ECS763P/U: NATURAL LANGUAGE PROCESSING

Lecture 2: Text Classification 1

Arkaitz Zubiaga

Contents

- Text preprocessing: why?
  - Word tokenisation.
  - Text normalisation.
  - Stopword removal.
  - Lemmatisation and stemming.
  - Sentence segmentation.

- What is text classification?
  - Examples of text classification.

- Evaluation.
- Error Analysis.

Text Preprocessing: Why?

- Inconsistent use of words:

  Welcome to the UK

  Welcome to the U.K.

  Welcome to the uk

  Welcome to the u.k.

Text Preprocessing: Why?

- Linguistic variations with similar meanings.

  I am **happy** today.

  I am **happier** than yesterday.

- For sentiment analysis, maybe all we need to know is that both have the word “happy”, irrespective of the variation.

Text Preprocessing

- Many NLP tasks need to do text preprocessing:
  - Stopword removal.
  - Segmenting/tokenising words in running text.
  - Normalising word formats.
  - Segmenting sentences in running text.

Stop Word Removal

- Some words are more **meaningful** than others.

  I am excited to be a member of Team GB!

  Words like **"to", "be", "a", "of" are not very meaningful** for many analyses.

  We may consider removing them to focus on the rest of the words.

Stop Word Removal

- Words like **“to”, “be”, “a”, “of” are not very meaningful** for some analyses.
  - We call them “stop words”, i.e. frequently used words that are not informative for some tasks.
  - If they’re not useful for our task, we may remove them.
  - NLTK provides a list of stop words.

Corpora

- **Corpus:** collection or dataset of text or speech. One or more documents, e.g. corpus with all of Shakespeare's works.

- We can split each document/text into **sentences**, and these sentences into **words**.

Sentences

- **Sentences:** shortest sequence of words that are grouped together to convey some grammatically correct self-contained meaning.

- How do we go about splitting a text into sentences?

  - For practical purposes, sequence between full stops or “?|!|:|;”.

  - We can do more advanced segmentation, e.g. break long sentences with conjunctions like “and” or “or”.

How Many Words in a Sentence?

- It can be as simple as **counting** the elements we get after **splitting a text by spaces**, but it depends.

- How many words in the following?

  My *cat is different from other cats.*

Words and Lemmas

- My **cat** is different from other **cats**.

- Cat and cats both have the same **lemma** (cat), but two different **wordforms**:
  - Cat: cat (lemma)
  - Cats: cat (lemma) + s (suffix)

How Many Words?

- **Type:** a unique element of the vocabulary.
- **Token:** an instance of that type in the running text.

How Many Words?

- **Type:** a unique element of the vocabulary.
- **Token:** an instance of that type in the running text.

The house on the hill is the best

8 tokens.
6 types: the, house, on, hill, is, best.
(3 of the tokens belong to the same type, ‘the’)

If we remove stop words (on, the), these numbers will decrease.

Corpora

- **N** = number of tokens
- **V** = vocabulary = set of types
  - **|V|** is the size of the vocabulary

	Tokens = N	Types = |V|
Switchboard phone conversations	2.4 million	20 thousand
Shakespeare	884,000	31 thousand
Google N-grams	1 trillion	13 million

Word Tokenisation

- The Complete Works of William Shakespeare (shakes.txt[1]):

Open shakes.txt ~/Desktop

```
The Project Gutenberg EBook of The Complete Works of William Shakespeare, by
William Shakespeare

This eBook is for the use of anyone anywhere at no cost and with
almost no restrictions whatsoever.  You may copy it, give it away or
re-use it under the terms of the Project Gutenberg License included
with this eBook or online at www.gutenberg.org

** This is a COPYRIGHTED Project Gutenberg eBook, Details Below **
**     Please follow the copyright guidelines in this file.     **

Title: The Complete Works of William Shakespeare

Author: William Shakespeare

Posting Date: September 1, 2011 [EBook #100]
Release Date: January, 1994

Language: English


*** START OF THIS PROJECT GUTENBERG EBOOK COMPLETE WORKS--WILLIAM SHAKESPEARE ***




Produced by World Library, Inc., from their Library of the Future
```


1 https://raw.githubusercontent.com/matth/ruby-nlp/master/corpora/shakes.txt

Simple Tokenisation in UNIX

```
> tr -sc 'A-Za-z' '\n' < shakes.txt
```

- Having shakes.txt as input (< shakes.txt)
- Convert all non-alphabetic characters (-sc ‘A-Za-z’)
- Into new lines (‘\n’)

Step 1: Tokenising

```
> tr -sc 'A-Za-z' '\n' < shakes.txt | head
The
Project
Gutenberg
EBook
of
The
Complete
Works
of
...
```

Step 2: Sorting

```
> tr -sc 'A-Za-z' '\n' < shakes.txt | sort | head
a
a
a
a
a
a
a
a
a
...
```

Step 3: Counting

```
> tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c | head
  12851 a
   1949 A
     25 Aaron
     72 AARON
      1 abaissiez
     10 abandon
      2 abandoned
      2 abase
      1 abash
...
```

Step 4: Sort by Count

```
> tr -sc 'A-Za-z' '\n' < shakes.txt | sort | uniq -c | sort -r -n | head
  23455 the
  22225 I
  18715 and
  16433 to
  15830 of
  12851 a
  12236 you
  10840 my
  10074 in
   8954 d  → what happened here?
...
```

Step 5: Lowercasing Text

```
> tr -sc 'A-Za-z' '\n' < shakes.txt | tr 'A-Z' 'a-z' | sort | uniq -c |
sort -r -n | head
  27843 the
  26847 and
  22538 i
  19882 to
  18307 of
  14800 a
  13928 you
  12490 my
  11563 that
  11183 in
...
```

Issues in Tokenisation

- England’s capital → England, Englands, England’s
- what’re, I’m, isn’t → What are, I am, is not
- Hewlett-Packard → Hewlett Packard
- state-of-the-art → state of the art ?
- lower-case, lowercase, lower case
- Leamington Spa → one token or two?
- U.K./UK, U.S.A./USA

Tokenisation in NLTK

- NLTK has a package with lots of functionalities for tokenisation:

  https://www.nltk.org/api/nltk.tokenize.html

Tokenisation: Language Issues

- French:
  - **L'ensemble** → one token or two?
  - L ? L’ ? Le ?
  - We want l’ensemble to match other instances of ensemble

- German noun compounds are not segmented:
  - **Lebensversicherungsgesellschaftsangestellter**
  - ‘life insurance company employee’
  - German information retrieval needs compound splitter

Tokenisation in Chinese

- Also called **Word Segmentation**.

