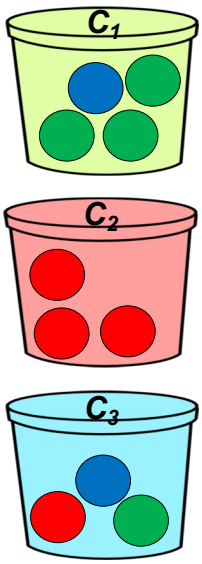

# Evaluation: Multi-class

- Accuracy = (3+3+1)/10 = 0.7
- Good measure when
  - Classes are nearly balanced
- Preferred:
  - Precision/recall/F1 for each class

| | | | |
|---|---|---|---|
| P | 0.75 | 1 | 0.333 |
| R | 0.75 | 0.75 | 0.5 |
| F1 | 0.75 | 0.86 | 0.4 |

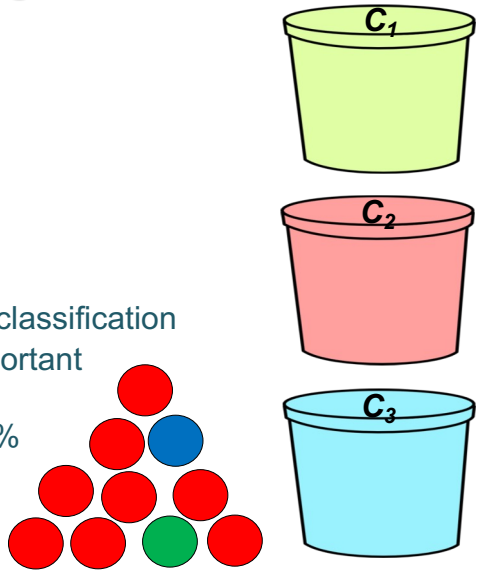
- **Macro-F1**
  = (0.75+0.86+0.4)/3
  = **0.67**





# Evaluation: Multi-class

- Majority class baseline
  - Accuracy = 0.8
  - Macro-F1 = 0.296
- Macro-F1:
  - Should be used in binary classification when two classes are important
  - e.g.: males/females while distribution is 80/20%

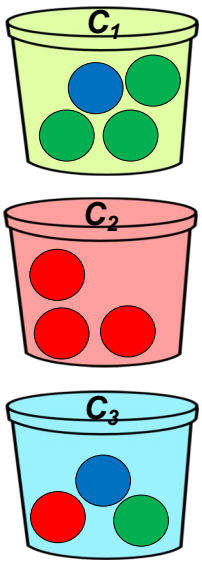





# Error Analysis

- **Confusion Matrix**

| | | Predicted class | | |
|---|---|---|---|---|
| | | 🟢 | 🔴 | 🔵 |
| Actual class | 🟢 | 3 | 0 | 1 |
| | 🔴 | 0 | 3 | 1 |
| | 🔵 | 1 | 0 | 1 |




- Useful:
  - Find classes that are confused with others
  - Develop better features to solve the problem

## Data splitting

- It's important to avoid overfitting
- Labelled data could be split into two parts
  - Training: used to train the classifier (e.g. **80%** of the data)
  - Test: used to test the performance of the trained classifier on unseen data (e.g. **20%** of the data)



## Hyperparameter optimisation

- Most classifiers have some hyperparameters to be optimized
  - The C parameter in soft-margin SVMs
  - The *r*, *d* parameters of non-linear kernels
  - Decision threshold for binary SVM
- We may also try different models (SVM, Fine-tuned RoBERTa, a prompted LLM…) so we could overfit to this choice
- Usually labelled data is split into three parts
  - Training: used for training / fine-tuning (typically **80%** of the data)
  - Development: used to optimise hyperparameters. Apply the classifier on this data with different values of the hyperparameters and report the one that achieves the highest results (usually **10%** of the data)
  - Test: used to test the performance of the trained classifier with the optimal hyperparameters on these unseen data (usually **10%** of the data)
- Optimising the hyperparameters on test data is cheating!

## Keeping the test data unseen

- Fit everything on the training split only
  - vocabulary, idf, feature selection (MI, $\chi^2$), scaling, the classifier
  - few-shot examples come from the training data
- Make every choice on the development split
  - hyperparameters, features, model, prompt wording, parsing rule
- Use the test set once, at the end
- The split must match the claim
  - "more tweets like these" → random, stratified split
  - "new users" → no user in more than one split
  - "next year" → train on earlier, test on later data
  - retweets and near-duplicates → keep copies in the same split
- For pre-trained models, including hosted LLMs: was the test data for your task in their training data?



## Summary: can you…

- name the steps of a supervised text classification pipeline, from labelling to evaluation?
- say what the training, development and test sets are used for?
- compute accuracy and macro-F1 from a confusion matrix, and explain why the majority baseline matters?
- describe four routes to a tweet classifier and what labelled data each one needs?
- choose a split that matches the claim you want to make?

# Resources

- *Fabrizio Sebastiani*
  **Machine Learning in Automated Text Categorization**
  *ACM Computing Surveys, 2002*
  *Link: https://arxiv.org/pdf/cs/0110053*
- *Yoav Goldberg*
  **A Primer on Neural Network Models for Natural Language Processing**
  *Link: https://arxiv.org/abs/1510.00726*
- *Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D.* (2020). **Language models are few-shot learners**. Advances in Neural Information Processing Systems, *33*, 1877-1901.