- Chinese words are composed of characters:
  - Characters are generally 1 syllable and 1 morpheme.
  - Average word is 2.4 characters long.

- Standard baseline segmentation algorithm:
  - **Maximum Matching** (also called Greedy)

Maximum Matching Algorithm

- Given a wordlist (dictionary) of Chinese, and a string as input:

  1) Start a pointer at the beginning of the string.
  2) Find the longest word in dictionary that matches the string starting at pointer.
  3) Move the pointer over the word in string.
  4) Go to 2.

Maximum Matching: Example in English

- Thetabledownthere

  Longest dictionary word from the beginning is ‘theta’, but we wanted ‘the’.

  We get ‘Theta bled own there’

  We probably wanted ‘The table down there’ though!

It’s quite a **bad algorithm for English**!

Maximum Matching: Example in Chinese

- But it’s actually very good for Chinese:
  - 莎拉波娃现在居住在美国东南部的佛罗里达。
  - 莎拉波娃 现在 居住 在 美国 东南部 的 佛罗里达

WORD NORMALISATION AND STEMMING

Normalisation

- 2 types of normalisation. Let’s think of a Google search query.
  - Symmetric normalisation:
    - User searching for ‘U.S.A.’ or ‘USA’ most likely looking for the same.
  - Asymmetric normalisation:
    - User enters ‘Windows’, we give results for ‘Windows’ (operating system)
    - User enters ‘windows’, do we give results for both ‘Windows’ and ‘windows’?

Case Folding

- Often the case is not meaningful, e.g. ‘the’ vs ‘The’.
  - We may reduce all to **lowercase**.
  - Possible exception: upper case in mid-sentence?
    - e.g., **General Motors**
- But the case is sometimes important!
  - e.g. **US vs us**

Lemmatisation and Stemming

- In both cases, we aim to **reduce vocabulary size**.
  - e.g. ‘cars’ and ‘car’ will both become ‘car’.

- **Lemmatisation:** finding dictionary headword form.

- **Stemming:** finding the stem by stripping off suffixes, usually using regular expressions.

Lemmatisation

- Reduce inflections or variant forms to headword form:
  - am, are, is → be
  - car, cars, car's, cars' → car
  - those cars are really beautiful → those car be really beautiful

- PRO: we guarantee that we get dictionary words.
- CON: costly and more complex, need to infer the meaning of each word.
  - Reading (verb) → read
  - Reading (city) → Reading

Stemming

- Reduce by stripping suffixes **following certain rules** (e.g. regular expressions):
  - am, are, is → am, ar, is
  - car, cars, car's, cars' → car, car, car ' s, car '
  - those cars are really beautiful → those car ar realli beauti
- CON: It does shorten words and reduce vocabulary, however **not always leading to dictionary words**!
- PRO: It’s much **faster than a lemmatiser**.

Porter: Well-known English Stemmer

Step 1a

```
sses → ss    posesses → posess
ies  → i     ponies   → poni
ss   → ss    posess   → posess
s    → ø    cats      → cat
```

Step 1b

```
(*v*)ing → ø  walking   → walk
               sing      → sing
(*v*)ed  → ø  plastered → plaster
…
```

https://tartarus.org/martin/PorterStemmer/

Porter: Well-known English Stemmer

- (*v*)ing → ø (remove ‘ing’ as long as *v* has 3+ chars)
- having → hav, living → liv, studying → study 
- king → ø, sing → ø, thing → ø 
- something → someth, morning → morn 

Lemmatiser vs Stemmer

for example compressed and compression are both accepted as equivalent to compress.

STEMMER:
for exampl compress and compress ar both accept as equival to compress

LEMMATISER:
for example compress and compress be both accept as equivalent to compress

SENTENCE SEGMENTATION

Sentence Segmentation

- !, ? are relatively unambiguous → almost always indicate end of sentence
- But period "." is much more ambiguous
  - Sentence boundary
  - Abbreviations like Inc., etc. or PhD.
  - Numbers like .02% or 4.3

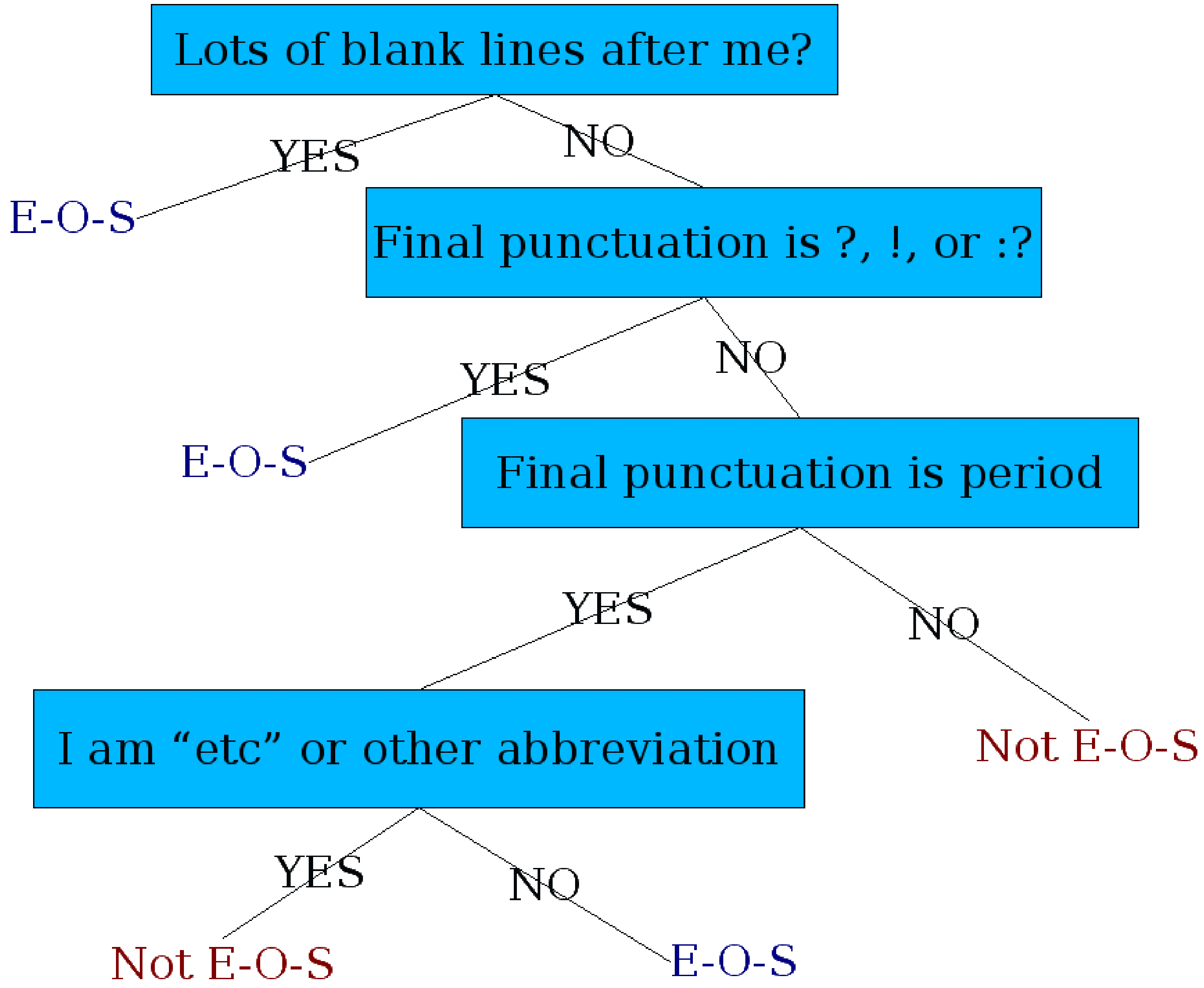
Sentence Segmentation

- So how do we deal with this ambiguity?

- We can build a binary classifier:
  - Look at occurrences of '.'
  - Classifies EndOfSentence vs NotEndOfSentence. How?
    - Hand-written rules (if-then).
    - Regular expressions.
    - Machine learning.

Sentence Segmentation: A Decision Tree

- Deciding if a word is at the end of a sentence.



Segmentation: More Features

- Is the word after the punctuation mark capitalised?
- What is the length of the word preceding the period?
- Are there more periods following? e.g. an ellipsis.
- Is there a space after the period?

TEXT CLASSIFICATION

What is Text Classification?

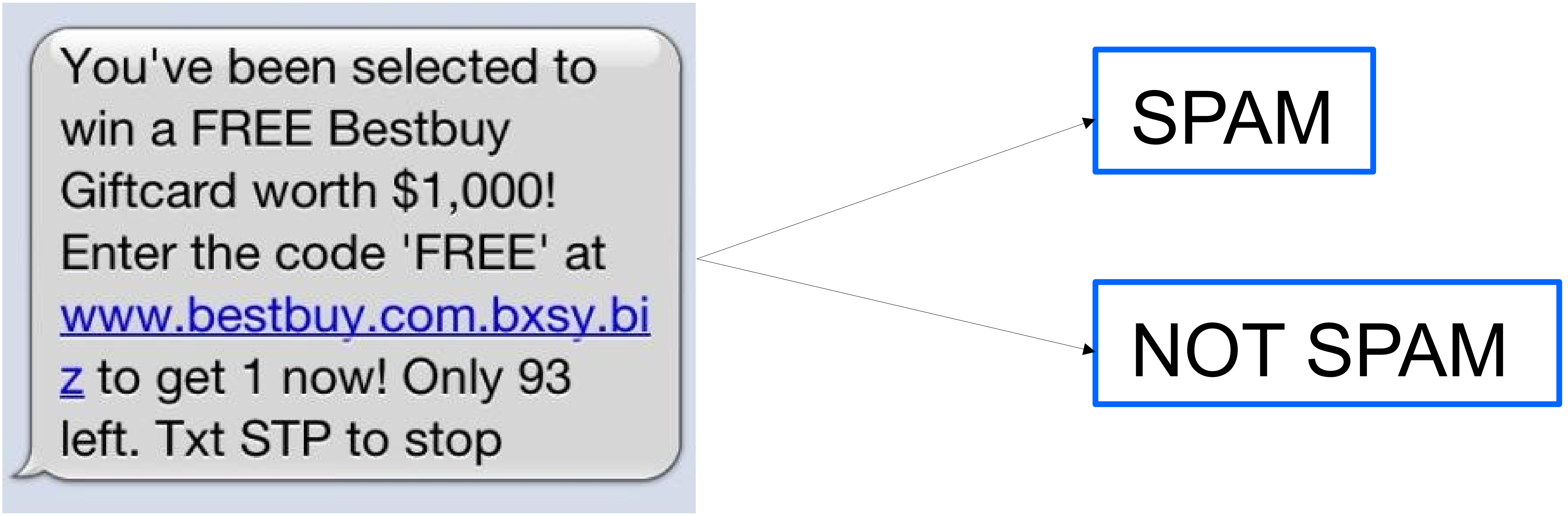
- Having as input:
  - A **text document d**
  - A **set of categories** $\mathbf{C=\{c_1, \ldots, c_m\}}$

- The **text classification** task **outputs**:
  - Predicted class **c*** that **document d belongs to**.

$$c^* \in C$$

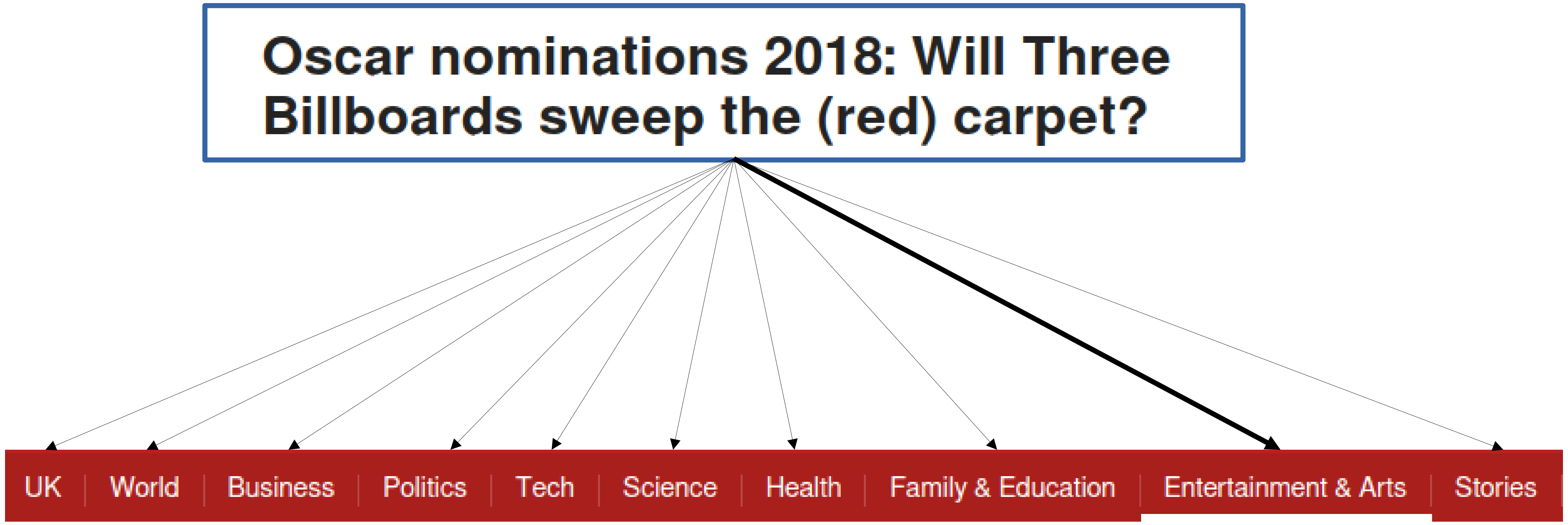
Examples of Text Classification

- **Spam detection**: classifying emails/web pages as spam (or not).



Examples of Text Classification

- **Classification by topic**: what is the text about?



Examples of Text Classification

- **Sentiment analysis**: is a text positive, negative or neutral?

☺ • I really **liked** the food at the restaurant.

😐 • We went to the restaurant for the first time.

☹ • The service was **terrible**.

Examples of Text Classification

- **Language identification**: what language a text is written in?

  Wieviel Uhr ist es? → {German, English, Spanish, French}

What is Text Classification?

- A range of different problems, with a **common goal**:
  - Assigning a **category/class to each document**.
  - We **know the set of categories** beforehand.

Text Classification: Approaches

- **Rule-based classifiers**, e.g. if email contains ‘viagra’ → spam
  - Significant **manual effort** involved defining a comprehensive set of rules.

- **Supervised classification**:
  - **Given**: a hand-labeled set of **document-class pairs** $(d_1,c_1), (d_2, c_j), \ldots, (d_m,c_2) \rightarrow$ classified into $C=\{c_1, \ldots, c_j\}$
  - The classifier **learns a model** that can **classify new documents into C**.

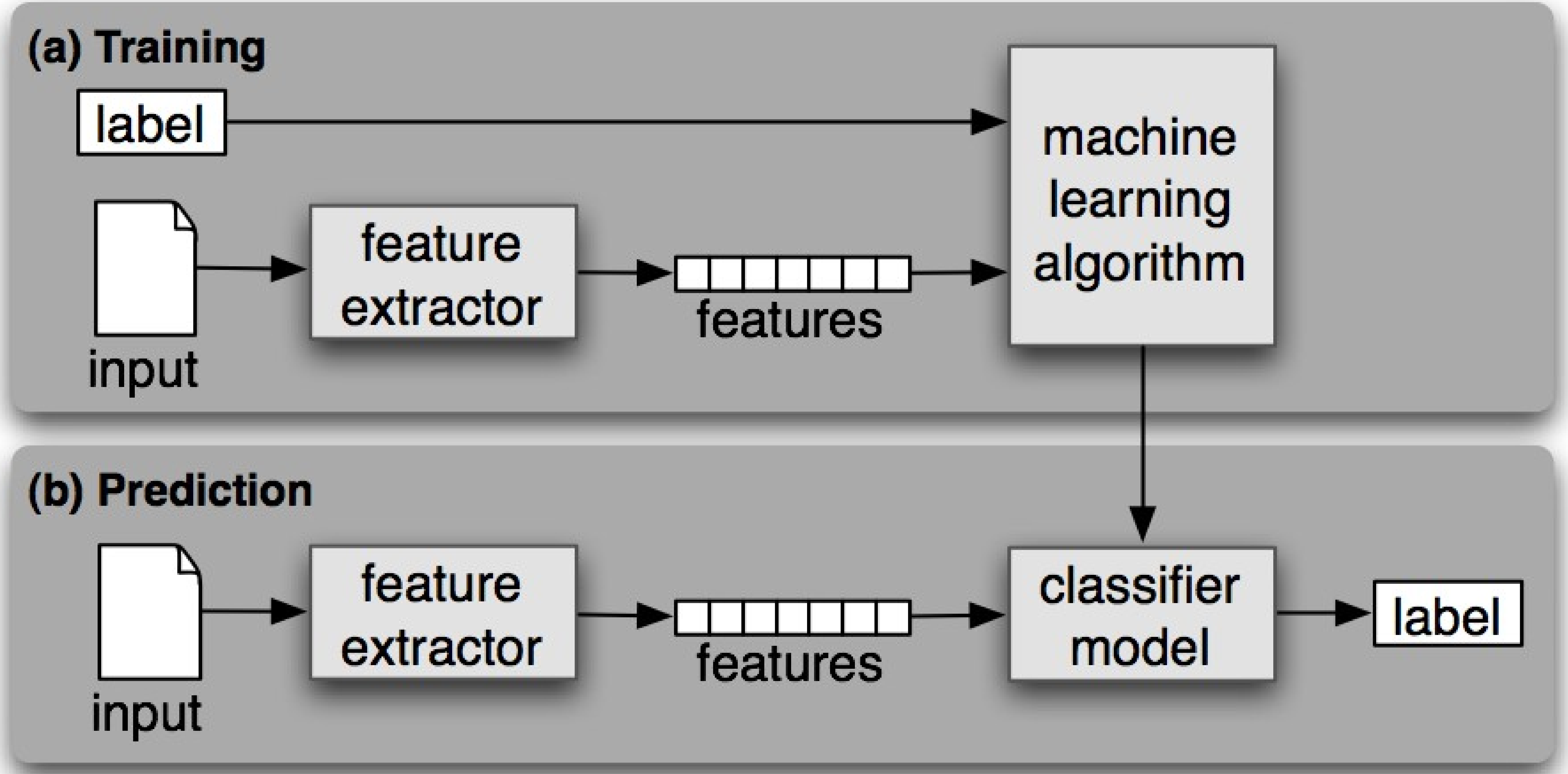
SUPERVISED TEXT CLASSIFICATION

Supervised Classification

- **Assumption:** We have a manually labelled dataset that we can use to train a model, e.g.:
  - d_1: ‘That’s really good, I love it’ → **positive**
  - d_2: ‘It was boring, don’t recommend it’ → **negative**
  - …
  - d_n: ‘I wouldn’t go again, awful’ → **negative**

- **If not**, we need to **find one or label one ourselves**.

Supervised Classification



Supervised Classification: Decisions to Make

- **Split the dataset** into train/dev/test sets.
    - or often just train/test.

- What **features** are we going to use to represent the documents?

- What **classifier** are we going to use?
    - Choose **settings, parameters**, etc. for the classifier.

Splitting the Dataset

- We can **split the dataset into 3 parts**:
  - Training set → normally largest as we want good training.
  - Development set.
  - Test set.

  - **Tweak classifier based on the development set**, then test it on the test set.
    - **Tweaking and testing on the test set may lead to overfitting** (doing the right things specifically for that test set, not necessarily generalisable)

What is Overfitting?

- **Overly adjusting** our classifier and features **to a specific test set**.
- The classifier **may not generalise** to new instances beyond the current test set.

Pretty much like designing a mattress based on a single observation of sleeping position!

How to Avoid Overfitting

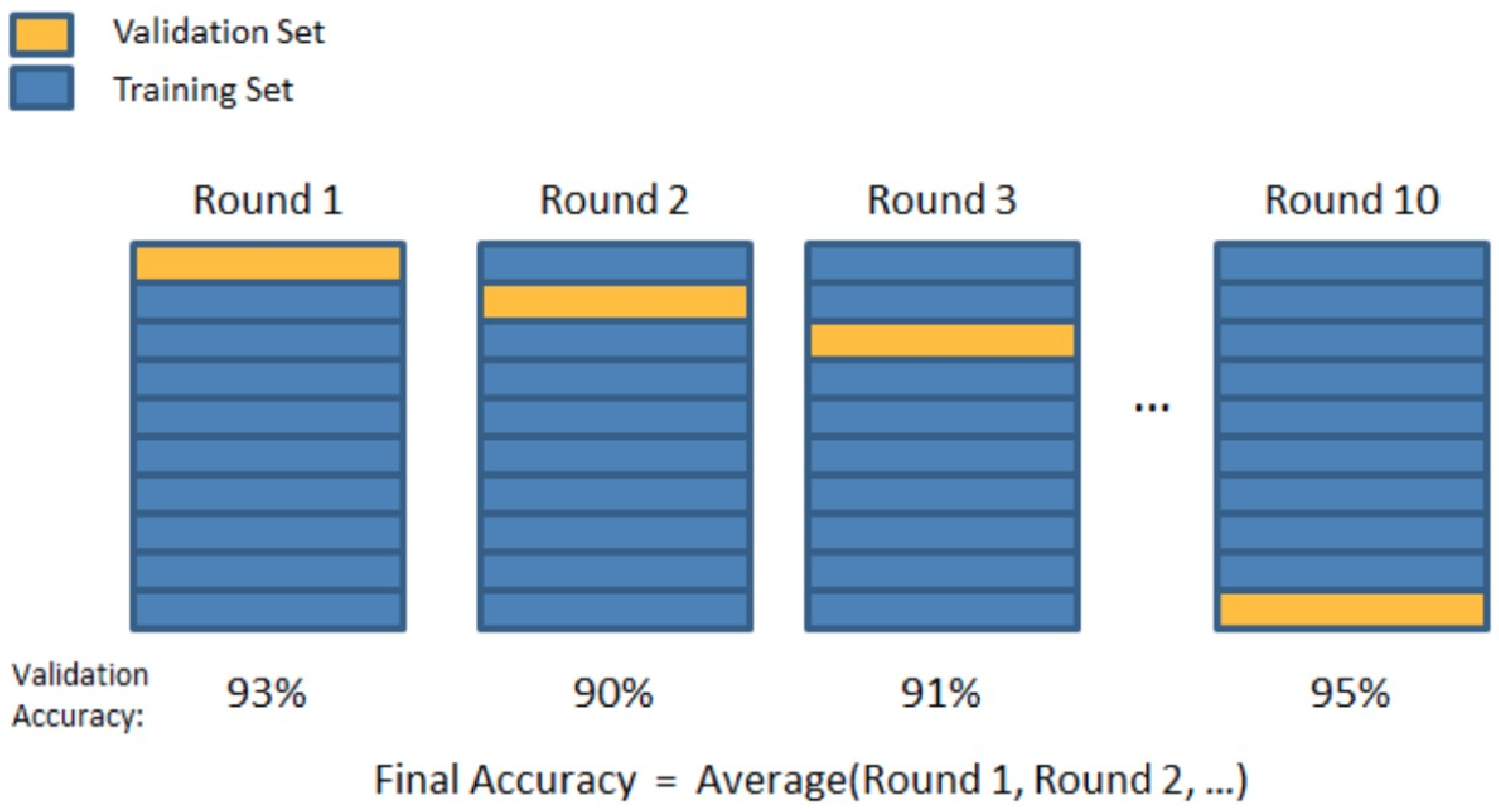
- Having train/dev/test, **tweak our classifier for the dev set**.
  - Then apply to the **test set**, does it **generalise**?

- **Cross-validation**.

Cross-validation

- **Cross-validation:** train and test on different "folds"
  - e.g. 10-fold cross-validation, split the data into 10 parts.
  - each time 1 fold is used for testing, the other 9 for training.
  - after all 10 runs, compute the average performance.

Cross-validation: Example



Choosing the Features

- Usually **start with some basic features**:
  - **Bag of words**.
  - Or **word embeddings** (later in the module)**.**

- Keep **adding new features**:
  - **Need to be creative**.
    Think of features could **characterise the problem at hand**.

Thinking of Features

- **Possible features:**
  - **Sentiment analysis** → counts of positive/negative words.
  - **Language identification** → probabilities of characters (how many k's, b's, v's…), features from word suffixes (e.g. many -ing words → English)
  - **Spam detection** → count words in blacklist, domain of URLs in email (looking for malicious URLs)

Choosing the Features

- How to **assess which features are good**?
- **Empirical evaluation**:
  - **Incremental testing**:
    keep adding features, see if adding improves performance
  - **Leave-one-out testing**:
    test all features, and combinations of all features except one.
    when leaving feature i out performs better than all features, remove feature i
  - **Error analysis:** (later in this lecture)
    look at classifier's errors, what features can we use to improve?

Choosing a Classifier

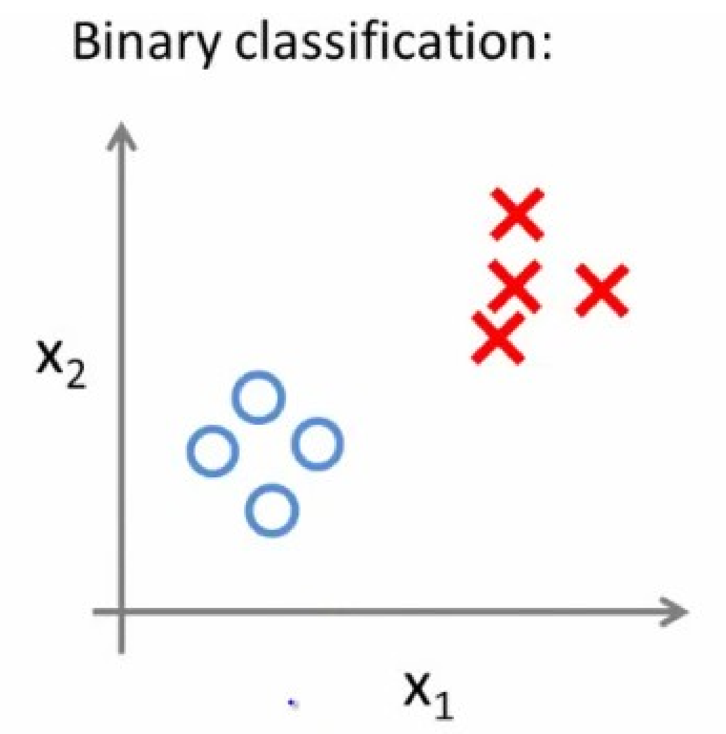
- Many different classifiers exist, **well-known classifiers** include:
  - **Naive Bayes**.
  - **Logistic Regression** (Maximum Entropy classifier)
  - **Support Vector Machines** (SVM).

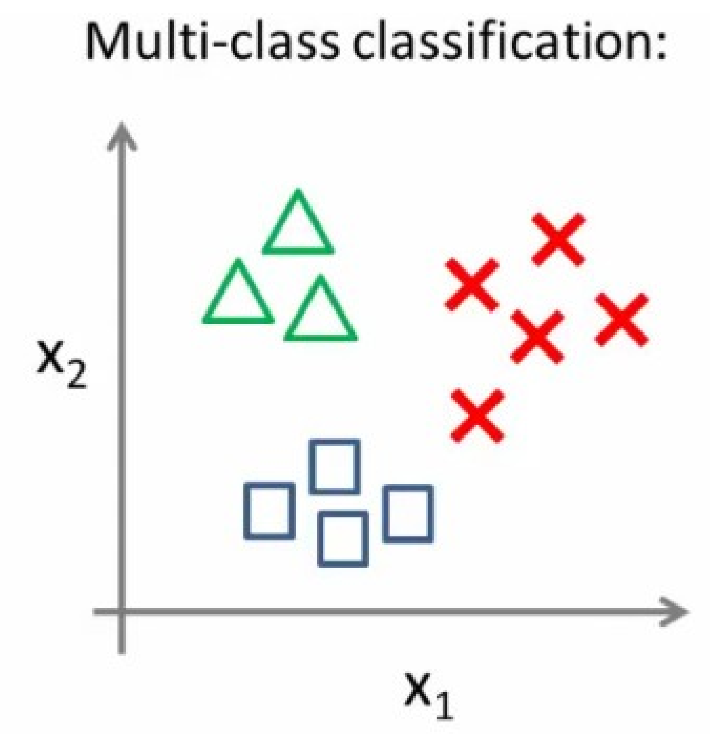
- Classifiers can be **binary** (k = 2) or **multiclass** (k > 2).

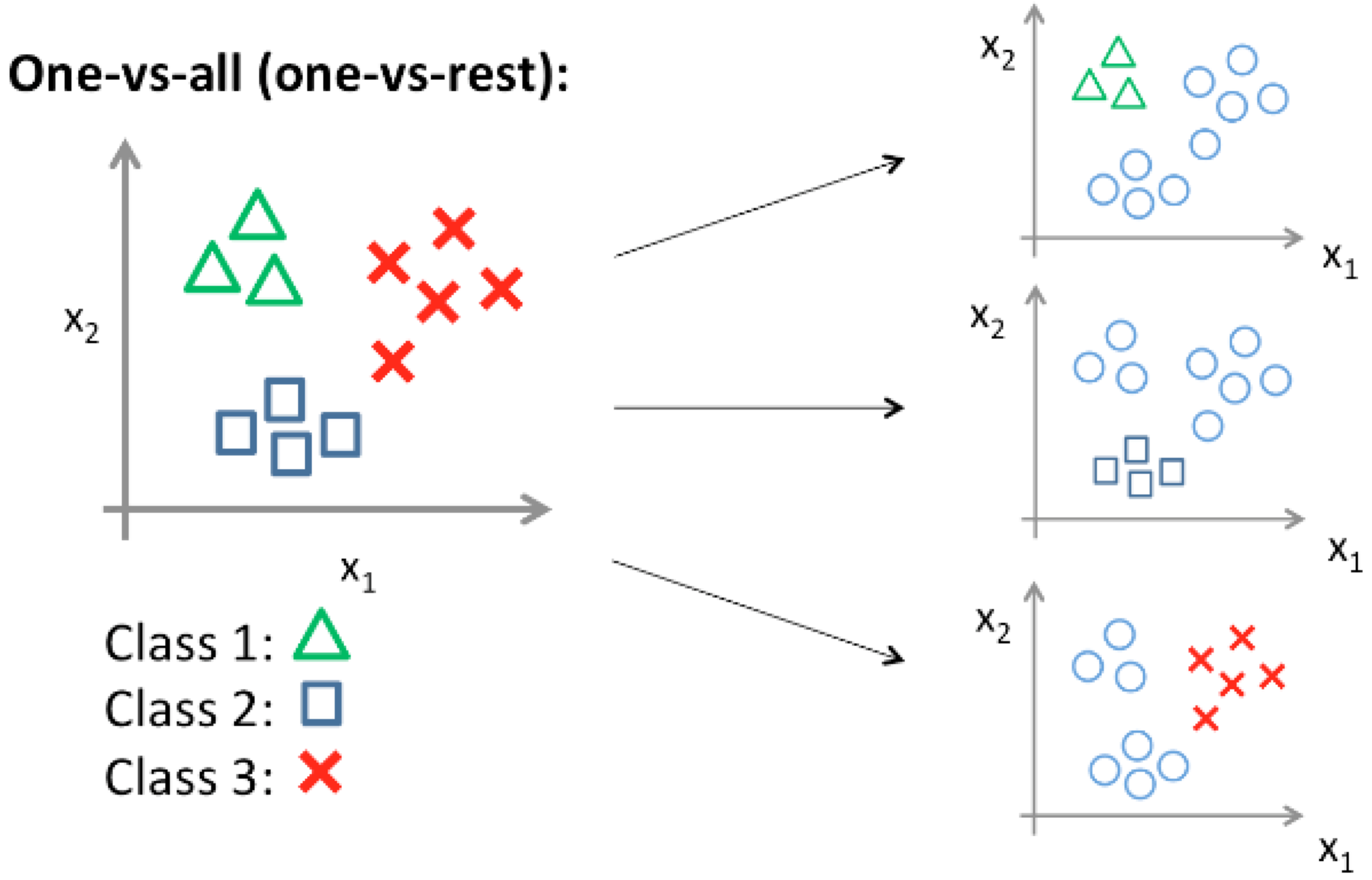
Choosing a Classifier

- **How many categories** (k)?
  - [k=2] **Binary** → binary classifier.
  - [k>2] **Multiclass**:
    - **One-vs-all** classifiers.
      Build *k* classifiers, each able to distinguish class *i* from the rest. Then combine output of all classifiers (e.g. based on their confidence scores)
    - **Multinomial/multiclass** classifiers.

Binary vs Multi-class Classifiers







When to Use Multinomial or One-vs-all?

- **Multinomial:**
  - **is generally faster**, a single classifier.
  - classes are **mutually exclusive**, no overlap.
- **One-vs-all:**
  - **Multilabel classification**, document can fall in **1+ categories**: e.g. classify by language:

    I said "bonjour mon ami" to my friend → English & French

    How? Out of k classifiers, those with **confidence > threshold**

EVALUATION OF TEXT CLASSIFICATION

Evaluation of Text Classification

- Evaluation is **different for binary and multiclass** classification.
  - **Binary:** we generally have a **positive and a negative class** (spam vs non-spam, medical test positive vs negative, exam pass vs fail). Classification **errors can only go the other class**.
  - **Multiclass:** multiple categories, may have different level of importance. Classification **errors can go to any other class**.

Evaluation of Binary Classification

- 2-by-2 contingency table:

	Actually positive	**Actually negative**
Classified as positive	True Positive (TP)	False Positive (FP)
Classified as negative	False Negative (FN)	True Negative (TN)

Evaluation of Binary Classification

- 2-by-2 contingency table:

	Actually positive	**Actually negative**
Classified as positive	True Positive (TP)	False Positive (FP)
Classified as negative	False Negative (FN)	True Negative (TN)

- **Precision:** ratio of items classified as positive that are correct.
- **Recall:** ratio of actual positive items that are classified as positive.

Evaluation of Binary Classification

- 2-by-2 contingency table:

	Actually positive	Actually negative
Classified as positive	True Positive (TP)	False Positive (FP)
Classified as negative	False Negative (FN)	True Negative (TN)

- **Precision:** ratio of items classified as positive that are correct.
- **Recall:** ratio of actual positive items that are classified as positive.

Evaluation of Binary Classification

- 2-by-2 contingency table:

	Actually positive	Actually negative
Classified as positive	True Positive (TP)	False Positive (FP)
Classified as negative	False Negative (FN)	True Negative (TN)

- **Precision:** ratio of items classified as positive that are correct.

$$\text{Precision} = \frac{tp}{tp + fp}$$

- **Recall:** ratio of actual positive items that are classified as positive.

$$\text{Recall} = \frac{tp}{tp + fn}$$

Evaluation of Binary Classification

- We want to optimise for both precision and recall:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$ (harmonic mean of precision and recall)

- General equation as follows, however generally ß = 1:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$$

Evaluation of Multiclass Classification

- Bigger **confusion matrix**:

	Actually UK	Actually World	Actually Tech	Actually Science	Actually Politics	Actually Business
Classified as UK	**95**	1	13	0	1	0
Classified as World	0	**1**	0	0	0	0
Classified as Tech	10	90	**0**	1	0	0
Classified as Science	0	0	0	**34**	3	7
Classified as Politics	0	1	2	13	**26**	5
Classified as Business	0	0	2	14	5	**10**

Evaluation of Multiclass Classification

- **Overall Accuracy:** ratio of correct classifications.

$$\frac{\sum_{i} c_{ii}}{\sum_{j}\sum_{i} c_{ij}}$$

i.e. out of N items in the test set, how many did we classify correctly? 62 out of 100, then accuracy = 0.62

Evaluation of Multiclass Classification

- **Overall Accuracy:** ratio of correct classifications.

- **Often an insufficient evaluation approach**, e.g.:
  - We classify 1,000 texts → data is skewed with 990 texts actually having positive sentiment.
  - We (naively) classify everything as positive.
  - 990 classified correctly: 990 / 1000 = 0.99 accuracy!

Is it fair?

Evaluation of Multiclass Classification

- **Per-class precision and recall**:

Precision: $$\frac{c_{ii}}{\sum_j c_{ji}}$$

c_{ii} → # of items we correctly classified as class i

$\sum_j c_{ji}$ → # of items we predicted as class i

Recall: $$\frac{c_{ii}}{\sum_j c_{ij}}$$

c_{ii} → # of items we correctly classified as class i

$\sum_j c_{ij}$ → # of actual i items

- With the **harmonic mean**, we can then get **per-class F1 score**.

Evaluation of Multiclass Classification

- We have **per-class precision, recall and F1 scores**.
  - How do we combine them all to **get a single score**?

Evaluation of Multiclass Classification

- Obtaining overall performances:
  - **Macroaveraging:**
    Compute **performance for each class, then average them**.
    All **classes contribute the same** to the final score (e.g. class with 990 and class with 10 instances).

  - **Microaveraging:**
    Compute **overall performance** without computing per-class performances.
    Large classes contribute more to the final score.

Evaluation of Multiclass Classification

- Macroaveraging:

$$\text{macroaveraged precision} = \frac{\sum_i \frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i}}{k}$$

$$\text{macroaveraged recall} = \frac{\sum_i \frac{\mathrm{TP}}{\mathrm{TP}_i + \mathrm{FN}_i}}{k}$$

- The macroaveraged F1 score is then the harmonic mean of those.

Evaluation of Multiclass Classification

- Microaveraging:

$$\text{microaveraged precision} = \frac{\sum_i \text{TP}_i}{\sum_i \text{TP}_i + \text{FP}_i}$$

$$\text{microaveraged recall} = \frac{\sum_i \text{TP}}{\sum_i \text{TP}_i + \text{FN}_i}$$

- The microaveraged F1 score is then the harmonic mean of those.

Micro- vs Macro-averaging Example

	S	D	Q	C
S	**366 (40.4%)**	32 (3.5%)	22 (2.4%)	487 (53.7%)
D	38 (11.1%)	**22 (6.4%)**	23 (6.7%)	260 (75.8%)
Q	11 (3.1%)	10 (2.8%)	**149 (41.6%)**	188 (52.5%)
C	261 (9.0%)	91 (3.1%)	133 (4.6%)	**2,421 (83.3%)**

- Microaveraged F1 score: **0.665**
- Macroaveraged F1 score: **0.440**

where I have so many instances with label C, which metric is fairer?

depends on our objective: do we need a classifier that performs well for all labels? Do we just want to perform well for C?

Further Weighting / Selection

- We can choose to **prioritise certain categories**:
  - Give **higher weight to important categories**:
    $0.3*\text{Prec}(c_1) + 0.3*\text{Prec}(c_2) + 0.4*\text{Prec}(c_3)$
    → unless we have clear criteria to choose these weights, it's rather arbitrary though!
  - Select some categories for inclusion in macro/microaverage:
    - e.g. in sentiment analysis, we could only **macroaverage over positive and negative sentiment classes**, if we choose to ignore performance on the neutral class.

ERROR ANALYSIS FOR TEXT CLASSIFICATION

Error Analysis

- **Error analysis:** can help us find out where our classifier can do better.
- No magic formula for performing error analysis.
  - Look **where we are doing wrong**, what labels particularly.
  - Do our errors **have some common characteristics**? Can we infer a new feature from that?
  - Could our **classifier be favouring one of the classes** (e.g. the majority class)?

Error Analysis

- **Error analysis:** where are we doing wrong? What labels?

  Look at frequent deviations in the confusion matrix.

	Actually UK	Actually World	Actually Tech	Actually Science	Actually Politics	Actually Business
Classified as UK	**95**	1	13	0	1	0
Classified as World	0	**1**	0	0	0	0
Classified as Tech	10	90	**0**	1	0	0
Classified as Science	0	0	0	**34**	3	7
Classified as Politics	0	1	2	13	**26**	5
Classified as Business	0	0	2	14	5	**10**

Error Analysis

- **Error analysis:** do our errors have some common characteristics?

Print some of our errors, e.g. classifying person names by gender.

```
>>> for (tag, guess, name) in sorted(errors):
...     print('correct={:<8} guess={:<8s} name={:<30}'.format(tag, guess, name))
correct=female   guess=male     name=Abigail
  ...
correct=female   guess=male     name=Cindelyn
  ...
correct=female   guess=male     name=Katheryn
correct=female   guess=male     name=Kathryn
  ...
correct=male     guess=female   name=Aldrich
  ...
correct=male     guess=female   name=Mitch
  ...
correct=male     guess=female   name=Rich
  ...
```

Error Analysis

- **Error analysis:** do our errors have some common characteristics?

Print some of our errors, e.g. classifying person names by gender.

```
>>> for (tag, guess, name) in sorted(errors):
...     print('correct={:<8} guess={:<8s} name={:<30}'.format(tag, guess, name))
correct=female   guess=male     name=Abigail
  ...
correct=female   guess=male     name=Cindelyn
  ...
correct=female   guess=male     name=Katheryn
correct=female   guess=male     name=Kathryn
  ...
correct=male     guess=female   name=Aldrich
  ...
correct=male     guess=female   name=Mitch
  ...
correct=male     guess=female   name=Rich
  ...
```

Error Analysis

- **Error analysis:** do our errors have some common characteristics?

  Print some of our errors.

```
>>> for (tag, guess, name) in sorted(errors):
...     print('correct={:<8} guess={:<8s} name={:<30}'.format(tag, guess, name))
correct=female   guess=male     name=Abigail
  ...
correct=female   guess=male     name=Cindelyn
  ...
correct=female   guess=male     name=Katheryn
correct=female   guess=male     name=Kathryn
  ...
correct=male     guess=female   name=Aldrich
  ...
correct=male     guess=female   name=Mitch
  ...
correct=male     guess=female   name=Rich
  ...
```

New feature: suffix, last 2-3 characters

Error Analysis

- **Error analysis:** could our classifier be favouring one of the classes?
  - Owing to class imbalance, classifiers tend to predict popular classes more often, e.g.:

	S	D	Q	C
S	**366 (40.4%)**	32 (3.5%)	22 (2.4%)	487 (53.7%)
D	38 (11.1%)	**22 (6.4%)**	23 (6.7%)	260 (75.8%)
Q	11 (3.1%)	10 (2.8%)	**149 (41.6%)**	188 (52.5%)
C	261 (9.0%)	91 (3.1%)	133 (4.6%)	**2,421 (83.3%)**

Error Analysis

- For example, if our dataset has instances:
  - class A (700)
  - class B (100)
  - class C (100)
  - class D (100)

- Classifier will generally tend to predict A more often, because it has seen many more samples.

Error Analysis

- **Error analysis:** could our classifier be favouring one of the classes?
  - How to deal with imbalance (i.e. A-700, B-100, C-100, D-100)?
    - 1) Undersample popular class → A-100, B-100, C-100, D-100

      randomly remove 600 instances of A

Error Analysis

- **Error analysis:** could our classifier be favouring one of the classes?
  - How to deal with imbalance (i.e. A-700, B-100, C-100, D-100)?
    - 2) Oversample other classes → A-700, B-700, C-700, D-700

      repeat instances of B, C, D to match the number of A's

Error Analysis

- **Error analysis:** could our classifier be favouring one of the classes?
  - How to deal with imbalance (i.e. A-700, B-100, C-100, D-100)?
    - 3) Create synthetic data → A-700, B-700, C-700, D-700

      generate new B, C, D items → needs some understanding of the contents of the classes to be able to produce sensible data items

Error Analysis

- **Error analysis:** could our classifier be favouring one of the classes?
  - How to deal with imbalance (i.e. A-700, B-100, C-100, D-100)?
    - 4) Cost sensitive learning

      e.g. higher probability to predict uncommon classes
      P(A)=1/700, P(B)=1/100, P(C)=1/100, P(D)=1/100

      `scikit → class_weight="auto"`

Error Analysis

- **Important** for the error analysis:
  - **Subset we analyse for errors** (dev set) has to be **different to the one where we ultimately apply the classifier** (test set).
  - If we **tweak the classifier looking at the test set**, we'll end up **overfitting**, developing a classifier that works very well for that particular test set.

Associated Reading

- Jurafsky, Daniel, and James H. Martin. 2023. Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics. 3rd edition.
  - **Chapters 2.2-2.5.**
  - **Chapters 4.7-4.8.**

Queen Mary
University of London